Full credit is given to the above companies including the OS that this PDF file was generated!

Red Hat Enterprise Linux Release 9.2 Manual Pages on 'lvmcache.7' command

$ man lvmcache.7

LVMCACHE(7) LVMCACHE(7)

NAME

lvmcache ? LVM caching

DESCRIPTION

lvm(8) includes two kinds of caching that can be used to improve the performance of a Logical Volume (LV). When caching, varying subsets of an LV's data are temporarily stored on a smaller, faster device (e.g. an SSD) to improve the performance of the LV.

To do this with lvm, a new special LV is first created from the faster device. This LV will hold the cache. Then, the new fast LV is attached to the main LV by way of an lvconvert command. lvconvert inserts one of the device mapper caching targets into the main LV's i/o path. The de? vice mapper target combines the main LV and fast LV into a hybrid de? vice that looks like the main LV, but has better performance. While the main LV is being used, portions of its data will be temporarily and transparently stored on the special fast LV.

The two kinds of caching are:

? A read and write hot-spot cache, using the dm-cache kernel module. This cache tracks access patterns and adjusts its content deliber? ately so that commonly used parts of the main LV are likely to be found on the fast storage. LVM refers to this using the LV type cache.

? A write cache, using the dm-writecache kernel module. This cache can

be used with SSD or PMEM devices to speed up all writes to the main LV. Data read from the main LV is not stored in the cache, only newly written data. LVM refers to this using the LV type writecache.

USAGE

1. Identify main LV that needs caching

   The main LV may already exist, and is located on larger, slower de? vices. A main LV would be created with a command like:

   # lvcreate -n main -L Size vg /dev/slow_hhd

2. Identify fast LV to use as the cache

   A fast LV is created using one or more fast devices, like an SSD. This special LV will be used to hold the cache:

   # lvcreate -n fast -L Size vg /dev/fast_ssd

   # lvs -a

   LV Attr Type Devices

   fast -wi------- linear /dev/fast_ssd

   main -wi------- linear /dev/slow_hhd

3. Start caching the main LV

   To start caching the main LV, convert the main LV to the desired caching type, and specify the fast LV to use as the cache:

   using dm-cache (with cachepool):

   # lvconvert --type cache --cachepool fast vg/main

   using dm-cache (with cachevol):

   # lvconvert --type cache --cachevol fast vg/main

   using dm-writecache (with cachevol):

   # lvconvert --type writecache --cachevol fast vg/main

   For more alteratives see:

   dm-cache command shortcut

   dm-cache with separate data and metadata LVs

4. Display LVs

   Once the fast LV has been attached to the main LV, lvm reports the main LV type as either cache or writecache depending on the type used. While attached, the fast LV is hidden, and renamed with a _cvol or _cpool suffix. It is displayed by lvs -a. The _corig or _wcorig LV

represents the original LV without the cache.

using dm-cache (with cachepool):

```
# lvs -ao+devices
  LV                Pool         Type       Devices
  main              [fast_cpool] cache      main_corig(0)
  [fast_cpool]                   cache-pool fast_pool_cdata(0)
  [fast_cpool_cdata]             linear     /dev/fast_ssd
  [fast_cpool_cmeta]             linear     /dev/fast_ssd
  [main_corig]                   linear     /dev/slow_hhd
```

using dm-cache (with cachevol):

```
# lvs -ao+devices
  LV          Pool        Type   Devices
  main        [fast_cvol] cache  main_corig(0)
  [fast_cvol]             linear /dev/fast_ssd
  [main_corig]            linear /dev/slow_hhd
```

using dm-writecache (with cachevol):

```
# lvs -ao+devices
  LV          Pool        Type       Devices
  main        [fast_cvol] writecache main_wcorig(0)
  [fast_cvol]             linear     /dev/fast_ssd
  [main_wcorig]           linear     /dev/slow_hhd
```

5. Use the main LV

   Use the LV until the cache is no longer wanted, or needs to be changed.

6. Stop caching

   To stop caching the main LV and also remove unneeded cache pool, use the --uncache:

```
# lvconvert --uncache vg/main
# lvs -a
  LV   VG Attr       Type   Devices
  main vg -wi------- linear /dev/slow_hhd
```

   To stop caching the main LV, separate the fast LV from the main LV. This changes the type of the main LV back to what it was before the cache was attached.

lvconvert --splitcache vg/main

lvs -a

LV VG Attr Type Devices

fast vg -wi------- linear /dev/fast_ssd

main vg -wi------- linear /dev/slow_hhd

7. Create a new LV with caching

A new LV can be created with caching attached at the time of creation using the following command:

lvcreate --type cache|writecache -n Name -L Size --cachedevice /dev/fast_ssd vg /dev/slow_hhd

The main LV is created with the specified Name and Size from the slow_hhd. A hidden fast LV is created on the fast_ssd and is then at? tached to the new main LV. If the fast_ssd is unused, the entire disk will be used as the cache unless the --cachesize option is used to specify a size for the fast LV. The --cachedevice option can be re? peated to use multiple disks for the fast LV.

