NAME

lvmvdo — EXPERIMENTAL LVM Virtual Data Optimizer support

DESCRIPTION

VDO (which includes kvdo and vdo) is software that provides inline block-level deduplication, compression, and thin provisioning capabilities for primary storage.

Deduplication is a technique for reducing the consumption of storage resources by eliminating multiple copies of duplicate blocks. Compression takes the individual unique blocks and shrinks them with coding algorithms; these reduced blocks are then efficiently packed together into physical blocks. Thin provisioning manages the mapping from LBAs presented by VDO to where the data has actually been stored, and also eliminates any blocks of all zeroes.

With deduplication, instead of writing the same data more than once each duplicate block is detected and recorded as a reference to the original block. VDO maintains a mapping from logical block addresses (used by the storage layer above VDO) to physical block addresses (used by the storage layer under VDO). After deduplication, multiple logical block addresses may be mapped to the same physical block address; these are called shared blocks and are reference-counted by the software.

With VDO's compression, multiple blocks (or shared blocks) are compressed with the fast LZ4 algorithm, and binned together where possible so that multiple compressed blocks fit within a 4 KB block on the underlying storage. Mapping from LBA is to a physical block address and index within it for the desired compressed data. All compressed blocks are individually reference counted for correctness.

Block sharing and block compression are invisible to applications using the storage, which read and write blocks as they would if VDO were not present. When a shared block is overwritten, a new physical block is allocated for storing the new block data to ensure that other logical block addresses that are mapped to the shared physical block are not modified.

For usage of VDO with **lvm**(8) standard VDO userspace tools **vdoformat**(8) and currently non-standard kernel VDO module "*kvdo*" needs to be installed on the system.

The "*kvdo*" module implements fine-grained storage virtualization, thin provisioning, block sharing, and compression; the "*uds*" module provides memory-efficient duplicate identification. The userspace tools include **vdostats**(8) for extracting statistics from those volumes.

VDO Terms

VDODataLV
: VDO data LV
large hidden LV with suffix _vdata created in a VG.
used by VDO target to store all data and metadata blocks.

VDOPoolLV
: VDO pool LV
maintains virtual for LV(s) stored in attached VDO data LV and it has same size.
contains VDOLV(s) (currently supports only a single VDOLV).

VDOLV
: VDO LV
created from VDOPoolLV
appears blank after creation

VDO Usage

The primary methods for using VDO with lvm2:

1. Create VDOPoolLV with VDOLV

Create an VDOPoolLV that will holds VDO data together with virtual size VDOLV, that user can use. When the virtual size is not specified, then such LV is created with maximum size that always fits into data volume even if there cannot happen any deduplication and compression (i.e. it can hold uncompressible content of /dev/urandom). When the name of VDOPoolLV is not specified, it tales name from sequence of vpool0, vpool1 ...

Note: As the performance of TRIM/Discard operation is slow for large volumes of VDO type, please try to avoid sending discard requests unless necessary as it may take considerable amount of time to finish discard operation.

lvcreate −−type vdo −n VDOLV −L DataSize −V LargeVirtualSize VG/VDOPoolLV
lvcreate −−vdo −L DataSize VG

Example
lvcreate −−type vdo −n vdo0 −L 10G −V 100G vg/vdopool0
mkfs.ext4 −E nodiscard /dev/vg/vdo0

2. Create VDOPoolLV and convert existing LV into VDODataLV

Convert an already created/existing LV into a volume that can hold VDO data and metadata (a volume reference by VDOPoolLV). User will be prompted to confirm such conversion as it is **IRREVERSIBLY DESTROYING** content of such volume, as it's being immediately formatted by **vdoformat**(8) as VDO pool data volume. User can specify virtual size of associated VDOLV with this VDOPoolLV. When the virtual size is not specified, it will set to the maximum size that can keep 100% uncompressible data there.

lvconvert −−type vdo−pool −n VDOLV −V VirtualSize VG/VDOPoolLV
lvconvert −−vdopool VG/VDOPoolLV

Example
lvconvert −−type vdo−pool −n vdo0 −V10G vg/existinglv

3. Change default setting used for creating VDOPoolLV

VDO allows to set large variety of option. Lots of these setting can be specified by lvm.conf or profile settings. User can prepare number of different profiles and just specify profile file name. Check output of **lvmconfig −−type full** for detailed description of all individual vdo settings.

