# Implementing in-memory cache in the BeaST architecture

Version 1.1 2016.05.19

Mikhail E. Zakharov, zmey20000@yahoo.com

Recently we have created the reliable dual-headed FreeBSD based storage concept. Then we improved it with the famous ZFS feature. Now we will try to implement a shared in-memory cache to make our model even more serious.

In the enterprise systems with the real hardware controllers we should think of connecting the storage cache directly with the high-speed buses which are shared between the controllers. But in our virtual environment, actually this is still an Oracle VM VirtualBox on my laptop, we have to use LAN for every interconnection. Earlier we decided to stop on the iSCSI protocol for our simple SAN solution.

In general the virtual environment is not changed much: two machines (ctrl-a and ctrl-b) are the storage controllers and one is the client (clnt-1). But in the previous case with ZFS tests we had to add four additional virtual drives to emulate SSDs as the storage space for the cache. We do not need them any more because now we intend to use RAM for this purpose. Therefore we can easily copy our previous virtual machine configurations with the exception of the f*xx* drives.

The configuration summary is shown in the table below:

| **Description** | **ctrl-a** | **ctrl-b** | **clnt-1** |
|---|---|---|---|
| Inter-controller (private) network. Host-only adapter (vboxnet0) | IP: 192.168.56.10<br>Mask: 255.255.255.0 | IP: 192.168.56.11<br>Mask: 255.255.255.0 | – |
| Public network. Host-only adapter (vboxnet1) | IP: 192.168.55.10<br>Mask: 255.255.255.0 | IP: 192.168.55.11<br>Mask: 255.255.255.0 | IP: 192.168.55.20<br>Mask: 255.255.255.0 |
| Base memory | 2048 MB or more | 2048 MB or more | Any appropriate value starting with 512 MB will do |
| Shareable, fixed-sized virtual drives for ZFS data volumes on the SATA controller. | d00, d01, d10, d11 – each drive is 100 MB size or more | d00, d01, d10, d11 – each drive is 100 MB or more | – |
| System virtual drives (Dynamic-sized) on the IDE controller | At least 5 GB to store FreeBSD 10.3-Release default installation | At least 5 GB to store FreeBSD 10.3-Release default installation | At least 5 GB to store FreeBSD 10.3-Release default installation |

Note, that it's still important to create data-drives (d00, d01, d10, d11) with the **fixed-sized** and **shareable** options **enabled** (see Oracle VM VirtualBox Virtual Media Manager for the details). Otherwise we will be unable to connect these drives simultaneously to the both machines (ctrl-a and ctrl-b) and emulate the drive enclosure which is shared between the storage controllers.

Also I will continue to use USB memory-stick to store fixed-sized shareable drives for our testing. **But this is being done only in order to slowdown I/O operations and see what is going on the model.**

The latest FreeBSD 10.3 Release can be installed on the non-shareable ada0 drives of the virtual storage machines with the configuration changes in /etc/rc.conf usual for our project:

| **ctrl-a** | **ctrl-b** |
|---|---|
| hostname="ctrl-a"<br><br>ifconfig_em0="inet 192.168.56.10 netmask 255.255.255.0"      # Inter-controller LAN<br>ifconfig_em1="inet 192.168.55.10 netmask 255.255.255.0"      # Public network<br><br>sshd_enable="YES"<br># Set dumpdev to "AUTO" to enable crash dumps, "NO" to disable<br>dumpdev="AUTO"<br><br># VirtualBox guest additions<br>vboxguest_enable="YES"<br>vboxservice_enable="YES"<br><br># iSCSI<br>ctld_enable="YES"       # Targets<br>iscsid_enable="YES"    # Initiators | hostname="ctrl-b"<br><br>ifconfig_em0="inet 192.168.56.11 netmask 255.255.255.0" # Inter-controller LAN<br>ifconfig_em1="inet 192.168.55.11 netmask 255.255.255.0" # Public network<br><br>sshd_enable="YES"<br># Set dumpdev to "AUTO" to enable crash dumps, "NO" to disable<br>dumpdev="AUTO"<br><br># VirtualBox guest additions<br>vboxguest_enable="YES<br>vboxservice_enable="YES"<br><br># iSCSI<br>ctld_enable="YES" # target<br>iscsid_enable="YES"    # initiator |

Set iSCSI “disconnection on fail” kernel variable in /etc/sysctl.conf on both systems to be able to failover to the alive controller in case of disaster:

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

After finishing all these basic FreeBSD installations and preparations, we can start in-memory cache configuration.

