dm-zoned

The dm-zoned device mapper target exposes a zoned block device (ZBC and ZAC compliant devices) as a regular block device without any write pattern constraints. In effect, it implements a drive-managed zoned block device which hides from the user (a file system or an application doing raw block device accesses) the sequential write constraints of host-managed zoned block devices and can mitigate the potential device-side performance degradation due to excessive random writes on host-aware zoned block devices.

For a more detailed description of the zoned block device models and their constraints see (for SCSI devices):

https://www.t10.org/drafts.htm#ZBC_Family

and (for ATA devices):

http://www.t13.org/Documents/UploadedDocuments/docs2015/di537r05-Zoned_Device_ATA_Command_Set_ZAC.pdf

The dm-zoned implementation is simple and minimizes system overhead (CPU and memory usage as well as storage capacity loss). For a 10TB host-managed disk with 256 MB zones, dm-zoned memory usage per disk instance is at most 4.5 MB and as little as 5 zones will be used internally for storing metadata and performing reclaim operations.

dm-zoned target devices are formatted and checked using the dmzadm utility available at:

https://github.com/hgst/dm-zoned-tools

Algorithm

dm-zoned implements an on-disk buffering scheme to handle non-sequential write accesses to the sequential zones of a zoned block device. Conventional zones are used for caching as well as for storing internal metadata. It can also use a regular block device together with the zoned block device; in that case the regular block device will be split logically in zones with the same size as the zoned block device. These zones will be placed in front of the zones from the zoned block device and will be handled just like conventional zones.

The zones of the device(s) are separated into 2 types:

1) Metadata zones: these are conventional zones used to store metadata. Metadata zones are not reported as useable capacity to the user.

2) Data zones: all remaining zones, the vast majority of which will be sequential zones used exclusively to store user data. The conventional zones of the device may be used also for buffering user random writes. Data in these zones may be directly mapped to the conventional zone, but later moved to a sequential zone so that the conventional zone can be reused for buffering incoming random writes.

dm-zoned exposes a logical device with a sector size of 4096 bytes, irrespective of the physical sector size of the backend zoned block device being used. This allows reducing the amount of metadata needed to manage valid blocks (blocks written).

The on-disk metadata format is as follows:

1) The first block of the first conventional zone found contains the super block which describes the on disk amount and position of metadata blocks.

2) Following the super block, a set of blocks is used to describe the mapping of the logical device blocks. The mapping is done per chunk of blocks, with the chunk size equal to the zoned block device size. The mapping table is indexed by chunk number and each mapping entry indicates the zone number of the device storing the chunk of data. Each mapping entry may also indicate if the zone number of a conventional zone used to buffer random modification to the data zone.

3) A set of blocks used to store bitmaps indicating the validity of blocks in the data zones follows the mapping table. A valid block is defined as a block that was written and not discarded. For a buffered data chunk, a block is always valid only in the data zone mapping the chunk or in the buffer zone of the chunk.

For a logical chunk mapped to a conventional zone, all write operations are processed by directly writing to the zone. If the mapping zone is a sequential zone, the write operation is processed directly only if the write offset within the logical chunk is equal to the write pointer offset within of the sequential data zone (i.e. the write operation is aligned on the zone write pointer). Otherwise, write operations are processed indirectly using a buffer zone. In that case, an unused conventional zone is allocated and assigned to the chunk being accessed. Writing a block to the buffer zone of a chunk will automatically invalidate the same block in the sequential zone mapping the chunk. If all blocks of the sequential zone become invalid, the zone is freed and the chunk buffer zone becomes the primary zone mapping the chunk, resulting in native random write performance similar to a regular block device.

Read operations are processed according to the block validity information provided by the bitmaps. Valid blocks are read either from the sequential zone mapping a chunk, or if the chunk is buffered, from the buffer zone assigned. If the accessed chunk has no mapping, or the accessed blocks are invalid, the read buffer is zeroed and the read operation terminated.

After some time, the limited number of conventional zones available may be exhausted (all used to map chunks or buffer sequential zones) and unaligned writes to unbuffered chunks become impossible. To avoid this situation, a reclaim process regularly scans used conventional zones and tries to reclaim the least recently used zones by copying the valid blocks of the buffer zone to a free sequential zone. Once the copy completes, the chunk mapping is updated to point to the sequential zone and the buffer zone freed for reuse.

Metadata Protection

To protect metadata against corruption in case of sudden power loss or system crash, 2 sets of metadata zones are used. One set, the primary set, is used as the main metadata region, while the secondary set is used as a staging area. Modified metadata is first written to the secondary set and validated by updating the super block in the secondary set, a generation counter is used to indicate that this set contains the newest metadata. Once this operation completes, in place of metadata block updates can be done in the primary metadata set. This ensures that one of the set is always consistent (all modifications committed or none at all). Flush operations are used as a commit point. Upon reception of a flush request, metadata modification activity is temporarily blocked (for both incoming BIO processing and reclaim process) and all dirty metadata blocks are staged and updated. Normal operation is then resumed. Flushing metadata thus only temporarily delays write and discard requests. Read requests can be processed concurrently while metadata flush is being executed.

If a regular device is used in conjunction with the zoned block device, a third set of metadata (without the zone bitmaps) is written to the start of the zoned block device. This metadata has a generation counter of '0' and will never be updated during normal operation; it just serves for identification purposes. The first and second copy of the metadata are located at the start of the regular block device.

Usage

A zoned block device must first be formatted using the dmzadm tool. This will analyze the device zone configuration, determine where to place the metadata sets on the device and initialize the metadata sets.

Ex:

```
dmzadm --format /dev/sdxx
```

If two drives are to be used, both devices must be specified, with the regular block device as the first device.

Ex:

```
dmzadm --format /dev/sdxx /dev/sdyy
```

Formatted device(s) can be started with the dmzadm utility, too.:

Ex:

```
dmzadm --start /dev/sdxx /dev/sdyy
```

Information about the internal layout and current usage of the zones can be obtained with the 'status' callback from dmsetup:

Ex:

```
dmsetup status /dev/dm-X
```

will return a line

0 <size> zoned <nr_zones> zones <nr_unmap_rnd>/<nr_rnd> random <nr_unmap_seq>/<nr_seq> sequential

where <nr_zones> is the total number of zones, <nr_unmap_rnd> is the number of unmapped (ie free) random zones, <nr_rnd> the total number of zones, <nr_unmap_seq> the number of unmapped sequential zones, and <nr_seq> the total number of sequential zones.

Normally the reclaim process will be started once there are less than 50 percent free random zones. In order to start the reclaim process manually even before reaching this threshold the 'dmsetup message' function can be used:

Ex:

```
dmsetup message /dev/dm-X 0 reclaim
```

will start the reclaim process and random zones will be moved to sequential zones.