Red Hat Enterprise Linux 8

ストレージの重複排除および圧縮

RHEL 8 で VDO を使用してストレージの容量を最適化

Last Updated: 2024-06-26

Red Hat Enterprise Linux 8 ストレージの重複排除および圧縮

RHEL 8 で VDO を使用してストレージの容量を最適化

法律上の通知

概要

本書は、Red Hat Enterprise Linux 8 で、VDO (Virtual Data Optimizer) を使用して、重複を排除して圧縮したストレージプールを管理する方法を説明します。

目次

RED HAT ドキュメントへのフィードバック (英語のみ)

Red Hat ドキュメントに関するご意見やご感想をお寄せください。また、改善点があればお知らせください。

Jira からのフィードバック送信 (アカウントが必要)

1. Jira の Web サイトにログインします。
2. 上部のナビゲーションバーで **Create** をクリックします。
3. **Summary** フィールドにわかりやすいタイトルを入力します。
4. **Description** フィールドに、ドキュメントの改善に関するご意見を記入してください。ドキュメントの該当部分へのリンクも追加してください。
5. ダイアログの下部にある **Create** をクリックします。

第1章 VDO のデプロイメント

システム管理者は、VDO を使用してストレージプールの重複を排除して、圧縮できます。

1.1. VDO の概要

VDO (Virtual Data Optimizer) は、重複排除、圧縮、およびシンプロビジョニングの形で、Linux でインラインのデータ削減を行います。VDO ボリュームを設定する場合は、VDO ボリュームを構築するブロックデバイスと、作成する論理ストレージのサイズを指定します。

- アクティブな仮想マシンまたはコンテナーをホストする場合、Red Hat は、物理と論理の割合を 1 対 10 にすることを推奨します。つまり、物理ストレージを 1 TB にした場合は、論理ストレージを 10 TB にします。
- Ceph が提供するタイプなどのオブジェクトストレージの場合、Red Hat は、物理と論理の割合を 1 対 3 にすることを推奨します。つまり、物理ストレージを 1 TB にした場合は、論理ストレージを 3 TB にします。

いずれの場合も、VDO が作成する論理デバイスにファイルシステムを置くだけで、直接使用することも、分散クラウドストレージアーキテクチャーの一部として使用することもできます。

VDO はシンプロビジョニングされているため、ファイルシステムとアプリケーションは、使用中の論理領域だけを認識し、実際に利用可能な物理領域は認識しません。スクリプトを使用して、実際に利用可能な領域を監視し、使用量がしきい値を超えた場合 (たとえば、VDO ボリュームの使用量が 80% になった場合) にアラートを生成します。

関連情報

- 物理領域の監視に関する詳細は、「VDO ボリューム上の空き領域の管理」 を参照してください。

1.2. VDO デプロイメントシナリオ

VDO は、様々な方法でデプロイして、以下に対して、重複排除したストレージを提供できます。

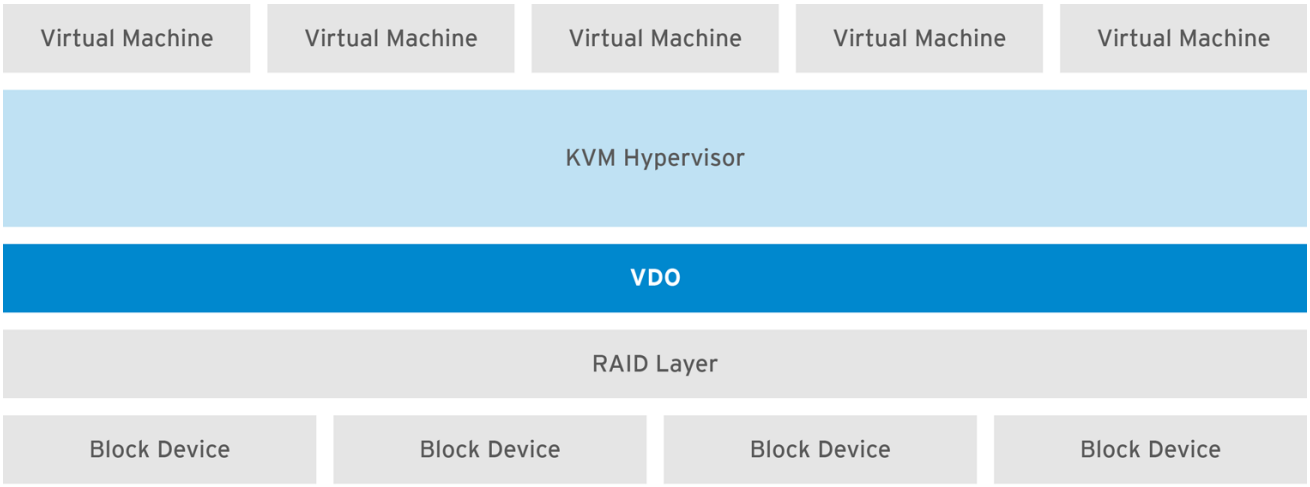
- ブロックおよびファイルアクセスの両方
- ローカルストレージおよびリモートストレージの両方

VDO は、標準の Linux ブロックデバイスとして重複排除したストレージを公開するため、そのストレージを標準ファイルシステム、iSCSI および FC のターゲットドライバー、または統合ストレージとして使用できます。

注記

現在、Ceph RADOS ブロックデバイス (RBD) 上での VDO ボリュームのデプロイがサポートされています。ただし、VDO ボリューム上での Red Hat Ceph Storage クラスターコンポーネントのデプロイは現在サポートされていません。

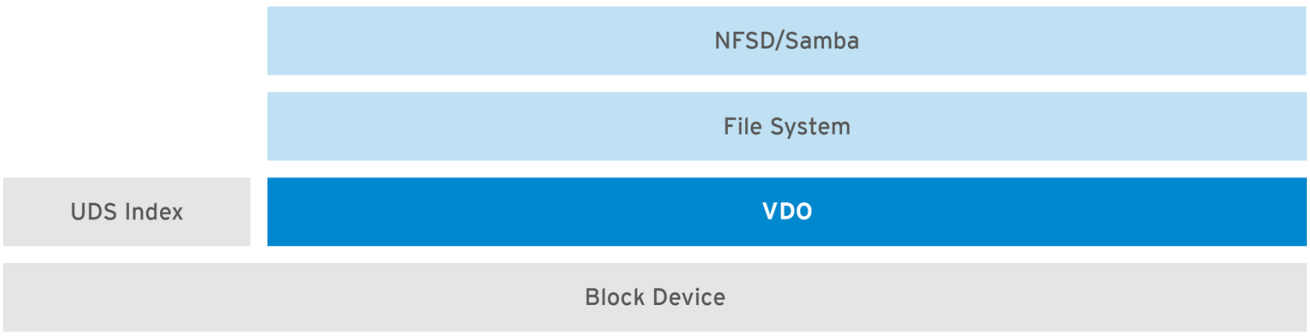
KVM
DAS (Direct Attached Storage) を使用して設定した KVM サーバーに VDO をデプロイできます。



RHEL_462492_1117

ファイルシステム

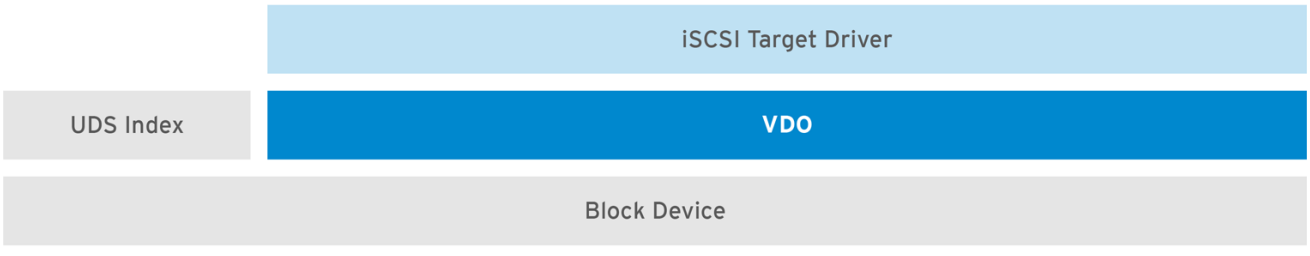
VDO にファイルシステムを作成して、NFS サーバーまたは Samba で、NFS ユーザーまたは CIFS ユーザーに公開します。



RHEL_466924_0218

iSCSI への VDO の配置

VDO ストレージターゲット全体を、iSCSI ターゲットとしてリモート iSCSI イニシエーターにエクスポートできます。



RHEL_466924_0218

iSCSI で VDO ボリュームを作成する場合は、VDO ボリュームを iSCSI レイヤーの上または下に配置できます。考慮すべき点はたくさんありますが、ここでは、環境に最適な方法を選択するのに役立つガイドラインをいくつか示します。

VDO ボリュームを iSCSI レイヤーの下の iSCSI サーバー (ターゲット) に配置する場合:

- VDO ボリュームは、他の iSCSI LUN と同様に、イニシエーターに対して透過的です。シンプロビジョニングとスペースの節約をクライアントから隠すことで、LUN の外観の監視と保守が容易になります。

- VDO メタデータの読み取りまたは書き込みがないため、ネットワークトラフィックが減少し、重複排除アドバイスの読み取り検証がネットワーク全体で発生しません。
- iSCSI ターゲットで使用されているメモリーと CPU リソースにより、パフォーマンスが向上する可能性があります。たとえば、iSCSI ターゲットでボリュームの削減が行われているため、ハイパーバイザーの数を増やすことができます。
- クライアントがイニシエーターに暗号化を実装し、ターゲットの下に VDO ボリュームがある場合、スペースの節約は実現しません。

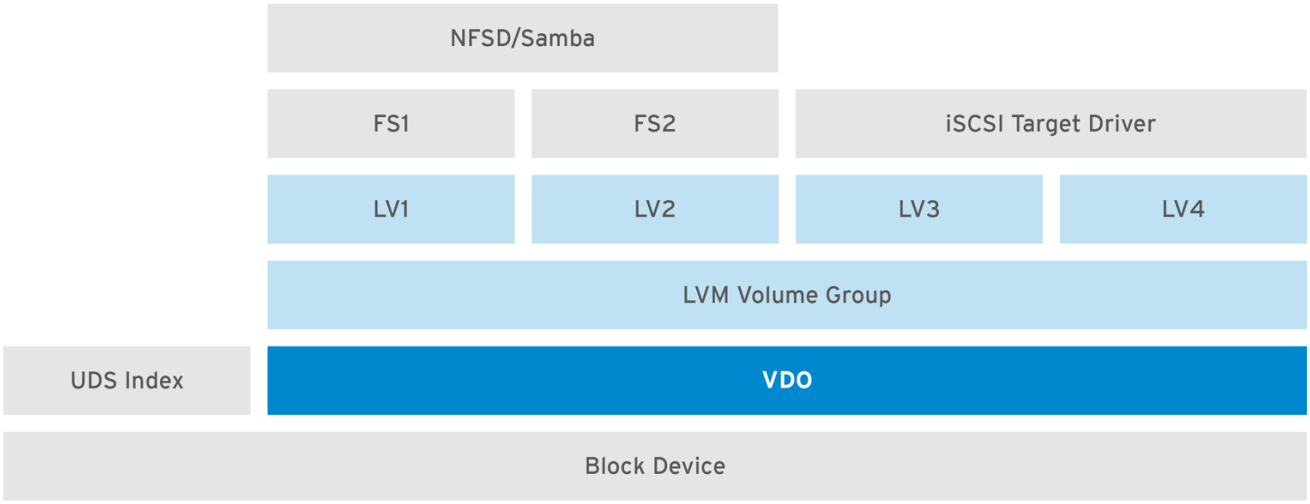
VDO ボリュームを iSCSI レイヤーの上の iSCSI クライアント (イニシエーター) に配置する場合:

- 高いスペース節約率を達成する場合、ASYNC モードでネットワーク全体のネットワークトラフィックが低下する可能性があります。
- スペースの節約を直接表示および制御し、使用状況を監視できます。
- たとえば、**dm-crypt** を使用してデータを暗号化する場合は、暗号の上に VDO を実装して、スペース効率を利用できます。

LVM

より機能豊富なシステムでは、LVM を使用して、重複排除した同じストレージプールですべて対応している複数の論理ユニット番号 (LUN) を提供できます。

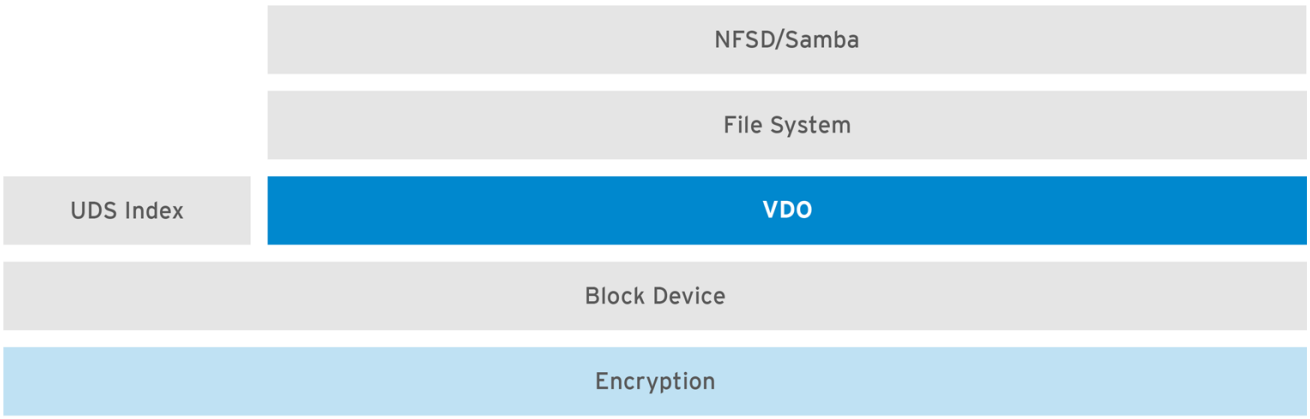
以下の図は、VDO ターゲットが物理ボリュームとして登録されるため、LVM で管理できます。複数の論理ボリューム (**LV1** から **LV4**) が、重複排除したストレージプールから作成されます。これにより、VDO は、基となる重複排除したストレージプールへのマルチプロトコル統合ブロックまたはファイルアクセスに対応できます。



