Using SSD as the level two cache for the FreeBSD dual-controller storage array

Mikhail Zakharov zmey20000@yahoo.com

Preface

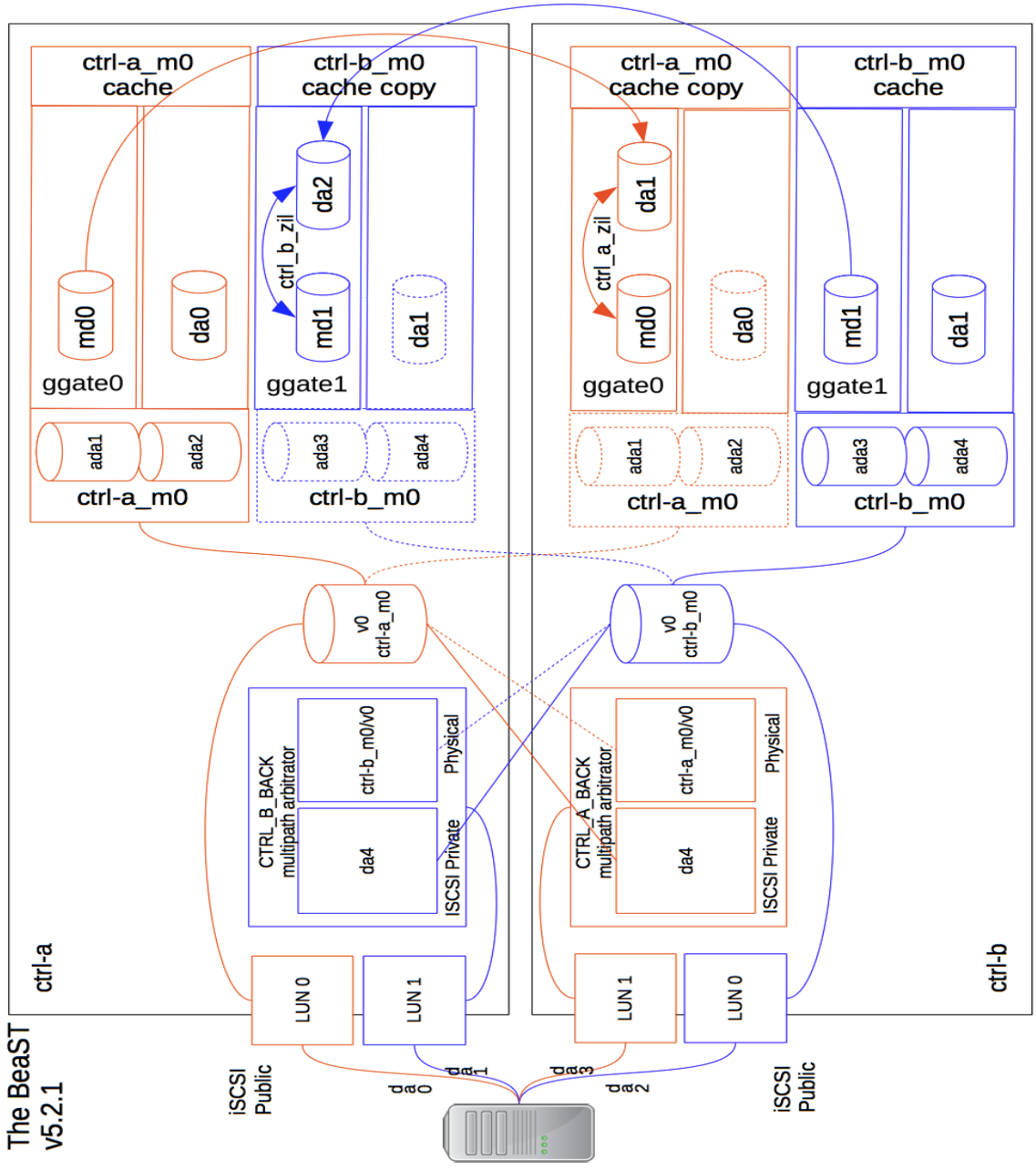
Nowadays storage systems use fast solid-state drives (SSD) not only for storing data volumes, but even as the second layer of cache. In ZFS this feature is implemented by L2ARC and it means that we can easily enable this feature on the BeaST storage system.

We do not need to describe the BeaST VirtualBox environment in detail, as it was already done many times before. Therefore we show only the configuration summary useful for the reproduction of the virtual environment:

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. Host-only adapter (vboxnet1)	IP: 192.168.55.10 Mask: 255.255.255.0	IP: 192.168.55.11 Mask: 255.255.255.0	IP: 192.168.55.20 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 (ada1, ada2, ada3, ada4) – each drive is 100 MB size or more	d00, d01, d10, d11 (ada1, ada2, ada3, ada4) – each drive is 100 MB or more	–
Shareable, fixed-sized virtual drives for ZFS cache on SAS controller.	f00, f10 (da0, da1) – at least 64 MB each	f00, f10 (da0, da1) – at least 64 MB each	–
System virtual drives (Dynamic-sized) on the IDE controller.	ada0 – at least 5 GB to store FreeBSD 10.3-Release default installation	ada0 – at least 5 GB to store FreeBSD 10.3-Release default installation	ada0 – at least 5 GB to store FreeBSD 10.3-Release default installation

As you can see, in addition to the previous version of the BeaST, we will only need to configure two more **fixed-sized shareable** virtual drives attached to the SAS controllers to emulate SSDs.

