Optimizing in-memory cache of the BeaST architecture

Mikhail E. Zakharov zmey20000@yahoo.com

Preface

The BeaST is the new FreeBSD based dual-headed reliable storage system concept. Recently we implemented both ZFS and in-memory cache in our architecture. And after this last improvement the BeaST system has become quite complex comparing to the predecessors.

Current BeaST version uses full-mirrored in-memory cache. In other words, all read- and write-cache partitions are mirrored between controllers. This architecture was chosen with the only aim to simplify ZFS and in-memory cache neighboring tricks.

But cache is one of the most important yet quite expensive, from multiple points of view, storage system components. Therefore, it may be a good idea to reorganize cache architecture in order to save more resources. And the main target in the BeaST concept is to avoid unnecessary read-cache mirroring as this type of cache consumes resources but contains only not-unique data, which can be anytime read again from the drives.

Another interesting thing regarding the BeaST read-cache is related to ZFS algorithms. It appears that L2ARC is completely unused if main ARC is disabled. This case was investigated by Adam Stylinski <stylinae@mail.uc.edu>:

Just tested this and verified it for myself -- it is still the case that if no primarycache is enabled, secondary caching will not take effect.

I tested this by constructing a quick zpool in a 10-STABLE VM by using a file for a vdev, and carving out from the existing zpool a small zvol for the cache (since cache vdevs have to consist of drives or partitions). I made it tiny but disabled primary cache, ran dd from /dev/urandom into files on that pool to the point where it was about 50% full, then I did sequential and random reads with grep and dd to those files, watch both the output of iostat and the output of the following systats: kstat.zfs.misc.arcstats.l2_misses & kstat.zfs.misc.arcstats.l2_hits.

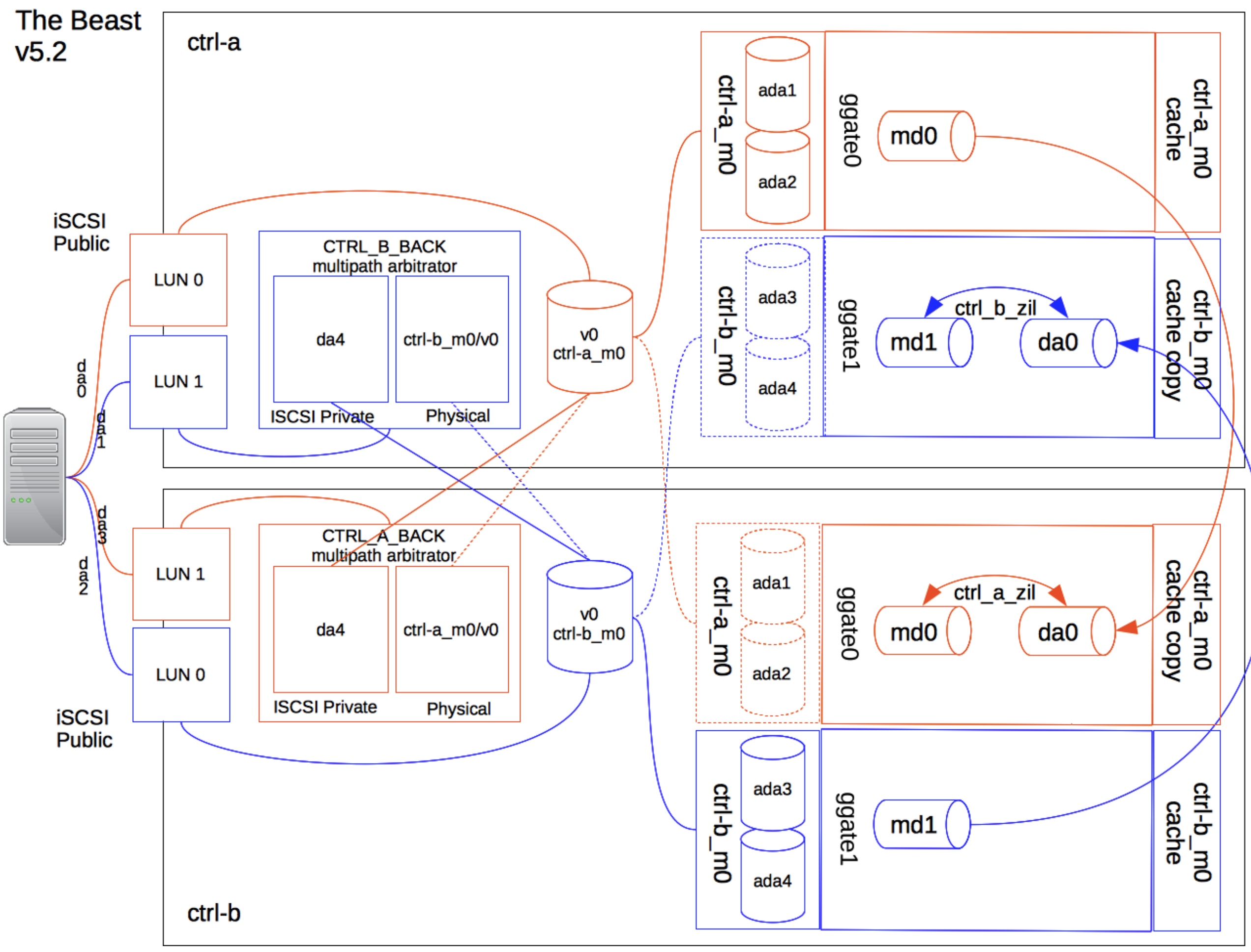
```
…
[adam@fbsd-stable-vm:/home/adam]%sysctl -a | grep -i arc | egrep "(l2_hits|l2_misses)"
kstat.zfs.misc.arcstats.l2_misses: 57049
kstat.zfs.misc.arcstats.l2_hits: 0
[adam@fbsd-stable-vm:/home/adam]%sysctl -a | grep -i arc | egrep "(l2_hits|l2_misses)"
kstat.zfs.misc.arcstats.l2_misses: 58751
kstat.zfs.misc.arcstats.l2_hits: 0
[adam@fbsd-stable-vm:/home/adam]%sysctl -a | grep -i arc | egrep "(l2_hits|l2_misses)"
kstat.zfs.misc.arcstats.l2_misses: 61065
kstat.zfs.misc.arcstats.l2_hits: 0
```