重複排除した統合ストレージ設計により、複数のファイルシステムが、LVM ツールを介して同じ重複排除ドメインを共同で使用できます。また、ファイルシステムは、LVM スナップショット、コピーオンライト、縮小機能、拡大機能、および VDO にある全機能を利用できます。

暗号化

DM Crypt などのデバイスマッパー (DM) メカニズムは VDO と互換性があります。VDO ボリュームの暗号化により、データセキュリティーと、VDO にある全ファイルシステムが重複排除されるようになります。



重要

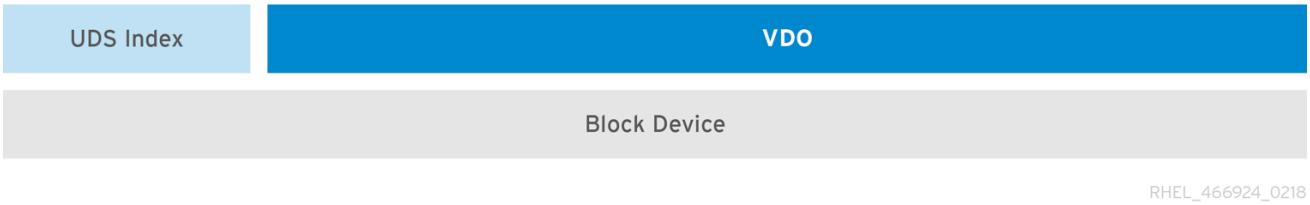
VDO で暗号化層を適用すると、データの重複排除が行われてもほとんど行われません。暗号化により、VDO が重複を排除する前に、重複ブロックを変更します。

常に VDO の下に暗号化層を配置します。

1.3. VDO ボリュームのコンポーネント

VDO は、バッキングストアとしてブロックデバイスを使用します。これは、複数のディスク、パーティション、またはフラットファイルで設定される物理ストレージの集約を含めることができます。ストレージ管理ツールが VDO ボリュームを作成すると、VDO は、UDS インデックスおよび VDO ボリュームのボリューム領域を予約します。UDS インデックスと VDO ボリュームは対話して、重複排除したブロックストレージを提供します。

図1.1 VDO ディスク組織



VDO ソリューションは、以下のコンポーネントで設定されます。

kvdo

Linux Device Mapper 層に読み込まれるカーネルモジュールは、重複排除され、圧縮され、シンプロビジョニングされたブロックストレージボリュームを提供します。
kvdo モジュールはブロックデバイスを公開します。ブロックストレージ用にこのブロックデバイスに直接アクセスするか、XFS や ext4 などの Linux ファイルシステムを介して提示することができます。

kvdo が VDO ボリュームからデータ論理ブロックを読み取る要求を受信すると、要求された論理ブロックを基礎となる物理ブロックにマッピングし、要求したデータを読み取り、返します。

kvdo が VDO ボリュームにデータブロックを書き込む要求を受信すると、まず要求が DISCARD または TRIM のものであるか、またはデータが一貫してゼロかどうかを確認します。これらの条件のいずれかが true の場合、**kvdo** はブロックマップを更新し、リクエストを承認します。そうでない

場合は、VDO はデータを処理して最適化します。

uds

ボリューム上の Universal Deduplication Service (UDS) インデックスと通信し、データの重複を分析するカーネルモジュール。新しい各データについて、その部分が保存してあるデータ内容と同一であるかどうかを UDS が素早く判断します。インデックスが一致すると、ストレージシステムは、同じ情報を複数格納しないように、既存の項目を内部的に参照できます。
UDS インデックスは、**uds** カーネルモジュールとしてカーネル内で実行します。

コマンドラインツール

最適化されたストレージの設定および管理

1.4. VDO ボリュームの物理サイズおよび論理サイズ

VDO は、物理サイズ、利用可能な物理サイズ、および論理サイズを次の方法で利用します。

物理サイズ

これは、基礎となるブロックデバイスと同じサイズです。VDO は、以下の目的でこのストレージを使用します。

- 重複排除および圧縮される可能性があるユーザーデータ
- UDS インデックスなどの VDO メタデータ

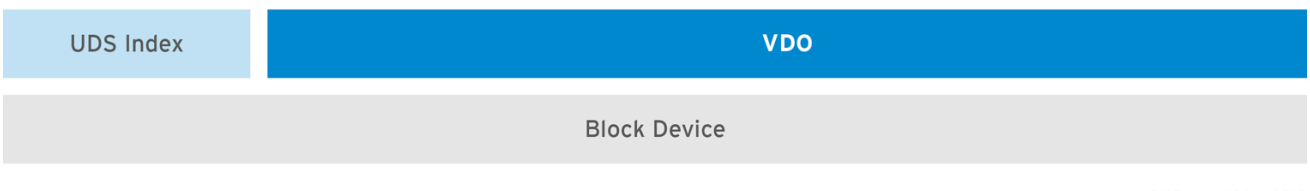
利用可能な物理サイズ

これは、VDO がユーザーデータに使用できる物理サイズの一部です。
これは、メタデータのサイズを引いた物理サイズと同等で、指定のスラブサイズでボリュームをスラブに分割した後の残りを引いたものと同じです。

論理サイズ

これは、VDO ボリュームがアプリケーションに提示するプロビジョニングされたサイズです。通常、これは利用可能な物理サイズよりも大きくなります。**--vdoLogicalSize** オプションを指定しないと、論理ボリュームのプロビジョニングが **1:1** の比率にプロビジョニングされます。たとえば、VDO ボリュームが 20 GB ブロックデバイスの上に置かれている場合は、2.5 GB が UDS インデックス用に予約されます (デフォルトのインデックスサイズが使用される場合)。残りの 17.5 GB は、VDO メタデータおよびユーザーデータに提供されます。そのため、消費する利用可能なストレージは 17.5 GB を超えません。実際の VDO ボリュームを設定するメタデータにより、これよりも少なくなる可能性があります。
VDO は現在、絶対最大論理サイズ 4PB の物理ボリュームの最大 254 倍の論理サイズに対応します。

図1.2 VDO ディスク組織



この図では、VDO で重複排除したストレージターゲットがブロックデバイス上に完全に配置されています。つまり、VDO ボリュームの物理サイズは、基礎となるブロックデバイスと同じサイズになります。

関連情報

- さまざまなサイズのブロックデバイスに必要なストレージ VDO メタデータのサイズは、「物理サイズ別の VDO 要件の例」を参照してください。

1.5. VDO のスラブサイズ

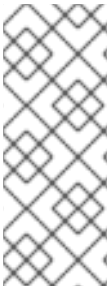
VDO ボリュームの物理ストレージは、複数のスラブに分割されます。各スラブは、物理領域における連続した領域です。特定のボリュームのスラブはすべて同じサイズで、128 MB の 2 のべき乗倍のサイズ (最大 32 GB) になります。

小規模なテストシステムで VDO を評価しやすくするため、デフォルトのスラブサイズは 2 GB です。1 つの VDO ボリュームには、最大 8192 個のスラブを含めることができます。したがって、デフォルト設定の 2 GB のスラブを使用する場合、許可される物理ストレージは最大 16 TB です。32 GB のスラブを使用する場合、許可される物理ストレージは最大 256 TB です。VDO は常に少なくとも 1 つのスラブ全体をメタデータ用に予約します。そのため、この予約されたスラブをユーザーデータの保存に使用することはできません。

スラブサイズは、VDO ボリュームのパフォーマンスには影響しません。

表1.1 物理ボリュームサイズ別の推奨 VDO スラブサイズ

物理ボリュームのサイズ	推奨されるスラブサイズ
10 - 99 GB	1 GB
100 GB - 1 TB	2 GB
2 - 256 TB	32 GB



注記

VDO ボリュームの最小ディスク使用量は、デフォルト設定のスラブサイズ 2 GB、dense インデックス 0.25 を使用した場合、約 4.7 GB が必要です。これにより、0% の重複排除または圧縮で書き込むための 2 GB 弱の物理データが提供されます。

ここでの最小のディスク使用量は、デフォルトのスラブサイズと dense インデックスの合計です。

lvcreate コマンドに **--config 'allocation/vdo_slab_size_mb=size-in-megabytes'** オプションを指定すると、スラブサイズを制御できます。

1.6. VDO 要件

VDO は、配置とシステムリソースに特定の要件があります。

1.6.1. VDO メモリー要件

各 VDO ボリュームには、2 つの異なるメモリー要件があります。

VDO モジュール

VDO には、固定メモリー 38 MB と変動用に容量を確保する必要があります。

- 設定済みのブロックマップキャッシュサイズ 1 MB ごとに 1.15 MB のメモリー。ブロックマップキャッシュには、少なくとも 150 MB のメモリーが必要です。
- 1 TB の論理領域ごとに 1.6 MB のメモリー。
- ボリュームが管理する物理ストレージの 1 TB ごとに 268 MB のメモリー。

UDS インデックス

Universal Deduplication Service (UDS) には、最低 250 MB のメモリーが必要です。このメモリー量は、重複排除が使用するデフォルトの容量です。この値は、インデックスに必要なストレージ容量にも影響するため、VDO ボリュームをフォーマットするときに設定できます。
UDS インデックスに必要なメモリーは、インデックスタイプと、重複排除ウィンドウに必要なサイズで決定されます。

インデックスタイプ	重複排除ウィンドウ	備考
Dense	RAM 1 GB あたり 1 TB	通常、最大 4 TB の物理ストレージには、1 GB の dense インデックスで十分です。
Sparse	RAM 1 GB あたり 10 TB	通常、最大 40 TB の物理ストレージには、1 GB の sparse インデックスで十分です。

注記

VDO ボリュームの最小ディスク使用量は、デフォルト設定のスラブサイズ 2 GB、dense インデックス 0.25 を使用した場合、約 4.7 GB が必要です。これにより、0% の重複排除または圧縮で書き込むための 2 GB 弱の物理データが提供されます。

ここでの最小のディスク使用量は、デフォルトのスラブサイズと dense インデックスの合計です。

VDO で推奨されるモードは、UDS の sparse インデックス機能です。この機能は、データの一時的な局所性に依存し、メモリー内で最も関連性の高いインデックスエントリーのみを保持しようとします。sparse インデックスでは、UDS は、同じ量のメモリーを使用しながら、dense を使用したときの 10 倍以上長い重複排除ウィンドウを維持できます。

sparse インデックスを使用すると対象範囲が広くなりますが、dense インデックスの方が提供する重複排除アドバイスが多くなります。ほとんどのワークロードでは、メモリー量が同じであれば、dense インデックスと sparse インデックスの重複排除率の差はごくわずかです。

関連情報

- 物理サイズ別の VDO 要件の例

1.6.2. VDO ストレージの領域要件

VDO ボリュームを設定して、最大 256 TB の物理ストレージを使用するように設定できます。データを格納するのに使用できるのは、物理ストレージの一部のみです。このセクションでは、VDO に管理されるボリュームで使用可能なサイズを特定するための計算方法を説明します。

VDO では、2 種類の VDO メタデータと UDS インデックスにストレージが必要です。

- 最初のタイプの VDO メタデータは、4 GB の **物理ストレージ** ごとに約 1 MB を使用し、スラブごとにさらに 1 MB を使用します。
- 2 番目のタイプの VDO メタデータは、**論理ストレージ** 1 GB ごとに約 1.25 MB を消費し、最も近いスラブに切り上げられます。
- UDS インデックスに必要なストレージの容量は、インデックスの種類と、インデックスに割り当てられている RAM の容量によって異なります。RAM 1 GB ごとに、dense の UDS インデックスはストレージを 17 GB 使用し、sparse の UDS インデックスはストレージを 170 GB 使用します。

関連情報

- 「物理サイズ別の VDO 要件の例」
- 「VDO のスラブサイズ」

1.6.3. ストレージスタックの VDO の 配置

配置要件に合わせて、Virtual Data Optimizer (VDO) の上または下にストレージ層を配置します。

VDO ボリュームは、シンプロビジョニングしたブロックデバイスです。後で拡張できるストレージ層の上にボリュームを配置することで、物理スペースの不足を防ぐことができます。このような拡張可能なストレージの例として、論理ボリュームマネージャー (LVM) ボリューム、または Multiple Device Redundant Array of Inexpensive/Independent Disks (MD RAID) アレイがあります。

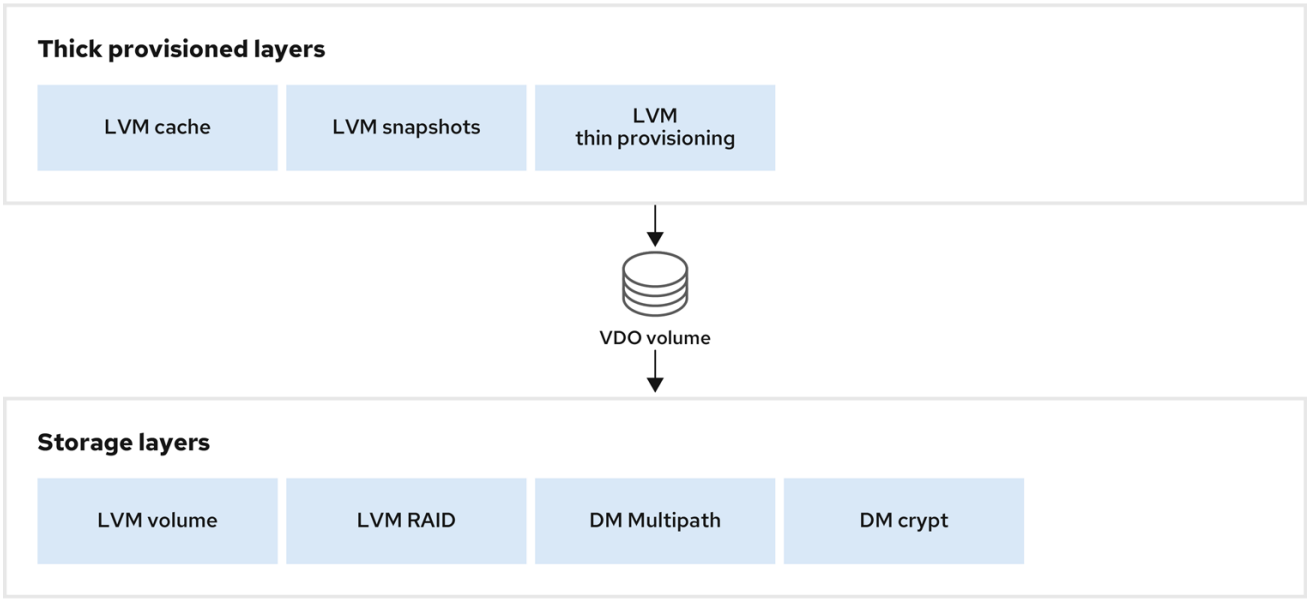
VDO の上にシックプロビジョニングレイヤーを配置できます。シックプロビジョニングレイヤーについては、次の 2 つの側面を考慮する必要があります。

