PERMABIT
TECHNOLOGY CORPORATION

Albireo
Virtual Data Optimizer

Integration Guide

For Albireo VDO v1.4.2

Permabit Technology Corporation
One Alewife Center, Suite 410
Cambridge, MA 02140 USA

Phone: (+1) 617.252.9600
Fax: (+1) 617.252.9977

info@permabit.com
www.permabit.com

Copyright © 2011 – 2014 Permabit Technology Corporation.

All Rights Reserved. Permabit is a registered trademark and the Permabit logo, Permabit Enterprise Archive, and Scalable Data Reduction are trademarks of the Permabit Technology Corporation. All other products or services mentioned may be covered by registered trademarks, trademarks, service marks, or product names as designated by the companies who market those products.

Legal Notices

Portions of this product are Copyright 2009 – 2014 by Permabit Technology Corporation. THESE PORTIONS ARE PROVIDED ON AN “AS IS” BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED INCLUDING, WITHOUT LIMITATION, ANY WARRANTIES OR CONDITIONS OF TITLE, NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

You are solely responsible for determining the appropriateness of using the product and assume all risks associated therewith, including but not limited to the risks and costs of program errors, compliance with applicable laws, damage to or loss of data, programs or equipment, and unavailability or interruption of operations. You may not reverse engineer, decompile, or disassemble this software except as specifically permitted by applicable law.

NEITHER PERMABIT NOR ITS LICENSORS SHALL HAVE ANY LIABILITY FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING WITHOUT LIMITATION LOST PROFITS), HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OR DISTRIBUTION OF THE PROGRAM OR THE EXERCISE OF ANY RIGHTS GRANTED HEREUNDER, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

Table of Contents

Chapter 1: About This Manual 1
Purpose of This Manual 1
Audience 1
Conventions 1
Related Documentation 2

Chapter 2: Albireo VDO Overview 3
The Albireo Index 3
The VDO Kernel Module (kvdo) 4
Command Line Tools 4

Chapter 3: System Requirements 5
Albireo Index Memory Requirements 5
VDO Storage Requirements 6

Chapter 4: Integrating VDO 7
Introduction 7
Installing the VDO Software 8
Creating VDO Volumes 9
Starting or Stopping VDO 10
Configuring the Albireo Index 11
VDO Recovery After Unclean Shutdown 12
Offline Recovery 12
Online Recovery 13
Forcing a Rebuild 14
Rebuilding Volume Statistics 14
Automatically Enabling or Disabling VDO Volumes at Startup 15
Disabling and Re-enabling Deduplication 16
Managing Free Space 16
Increasing Physical Volume Size 17

Chapter 5: Deployment Scenarios 19

Chapter 6: Using COMPRESS 21
Introduction 21
Installing the COMPRESS Plug-In 21
Enabling and Disabling Compression 21

Chapter 7: VDO Commands 22
vdoInstaller 23
vdo 24

vdoStats 33

List of Figures

4.1: VDO Disk Organization 9
5.1: Deduplicated Block Storage Target 19
5.2: Deduplicated NAS 19
5.3: Deduplicated Unified Storage 20
5.4: Using VDO with Encryption 20

List of Tables

7.1: vdoInstaller Commands 23
7.2: vdoInstaller Options 23
7.3: vdo Sub-Commands 24
7.4: vdo Options 29
7.5: vdo Status Output 32
7.6: vdoStats Options 33
7.7: Default vdoStats Output 34
7.8: vdoStats --queues Output 35
7.9: vdoStats --verbose Output 35

Chapter 1: About This Manual

Purpose of This Manual

This manual describes how to enhance a storage management application to install and configure Albireo Virtual Data Optimizer (VDO).

Audience

This manual is intended for systems administrators and integration engineers. It assumes familiarity with the Linux operating system and system administration tools such as LVM2.

Conventions

The Albireo documentation uses the following typographical conventions:

- *Italics*

  Used primarily for file and path names, command line options, and URLs. Glossary term links also appear italicized in books that contain glossaries (terms are presented in *green italics* when first used).

- `Monospace Font`

  Used in examples to show code, literal strings, and output from commands and programs.

- **`Bold Monospace Font`**

  Used in examples to show commands or other text that should be entered on a command line.

- *`Italic Monospace Font`*

  Used in examples to represent command variables/parameters that should be replaced when entered.

Related Documentation

The Albireo VDO documentation set includes these additional guides:

- ***Albireo VDO Evaluation Guide*****:** Provides guidance to first-time evaluators of Albireo VDO and Albireo COMPRESS.
- ***Albireo VDO Release Notes*****:** Provides a release-specific overview of new features, known issues, and usage recommendations.

Chapter 2:
Albireo VDO Overview

Albireo Virtual Data Optimizer (VDO) is a block virtualization technology that allows OEMs to easily create deduplicated pools of block storage. Deduplication is a technique for reducing the consumption of storage resources by eliminating multiple copies of duplicate data. Instead of writing the same data more than once, each duplicate block is detected and recorded as a reference to the original block. Albireo 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*. Block sharing is invisible to users of the storage, who 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.

The Albireo VDO solution consists of three components:

- **Albireo Index:** Used to detect duplicate blocks
- **kvdo:** A kernel module that loads into the Linux device-mapper layer to provide a deduplicated, thinly provisioned block storage volume
- Command line tools for installing, configuring, and managing deduplicated storage

The Albireo Index

The Albireo index provides the foundation of the VDO product. The single greatest challenge when implementing a deduplication system is to rapidly identify duplicate information across a storage pool that can contain hundreds of billions of items. To achieve acceptable levels of performance the system must, for each new piece of data, quickly determine if that piece is identical to any previously stored piece of data. If a match is found, the storage system can then internally reference the existing item to avoid storing the same information more than once.

The Albireo index is able to identify duplicates in memory more than 99.95% of the time while consuming just one-tenth of a byte of memory per block of stored data (in a typical VDO configuration). VDO thus eliminates the largest deduplication bottleneck, increased I/Os for duplicate chunk detection, in a very low memory footprint. Index lookup averages just *5 microseconds* on flash or 10 *microseconds* on traditional hard disk-based storage. The Albireo index is run in user space via the Albireo `albserver` daemon.

The VDO Kernel Module (kvdo)

`kvdo` is a Linux kernel module providing block-layer deduplication services within the Linux device-mapper layer. In the Linux kernel, the device-mapper serves as a generic framework for managing pools of block storage, allowing the insertion of block processing modules into the storage stack between the kernel's block interface and the actual storage device drivers. It forms the foundation of LVM2 and EVMS, software RAID, dm-crypt disk encryption, and additional features such as file-system snapshots.

`kvdo` is exposed as a block device that can be accessed directly for block storage or presented through one of the many available Linux file systems (such as ext3 and XFS). When `kvdo` receives a request to read a (logical) block of data from a VDO volume, it maps the requested logical block to the underlying physical block, and then reads and returns the requested data.

When `kvdo` receives a request to write a block of data to a VDO volume, it does one of two things:

- If the block contains all binary zeros or a DISCARD/TRIM request, `kvdo` updates its block map.
- Otherwise, `kvdo` allocates a temporary physical block and writes the data to that location.

Next, kvdo acknowledges the write to the caller and then computes a MurmurHash-3 hash of the block contents and sends it to the Albireo index for deduplication advice.

If the Albireo index contains a block with the same hash value, `kvdo` reads the indicated block and does a byte-by-byte comparison of the two blocks to verify that they are identical. If they are indeed identical then `kvdo` updates its block map so that logical block points to the corresponding physical block and releases the temporary physical block. If the Albireo index did not contain an entry for the hash value of the block being written, or the indicated block does not actually contain the same data, `kvdo` updates its block map to make the temporary physical block permanent.

