



Red Hat Enterprise Linux 8

RHEL 中逻辑卷的重复数据删除和压缩。

在 LVM 上部署 VDO 以增加存储容量

Red Hat Enterprise Linux 8 RHEL 中逻辑卷的重复数据删除和压缩。

在 LVM 上部署 VDO 以增加存储容量

法律通告

Copyright © 2024 Red Hat, Inc.

The text of and illustrations in this document are licensed by Red Hat under a Creative Commons Attribution–Share Alike 3.0 Unported license ("CC-BY-SA"). An explanation of CC-BY-SA is available at

<http://creativecommons.org/licenses/by-sa/3.0/>

. In accordance with CC-BY-SA, if you distribute this document or an adaptation of it, you must provide the URL for the original version.

Red Hat, as the licensor of this document, waives the right to enforce, and agrees not to assert, Section 4d of CC-BY-SA to the fullest extent permitted by applicable law.

Red Hat, Red Hat Enterprise Linux, the Shadowman logo, the Red Hat logo, JBoss, OpenShift, Fedora, the Infinity logo, and RHCE are trademarks of Red Hat, Inc., registered in the United States and other countries.

Linux[®] is the registered trademark of Linus Torvalds in the United States and other countries.

Java[®] is a registered trademark of Oracle and/or its affiliates.

XFS[®] is a trademark of Silicon Graphics International Corp. or its subsidiaries in the United States and/or other countries.

MySQL[®] is a registered trademark of MySQL AB in the United States, the European Union and other countries.

Node.js[®] is an official trademark of Joyent. Red Hat is not formally related to or endorsed by the official Joyent Node.js open source or commercial project.

The OpenStack[®] Word Mark and OpenStack logo are either registered trademarks/service marks or trademarks/service marks of the OpenStack Foundation, in the United States and other countries and are used with the OpenStack Foundation's permission. We are not affiliated with, endorsed or sponsored by the OpenStack Foundation, or the OpenStack community.

All other trademarks are the property of their respective owners.

摘要

使用逻辑卷管理器(LVM)中的 Virtual Data Optimizer (VDO)功能来管理去重和压缩的逻辑卷。您可以将 VDO 作为 LVM 逻辑卷(LV)一种类型（类似于 LVM 精简配置的卷）来管理。您可以在 LVM (LVM-VDO)上部署 VDO，来为块访问、文件访问、本地存储和远程存储提供去重的存储。您还可以配置精简配置的 VDO 卷，以避免 VDO 卷的物理空间被 100% 使用。在将 VDO 卷导入到 LVM 后，您可以使用 LVM 工具管理 VDO 卷。

目录

对红帽文档提供反馈	3
第1章 LVM 的 VDO 介绍	4
第2章 LVM-VDO 要求	5
2.1. VDO 内存要求	5
2.2. VDO 存储空间要求	5
2.3. 按物理大小划分的 VDO 要求示例	6
2.4. 在存储堆栈中放置 LVM-VDO	7
第3章 创建重复数据删除和压缩的逻辑卷	8
3.1. LVM-VDO 部署情况	8
3.2. LVM-VDO 卷的物理和逻辑大小	10
3.3. VDO 中的 LAB 大小	11
3.4. 安装 VDO	11
3.5. 创建 LVM-VDO 卷	12
3.6. 挂载 LVM-VDO 卷	13
3.7. 更改 LVM-VDO 卷中的压缩设置	13
3.8. 在 LVM-VDO 卷中更改重复数据删除设置	14
3.9. 使用虚拟数据优化器管理精简配置	15
第4章 将现有 VDO 卷导入到 LVM 中	19
第5章 在 LVM-VDO 卷中修剪选项	21
5.1. 在 VDO 中启用丢弃挂载选项	21
5.2. 设置定期 TRIM 操作	21
第6章 优化 VDO 性能	23
6.1. VDO 线程类型	23
6.2. 识别性能瓶颈	23
6.3. 重新分发 VDO 线程	27
6.4. 增加块映射缓存大小以增强性能	29
6.5. 加快丢弃操作	30
6.6. 优化 CPU 频率扩展	31

对红帽文档提供反馈

我们感谢您对我们文档的反馈。让我们了解如何改进它。

通过 Jira 提交反馈（需要帐户）

1. 登录到 [Jira](#) 网站。
2. 单击顶部导航栏中的 **Create**。
3. 在 **Summary** 字段中输入描述性标题。
4. 在 **Description** 字段中输入您的改进建议。包括到文档相关部分的链接。
5. 点对话框底部的 **Create**。

第 1 章 LVM 的 VDO 介绍

Virtual Data Optimizer (VDO) 为存储提供内联块级的重复数据删除 (deduplication)、压缩和精简置备。您可以将 VDO 作为逻辑卷管理器(LVM)逻辑卷(LVM)的一种类型 (类似于 LVM 精简配置的卷) 来管理。

LVM (LVM-VDO)上的 VDO 卷包含以下组件：

VDO 池 LV

- 这是用于 VDO LV 存储、重复数据删除和压缩的后端物理设备。VDO 池 LV 设置 VDO 卷的物理大小，即 VDO 可保存到磁盘中的数据量。
- 目前，每个 VDO 池 LV 只能有一个 VDO LV。因此，VDO 会单独去重和压缩每个 VDO LV。存储在单独 LV 上的重复数据不会从同一 VDO 卷的数据优化中受益。

VDO LV

- 这是 VDO 池 LV 上的虚拟置备设备。VDO LV 设定 VDO 卷的置备和逻辑大小，即应用程序在重复数据删除和压缩发生前可写入卷的数据量。

如果您已经熟悉了 LVM 精简配置实现的结构，您可以参考表 1.1，以了解 VDO 的不同方面是如何呈现给系统的。

表 1.1. LVM 和 LVM 精简置备的 VDO 组件的比较

	物理设备	置备的设备
LVM 上的 VDO	VDO 池 LV	VDO LV
LVM 精简配置	精简池 (thin pool)	精简卷 (thin volume)

因为 VDO 是精简置备的，所以文件系统和应用程序只会看到正在使用的逻辑空间，而不是实际可用的物理空间。使用脚本来监控可用的物理空间，并在使用超过阈值时产生一个报警。有关监控可用的 VDO 空间的详情，请参考 [监控 VDO](#) 部分。

其他资源

- [重复数据删除和压缩存储](#)
- [创建并管理精简配置的卷 \(精简卷\)](#)

第 2 章 LVM-VDO 要求

LVM 上的 VDO 对其放置和系统资源有一定要求。

2.1. VDO 内存要求

每个 VDO 卷有不同的内存要求：

VDO 模块

VDO 需要固定的 38 MB RAM 和几个变化的量：

- 配置的每 1 MB 的块映射缓存需要 1.15 MB 的 RAM。块映射缓存至少需要 150MB RAM。
- 每 1TB 的逻辑空间需要 1.6 MB 的 RAM。
- 卷管理的每 1TB 的物理存储需要 268 MB 的 RAM。

UDS 索引

通用去除重复数据服务(UDS)至少需要 250 MB 的 RAM，这也是去除重复数据所使用的默认数量。您可以在格式化 VDO 卷时配置值，因为该值也会影响索引所需的存储量。

UDS 索引所需的内存由索引类型和重复数据删除窗口所需大小决定：

索引类型	重复数据删除窗口	备注
密度	每 1GB RAM 为 1TB	1GB 密度索引一般足以满足 4TB 物理存储空间。
稀疏	每 1GB RAM 为 10 TB	1GB 稀疏索引一般足以满足 40TB 物理存储空间。



注意

使用默认设置的 2 GB slab 和 0.25 密集索引的 VDO 卷的最小磁盘使用量大约需要 4.7 GB。这以 0% 去除重复数据或压缩的比率提供了略低于 2 GB 的物理数据的写入。

这里,最小的磁盘用量是默认 slab 大小和密度索引的总和。

UDS 稀疏索引功能是 VDO 推荐的模式。它依赖于数据的时间局部性，并尝试只在内存中保留最重要的索引条目。使用稀疏索引，UDS 维护一个重复数据删除窗口，它是密度的 10 倍，但使用相同数量的内存。