- シックデバイスの未使用の論理領域への新しいデータの書き込み。VDO またはその他のシンプロビジョニングストレージを使用している場合、この種の書き込み中にデバイスの領域が不足していると報告されることがあります。
- 新しいデータによるシックデバイスの使用済み論理領域の上書き。VDO を使用している場合、データを上書きすると、デバイスの領域が不足しているという報告が表示されることがあります。

これらの制限は、VDO 層より上のすべてのレイヤーに影響します。VDO デバイスが監視されていない場合には、VDO 層の上にあるシックプロビジョニングのボリュームで、物理領域が予期せず不足する可能性があります。

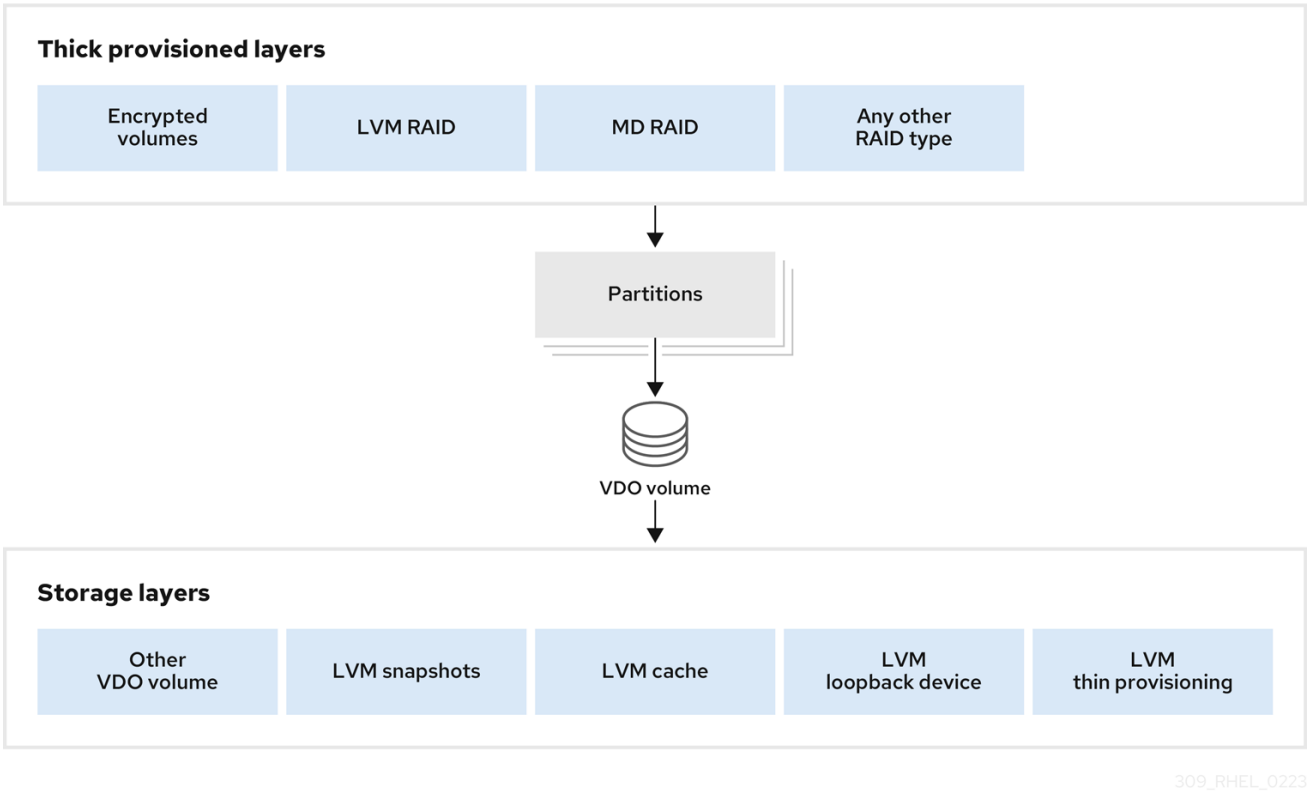
次のサポート対象およびサポート対象外の VDO ボリューム設定の例を参照してください。

図1.3 サポートされている VDO ボリューム設定



309_RHEL_0223

図1.4 サポートされていない VDO ボリューム設定



309_RHEL_0223

関連情報

- LVM レイヤーによる VDO のスタックの詳細は、LVM ボリュームのスタック を参照してください。

1.6.4. 物理サイズ別の VDO 要件の例

以下の表は、基盤となるボリュームの物理サイズに基づいた、VDO のシステム要件の概算を示しています。それぞれの表には、プライマリーストレージ、バックアップストレージなどの、目的のデプロイメントに適した要件が記載されています。

正確な数値は、VDO ボリュームの設定により異なります。

プライマリーストレージのデプロイメント

プライマリーストレージの場合、UDS インデックスのサイズは、物理サイズの 0.01% から 25% になります。

表1.2 プライマリーストレージのストレージ要件およびメモリー要件

物理サイズ	メモリー使用率: UDS	メモリー使用率: VDO	ディスク使用率	インデックスタイプ
10 GB - 1 TB	250 MB	472 MB	2.5 GB	Dense
2 - 10 TB	1 GB	3 GB	10 GB	Dense
	250 MB		22 GB	Sparse
11 - 50 TB	2 GB	14 GB	170 GB	Sparse
51 - 100 TB	3 GB	27 GB	255 GB	Sparse
101 - 256 TB	12 GB	69 GB	1020 GB	Sparse

バックアップストレージのデプロイメント

バックアップストレージの場合、UDS インデックスは、バックアップセットのサイズよりは大きくなりますが、物理サイズと同じか、より小さくなります。バックアップセットや物理サイズが今後大きくなる可能性がある場合は、これをインデックスサイズに組み込んでください。

表1.3 バックアップストレージのストレージ要件およびメモリー要件

物理サイズ	メモリー使用率: UDS	メモリー使用率: VDO	ディスク使用率	インデックスタイプ
10 GB - 1 TB	250 MB	472 MB	2.5 GB	Dense
2 - 10 TB	2 GB	3 GB	170 GB	Sparse
11 - 50 TB	10 GB	14 GB	850 GB	Sparse
51 - 100 TB	20 GB	27 GB	1700 GB	Sparse
101 - 256 TB	26 GB	69 GB	3400 GB	Sparse

1.7. VDO のインストール

この手順では、VDO ボリュームの作成、マウント、および管理に必要なソフトウェアをインストールします。

手順

- VDO ソフトウェアをインストールします。

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

1.8. VDO ボリュームの作成

この手順では、ブロックデバイスに VDO ボリュームを作成します。

前提条件

- VDO ソフトウェアをインストールしている。「VDO のインストール」を参照してください。
- 拡張可能なストレージをバッキングブロックデバイスとして使用している。詳細は、「ストレージスタックの VDO の 配置」 を参照してください。

手順

以下のすべての手順で、**vdo-name** を、VDO ボリュームに使用する識別子 (**vdo1** など) に置き換えます。システムの VDO の各インスタンスに、それぞれ別の名前とデバイスを使用する必要があります。

1. VDO ボリュームを作成するブロックデバイスの永続的な名前を確認してください。永続的な名前の詳細は、*6章永続的な命名属性の概要* を参照してください。
   永続的なデバイス名を使用しないと、今後デバイスの名前が変わった場合に、VDO が正しく起動しなくなることがあります。

2. VDO ボリュームを作成します。

```
# vdo create \
    --name=vdo-name \
    --device=block-device \
    --vdoLogicalSize=logical-size
```

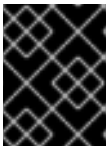
   - **block-device** を、VDO ボリュームを作成するブロックデバイスの永続名に置き換えます。たとえば、**/dev/disk/by-id/scsi-3600508b1001c264ad2af21e903ad031f** です。
   - **logical-size** を、VDO ボリュームが含まれる論理ストレージのサイズに置き換えます。
     - アクティブな仮想マシンまたはコンテナーストレージの場合は、使用する論理サイズが、ブロックデバイスの物理サイズの 10 倍になるようにします。たとえば、ブロックデバイスのサイズが 1 TB の場合は、**10T** を使用します。
     - オブジェクトストレージの場合は、使用する論理サイズを、ブロックデバイスの物理サイズの 3 倍になるようにします。たとえば、ブロックデバイスのサイズが 1 TB の場合は、**3T** を使用します。
   - 物理ブロックデバイスが 16TiB を超える場合は、**--vdoSlabSize=32G** オプションを指定して、ボリューム上のスラブサイズを 32GiB に増やします。
     16TiB を超えるブロックデバイスでデフォルトのスラブサイズ 2GiB を使用すると、**vdo create** コマンドが失敗し、以下のエラーが出力されます。

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

例1.1 コンテナーストレージ用に VDO の作成

たとえば、1 TB ブロックデバイスでコンテナーストレージ用に VDO ボリュームを作成するには、次のコマンドを実行します。

```
# vdo create \
      --name=vdo1 \
      --device=/dev/disk/by-id/scsi-3600508b1001c264ad2af21e903ad031f \
      --vdoLogicalSize=10T
```



重要

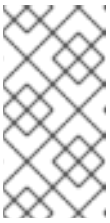
VDO ボリュームの作成中に問題が発生した場合は、ボリュームを削除します。詳しくは 「作成に失敗した VDO ボリュームの削除」 を参照してください。

3. VDO ボリュームにファイルシステムを作成します。
   - XFS ファイルシステムの場合:

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

   - ext4 ファイルシステムの場合:

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



注記

新しく作成された VDO ボリュームでの **-K** および **-E nodiscard** オプションの目的は、割り当てられていないブロックには影響しないため、リクエストの送信に時間を費やさないようにすることです。新しい VDO ボリュームは、100% 割り当てられていない状態で開始されます。

4. 次のコマンドを使用して、システムが新しいデバイスノードを登録するまで待機します。

   ```
   # udevadm settle
   ```

次のステップ

1. ファイルシステムをマウントします。詳しくは 「VDO ボリュームのマウント」 を参照してください。
2. VDO デバイスのファイルシステムで **discard** 機能を有効にします。詳しくは 「定期的なブロック破棄の有効化」 を参照してください。

関連情報

- man ページの **vdo(8)**

1.9. VDO ボリュームのマウント

この手順では、手動で、または永続的に、VDO ボリュームにファイルシステムをマウントします。

前提条件

- システムで VDO ボリュームが作成されている。手順は、「VDO ボリュームの作成」 を参照してください。

手順

- VDO ボリュームに手動でファイルシステムをマウントするには、以下のコマンドを使用します。

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

- システムの起動時にファイルシステムを自動的にマウントするように設定するには、**/etc/fstab** ファイルに以下の行を追加します。
  - XFS ファイルシステムの場合:

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

  - ext4 ファイルシステムの場合:

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

  VDO ボリュームが、iSCSI などのネットワークを必要とするブロックデバイスに配置されている場合は、**_netdev** マウントオプションを追加します。

関連情報

- **vdo(8)** man ページ
- iSCSI や、ネットワークを必要とするその他のブロックデバイスの **_netdev** マウントオプションに関する情報は、**systemd.mount(5)** の man ページの を参照してください。

1.10. 定期的なブロック破棄の有効化

systemd タイマーを有効にして、サポートしているすべてのファイルシステムで未使用ブロックを定期的に破棄できます。

手順

- **systemd** タイマーを有効にして起動します。

```
# systemctl enable --now fstrim.timer
Created symlink /etc/systemd/system/timers.target.wants/fstrim.timer →
/usr/lib/systemd/system/fstrim.timer.
```

検証

- タイマーのステータスを確認します。

```
# systemctl status fstrim.timer
fstrim.timer - Discard unused blocks once a week
   Loaded: loaded (/usr/lib/systemd/system/fstrim.timer; enabled; vendor preset: disabled)
   Active: active (waiting) since Wed 2023-05-17 13:24:41 CEST; 3min 15s ago
  Trigger: Mon 2023-05-22 01:20:46 CEST; 4 days left
     Docs: man:fstrim

May 17 13:24:41 localhost.localdomain systemd[1]: Started Discard unused blocks once a
week.
```

1.11. VDO の監視

この手順では、VDO ボリュームから、使用方法と効率に関する情報を取得する方法を説明します。

前提条件

- VDO ソフトウェアをインストールしている。「VDO のインストール」を参照してください。

手順

- **vdostats** ユーティリティーは、VDO ボリュームに関する情報を取得します。

```
# vdostats --human-readable

Device                 1K-blocks     Used     Available    Use%    Space saving%
/dev/mapper/node1osd1    926.5G       21.0G    905.5G       2%      73%
/dev/mapper/node1osd2    926.5G       28.2G    898.3G       3%      64%
```

関連情報

- man ページの **vdostats(8)**

第2章 VDO のメンテナンス

VDO ボリュームのデプロイ後、特定のタスクを実行して、ボリュームを維持または最適化することができます。VDO ボリュームの適切な機能には、以下の一部のタスクが必要です。

前提条件

- VDO がインストールされ、デプロイされます。1章*VDO のデプロイメント*を参照してください。

2.1. VDO ボリューム上の空き領域の管理

VDO は、シンプロビジョニングされたブロックストレージターゲットです。そのため、VDO ボリュームで領域の使用状況をアクティブに監視し、管理する必要があります。

