

facebook

facebook

FlashCache

Mohan Srinivasan
Mark Callaghan
July 2010

FlashCache at Facebook

- What
 - We want to use some Flash storage on existing servers
 - We want something that is simple to deploy and use
 - Our IO access patterns benefit from a cache
- Who
 - Mohan Srinivasan – design and implementation
 - Paul Saab – platform and MySQL integration
 - Mark Callaghan - benchmarking

Introduction

- Block cache for Linux - write back and write through modes
- Layered below the filesystem at the top of the storage stack
- Cache Disk Blocks on fast persistent storage (Flash, SSD)
- Loadable Linux Kernel module, built using the Device Mapper (DM)
- Primary use case InnoDB, but general purpose
- Based on dm-cache by Prof. Ming

Caching Modes

Write Back

- Lazy writing to disk
- Persistent across reboot
- Persistent across device removal

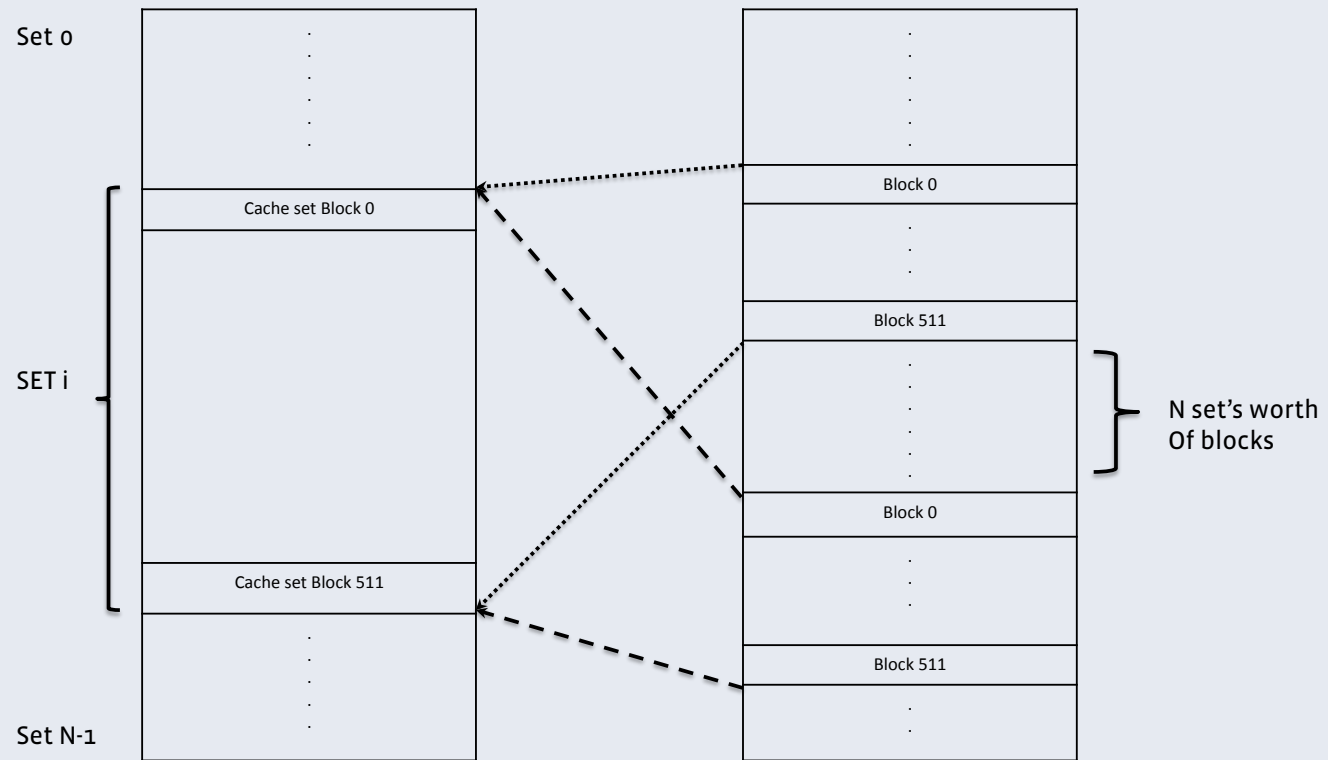
Write Through

- Non-persistent
- Are you a pessimist?