稀疏索引提供了最高的覆盖，但密度索引提供了更多的重复数据删除建议。对于大多数工作负载，如果内存量相同，则密度和稀疏索引间的重复数据删除率的不同会微不足道。

其他资源

- [按物理大小的 VDO 要求示例](#)

2.2. VDO 存储空间要求

您可以将 VDO 卷配置为使用最多 256TB 物理存储。只有物理存储的某个部分可用来存储数据。本节提供了计算 VDO 管理的卷的可用空间大小的方法。

VDO 需要为两种类型的 VDO 元数据和 UDS 索引进行存储：

- 第一类 VDO 元数据对于每 4GB 物理存储使用 1 MB，再加上每个 slab 的额外的 1 MB。
- 第二类 VDO 元数据对于每 1GB 逻辑存储使用 1.25 MB，并舍入到最近的 slab。
- UDS 索引所需的存储量取决于索引类型以及分配给索引的 RAM 量。对于每 1 GB RAM，密度 UDS 索引使用 17GB 存储，稀疏 UDS 索引使用 170 GB 存储。

其他资源

- [按物理大小划分的 VDO 要求示例](#)
- [VDO 中的 Lab 大小](#)

2.3. 按物理大小划分的 VDO 要求示例

下表根据基础卷的物理大小提供 VDO 的最大系统要求。每个表列出了适合预期部署的需求，如主存储或备份存储。

具体数量取决于您的 VDO 卷的配置。

主存储部署

在主存储中，UDS 索引是物理大小的 0.01% 到 25%。

表 2.1. 主存储的存储和内存要求

物理大小	RAM 使用量： UDS	RAM 使用量： VDO	磁盘用量	索引类型
10GB-1TB	250MB	472MB	2.5GB	密度
2-10TB	1GB	3GB	10GB	密度
	250MB		22GB	稀疏
11-50TB	2GB	14GB	170GB	稀疏
51-100TB	3GB	27GB	255GB	稀疏
101-256TB	12GB	69GB	1020GB	稀疏

备份存储部署

在备份存储中，UDS 索引覆盖了备份组的大小，但小于物理大小。如果您预期备份集或物理大小在以后会增大，则需要把这个值加到索引大小中。

表 2.2. 备份存储的存储和内存要求

物理大小	RAM 使用量： UDS	RAM 使用量： VDO	磁盘用量	索引类型
10GB-1TB	250MB	472MB	2.5 GB	密度
2-10TB	2GB	3GB	170GB	稀疏
11-50TB	10GB	14GB	850GB	稀疏
51-100TB	20GB	27GB	1700GB	稀疏
101-256TB	26GB	69GB	3400GB	稀疏

2.4. 在存储堆栈中放置 LVM-VDO

您必须将特定的存储层放在 VDO 逻辑卷下，并在上面放置其他存储层。

您可以将thick 置备的层放在 VDO 的顶部，但您不能依赖该情况下精简置备的保证。因为 VDO 层是精简置备的，精简置备的效果适用于所有在它上面的层。如果您不监控 VDO 卷，您可能使用完在 VDO 以上的 thick-provisioned 卷的所有物理空间。

以下层支持的放置位于 VDO 下。不要将它们放在 VDO 上：

- DM Multipath
- DM Crypt
- Software RAID (LVM 或 MD RAID)

不支持以下配置：

- VDO 位于回送设备之上
- 加密的卷位于 VDO 之上
- VDO 卷中的分区
- 位于 VDO 卷之上的 RAID，比如 LVM RAID、MD RAID 或者其它类型
- 在 LVM-VDO 上部署 Ceph Storage

其他资源

- [堆栈 LVM 卷知识库文章](#)