These SSDs are connected with both storage controllers, therefore in case of one controller's death the BeaST will not lose access to L2 cache through the alive controller. And as we stop L2ARC mirroring for the reasons described in the "Optimizing in-memory cache of the BeaST architecture", we also need not use ggate (GEOM Gate) interlayer for these drives. These changes in the BeaST architecture is shown in the figure below.



The BeaST architecture with cache level two cache on solid-state drives.

Now lets try to run commands to implement the fresh improvements in our architecture.

Basic preparations

As usual for our case install FreeBSD 10.3 Release on the non-shareable drives (ada0 in our case) of both virtual machines which were chosen to be the storage controllers. Typical changes in /etc/rc.conf for our project configuration are shown in the table below:

ctrl-a	**ctrl-b**

hostname="ctrl-a"
 	
ifconfig_em0="inet 192.168.56.10 netmask 255.255.255.0" # Inter-controller LAN
ifconfig_em1="inet 192.168.55.10 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 	
ctld_enable="YES" # Targets
iscsid_enable="YES"   # Initiators
	hostname="ctrl-b"
	
	ifconfig_em0="inet 192.168.56.11 netmask 255.255.255.0" # Inter-controller LAN
	ifconfig_em1="inet 192.168.55.11 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
	ctld_enable="YES" # target
	iscsid_enable="YES" # initiator

Do not forget to 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

ZFS basic configuration

It is very well known from all our previous experiments and it is not complex at all. Therefore just create appropriate zpools and volumes:

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

In-memory cache

Our memory drive (md0/1) to GEOM-gate (ggate0/1) map is the same as in the previous version, as we are not mirroring L2ARC drives anymore:

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

And the commands to enable this structure:

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 (GEOM mirror) gmirror module as we will need it very 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`
`}`
 	
`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`
`  }`
`}`
	`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`
	`  }`
	`}`

Then establish iSCSI connections:

ctrl-a	**ctrl-b**

`service ctld start`
 	
`iscsictl -A -p 192.168.56.11 -t iqn.2016-01.local.sss.private:target0`
	`service ctld start`
	
	`iscsictl -A -p 192.168.56.10 -t iqn.2016-01.local.sss.private:target0`

And start mirroring processes:

ctrl-a	ctrl-b
gmirror label ctrl_b_zil /dev/da0 /dev/md1 ggatel create -t 1 -u 1 /dev/mirror/ctrl_b_zil	gmirror label ctrl_a_zil /dev/da0 /dev/md0 ggatel create -t 1 -u 0 /dev/mirror/ctrl_a_zil

Now we can enable caches: ZIL and both ARC/L2ARC:

ctrl-a	ctrl-b
# ZIL: zpool add -f ctrl-a_m0 log /dev/ggate0 zfs set sync=always ctrl-a_m0 # L2ARC: zpool add ctrl-a_m0 cache /dev/da0 zfs set primarycache=all ctrl-a_m0 zfs set secondarycache=all ctrl-a_m0	# ZIL: zpool add -f ctrl-b_m0 log /dev/ggate1 zfs set sync=always ctrl-b_m0 # L2ARC: zpool add ctrl-b_m0 cache /dev/da1 zfs set primarycache=all ctrl-b_m0 zfs set secondarycache=all ctrl-b_m0

Finally we must import both pools on both controllers and set zpool “failmode” variable to “continue” value in order not to stop on any failure:

ctrl-a	ctrl-b
zpool import -N ctrl-b_m0 zpool set failmode=continue ctrl-a_m0 zpool set failmode=continue ctrl-b_m0	zpool import -N ctrl-a_m0 zpool set failmode=continue ctrl-a_m0 zpool set failmode=continue ctrl-b_m0

The failover arbitrator

Lets add appropriate iSCSI target definitions to the /etc/ctl.conf:

ctrl-a	ctrl-b

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
	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

```
        }
}
```

```
        }
}
```

At last assemble the arbitration construction:

ctrl-a	**ctrl-b**
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	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 all previous versions we use portal-group pg1 to enable public access:

ctrl-a	**ctrl-b**

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
}
 	
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
	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-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-a_m0/v0`
`}`
 	
`lun 1 {`
`  path /dev/multipath/CTRL_B_BACK`
`}`
`}`
	`lun 0 {`
	`  path /dev/zvol/ctrl-b_m0/v0`
	`}`
	
	`lun 1 {`
	`  path /dev/multipath/CTRL_A_BACK`
	`}`
	`}`

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

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

That is all from the storage system side. We can connect now our old clnt-1 client virtual machine to test the newly created BeaST storage system. This procedure was described scrupulously in the earlier papers, therefore refer to the instructions on the BeaST project page.

Conclusions

As you can see, the difference between this and the previous version of the architecture concerns only the cache configuration. But that's one small piece of code, one giant leap for the BeaST as a whole. Putting L2 cache to SSD we can confirm that our project has the power to implement all common signs of the serious modern and reliable storage system.

We will continue to develop it by adding features, software, HA, fail-back and other automation mechanisms, improving algorithms and stability. But now we are really in need of physical hardware to test our ideas, concepts and prototypes.

And our traditional warning at the end: the BeaST is currently in the early development stage! Use it for testing purposes only! Do not implement it in production or for storing essential data, as you can lose your data!