dm-clone

Introduction

dm-clone is a device mapper target which produces a one-to-one copy of an existing, read-only source device into a writable destination device: It presents a virtual block device which makes all data appear immediately, and redirects reads and writes accordingly.

The main use case of dm-clone is to clone a potentially remote, high-latency, read-only, archival-type block device into a writable, fast, primary-type device for fast, low-latency I/O. The cloned device is visible/mountable immediately and the copy of the source device to the destination device happens in the background, in parallel with user I/O.

For example, one could restore an application backup from a read-only copy, accessible through a network storage protocol (NBD, Fibre Channel, iSCSI, AoE, etc.), into a local SSD or NVMe device, and start using the device immediately, without waiting for the restore to complete.

When the cloning completes, the dm-clone table can be removed altogether and be replaced, e.g., by a linear table, mapping directly to the destination device.

The dm-clone target reuses the metadata library used by the thin-provisioning target.

Glossary

Hydration
: The process of filling a region of the destination device with data from the same region of the source device, i.e., copying the region from the source to the destination device.

Once a region gets hydrated we redirect all I/O regarding it to the destination device.

Design

Sub-devices

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

1. A source device - the read-only device that gets cloned and source of the hydration.
2. A destination device - the destination of the hydration, which will become a clone of the source device.
3. A small metadata device - it records which regions are already valid in the destination device, i.e., which regions have already been hydrated, or have been written to directly, via user I/O.

The size of the destination device must be at least equal to the size of the source device.

Regions

dm-clone divides the source and destination devices in fixed sized regions. Regions are the unit of hydration, i.e., the minimum amount of data copied from the source to the destination device.

The region size is configurable when you first create the dm-clone device. The recommended region size is the same as the file system block size, which usually is 4KB. The region size must be between 8 sectors (4KB) and 2097152 sectors (1GB) and a power of two.

Reads and writes from/to hydrated regions are serviced from the destination device.

A read to a not yet hydrated region is serviced directly from the source device.

A write to a not yet hydrated region will be delayed until the corresponding region has been hydrated and the hydration of the region starts immediately.

Note that a write request with size equal to region size will skip copying of the corresponding region from the source device and overwrite the region of the destination device directly.

Discards

dm-clone interprets a discard request to a range that hasn't been hydrated yet as a hint to skip hydration of the regions covered by the request, i.e., it skips copying the region's data from the source to the destination device, and only updates its metadata.

If the destination device supports discards, then by default dm-clone will pass down discard requests to it.

Background Hydration

dm-clone copies continuously from the source to the destination device, until all of the device has been copied.

Copying data from the source to the destination device uses bandwidth. The user can set a throttle to prevent more than a certain amount of copying occurring at any one time. Moreover, dm-clone takes into account user I/O traffic going to the devices and pauses the background hydration when there is I/O in-flight.

A message *hydration_threshold <#regions>* can be used to set the maximum number of regions being copied, the default being 1 region.

dm-clone employs dm-kcopyd for copying portions of the source device to the destination device. By default, we issue copy requests of size equal to the region size. A message *hydration_batch_size <#regions>* can be used to tune the size of these copy requests. Increasing the hydration batch size results in dm-clone trying to batch together contiguous regions, so we copy the data in batches of this many regions.

When the hydration of the destination device finishes, a dm event will be sent to user space.

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 dm-clone device 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.

Target Interface

Constructor

```
clone <metadata dev> <destination dev> <source dev> <region size>
      [<#feature args> [<feature arg>]* [<#core args> [<core arg>]*]]
```

metadata dev	Fast device holding the persistent metadata
destination dev	The destination device, where the source will be cloned
source dev	Read only device containing the data that gets cloned
region size	The size of a region in sectors
#feature args	Number of feature arguments passed
feature args	no_hydration or no_discard_passdown
#core args	An even number of arguments corresponding to key/value pairs passed to dm-clone
core args	Key/value pairs passed to dm-clone, e.g. *hydration_threshold 256*

Optional feature arguments are:

no_hydration	Create a dm-clone instance with background hydration disabled
no_discard_passdown	Disable passing down discards to the destination device

Optional core arguments are:

hydration_threshold <#regions>	Maximum number of regions being copied from the source to the destination device at any one time, during background hydration.
hydration_batch_size <#regions>	During background hydration, try to batch together contiguous regions, so we copy data from the source to the destination device in batches of this many regions.