第 3 章 创建重复数据删除和压缩的逻辑卷

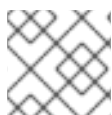
您可以创建使用 VDO 功能的 LVM 逻辑卷来重复数据删除和压缩数据。

3.1. LVM-VDO 部署情况

您可以以不同的方式部署 VDO on LVM (LVM-VDO) 以提供重复数据删除的存储：

- 块访问
- 文件访问
- 本地存储
- 远程存储

因为 LVM-VDO 会将重复数据删除存储作为常规逻辑卷 (LV) 形式公开，所以您可以在标准文件系统、iSCSI 和 FC 目标驱动程序或者统一存储中使用它。

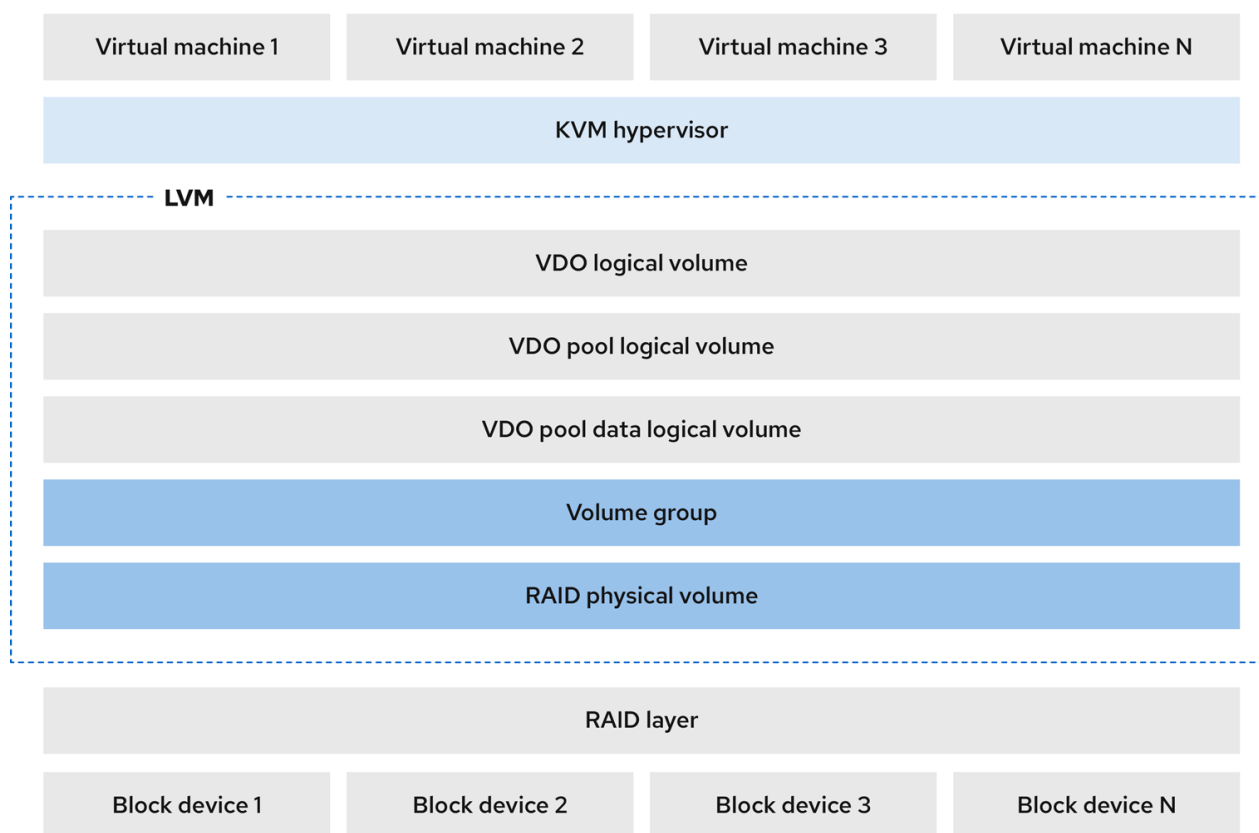


注意

目前不支持在 LVM-VDO 上部署 Ceph Storage。

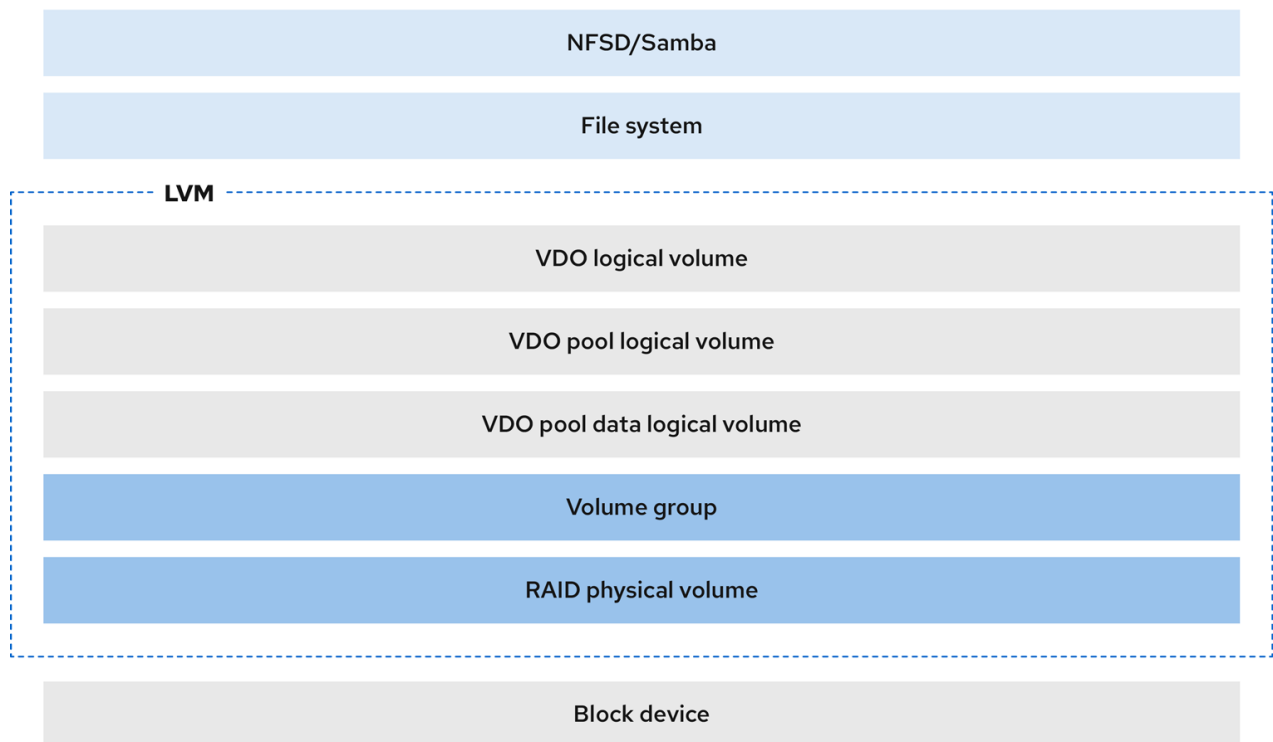
KVM

您可以在配置了直接附加存储的 KVM 服务器中部署 LVM-VDO。



275_RHEL_1022

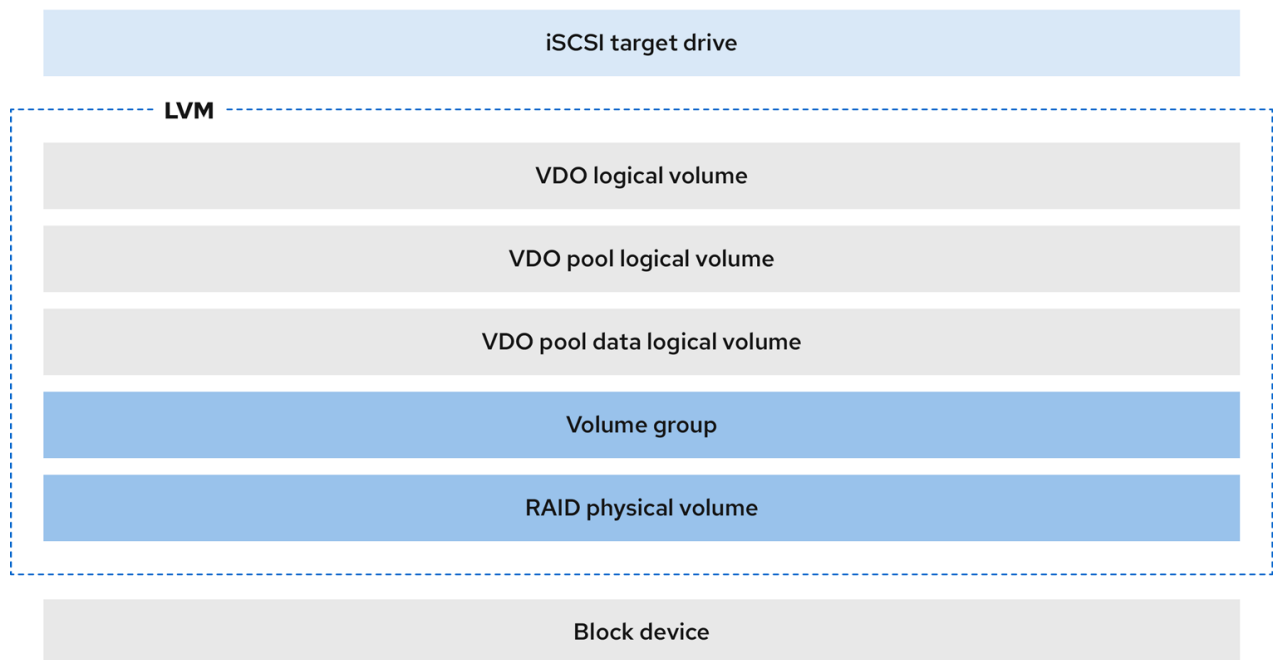
您可以在 VDO LV 上创建文件系统，并将其公开给使用 NFS 服务器或 Samba 的 NFS 或 CIFS 用户。



275_RHEL_1022

iSCSI 目标

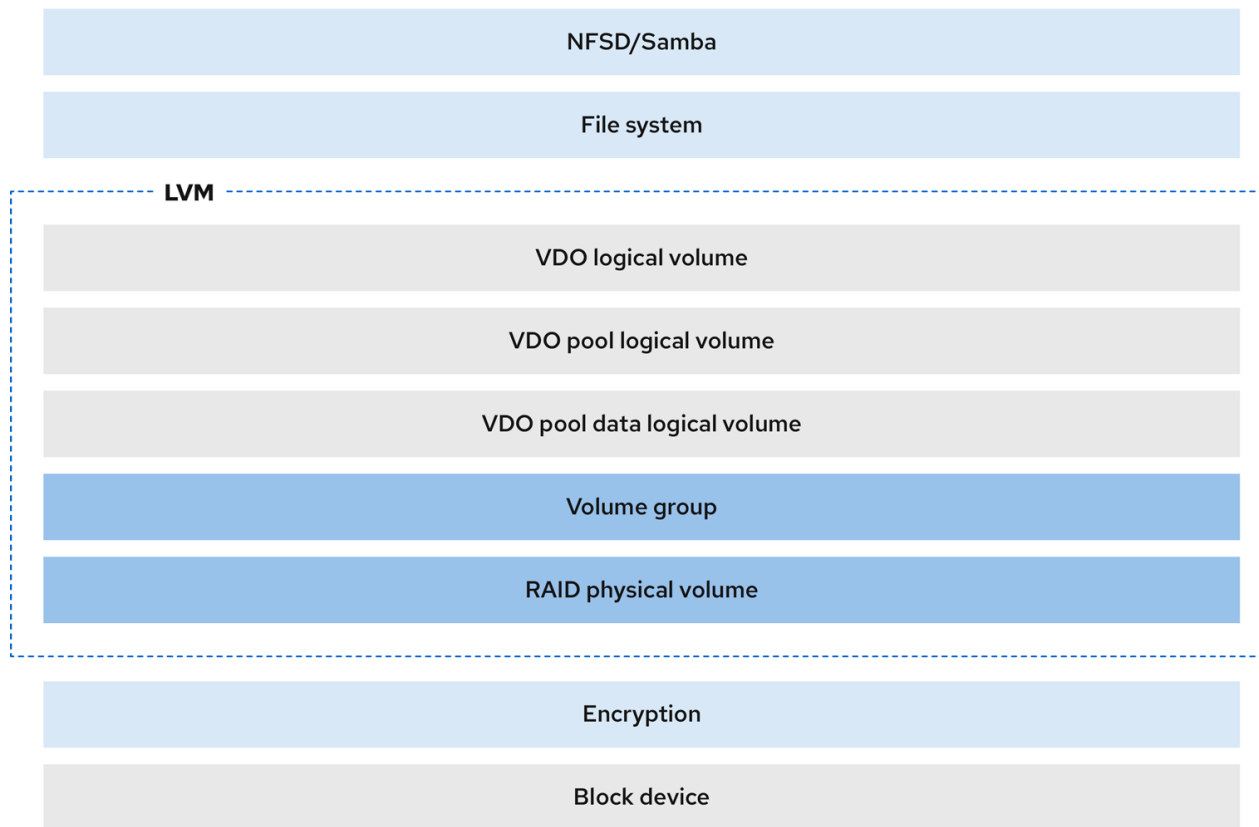
您可以将 VDO LV 的整个导出为 iSCSI 目标到远程 iSCSI 启动器。