Many thanks to Adam Stylinski for his great work!

Cache architectural improvements

As for our system, it means that these memory partitions are reserved but are not used by read-cache. Therefore, we must review the architecture in order to remove read-cache mirroring between the controllers.

Being reorganized the BeaST architecture looks much simpler and, what is more important, it allows ARC to cache data:



Basic preparations

To run the new version we will use exactly the same environment, which is left from the tests described in the Implementing in-memory cache in the BeaST architecture-1.1 paper. The only change is that we finally decided to drop out that slow and thus annoying USB-memory stick.

The configuration summary to reproduce the environment is shown below:

Description	**ctrl-a**	**ctrl-b**	**clnt-1**
Inter-controller (private) network. Host-only adapter (vboxnet0)	IP: 192.168.56.10 Mask: 255.255.255.0	IP: 192.168.56.11 Mask: 255.255.255.0	-
Public network.	IP: 192.168.55.10	IP: 192.168.55.11	IP: 192.168.55.20

Host-only adapter (vboxnet1)	Mask: 255.255.255.0	Mask: 255.255.255.0	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

Install FreeBSD 10.3 Release on the non-shareable drives (ada0 in our case) of the virtual storage machines with the typical for our project configuration changes 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 LAN	ifconfig_em0="inet 192.168.56.11 netmask 255.255.255.0" # Inter-controller LAN
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
sshd_enable="YES"	sshd_enable="YES"
# Set dumpdev to "AUTO" to enable crash dumps, "NO" to disable	# Set dumpdev to "AUTO" to enable crash dumps, "NO" to disable
dumpdev="AUTO"	dumpdev="AUTO"
# VirtualBox guest additions	# VirtualBox guest additions
vboxguest_enable="YES"	vboxguest_enable="YES
vboxservice_enable="YES"	vboxservice_enable="YES"
# iSCSI	# iSCSI
ctld_enable="YES" # Targets	ctld_enable="YES" # target
iscsid_enable="YES" # Initiators	iscsid_enable="YES" # initiator

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

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

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

ZFS basic configuration

It is very simple as now we create only pools on both controllers and volumes to store data:

ctrl-a	**ctrl-b**

`zpool create -m none ctrl-a_m0 /dev/ada1 /dev/ada2`
 	
`zfs create -V 120M ctrl-a_m0/v0`
	`zpool create -m none ctrl-b_m0 /dev/ada3 /dev/ada4`
	
	`zfs create -V 120M ctrl-b_m0/v0`

ZIL configuration and cross-controller pools import are described in the next section.