Command Line Tools

Albireo VDO includes the following command line tools for installation, configuration and management:

- **`vdoInstaller`:** Compiles, copies, and loads the `kvdo` module into the Linux kernel via DKMS
- **`vdo`:** Creates, configures, and controls VDO volumes
- **`vdoStats`:** Provides utilization and performance statistics

Chapter 3: System Requirements

Albireo VDO has the following system requirements:

- **Processor Architecture:** 64-bit x86
- **Operating Systems (64-bit):**
  - VDO v1.4.2 has been tested on the following Linux distributions:
    - RedHat Enterprise Linux (RHEL) / CentOS 6.4
    - SLES 11 SP3
    - Debian 6 (using the backported 3.2 kernel*) and Debian 7
  - Based upon request, Permabit can support OEMs on other Linux distributions and derivatives, including:
    - RedHat Enterprise Linux (RHEL) / CentOS 6.3
    - SLES 11 SP2
    - Ubuntu 13.04

  ***** Debian backported packages can be obtained from http://backports-master.debian.org/Mirrors/.
- **Additional System Software:** LVM2, DKMS* 2.x, gcc version 4.4 or 4.3, python 2.6, libc 2.7 or higher, libz zlib1g, kernel headers (or source)

  ***** DKMS is not contained in the SLES repositories but it can be downloaded from http://linux.dell.com/dkms/.
- **RAM:** VDO has two distinct memory requirements. The VDO module itself has a minimum requirement of 150 MB plus an additional 86 MB per 1 TB of physical storage managed. In addition, memory is required for the Albireo index, as described in the Albireo Index Memory Requirements section on page 5.
- **Storage:** VDO can be configured to use up to 256 TB of physical storage. the VDO Storage Requirements section on page 6 gives the calculations for determining the usable size of a VDO managed volume from the physical size of the storage pool the VDO is given.

Albireo Index Memory Requirements

The Albireo index consists of two parts — a compact representation is used in memory which contains at most one entry per unique block, and an on-disk component that records the associated block names presented to the index as they occur, in order. Albireo uses an average of 4 bytes per entry in memory (including cache).

The on-disk component maintains a bounded history of data passed to Albireo. Albireo provides deduplication advice for data that falls within this deduplication window, which contains the names of the most recently seen blocks. The deduplication window allows Albireo to index data as efficiently as possible while limiting the amount of memory required to index large data repositories. Despite the bounded nature of the deduplication window, most datasets show high levels of deduplication as they exhibit high degrees of *temporal locality* — in other words, most deduplication occurs among sets of blocks which were written at about the same time. Furthermore, in general, data being written is more likely to duplicate data that was recently written than data that was written a long time ago. Therefore, for a given workload over a given time interval, deduplication rates will often be the same whether Albireo indexes only the most recent data or all the data.

Because of the effects of temporal locality, it is rarely necessary to allocate enough memory to uniquely index every block on the storage system, which can be prohibitively expensive. Index size requirements are more closely related to the rate of data ingested over a given period of time. For example, consider a storage system with 100 TB of total capacity that grows at a rate of 1 TB per week. With a deduplication window of 4 TB Albireo can detect most redundancy in the dataset on a month-to-month basis.

Albireo's *Sparse Indexing* feature (the recommended mode for VDO) further exploits temporal locality by attempting to retain only the most relevant index entries in memory. Albireo VDO can maintain a deduplication window that is ten times larger while using the same amount of memory. While the sparse index provides the greatest coverage, the dense index provides more advice. For most workloads, given the same amount of memory, the difference in deduplication rates between dense and sparse indexes is negligible.

The memory required for the index is determined by the desired size of the deduplication window. For a dense index, Albireo will provide a deduplication window of 1 TB per 1 GB of RAM. For a sparse index, Albireo will provide a deduplication window of 10 TB per 1 GB of RAM. These windows are generally suitable to address 4 TB (dense) or 40 TB (sparse) of unique data because of the temporal locality effects described above.

VDO Storage Requirements

Albireo VDO requires storage both for VDO metadata, and for the actual Albireo deduplication index:

- VDO writes two types of metadata to its underlying physical storage. The first type scales with the physical size of the VDO volume and uses approximately 1 MB for each 14 GB of physical storage. The second type scales with the logical size of the VDO volume and consumes approximately 1.25 MB for each 1 GB of logical storage.

- The Albireo index is stored on its own volume within the VDO volume group, but is not managed by the associated VDO instance. The amount of storage required depends on the type of index and the amount of RAM allocated to the index. For each 1 GB of RAM, a dense Albireo index will use 17 GB of storage, and a sparse Albireo index will use 170 GB of storage

Chapter 4:
Integrating VDO

Introduction

The `vdo` management tool is used to create, configure, and control VDO volumes. Many VDO management commands must be run with root privileges. If the invoking application does not have the required permissions to run a command, `vdo` will report an error.

The `vdo` tool will locate the requisite Albireo binaries if they can be found in the application's command search path (the `PATH` environment variable). To use the tool without modifying the `PATH`, an additional directory or a colon-separated list of directories can be specified with the `--albireoBinaryPath` option. These binaries are normally located in the *`vdo-directory`*`/bin` directory with the Albireo VDO tarball. Note that, because NFS-mounted directories may not be available at system boot, Albireo binaries should not typically be stored on an NFS-mounted file system.

See the vdo section on page 24 for details on `vdo` commands and options.

The Albireo VDO command line utilities are intended to enable rapid integration with existing software installation, configuration, and management tools. Using these utilities, existing tools may be enhanced to perform the following tasks:

- Installing the VDO Software
- Creating VDO Volumes
- Starting or Stopping VDO
- Configuring the Albireo Index
- VDO Recovery After Unclean Shutdown
- Forcing a Rebuild
- Automatically Enabling or Disabling VDO Volumes at Startup
- Disabling and Re-enabling Deduplication
- Managing Free Space
- Increasing Physical Volume Size

Installing the VDO Software

Albireo VDO is delivered as a tar archive containing the following components:

- The Albireo index daemon, **`albserver`**, used to detect duplicate blocks
- The **`kvdo`** kernel module package, which loads into the Linux device-mapper layer to provide a deduplicated, thinly provisioned block storage volume
- The **`vdo`** management utility for managing and configuring deduplicated storage
- The **`vdoStats`** utility for calculating utilization and performance statistics
- A collection of management scripts

The VDO `kvdo` module package consists of a precompiled binary blob and a kernel glue portion that is shipped as source and gets compiled against the running kernel's header files. The VDO kernel module package is compatible with Linux Dynamic Kernel Module Support (DKMS), a collection of scripts that streamline the process for building out-of-tree kernel modules and managing subsequent updates to either the module package or the kernel.

Prior to using VDO, the gcc compiler and the system's kernel headers must be installed. The kernel module can then be compiled, copied to the system's kernel modules directory, and loaded into the kernel.

The included `vdoInstaller` shell script invokes the appropriate DKMS add, build, and install commands. `vdoInstaller` is designed to be invoked as part of installation and upgrade scripts supplied with a VDO-enabled solution.

A simple example to invoke from an installation utility is:

```
cd vdo-directory/bin
./vdoInstaller install
```

If it is necessary to upgrade the kernel (e.g., for security updates or minor patches), the `kvdo` DKMS package has been enabled for *`AUTOINSTALL`*. If the target system is configured properly for DKMS to detect kernel upgrades then the `kvdo` module will automatically get rebuilt and installed for the new kernel; see the system or DKMS documentation on the `dkms_autoinstaller` service.

If the autoinstaller is not available, the `vdoInstaller` wrapper script may be used to invoke DKMS as follows:

```
cd vdo-directory/bin
./vdoInstaller update --kver=new_kernel_version
```

See the vdoInstaller section on page 23 for details.

Creating VDO Volumes

VDO uses a storage pool delimited by an LVM2 *volume group*, which is an aggregation of physical storage containing of one or more disks, partitions, or even flat files. When a VDO volume is created by a storage management tool, VDO reserves space from the volume group for both an Albireo index and the VDO volume, which interact together to provide deduplicated block storage to users and applications. Figure 4.1 illustrates how these pieces fit together.