2.1.1. VDO ボリュームの物理サイズおよび論理サイズ

VDO は、物理サイズ、利用可能な物理サイズ、および論理サイズを次の方法で利用します。

物理サイズ

これは、基礎となるブロックデバイスと同じサイズです。VDO は、以下の目的でこのストレージを使用します。

- 重複排除および圧縮される可能性があるユーザーデータ
- UDS インデックスなどの VDO メタデータ

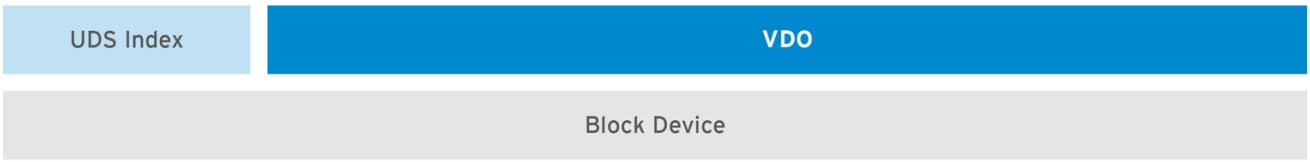
利用可能な物理サイズ

これは、VDO がユーザーデータに使用できる物理サイズの一部です。
これは、メタデータのサイズを引いた物理サイズと同等で、指定のスラブサイズでボリュームをスラブに分割した後の残りを引いたものと同じです。

論理サイズ

これは、VDO ボリュームがアプリケーションに提示するプロビジョニングされたサイズです。通常、これは利用可能な物理サイズよりも大きくなります。**--vdoLogicalSize** オプションを指定しないと、論理ボリュームのプロビジョニングが **1:1** の比率にプロビジョニングされます。たとえば、VDO ボリュームが 20 GB ブロックデバイスの上に置かれている場合は、2.5 GB が UDS インデックス用に予約されます (デフォルトのインデックスサイズが使用される場合)。残りの 17.5 GB は、VDO メタデータおよびユーザーデータに提供されます。そのため、消費する利用可能なストレージは 17.5 GB を超えません。実際の VDO ボリュームを設定するメタデータにより、これよりも少なくなる可能性があります。
VDO は現在、絶対最大論理サイズ 4PB の物理ボリュームの最大 254 倍の論理サイズに対応します。

図2.1 VDO ディスク組織



この図では、VDO で重複排除したストレージターゲットがブロックデバイス上に完全に配置されています。つまり、VDO ボリュームの物理サイズは、基礎となるブロックデバイスと同じサイズになります。

関連情報

- さまざまなサイズのブロックデバイスに必要なストレージ VDO メタデータのサイズは、「物理サイズ別の VDO 要件の例」を参照してください。

2.1.2. VDO でのシンプロビジョニング

VDO は、シンプロビジョニングされたブロックストレージターゲットです。VDO ボリュームが使用する物理領域のサイズは、ストレージのユーザーに示されるボリュームのサイズとは異なる可能性があります。この相違を活用して、ストレージのコストを削減できます。

容量不足の条件

書き込んだデータが、予想される最適化率に到達できない場合は、ストレージ領域が予想外に不足しないように注意してください。

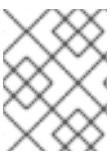
論理ブロック (仮想ストレージ) の数が物理ブロック (実際のストレージ) の数を超えると、ファイルシステムおよびアプリケーションで領域が予想外に不足する可能性があります。このため、VDO を使用するストレージシステムは、VDO ボリュームの空きプールのサイズを監視する方法で提供する必要があります。

vdostats ユーティリティーを使用すると、この空きプールのサイズを確認できます。このユーティリティーのデフォルト出力には、Linux の **df** ユーティリティーと同様の形式で稼働しているすべての VDO ボリュームの情報が記載されます。以下に例を示します。

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

VDO ボリュームの物理ストレージ領域が不足しそうになると、VDO は、システムログに、以下のような警告を出力します。

```
Oct  2 17:13:39 system lvm[13863]: Monitoring VDO pool vdo-name.
Oct  2 17:27:39 system lvm[13863]: WARNING: VDO pool vdo-name is now 80.69% full.
Oct  2 17:28:19 system lvm[13863]: WARNING: VDO pool vdo-name is now 85.25% full.
Oct  2 17:29:39 system lvm[13863]: WARNING: VDO pool vdo-name is now 90.64% full.
Oct  2 17:30:29 system lvm[13863]: WARNING: VDO pool vdo-name is now 96.07% full.
```



注記

この警告メッセージは、**lvm2-monitor** サービスが実行している場合に限り表示されます。これは、デフォルトで有効になっています。

容量不足の状況を防ぐ方法

空きプールのサイズが特定のレベルを下回る場合は、以下を行うことができます。

- データの削除。これにより、削除したデータが重複していないと、領域が回収されます。データを削除しても、破棄が行われないと領域を解放しません。
- 物理ストレージの追加

重要

VDO ボリュームで物理領域を監視し、領域不足を回避します。物理ブロックが不足すると、VDO ボリュームに最近書き込まれたデータや、未承認のデータが失われることがあります。

シンプロビジョニング、TRIM コマンド、および DISCARD コマンド

シンプロビジョニングによるストレージの削減から利益を得るには、データを削除するタイミングを物理ストレージ層が把握する必要があります。シンプロビジョニングされたストレージで動作するファイルシステムは、**TRIM** コマンドまたは **DISCARD** コマンドを送信して、論理ブロックが不要になったときにストレージシステムに通知します。

TRIM コマンドまたは **DISCARD** コマンドを送信する方法はいくつかあります。

- **discard** マウントオプションを使用すると、ブロックが削除されるたびに、ファイルシステムがそのコマンドを送信できます。
- **fstrim** などのユーティリティーを使用すると、制御された方法でコマンドを送信できます。このようなユーティリティーは、どの論理ブロックが使用されていないかを検出し、**TRIM** コマンドまたは **DISCARD** コマンドの形式でストレージシステムに情報を送信するようにファイルシステムに指示します。

未使用のブロックで **TRIM** または **DISCARD** を使用する必要は、VDO に特有のものではありません。シンプロビジョニングしたストレージシステムでも、同じ課題があります。

2.1.3. VDO の監視

この手順では、VDO ボリュームから、使用方法と効率に関する情報を取得する方法を説明します。

前提条件

- VDO ソフトウェアをインストールしている。「VDO のインストール」を参照してください。

手順

- **vdostats** ユーティリティーは、VDO ボリュームに関する情報を取得します。

```
# vdostats --human-readable

Device                 1K-blocks    Used    Available    Use%    Space saving%
/dev/mapper/node1osd1    926.5G      21.0G    905.5G      2%      73%
/dev/mapper/node1osd2    926.5G      28.2G    898.3G      3%      64%
```

関連情報

- man ページの **vdostats(8)**

2.1.4. ファイルシステムで VDO の領域の回収

この手順では、ファイルシステムをホストする VDO ボリュームのストレージの領域を回収する方法を説明します。

ファイルシステムが、**DISCARD** コマンド、**TRIM** コマンド、または **UNMAP** コマンドを使用して、ブロックが空いていることを伝えない限り、VDO は領域を回収できません。

手順

- VDO ボリュームのファイルシステムが破棄操作に対応している場合は、その操作を有効化してください。5章*未使用ブロックの破棄*を参照してください。
- ファイルシステムが、**DISCARD**、**TRIM**、または **UNMAP** を使用していない場合は、空き領域を手動で回収できます。バイナリーゼロで設定されるファイルを保存し、空き領域を埋め、そのファイルを削除します。

2.1.5. ファイルシステムを使用しない VDO の領域の回収

この手順では、ファイルシステムを使用せずにブロックストレージターゲットとして使用される VDO ボリュームでストレージ領域を回収します。

手順

- **blkdiscard** ユーティリティーを使用します。
  たとえば、VDO ボリュームは、LVM をその上にデプロイすることで、1 つの VDO を複数のサブボリュームに分類できます。論理ボリュームのプロビジョニングを解除する前に、**blkdiscard** ユーティリティーを使用して、その論理ボリュームにより使用されていた領域を解放します。

  LVM は、**REQ_DISCARD** コマンドに対応し、領域を解放するために適切な論理ブロックアドレスで VDO にリクエストを転送します。その他のボリュームマネージャーを使用する場合は、**REQ_DISCARD** に対応する必要があります。同じように、SCSI デバイスの場合は **UNMAP**、または ATA デバイスの場合は **TRIM** に対応している必要があります。

関連情報

- man ページの **blkdiscard(8)**

2.1.6. ファイバーチャンネルまたはイーサネットネットワーク上で VDO の領域の回収

この手順では、LIO、SCST などの SCSI ターゲットフレームワークを使用して、ファイバーチャネルストレージファブリック、またはイーサネットネットワークでホストにプロビジョニングされる VDO ボリューム (またはボリュームの一部) でストレージ領域を回収します。

手順

- SCSI イニシエーターは、**UNMAP** コマンドを使用して、シンプロビジョニングしたストレージターゲットで領域を解放できますが、SCSI ターゲットフレームワークに、このコマンドのサポートを通知するように設定する必要があります。これは、通常、このボリュームでシンプロビジョニングを有効にすることで行います。

次のコマンドを実行して、Linux ベースの SCSI イニシエーターで **UNMAP** のサポートを検証します。

```
# sg_vpd --page=0xb0 /dev/device
```

出力では、**Maximum unmap LBA count(未マッピングの LBA の最大数)**の値がゼロより大きいことを確認します。

2.2. VDO ボリュームの開始または停止

指定した VDO ボリューム、またはすべての VDO ボリューム、および関連の UDS インデックスを起動または停止できます。

2.2.1. 起動して有効にした VDO ボリューム

システムの起動時に、**vdo systemd** ユニットは、**有効** に設定されている VDO デバイスをすべて自動的に **起動** します。

vdo パッケージをインストールすると、**vdo systemd** ユニットがインストールされ、デフォルトで有効になっています。このユニットは、システム起動時に **vdo start --all** コマンドを自動的に実行して、有効になっている VDO ボリュームをすべて起動します。

--activate=disabled オプションを **vdo create** コマンドに指定することで、自動的に起動しない VDO ボリュームを作成できます。

開始順序

システムによっては、LVM ボリュームを、VDO ボリュームの上または下の両方に配置できるものがあります。このようなシステムでは、適切な順序でサービスを開始する必要があります。

1. 下層の LVM を最初に起動する必要があります。大概のシステムでは、LVM パッケージがインストールされていれば、この層が起動するように自動的に設定されます。
2. 次に、**vdo systemd** ユニットを起動する必要があります。
3. 最後に、稼働している VDO の上にある LVM ボリュームやその他のサービスを実行するために、その他のスクリプトを実行する必要があります。

ボリュームの停止にかかる時間

VDO ボリュームの停止にかかる時間は、ストレージデバイスの速度と、ボリュームへの書き込みが必要なデータ量により異なります。

- ボリュームは、UDS インデックスの 1GiB ごとに、約 1 GiB のデータを常に書き込みます。
- ボリュームはブロックマップキャッシュのサイズと、スラブごとに最大 8MiB のデータ量を書き込みます。
- ボリュームは、未処理のすべての IO 要求の処理を終了する必要があります。

2.2.2. VDO ボリュームの作成

この手順では、システムにある一部の VDO ボリューム、またはすべての VDO ボリュームを起動します。

手順

- 特定の VDO ボリュームを起動するには、以下のコマンドを使用します。

```
# vdo start --name=my-vdo
```

- すべての VDO ボリュームを起動するには、次のコマンドを実行します。

```
# vdo start --all
```

関連情報

- man ページの **vdo(8)**

2.2.3. VDO ボリュームの停止

この手順では、システムで一部の VDO ボリュームまたはすべての VDO ボリュームを停止します。

手順

1. ボリュームを停止します。
   - 特定の VDO ボリュームを停止するには、以下のコマンドを使用します。

```
# vdo stop --name=my-vdo
```

   - すべての VDO ボリュームを停止するには、以下のコマンドを使用します。

```
# vdo stop --all
```

2. ボリュームが、ディスクへのデータの書き込みを終了するまで待ちます。

関連情報

- man ページの **vdo(8)**

2.2.4. 関連情報

- シャットダウンが正常に行われなかった場合は、システムを再起動したときに VDO が再構築を行いメタデータの一貫性を確認し、必要であれば修復します。再ビルドプロセスの詳細は、「シャットダウンが適切に行われない場合の VDO ボリュームの復旧」 を参照してください。

2.3. システムブートで VDO ボリュームを自動的に起動

VDO ボリュームを設定し、システムの起動時に自動的に起動するようにします。自動起動を無効にすることもできます。

2.3.1. 起動して有効にした VDO ボリューム

システムの起動時に、**vdo systemd** ユニットは、**有効** に設定されている VDO デバイスをすべて自動的に **起動** します。

vdo パッケージをインストールすると、**vdo systemd** ユニットがインストールされ、デフォルトで有効になっています。このユニットは、システム起動時に **vdo start --all** コマンドを自動的に実行して、有効になっている VDO ボリュームをすべて起動します。

--activate=disabled オプションを **vdo create** コマンドに指定することで、自動的に起動しない VDO ボリュームを作成できます。

開始順序

システムによっては、LVM ボリュームを、VDO ボリュームの上または下の両方に配置できるものがあります。このようなシステムでは、適切な順序でサービスを開始する必要があります。

1. 下層の LVM を最初に起動する必要があります。大概のシステムでは、LVM パッケージがインストールされていれば、この層が起動するように自動的に設定されます。
2. 次に、**vdo systemd** ユニットを起動する必要があります。
3. 最後に、稼働している VDO の上にある LVM ボリュームやその他のサービスを実行するために、その他のスクリプトを実行する必要があります。

ボリュームの停止にかかる時間

VDO ボリュームの停止にかかる時間は、ストレージデバイスの速度と、ボリュームへの書き込みが必要なデータ量により異なります。

- ボリュームは、UDS インデックスの 1GiB ごとに、約 1 GiB のデータを常に書き込みます。
- ボリュームはブロックマップキャッシュのサイズと、スラブごとに最大 8MiB のデータ量を書き込みます。
- ボリュームは、未処理のすべての IO 要求の処理を終了する必要があります。

