redhat.

DM-Cache

Marc Skinner
Principal Solutions Architect

Why Cache?

- Spinning disks are slow!

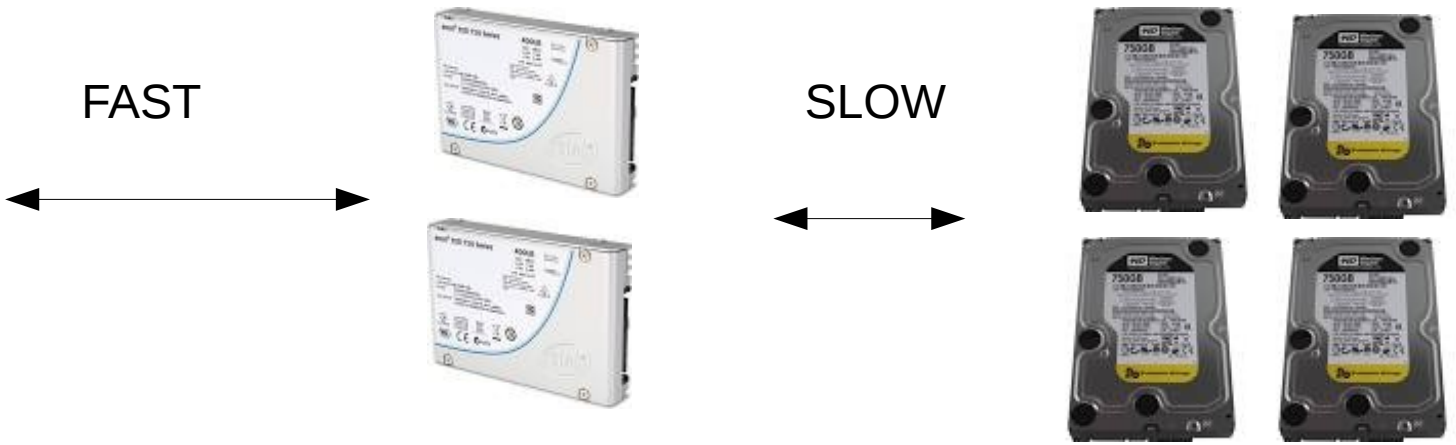
- Solid state disks (SSD) are fast!!

- Non-Volatile Memory Express (NVMe) devices are insane!!!

Why not?

- Hybrid drives make 5900rpm drives act like 7200rpm drives with very small on board SSD cache

- Hmm, talk to FAST and move to SLOW?



Linux Caching Options

- DM-Cache
  - Oldest and most stable. Developed in 2006 by IBM research group, and merged into Linux kernel tree in version 3.9. Uses the device-mapper framework to cache a slower device
- FlashCache
  - Kernel module inspired by dm-cache and developed/maintained by Facebook. Also uses the device-mapper framework.
- EnhanceIO, RapidCache
  - Both variations of FlashCache
- BCache
  - Newest option and does not rely on device-mapper framework

DM-Cache Modes

- write-through
  - Red Hat default
  - Write requests are not returned until the data reaches the origin and the cache device
- write-back
  - Writes go only to the cache device
- pass-through
  - Used to by pass the cache, used if cache is corrupt

DM-Cache Setup

- Enable discards first

```
# vi /etc/lvm/lvm.conf
issue_discards = 1
# dracut -f
# sync
# reboot
```

DM-Cache Setup

- Create PV, VG and LV with Cache

pvcreate /dev/md2 (raid 10 - 6 x 250gb SSD)

pvcreate /dev/md3 (raid 10 - 6 x 2tb SATA)

vgcreate vg_iscsi /dev/md3 /dev/md2

lvcreate -l 100%FREE -n lv_sata vg_iscsi /dev/md3

lvcreate -L **5G** -n **lv_cache_meta** vg_iscsi /dev/md2

lvcreate -L 650G -n **lv_cache** vg_iscsi /dev/md2

lvconvert **--type cache-pool** /dev/vg_iscsi/lv_cache **--poolmetadata** /dev/vg_iscsi/lv_cache_meta --chunksize 256

lvconvert **--type cache** /dev/vg_iscsi/lv_sata **--cachepool** /dev/vg_iscsi/lv_cache

dmsetup status vg_iscsi-lv_sata

0 1172028620 8 cache 8 32938/1310720 128 1995192/11059200 3349 79 2008845 4646 0 1758463 0 1 **writethrough** 2 migration_threshold 2048 smq 0 rw -