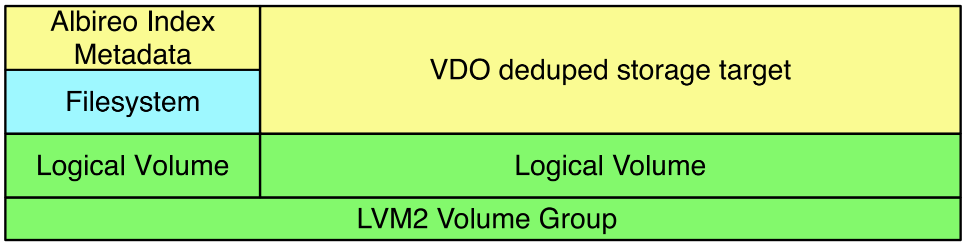

Figure 4.1. VDO Disk Organization

To create a volume group, the elements of the aggregate are first tagged as *physical volumes* for use by LVM2 using the Linux `pvcreate` command, as in this example:

```
    pvcreate /dev/sda2 /dev/sdb2 /dev/sdc2
```

The above command adds three disk partitions to the list of partitions, which may be aggregated into an LVM2 volume group. To form the aggregation, the Linux `vgcreate` command is used, as in this example:

```
    vgcreate my_vg /dev/sda2 /dev/sdb2 /dev/sdc2
```

The new volume group is reported in the output of the Linux `vgs` command, for example:

```
    VG     #PV #LV #SN Attr   VSize VFree
    my_vg 1   2   0 wz--n- 2.70T 137.66G
```

The `VSize` and `VFree` attributes of the volume group will vary based on the sizes of the underlying devices added to the volume group.

After creating a volume group, a VDO volume is created by invoking the `vdo` utility as follows:

```
vdo create \
--name=my_vdo \
--volumeGroup=my_vg \
--vdoLogicalSize=27T \
--albireoSparse \
--albireoBinaryPath=absolute-path-to-binaries
```

This command creates a sparse Albireo index and a new VDO volume at `/dev/mapper/my_vdo`. The exported size (*logical size*) of the volume will be 27 TB, though the `--vdoLogicalSize` option may be used to configure the exported size to some other value; if `--vdoLogicalSize` is not specified, the logical volume size defaults to the physical volume size (note that, in Figure 4.1, the VDO deduped storage target sits completely on top of the logical volume, meaning VDO's physical size is the same size as the logical volume it sits under). VDO currently supports any export size up to 800 times the size of the physical volume.

When VDO volumes are created, VDO will register a startup and shutdown script in `/etc/init.d/` by invoking `insserv` to resolve its dependencies. If the `--albireoBinaryPath` option was specified, this will be added to the command search path used by the script. Otherwise, the directory in which Albireo binaries are found will be added automatically.

A VDO volume can be removed from the system by running:

```
vdo remove --name=my_vdo
```

Prior to removing a VDO volume, unmount file systems and stop applications that are using the storage. The `vdo remove` command removes the VDO volume and its associated Albireo index, as well as logical volumes where they reside.

Starting or Stopping VDO

To start a given VDO volume, or all VDO volumes, and the associated Albireo index(es), storage management utilities should invoke one of these commands:

```
vdo start --name=my_vdo
vdo start --all
```

The VDO script in `/etc/init.d/` (installed when a volume is first created) automatically runs `vdo start --all` at system startup to bring up all *enabled* VDO volumes (see the Automatically Enabling or Disabling VDO Volumes at Startup section on page 15).

To stop a given VDO volume, or all VDO volumes, and the associated Albireo index(es), use one of these commands:

```
vdo stop --name=my_vdo
vdo stop --all
```

If restarted after an unclean shutdown, VDO will perform a rebuild to verify the consistency of its metadata, and will repair it if necessary. Rebuilds are automatic and do not require user intervention. See the section called “VDO Recovery After Unclean Shutdown” for more information on the rebuild process.

VDO supports two operating modes: a synchronous mode designed for flash-based storage, and an asynchronous mode that should be used for hard disk-based storage. In synchronous mode, VDO will not acknowledge a write request until the data and VDO's associated metadata are stably stored. In asynchronous mode, data resiliency is guaranteed only after VDO processes a flush request, which requires that all writes (of data and VDO's associated metadata) that preceded the flush request reside on stable storage.

In synchronous mode, all writes which were acknowledged by VDO prior to the shutdown will be rebuilt. In asynchronous mode, all writes which were acknowledged prior to the last acknowledged flush request will be rebuilt. In either mode, some writes which were either unacknowledged or which were not followed by a flush may also be rebuilt.

Configuring the Albireo Index

VDO uses a high performance deduplication index called Permabit Albireo to detect duplicate blocks of data as they are being stored. Albireo can be configured to recognize duplicate blocks from a configurable history, called the "deduplication window." Blocks within that window, if seen again, will be recognized as duplicate blocks that can be deduplicated.

This index has an in-memory portion as well as an on-disk portion. When configuring the index, you typically specify the size of the in-memory portion (using `--albireoMem=XXX`), and VDO automatically determines how much disk space will be required on disk.

In general, Permabit recommends using a "sparse" Albireo index for all production use cases. This is an extremely efficient indexing data structure, requiring approximately one tenth of a byte of DRAM per block in its deduplication window. On disk, it requires approximately 72 bytes of disk space per block. The minimum configuration of this index uses 256 MB of DRAM, and approximately 45 GB of space on disk. To use this configuration, you would pass the arguments `--albireoSparse --albireoMem=0.25` to the `vdo create`. This configuration results in a deduplication window of 2.5 TB (meaning it will remember a history of 2.5 TB). For most use cases, a deduplication window of 2.5 TB is appropriate for deduplicating storage pools that are up to 10 TB in size.

The default configuration of the index, however is to use a "dense" index. This index is considerably less efficient (by a factor of 10) in DRAM, but it has much lower (also by a factor of 10) minimum required disk space, making it more convenient for evaluation in constrained environments. The default configuration of the albireo index uses 1 GB of DRAM and approximately 18 GB on disk, resulting in a deduplication window of 1 TB (which will be sufficient for storage pools up to 4 TB in size). The minimum configuration (with `--albireoMem=0.25`) uses 256 MB of DRAM and 4.5 GB of disk space, which is appropriate for storage pools up to 1 TB in size.

In general, a deduplication window of *X TB* is recommended for deduplicating up to *4 * X TB* of physical storage. This is not a hard requirement, however. Even small deduplication windows (compared to the amount of physical storage) can find significant amounts of duplicate data in many use cases.

Speak with your Permabit representative for additional guidelines on tuning this important system parameter.

VDO Recovery After Unclean Shutdown

If a volume is restarted without having been shut down cleanly, VDO will need to rebuild a portion of its metadata to continue operating, which occurs automatically when the volume is started. (Also see the section called “Forcing a Rebuild” to invoke this process on a volume that was cleanly shut down.)

If VDO was running with `--writePolicy=sync`, then all data written to the volume will be fully recovered. If the write policy was `async`, then some writes may not be recovered if they were not made durable by sending VDO a `FLUSH` command, or a write I/O tagged with the `FUA` flag (force unit access). This is accomplished from user mode by invoking a data integrity operation like `fsync`, `fdatasync`, `sync`, or `umount`.

VDO offers *offline* and *online* recovery modes. These modes change the characteristics of the recovery process of VDO to better suit the storage environment. The characteristics in question are:

- The time it takes to recover to a full performance state
- The time it takes for the VDO volume to resume user I/O

The recovery option and related parameters must be chosen at volume creation time and cannot be modified later.

Offline Recovery

Offline recovery is the default VDO rebuild setting. In this mode, VDO performs the entire rebuild process before coming back online. This mode provides the fastest recovery of lost data and resumption of deduplication and compression services. However, with offline recovery, the VDO volume is unavailable until the recovery process completes. The time this takes will vary based on the system configuration (described below).

During offline recovery, VDO reads and processes all of its *block map metadata*. This metadata provides the mapping between logical and physical addresses. There will be approximately 1.25 GB of this metadata per TB of logical storage, so a 100 TB logical volume will have 125 GB of metadata to be processed during recovery.

The recovery process requires both CPU and I/O resources to complete. The time it takes will vary depending on how much of the logical space is in use, whether the remaining logical addresses have never been used, or if logical addresses were previously used and then TRIMmed or zeroed. The following measurements of recovery time were made on a 3.3 GHz Intel Xeon processor with an SSD array capable of 2 GB/s of sequential read throughput. Recovery times will differ based on the performance and available resources of the target platform.

- Never-before-used areas of the volume are recovered at a rate of 6.25 seconds per 10 TB of logical address space.

- Used and TRIMmed areas of the volume are recovered at a rate of 12.5 seconds per 10 TB of logical address space.
- Used areas of the volume are recovered at a rate of 77 seconds per 10 TB of logical address space.