Cache Structure

- Set associative hash
- Hash with fixed sized buckets (sets) with linear probing within a set
- 512-way set associative by default
- dbn: Disk Block Number, address of block on disk
- $\text{Set} = (\text{dbn} / \text{block size} / \text{set size}) \bmod (\text{number of sets})$
- Sequential range of dbns map onto a single sets

Cache Structure



Write Back

- Replacement policy is FIFO (default) or LRU within a set
- Switch on the fly between FIFO/LRU (sysctl)
- Metadata per cache block: 24 bytes in memory, 16 bytes on ssd
- On ssd metadata per-slot
 - <dbn, block state>
- In memory metadata per-slot:
 - <dbn, block state, LRU chain pointers, misc>

Write Through

- Replacement policy is FIFO
- Metadata per slot
 - 17 bytes (memory), no metadata stored on ssd
- In memory metadata per-slot
 - <dbn, block state, checksum>

Reads

- Compute cache set for dbn
- Cache Hit
 - Verify checksums if configured
 - Serve read out of cache
- Cache Miss
 - Find free block or reclaim block based on replacement policy
 - Read block from disk and populate cache
 - Update block checksum if configured
 - Return data to user

Write Through - writes

- Compute cache set for dbn
- Cache hit
 - Get cached block
- Cache miss
 - Find free block or reclaim block
- Write data block to disk
- Write data block to cache
- Update block checksum

Write Back - writes

- Compute cache set for dbn
- Cache Hit
 - Write data block into cache
 - If data block not DIRTY, synchronously update on-ssd cache metadata to mark block DIRTY
- Cache miss
 - Find free block or reclaim block based on replacement policy
 - Write data block to cache
 - Synchronously update on-ssd cache metadata to mark block DIRTY

Small or uncacheable requests

- First invalidate blocks that overlap the requests
 - There are at most 2 such blocks
 - For Write Back, if the overlapping blocks are DIRTY they are cleaned first then invalidated
- Uncacheable full block reads are served from cache in case of a cache hit.
- Perform disk IO
- Repeat invalidation to close races which might have caused the block to be cached while the disk IO was in progress

Write Back policy

- Default expiration of 30 seconds (work in progress)
- When dirty blocks in a set exceeds configurable threshold, clean some blocks
 - Blocks selected for writeback based on replacement policy
 - Default dirty threshold 20%. Set higher for write heavy workloads
- Sort selected blocks and pickup any other blocks in set that can be contiguously merged with these
- Writes merged by the IO scheduler

Write Back – cache metadata overhead

- In-Memory cache metadata memory footprint
 - 300GB/4KB cache -> ~1.8GB
 - 160GB/4KB cache -> ~960MB
- Cache metadata writes/file system write
 - Worst case is 2 cache metadata updates per write
 - (VALID->DIRTY, DIRTY->VALID)
 - Average case is much lower because of cache write hits and batching of cache metadata updates

Write Through – cache metadata overhead

- In-Memory Cache metadata footprint
 - 300GB/4KB cache -> ~1.3GB
 - 160GB/4KB cache -> ~700MB
- Cache metadata writes per file system write
 - 1 cache data write per file system write

Write Back – metadata updates

- Cache (on-ssd) metadata only updated on writes and block cleanings (VALID-→DIRTY or DIRTY-→VALID)
- Cache (on-ssd) metadata not updated on cache population for reads
- Reload after an unclean shutdown only loads DIRTY blocks
- Fast and Slow cache shutdowns
 - Only metadata is written on fast shutdown. Reload loads both dirty and clean blocks
 - Slow shutdown writes all dirty blocks to disk first, then writes out metadata to the ssd. Reload only loads clean blocks.
- Metadata updates to multiple blocks in same sector are batched

Torn Page Problem

- Handle partial block write caused by power failure or other causes
- Problem exists for Flashcache in Write Back mode
- Detected via block checksums
 - Checksums are disabled by default
 - Pages with bad checksums are not used
- Checksums increase cache metadata writes and memory footprint
 - Update cache metadata checksums on DIRTY->DIRTY block transitions for Write Back
 - Each per-cache slot grows by 8 bytes to hold the checksum (a 33% increase from 24 bytes to 32 bytes for the Write Back case).