2.3.2. VDO ボリュームの有効化

この手順では、VDO ボリュームを有効にして、自動的に開始できるようにします。

手順

- 特定のボリュームを有効にするには、以下のコマンドを実行します。

```
# vdo activate --name=my-vdo
```

- すべてのボリュームを有効にするには、以下のコマンドを実行します。

```
# vdo activate --all
```

関連情報

- man ページの **vdo(8)**

2.3.3. VDO ボリュームの無効化

この手順では、VDO ボリュームを無効にして、自動的に起動しないようにします。

手順

- 特定のボリュームを無効にするには、以下のコマンドを実行します。

```
# vdo deactivate --name=my-vdo
```

- すべてのボリュームを無効にするには、以下のコマンドを実行します。

```
# vdo deactivate --all
```

関連情報

- man ページの **vdo(8)**

2.4. VDO 書き込みモードの選択

基となるブロックデバイスでの要件に応じて、VDO ボリュームに書き込みモードを設定できます。デフォルトでは、VDO は書き込みモードを自動的に選択します。

2.4.1. VDO 書き込みモード

VDO は、以下の書き込みモードに対応します。

sync

VDO が **sync** モードの場合、その上の層は、書き込みコマンドがデータを永続ストレージに書き込むことを想定します。したがって、このモードは、ファイルシステムやアプリケーションには必要ありません。FLUSH リクエストまたは FUA (強制ユニットアクセス) リクエストを発行すると、データは、重要な点で持続します。
VDO は、書き込みコマンドが完了したときに、基となるストレージが、データが永続ストレージに書き込まれることを保証する場合に限り、**sync** モードに設定する必要があります。つまり、ストレージには揮発性の書き込みキャッシュがないか、ライトスルーキャッシュが存在する必要があります。

async

VDO が **async** モードの場合は、書き込みコマンドが承認されたときに、データが永続ストレージに書き込まれることを VDO が保証しません。ファイルシステムまたはアプリケーションは、各トランザクションの重要な点でデータの永続性を保証するために、FLUSH リクエストまたは FUA リクエストを発行する必要があります。
書き込みコマンドが完了したときに、基となるストレージが永続ストレージに対するデータの書き込みを保証しない場合は、VDO を **async** モードに設定する必要があります。これは、ストレージに揮発性のあるライトバックキャッシュがある場合です。

async-unsafe

このモードには、**async** と同じプロパティーがありますが、ACID (Atomicity, Consistency, Isolation, Durability) に準拠していません。**async** と比較して、**async-unsafe** のパフォーマンスは向上します。

警告

VDO ボリュームに関する ACID コンプライアンスを想定するアプリケーションまたはファイルシステムが稼働する場合は、**async-unsafe** モードにより予想外のデータ損失が生じる可能性があります。

auto

auto モードは、各デバイスの性質に基づいて、**sync** または **async** を自動的に選択します。以下はデフォルトのオプションになります。

2.4.2. VDO 書き込みモードの内部処理

VDO の書き込みモードは **sync** と **async** です。以下では、これらのモードの操作について説明します。

kvdo モジュールが同期 (**synch**) モードで動作している場合は、以下を行います。

1. リクエストのデータを一時的に、割り当てられたブロックに書き込み、リクエストを承認します。
2. 承認が完了すると、ブロックデータの MurmurHash-3 署名を計算してブロックの重複排除が試行されます。これは、VDO インデックスに送信されます。
3. VDO インデックスに同じ署名とともにブロックのエントリーが含まれる場合、**kvdo** は示されたブロックを読み込み、同一であるかを検証するために 2 つのブロックのバイト対バイトの比較を行います。
4. 同一であることが確認されると、**kvdo** はブロックマップを更新して、論理ブロックが、一致する物理ブロックを指定し、割り当てられた物理ブロックをリリースします。
5. VDO インデックスに、書き込まれているブロックの署名のエントリーを含まない場合や、示されたブロックが同じデータを含まない場合は、**kvdo** はブロックマップを更新して、一時的な物理ブロックを永続的にします。

kvdo が非同期 (**async**) モードで動作している場合は、以下のコマンドを実行します。

1. データを書き込む代わりに、リクエストをすぐに承認します。
2. 上記の説明と同じように、ブロックの重複排除試行が行われます。
3. ブロックが重複していることになると、**kvdo** はブロックマップを更新し、割り当てられたブロックを解放します。解放しない場合は、リクエストのデータが、割り当てられたブロックに書き込み、ブロックマップを更新して物理ブロックを永続的にします。

2.4.3. VDO ボリュームにおける書き込みモードの確認

この手順では、選択した VDO ボリュームに対するアクティブな書き込みモードをリスト表示します。

手順

- 以下のコマンドを使用して、VDO ボリュームが使用する書き込みモードを表示します。

```
# vdo status --name=my-vdo
```

出力されるファイルには、以下が記載されます。

- **設定した書き込みポリシー** - **sync**、**async**、または **auto** から選択されるオプションです。
- **書き込みポリシー** - VDO が適用される特定の書き込みモードで、**sync** または **async** になります。

2.4.4. 揮発性キャッシュの確認

この手順では、ブロックデバイスに揮発性キャッシュがあるかどうかを確認します。情報を使用して、VDO 書き込みモードである **sync** と **async** のいずれかを選択できます。

手順

1. デバイスにライトバックキャッシュがあるかどうかを判断する場合は、以下のいずれか方法を使用します。
   - **sysfs** ファイルの **/sys/block/block-device/device/scsi_disk/identifier/cache_type** を読み込みます。以下に例を示します。

```
$ cat '/sys/block/sda/device/scsi_disk/7:0:0:0/cache_type'

write back
```

```
$ cat '/sys/block/sdb/device/scsi_disk/1:2:0:0/cache_type'

None
```

   - また、カーネルブートログでは、上記のデバイスに書き込みキャッシュがあるかどうかを調べることができます。

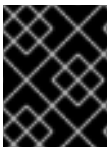
```
sd 7:0:0:0: [sda] Write cache: enabled, read cache: enabled, does not support DPO or FUA
sd 1:2:0:0: [sdb] Write cache: disabled, read cache: disabled, supports DPO and FUA
```

2. 上の例では、以下のようになります。
   - デバイス **sda** には、ライトバックキャッシュが**ある** ことが示されています。**async** モードを使用します。
   - デバイス **sdb** には、ライトバックキャッシュが **ない** ことが示されています。**sync** モードを使用します。

cache_type の値が **None** または **write through** である場合は、**sync** 書き込みモードを使用するように VDO を設定する必要があります。

2.4.5. VDO 書き込みモードの設定

この手順では、既存のボリュームの場合、またはボリュームの新規作成時に、VDO ボリュームに書き込みモードを設定します。



重要

誤った書き込みモードを使用すると、停電、システムクラッシュ、またはディスクとの接続が予期せず失われた時に、データが失われることがあります。

前提条件

- デバイスに対してどの書き込みモードが正しいかを確認している。「揮発性キャッシュの確認」を参照してください。

手順

- 既存の VDO ボリュームまたは新規ボリュームの作成時に、書き込みモードを設定できます。
  - 既存の VDO ボリュームを変更するには、以下のコマンドを使用します。

```
# vdo changeWritePolicy --writePolicy=sync|async|async-unsafe|auto \
                        --name=vdo-name
```

  - VDO ボリュームの作成時に書き込みモードを指定するには、**--writePolicy=sync|async|async-unsafe|auto** オプションを **vdo create** コマンドに追加します。

2.5. シャットダウンが適切に行われない場合の VDO ボリュームの復旧

シャットダウンが適切に行われない場合に VDO ボリュームを復旧して、動作を継続できます。多くのタスクは自動化されています。また、プロセスの障害により VDO ボリュームの作成に失敗した場合は、クリーンアップできます。

2.5.1. VDO 書き込みモード

VDO は、以下の書き込みモードに対応します。

sync

VDO が **sync** モードの場合、その上の層は、書き込みコマンドがデータを永続ストレージに書き込むことを想定します。したがって、このモードは、ファイルシステムやアプリケーションには必要ありません。FLUSH リクエストまたは FUA (強制ユニットアクセス) リクエストを発行すると、データは、重要な点で持続します。
VDO は、書き込みコマンドが完了したときに、基となるストレージが、データが永続ストレージに書き込まれることを保証する場合に限り、**sync** モードに設定する必要があります。つまり、ストレージには揮発性の書き込みキャッシュがないか、ライトスルーキャッシュが存在する必要があります。

async

VDO が **async** モードの場合は、書き込みコマンドが承認されたときに、データが永続ストレージに書き込まれることを VDO が保証しません。ファイルシステムまたはアプリケーションは、各トランザクションの重要な点でデータの永続性を保証するために、FLUSH リクエストまたは FUA リクエストを発行する必要があります。
書き込みコマンドが完了したときに、基となるストレージが永続ストレージに対するデータの書き込みを保証しない場合は、VDO を **async** モードに設定する必要があります。これは、ストレージに揮発性のあるライトバックキャッシュがある場合です。

async-unsafe

このモードには、**async** と同じプロパティーがありますが、ACID (Atomicity, Consistency, Isolation, Durability) に準拠していません。**async** と比較して、**async-unsafe** のパフォーマンスは向上します。

警告

VDO ボリュームに関する ACID コンプライアンスを想定するアプリケーションまたはファイルシステムが稼働する場合は、**async-unsafe** モードにより予想外のデータ損失が生じる可能性があります。

auto

auto モードは、各デバイスの性質に基づいて、**sync** または **async** を自動的に選択します。以下はデフォルトのオプションになります。

2.5.2. VDO ボリュームの復旧

シャットダウンが適切に行われなかった場合に VDO ボリュームを再起動すると、VDO は以下の操作を実行します。

- ボリューム上のメタデータの一貫性を検証する
- メタデータの一部を再構築して、必要に応じて修復する

再構築は自動で行われ、ユーザーの介入は必要ありません。

VDO は、アクティブな書き込みモードに依存する各種書き込みを再構築します。

sync

VDO が同期ストレージで稼働していて、書き込みポリシーが **sync** に設定されていた場合は、ボリュームに書き込まれたすべてのデータが完全に復元されます。

async

書き込みポリシーが **async** であった場合、一部の書き込みは永続性が保たれないと復元されないことがあります。これは、VDO に **FLUSH** コマンド、または FUA (強制ユニットアクセス) フラグでタグ付けされた書き込み I/O を送信することで行われます。これは、**fsync**、**fdatasync**、**sync**、**umount** などのデータ整合性の操作を呼び出すことで、ユーザーモードから実行できます。

どちらのモードであっても、フラッシュによって承認されていない、あるいは確認されていない一部の書き込みも再構築されることがあります。

自動復元および手動復元

VDO ボリュームが **復旧** 操作モードになると、VDO は、オンラインに戻ってから、適切ではない VDO ボリュームを自動的に再構築します。これは **オンラインリカバリー** と呼ばれます。

VDO が正常に VDO ボリュームを復元できない場合は、ボリュームの再起動後も持続する **読み取り専用** モードにボリュームを置きます。再構築が強制されるため、問題を手動で修正する必要があります。

関連情報

- 自動リカバリーおよび手動リカバリー、ならびに VDO 操作モードの詳細は、「VDO 操作モード」 を参照してください。

2.5.3. VDO 操作モード

ここでは、VDO ボリュームが正常に動作しているか、エラーからの復旧であることを示すモードを説明します。

vdostats --verbose device コマンドを使用すると、VDO ボリュームの現在の操作モードを表示できます。出力内の **Operating mode** 属性を参照してください。

normal

これがデフォルトの操作モードです。以下のいずれかのステータスにより別のモードが強制されない限りに、VDO ボリュームは常に **normal** モードになります。新規作成された VDO ボリュームは **normal** モードで起動します。

recovering

シャットダウンの前に、VDO ボリュームがすべてのメタデータを保存しない場合は、次に起動した時に自動的に **recovering** モードになります。このモードになる一般的な理由は、突然の停電や、基となるストレージデバイスの問題です。
recovering モードでは、VDO は、デバイスのデータの物理ブロックごとに参照カウントを修正します。通常、リカバリーにはかなり時間がかかります。その時間は、VDO ボリュームの大きさ、基となるストレージデバイスの速度、VDO が同時に処理するリクエスト数により異なります。VDO ボリュームは、通常は以下の例外で動作します。

- 最初に、ボリュームに対する書き込み要求に利用できる領域の量が制限される場合があります。復旧するメタデータの数が多いと、より多くの空き領域が利用可能になります。
- VDO ボリュームの復旧中に書き込まれたデータは、そのデータが、復旧していないボリュームに含まれる場合に、クラッシュする前に書き込まれたデータに対する重複排除に失敗することがあります。VDO は、ボリュームの復旧中にデータを圧縮できます。圧縮したブロックの読み取りや上書きは可能です。
- オンラインリカバリーを行う際、一部の統計 (**blocks in use** や **blocks free** など) は利用できません。この統計は、再構築が完了すると利用できます。
- 継続中の復旧作業により、読み取りと書き込みの応答時間が通常よりも遅いことがあります。

recovering モードでは、VDO ボリュームを問題なくシャットダウンできます。シャットダウンする前に復元が完了しないと、デバイスは、次回起動時に再度 **recovering** モードになります。

VDO ボリュームは自動的に **recovering** モードを終了し、すべての参照カウントが修正されると、**normal** モードに移行します。管理者アクションは必要ありません。詳細は、「VDO ボリュームのオンラインリカバリー」 を参照してください。

read-only

VDO ボリュームが致命的な内部エラーに遭遇すると、**read-only** モードになります。**read-only** モードになるイベントには、メタデータの破損や、バッキングストレージデバイスが読み取り専用になるなどが挙げられます。このモードはエラー状態です。
read-only モードでは、データ読み取りは正常に機能しますが、データの書き込みは常に失敗します。管理者が問題を修正するまで、VDO ボリュームは **read-only** モードのままになります。

VDO ボリュームを **read-only** モードで問題なくシャットダウンできます。通常、モードは、VDO ボリュームが再起動した後も持続します。まれに、VDO ボリュームはバッキングストレージデバイスに **read-only** 状態を記録することができません。このような場合、VDO は代わりに復旧を試みま

す。

ボリュームが読み取り専用モードの場合、ボリュームのデータが損失または破損していないという保証はありません。このような場合、Red Hat は、読み取り専用のボリュームからデータをコピーして、バックアップからボリュームを復旧することを推奨します。

データ破損のリスクを許容できる場合は、VDO ボリュームのメタデータのオフライン再構築を強制することで、ボリュームをオンラインに戻して利用できるようにできます。再構築されたデータの整合性は保証できません。詳細は、「VDO ボリュームメタデータのオフライン再構築の強制」 を参照してください。