275_RHEL_1022

加密

DM Crypt 等设备映射器 (DM) 机制与 VDO 兼容。加密 VDO LV 卷有助于确保数据安全性，且所有 VDO LV 以上的文件系统仍会重复使用。



275_RHEL_1022



重要

如果对数据进行了重复数据删除，则在应用 VDO LV 上面应用加密层不会产生太大效果。在 VDO 可以对它们进行重复数据删除前，加密会使重复数据块不同。

始终将加密层放在 VDO LV 下。

3.2. LVM-VDO 卷的物理和逻辑大小

这部分论述了 VDO 可以使用的物理大小、可用物理大小和逻辑大小。

物理大小

这与分配给 VDO 池 LV 的物理区块大小相同。VDO 使用这个存储用于：

- 用户数据，这些数据可能会进行重复数据删除和压缩
- VDO 元数据，如 UDS 索引

可用物理大小

这是 VDO 可用于用户数据的物理大小的一部分。

它等同于物理大小减去元数据的大小，向下舍入到 slab 大小的倍数。

逻辑大小

这是 VDO LV 出现在应用程序中置备的大小。它通常大于可用的物理大小。VDO 目前支持任意逻辑卷大小最多为物理卷的 254 倍，但不能超过 4 PB。

当您设置 VDO 逻辑卷 (LV) 时，可以指定 VDO LV 出现的逻辑存储量。在托管活跃虚拟机或容器时，红帽建议使用 10:1 逻辑和物理比例置备存储：也就是说，如果您使用 1TB 物理存储，您将会把它显示为 10TB 逻辑存储。

如果没有指定 `--virtualsize` 选项，VDO 会将卷置备为 **1:1** 比例。例如，如果您将 VDO LV 放在 20GB VDO 池 LV 的上面，如果使用默认索引大小，VDO 为 UDS 索引保留 2.5 GB。剩余的 17.5 GB 为 VDO 元数据和用户数据提供。因此，要消耗的可用存储不超过 17.5 GB，且可能会因为组成实际 VDO 卷的元数据而减少。

其他资源

- [按物理大小划分的 VDO 要求示例](#)

3.3. VDO 中的 LAB 大小

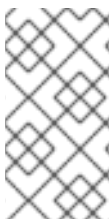
VDO 卷的物理存储被分成几个 slab。每个 slab 都是物理空间的连续区域。给定卷的所有 slab 的大小相同，可以是基于 128 MB 的 2 的指数的任何值，最大值为 32 GB。

默认的 slab 大小为 2 GB，以便于在较小的测试系统上评估 VDO。单个 VDO 卷最多可有 8192 个 slabs。因此，在使用 2GB slab 的默认配置中，允许的最大物理存储为 16 TB。当使用 32GB 的 slab 时，允许的最大物理存储为 256 TB。VDO 总是保留至少一个整个 slab 来保存元数据，因此预留 slab 无法用于存储用户数据。

slab 大小不影响 VDO 卷的性能。

表 3.1. 根据物理卷大小推荐的 VDO slab 大小

物理卷大小	推荐的 slab 大小
10-99 GB	1 GB
100 GB - 1 TB	2 GB
2-256 TB	32 GB



注意

使用默认设置的 2 GB slab 和 0.25 密集索引的 VDO 卷的最小磁盘使用量大约需要 4.7 GB。这以 0% 去除重复数据或压缩的比率提供了略低于 2 GB 的物理数据的写入。

这里,最小的磁盘用量是默认 slab 大小和密度索引的总和。

您可以通过向 `lvcreate` 命令提供 `--config 'allocation/vdo_slab_size_mb=size-in-megabytes'` 选项来控制 slab 大小。

3.4. 安装 VDO

此流程安装创建、挂载和管理 VDO 卷所需的软件。

流程

- 安装 VDO 软件：

```
# yum install lvm2 kmod-kvdo vdo
```

3.5. 创建 LVM-VDO 卷

这个过程在 VDO 池 LV 中创建 VDO 逻辑卷 (LV)。

先决条件

- 安装 VDO 软件。如需更多信息，请参阅[安装 VDO](#)。
- 在您的系统中有一个有可用存储容量的 LVM 卷组。

步骤

1. 为您的 VDO LV 选择一个名称，如 **vdo1**。您必须为系统中的每个 VDO LV 使用不同的名称和设备。

在以下步骤中，将 *vdo-name* 替换为名称。

2. 创建 VDO LV：

```
# lvcreate --type vdo \
  --name vdo-name
  --size physical-size
  --virtualsize logical-size \
  vg-name
```

- 使用您要放置 VDO LV 的现有 LVM 卷组的名称替换 *vg-name*。
- 使用 VDO LV 存在的逻辑存储数量替换 *logical-size*。
- 如果物理大小大于 16TiB，请添加以下选项以将卷的 slab 大小增加到 32GiB：

```
--config 'allocation/vdo_slab_size_mb=32768'
```

如果您在大于 16TiB 的物理大小中使用 2GiB 的默认 slab 大小，则 **lvcreate** 命令会失败并显示以下错误：

```
ERROR - vdoformat: formatVDO failed on '/dev/device': VDO Status: Exceeds maximum
number of slabs supported
```

例 3.1. 为容器存储创建 VDO LV

例如，要为 1TB VDO 池中的容器存储创建 VDO LV，您可以使用：

```
# lvcreate --type vdo \
  --name vdo1
  --size 1T
  --virtualsize 10T \
  vg-name
```


**重要**

如果在创建 VDO 卷时发生故障，请删除要清理的卷。

3. 在 VDO LV 上创建文件系统：

- 对于 XFS 文件系统：

```
# mkfs.xfs -K /dev/vg-name/vdo-name
```

- 对于 ext4 文件系统：

```
# mkfs.ext4 -E nodiscard /dev/vg-name/vdo-name
```

其他资源

- [lvmvdo\(7\) 手册页](#)

3.6. 挂载 LVM-VDO 卷

这个过程会在 LVM-VDO 卷中手动挂载文件系统，也可以永久挂载文件系统。

先决条件

- 您的系统中有 LVM-VDO 卷。如需更多信息，请参阅[创建 LVM-VDO 卷](#)。

步骤

- 要手动将文件系统挂载到 LVM-VDO 卷中，请使用：

```
# mount /dev/vg-name/vdo-name mount-point
```

- 要将文件系统配置为在引导时自动挂载，请在 `/etc/fstab` 文件中添加行：

- 对于 XFS 文件系统：

```
/dev/vg-name/vdo-name mount-point xfs defaults 0 0
```

- 对于 ext4 文件系统：

```
/dev/vg-name/vdo-name mount-point ext4 defaults 0 0
```

如果 LVM-VDO 卷位于需要网络的块设备中，如 iSCSI，请添加 `_netdev` 挂载选项。对于 iSCSI 和其它需要网络的块设备，请查看 [systemd.mount\(5\)](#) 手册页，了解有关 `_netdev` 挂载选项的信息。