OPTIONS

option args

--cachepool CachePoolLV|LV

Pass this option a cachepool LV or a standard LV. When using a cache pool, lvm places cache data and cache metadata on different LVs. The two LVs together are called a cache pool. This has a bit better per? formance for dm-cache and permits specific placement and segment type selection for data and metadata volumes. A cache pool is represented as a special type of LV that cannot be used directly. If a standard LV is passed with this option, lvm will first convert it to a cache pool by combining it with another LV to use for metadata. This option can be used with dm-cache.

--cachevol LV

Pass this option a fast LV that should be used to hold the cache. With a cachevol, cache data and metadata are stored in different parts of the same fast LV. This option can be used with dm-writecache or dm-cache.

--cachedevice PV

This option can be used in place of --cachevol, in which case a cachevol LV will be created using the specified device. This option can be repeated to create a cachevol using multiple devices, or a tag name can be specified in which case the cachevol will be created using any of the devices with the given tag. If a named cache device is un? used, the entire device will be used to create the cachevol. To create a cachevol of a specific size from the cache devices, include the --cachesize option.

dm-cache block size

A cache pool will have a logical block size of 4096 bytes if it is cre? ated on a device with a logical block size of 4096 bytes.

If a main LV has logical block size 512 (with an existing xfs file sys? tem using that size), then it cannot use a cache pool with a 4096 logi? cal block size. If the cache pool is attached, the main LV will likely fail to mount.

To avoid this problem, use a mkfs option to specify a 4096 block size for the file system, or attach the cache pool before running mkfs.

dm-writecache block size

The dm-writecache block size can be 4096 bytes (the default), or 512 bytes. The default 4096 has better performance and should be used ex? cept when 512 is necessary for compatibility. The dm-writecache block size is specified with --cachesettings block_size=4096|512 when caching is started.

When a file system like xfs already exists on the main LV prior to caching, and the file system is using a block size of 512, then the writecache block size should be set to 512. (The file system will likely fail to mount if writecache block size of 4096 is used in this case.)

Check the xfs sector size while the fs is mounted:

xfs_info /dev/vg/main

Look for sectsz=512 or sectsz=4096

The writecache block size should be chosen to match the xfs sectsz

value.

It is also possible to specify a sector size of 4096 to mkfs.xfs when creating the file system. In this case the writecache block size of 4096 can be used.

The writecache block size is displayed by the command:

lvs -o writecacheblocksize VG/LV

dm-writecache memory usage

The amount of main system memory used by dm-writecache can be a factor when selecting the writecache cachevol size and the writecache block size.

? writecache block size 4096: each 100 GiB of writecache cachevol uses slighly over 2 GiB of system memory.

? writecache block size 512: each 100 GiB of writecache cachevol uses a little over 16 GiB of system memory.

dm-writecache settings

To specify dm-writecache tunable settings on the command line, use:

--cachesettings 'option=N' or

--cachesettings 'option1=N option2=N ...'

For example, --cachesettings 'high_watermark=90 writeback_jobs=4'.

To include settings when caching is started, run:

```
# lvconvert --type writecache --cachevol fast \
    --cachesettings 'option=N' vg/main
```

To change settings for an existing writecache, run:

```
# lvchange --cachesettings 'option=N' vg/main
```

To clear all settings that have been applied, run:

```
# lvchange --cachesettings '' vg/main
```

To view the settings that are applied to a writecache LV, run:

```
# lvs -o cachesettings vg/main
```

Tunable settings are:

high_watermark = <percent>

Start writeback when the writecache usage reaches this percent (0-100).

low_watermark = <percent>

Stop writeback when the writecache usage reaches this percent (0-100).

writeback_jobs = <count>

Limit the number of blocks that are in flight during writeback. Setting this value reduces writeback throughput, but it may improve latency of read requests.

autocommit_blocks = <count>

When the application writes this amount of blocks without issuing the FLUSH request, the blocks are automatically commited.

autocommit_time = <milliseconds>

The data is automatically commited if this time passes and no FLUSH request is received.

fua = 0|1

Use the FUA flag when writing data from persistent memory back to the underlying device. Applicable only to persistent memory.

nofua = 0|1

Don't use the FUA flag when writing back data and send the FLUSH request afterwards. Some underlying devices perform better with fua, some with nofua. Testing is necessary to determine which. Applicable only to persistent memory.

cleaner = 0|1

Setting cleaner=1 enables the writecache cleaner mode in which data is gradually flushed from the cache. If this is done prior to detaching the writecache, then the splitcache command will have little or no flushing to perform. If not done beforehand, the splitcache command enables the cleaner mode and waits for flushing to complete before detaching the writecache. Adding cleaner=0 to the splitcache command will skip the cleaner mode, and any required flushing is performed in device suspend.

dm-writecache using metadata profiles

In addition to specifying writecache settings on the command line, they can also be set in lvm.conf, or in a profile file, using the allocation/cache_settings/writecache config structure shown below.

