Procedures for experimenting with SSD disk caching of regular hard disk partitions or logical volumes in Linux

Setup Proposal

You may have one very fast and modern internal SSD disk, say 128 GB. You may fdisk the disk in 4 partitions of 32 GB each. You use sda1 partition to host the main Linux OS. This sda1 will be your boot disk. The other sda2, sda3 and sda4 partitions could be 32 GB each and can be used as a very fast writeback cache for three other big, slow and enormous size partitions on these multi-Terabyte disks you have. These 3 SSD cache partitions can even cache three Linux LVM logical volumes (each containing many physical disk partitions behind it..)

Example: sdb1 can use sda2 as its fast cache,
vg2-lvol1 can use sda3 as its fast cache,
sdc2 can use sda4 as its fast cache.

For all other partitions of sdb, sdc and any other logical volumes we decided that they need no caching.
For them, linux own caching (its “cache” and “buffers”) can be more than enough for most usage scenarios.

So we can have fast machine boot and fast linux command execution, * AND * some important big sized partitions and logical volumes in our rotating disks transparently cached to the SSD.

A bit of theory

This setup is based on a Linux module called Flashcache. It was developed in 2010 by a Facebook team.

A detailed description, plus specific details (that you will surely need if you decide to put (or boot) your whole Linux OS from a cached partition) is found here:

https://wiki.archlinux.org/index.php/Flashcache

{ File flascache-sa-guide.txt has details on how to create and load flashcache volumes.}

Also these could be very interesting:

(Especially the first that nicely describes how to configure Linux LVM.)

http://linuxwindowsmaster.com/tag/vgdisplay/

http://www.cyberciti.biz/faq/howto-create-linux-ram-disk-filesystem/

Detailed procedure

First we see what we have now in our machine (lst is just a “ls -altr” alias of mine):

```
free -t
uname -a
mount
df -h
cat /etc/fstab
lst /dev/mapper/
lst /dev/ram*

cd /usr/src
lst
cd kernel-devel-3.2.18-pclos2.bfs/      (Your linux link should show your kernel.)
lst
find . -name "*.ko*" | less
du -m |tail
```

I create a ramdisk because I do not have a SSD. Linux will maybe need a “ramdisk_size=80000” parameter to be passed in its kernel (while booting, as a grub entry) to be able to create such a big sized ramdisk.

```
mkfs -q /dev/ram4 80000
```

Then we download from the distribution page https://github.com/facebook/flashcache
the source files as a ZIP (choose tab “ZIP”), we unzip and begin to compile:

```
cd ~/Downloads/
lst
cd flashcache-master/
lst

uname -r    (To find your kernel Release.)
make KERNEL_TREE=/usr/src/kernel-devel-3.2.18-pclos2.bfs    (Here you must put your own linux src dir.)

find /lib/modules/3.2.18-pclos2.bfs/ -name "*.ko*" | wc -l      (I count my Linux modules.)
pwd
lst
find . -name "*.ko"
```

If it compiled correctly, find should now show your new module.

We also check if the flashcache programs have been created,

```
lst src/utils/{flashcache_create,flashcache_load,flashcache_destroy}
```

and then we install the programs and the modules:

```
make KERNEL_TREE=/usr/src/kernel-devel-3.2.18-pclos2.bfs install

df -m
```

I will unmount now a partition of mine to experiment with. It is an encrypted one.

(Unmounting is also the way to “e2fsck -fpv” an encrypted partition if needed.)

```
umount /mnt/tc2
sync;sync

history | grep ram

lst /mnt
mkdir /mnt/cache4
```

This is the command that creates the mapping for our setup:

```
flashcache_create -p back cache_dev0 /dev/ram4 /dev/mapper/xyzcrypt2
```

If we were caching a logical volume we would use something like: /dev/mapper/vg2-lvol1

```
lst /dev/mapper/     (It should show the newly created device:  cache_dev0 )

mount /dev/mapper/cache_dev0 /mnt/cache4
df -m
df -h
```

The “df -m” output for /mnt/cache4 must now show the big slow regular disk partition (or logical volume) capacity size. (And NOT the fast SSD cache size.)

To get some info on current status:

```
lst /mnt/cache4
dmsetup status

modprobe
modinfo
```

```
modinfo flashcache
lsmod | grep "^flashcache"
```

We test the throughput of our cached disks by running some intensive and/or repeated jobs, and some programs or benchmarks if we have any.

Even an openoffice (that is loading its binaries from the cached slow hard disk and is also opening a huge excel sheet from another cached slow disk or logical volume) could do..

In an accompanying example pdf I have used the “hdparm -t” utility for testing.

When we finish our tests,

We unmount the cache and then completely remove it:

```
sync;sync
umount /mnt/cache4
sync;sync
df -m
dmsetup remove cache_dev0
sync;sync
lst /dev/mapper/
```

And do not worry, these commands “flash” the cache before removing it.

Since my encrypted disk is now unmounted (but accessible), I check its filesystem:

```
e2fsck -fpv /dev/mapper/xyzcrypt2
mount /dev/mapper/xyzcrypt2 /mnt/tc2
df -m
```

That is the basic theory. Experiment !! Build on it !!

Make combinations !! Use your old external disks !! Have fun !!

George Moraitis

February, 2013

PS. An accompanying pdf shows my test commands and my linux responses.
See file **flashcache-results-to-see.pdf**