## The in-memory cache

As we did not write a kernel module with a device driver for our own cache algorithm, we will use ZFS caching capabilities to reach the goal. In the previous article we have already learned how to configure ZIL and ARC/L2ARC on the shared drives to suite our needs as write- and read- caches. In this part we will try to allocate these caches in the RAM-drive spaces.

And at this point we have at least two major problems to solve.

First is that the storage system with its in-memory cache is very sensitive to the power-loss. Therefore, in the real life, we must protect both of controllers with the independent electrical power sources and UPSs. The same requirement is applied to the drive enclosures as well. Hopefully, now we are playing in the sandbox of the virtual lab without any useful data, so we should not worry to lose it.

Second, we need now to invent a synchronization mechanism to enable cache-mirroring feature between the controllers. And in our limited environment we have to use iSCSI protocol over the virtual LAN as the main transport for this interconnection.

To accomplish this task we will need to create four memory drives (md) on both controllers to maintain primary and secondary copies of the read- and write- caches and appropriate inter-layer devices – ggate (actually this is ggatel – a “local” version) to simplify ZFS management by using the unified device names. See the table below for the cache structure details:

| Memory drive | ZFS inter-layer | Location | Description |
|---|---|---|---|
| md0 | ggate0 | ctrl-a | ctrl-a write-cache ZFS ZIL **primary** copy |
| md1 | ggate1 | ctrl-a | ctrl-a read-cache ZFS L2ARC **primary** copy |
| md2 | ggate2 (on gmirror)* | ctrl-a | ctrl-b write-cache ZFS ZIL **secondary** copy |
| md3 | ggate3 (on gmirror)* | ctrl-a | ctrl-b read-cache ZFS L2ARC **secondary** copy |
| md0 | ggate0 (on gmirror)* | ctrl-b | ctrl-a write-cache ZFS ZIL **secondary** copy |
| md1 | ggate1 (on gmirror)* | ctrl-b | ctrl-a read-cache ZFS L2ARC **secondary** copy |
| md2 | ggate2 | ctrl-b | ctrl-b write-cache ZFS ZIL **primary** copy |
| md3 | ggate3 | ctrl-b | ctrl-b read-cache ZFS L2ARC **primary** copy |

* We cant do it right on this step, therefore see the gmirror configuration on the next page.

Now we can implement this structure by the commands:

| **ctrl-a** | **ctrl-b** |
|---|---|
| `mdconfig -a -t swap -s 128m -u 0` | `mdconfig -a -t swap -s 128m -u 0` |
| `mdconfig -a -t swap -s 128m -u 1` | `mdconfig -a -t swap -s 128m -u 1` |
| `mdconfig -a -t swap -s 128m -u 2` | `mdconfig -a -t swap -s 128m -u 2` |
| `mdconfig -a -t swap -s 128m -u 3` | `mdconfig -a -t swap -s 128m -u 3` |
| | |
| `ggatel create -t 1 -u 0 /dev/md0` | `ggatel create -t 1 -u 2 /dev/md2` |
| `ggatel create -t 1 -u 1 /dev/md1` | `ggatel create -t 1 -u 3 /dev/md3` |

We use eight 128 MB memory areas which gives us 1 GB of cache divided between two controllers. In other words we have 256 MB of read-cache and 256 MB of write-cache. Actually these sizes depend on the real memory of the controllers but the basic rule is simple: install more memory and use as much cache as possible.