2.5.4. VDO ボリュームのオンラインリカバリー

この手順では、シャットダウンが正常に行われなかったときに、VDO ボリュームでオンラインリカバリーを実行して、メタデータを復旧します。

手順

1. VDO ボリュームを起動していない場合は、起動します。

```
# vdo start --name=my-vdo
```

   その他に何か行う必要はありません。復元は、バックグラウンドで実行します。

2. **blocks in use**、**blocks free** などのボリューム統計を使用する場合は、利用可能になるまで待ちます。

2.5.5. VDO ボリュームメタデータのオフライン再構築の強制

この手順では、シャットダウンが正常に行われなかった場合に、VDO ボリュームメタデータの強制的なオフライン再構築を実行して、復旧します。

警告

この手順では、ボリュームでデータが失われることがあります。

前提条件

- VDO ボリュームが起動している。

手順

1. ボリュームが読み取り専用モードかどうかを確認します。コマンド出力で **operating mode** 属性を確認します。

```
# vdo status --name=my-vdo
```

ボリュームが読み取り専用モードではない場合は、オフラインの再構築を強制する必要はありません。「VDO ボリュームのオンラインリカバリー」 に従って、オンラインリカバリーを実行します。

2. ボリュームが稼働している場合は停止します。

```
# vdo stop --name=my-vdo
```

3. **--forceRebuild** オプションを指定して、ボリュームを再起動します。

```
# vdo start --name=my-vdo --forceRebuild
```

2.5.6. 作成に失敗した VDO ボリュームの削除

この手順では、中間状態で VDO ボリュームをクリーンアップします。ボリュームの作成時に障害が発生した場合、ボリュームは中間状態になります。たとえば、以下のような場合に発生する可能性があります。

- システムのクラッシュ
- 停電
- 管理者が、実行中の **vdo create** コマンドに割り込み

手順

- クリーンアップを行う場合は、**--force** オプションを使用して、作成に失敗したボリュームを削除します。

```
# vdo remove --force --name=my-vdo
```

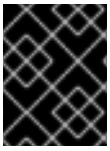
ボリュームの作成に失敗して、管理者がシステム設定を変更して競合を発生させたため、**--force** オプションが必要となります。

--force オプションを指定しないと、**vdo remove** コマンドが失敗して、以下のメッセージが表示されます。

```
[...]
A previous operation failed.
Recovery from the failure either failed or was interrupted.
Add '--force' to 'remove' to perform the following cleanup.
Steps to clean up VDO my-vdo:
umount -f /dev/mapper/my-vdo
udevadm settle
dmsetup remove my-vdo
vdo: ERROR - VDO volume my-vdo previous operation (create) is incomplete
```

2.6. UDS インデックスの最適化

UDS インデックスを特定の設定を行い、システムで最適化できます。



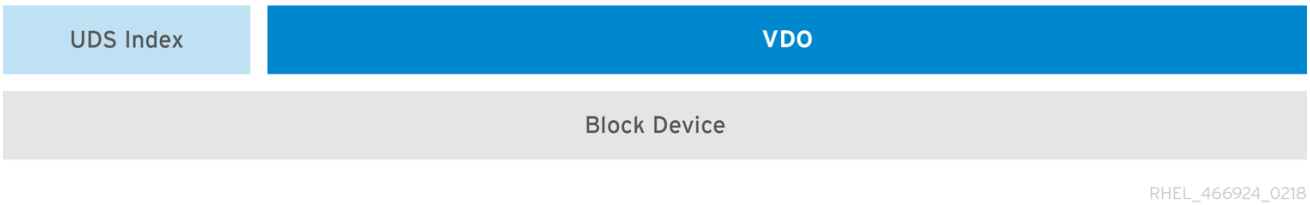
重要

VDO ボリュームを **作成したら**、UDS インデックスのプロパティーを変更することはできません。

2.6.1. VDO ボリュームのコンポーネント

VDO は、バッキングストアとしてブロックデバイスを使用します。これは、複数のディスク、パーティション、またはフラットファイルで設定される物理ストレージの集約を含めることができます。ストレージ管理ツールが VDO ボリュームを作成すると、VDO は、UDS インデックスおよび VDO ボリュームのボリューム領域を予約します。UDS インデックスと VDO ボリュームは対話して、重複排除したブロックストレージを提供します。

図2.2 VDO ディスク組織



VDO ソリューションは、以下のコンポーネントで設定されます。

kvdo

Linux Device Mapper 層に読み込まれるカーネルモジュールは、重複排除され、圧縮され、シンプロビジョニングされたブロックストレージボリュームを提供します。
kvdo モジュールはブロックデバイスを公開します。ブロックストレージ用にこのブロックデバイスに直接アクセスするか、XFS や ext4 などの Linux ファイルシステムを介して提示することができます。

kvdo が VDO ボリュームからデータ論理ブロックを読み取る要求を受信すると、要求された論理ブロックを基礎となる物理ブロックにマッピングし、要求したデータを読み取り、返します。

kvdo が VDO ボリュームにデータブロックを書き込む要求を受信すると、まず要求が DISCARD または TRIM のものであるか、またはデータが一貫してゼロかどうかを確認します。これらの条件のいずれかが true の場合、**kvdo** はブロックマップを更新し、リクエストを承認します。そうでない場合は、VDO はデータを処理して最適化します。

uds

ボリューム上の Universal Deduplication Service (UDS) インデックスと通信し、データの重複を分析するカーネルモジュール。新しい各データについて、その部分が保存してあるデータ内容と同一であるかどうかを UDS が素早く判断します。インデックスが一致すると、ストレージシステムは、同じ情報を複数格納しないように、既存の項目を内部的に参照できます。
UDS インデックスは、**uds** カーネルモジュールとしてカーネル内で実行します。

コマンドラインツール

最適化されたストレージの設定および管理

2.6.2. UDS インデックス

VDO は、UDS と呼ばれる高パフォーマンスの重複排除インデックスを使用して、格納されているときにデータの重複ブロックを検出します。

UDS インデックスは、VDO 製品の基盤を提供します。新しい各データについて、その部分が保存してあるデータ内容と同一であるかどうかを素早く判断します。インデックスが一致すると、ストレージシステムは、同じ情報を複数格納しないように、既存の項目を内部的に参照できます。

UDS インデックスは、**uds** カーネルモジュールとしてカーネル内で実行します。

重複排除ウィンドウ は、以前書き込んだことをインデックスが記憶しているブロックの数です。重複排除ウィンドウのサイズは設定可能です。特定のウィンドウサイズでは、インデックスに特定の RAM のサイズと、特定のディスク領域が必要です。ウィンドウのサイズは、通常、**--indexMem=size** オプションを使用してインデックスメモリーのサイズを指定して決定されます。VDO は、自動的に使用するディスク領域を決定します。

UDS インデックスは 2 つの部分から成ります。

- 一意のブロックごとに最大 1 つのエントリーを含むメモリーでは、コンパクトな表が用いられています。
- 発生する際に、インデックスに対して示される関連のブロック名を順に記録するオンディスクコンポーネント。

UDS は、メモリーの各エントリーに対して平均 4 バイトを使用します (キャッシュを含む)。

オンディスクコンポーネントは、UDS に渡されるデータの境界履歴を維持します。UDS は、直近で確認されたブロックの名前を含む、この重複排除ウィンドウ内のデータの重複排除アドバイスを提供します。重複排除ウィンドウでは、大規模なデータリポジトリーに対して必要なメモリーの量を制限する際に、UDS はできるだけ効率的にデータのインデックスを作成します。重複排除ウィンドウの境界の特徴により、重複排除レベルが高い多くのデータセットでは、一時的な局所性も多く確認されます。つまり、多くの重複排除は、同時に書き込まれたブロックセット間で発生します。さらに、通常、書き込まれたデータは、以前に書き込まれたデータよりも、最近書き込まれたデータを複製する可能性が高くなります。したがって、特定の時間間隔での特定のワークロードでは、UDS が最新のデータのみをインデックス付けしても、すべてのデータをインデックス付けしても、重複排除率は同じになることがよくあります。

重複データの場合では、一時的な局所性が示される傾向もあるため、ストレージシステム内のすべてのブロックにインデックスを作成する必要はほとんどありません。そうでない場合、インデックスメモリーのコストは、重複排除によるストレージコストの削減を上回ります。インデックスサイズの要件は、データの摂取率に密接に関連します。たとえば、合計容量が 100 TB のストレージシステムには、毎週 1 TB の摂取率を指定します。UDS は、4 TB の重複排除ウィンドウにより、前の月に書き込まれたデータで、最も冗長性が大きいものを検出できます。

2.6.3. 推奨される UDS インデックス設定

本セクションでは、目的のユースケースに基づいて、UDS インデックスとともに使用することが推奨されるオプションを説明します。

一般的には、Red Hat は、すべての実稼働環境に **sparse** の UDS インデックスを使用することを推奨します。これは、非常に効率的なインデックスデータ構造であり、重複排除ウィンドウでブロックごとに RAM の約 10 分の 1 バイトが必要です。ディスクの場合は、ブロックごとに約 72 バイトのディスク領域が必要です。このインデックスの最小設定は、ディスクで 256 MB の RAM と約 25 GB の領域を使用します。

この設定を使用するには、**--sparseIndex=enabled --indexMem=0.25** オプションを **vdo create** コマンドに指定します。この設定により、2.5 TB の重複排除ウィンドウが作成されます (つまり、2.5 TB の履歴が記録されます)。ほとんどのユースケースでは、2.5 TB の重複排除ウィンドウは、サイズが 10 TB までのストレージプールを重複排除するのに適しています。

ただし、インデックスのデフォルト設定は、**dense** のインデックスを使用します。このインデックスは RAM では非常に効率が悪い (10 倍) ですが、必要なディスク領域もはるかに少ない (10 倍) ため、制約された環境での評価に便利です。

一般に、推奨される設定は、VDO ボリュームの物理サイズの 4 分の 1 の重複排除ウィンドウです。ただし、これは実際の要件ではありません。小さな重複排除ウィンドウ (物理ストレージの量と比較) であっても、多くのユースケースで大量の重複データを確認できます。大概のウィンドウも使用できますが、ほとんどの場合、それを行う利点はほとんどありません。

関連情報

- この重要なシステムパラメーターの調整は、Red Hat テクニカルアカウントマネージャーまでご相談ください。

2.7. VDO での重複排除の有効化または無効化

一部の例では、ボリュームへの読み書きを行う機能を維持しながら、VDO ボリュームに書き込まれているデータの重複排除を一時的に無効にする場合があります。重複排除を無効にすると、後続の書き込みが重複排除できなくなりますが、すでに重複排除されたデータが残ります。

2.7.1. VDO での重複排除

重複排除とは、重複ブロックの複数のコピーを削除することで、ストレージリソースの消費を低減させるための技術です。

同じデータを複数回書き込むのではなく、VDO は各重複ブロックを検出し、元のブロックへの参照として記録します。VDO は、VDO の上にあるストレージ層により使用されている論理ブロックアドレスから、VDO の下にあるストレージ層で使用される物理ブロックアドレスへのマッピングを維持します。

重複排除を行った後、複数の論理ブロックアドレスが同じ物理ブロックアドレスにマッピングできます。共有ブロックと呼ばれます。ブロックストレージの共有は、VDO が存在しない場合に、読み込みブロックと書き込みブロックが行われるストレージのユーザーには表示されません。

共有ブロックが上書きされると、VDO は新しいブロックデータを保存する新しい物理ブロックを割り当て、共有物理ブロックにマッピングされたその他の論理ブロックアドレスが変更されないようにします。

2.7.2. VDO ボリュームでの重複排除の有効化

これにより、関連の UDS インデックスを再起動し、重複排除が再びアクティブになったことを VDO ボリュームに認識させます。

注記

重複排除はデフォルトで有効になっています。

手順

- VDO ボリュームでの重複排除を再起動するには、以下のコマンドを使用します。

```
# vdo enableDeduplication --name=my-vdo
```

2.7.3. VDO ボリュームでの重複排除の無効化

これにより、関連の UDS インデックスを停止し、重複排除がアクティブでなくなったことを VDO ボリュームに認識させます。

手順

- VDO ボリュームでの重複排除を停止するには、以下のコマンドを使用します。

```
# vdo disableDeduplication --name=my-vdo
```

- **--deduplication=disabled** オプションを **vdo create** コマンドに指定することで、新しい VDO ボリュームを作成するときに重複排除を無効にできます。

2.8. VDO で圧縮の有効化または無効化

VDO は、データ圧縮を提供します。これを無効にすると、パフォーマンスが最大化され、圧縮される可能性が低いデータの処理が高速化されます。再度有効にすると、スペースを節約できます。

2.8.1. VDO の圧縮

VDO は、ブロックレベルの重複排除の他に、HIOPS compression™ 技術を使用してインラインブロックレベルの圧縮も提供します。

VDO ボリューム圧縮はデフォルトでオンになっています。

重複排除は、仮想マシン環境とバックアップアプリケーションに最適なソリューションですが、圧縮は、通常、ログファイルやデータベースなどのブロックレベルの冗長性を見せない構造化および非構造化のファイル形式に非常に適しています。

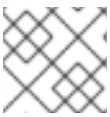
圧縮は、重複として認識されていないブロックで動作します。VDO が初めて一意のデータを見つけた場合は、データを圧縮します。その後の、保存しているデータのコピーは、追加の圧縮手順なしに重複排除されます。

圧縮機能は、同時に多くの圧縮操作に対応できるようにする並列化されたパッケージアルゴリズムに基づいています。最初にブロックを保存し、リクエストに応答したら、圧縮時に複数のブロックを検出する最適なパックアルゴリズムが、1つの物理ブロックに適合します。特定の物理ブロックが追加の圧縮ブロックを保持していないと判断すると、そのブロックはストレージに書き込まれます。圧縮されていないブロックは解放され、再利用されます。

要求したものに応答し、圧縮およびパッケージ化の操作を実行することにより、圧縮を使用することで課せられるレイテンシーが最小限に抑えられます。

2.8.2. VDO ボリュームでの圧縮の有効化

この手順では、VDO ボリュームで圧縮を有効化して、削減する領域を大きくします。



注記

デフォルトでは圧縮が有効になっています。

手順

- 再び起動するには、以下のコマンドを使用します。

```
# vdo enableCompression --name=my-vdo
```

2.8.3. VDO ボリュームでの圧縮の無効化

この手順では、VDO ボリュームで圧縮を停止して、パフォーマンスを最大化します。もしくは、圧縮できないと思われるデータの処理を高速化します。

手順

- 既存の VDO ボリュームで圧縮を停止するには、次のコマンドを使用します。

```
# vdo disableCompression --name=my-vdo
```

- もしくは、新規ボリュームの作成時に、**--compression=disabled** オプションを **vdo create** コマンドに指定して実行すると、圧縮を無効にできます。