Example
```
# cat <<EOF > vdo.profile
allocation {
        vdo_use_compression=1
        vdo_use_deduplication=1
        vdo_use_metadata_hints=1
        vdo_minimum_io_size=4096
        vdo_block_map_cache_size_mb=128
        vdo_block_map_period=16380
        vdo_check_point_frequency=0
        vdo_use_sparse_index=0
        vdo_index_memory_size_mb=256
        vdo_slab_size_mb=2048
        vdo_ack_threads=1
```

```
        vdo_bio_threads=1
        vdo_bio_rotation=64
        vdo_cpu_threads=2
        vdo_hash_zone_threads=1
        vdo_logical_threads=1
        vdo_physical_threads=1
        vdo_write_policy="auto"
        vdo_max_discard=1
}
EOF

# lvcreate --vdo -L10G --metadataprofile vdo.profile vg/vdopool0
# lvcreate --vdo -L10G --config 'allocation/vdo_cpu_threads=4' vg/vdopool1
```

4. Change compression and deduplication of VDOPoolLV

Disable or enable compression and deduplication for VDO pool LV (the volume that maintains all VDO LV(s) associated with it).

lvchange --compression [y|n] --deduplication [y|n] VG/VDOPoolLV

Example

```
# lvchange --compression n  vg/vdpool0
# lvchange --deduplication y vg/vdpool1
```

4. Checking usage of VDOPoolLV

To quickly check how much data of VDOPoolLV are already consumed use **lvs**(8). Field Data% will report how much data occupies content of virtual data for VDOLV and how much space is already consumed with all the data and metadata blocks in VDOPoolLV. For a detailed description use **vdostats**(8) command.

Note: **vdostats**(8) currently understands only /dev/mapper device names.

Example

```
# lvcreate --type vdo -L10G -V20G -n vdo0 vg/vdopool0
# mkfs.ext4 -E nodiscard /dev/vg/vdo0
# lvs -a vg

  LV              VG Attr       LSize  Pool     Origin Data%
  vdo0            vg vwi-a-v--- 20.00g vdopool0        0.01
  vdopool0        vg dwi-ao---- 10.00g                30.16
  [vdopool0_vdata] vg Dwi-ao---- 10.00g

# vdostats --all /dev/mapper/vg-vdopool0
/dev/mapper/vg-vdopool0 :
  version                         : 30
  release version                 : 133524
  data blocks used                : 79
  ...
```

4. Extending VDOPoolLV size

Adding more space to hold VDO data and metadata can be made via extension of VDODataLV with commands **lvresize**(8), **lvextend**(8).

Note: Size of VDOPoolLV cannot be reduced.

lvextend −L+AddingSize VG/VDOPoolLV

Example
lvextend −L+50G vg/vdopool0
lvresize −L300G vg/vdopool1

4. Extending or reducing VDOLV size

VDO LV can be extended or reduced as standard LV with commands **lvresize**(8), **lvextend**(8), **lvreduce**(8).

Note: Reduction needs to process TRIM for reduced disk area to unmap used data blocks from VDOPoolLV and it may take a long time.

lvextend −L+AddingSize VG/VDOLV
lvreduce −L-ReducingSize VG/VDOLV

Example
lvextend −L+50G vg/vdo0
lvreduce −L-50G vg/vdo1
lvresize −L200G vg/vdo2

5. Component activation of VDODataLV

VDODataLV can be activated separately as component LV for examination purposes. It activates data LV in read-only mode and cannot be modified. If the VDODataLV is active as component, any upper LV using this volume CANNOT be activated. User has to deactivate VDODataLV first to continue to use VDOPoolLV.

Example
lvchange −ay vg/vpool0_vdata
lvchange −an vg/vpool0_vdata

VDO Topics

1. Stacking VDO

User can convert/stack VDO with existing volumes.

2. VDO on top of raid

Using Raid type LV for VDO Data LV.

Example
lvcreate −−type raid1 −L 5G −n vpool vg
lvconvert −−type vdo−pool −V 10G vg/vpool

3. Caching VDODataLV, VDOPoolLV

Cache VDO Data LV (accepts also VDOPoolLV.

Example
lvcreate −L 5G −V 10G −n vdo1 vg/vpool
lvcreate −−type cache−pool −L 1G −n cpool vg
lvconvert −−cache −−cachepool vg/cpool vg/vpool
lvconvert −−uncache vg/vpool

3. Caching VDOLV

Cache VDO LV.

Example

```
# lvcreate −L 5G −V 10G −n vdo1 vg/vpool
# lvcreate −−type cache−pool −L 1G −n cpool vg
# lvconvert −−cache −−cachepool vg/cpool vg/vdo1
# lvconvert −−uncache vg/vdo1
```

SEE ALSO

lvm(8), **lvm.conf**(5), **lvmconfig**(8), **lvcreate**(8), **lvconvert**(8), **lvchange**(8), **lvextend**(8), **lvreduce**(8), **lvresize**(8), **lvremove**(8), **lvs**(8), **vdo**(8), **vdoformat**(8), **vdostats**(8), **mkfs**(8)