Also we can load gmirror module as we will need it soon:

| **ctrl-a** | **ctrl-b** |
|---|---|
| `gmirror load` | `gmirror load` |

Now we can establish iSCSI interconnections between cache areas of the opposite controllers. So edit /etc/ctl.conf on both controllers to enable iSCSI-targets. Lets use even LUNs (0 and 2) to access the write-cache and odd LUNs (1 and 3) for the read-cache:

| **ctrl-a** | **ctrl-b** |
|---|---|
| `portal-group pg0 {` | `portal-group pg0 {` |
| `    discovery-auth-group no-authentication` | `    discovery-auth-group no-authentication` |
| `    listen 192.168.56.10` | `    listen 192.168.56.11` |

| | |
|---|---|
| } | } |
| | |
| target iqn.2016-01.local.sss.private:target0 { | target iqn.2016-01.local.sss.private:target0 { |
| auth-group no-authentication | auth-group no-authentication |
| portal-group pg0 | portal-group pg0 |
| | |
| lun 0 { | lun 0 { |
| path /dev/md0 | path /dev/md2 |
| } | } |
| | |
| lun 1 { | lun 1 { |
| path /dev/md1 | path /dev/md3 |
| } | } |
| } | } |

Tell ctld to re-read its target configurations on and then initiate iSCSI interconnections:

| **ctrl-a** | **ctrl-b** |
|---|---|
| killall -HUP ctld | killall -HUP ctld |
| | |
| iscsictl -A -p 192.168.56.11 -t iqn.2016-01.local.sss.private:target0 | iscsictl -A -p 192.168.56.10 -t iqn.2016-01.local.sss.private:target0 |

But this was only a half of the job. The other part is to create cache synchronization mechanisms to maintain cache mirroring between the controllers. And we also will use gmirror for this task and ggate (ggatel) to hide these mirrors under the inter-layer devices (see the remark on the page above):

| **ctrl-a** | **ctrl-b** |
|---|---|
| gmirror label ctrl_b_zil /dev/da0 /dev/md2 | gmirror label ctrl_a_zil /dev/da0 /dev/md0 |
| gmirror label ctrl_b_arc /dev/da1 /dev/md3 | gmirror label ctrl_a_arc /dev/da1 /dev/md1 |
| | |
| ggatel create -t 1 -u 2 /dev/mirror/ctrl_b_zil | ggatel create -t 1 -u 0 /dev/mirror/ctrl_a_zil |
| ggatel create -t 1 -u 3 /dev/mirror/ctrl_b_arc | ggatel create -t 1 -u 1 /dev/mirror/ctrl_a_arc |

We have just invented the structure of the mirrored devices and their unified inter-layer interface. Now we can try to populate it with the ZFS cache.

## ZFS

The Pool configuration is quite simple and well known from the previous article.

We create the ctrl-a_m0 and the ctrl-b_m0 pools on the shareable drives (ada1/ada2 and ada3/ada4). Then we allocate volumes on these pools:

| **ctrl-a** | **ctrl-b** |
|---|---|
| zpool create -m none ctrl-a_m0 /dev/ada1 /dev/ada2 | zpool create -m none ctrl-b_m0 /dev/ada3 /dev/ada4 |
| zfs create -V 120M ctrl-a_m0/v0 | zfs create -V 120M ctrl-b_m0/v0 |

Now we can configure ZFS cache. We will use RAM-drives (hided by the ggate pseudo-devices) for this purpose:

| **ctrl-a** | **ctrl-b** |
|---|---|
| # ZIL<br>zpool add -f ctrl-a_m0 log /dev/ggate0<br>zfs set sync=always ctrl-a_m0<br><br># ARC/L2ARC<br>zpool add ctrl-a_m0 cache /dev/ggate1<br>zfs set primarycache=none ctrl-a_m0<br>zfs set secondarycache=all ctrl-a_m0 | # ZIL<br>zpool add -f ctrl-b_m0 log /dev/ggate2<br>zfs set sync=always ctrl-b_m0<br><br># ARC/L2ARC<br>zpool add ctrl-b_m0 cache /dev/ggate3<br>zfs set primarycache=none ctrl-b_m0<br>zfs set secondarycache=all ctrl-b_m0 |