Online Recovery

In many environments, extended interruptions in I/O service cannot be tolerated. In those situations, the "online recovery" option may be used, which will allow VDO to restore availability very quickly. When VDO is started, the recovery journal is processed and then the VDO volume is available for read and write operations (usually in under 10 seconds) while the majority of the metadata is recovered in the background. Deduplication and compression services do not run when writing to the recovery reserve; they will resume after the recovery is complete.

Until VDO completes its recovery, it will not be able to tell which physical data blocks are free and available for use, versus which ones are allocated. Therefore, when the volume is created, it is necessary to set aside some number of data blocks specifically for use during the recovery process. This is called the "recovery reserve" and it allows VDO to service write requests while it completes its metadata rebuild process in the background. This is called the "recovery reserve" and it allows VDO to service write requests while it completes its metadata rebuild process in the background. The amount of free space during the online recovery process will therefore be limited to the "recovery reserve," and if too much data is written before the recovery completes, VDO will eventually run out of space (resulting in I/O errors).

After the online recovery completes, VDO will attempt to replenish its recovery reserve using other free blocks of storage. However, if there is another failure before this process completes, then subsequent online recovery processes may not have as much reserved space as expected.

The duration of the online recovery will depend on the amount of logical space belonging to the VDO volume, as well as the recovery rate parameters (described below) that control how fast the recovery process will be allowed to proceed. If recovery rate parameters are see to low values, then I/Os from clients will be minimally impacted, but the recovery process may take a long time to complete. If recovery rate parameters are set to high values, client I/O may be severally impacted, but the recovery process will complete more quickly.

During an online recovery, a number of statistics (e.g., blocks in use, blocks free) will be unavailable. vdoStats will return NaN for these items during recovery. These statistics will become available once the rebuild is complete.

To enable online recovery, three additional parameters must be supplied to the `vdo create` command:

- `--vdoRecoveryReserveSize=`*`megabytes`*

  `vdoRecoveryReserveSize` specifies the maximum amount of space to reserve for servicing write requests while the VDO volume is being rebuilt. During normal operations, this space will be unavailable. To ensure that writes to not fail during an online rebuild, the reserve should be at least as large as the amount of data expected to be written to the VDO volume during that time.

- `--vdoRecoveryScanRate=`*`value`*

  As described in the section called “Offline Recovery” rebuilding a VDO volume requires reading all of its *block map metadata*, which is the longest job during a rebuild. `vdoRecoveryScanRate` limits the read rate of this process. The `vdoRecoveryScanRate` can be set to any positive integer, and results in a read rate limited to 78 KB * `vdoRecoveryScanRate`. If the `blockMapPageSize` is set to 32 KB, this rate will be rounded up to a multiple of 624 KB. These are upper bounds for the read rate; the system may not be able to achieve the rate and take a longer time to complete the recovery than expected. Setting a higher rate will shorten the rebuild time, but it will also degrade user I/O performance during the rebuild.

- `--vdoRecoverySweepRate=`*`value`*

  Rebuilding a VDO volume requires CPU resources in addition to I/O resources. One phase of rebuilding the VDO volume's in-memory state requires an amount of processing proportional to the physical size of the VDO volume. This parameter limits the rate at which that processing is done. Setting a higher rate will shorten the rebuild time, but will also degrade user I/O performance during the rebuild. A value of `1` corresponds to a maximum processing rate of 2 GB of physical storage per second; the limit scales up linearly as this parameter is increased (positive integer values only).

Selecting values for these three parameters is a complex task. The recovery reserve should be as small as possible to maximize usable storage, and therefore the scan and sweep rates should be as large as possible. However, if the scan and sweep rates are too high, user I/O will be starved by the recovery process. The maximum achievable scan and sweep rates are also limited by system performance. Increasing these values beyond that level will not result in higher rates. It should also be noted that even with very large scan and sweep rates, the online recovery time will be longer than the corresponding offline recovery, because VDO must deal with the possibility of accepting user I/O. Permabit's support team should be contacted for help in tuning these parameters.

Forcing a Rebuild

VDO can recover from most hardware and software errors. If a VDO volume cannot be recovered successfully, it is placed into a read-only mode that persists across volume restarts. Once a volume is in read-only mode, there is no guarantee that data has not been lost or corrupted. In such cases, Permabit recommends copying the data out of the read-only volume and possibly restoring the volume from backup. (`vdoStats`' `operating mode` attribute indicates whether a VDO volume is in read-only mode.)

If the risk of data corruption is acceptable, it is possible to force an offline rebuild of the VDO volume metadata so the volume can be brought back online and made available. Again, the integrity of the rebuilt data cannot be guaranteed.

To force a rebuild, first stop the volume if it is running:

```
vdo stop --name=my_vdo
```

Then restart the volume using the `--forceRebuild` option:

```
vdo start --name=my_vdo --forceRebuild
```

Rebuilding Volume Statistics

It is possible to force an offline rebuild of VDO volume metadata to bring its statistics up to date, which can be useful in some testing scenarios.

First stop the volume, if it is running:

```
vdo stop --name=my_vdo
```

Then restart the volume using the `--rebuildStatistics` option:

```
vdo start --name=my_vdo --rebuildStatistics
```

Automatically Enabling or Disabling VDO Volumes at Startup

When `vdo create` is used for the first time, VDO registers itself in `/etc/init.d/` so that volumes will be restarted whenever the system reboots. It accomplishes this by running `vdo start --all`.

To prevent certain volumes from being automatically started, management tools may *disable* those volumes by running either of these commands:

```
vdo disable --name=my_vdo
vdo disable --all
```

Conversely, to enable volumes, management tools can run one of these commands:

```
vdo enable --name=my_vdo
vdo enable --all
```

Note that, if you require multiple layers of LVM (e.g., Figure 5.3 on page 20), it is vital that services are started in order:

1. The first layer of LVM first needs to be started (this is done automatically by an init script configured in most systems by default when the LVM2 package is installed).

2. VDO must then be started (via the script it installs in `/etc/init.d/`)

3. A second startup script needs to run so that it can identify and start the volumes that reside in the volume group on top of VDO, as well as any other services such as file systems that are layered on top of VDO.

A sample of the second startup script is provided in `*vdo-directory*/bin/vdoMounts.init` in the Albireo VDO tarball. This can be used as a model for writing a script for a specific deployment, and should be modified to control any other services layered on top of VDO.

Disabling and Re-enabling Deduplication

Disabling deduplication provides users with a way of continuing to write and read data to a VDO volume without deduplication present; doing so may be desirable in situations where memory used by the Albireo index needs to be freed, or if the performance impact of deduplication is too great.

To stop deduplication on a VDO volume, management tools should use the following command:

```
vdo disableDeduplication --name=my_vdo
```

This stops the associated Albireo index and informs the VDO volume that deduplication is no longer active.

To restart deduplication on a VDO volume, management tools should use the following command:

```
vdo enableDeduplication --name=my_vdo
```

This restarts the associated Albireo index and informs the VDO volume that deduplication is active again.

Managing Free Space

Because Albireo VDO is a thinly provisioned block storage target, the amount of physical space VDO uses may differ from the size of the volume exported to users of the storage. Integrators and systems administrators can exploit this disparity to save on storage costs, but must take care to avoid unexpectedly running out of storage space if the data written does not achieve the expected rate of deduplication.

When using Albireo VDO as a thinly provisioned storage target, it is possible that there will be more logical blocks exposed to end-users than there are actual physical blocks backing the device. By allowing the physical and logical sizes of the devices to differ, storage users can increase their storage efficiency by making use of free space recovered via deduplication, or by deferring the purchase of new storage until it becomes needed.

Whenever the number of logical blocks (virtual storage) exceeds the number of physical blocks (actual storage), it becomes possible for file systems and applications to unexpectedly run out of space. For that reason, storage systems using Albireo VDO must provide storage administrators with a way of monitoring the size of the VDO's free pool. The size of this free pool may be determined by using the `vdoStats` utility. The default output of this utility lists information for all volumes configured on the system in a format similar to the Linux `df` utility. For example:

```
Device             1K-blocks   Used        Available   Use%
/dev/mapper/my_vdo 211812352   105906176   0           50%
```

If the size of VDO's free pool drops below a certain level, the storage administrator can take action by deleting data (which will reclaim space whenever the deleted data is not duplicated), adding physical storage, or even deleting LUNs.