Cache controls for Write Back

- Work best with O_DIRECT file access
- Global modes – Cache All or Cache Nothing
 - Cache All has a blacklist of pids and tgids
 - Cache Nothing has a whitelist of pids and tgids
- tgids can be used to tag all pthreads in the group as cacheable
- Exceptions for threads within a group are supported
- List changes done via FlashCache ioctls
- Cache can be read but is not written for non-cacheable tgids and pids
- We modified MySQL and scp to use this support

Cache Nothing policy

- If the thread id is whitelisted, cache all IOs for this thread
- If the tgid is whitelisted, cache all IOs for this thread
- If the thread id is blacklisted do not cache IOs

Cache control example

- We use Cache Nothing mode for MySQL servers
- The mysqld tgid is added to the whitelist
 - All IO done by it is cacheable
 - Writes done by other processes do not update the cache
- Full table scans done by mysqldump use a hint that directs mysqld to add the query's thread id to the blacklist to avoid wiping FlashCache
 - `select /* SQL_NO_FCACHE */ pk, col1, col2 from foobar`

Utilities

- flashcache_create
 - flashcache_create -b 4k -s 10g mysql /dev/flash /dev/disk
- flashcache_destroy
 - flashcache_destory /dev/flash
- flashcache_load

sysctl -a | grep flash

dev.flashcache.cache_all = 0

dev.flashcache.fast_remove = 0

dev.flashcache.zero_stats = 1

dev.flashcache.write_merge = 1

dev.flashcache.reclaim_policy = 0

dev.flashcache.pid_expiry_secs = 60

dev.flashcache.max_pids = 100

dev.flashcache.do_pid_expiry = 0

dev.flashcache.max_clean_ios_set = 2

dev.flashcache.max_clean_ios_total = 4

dev.flashcache.debug = 0

dev.flashcache.dirty_thresh_pct = 20

dev.flashcache.stop_sync = 0

dev.flashcache.do_sync = 0

Removing FlashCache

- `umount /data`
- `dmesetup remove mysql`
- `flashcache_destroy /dev/flash`

cat /proc/flashcache_stats

```
reads=4 writes=0 read_hits=0 read_hit_percent=0 write_hits=0  
write_hit_percent=0 dirty_write_hits=0 dirty_write_hit_percent=0  
replacement=0 write_replacement=0 write_invalidates=0  
read_invalidates=0 pending_enqueues=0 pending_inval=0  
metadata_dirties=0 metadata_cleans=0 cleanings=0 no_room=0  
front_merge=0 back_merge=0 nc_pid_adds=0 nc_pid_dels=0  
nc_pid_drops=0 nc_expiry=0 disk_reads=0 disk_writes=0 ssd_reads=0  
ssd_writes=0 uncached_reads=169 uncached_writes=128
```

Future Work

- Cache mirroring
 - SW RAID 0 block device as a cache
- Online cache resize
 - No shutdown and recreate
- Support for ATA trim
 - Discard blocks no longer in use
- Fix the torn page problem
 - Use shadow pages

Resources

- GitHub : facebook/flashcache
- Mailing list : flashcache-dev@googlegroups.com
- <http://facebook.com/MySQLatFacebook>
- Email :
 - mohan@facebook.com (Mohan Srinivasan)
 - ps@facebook.com (Paul Saab)
 - mcallaghan@facebook.com (Mark Callaghan)

facebook

(c) 2009 Facebook, Inc. or its licensors. "Facebook" is a registered trademark of Facebook, Inc.. All rights reserved. 1.0