At the final step of ZFS configuration we will import pools from opposite controllers and set special flags to ignore all errors on them. Yes, this is dangerous but we must do it to be able to fail to the alive controller in case of failure:

| **ctrl-a** | **ctrl-b** |
|---|---|
| zpool import -N ctrl-b_m0<br><br>zpool set failmode=continue ctrl-a_m0<br>zpool set failmode=continue ctrl-b_m0 | zpool import -N ctrl-a_m0<br><br>zpool set failmode=continue ctrl-a_m0<br>zpool set failmode=continue ctrl-b_m0 |

Note, we set -N option to disable auto-mounting ZFS function.

At this point ZFS pools of both controllers with their mirrored in-memory caches should work, therefore we can setup the volume failover arbitrator mechanism.

## The Arbitrator

The volume arbitrator, which we use to switch volumes between the controllers, is pretty much the same as in both previous test.

Therefore just check and edit /etc/ctl.conf to add appropriate volumes (we use LUN 10 on both controllers for them):

| **ctrl-a** | **ctrl-b** |
|---|---|
| portal-group pg0 {<br>    discovery-auth-group no-authentication<br>    listen 192.168.56.10<br>}<br><br>target iqn.2016-01.local.sss.private:target0 {<br>    auth-group no-authentication<br>    portal-group pg0<br><br>    # cache<br>    lun 0 {<br>        path /dev/md0<br>    }<br><br>    lun 1 {<br>        path /dev/md1 | portal-group pg0 {<br>    discovery-auth-group no-authentication<br>    listen 192.168.56.11<br>}<br><br>target iqn.2016-01.local.sss.private:target0 {<br>    auth-group no-authentication<br>    portal-group pg0<br><br>    # cache<br>    lun 2 {<br>        path /dev/md2<br>    }<br><br>    lun 3 {<br>        path /dev/md3 |

| | |
|---|---|
|   }<br><br>  # data volumes<br>  lun 10 {<br>    path /dev/zvol/ctrl-a_m0/v0<br>  }<br>} |   }<br><br>  # data volumes<br>  lun 10 {<br>    path /dev/zvol/ctrl-b_m0/v0<br>  }<br>} |

Then force ctld to re-read its configuration file and reinitialize iSCSI connections to discover the data-volumes. Finally assembly the arbitrator mechanism by creating active-passive multipath pseudo-device:

| **ctrl-a** | **ctrl-b** |
|---|---|
| killall -HUP ctld<br><br># reinitialize iSCSI backend session<br>iscsictl -M -i 1 -p 192.168.56.11 -t iqn.2016-01.local.sss.private:target0<br><br># Create the arbitrator<br>gmultipath create CTRL_B_BACK /dev/da2 /dev/zvol/ctrl-b_m0/v0 | killall -HUP ctld<br><br># reinitialize iSCSI backend session<br>iscsictl -M -i 1 -p 192.168.56.10 -t iqn.2016-01.local.sss.private:target0<br><br># Create the arbitrator<br>gmultipath create CTRL_A_BACK /dev/da2 /dev/zvol/ctrl-a_m0/v0 |

Note the -M option of the iscsictl command which we use to modify the previously established session.