其他资源

- [systemd.mount\(5\) 手册页](#)

3.7. 更改 LVM-VDO 卷中的压缩设置

默认情况下，启用了 VDO 池逻辑卷(LV)的压缩。要保存 CPU 用量，您可以禁用它。使用 **lvchange** 命令启用或禁用压缩。

先决条件

- 您的系统中有 LVM-VDO 卷。

步骤

1. 检查逻辑卷的压缩状态：

```
# lvs -o+vdo_compression,vdo_compression_state
LV      VG      Attr      LSize Pool Origin Data% Meta% Move Log Cpy%Sync
Convert VDOCompression VDOCompressionState
vdo_name vg_name vwi-a-v--- 1.00t vpool0 0.00          enabled
online
vpool0  vg_name dwi----- <15.00g          20.03          enabled
online
```

2. 禁用 VDOPoolLV 的压缩：

```
# lvchange --compression n vg-name/vdopoolname
```

如果要启用压缩，请使用 **y** 选项，而不是 **n** 选项。

验证

- 查看压缩的当前状态：

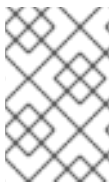
```
# lvs -o+vdo_compression,vdo_compression_state
LV      VG      Attr      LSize Pool Origin Data% Meta% Move Log Cpy%Sync
Convert VDOCompression VDOCompressionState
vdo_name vg_name vwi-a-v--- 1.00t vpool0 0.00          offline
vpool0  vg_name dwi----- <15.00g          20.03          offline
```

其他资源

- [lvmvdo\(7\) 手册页](#)
- [lvcreate\(8\) 手册页](#)

3.8. 在 LVM-VDO 卷中更改重复数据删除设置

默认情况下，启用 VDO 池逻辑卷(LV)的重复数据删除。要保存内存，您可以禁用 deduplication。使用 **lvchange** 命令启用或禁用 deduplication。



注意

由于 VDO 处理持续并行 I/O 操作的方式，VDO 卷将继续识别这些操作中的重复数据。例如，如果虚拟机克隆操作正在进行，且 VDO 卷在接近时有很多重复块，则卷仍可使用 deduplication 来实现一些空间节省。卷的索引状态不会影响进程。

先决条件

- 您的系统中有 LVM-VDO 卷。

步骤

1. 检查逻辑卷的重复数据删除状态：

```
# lvs -o+vdo_deduplication,vdo_index_state
LV      VG      Attr      LSize Pool  Origin Data%  Meta%  Move Log Cpy%Sync
Convert VDOdeduplication VDOIndexState
  vdo_name vg_name vwi-a-v--- 1.00t vpool0  0.00                      enabled
online
  vpool0  vg_name dwi----- <15.00g          20.03                      enabled
online
```

2. 禁用 VDOPoolLV 的 deduplication：

```
# lvchange --deduplication n vg-name/vdopoolname
```

如果要启用 deduplication，请使用 **y** 选项而不是 **n**。

验证

- 查看 deduplication 的当前状态：

```
# lvs -o+vdo_deduplication,vdo_index_state
LV      VG      Attr      LSize Pool  Origin Data%  Meta%  Move Log Cpy%Sync
Convert VDOdeduplication VDOIndexState
  vdo_name vg_name vwi-a-v--- 1.00t vpool0  0.00                      closed
closed
  vpool0  vg_name dwi----- <15.00g          20.03                      closed
closed
```

其他资源

- [lvmvdo\(7\) 手册页](#)
- [lvcreate\(8\) 手册页](#)

3.9. 使用虚拟数据优化器管理精简配置

可以通过配置精简配置的 VDO 卷准备以后扩展物理空间，以便解决 VDO 卷使用 100% 的条件。例如，在 **lvcreate** 操作中不使用 **-l 100%FREE** 而是使用例如 '95%FREE'，以确保稍后会根据需要进行恢复。这个步骤描述了如何解决这个问题：

- 卷耗尽空间
- 文件系统进入只读模式
- 卷报告的 ENOSPC



注意

解决 VDO 卷中高物理空间使用的最佳方法是删除未使用的文件，并使用在线丢弃这些未使用文件的块或 **fstrim** 程序。VDO 卷的物理空间只能增加到 8192 slab，对于一个默认 slab 大小为 2 GB 的 VDO 卷为 16 TB，或对于一个具有 32 GB 的 VDO 卷为 256 TB。

在以下步骤中，将 *myvg* 和 *myvdo* 分别替换为卷组和逻辑卷名称。

先决条件

1. 安装 VDO 软件。如需更多信息，请参阅[安装 VDO](#)。
2. 在您的系统中有一个有可用存储容量的 LVM 卷组。
3. 使用 **lvcreate --type vdo --name myvdo myvg -L logical-size-of-pool --virtualsize virtual-size-of-vdo** 命令的精简配置 VDO 卷。如需更多信息，请参阅[创建 LVM-VDO 卷](#)。

步骤

1. 确定精简置备 VDO 卷的最佳逻辑大小

```
# vdostats myvg-vpool0-vpool

Device          1K-blocks Used   Available Use% Space saving%
myvg-vpool0-vpool 104856576 29664088 75192488 28% 69%
```

要计算空间节省率，请使用以下公式：

```
Savings ratio = 1 / (1 - Space saving%)
```

在本例中，

- 大约有 **3.22:1** 个空间节省率（大约 80 GB）。
 - 如果对使用相同空间节省的数据写入 VDO 卷，则按比例增加数据集大小乘以 256 GB。
 - 将这个数字调整到 200 GB 时，如果出现相同的空间节省率，则会产生一个具有安全可用磁盘空间的逻辑大小。
2. 监控 VDO 卷中的空闲物理空间：

```
# vdostats myvg-vpool0-vpool
```

可定期执行这个命令，以提供对 VDO 卷使用的和空闲物理空间的监控。

3. 可选：使用可用的 `/usr/share/doc/vdo/examples/monitor/monitor_check_vdostats_physicalSpace.pl` 脚本，查看 VDO 卷上的物理空间使用量警告：

```
# /usr/share/doc/vdo/examples/monitor/monitor_check_vdostats_physicalSpace.pl myvg-vpool0-vpool
```

4. 在创建 VDO 卷时，**dmeventd** 监控服务监控 VDO 卷中物理空间的使用情况。当 VDO 卷被创建或启动时，这会被默认启用。

在监控 VDO 卷时，使用 `journalctl` 命令查看日志中的 `dmeventd` 的输出：

```
lvm[8331]: Monitoring VDO pool myvg-vpool0-vpool.
...
lvm[8331]: WARNING: VDO pool myvg-vpool0-vpool is now 84.63% full.
lvm[8331]: WARNING: VDO pool myvg-vpool0-vpool is now 91.01% full.
lvm[8331]: WARNING: VDO pool myvg-vpool0-vpool is now 97.34% full.
```

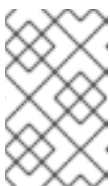
5. 修复快要没有可用物理空间的 VDO 卷。当可以在 VDO 卷中添加物理空间时，但卷空间在可以增大前已满时，可能需要临时将 I/O 返回到卷。要临时停止 I/O 到卷，请执行以下步骤，其中 VDO 卷 `myvdo` 包含挂载在 `/users/homeDir` 路径中的文件系统：

- a. 冻结文件系统：

```
# xfs_freeze -f /users/homeDir
# vgextend myvg /dev/vdc2
# lvextend -l new_size myvg/vpool0-name
# xfs_freeze -u /users/homeDir
```

- b. 卸载文件系统：

```
# umount /users/homeDir
# vgextend myvg /dev/vdc2
# lvextend -l new_size myvg/vpool0-name
# mount -o discard /dev/myvg/myvdo /users/homeDir
```