See the vdoStats section on page 33 for details.

NOTE: VDO cannot reclaim space unless file systems communicate that blocks are free using DISCARD/TRIM/UNMAP commands. For file systems that do not use DISCARD/TRIM/UNMAP, free space may be manually reclaimed by storing a file consisting of binary zeros and then deleting that file.

Filesystems may generally be configured to issue DISCARD requests in either of two ways: *realtime discard* (a.k.a. online discard, or inline discard) or *batch discard*. Typically, online discard is enabled by setting the "discard" option at mount time.

When realtime discard is enabled, filesystems will send REQ_DISCARD requests to the block layer whenever a user deletes a file and frees space. VDO will recieve these requests and return space to its free pool (assuming the block was not shared). **In general, realtime discard is not recommended** because of its impact on filesystem performance.

For most applications, batch discard will be the preferred option. Batch discard is a user-initiated operation which causes the filesystem to notify the block layer (VDO) of any unused blocks. This is accomplished by sending the filesystem an `ioctl()` called `FITRIM`. The Linux command-line tool called `fstrim` may be used (e.g., from cron) to send this `ioctl` to the filesystem.

It is also possible to manage free space when the storage is being used as a block storage target without a filesystem. For example, a single VDO volume can be carved up into multiple subvolumes by installing the linux Logical Volume Manager (LVM2) on top of it. Before deprovisioning a volume, the `blkdiscard` command can be used in order to free the space previously used by that logical volume. LVM2 supports the REQ_DISCARD command and will forward the requests to VDO at the appropraite logical block addresses in order to free the space. If other volume managers are being used, they would also need to support REQ_DISCARD (or equivalently, UNMAP for SCSI devices or TRIM for ATA devices).

VDO volumes (or portions of volumes) may also be provisioned to hosts on a Fibre Channel storage fabric or an Ethernet network using SCSI target frameworks such as LIO or SCST. SCSI initiators can use the "UNMAP" command to free space on thinly provisioned storage targets, but the SCSI target framework will typically need to be configured to advertise support for this command. This is typically done by enabling "thin provisioning" on these volumes. Support for UNMAP can be verified on Linux-based SCSI initiators by running "`sg_vpd --page=0xb0 /dev/path`" and verifying that the "Maximum unmap LBA count" is greater than zero, or by using the Visual SCSI Explorer tool on Windows.

Increasing Physical Volume Size

Management applications can increase the amount of physical storage utilized by a VDO volume using the `vdo growPhysical` command.

Depending on the amount of free space in the volume group in which a VDO volume resides, the volume group may require additional physical storage. Storage may be added to LVM by using the Linux `pvcreate` and `vgextend` utilities as follows:

```
pvcreate /dev/sdd2
vgextend my_vg /dev/sdd2
```

The new size of the volume group appears in the output of the `vgs` command, for example:

```
VG    #PV #LV #SN Attr   VSize VFree
```

```
    my_vg 1   2   0 wz--n-   3.60T 1027.66G
```

Before changing the VDO volume, the invoking management application should terminate any programs that are using the volume and disable any service running on top of the device (such as a file system). Management utilities may then allocate new space to a VDO volume as follows:

```
    vdo growPhysical --name=my_vdo --vdoPhysicalSize=3.60T
```

The value specified using the `--vdoPhysicalSize` option is the desired physical size of the VDO volume. The `vdo` command cannot shrink a volume.

Chapter 5: Deployment Scenarios

Albireo VDO can be deployed in a variety of ways to provide deduplicated storage for both block and file access, and for both local and remote storage. Because Albireo VDO exposes its deduplicated storage as a standard Linux block device, it can be used with existing file systems, iSCSI and FC target drivers, or as unified storage.

As a simple example, the entirety of the Albireo VDO storage target can be exported as an iSCSI Target to remote iSCSI initiators (see `linux-iscsi.org`), as illustrated in Figure 5.1.

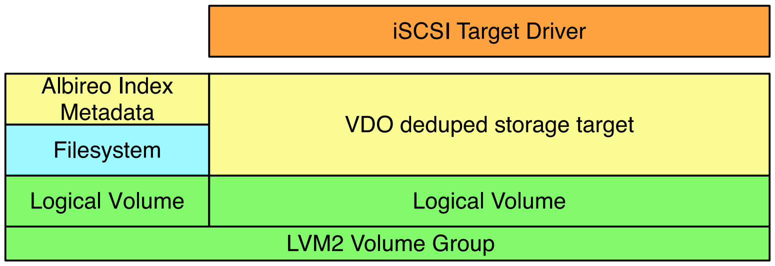

Figure 5.1. Deduplicated Block Storage Target

If file access is desired instead, file systems can be created on top of VDO and exposed to NFS or CIFS users via either the Linux NFS server or Samba (Figure 5.2).

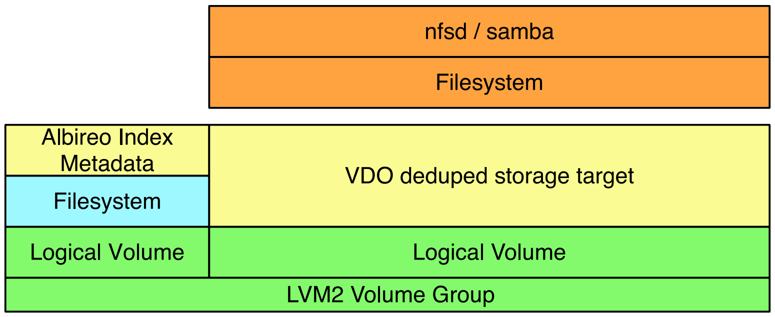

Figure 5.2. Deduplicated NAS

More feature-rich systems may make further use of LVM2 to provide multiple LUNs that are all backed by the same deduplicated storage pool. In Figure 5.3, the VDO target is registered as a physical volume so that it can be managed by LVM2. Multiple logical volumes (LV1-LV4) are created out of the deduplicated storage pool. In this way, Albireo VDO can support multiprotocol unified block/file access to the underlying deduplicated storage pool.

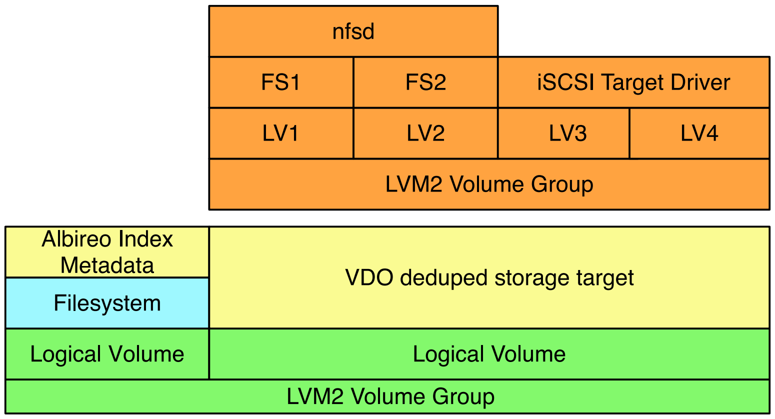

Figure 5.3. Deduplicated Unified Storage

Deduplicated unified storage design allows for multiple file systems to collectively use the same deduplication domain through the LVM tools. Also, file systems can take advantage of LVM's snapshot, copy-on-write, and shrink/grow features all on top of VDO.

Data security is critical today, as more and more companies have internal policies regarding data encryption. VDO works with the Linux device-mapper to support mechanisms such as dm-crypt (Figure 5.4). Encrypting VDO volumes will help ensure data security, and any file system(s) above VDO gain the deduplication feature for disk optimizations. Note that applying encryption above VDO may result in little if any data deduplication because encryption changes the blocks prior to hashing and indexing.