2.9. VDO ボリュームのサイズの拡大

VDO ボリュームの物理サイズを増やして、基となるストレージの容量をさらに利用したり、ボリュームの論理サイズを大きくして、ボリュームが多くの領域を使用できるようにできます。

2.9.1. VDO ボリュームの物理サイズおよび論理サイズ

VDO は、物理サイズ、利用可能な物理サイズ、および論理サイズを次の方法で利用します。

物理サイズ

これは、基礎となるブロックデバイスと同じサイズです。VDO は、以下の目的でこのストレージを使用します。

- 重複排除および圧縮される可能性があるユーザーデータ
- UDS インデックスなどの VDO メタデータ

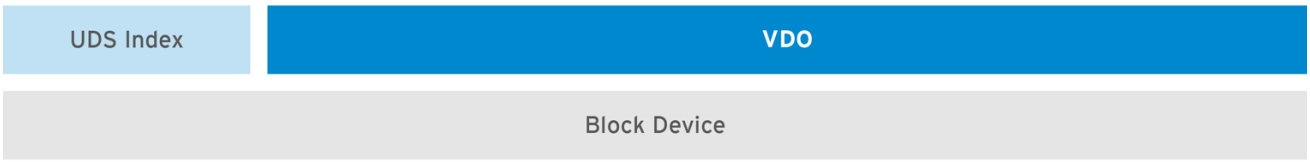
利用可能な物理サイズ

これは、VDO がユーザーデータに使用できる物理サイズの一部です。
これは、メタデータのサイズを引いた物理サイズと同等で、指定のスラブサイズでボリュームをスラブに分割した後の残りを引いたものと同じです。

論理サイズ

これは、VDO ボリュームがアプリケーションに提示するプロビジョニングされたサイズです。通常、これは利用可能な物理サイズよりも大きくなります。**--vdoLogicalSize** オプションを指定しないと、論理ボリュームのプロビジョニングが **1:1** の比率にプロビジョニングされます。たとえば、VDO ボリュームが 20 GB ブロックデバイスの上に置かれている場合は、2.5 GB が UDS インデックス用に予約されます (デフォルトのインデックスサイズが使用される場合)。残りの 17.5 GB は、VDO メタデータおよびユーザーデータに提供されます。そのため、消費する利用可能なストレージは 17.5 GB を超えません。実際の VDO ボリュームを設定するメタデータにより、これよりも少なくなる可能性があります。
VDO は現在、絶対最大論理サイズ 4PB の物理ボリュームの最大 254 倍の論理サイズに対応します。

図2.3 VDO ディスク組織



この図では、VDO で重複排除したストレージターゲットがブロックデバイス上に完全に配置されています。つまり、VDO ボリュームの物理サイズは、基礎となるブロックデバイスと同じサイズになります。

関連情報

- さまざまなサイズのブロックデバイスに必要なストレージ VDO メタデータのサイズは、「物理サイズ別の VDO 要件の例」を参照してください。

2.9.2. VDO でのシンプロビジョニング

VDO は、シンプロビジョニングされたブロックストレージターゲットです。VDO ボリュームが使用する物理領域のサイズは、ストレージのユーザーに示されるボリュームのサイズとは異なる可能性があります。この相違を活用して、ストレージのコストを削減できます。

容量不足の条件

書き込んだデータが、予想される最適化率に到達できない場合は、ストレージ領域が予想外に不足しないように注意してください。

論理ブロック (仮想ストレージ) の数が物理ブロック (実際のストレージ) の数を超えると、ファイルシステムおよびアプリケーションで領域が予想外に不足する可能性があります。このため、VDO を使用するストレージシステムは、VDO ボリュームの空きプールのサイズを監視する方法で提供する必要があります。

vdostats ユーティリティーを使用すると、この空きプールのサイズを確認できます。このユーティリティーのデフォルト出力には、Linux の **df** ユーティリティーと同様の形式で稼働しているすべての VDO ボリュームの情報が記載されます。以下に例を示します。

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

VDO ボリュームの物理ストレージ領域が不足しそうになると、VDO は、システムログに、以下のような警告を出力します。

```
Oct  2 17:13:39 system lvm[13863]: Monitoring VDO pool vdo-name.
Oct  2 17:27:39 system lvm[13863]: WARNING: VDO pool vdo-name is now 80.69% full.
Oct  2 17:28:19 system lvm[13863]: WARNING: VDO pool vdo-name is now 85.25% full.
Oct  2 17:29:39 system lvm[13863]: WARNING: VDO pool vdo-name is now 90.64% full.
Oct  2 17:30:29 system lvm[13863]: WARNING: VDO pool vdo-name is now 96.07% full.
```

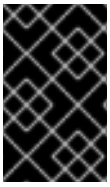
注記

この警告メッセージは、**lvm2-monitor** サービスが実行している場合に限り表示されます。これは、デフォルトで有効になっています。

容量不足の状況を防ぐ方法

空きプールのサイズが特定のレベルを下回る場合は、以下を行うことができます。

- データの削除。これにより、削除したデータが重複していないと、領域が回収されます。データを削除しても、破棄が行われないと領域を解放しません。
- 物理ストレージの追加



重要

VDO ボリュームで物理領域を監視し、領域不足を回避します。物理ブロックが不足すると、VDO ボリュームに最近書き込まれたデータや、未承認のデータが失われることがあります。

シンプロビジョニング、TRIM コマンド、および DISCARD コマンド

シンプロビジョニングによるストレージの削減から利益を得るには、データを削除するタイミングを物理ストレージ層が把握する必要があります。シンプロビジョニングされたストレージで動作するファイルシステムは、**TRIM** コマンドまたは **DISCARD** コマンドを送信して、論理ブロックが不要になったときにストレージシステムに通知します。

TRIM コマンドまたは **DISCARD** コマンドを送信する方法はいくつかあります。

- **discard** マウントオプションを使用すると、ブロックが削除されるたびに、ファイルシステムがそのコマンドを送信できます。
- **fstrim** などのユーティリティーを使用すると、制御された方法でコマンドを送信できます。このようなユーティリティーは、どの論理ブロックが使用されていないかを検出し、**TRIM** コマンドまたは **DISCARD** コマンドの形式でストレージシステムに情報を送信するようにファイルシステムに指示します。

未使用のブロックで **TRIM** または **DISCARD** を使用する必要は、VDO に特有のものではありません。シンプロビジョニングしたストレージシステムでも、同じ課題があります。

2.9.3. VDO ボリュームの論理サイズの拡大

この手順では、指定した VDO ボリュームの論理サイズを増やします。最初に、領域が不足しないサイズの論理が含まれる VDO ボリュームを作成できます。しばらくすると、データ消費の実際のレートを評価することができ、十分な場合には、VDO ボリュームの論理サイズを拡張して、削減して得られた領域を活用することができます。

VDO ボリュームの論理サイズを縮小することはできません。

手順

- 論理サイズを拡張するには、次のコマンドを実行します。

```
# vdo growLogical --name=my-vdo \
                  --vdoLogicalSize=new-logical-size
```

論理サイズが増大すると、VDO は新しいサイズのボリュームの上にデバイスまたはファイルシステムに通知します。

2.9.4. VDO ボリュームの物理サイズの拡大

この手順では、VDO ボリュームに利用できる物理ストレージの量を増やします。

このように VDO ボリュームを縮小することはできません。

前提条件

- 基となるブロックデバイスのサイズが、VDO ボリュームの現在の物理サイズよりも大きい。そうでない場合は、デバイスのサイズを大きくできます。正確な手順は、デバイスの種類によって異なります。たとえば、MBR パーティションまたは GPT パーティションのサイズ変更は、**ストレージデバイスの管理**の パーティションの使用 を参照してください。

手順

- 新しい物理ストレージ領域を VDO ボリュームに追加します。

```
# vdo growPhysical --name=my-vdo
```

2.10. VDO ボリュームの削除

システムで既存の VDO ボリュームを削除できます。

2.10.1. 作業中の VDO ボリュームの削除

この手順では、VDO ボリュームと、関連する UDS インデックスを削除します。

手順

1. ファイルシステムのマウントを解除し、VDO ボリュームでストレージを使用しているアプリケーションを停止します。
2. システムから VDO ボリュームを削除するには、次のコマンドを使用します。

```
# vdo remove --name=my-vdo
```

2.10.2. 作成に失敗した VDO ボリュームの削除

この手順では、中間状態で VDO ボリュームをクリーンアップします。ボリュームの作成時に障害が発生した場合、ボリュームは中間状態になります。たとえば、以下のような場合に発生する可能性があります。

- システムのクラッシュ
- 停電
- 管理者が、実行中の **vdo create** コマンドに割り込み

手順

- クリーンアップを行う場合は、**--force** オプションを使用して、作成に失敗したボリュームを削除します。

```
# vdo remove --force --name=my-vdo
```

ボリュームの作成に失敗して、管理者がシステム設定を変更して競合を発生させたため、**--force** オプションが必要となります。

--force オプションを指定しないと、**vdo remove** コマンドが失敗して、以下のメッセージが表示されます。

```
[...]
A previous operation failed.
Recovery from the failure either failed or was interrupted.
Add '--force' to 'remove' to perform the following cleanup.
Steps to clean up VDO my-vdo:
umount -f /dev/mapper/my-vdo
udevadm settle
dmsetup remove my-vdo
vdo: ERROR - VDO volume my-vdo previous operation (create) is incomplete
```

2.11. 関連情報

- **Ansible** ツールを使用することで、VDO デプロイメントと管理を自動化できます。詳細は、次を参照してください。
  - Ansible ドキュメント -https://docs.ansible.com/
  - VDO Ansible モバイルドキュメンテーション - https://docs.ansible.com/ansible/latest/modules/vdo_module.html

第3章 VDO 領域削減のテスト

一連のテストを実施して、VDO を使用して削減できるストレージ容量を判断できます。

前提条件

- 1 つ以上の物理ブロックデバイスが使用できる。
- ターゲットブロックデバイスが 512 GiB を超える。
- VDO がインストールされている。

3.1. VDO をテストする目的および結果

Red Hat が提供する VDO テストは、既存のストレージデバイスへの VDO の統合を評価できます。これは、社内の評価作業を置き換えるのではなく、増強することを目的としています。

テスト結果は、Red Hat エンジニアが特定のストレージ環境で VDO の動作を理解するのに役立ちます。OEM (Original equipment manufacturer) は、重複排除および圧縮可能なデバイスを設計する方法と、OEM のお客様がそのデバイスに合わせてアプリケーションを調整する方法を学ぶことができます。

ゴール

- テストデバイスから最適な応答を排除する設定を特定します。
- 製品の設定ミスを回避するために、基本的なチューニングパラメーターを説明します。
- パフォーマンス結果への参照を作成して、実際のユースケースと比較します。
- さまざまなワークロードがパフォーマンスやデータ効率にどのように影響するかを特定します。
- VDO 実装により、市場投入までの時間を短縮します。

テスト計画およびテスト条件

VDO テストは、VDO を最も現実的に評価できる条件を提供します。テスト手順やパラメーターを変更すると、結果が無効になる可能性があります。Red Hat セールスエンジニアは、テスト計画を変更する際のガイドを提供できます。

効果的なテスト計画を立てるには、VDO アーキテクチャーを調べて、次の項目を確認する必要があります。

- 高負荷環境におけるパフォーマンス
- エンドユーザーアプリケーションのパフォーマンスチューニング用 VDO の設定可能なプロパティー
- VDO がネイティブ 4 KiB ブロックデバイスであることの影響
- 重複排除と圧縮のアクセスパターンとディストリビューションへの応答
- 特定のアプリケーションのコスト、容量、パフォーマンスの比較

3.2. VDO でのシンプロビジョニング

VDO は、シンプロビジョニングされたブロックストレージターゲットです。VDO ボリュームが使用する物理領域のサイズは、ストレージのユーザーに示されるボリュームのサイズとは異なる可能性があります。この相違を活用して、ストレージのコストを削減できます。

容量不足の条件

書き込んだデータが、予想される最適化率に到達できない場合は、ストレージ領域が予想外に不足しないように注意してください。

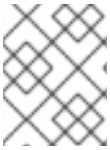
論理ブロック (仮想ストレージ) の数が物理ブロック (実際のストレージ) の数を超えると、ファイルシステムおよびアプリケーションで領域が予想外に不足する可能性があります。このため、VDO を使用するストレージシステムは、VDO ボリュームの空きプールのサイズを監視する方法で提供する必要があります。

vdostats ユーティリティーを使用すると、この空きプールのサイズを確認できます。このユーティリティーのデフォルト出力には、Linux の **df** ユーティリティーと同様の形式で稼働しているすべての VDO ボリュームの情報が記載されます。以下に例を示します。

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

VDO ボリュームの物理ストレージ領域が不足しそうになると、VDO は、システムログに、以下のような警告を出力します。

```
Oct  2 17:13:39 system lvm[13863]: Monitoring VDO pool vdo-name.
Oct  2 17:27:39 system lvm[13863]: WARNING: VDO pool vdo-name is now 80.69% full.
Oct  2 17:28:19 system lvm[13863]: WARNING: VDO pool vdo-name is now 85.25% full.
Oct  2 17:29:39 system lvm[13863]: WARNING: VDO pool vdo-name is now 90.64% full.
Oct  2 17:30:29 system lvm[13863]: WARNING: VDO pool vdo-name is now 96.07% full.
```



注記

この警告メッセージは、**lvm2-monitor** サービスが実行している場合に限り表示されます。これは、デフォルトで有効になっています。

容量不足の状況を防ぐ方法

空きプールのサイズが特定のレベルを下回る場合は、以下を行うことができます。

- データの削除。これにより、削除したデータが重複していないと、領域が回収されます。データを削除しても、破棄が行われないと領域を解放しません。
- 物理ストレージの追加

重要

VDO ボリュームで物理領域を監視し、領域不足を回避します。物理ブロックが不足すると、VDO ボリュームに最近書き込まれたデータや、未承認のデータが失われることがあります。

シンプロビジョニング、TRIM コマンド、および DISCARD コマンド

シンプロビジョニングによるストレージの削減から利益を得るには、データを削除するタイミングを物

理ストレージ層が把握する必要があります。シンプロビジョニングされたストレージで動作するファイルシステムは、**TRIM** コマンドまたは **DISCARD** コマンドを送信して、論理ブロックが不要になったときにストレージシステムに通知します。