注意

卸载或释放缓存数据的文件系统将产生缓存数据的写入，这可能会填满 VDO 卷的物理空间。当为 VDO 卷上的空闲物理空间设置监控阈值时，请考虑缓存的最大缓存文件系统数据量。

6. 可以使用 `fstrim` 程序清理文件系统不再使用的块。对 VDO 卷上的挂载的文件系统执行 `fstrim` 可能会导致该卷的可用空间增加。`fstrim` 工具将丢弃到 VDO 卷，然后用于删除对之前使用的块的引用。如果这些块中有单一引用，则使用物理空间。

- a. 检查 VDO stats 以查看当前可用空间量：

```
# vdostats --human-readable myvg-vpool0-vpool

Device      Size Used Available Use% Space saving%
myvg-vpool0-vpool 100.0G 95.0G 5.0G   95% 73%
```

- b. 丢弃未使用块：

```
# fstrim /users/homeDir
```

c. 查看 VDO 卷的空闲物理空间：

```
# vdostats --human-readable myvg-vpool0-vpool
```

Device	Size	Used	Available	Use%	Space saving%
<i>myvg-vpool0-vpool</i>	100.0G	30.0G	70.0G	30%	43%

在这个示例中，在文件系统上执行 **fstrim** 后，丢弃可以返回 65G 物理空间以便在 VDO 卷中使用。



注意

丢弃较低级别的重复数据删除和压缩卷时，可能会回收物理空间，而不是丢弃更高级别的重复数据删除和压缩卷。具有高水平的重复数据删除和压缩卷可能需要进行更广泛的清理来回收物理空间，而不是只是丢弃尚未使用的块。

第 4 章 将现有 VDO 卷导入到 LVM 中

您可以将 VDO 管理器创建的 VDO 卷导入到 LVM 中。因此，您可以使用 LVM 工具将卷作为逻辑卷进行管理。



注意

导入操作是不可逆的。将现有的 VDO 卷转换为 LVM 后，只能使用 LVM 命令和 VDO 管理器来访问 VDO 数据，VDO 管理器不再控制卷。

先决条件

- 安装 VDO 软件。如需更多信息，请参阅[安装 VDO](#)。

步骤

1. 将由 VDO 管理器创建的现有 **VDO** 卷转换为逻辑卷。在以下命令中，将 *vg-name* 替换为卷组名称，将 *lv-name* 替换为逻辑卷名称，将 */dev/sdg1* 替换为 VDO 设备：

```
# lvm_import_vdo --name vg-name/lv-name /dev/sdg1

Convert VDO device "/dev/sdg1" to VDO LV "vg-name/lv-name"? [y|N]: Yes
Stopping VDO vdo-name
Converting VDO vdo-name
  Opening /dev/disk/by-id/scsi-36d094660575ece002291bd67517f677a-part1 exclusively
  Loading the VDO superblock and volume geometry
  Checking the VDO state
  Converting the UDS index
  Converting the VDO
  Conversion completed for '/dev/disk/by-id/scsi-36d094660575ece002291bd67517f677a-part1': VDO is now offset by 2097152 bytes
Physical volume "/dev/sdg1" successfully created.
Volume group "vg-name" successfully created
WARNING: Logical volume vg-name/lv-name_vpool not zeroed.
Logical volume "lv-name_vpool" created.
WARNING: Converting logical volume vg-name/lv-name_vpool to VDO pool volume WITHOUT formatting.
WARNING: Using invalid VDO pool data MAY DESTROY YOUR DATA!
Logical volume "lv-name" created.
Converted vg-name/lv-name_vpool to VDO pool volume and created virtual vg-name/lv-name VDO volume.
```

2. 可选：在 VDO LV 上创建文件系统。
3. 可选：挂载 LVM-VDO 卷。如需更多信息，请参阅[挂载 LVM-VDO 卷](#)。

验证

- 列出 LVM 设备以验证将 VDO 卷导入到 LVM 是否成功：

```
# lvs -a -o +devices
LV          VG      Attr      LSize Pool      Origin Data%  Meta%  Move Log Cpy%Sync
Convert Devices
```

```
lv-name          vg-name vwi-a-v--- 25.00g lv-name_vpool    0.00
lv-name_vpool(0)
lv-name_vpool    vg-name dwi----- <1.82t          0.31          lv-
name_vpool_vdata(0)
[lv-name_vpool_vdata] vg-name Dwi-ao---- <1.82t
/dev/sdg1(0)
```

其他资源

- [lvm_import_vdo\(8\)](#), [lvmvdo\(7\)](#) 和 [systemd.mount\(5\)](#) 手册页

第 5 章 在 LVM-VDO 卷中修剪选项

您可以使用 **discard** 选项挂载文件系统，它会告知 VDO 卷未使用的空间。另一种选择是使用 **fstrim** 应用，它是一个按需丢弃的，或 **mount -o discard** 命令用于立即丢弃。

在使用 **fstrim** 应用时，管理员需要调度和监控额外的进程，而使用 **mount -o discard** 命令则可尽可能立即恢复空间。

请注意，目前建议使用 **fstrim** 应用程序丢弃未使用的块，而不是 **discard** 挂载选项，因为此选项的性能影响可能非常严重。因此，**nodiscard** 是默认值。

5.1. 在 VDO 中启用丢弃挂载选项

此流程在 VDO 卷中启用 **discard** 选项。

先决条件

- 您的系统中有 LVM-VDO 卷。

步骤

- 在卷中启用 **discard** :

```
# mount -o discard /dev/vg-name/vdo-name mount-point
```

其他资源

- **XFS(5)**、**mount(8)** 和 **lvmvdo(7)** man page

5.2. 设置定期 TRIM 操作

这个过程在您的系统中启用调度的 TRIM 操作。

先决条件

- 您的系统中有 LVM-VDO 卷。

步骤

- 启用并启动计时器 :

```
# systemctl enable --now fstrim.timer
```

验证

- 验证计时器是否已启用 :

```
# systemctl list-timers fstrim.timer
```

例 5.1. 验证过程的可能输出

```
# systemctl list-timers fstrim.timer
NEXT          LEFT    LAST PASSED UNIT    ACTIVATES
Mon 2021-05-10 00:00:00 EDT 5 days left n/a  n/a    fstrim.timer fstrim.service
```



注意

您没有对 VDO 卷的引用，因为 **fstrim.timer** 在所有挂载的文件系统中运行。

其他资源

- **fstrim(8)** man page

第 6 章 优化 VDO 性能

VDO 内核驱动程序通过使用多个线程来加快任务的速度。它不是一个线程对 I/O 请求执行一切，而是将工作成分配给不同线程的较小的部分。这些线程在处理请求时相互通信。这样，一个线程可以在不永久锁定和解锁的情况下处理共享数据。

当一个线程完成一个任务时，VDO 已经有另一个任务可供它。这会保持线程忙碌，并减少了切换任务所花费的时间。VDO 还对较慢的任务使用单独的线程，如将 I/O 操作添加到队列中，或处理消息到重复数据删除索引中。

6.1. VDO 线程类型

VDO 使用各种线程类型来处理特定的操作：

逻辑区线程(kvdo:logQ)

维护提供给 VDO 设备用户和底层存储系统中的物理块号(PBN)的逻辑块号(LBN)之间的映射。它们还阻止并发写入同一块。逻辑线程在读取和写入操作过程中处于活动状态。处理通常平均分布，但特定的访问模式偶尔可能集中在一个线程中。例如，在特定块映射页面中频繁访问 LBN 可能会导致一个逻辑线程处理所有这些操作。

物理区线程(kvdo:physQ)

在写入操作过程中处理数据块分配和引用计数。

I/O 提交线程(kvdo:bioQ)

处理从 VDO 将块 I/O (**bio**)操作传送到存储系统。它们处理来自其他 VDO 线程的 I/O 请求，并将它们传递给底层设备驱动程序。这些线程与设备相关的数据结构交互，为设备驱动程序内核线程创建请求，并防止 I/O 请求因为完整设备请求队列而被阻止时出现延迟。

CPU-processing threads (kvdo:cpuQ)