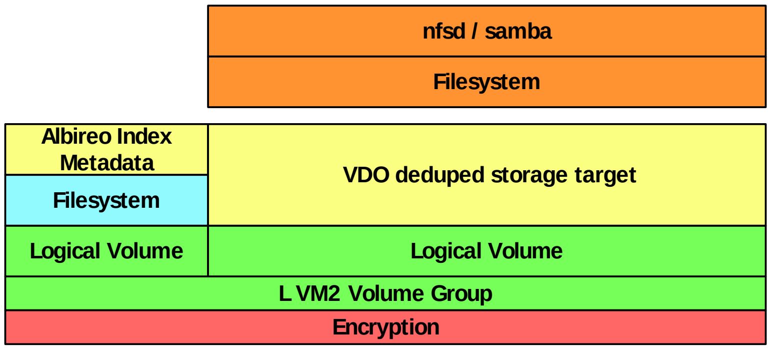

Figure 5.4. Using VDO with Encryption

Chapter 6: Using COMPRESS

Introduction

Albireo COMPRESS is a separately licensed data optimization plug-in that enhances VDO by providing inline, block-level compression. While deduplication is the optimum solution for virtual machine environments and backup applications, compression works very well with structured and unstructured file formats that do not typically exhibit block level redundancy, such as log files and databases.

Albireo COMPRESS operates on blocks that have not been identified as duplicates. After first writing the block to the storage medium and responding to the requestor, Albireo COMPRESS uses a best-fit packing algorithm to find multiple blocks which, when compressed, can fit into a single physical block. When Albireo COMPRESS determines that a particular physical block is unlikely to hold additional compressed blocks, it is written to the storage and the uncompressed blocks can be freed and reused. By performing the compression and packing operations after having already responded to the requestor, Albireo COMPRESS imposes a minimal latency penalty.

Installing the COMPRESS Plug-In

Albireo COMPRESS is delivered as a tar archive; see the README file within that archive for installation instructions.

Enabling and Disabling Compression

VDO volume compression is off by default. To start it, management tools should use the following command:

```
vdo enableCompression --name=volume
```

Compression can be stopped if necessary to maximize performance or to speed processing of data that is unlikely to compress. To stop compression on a VDO volume, management tools should use the following command:

```
vdo disableCompression --name=volume
```

Once compression stops, partially-full bins of compressed fragments are immediately written to disk.

Chapter 7: VDO Commands

This chapter describes the following Albireo VDO binary services and utilities:

- **vdoInstaller:** The `vdoInstaller` utility installs, uninstalls, updates, and provides status on the `kvdo` device mapper module.
- **vdo:** The `vdo` utility manages both the `kvdo` and `albserver` components of VDO.

  The `vdo` is also used to enable or disable compression via the Albireo COMPRESS plug-in, if installed.
- **vdoStats:** The `vdoStats` utility displays output for each configured (or specified) device in a format similar to the Linux `df`.

vdoInstaller

The `vdoInstaller` utility installs, uninstalls, updates, and provides status on the `kvdo` device mapper module.

Synopsis

`vdoInstaller` [install **|** uninstall **|** update **|** status] [OPTIONS...]

Commands

Table 7.1. vdoInstaller Commands

Command	Description
`install`	Compiles, installs, and loads the `vdo` kernel module and arranges for it to be loaded automatically at system startup. This command only needs to be run once for a given system — if the `vdo` kernel module needs to be installed for a new kernel, use the `update` command.
`uninstall`	Deletes the `vdo` kernel module (if installed) for all kernel versions and removes it from the system startup list. Does nothing if the module is not installed.
`update`	If a new kernel has been installed, then a manual update may be required. This action builds and installs the `vdo` kernel module for the new kernel.
`status`	Displays the current installation status of the `vdo` kernel module. This command has no options, and does not need to be run with root privileges.

Options

Table 7.2. vdoInstaller Options

Option	Description
`--kver=`*`version`*	Sets the kernel version to build and install against; defaults to the running kernel.
`--moduleName=`*`name`*	The module name to use when it is installed; defaults to `kvdo`.
`--moduleSourceDir=`*`directory`*	The directory from which the `install` command copies the `vdo` kernel module package; defaults to the "drivers" directory.
`--noRun`	Displays the commands instead of executing them.
`--verbose`	Displays the commands before executing them.

vdo

The `vdo` utility manages both the `kvdo` and `albserver` components of VDO.

Synopsis

`vdo` [--name=*`name`* | --all] [OPTIONS...]
[create | remove | start | stop | enable | disable | status | list | modify | enableDeduplication | disableDeduplication | enableCompression | disableCompression | growPhysical | printConfigFile]

Sub-Commands

Table 7.3. vdo Sub-Commands

Sub-Command	Description
`create`	Creates a VDO volume and its associated Albireo index and makes it available unless the `--noEnable` option is provided. This command must be run with root privileges. Applicable options include: • `--name=`*`volume`* (required) • `--volumeGroup=`*`group`* • `--albireoIndexDir=`*`absolute_path`* • `--lvIndex=`*`volume`* • `--lvVdo=`*`volume`* • `--port=`*`port`* • `--albireoBinaryPath=` *`path[:path ...]`* • `--albireoSize=`*`megabytes`* • `--albireoMem=`*`gigabytes`* • `--albireoSparse` • `--blockMapCacheSize=`*`megabytes`* • `--blockMapPageSize=`*`bytes`* • `--confFile=`*`file`* • `--enable512e` • `--mdRaid5Mode=[ on | off ]` • `--noEnable` • `--vdoLogLevel=`*`level`* • `--vdoLogicalSize=`*`megabytes`* • `--vdoPhysicalSize=`*`megabytes`*

Sub-Command	Description
	• `--vdoReadCacheSize=`*`megabytes`* • `--vdoRecoveryReserveSize=`*`megabytes`* • `--vdoRecoveryScanRate=`*`value`* • `--vdoRecoverySweepRate=`*`value`* • `--verbose` • `--writePolicy=[sync | async]` • `--noRun`
`remove`	Removes one or more stopped VDO volumes and associated Albireo indexes. This command must be run with root privileges. Applicable options include: • `--name=`*`volume`* • `--all` • `--albireoBinaryPath=` *`path[:path ...]`* • `--confFile=`*`file`* • `--force` • `--verbose` • `--noRun`
`start`	Starts one or more stopped, enabled VDO volumes and associated Albireo services. This command must be run with root privileges. Applicable options include: • `--name=`*`volume`* • `--all` • `--albireoBinaryPath=` *`path[:path ...]`* • `--confFile=`*`file`* • `--forceRebuild` • `--rebuildStatistics` • `--verbose` • `--noRun`
`stop`	Stops one or more running VDO volumes and associated Albireo services. This command must be run with root privileges. Applicable options include: • `--name=`*`volume`* • `--all` • `--albireoBinaryPath=` *`path[:path ...]`* • `--confFile=`*`file`* • `--force`

Sub-Command	Description
	• `--verbose` • `--noRun`
`enable`	Enables one or more VDO volumes. Enabled volumes can be started using the `start` command. This command must be run with root privileges. Applicable options include: • `--name=`*`volume`* • `--all` • `--albireoBinaryPath=` *`path[:path ...]`* • `--confFile=`*`file`* • `--verbose` • `--noRun`
`disable`	Disables one or more VDO volumes. Disabled volumes cannot be started by the `start` command. Disabling a currently running volume does not stop it, but once stopped it must be enabled before it can be started again. This command must be run with root privileges. Applicable options include: • `--name=`*`volume`* • `--all` • `--albireoBinaryPath=` *`path[:path ...]`* • `--confFile=`*`file`* • `--verbose` • `--noRun`
`status`	Reports VDO system and volume status in YAML format. Status information will be incomplete if the command is not run with root privileges. Applicable options include: • `--albireoBinaryPath=` *`path[:path ...]`* • `--confFile=`*`file`* See Table 7.5 for the output provided.
`list`	Displays a list of available VDO volumes. Applicable options include: • `--albireoBinaryPath=` *`path[:path ...]`*
`modify`	Modify the configuraion a VDO volume. Configuration changes may not take effect until the VDO volume is restarted. Applicable options include: • `--name=`*`volume`* • `--mdRaid5Mode=[ on | off ]`