## Front-End

In the final part of the storage-side preparations we should configure the iSCSI front-end to accept the client connections. Therefore we add “public” related sections to the /etc/ctl.conf:

| **ctrl-a** | **ctrl-b** |
|---|---|
| portal-group pg0 {<br>  discovery-auth-group no-authentication<br>  listen 192.168.56.10<br>}<br><br>portal-group pg1 {<br>  discovery-auth-group no-authentication<br>  listen 192.168.55.10<br>}<br><br>target iqn.2016-01.local.sss.private:target0 {<br>  auth-group no-authentication<br>  portal-group pg0<br><br>  # cache<br>  lun 0 {<br>    path /dev/md0<br>  }<br><br>  lun 1 {<br>    path /dev/md1<br>  } | portal-group pg0 {<br>  discovery-auth-group no-authentication<br>  listen 192.168.56.11<br>}<br><br>portal-group pg1 {<br>  discovery-auth-group no-authentication<br>  listen 192.168.55.11<br>}<br><br>target iqn.2016-01.local.sss.private:target0 {<br>  auth-group no-authentication<br>  portal-group pg0<br><br>  # cache<br>  lun 2 {<br>    path /dev/md2<br>  }<br><br>  lun 3 {<br>    path /dev/md3<br>  } |

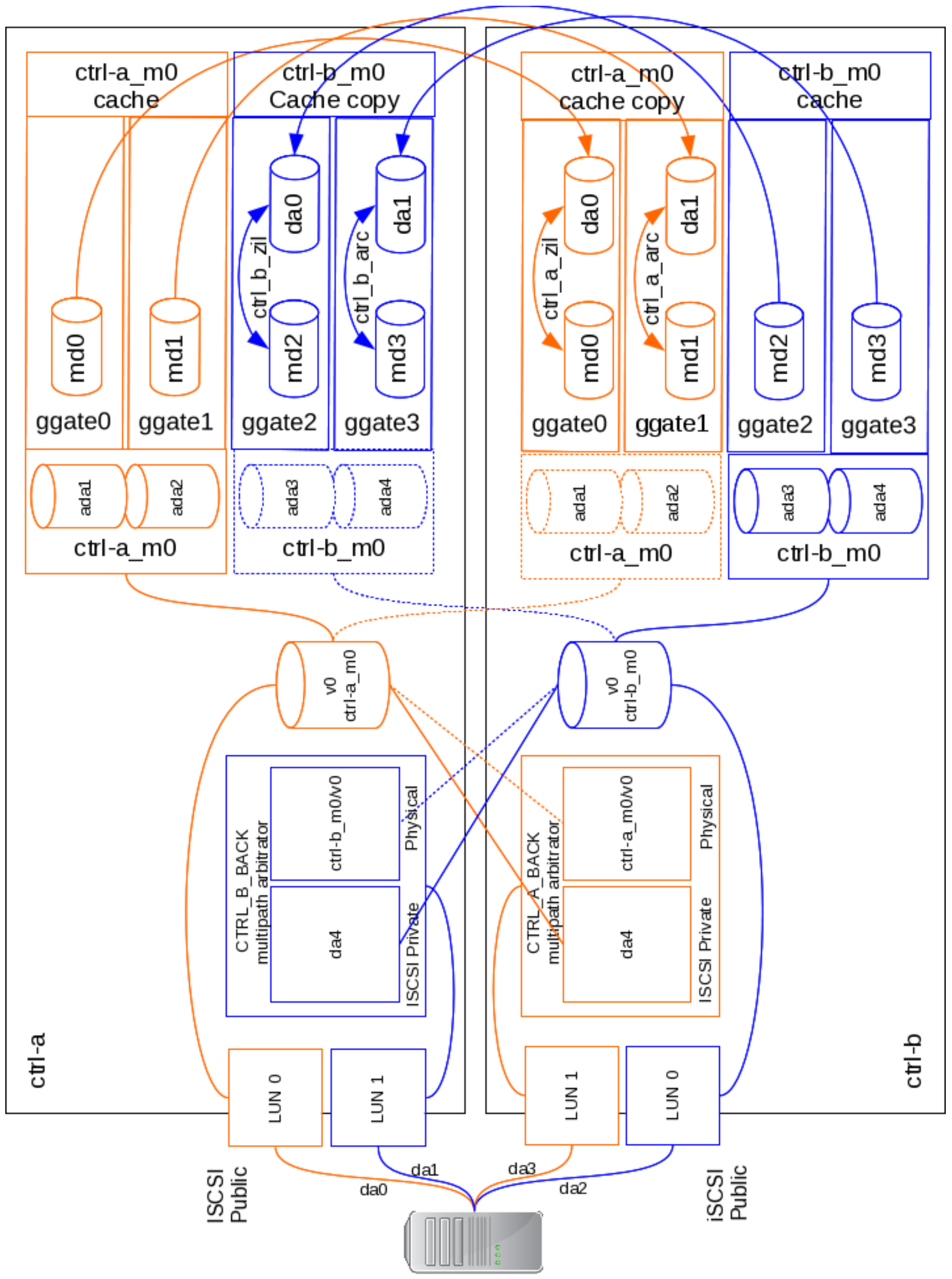
```
	# data volumes
	lun 10 {
		path /dev/zvol/ctrl-a_m0/v0
	}
}

target iqn.2016-01.local.sss.public:target0 {
	auth-group no-authentication
	portal-group pg1

	lun 0 {
		path /dev/zvol/ctrl-a_m0/v0
	}

	lun 1 {
		path /dev/multipath/CTRL_B_BACK
	}
}
```

```
	# data volumes
	lun 10 {
		path /dev/zvol/ctrl-b_m0/v0
	}
}

target iqn.2016-01.local.sss.public:target0 {
	auth-group no-authentication
	portal-group pg1

	lun 0 {
		path /dev/zvol/ctrl-b_m0/v0
	}

	lun 1 {
		path /dev/multipath/CTRL_A_BACK
	}
}
```