DM-Cache Setup – use writeback

- Create PV, VG and LV with Cache

pvcreate /dev/md2 (raid 10 - 6 x 250gb SSD)

pvcreate /dev/md3 (raid 10 - 6 x 2tb SATA)

vgcreate vg_iscsi /dev/md3 /dev/md2

lvcreate -l 100%FREE -n lv_sata vg_iscsi /dev/md3

lvcreate -L 5G -n lv_cache_meta vg_iscsi /dev/md2

lvcreate -L 650G -n lv_cache vg_iscsi /dev/md2

lvconvert --type cache-pool **--cachemode writeback** /dev/vg_iscsi/lv_cache --poolmetadata /dev/vg_iscsi/lv_cache_meta --chunksize 256

lvconvert --type cache /dev/vg_iscsi/lv_sata --cachepool /dev/vg_iscsi/lv_cache

dmsetup status vg_iscsi-lv_sata

0 11720286208 cache 8 21175/1310720 128 2285546/10649600 1543940 178 2497985 11513882 0 2285546 457855 1 **writeback** 2 migration_threshold 2048 smq 0 rw -

DM-Cache Status

View cache hits/misses

lvs -o name,cache_read_hits,cache_read_misses vg_iscsi/lv_sata

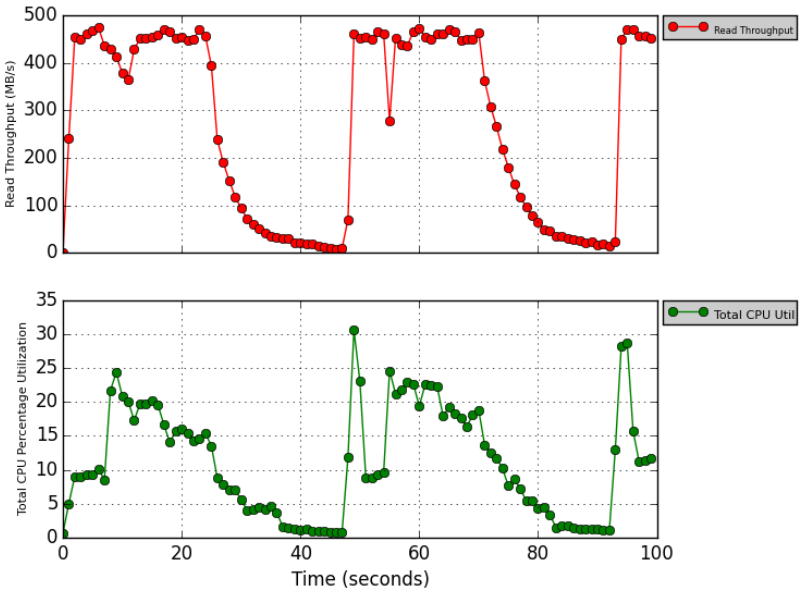
LV	CacheReadHits	CacheReadMisses
lv_sata	123	62

