Cache

Introduction

dm-cache is a device mapper target written by Joe Thornber, Heinz Mauelshagen, and Mike Snitzer.

It aims to improve performance of a block device (eg, a spindle) by dynamically migrating some of its data to a faster, smaller device (eg, an SSD).

This device-mapper solution allows us to insert this caching at different levels of the dm stack, for instance above the data device for a thin-provisioning pool. Caching solutions that are integrated more closely with the virtual memory system should give better performance.

The target reuses the metadata library used in the thin-provisioning library.

The decision as to what data to migrate and when is left to a plug-in policy module. Several of these have been written as we experiment, and we hope other people will contribute others for specific io scenarios (eg. a vm image server).

Glossary

Migration
: Movement of the primary copy of a logical block from one device to the other.

Promotion
: Migration from slow device to fast device.

Demotion
: Migration from fast device to slow device.

The origin device always contains a copy of the logical block, which may be out of date or kept in sync with the copy on the cache device (depending on policy).

Design

Sub-devices

The target is constructed by passing three devices to it (along with other parameters detailed later):

1. An origin device - the big, slow one.
2. A cache device - the small, fast one.
3. A small metadata device - records which blocks are in the cache, which are dirty, and extra hints for use by the policy object. This information could be put on the cache device, but having it separate allows the volume manager to configure it differently, e.g. as a mirror for extra robustness. This metadata device may only be used by a single cache device.

Fixed block size

The origin is divided up into blocks of a fixed size. This block size is configurable when you first create the cache. Typically we've been using block sizes of 256KB - 1024KB. The block size must be between 64 sectors (32KB) and 2097152 sectors (1GB) and a multiple of 64 sectors (32KB).

Having a fixed block size simplifies the target a lot. But it is something of a compromise. For instance, a small part of a block may be getting hit a lot, yet the whole block will be promoted to the cache. So large block sizes are bad because they waste cache space. And small block sizes are bad because they increase the amount of metadata (both in core and on disk).

Cache operating modes

The cache has three operating modes: writeback, writethrough and passthrough.

If writeback, the default, is selected then a write to a block that is cached will go only to the cache and the block will be marked dirty in the metadata.

If writethrough is selected then a write to a cached block will not complete until it has hit both the origin and cache devices. Clean blocks should remain clean.

If passthrough is selected, useful when the cache contents are not known to be coherent with the origin device, then all reads are served from the origin device (all reads miss the cache) and all writes are forwarded to the origin device; additionally, write hits cause cache block invalidates. To enable passthrough mode the cache must be clean. Passthrough mode allows a cache device to be activated without having to worry about coherency. Coherency that exists is maintained, although the cache will gradually cool as writes take place. If the coherency of the cache can later be verified, or established through use of the "invalidate_cblocks" message, the cache device can be transitioned to writethrough or writeback mode while still warm. Otherwise, the cache contents can be discarded prior to transitioning to the desired operating mode.

A simple cleaner policy is provided, which will clean (write back) all dirty blocks in a cache. Useful for decommissioning a cache or when shrinking a cache. Shrinking the cache's fast device requires all cache blocks, in the area of the cache being removed, to be clean. If the area being removed from the cache still contains dirty blocks the resize will fail. Care must be taken to never reduce the volume used for the cache's fast device until the cache is clean. This is of particular importance if writeback mode is used. Writethrough and passthrough modes already maintain a clean cache. Future support to partially clean the cache, above a specified threshold, will allow for keeping the cache warm and in writeback mode during resize.

Migration throttling

Migrating data between the origin and cache device uses bandwidth. The user can set a throttle to prevent more than a certain amount of migration occurring at any one time. Currently we're not taking any account of normal io traffic going to the devices. More work needs doing here to avoid migrating during those peak io moments.

For the time being, a message "migration_threshold <#sectors>" can be used to set the maximum number of sectors being migrated, the default being 2048 sectors (1MB).

Updating on-disk metadata

On-disk metadata is committed every time a FLUSH or FUA bio is written. If no such requests are made then commits will occur every second. This means the cache behaves like a physical disk that has a volatile write cache. If power is lost you may lose some recent writes. The metadata should always be consistent in spite of any crash.

The 'dirty' state for a cache block changes far too frequently for us to keep updating it on the fly. So we treat it as a hint. In normal operation it will be written when the dm device is suspended. If the system crashes all cache blocks will be assumed dirty when restarted.

Per-block policy hints

Policy plug-ins can store a chunk of data per cache block. It's up to the policy how big this chunk is, but it should be kept small. Like the dirty flags this data is lost if there's a crash so a safe fallback value should always be possible.

Policy hints affect performance, not correctness.

Policy messaging

Policies will have different tunables, specific to each one, so we need a generic way of getting and setting these. Device-mapper messages are used. Refer to cache-policies.txt.

Discard bitset resolution

We can avoid copying data during migration if we know the block has been discarded. A prime example of this is when mkfs discards the whole block device. We store a bitset tracking the discard state of blocks. However, we allow this bitset to have a different block size from the cache blocks. This is because we need to track the discard state for all of the origin device (compare with the dirty bitset which is just for the smaller cache device).