处理不阻止或需要独占访问其他线程类型的 CPU 密集型任务。这些任务包括计算哈希值和压缩数据块。

I/O 确认线程(kvdo:ackQ)

表示 I/O 请求到更高级别的组件的完成情况，如内核页面缓存或应用程序线程执行直接 I/O。其 CPU 使用量和对内存争用的影响会受到内核级别的代码的影响。

hash zone threads (kvdo:hashQ)

与匹配哈希协调 I/O 请求，以处理潜在的重复数据删除任务。虽然它们创建和管理 deduplication 请求，但它们不会执行大量计算。单个哈希区线程通常就足够了。

去除重复数据线程(kvdo:dedupeQ)

处理 I/O 请求并与重复数据删除索引通信。这个工作是在单独的线程上执行的，以防止阻塞。如果索引没有快速响应，它也具有跳过 deduplication 的超时机制。每个 VDO 设备只有一个 deduplication 线程。

日志线程(kvdo:journalQ)

更新恢复日志并调度日志块以进行写入。此任务不能划分多个线程。每个 VDO 设备只有一个日志线程。

packer thread (kvdo:packerQ)

启用压缩时，在写入操作过程中工作。它从 CPU 线程收集压缩数据块，以减少浪费空间。每个 VDO 设备只有一个 packer 线程。

6.2. 识别性能瓶颈

识别 VDO 性能中的瓶颈对于优化系统效率至关重要。您可以采取的一个主要步骤是确定瓶颈是 CPU、内存还是后备存储的速度。在找出最慢的组件后，您可以制定策略以增强性能。

为确保低性能的根本原因不是硬件问题，请在存储堆栈中运行带有和不使用 VDO 的测试。

VDO 中的 **journalQ** 线程是自然瓶颈，特别是在 VDO 卷处理写入操作时。请注意，另一个线程类型使用率高于 **journalQ** 线程，您可以通过添加更多类型的线程来修复此问题。

6.2.1. 使用 top 分析 VDO 性能

您可以使用 **top** 实用程序检查 VDO 线程的性能。

步骤

1. 显示单个线程：

```
$ top -H
```



注意

top 等工具无法区分 CPU 周期和因缓存或内存延迟而停止的周期。这些工具将缓存争用解释，并将内存访问速度减速为实际工作。在节点间移动线程可能会如减少 CPU 使用率，同时增加每秒操作。

2. 按 **f** 键显示字段管理器。
3. 使用 (**criu**) 键导航到 **P = Last Used Cpu (SMP)** 字段。
4. 按空格栏选择 **P = Last Used Cpu (SMP)** 字段。
5. 按 **q** 键关闭字段管理器。**top** 工具现在显示各个内核的 CPU 负载，并指示最近使用的每个进程或线程的 CPU。您可以通过按 **1** 切换到每个 CPU 统计信息。

其他资源

- [top \(1\) 手册页](#)
- [解释 top 结果](#)

6.2.2. 解释 top 结果

在分析 VDO 线程的性能时，请使用下表来解释 **top** 工具的结果。

表 6.1. 解释 top 结果

值	描述	建议
线程或 CPU 使用率超过 70%	线程或 CPU 过载。高使用量可能会导致 VDO 线程调度到没有实际工作的 CPU 上。这可能是由于硬件中断、内存冲突或资源竞争造成的问题。	增加运行此内核的类型的线程数量。
低 %id 和 %wa 值	内核正在主动处理任务。	不需要操作。

值	描述	建议
低 %hi 值	核心正在执行标准处理工作。	添加更多内核以提高性能。避免 NUMA 冲突。
<ul style="list-style-type: none"> ● 高 %hi 值 [a] ● 只有一个线程分配给内核 ● %ID 为零 ● %wa 值为零 	内核被过度使用。	将内核线程和设备中断处理重新分配给不同的内核。
<ul style="list-style-type: none"> ● kvdo:bioQ 线程通常处于 D 状态。 	VDO 始终使存储系统忙于 I/O 请求。 [b]	如果 CPU 使用率非常低，则减少 I/O 提交线程数量。
kvdo:bioQ 线程通常处于 S 状态。	VDO 具有比需要更多的 kvdo:bioQ 线程。	减少 kvdo:bioQ 线程的数量。
每个 I/O 请求高 CPU 使用率。	每个 I/O 请求的 CPU 使用率随着更多线程增加而增加。	检查 CPU、内存或锁定争用。
<p>[a] 超过百分比</p> <p>[b] 如果存储系统可以处理多个请求，或者请求处理效率效率，则这非常有用。</p>		

6.2.3. 使用 perf 分析 VDO 性能

您可以使用 **perf** 实用程序检查 VDO 的 CPU 性能。

先决条件

- 已安装 **perf** 软件包。

步骤

1. 显示性能配置集：

```
# perf top
```

2. 通过解释 **perf** 结果来分析 CPU 性能：

表 6.2. 解释 **perf** 结果

值	描述	建议
kvdo:bioQ 线程花费过量周期，从而获得 spin 锁定	VDO 下的设备驱动程序中可能会出现太多竞争	减少 kvdo:bioQ 线程的数量
高 CPU 用量	NUMA 节点之间争用。 如果处理器支持，请检查 stalled-cycles-backend 、 cache-misses 和 node-load-misses 等计数器。高未命中率可能会导致停滞，在其他工具中调整高 CPU 使用量，指示可能争用。	为中断处理程序实施 VDO 内核线程或 IRQ 关联性的 CPU 关联性，以限制单个节点的处理工作。

其他资源

- **perf-top (1)** man page

6.2.4. 使用 sar 分析 VDO 性能

您可以使用 **sar** 实用程序在 VDO 性能上创建定期报告。



注意

并非所有块设备驱动程序都可以提供 **sar** 实用程序所需的数据。例如：MD RAID 等设备不会报告 **%util** 值。

先决条件

- 安装 **sysstat** 工具：

```
# yum install sysstat
```

步骤

1. 以 1 秒的间隔显示磁盘 I/O 统计信息：

```
$ sar -d 1
```

2. 通过解释 **sar** 结果分析 VDO 性能：

表 6.3. 解释 **sar** 结果

值	描述	建议
<ul style="list-style-type: none"> 底层存储设备的 %util 值在 100% 下良好。 VDO 忙碌为 100%。 bioQ 线程使用大量 CPU 时间。 	对于快速设备，VDO 太少的 bioQ 线程。	<p>添加更多 bioQ 线程。</p> <p>请注意，由于锁争用，某些存储驱动程序可能会在添加 bioQ 线程时减慢。</p>

其他资源

- [sar \(1\) 手册页](#)

6.3. 重新分发 VDO 线程

在处理请求时，VDO 将各种线程池用于不同的任务。最佳性能取决于在每个池中设置正确的线程数量，这因可用的存储、CPU 资源以及工作负载的类型而异。您可以将 VDO 分散到多个线程中，以提高 VDO 性能。

VDO 旨在通过并行性最大化性能。您可以根据可用 CPU 资源等因素和瓶颈的根本原因，为瓶颈任务分配更多线程来提高它。高线程利用率(bove 70-80%)可能会导致延迟。因此，在这种情况下增加线程数可能会有所帮助。但是，过量线程可能会妨碍性能，并产生额外的成本。

要获得最佳性能，请执行以下操作：

- 测试具有各种预期工作负载的 VDO 来评估和优化其性能。
- 为使用率超过 50% 的池增加线程数。
- 如果总体利用率大于 50%，增加 VDO 可用的内核数，即使单个线程利用率较低。

6.3.1. 在 NUMA 节点之间对 VDO 线程进行分组

访问 NUMA 节点的内存比本地内存访问要慢。在内核共享节点中的最后一个缓存的 Intel 处理器上，当数据在节点间共享时，缓存问题会比在单一节点间共享时更显著。虽然很多 VDO 内核线程管理专用数据结构，但它们通常会交换有关 I/O 请求的消息。VDO 线程分散到多个节点上，或者调度程序在节点间重新分配线程可能会导致竞争，这是竞争同一资源的多个节点。