Finally we must tell ctld to update its LUN information once more. Therefore run on both controllers:

```
# killall -HUP ctld
```

After fresh additions and changes to the concept of the BeaST storage, the structure of our model became quite complex. Therefore see the full architecture layout below to get the whole picture with a bird's-eye view.



The BeaST storage architecture with ZFS and in-memory cache overview

## The client-side

The client configuration is not changed since the first test. So take the previous client virtual machine as-is and check FreeBSD basic configuration in /etc/rc.conf:

```
hostname="clnt-1"

ifconfig_em0="inet 192.168.55.20 netmask 255.255.255.0"    # Public network

sshd_enable="YES"
```

```
# Set dumpdev to "AUTO" to enable crash dumps, "NO" to disable
dumpdev="AUTO"

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

# iSCSI
iscsid_enable="YES"    # Initiators
```

The /etc/sysctl.conf file is also the same:

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

Then run all the very well known commands to bring iSCSI connections online, form multipath pseudo-devices, stripe the volume, create and mount the new file system:

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

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

root@clnt-1:/home/beast # gstripe create ZDATA /dev/multipath/CTRL_A /dev/multipath/CTRL_B
root@clnt-1:/home/beast # newfs /dev/stripe/ZDATA
root@clnt-1:/home/beast # mount /dev/stripe/ZDATA /storage
```

Now you can start the common tasks to test the BeaST solution for its functionality and reliability. See our previous works “FreeBSD based dual-controller storage system concept” and “Adding ZFS to the FreeBSD dual-controller storage concept” for the details of these processes.

## Conclusion

In this paper we have offered the mechanism of the in-memory cache implementation for the storage system based only on the standard FreeBSD and ZFS features.

And this was the most important and interesting part of this paper indeed. So most common and boring tasks (like FreeBSD installations, CLI outputs along with functional tests and etc) were cut off from this paper by the Occam's razor as they completely repeat the previous two articles.
We have done much, but there are still more things to do. We must test the BeaST on the real hardware, write installation and clustering scripts to automate typical tasks (like volume creation, migration, recovery, etc) on both controllers. Also we should check the new and interesting changes in CAM target layer of FreeBSD 10.3.

**Finally we must warn you that the BeaST concept 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 in production as nobody can guarantee that you will not lose data. Be careful and always do backups!**