Sub-Command	Description
	• `--writePolicy=[sync | async]` • `--albireoBinaryPath=` *`path[:path ...]`*
`enableDeduplication`	Enables deduplication on one or more VDO volumes. This command must be run with root privileges. Applicable options include: • `--name=`*`volume`* • `--all` • `--albireoBinaryPath=` *`path[:path ...]`* • `--confFile=`*`file`* • `--verbose` • `--noRun`
`disableDeduplication`	Disables deduplication on one or more VDO volumes. This command must be run with root privileges. Applicable options include: • `--name=`*`volume`* • `--all` • `--albireoBinaryPath=` *`path[:path ...]`* • `--confFile=`*`file`* • `--verbose` • `--noRun`
`enableCompression`	Enables compression on one or more VDO volumes (requires the COMPRESS plug-in). This command must be run with root privileges. Applicable options include: • `--name=`*`volume`* • `--all` • `--albireoBinaryPath=` *`path[:path ...]`* • `--confFile=`*`file`* • `--verbose` • `--noRun`
`disableCompression`	Disables compression on one or more VDO volumes (requires the COMPRESS plug-in). This command must be run with root privileges. Applicable options include: • `--name=`*`volume`* • `--all` • `--albireoBinaryPath=` *`path[:path ...]`* • `--confFile=`*`file`*

Sub-Command	Description
	• `--verbose` • `--noRun`
`growPhysical`	Grows the physical size of a VDO volume. The volume must exist and must be running. This command must be run with root privileges. Applicable options include: • `--name=`*`volume`* • `--all` • `--confFile=`*`file`* • `--vdoPhysicalSize=`*`megabytes`* • `--verbose` • `--noRun`
`printConfigFile`	Prints the configuration file to stdout. Available options include: • `--confFile=`*`file`*

Options

Table 7.4. vdo Options

Option	Description
`--albireoBinaryPath=` *`path[:path ...]`*	Appends one or more directories to this command's search path. Used to locate directories containing Albireo binaries. Separate multiple paths with colons.
`--albireoIndexDir=`*`absolute_path`*	Specifies the directory in which to write the Albireo index, which must be given as an absolute pathname. If the directory does not already exist, it will be created. If the directory does exist, it must be empty. The default is `/mnt/dedupe-index-`*`volume`* where *`volume`* is the VDO volume specified using the `--name` option.
`--albireoMem=`*`gigabytes`*	Specifies the amount of Albireo server memory in gigabytes; the default size is 1 gigabyte. The special decimal values 0.25, 0.5, 0.75 can be used, as can any positive integer.
`--albireoSize=`*`megabytes`*	Specifies the Albireo index size in megabytes. Using a value with a `K`(ilobytes), `M`(egabytes), `G`(igabytes), or `T`(erabytes) suffix is optional. If not specified, a default is calculated based on the memory allocated to the Albireo server via the `--albireoMem` option.
`--albireoSparse`	Enables sparse indexing.
`--all`	Operates on all configured VDO volumes. May not be used with `--name`.
`--blockMapCacheSize=`*`megabytes`*	Specifies the amount of memory allocated for cached block map pages in megabytes. This must be a multiple of `--blockMapPageSize`. Using a value with a `K`(ilobytes), `M`(egabytes), `G`(igabytes), or `T`(erabytes) suffix is optional. The default is `128M`, which is the required minimum. Increasing the cache size can improve performance for some use cases.
`--blockMapPageSize=`*`bytes`*	Specifies page size of the block map in bytes. This must be a multiple of 4096. Using a value with a `K`(ilobytes) or `M`(egabytes) suffix is optional. The default is `32K`, which is recommended for HDD use; a `4K` page size is recommended for SSDs.
`--confFile=`*`file`*	Specifies an alternate configuration file. The default is `vdoconf.xml`.
`--enable512e`	Enables 512 byte block device emulation mode.
`--force`	Unmounts mounted file systems before stopping a VDO volume.
`--forceRebuild`	Forces an offline rebuild before starting a read-only VDO volume so that it may be brought back online and made available. **This option may result in data loss or corruption.**

Option	**Description**
`--help`	Displays the vdo help file.
`--lvIndex=`*`volume`*	Specifies a unique logical volume name for the Albireo index. The name must not already be in use in the volume group (VG) specified by the `--volumeGroup` option. The default is *`volume`*`-index` where *`volume`* is the name of the VDO volume.
`--lvVdo=`*`volume`*	Specifies a unique logical VDO volume name. The name must not already be in use as a volume group (VG) specified by the `--volumeGroup` option. The default is *`volume`*`-backing` where *`volume`* is the name of the VDO volume.
`--mdRaid5Mode=[ on | off ]`	Enables or disables performance optimizations for MD RAID5 storage configurations. The default is `on`.
`--name=`*`volume`*	Operates on the specified VDO volume. May not be used with `--all`.
`--noEnable`	Creates a VDO volume without enabling or starting it.
`--noRun`	Prints commands instead of executing them.
`--port=`*`port`*	Specifies the Albireo server TCP port; must be a positive integer. The port number must not be in use by other network services. The default is `8000`.
`--rebuildStatistics`	Forces an offline rebuild before starting the VDO volume to ensure that the free block statistics are accurate.
`--vdoLogLevel=`*`level`*	Specifies the VDO driver log level: `critical`, `error`, `warning`, `notice`, `info`, or `debug`. Levels are case-sensitive; the default is `info`.
`--vdoLogicalSize=`*`megabytes`*	Specifies the logical VDO volume size in megabytes; used for over-provisioning volumes. Using a value with a `K`(ilobytes), `M`(egabytes), `G`(igabytes), or `T`(erabytes) suffix is optional. This defaults to the same value as `--vdoPhysicalSize`.
`--vdoPhysicalSize=`*`megabytes`*	Specifies the physical size of the VDO volume in megabytes. Using a value with a `K`(ilobytes), `M`(egabytes), `G`(igabytes), or `T`(erabytes) suffix is optional. This defaults to the available free space in the volume group.
`--vdoReadCacheSize=`*`megabytes`*	Specifies the extra VDO volume read cache size in megabytes. This space is in addition to a system-defined minimum. Using a value with a `K`(ilobytes), `M`(egabytes), `G`(igabytes), or `T`(erabytes) suffix is optional. This defaults to `0`.
`--vdoRecoveryReserveSize=`*`megabytes`*	Specifies the maximum amount of space to reserve for servicing write requests during online VDO volume recovery, in megabytes. To ensure that writes to the VDO volume do not fail during an online rebuild, the reserve size should be sufficient to accommodate the amount of data expected to be written

Option	Description
	during that time. Using a value with a `K`(ilobytes), `M`(egabytes), `G`(igabytes), or `T`(erabytes) suffix is optional. This defaults to `0`.
`--vdoRecoveryScanRate=`*`value`*	Specifies the maximum rate at which block map metadata is read during online recovery. The default value is `1`, which corresponds to a maximum read rate of 78 KB/second; the limit scales linearly with this parameter. When the `blockMapPageSize` is set to 32 KB, the resulting rate will be rounded up to a multiple of 624 KB. Setting a higher rate (positive integers only) will shorten the rebuild time, but will also degrade user I/O performance during the rebuild.
`--vdoRecoverySweepRate=`*`value`*	Limits the rate of online recovery processing. The default value is `1`, which corresponds to a maximum processing rate of 2 GB of physical storage per second; the limit scales linearly with this parameter. Setting a higher rate (positive integers only) will shorten the rebuild time, but will also degrade user I/O performance during the rebuild.
`--verbose`	Prints commands before executing them.
`--volumeGroup=`*`group`*	Specifies the volume group to use. The default is `dedupevg`.
`--writePolicy=[sync | async]`	Specifies the write policy: • `sync`: Writes are acknowledged only after data is on stable storage. This is the default. • `async`: Writes are acknowledged after data has been *cached* for writing to stable storage.