您可以通过在同一 NUMA 节点上对某些线程进行分组来提高 VDO 性能。

在一个 NUMA 节点上将相关的线程分组到一起

- I/O 确认(ackQ)线程
- 高级别 I/O 提交线程：
 - 用户模式线程处理直接 I/O
 - 内核页面缓存清除线程

优化设备访问

- 如果设备访问时间因 NUMA 节点而异，请在最接近存储设备控制器的节点上运行 **bioQ** 线程

最小化竞争

- 在与 **logQ** 或 **physQ** 线程相同的节点上运行 I/O 提交和存储设备中断处理。
- 在同一节点上运行其他与 VDO 相关的工作。
- 如果一个节点无法处理所有 VDO 工作，在将线程移到其他节点时请考虑内存争用。例如，将中断处理和 **bioQ** 线程的设备移到另一个节点。

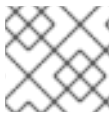
6.3.2. 配置 CPU 关联性

如果调整 VDO 线程的 CPU 关联性，您可以提高某些存储设备驱动程序的 VDO 性能。

当存储设备驱动程序的中断(IRQ)处理程序具有大量工作，并且驱动程序不使用线程 IRQ 处理程序时，它可能会限制系统调度程序优化 VDO 性能的能力。

要获得最佳性能，请执行以下操作：

- 如果内核过载，则指定特定的内核到 IRQ 处理并调整 VDO 线程关联性。如果 **%hi** 值超过其他内核的百分比，则内核会超载。
- 避免在忙碌的 IRQ 内核上运行单例 VDO 线程，如 **kvdo:journalQ** 线程。
- 只有在单个 CPU 使用较高时，才会将其他线程类型保留为忙碌的 IRQ。



注意

配置在系统重启后不会保留。

步骤

- 设置 CPU 关联性：

```
# taskset -c <cpu-numbers> -p <process-id>
```

将 **<cpu-numbers>** 替换为您要为其分配进程的 CPU 号的逗号分隔列表。将 **<process-id>** 替换为您要设置 CPU 关联性的正在运行的进程 ID。

例 6.1. 在 CPU 内核 1 和 2 上为 kvdo 进程设置 CPU 关联性

```
# for pid in `ps -eo pid,comm | grep kvdo | awk '{ print $1 }'`
do
    taskset -c "1,2" -p $pid
done
```

验证

- 显示关联性集：

```
# taskset -p <cpu-numbers> -p <process-id>
```


将 `<cpu-numbers >` 替换为您要为其分配进程的 CPU 号的逗号分隔列表。将 `<process-id >` 替换为您要设置 CPU 关联性的正在运行的进程 ID。

其他资源

- [taskset \(1\) 手册页](#)

6.4. 增加块映射缓存大小以增强性能

您可以通过增加 VDO 卷的缓存大小来提高读取和写入性能。

如果您扩展了读取和写入延迟，或者从存储中读取的大量数据与应用程序要求不匹配，您可能需要调整缓存大小。



警告

当您增加块映射缓存时，缓存会使用您指定的内存量，再加上额外的 15% 内存。较大的缓存大小使用更多 RAM，并影响整体系统稳定性。

以下示例演示了如何将系统中的缓存大小从 128Mb 更改为 640Mb。

步骤

1. 检查 VDO 卷的当前缓存大小：

```
# lvs -o vdo_block_map_cache_size
VDOBlockMapCacheSize
      128.00m
      128.00m
```

2. 取消激活 VDO 卷：

```
# lvchange -an vg_name/vdo_volume
```

3. 更改 VDO 设置：

```
# lvchange --vdosettings "block_map_cache_size_mb=640" vg_name/vdo_volume
```

以 MB 为单位，将 **640** 替换为您的新缓存大小。



注意

缓存大小必须是 4096 的倍数，范围为 128MB 到 16TB，每个逻辑线程至少有 16MB。更改在下次启动 VDO 设备时生效。已在运行的设备不受影响。

4. 激活 VDO 卷：

```
# lvchange -ay vg_name/vdo_volume
```

验证

- 检查当前的 VDO 卷配置：

```
# lvs -o vdo_block_map_cache_size vg_name/vdo_volume
VDOBlockMapCacheSize
        640.00m
```

其他资源

- [lvchange \(8\) 手册页](#)

6.5. 加快丢弃操作

VDO 为系统中的所有 VDO 设备设置允许的最大 DISCARD (TRIM)扇区。默认大小为 8 个扇区，对应于一个 4KiB 块。增加 DISCARD 大小可能会显著提高丢弃操作的速度。但是，在改进丢弃性能和维护其他写入操作速度之间有一个利弊。

最佳 DISCARD 大小因存储堆栈而异。非常大且非常小的 DISCARD 扇区可能会降低性能。使用不同的值进行试验，以发现一个提供满意的结果。

对于存储本地文件系统的 VDO 卷，最好使用 DISCARD 大小 8 个扇区，这是默认设置。对于充当 SCSI 目标的 VDO 卷，中等大的 DISCARD 大小（如 2048 扇区（对应于 1MB 丢弃）可以正常工作。建议最大 DISCARD 大小不超过 10240 扇区，它转换为 5MB 丢弃。在选择大小时，请确保它是 8 的倍数，因为它们小于 8 个扇区，VDO 可能无法有效地处理丢弃。

步骤

1. 为 DISCARD 扇区设置新最大大小：

```
# echo <number-of-sectors> > /sys/kvdo/max_discard_sectors
```

将 **<number-of-sectors>** 替换为扇区数。此设置会一直保留，直到重启为止。

2. 可选：要在重启后对 DISCARD 扇区进行持久性更改，请创建一个自定义 **systemd** 服务：

- a. 使用以下内容创建一个新的 **/etc/systemd/system/max_discard_sectors.service** 文件：

```
[Unit]
Description=Set maximum DISCARD sector
[Service]
ExecStart=/usr/bin/echo <number-of-sectors> > /sys/kvdo/max_discard_sectors

[Install]
WantedBy=multi-user.target
```

将 **<number-of-sectors>** 替换为扇区数。

- b. 保存文件并退出。
- c. 重新载入服务文件：

```
# systemctl daemon-reload
```

d. 启用新服务：

```
# systemctl enable max_discard_sectors.service
```

验证

- 可选：如果您使 scaling governor 持久更改，请检查是否启用了 **max_discard_sectors.service**：

```
# systemctl is-enabled max_discard_sectors.service
```

6.6. 优化 CPU 频率扩展

默认情况下，RHEL 使用 CPU 频率扩展来省电，并在 CPU 负载不足时减少 heat。为了在节能上优先选择性能，您可以将 CPU 配置为以最大时钟速度运行。这样可确保 CPU 可以处理最大效率的数据重复数据删除和压缩过程。通过以最高频率运行 CPU，可以更快地执行资源密集型操作，从而提高 VDO 的整体性能，以防数据减少和存储优化。



警告

为更高的性能调整 CPU 频率扩展可以提高功耗和热生成。在不仲裁的系统中，这可能导致过度使用，并可能导致热流，这限制了性能提升。

步骤

1. 显示可用的 CPU 调控器：

```
$ cpupower frequency-info -g
```

2. 更改 scaling governor 以优先选择性能：

```
# cpupower frequency-set -g performance
```

此设置会一直保留，直到重启为止。

3. 可选：要在重启后扩展 governor 中的持久更改，请创建一个自定义 **systemd** 服务：

- a. 使用以下内容创建一个新的 **/etc/systemd/system/cpufreq.service** 文件：

```
[Unit]
Description=Set CPU scaling governor to performance

[Service]
ExecStart=/usr/bin/cpupower frequency-set -g performance

[Install]
WantedBy=multi-user.target
```

b. 保存文件并退出。

c. 重新载入服务文件：

```
# systemctl daemon-reload
```

d. 启用新服务：

```
# systemctl enable cpufreq.service
```

验证

- 显示当前使用的 CPU 频率策略：

```
$ cpupower frequency-info -p
```

- 可选：如果您使 scaling governor 持久更改，请检查是否启用了 **cpufreq.service**：

```
# systemctl is-enabled cpufreq.service
```