In-memory cache

Reflecting the changes in the cache architecture design, our memory drive to GEOM-gate map has also been changed. It is simpler now as the drives will contain only write-cache:

Memory drive	ZFS inter-layer	Controller	Description
md0	ggate0	ctrl-a	ctrl-a write-cache (ZFS ZIL) **primary** copy
md1	ggate1	ctrl-a	ctrl-b write-cache (ZFS ZIL) **secondary** copy
md0	ggate0	ctrl-b	ctrl-a write-cache (ZFS ZIL) **secondary** copy
md1	ggate1	ctrl-b	ctrl-b write-cache (ZFS ZIL) **primary** copy

To reproduce the renewed cache structure on the system run:

ctrl-a	**ctrl-b**

`mdconfig -a -t swap -s 128m -u 0`
`mdconfig -a -t swap -s 128m -u 1`
 	
`ggatel create -t 1 -u 0 /dev/md0`
	`mdconfig -a -t swap -s 128m -u 0`
	`mdconfig -a -t swap -s 128m -u 1`
	
	`ggatel create -t 1 -u 1 /dev/md1`

Don’t forget to load gmirror as we will need it soon:

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

Now we can prepare iSCSI targets part for the cache synchronization mechanism in the /etc/ctl.conf file:

ctrl-a	**ctrl-b**

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

        # ctrl-a ZIL primary copy
        lun 0 {
                path /dev/md0
        }
}
```

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

        # ctrl-b ZIL primary copy
        lun 1 {
                path /dev/md1
        }
}
```

Then establish iSCSI connections:

ctrl-a

```
service ctld start

iscsictl -A -p 192.168.56.11 -t
iqn.2016-01.local.sss.private:target0
```

ctrl-b

```
service ctld start

iscsictl -A -p 192.168.56.10 -t
iqn.2016-01.local.sss.private:target0
```

And start mirroring processes:

ctrl-a

```
gmirror label ctrl_b_zil /dev/da0
/dev/md1

ggatel create -t 1 -u 1
/dev/mirror/ctrl_b_zil
```

ctrl-b

```
gmirror label ctrl_a_zil /dev/da0
/dev/md0

ggatel create -t 1 -u 0
/dev/mirror/ctrl_a_zil
```

Now we can enable ZIL on the in-memory mirrored drives:

ctrl-a

```
# ZIL:

zpool add -f ctrl-a_m0 log /dev/ggate0
zfs set sync=always ctrl-a_m0
```

ctrl-b

```
# ZIL:

zpool add -f ctrl-b_m0 log /dev/ggate1
zfs set sync=always ctrl-b_m0
```

Finally we must import both pools on both controllers and disable ZFS stop on any failure:

ctrl-a

```
zpool import -N ctrl-b_m0

zpool set failmode=continue ctrl-a_m0
zpool set failmode=continue ctrl-b_m0
```

ctrl-b

```
zpool import -N ctrl-a_m0

zpool set failmode=continue ctrl-a_m0
zpool set failmode=continue ctrl-b_m0
```

The failover arbitrator

Failover mechanism is not changed from the previous version. Let’s add appropriate iSCSI target definitions to the /etc/ctl.conf file so it will look like:

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

        # ctrl-a ZIL primary copy
        lun 0 {
                path /dev/md0
        }

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

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

        # ctrl-b ZIL primary copy
        lun 1 {
                path /dev/md1
        }

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

And finally assemble the complete arbitration construction:

ctrl-a

```
killall -HUP ctld

iscsictl -M -i 1 -p 192.168.56.11 -t iqn.2016-01.local.sss.private:target0

gmultipath create CTRL_B_BACK /dev/da1 /dev/zvol/ctrl-b_m0/v0
```

ctrl-b

```
killall -HUP ctld

iscsictl -M -i 1 -p 192.168.56.10 -t iqn.2016-01.local.sss.private:target0

gmultipath create CTRL_A_BACK /dev/da1 /dev/zvol/ctrl-a_m0/v0
```

Front-end configuration

Front-end configuration is obviously simple. Change /etc/ctl.conf to add iSCSI target information for the LUNs, accessible for client-hosts. As in previous versions we use portal-group pg1 for the public access:

ctrl-a

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

portal-group pg1 {
        discovery-auth-group no-authentication
        listen 192.168.55.10
}
```

ctrl-b

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

portal-group pg1 {
        discovery-auth-group no-authentication
        listen 192.168.55.11
}
```

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

        # ctrl-a ZIL primary copy
        lun 0 {
                path /dev/md0
        }

        # 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
        }
}
```

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

        # ctrl-b ZIL primary copy
        lun 1 {
                path /dev/md1
        }

        # 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
        }
}
```

The last step is to tell ctld daemon to renew its configuration. Therefore:

ctrl-a	**ctrl-b**
`killall -HUP ctld`	`killall -HUP ctld`

Now we have fully functioning dual-controlled storage system with ZFS and working in-memory cache.

You can test it with our clnt-1 client-host. The testing procedure was completely described in all past papers, therefore we will not repeat it here word for word once again.

Instead we will think of implementing level 2 cache into the BeaST architecture, but it is a story for the future article.

Finally, our traditional warning: the BeaST is in the early development stage! It is for testing only! Do not use it in production or for storing essential data, as you can easily lose your data!