Status

```
<metadata block size> <#used metadata blocks>/<#total metadata blocks>
<region size> <#hydrated regions>/<#total regions> <#hydrating regions>
<#feature args> <feature args>* <#core args> <core args>*
<clone 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
region size	Configurable region size for the device in sectors
#hydrated regions	Number of regions that have finished hydrating
#total regions	Total number of regions to hydrate
#hydrating regions	Number of regions currently hydrating
#feature args	Number of feature arguments to follow
feature args	Feature arguments, e.g. *no_hydration*
#core args	Even number of core arguments to follow

core args	Key/value pairs for tuning the core, e.g. *hydration_threshold 256*
clone 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'. If the metadata mode changes, a dm event will be sent to user space.

Messages

disable_hydration
: Disable the background hydration of the destination device.

enable_hydration
: Enable the background hydration of the destination device.

hydration_threshold <#regions>
: Set background hydration threshold.

hydration_batch_size <#regions>
: Set background hydration batch size.

Examples

Clone a device containing a file system

1. Create the dm-clone device.

```
dmsetup create clone --table "0 1048576000 clone $metadata_dev $dest_dev \
  $source_dev 8 1 no_hydration"
```

2. Mount the device and trim the file system. dm-clone interprets the discards sent by the file system and it will not hydrate the unused space.

```
mount /dev/mapper/clone /mnt/cloned-fs
fstrim /mnt/cloned-fs
```

3. Enable background hydration of the destination device.

```
dmsetup message clone 0 enable_hydration
```

4. When the hydration finishes, we can replace the dm-clone table with a linear table.

```
dmsetup suspend clone
dmsetup load clone --table "0 1048576000 linear $dest_dev 0"
dmsetup resume clone
```

   The metadata device is no longer needed and can be safely discarded or reused for other purposes.

Known issues

1. We redirect reads, to not-yet-hydrated regions, to the source device. If reading the source device has high latency and the user repeatedly reads from the same regions, this behaviour could degrade performance. We should use these reads as hints to hydrate the relevant regions sooner. Currently, we rely on the page cache to cache these regions, so we hopefully don't end up reading them multiple times from the source device.
2. Release in-core resources, i.e., the bitmaps tracking which regions are hydrated, after the hydration has finished.
3. During background hydration, if we fail to read the source or write to the destination device, we print an error message, but the hydration process continues indefinitely, until it succeeds. We should stop the background hydration after a number of failures and emit a dm event for user space to notice.

Why not...?

We explored the following alternatives before implementing dm-clone:

1. Use dm-cache with cache size equal to the source device and implement a new cloning policy:
   - The resulting cache device is not a one-to-one mirror of the source device and thus we cannot remove the cache device once cloning completes.
   - dm-cache writes to the source device, which violates our requirement that the source device must be treated as read-only.
   - Caching is semantically different from cloning.
2. Use dm-snapshot with a COW device equal to the source device:
   - dm-snapshot stores its metadata in the COW device, so the resulting device is not a one-to-one mirror of the source device.

- No background copying mechanism.
- dm-snapshot needs to commit its metadata whenever a pending exception completes, to ensure snapshot consistency. In the case of cloning, we don't need to be so strict and can rely on committing metadata every time a FLUSH or FUA bio is written, or periodically, like dm-thin and dm-cache do. This improves the performance significantly.

3. Use dm-mirror: The mirror target has a background copying/mirroring mechanism, but it writes to all mirrors, thus violating our requirement that the source device must be treated as read-only.
4. Use dm-thin's external snapshot functionality. This approach is the most promising among all alternatives, as the thinly-provisioned volume is a one-to-one mirror of the source device and handles reads and writes to un-provisioned/not-yet-cloned areas the same way as dm-clone does.

   Still:

   - There is no background copying mechanism, though one could be implemented.
   - Most importantly, we want to support arbitrary block devices as the destination of the cloning process and not restrict ourselves to thinly-provisioned volumes. Thin-provisioning has an inherent metadata overhead, for maintaining the thin volume mappings, which significantly degrades performance.

   Moreover, cloning a device shouldn't force the use of thin-provisioning. On the other hand, if we wish to use thin provisioning, we can just use a thin LV as dm-clone's destination device.