The `status` command returns the following information in YAML format, divided into keys as follows:

Table 7.5. vdo Status Output

Key	Description	
VDO Status	Information in this key covers the name of the host and date and time at which the status inquiry is being made. Parameters reported in this area include:	
	Node	The hostname of the system on which VDO is running.
	Date	The date and time at which the vdo status command is run.
Kernel module	Information in this key covers the configured kernel.	
	Loaded	Whether or not the kernel module is loaded (True or False).
	Version Information	Information on the version of kvdo that is configured.
Configuration	Information in this key covers the location and status of the VDO configuration file.	
	File	Location of the VDO configuration file.
	Last modified	The last-modified date of the VDO configuration file.
VDOs	Provides configuration information for a list of VDO volumes on the system along with the output of vdoStats. Parameters reported for each VDO volume include:	
	Block size	The block size of the VDO volume, in bytes.
	512 byte emulation	Indicates whether the volume is running in 512 byte emulation mode.
	Enable deduplication	If deduplication is enabled, it will be started automatically at system startup.
	Logical size	The logical size of the VDO volume.
	Physical size	The size of a VDO volume's underlying physical storage.
	Server	The address and port number that kvdo is using to communicate with an Albireo index.
	configured write policy	The configured value of the write policy (sync or async).
	MD RAID5 Mode	Indicates whether optimizations for MD RAID5 are enabled.
VDO Statistics	Output of the vdoStats utility	
Albservers	Displays the configuration for a list of Albireo servers on the system.	

vdoStats

The `vdoStats` utility displays output for each configured (or specified) device in a format similar to the Linux `df`.

The `vdoStats` utility should be run with root privileges, else the output may not be complete.

Synopsis

`vdoStats` [--queues | --verbose | --human-readable | --si | --all] [--version] [device...]

Options

Table 7.6. vdoStats Options

Option	Description
`--queues`	Displays the number of pending, processed, timed out, and waiting requests for each VDO volume and its worker threads. See Table 7.8 for details.
`--verbose`	Displays the utilization and block I/O (bios) statistics for one (or more) VDO devices. See Table 7.9 for details.
`--human-readable`	Displays block values in readable form (Base 2: 1 KB = 2^{10} bytes = 1024 bytes).
`--si`	Displays output similar to that of `--human-readable` but using SI units (Base 10: 1 KB = 10^3 bytes = 1000 bytes).
`--all`	Displays the combined output of `--verbose` and `--queues`.
`--version`	Displays the `vdoStats` version.
`device ...`	Specifies one or more specific volumes to report on. If this argument is omitted, `vdoStats` will report on all devices.

Output

The following example shows sample output if no options are provided, which is described in Table 7.7:

```
Device               1K-blocks    Used         Available     Use%    Space Saving%
/dev/mapper/my_vdo   1932562432   427698104    1504864328    22%     21%
```

Table 7.7. Default vdoStats Output

Item	Description
Device	The path to the VDO volume
1K-blocks	The total number of 1K blocks allocated for a VDO volume (= physical volume size * block size / 1024)
Used	The total number of 1K blocks used on a VDO volume (= physical blocks used * block size / 1024)
Available	The total number of 1K blocks available on a VDO volume (= physical blocks free * block size / 1024)
Use%	The percentage of physical blocks used on a VDO volume (= used blocks / allocated blocks * 100)
Space Saving%	The percentage of physical blocks saved on a VDO volume (= [logical blocks used - physical blocks used] / logical blocks used)

The `--human-readable` option converts block counts into conventional units (1 KB = 1024 bytes):

```
Device               Size   Used     Available     Use%   Space Saving%
/dev/mapper/my_vdo   1.8T   407.9G   1.4T          22%    21%
```

The `--human-readable` and `--si` options convert block counts into SI units (1 KB = 1000 bytes):

```
Device               Size   Used     Available     Use%    Space Saving%
/dev/mapper/my_vdo   2.0T   438G     1.5T          22%     21%
```

The `--queues` (Table 7.8) and `--verbose` (Table 7.9) options display VDO device statistics in YAML format for one (or all) VDO devices. use `--all` to display both sets of statistics.

Table 7.8. vdoStats --queues Output

Item	Description
name	The VDO volume name
pending	The number of pending index queries
processed	The number of index queries completed
timed out	The number of index queries that timed out
waits	The number of pending index requests
workers	Lists the name, priority, and queue statistics for each VDO worker thread

Note that, in Table 7.9, the **bolded** statistics at the top of the table will continue to be reported in future releases. The remaining fields are primarily intended for software support and are subject to change in future releases. Avoid writing management tools or other applications that depend upon those fields or the order in which they are presented.

Table 7.9. vdoStats --verbose Output

Item	Description
data blocks used	The number of physical data blocks being used in a VDO volume
overhead blocks used	The number of physical data blocks used for metadata overhead by a VDO volume (in addition to the `data blocks used` statistic)
logical blocks used	The number of logical data blocks mapped
reserved blocks	The number of physical blocks reserved for use in recovery; this value grows as blocks are reserved, until it equals the value of `recovery reserve capacity`
physical blocks	The total number of physical blocks allocated for a VDO volume
logical blocks	The maximum number of logical blocks that can be presented by a VDO volume
active write policy	The active write policy (sync or async). This is configured via `vdo modidy --writePolicy=sync|async`.
recovery reserve capacity	The number of physical blocks reserved for recovery (as configured via `vdo create --vdoRecoveryReserveSize )`
block size	The configured block size of a VDO volume, in bytes
recovered count	The number of times a VDO volume has been recovered from read-only mode (via `vdo start --forceRebuild`)
operating mode	Indicates whether a VDO volume is operating normally, is in recovery mode, or is in read-only mode
recovery progress	Indicates online recovery progress, or `N/A` if the volume is not in recovery mode
compressed fragments written	The number of compressed fragments that have been written since the VDO volume was last restarted (if the COMPRESS plug-in is installed)

Item	Description
compressed blocks written	The number of physical blocks of compressed data that have been written since the VDO volume was last restarted (if the COMPRESS plug-in is installed)
compressed fragments in packer	The number of compressed fragments being processed that have not yet been written (if the COMPRESS plug-in is installed)
journal tail full count	The number of times recovery journal metadata was generated faster than it could be written from memory to disk
journal disk full count	The number of times the disk location of the recovery journal metadata filled
journal blocks batching	The number of journal block writes started minus the number of journal blocks written
journal blocks writing	The number of journal blocks written (with metadatata in active memory) minus the number of journal blocks committed
journal blocks committed	The number of journal blocks written to disk
journal entries batching	The number of journal entry writes started minus the number of journal entries written
journal entries writing	The number of journal entries written (with metadata in active memory) minus the number of journal entries committed
journal entries committed	The number of journal entries written to disk
512 byte emulation	Indicates whether the volume is running in 512 byte emulation mode
current bios in progress	The number of active block I/O requests
maximum bios in progress	The peak number of active block I/O requests
current dedupe queries	The number of currently active index queries
maximum dedupe queries	The peak number of active index queries
dedupe advice valid	The number of times Albireo advice proved correct
dedupe advice stale	The number of times Albireo advice proved incorrect
dedupe advice timeouts	The number of times the Albireo server was too slow in responding
bios in... bios partial in... bios out... bios meta...	The bios in/partial in/out/meta statistics count the number of Linux block I/O request flags in each category: • **bios in:** The number of block I/O requests received by VDO • **bios partial in:** The number of partial block I/O requests received by VDO. Applies only to 512 byte emulation mode. • **bios out:** The number of non-metadata block I/O requests submitted by VDO to the storage device • **bios meta:** The number of metadata block I/O requests submitted by VDO to the storage device There are seven types of flags: • **read:** The number of non-write bios (bios without the REQ_WRITE flag set)

Item	Description
	• **write:** The number of write bios (bios with the REQ_WRITE flag set) • **prio:** The number of bios with a REQ_PRIO flag set • **discard:** The number of bios with a REQ_DISCARD flag set • **fua:** The number of bios with the REQ_FUA flag set • **flush:** The number of bios with the REQ_FLUSH flag set • **rahead:** The number of bios with the REQ_RAHEAD flag set
read cache accesses	The number of times VDO searched the read cache for a duplicate block
read cache hits	The number of times VDO found a duplicate block in the read cache
1K-blocks	The total number of 1K blocks allocated for a VDO volume (= physical volume size * block size / 1024)
1K-blocks used	The total number of 1K blocks used on a VDO volume (= physical blocks used * block size / 1024)
1K-blocks available	The total number of 1K blocks available on a VDO volume (= physical blocks free * block size / 1024)
used percent	The percentage of physical blocks used on a VDO volume (= used blocks / allocated blocks * 100)
saving percent	The percentage of physical blocks saved on a VDO volume (= [logical blocks used - physical blocks used] / logical blocks used)