View size/status

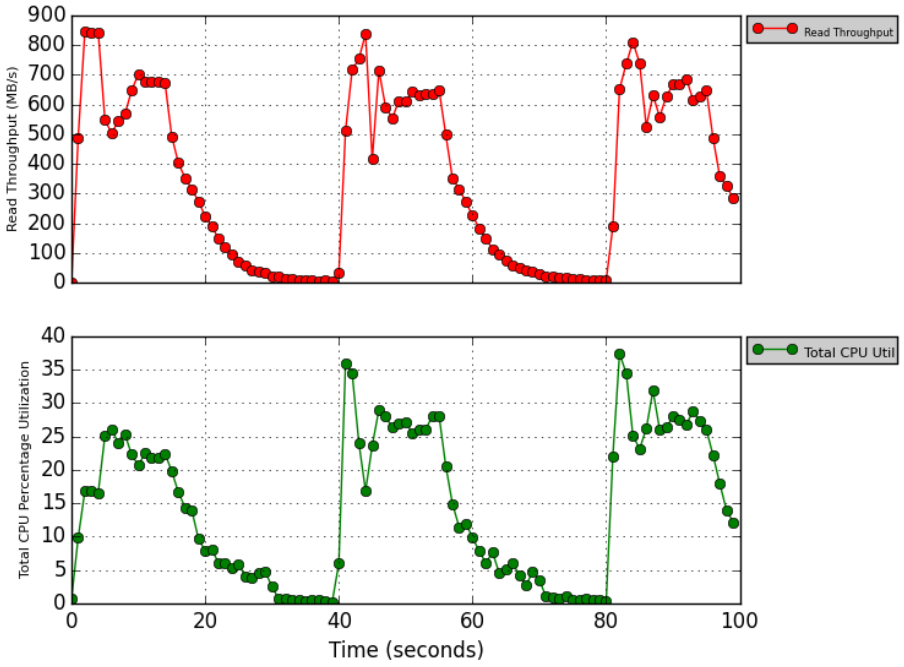
lvs vg_iscsi/lv_sata

LV	VG	Attr	LSize	Pool	Origin	Data%	Meta%	Move	Log	Cpy%Sync	Convert
lv_sata	vg_iscsi	Cwi-a-C---	**5.46t**	[lv_cache]	[lv_sata_corig]	0.00	1.62			0.00	