Target interface

Constructor

```
cache <metadata dev> <cache dev> <origin dev> <block size>
      <#feature args> [<feature arg>]*
      <policy> <#policy args> [policy args]*
```

metadata dev	fast device holding the persistent metadata
cache dev	fast device holding cached data blocks
origin dev	slow device holding original data blocks
block size	cache unit size in sectors
#feature args	number of feature arguments passed
feature args	writethrough or passthrough (The default is writeback.)
policy	the replacement policy to use
#policy args	an even number of arguments corresponding to key/value pairs passed to the policy
policy args	key/value pairs passed to the policy E.g. 'sequential_threshold 1024' See cache-policies.txt for details.

Optional feature arguments are:

writethrough	write through caching that prohibits cache block content from being different from origin block content. Without this argument, the default behaviour is to write back cache block contents later for performance reasons, so they may differ from the corresponding origin blocks.
passthrough	a degraded mode useful for various cache coherency situations (e.g., rolling back snapshots of underlying storage). Reads and writes always go to the origin. If a write goes to a cached origin block, then the cache block is invalidated. To enable passthrough mode the cache must be clean.
metadata2	use version 2 of the metadata. This stores the dirty bits in a separate btree, which improves speed of shutting down the cache.
no_discard_passdown	disable passing down discards from the cache to the origin's data device.

A policy called 'default' is always registered. This is an alias for the policy we currently think is giving best all round performance.

As the default policy could vary between kernels, if you are relying on the characteristics of a specific policy, always request it by name.

Status

```
<metadata block size> <#used metadata blocks>/<#total metadata blocks>
<cache block size> <#used cache blocks>/<#total cache blocks>
<#read hits> <#read misses> <#write hits> <#write misses>
<#demotions> <#promotions> <#dirty> <#features> <features>*
<#core args> <core args>* <policy name> <#policy args> <policy args>*
<cache metadata mode>
```

metadata block size	Fixed block size for each metadata block in sectors
#used metadata blocks	Number of metadata blocks used
#total metadata blocks	Total number of metadata blocks
cache block size	Configurable block size for the cache device in sectors
#used cache blocks	Number of blocks resident in the cache
#total cache blocks	Total number of cache blocks
#read hits	Number of times a READ bio has been mapped to the cache
#read misses	Number of times a READ bio has been mapped to the origin
#write hits	Number of times a WRITE bio has been mapped to the cache
#write misses	Number of times a WRITE bio has been mapped to the origin
#demotions	Number of times a block has been removed from the cache
#promotions	Number of times a block has been moved to the cache
#dirty	Number of blocks in the cache that differ from the origin
#feature args	Number of feature args to follow
feature args	'writethrough' (optional)
#core args	Number of core arguments (must be even)
core args	Key/value pairs for tuning the core e.g. migration_threshold
policy name	Name of the policy
#policy args	Number of policy arguments to follow (must be even)
policy args	Key/value pairs e.g. sequential_threshold
cache metadata mode	ro if read-only, rw if read-write In serious cases where even a read-only mode is deemed unsafe no further I/O will be permitted and the status will just contain the string 'Fail'. The userspace recovery tools should then be used.
needs_check	'needs_check' if set, '-' if not set A metadata operation has failed, resulting in the needs_check flag being set in the metadata's superblock. The metadata device must be deactivated and checked/repaired before the cache can be made fully operational again. '-' indicates needs_check is not set.

Messages

Policies will have different tunables, specific to each one, so we need a generic way of getting and setting these. Device-mapper messages are used. (A sysfs interface would also be possible.)

The message format is:

```
<key> <value>
```

E.g.:

```
dmsetup message my_cache 0 sequential_threshold 1024
```

Invalidation is removing an entry from the cache without writing it back. Cache blocks can be invalidated via the invalidate_cblocks message, which takes an arbitrary number of cblock ranges. Each cblock range's end value is "one past the end", meaning 5-10 expresses a range of values from 5 to 9. Each cblock must be expressed as a decimal value, in the future a variant message that takes cblock ranges expressed in hexadecimal may be needed to better support efficient invalidation of larger caches. The cache must be in passthrough mode when invalidate_cblocks is used:

```
invalidate_cblocks [<cblock>|<cblock begin>-<cblock end>]*
```

E.g.:

```
dmsetup message my_cache 0 invalidate_cblocks 2345 3456-4567 5678-6789
```

Examples

The test suite can be found here:

https://github.com/jthornber/device-mapper-test-suite

```
dmsetup create my_cache --table '0 41943040 cache /dev/mapper/metadata \
        /dev/mapper/ssd /dev/mapper/origin 512 1 writeback default 0'
dmsetup create my_cache --table '0 41943040 cache /dev/mapper/metadata \
        /dev/mapper/ssd /dev/mapper/origin 1024 1 writeback \
        mq 4 sequential_threshold 1024 random_threshold 8'
```