It's possible to prepare a number of different profile files in the /etc/lvm/profile directory and specify the file name of the profile when starting writecache.

Example

```
# cat <<EOF > /etc/lvm/profile/cache_writecache.profile
allocation {
        cache_settings {
                writecache {
                        high_watermark=60
                        writeback_jobs=1024
                }
        }
}
EOF
# lvcreate -an -L10G --name fast vg /dev/fast_ssd
# lvcreate --type writecache -L10G --name main  --cachevol fast \
   --metadataprofile cache_writecache vg /dev/slow_hdd
```

dm-cache with separate data and metadata LVs

Preferred way of using dm-cache is to place the cache metadata and cache data on separate LVs. To do this, a "cache pool" is created, which is a special LV that references two sub LVs, one for data and one for metadata.

To create a cache pool of given data size and let lvm2 calculate appro? priate metadata size:

```
# lvcreate --type cache-pool -L DataSize -n fast vg /dev/fast_ssd1
```

To create a cache pool from separate LV and let lvm2 calculate appro? priate cache metadata size:

```
# lvcreate -n fast -L DataSize vg /dev/fast_ssd1
# lvconvert --type cache-pool vg/fast /dev/fast_ssd1
```

To create a cache pool from two separate LVs:

```
# lvcreate -n fast -L DataSize vg /dev/fast_ssd1
# lvcreate -n fastmeta -L MetadataSize vg /dev/fast_ssd2
# lvconvert --type cache-pool --poolmetadata fastmeta vg/fast
```

Then use the cache pool LV to start caching the main LV:

lvconvert --type cache --cachepool fast vg/main

A variation of the same procedure automatically creates a cache pool when caching is started. To do this, use a standard LV as the --cachepool (this will hold cache data), and use another standard LV as the --poolmetadata (this will hold cache metadata). LVM will create a cache pool LV from the two specified LVs, and use the cache pool to start caching the main LV.

lvcreate -n fast -L DataSize vg /dev/fast_ssd1

lvcreate -n fastmeta -L MetadataSize vg /dev/fast_ssd2

lvconvert --type cache --cachepool fast \
    --poolmetadata fastmeta vg/main

dm-cache cache modes

The default dm-cache cache mode is "writethrough". Writethrough en? sures that any data written will be stored both in the cache and on the origin LV. The loss of a device associated with the cache in this case would not mean the loss of any data.

A second cache mode is "writeback". Writeback delays writing data blocks from the cache back to the origin LV. This mode will increase performance, but the loss of a cache device can result in lost data.

With the --cachemode option, the cache mode can be set when caching is started, or changed on an LV that is already cached. The current cache mode can be displayed with the cache_mode reporting option:

lvs -o+cache_mode VG/LV

lvm.conf(5) allocation/cache_mode

defines the default cache mode.

lvconvert --type cache --cachemode writethrough \
    --cachepool fast vg/main

lvconvert --type cache --cachemode writethrough \
    --cachevol fast vg/main

dm-cache chunk size

The size of data blocks managed by dm-cache can be specified with the --chunksize option when caching is started. The default unit is KiB.

The value must be a multiple of 32 KiB between 32 KiB and 1 GiB. Cache chunks bigger then 512KiB shall be only used when necessary.

Using a chunk size that is too large can result in wasteful use of the cache, in which small reads and writes cause large sections of an LV to be stored in the cache. It can also require increasing migration threshold which defaults to 2048 sectors (1 MiB). Lvm2 ensures migra? tion threshold is at least 8 chunks in size. This may in some cases re? sult in very high bandwidth load of transfering data between the cache LV and its cache origin LV. However, choosing a chunk size that is too small can result in more overhead trying to manage the numerous chunks that become mapped into the cache. Overhead can include both excessive CPU time searching for chunks, and excessive memory tracking chunks.

Command to display the chunk size:

lvs -o+chunksize VG/LV

lvm.conf(5) allocation/cache_pool_chunk_size

controls the default chunk size.

The default value is shown by:

lvmconfig --type default allocation/cache_pool_chunk_size

Checking migration threshold (in sectors) of running cached LV:

lvs -o+kernel_cache_settings VG/LV

dm-cache migration threshold

Migrating data between the origin and cache LV uses bandwidth. The user can set a throttle to prevent more than a certain amount of migra? tion occurring at any one time. Currently dm-cache is not taking any account of normal io traffic going to the devices.