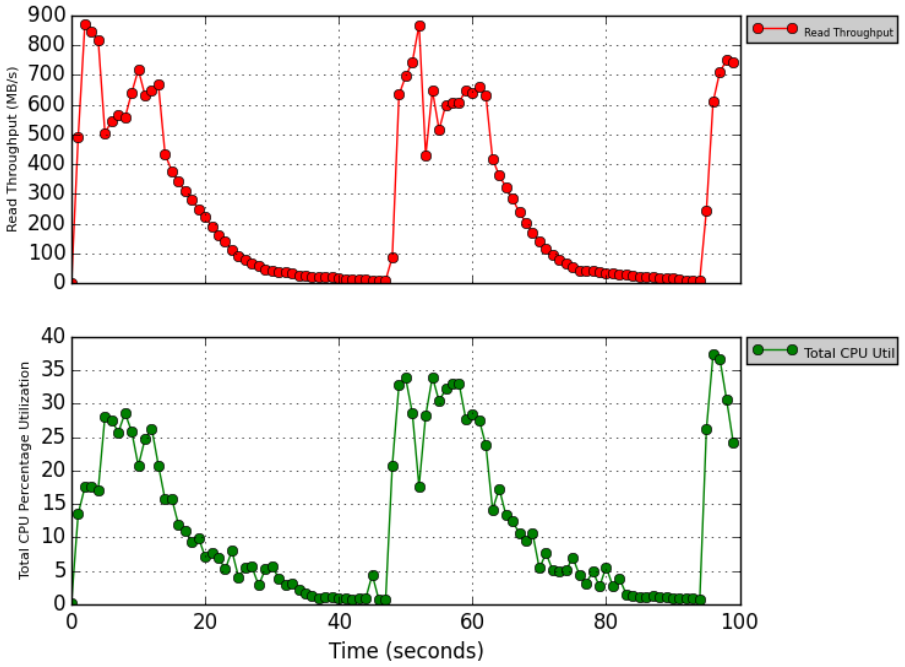
42G read on Ext4



SPINNING R10

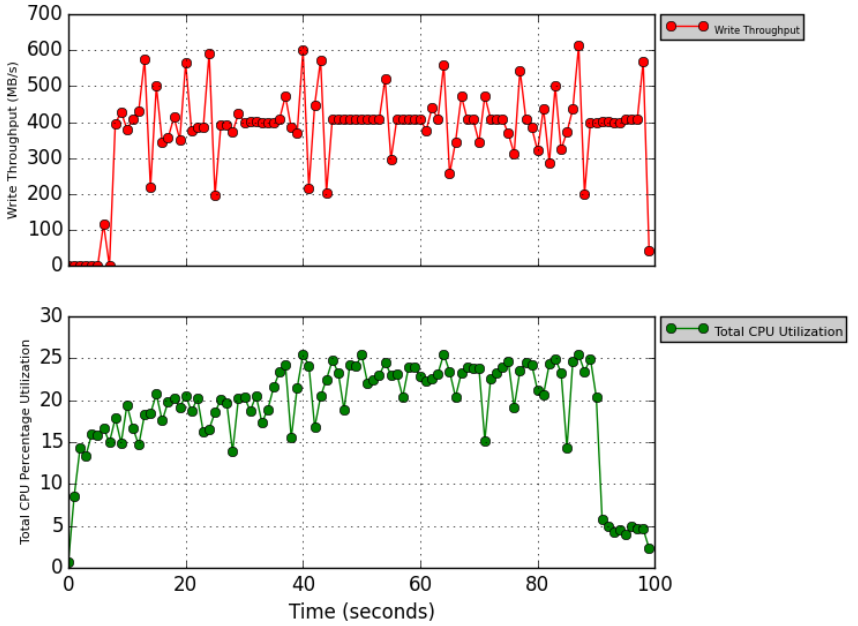


SSD R10

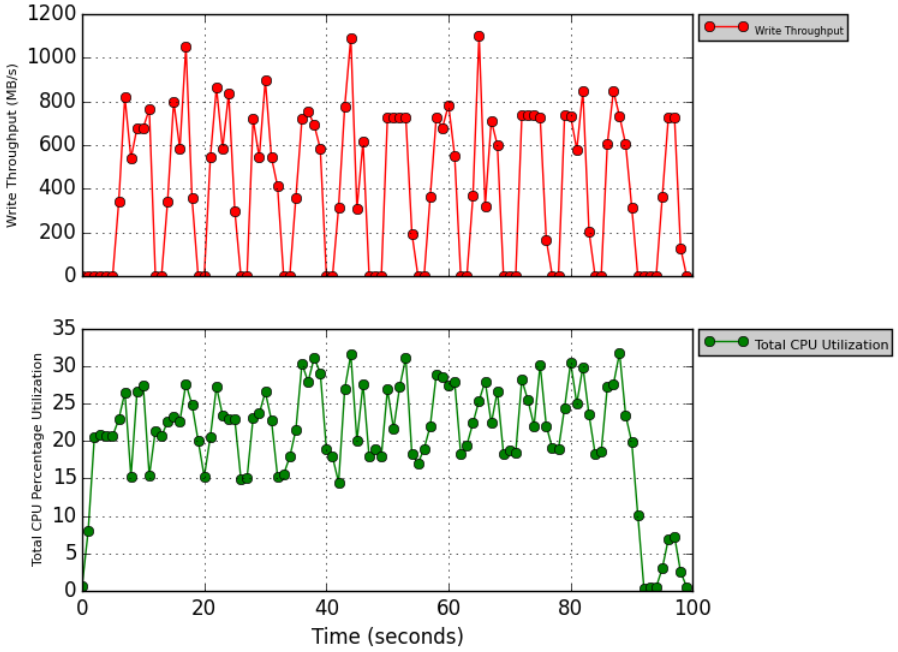


DM CACHE R10/R10 WB

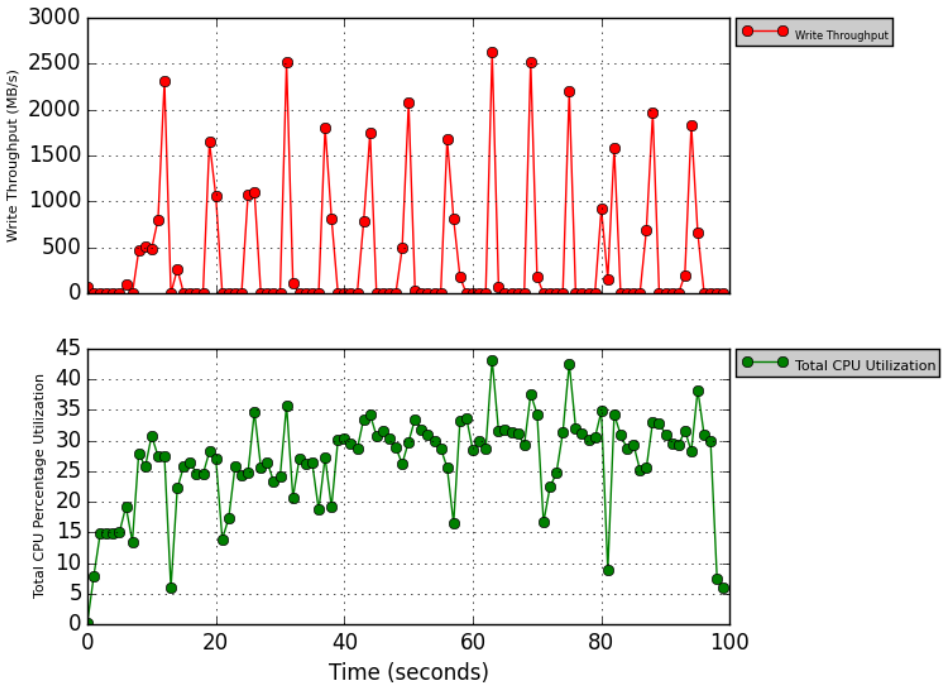
42G write on Ext4



SPINNING R10

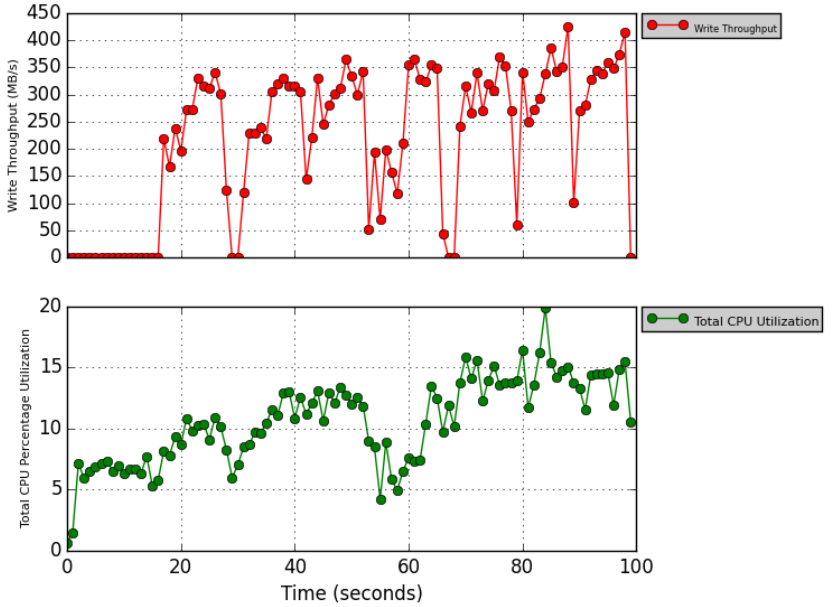


SSD R10

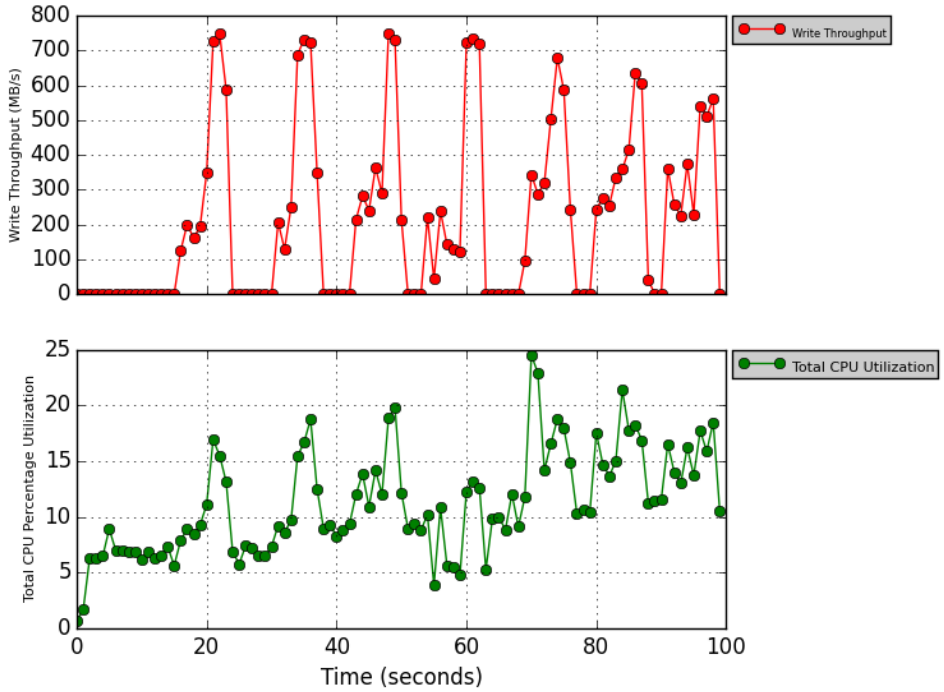


DM CACHE R10/R10 WB

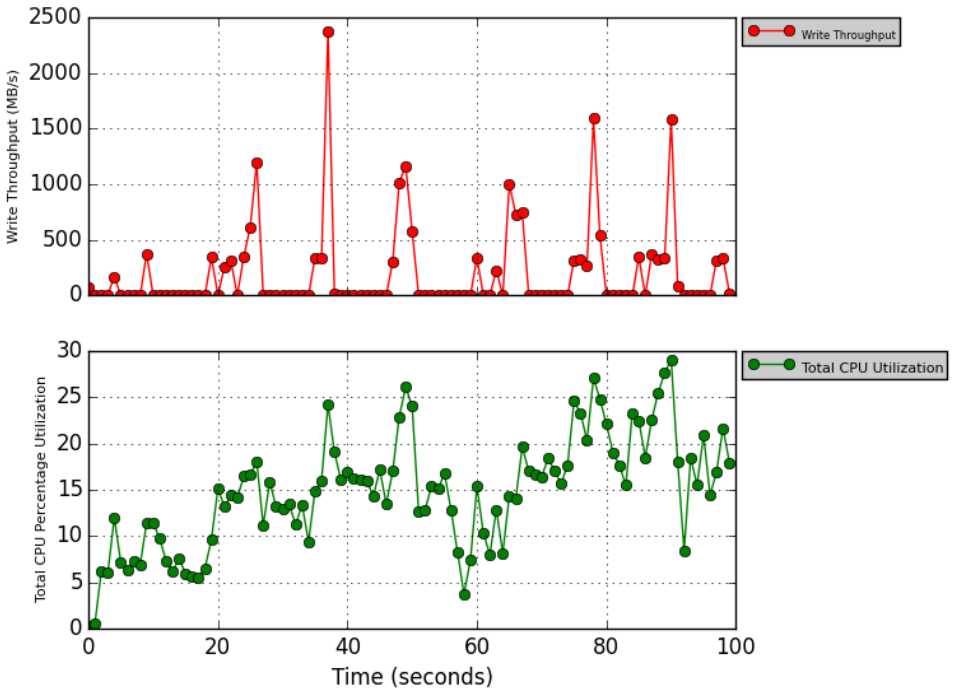
42G small files EXT4



SPINNING R10



SSD R10



DM CACHE R10/R10 WB

Did you know?

mkfs.ext4

Fast format ... but there is a catch

Once you mount the new ext4 file system – watch out!

Look for running process "ext4lazyinit"

According to iotop it was consuming 5-11% of my overall IO

yum -y install iotop

What is it: A process that backgrounds the creation of the remaining index nodes which are used to reference leaf nodes which is referencing multiple extents. Basically pointers to data on the filesystem.

Lazy vs non-lazy

Force format to not be LAZY

mkfs -t ext4 **-E lazy_itable_init=0,lazy_journal_init=0** /dev/mapper/myVG/myLV

Takes MUCH, MUCH longer, like ext3 :(

Format OFF

EXT3 vs EXT4 (650gb LV running on raid 10 - 6 x 250gb SSD)

EXT3: **13.46s**

EXT4 lazy: 0.17s (+ about 2-3 minutes for ext4lazyinit to finish in the background)

EXT4 not lazy: **12.7s**

Questions?