User can set migration threshold via cache policy settings as "migra? tion_threshold=<#sectors>" to set the maximum number of sectors being migrated, the default being 2048 sectors (1 MiB).

Command to set migration threshold to 2 MiB (4096 sectors):

lvcreate --cachepolicy 'migration_threshold=4096' VG/LV

Command to display the migration threshold:

lvs -o+kernel_cache_settings,cache_settings VG/LV

lvs -o+chunksize VG/LV

dm-cache cache policy

The dm-cache subsystem has additional per-LV parameters: the cache pol?
icy to use, and possibly tunable parameters for the cache policy.
Three policies are currently available: "smq" is the default policy,
"mq" is an older implementation, and "cleaner" is used to force the
cache to write back (flush) all cached writes to the origin LV.
The older "mq" policy has a number of tunable parameters. The defaults
are chosen to be suitable for the majority of systems, but in special
circumstances, changing the settings can improve performance.
With the --cachepolicy and --cachesettings options, the cache policy
and settings can be set when caching is started, or changed on an ex?
isting cached LV (both options can be used together). The current
cache policy and settings can be displayed with the cache_policy and
cache_settings reporting options:
lvs -o+cache_policy,cache_settings VG/LV
Change the cache policy and settings of an existing LV.
lvchange --cachepolicy mq --cachesettings \
'migration_threshold=2048 random_threshold=4' vg/main
lvm.conf(5) allocation/cache_policy
defines the default cache policy.
lvm.conf(5) allocation/cache_settings
defines the default cache settings.

dm-cache using metadata profiles

Cache pools allows to set a variety of options. Lots of these settings
can be specified in lvm.conf or profile settings. You can prepare a
number of different profiles in the /etc/lvm/profile directory and just
specify the metadata profile file name when caching LV or creating
cache-pool. Check the output of lvmconfig --type default --withcom?
ments for a detailed description of all individual cache settings.
Example
# cat <<EOF > /etc/lvm/profile/cache_big_chunk.profile
allocation {
cache_pool_metadata_require_separate_pvs=0

```
        cache_pool_chunk_size=512
        cache_metadata_format=2
        cache_mode="writethrough"
        cache_policy="smq"
        cache_settings {
            smq {
                migration_threshold=8192
                random_threshold=4096
            }
        }
    }
    EOF
    # lvcreate --cache -L10G --metadataprofile cache_big_chunk vg/main \
        /dev/fast_ssd
    # lvcreate --cache -L10G vg/main --config \
        'allocation/cache_pool_chunk_size=512' /dev/fast_ssd
```

dm-cache spare metadata LV

See lvmthin(7) for a description of the "pool metadata spare" LV. The same concept is used for cache pools.

dm-cache metadata formats

There are two disk formats for dm-cache metadata. The metadata format can be specified with --cachemetadataformat when caching is started, and cannot be changed. Format 2 has better performance; it is more compact, and stores dirty bits in a separate btree, which improves the speed of shutting down the cache. With auto, lvm selects the best op? tion provided by the current dm-cache kernel module.

RAID1 cache device

RAID1 can be used to create the fast LV holding the cache so that it can tolerate a device failure. (When using dm-cache with separate data and metadata LVs, each of the sub-LVs can use RAID1.)

```
    # lvcreate -n main -L Size vg /dev/slow
    # lvcreate --type raid1 -m 1 -n fast -L Size vg /dev/ssd1 /dev/ssd2
    # lvconvert --type cache --cachevol fast vg/main
```

dm-cache command shortcut

A single command can be used to cache main LV with automatic creation of a cache-pool:

lvcreate --cache --size CacheDataSize VG/LV [FastPVs]

or the longer variant

lvcreate --type cache --size CacheDataSize \
--name NameCachePool VG/LV [FastPVs]

In this command, the specified LV already exists, and is the main LV to be cached. The command creates a new cache pool with size and given name or the name is automatically selected from a sequence lvolX_cpool, using the optionally specified fast PV(s) (typically an ssd). Then it attaches the new cache pool to the existing main LV to begin caching. (Note: ensure that the specified main LV is a standard LV. If a cache pool LV is mistakenly specified, then the command does something dif? ferent.)

(Note: the type option is interpreted differently by this command than by normal lvcreate commands in which --type specifies the type of the newly created LV. In this case, an LV with type cache-pool is being created, and the existing main LV is being converted to type cache.)

SEE ALSO

lvm.conf(5), lvchange(8), lvcreate(8), lvdisplay(8), lvextend(8), lvremove(8), lvrename(8), lvresize(8), lvs(8), vgchange(8), vgmerge(8), vgreduce(8), vgsplit(8), cache_check(8), cache_dump(8), cache_repair(8)