TRIM コマンドまたは **DISCARD** コマンドを送信する方法はいくつかあります。

- **discard** マウントオプションを使用すると、ブロックが削除されるたびに、ファイルシステムがそのコマンドを送信できます。
- **fstrim** などのユーティリティーを使用すると、制御された方法でコマンドを送信できます。このようなユーティリティーは、どの論理ブロックが使用されていないかを検出し、**TRIM** コマンドまたは **DISCARD** コマンドの形式でストレージシステムに情報を送信するようにファイルシステムに指示します。

未使用のブロックで **TRIM** または **DISCARD** を使用する必要は、VDO に特有のものではありません。シンプロビジョニングしたストレージシステムでも、同じ課題があります。

3.3. 各 VDO テストの前に記録する情報

テスト環境を完全に把握できるように、各テストの開始時に次の情報を記録する必要があります。**sosreport** ユーティリティーを使用すると、必要な多くの情報を取得できます。

必要な情報

- 使用される Linux ビルド (カーネルビルド番号を含む)
- **rpm -qa** コマンドから取得した、インストールされているパッケージの完全なリスト
- 完全なシステム仕様
  - CPU の種類および数 (**/proc/cpuinfo** ファイルで利用可能)
  - 設置したメモリーと、rase OS の実行後に使用可能な量 (**/proc/meminfo** ファイルで利用可能)
  - 使用されているドライブコントローラーの種類
  - 使用されているディスクの種類と数量
- 実行中のプロセスの完全なリスト (**ps aux** コマンドから取得した、または同様のリスト)
- VDO で使用するために作成した物理ボリュームおよびボリュームグループの名前 (**pvs** コマンドおよび **vgs** コマンド)
- VDO ボリュームをフォーマットするときに使用されるファイルシステム (存在する場合)
- マウントしたディレクトリーの権限
- **/etc/vdoconf.yaml** ファイルの内容
- VDO ファイルの場所

3.4. VDO テストボリュームの作成

この手順では、テスト目的で、512 GiB の物理ボリュームに論理サイズ 1 TiB の VDO ボリュームを作成します。

手順

1. VDO ボリュームを作成します。

```
# vdo create --name=vdo-test \
        --device=/dev/sdb \
        --vdoLogicalSize=1T \
        --writePolicy=policy \
        --verbose
```

   - **/dev/sdb** を、ブロックデバイスへのパスに置き換えます。
   - 非同期ストレージで VDO の **async** モードをテストするには、**--writePolicy=async** オプションで非同期ボリュームを作成します。
   - 同期ストレージで VDO の **sync** モードをテストするには、**--writePolicy=sync** オプションを使用して同期ボリュームを作成します。

2. XFS ファイルシステムまたは ext4 ファイルシステムで新しいボリュームをフォーマットします。
   - XFS の場合:

```
# mkfs.xfs -K /dev/mapper/vdo-test
```

   - ext4 の場合:

```
# mkfs.ext4 -E nodiscard /dev/mapper/vdo-test
```

3. フォーマット済みのボリュームをマウントします。

```
# mkdir /mnt/vdo-test

# mount /dev/mapper/vdo-test /mnt/vdo-test && \
  chmod a+rwx /mnt/vdo-test
```

3.5. VDO テストボリュームのテスト

この手順では、VDO テストボリュームの読み取りと書き込みが機能するかどうかをテストします。

前提条件

- 新たに作成した VDO テストボリュームがマウントされている。詳細は、「VDO テストボリュームの作成」 を参照してください。

手順

1. 32 GiB のランダムデータを VDO ボリュームに書き込みます。

```
$ dd if=/dev/urandom of=/mnt/vdo-test/testfile bs=4096 count=8388608
```

2. VDO ボリュームからデータを読み込み、別のボリュームに書き込みます。

```
$ dd if=/mnt/vdo-test/testfile of=another-location/testfile bs=4096
```

- **another-location** を、VDO テストボリュームにない書き込みアクセス権があるディレクトリーに置き換えます。たとえば、ホームディレクトリーを使用できます。

3. 2 つのファイルを比較します。

```
$ diff --report-identical-files /mnt/vdo-test/testfile another-location/testfile
```

このコマンドは、ファイルが同じであることを報告する必要があります。

4. VDO ボリュームの新しい場所にファイルをコピーします。

```
$ dd if=another-location/testfile of=/mnt/vdo-test/testfile2 bs=4096
```

5. 3 番目のファイルと 2 番目のファイルを比較します。

```
$ diff --report-identical-files /mnt/vdo-test/testfile2 another-location/testfile
```

このコマンドは、ファイルが同じであることを報告する必要があります。

クリーンアップ手順

- 「VDO テストボリュームのクリーンアップ」 の説明に従って、VDO テストボリュームを削除します。

3.6. VDO テストボリュームのクリーンアップ

この手順では、システムから VDO 効率をテストするのに使用する VDO ボリュームを削除します。

前提条件

- VDO テストボリュームがマウントされている。

手順

1. VDO ボリュームに作成されたファイルシステムをアンマウントします。

```
# umount /mnt/vdo-test
```

2. システムから VDO テストボリュームを削除します。

```
# vdo remove --name=vdo-test
```

検証手順

- ボリュームが削除されたことを確認します。

```
# vdo list --all | grep vdo-test
```

このコマンドは、VDO テストパーティションのリストを表示しません。

3.7. VDO 重複排除の測定

この手順では、VDO テストボリュームで VDO データの重複排除の効率をテストします。

前提条件

- 新たに作成した VDO テストボリュームがマウントされている。詳細は、「VDO テストボリュームの作成」 を参照してください。

手順

1. テスト結果を記録できるテーブルを準備します。

統計	ベアファイルシステム	シード後	10 回コピーした後
使用されるファイルシステムのサイズ			
使用されている VDO データ			
使用されている VDO 論理			

2. VDO ボリュームにディレクトリーを 10 個作成して、テストデータセットのコピーを 10 個保持します。

```
$ mkdir /mnt/vdo-test/vdo{01..10}
```

3. ファイルシステムにより報告されるディスク使用量を調べます。

```
$ df --human-readable /mnt/vdo-test
```

例3.1 ディスク使用率

```
Filesystem            Size  Used Avail Use% Mounted on
/dev/mapper/vdo-test  1.5T  198M  1.4T   1% /mnt/vdo-test
```

4. 以下の値を記録します。

```
# vdostats --verbose | grep "blocks used"
```

例3.2 使用されるブロック

```
data blocks used               : 1090
overhead blocks used           : 538846
logical blocks used            : 6059434
```

- **data blocks used** の値は、VDO で実行している物理デバイスの最適化後のユーザーデータにより使用されているブロック数です。

- **logical blocks used** の値は、最適化前に使用したブロックの数になります。これは、測定の始点として使用されます。

5. VDO ボリュームにデータソースファイルを作成します。

```
$ dd if=/dev/urandom of=/mnt/vdo-test/sourcefile bs=4096 count=1048576

4294967296 bytes (4.3 GB) copied, 540.538 s, 7.9 MB/s
```

6. 使用されている物理ディスク容量を再確認します。

```
$ df --human-readable /mnt/vdo-test
```

例3.3 データソースファイルによるディスク使用状況

```
Filesystem            Size  Used Avail Use% Mounted on
/dev/mapper/vdo-test  1.5T  4.2G  1.4T   1% /mnt/vdo-test
```

```
# vdostats --verbose | grep "blocks used"
```

例3.4 データソースファイルで使用されているブロック

```
data blocks used               : 1050093  # Increased by 4GiB
overhead blocks used           : 538846   # Did not significantly change
logical blocks used            : 7108036  # Increased by 4GiB
```

このコマンドは、書き込まれたファイルのサイズに応じて、使用されているブロック数の増加を示しているはずです。

7. 10 個のサブディレクトリーのそれぞれにファイルをコピーします。

```
$ for i in {01..10}; do
  cp /mnt/vdo-test/sourcefile /mnt/vdo-test/vdo$i
  done
```

8. 使用されている物理ディスク容量を再確認します。

```
$ df -h /mnt/vdo-test
```

例3.5 ファイルをコピーした後のディスク使用量

```
Filesystem            Size  Used Avail Use% Mounted on
/dev/mapper/vdo-test  1.5T   45G  1.3T   4% /mnt/vdo-test
```

```
# vdostats --verbose | grep "blocks used"
```

例3.6 ファイルをコピーした後に使用されたブロック

```
data blocks used        : 1050836   # Increased by 3 MiB
overhead blocks used    : 538846
logical blocks used     : 17594127  # Increased by 41 GiB
```

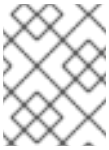
data blocks used は、ファイルシステムのジャーナリングとメタデータによるわずかな増加を除いて、以前のリスト表示の結果と類似している必要があります。

9. テストデータを書き込む前に見つかった値から、ファイルシステムが使用する領域の値を減算します。これは、ファイルシステムの観点からこのテストで使用される領域の量です。

10. 記録された統計で節約する容量を確認します。

例3.7 記録値

統計	ベアファイルシステム	シード後	10 回コピーした後
使用されるファイルシステムのサイズ	198 MiB	4.2 GiB	45 GiB
使用されている VDO データ	4 MiB	4.1 GiB	4.1 GiB
使用されている VDO 論理	23.6 GiB **(1.6 TiB フォーマットドライブのファイルシステムオーバーヘッド)**	27.8 GiB	68.7 GiB



注記

表では、値は MiB または GiB に変換されています。**vdostats** 出力のブロックのサイズは 4,096 B です。

クリーンアップ手順

- 「VDO テストボリュームのクリーンアップ」 の説明に従って、VDO テストボリュームを削除します。

3.8. VDO 圧縮の測定

この手順では、VDO テストボリュームで VDO データ圧縮の効率をテストします。

前提条件

- 新たに作成した VDO テストボリュームがマウントされている。詳細は、「VDO テストボリュームの作成」 を参照してください。

手順

1. VDO テストボリュームで重複排除を無効にして圧縮を有効にします。

```
# vdo disableDeduplication --name=vdo-test
# vdo enableCompression --name=vdo-test
```

2. VDO ボリュームを同期して、未完了の圧縮を完了します。

```
# sync && dmsetup message vdo-test 0 sync-dedupe
```

3. 転送する前に VDO 統計を調べます。

```
# vdostats --verbose | grep "blocks used"
```

data blocks used と **logical blocks used** の値を書き留めます。

4. VDO は、ファイルシステムのオーバーヘッドと、実際のユーザーデータを最適化します。**logical blocks used** から **data blocks used** を引いて、空のファイルシステムの圧縮により削減された 4 KiB ブロックの数を計算します。

5. **/lib** ディレクトリーの内容を VDO ボリュームにコピーします。

```
# cp --verbose --recursive /lib /mnt/vdo-test

...
sent 152508960 bytes  received 60448 bytes  61027763.20 bytes/sec
total size is 152293104  speedup is 1.00
```

コピーしたデータの合計サイズを記録します。

6. Linux キャッシュと VDO ボリュームを同期します。

```
# sync && dmsetup message vdo-test 0 sync-dedupe
```

7. VDO 統計を再検証します。

```
# vdostats --verbose | grep "blocks used"
```

logical blocks used と **data blocks used** を確認します。

8. 次の式を使用して、圧縮により削減されたバイト量を計算します。

```
saved_bytes = (logical_blocks_used - data_blocks_used) * 4096
```

クリーンアップ手順

- 「VDO テストボリュームのクリーンアップ」 の説明に従って、VDO テストボリュームを削除します。

3.9. 削減された VDO 領域の測定

この手順では、VDO テストボリュームで VDO データの重複排除および圧縮の複合効率をテストします。

手順

1. 「VDO テストボリュームの作成」 の説明に従って、VDO ボリュームを作成してマウントします。
2. 同じボリュームで VDO 重複排除の測定 と VDO 圧縮の測定 で説明されているテストを削除せずに行います。**vdostats** の出力で削減する領域への変更を調べます。
3. 用意したデータセットを試します。

3.10. VDO での TRIM および DISCARD の効果のテスト

この手順では、**TRIM** コマンドおよび **DISCARD** コマンドが、VDO テストボリュームで削除したファイルからブロックを適切に解放するかどうかをテストします。これは、discard (破棄) が VDO に、その領域が使用されなくなったことを通知することを示しています。

前提条件

- 新たに作成した VDO テストボリュームがマウントされている。詳細は、「VDO テストボリュームの作成」 を参照してください。

手順

1. テスト結果を記録できるテーブルを準備します。

手順	使用されているファイル領域 (MB)	使用されているデータブロック	使用されている論理ブロック
初期			
1 GiB ファイルの追加			
fstrim の実行			
1 GiB ファイルの削除			
fstrim の実行			

2. ファイルシステムをトリミングして、不要なブロックを削除します。

```
# fstrim /mnt/vdo-test
```

コマンドには時間がかかる場合があります。

3. ファイルシステムで初期領域の使用量を記録します。

```
$ df -m /mnt/vdo-test
```

4. VDO ボリュームが使用する物理データブロックおよび論理データブロックの数を確認します。

```
# vdostats --verbose | grep "blocks used"
```

5. VDO ボリュームに重複しないデータを含む 1 GiB ファイルを作成します。

```
$ dd if=/dev/urandom of=/mnt/vdo-test/file bs=1M count=1K
```

6. 領域の使用量を再度記録します。

```
$ df -m /mnt/vdo-test

# vdostats --verbose | grep "blocks used"
```

ファイルシステムは、追加の 1 GiB を使用する必要があります。**data blocks used** と **logical blocks used** の値も同じように増加するはずです。

7. ファイルシステムを再度トリミングします。

```
# fstrim /mnt/vdo-test
```

8. 領域使用量を再度調べて、トリミングが物理ボリュームの使用量に影響がないことを確認します。

```
$ df -m /mnt/vdo-test

# vdostats --verbose | grep "blocks used"
```

9. 1 GiB ファイルを削除します。

```
$ rm /mnt/vdo-test/file
```

10. 領域の使用量を再度確認して記録します。

```
$ df -m /mnt/vdo-test

# vdostats --verbose | grep "blocks used"
```

ファイルシステムは、ファイルが削除されたことを認識していますが、ファイルの削除が基礎となるストレージに通知されていないため、物理ブロックまたは論理ブロックの数に変更がありません。

11. ファイルシステムを再度トリミングします。

```
# fstrim /mnt/vdo-test
```

12. 領域の使用量を再度確認して記録します。

```
$ df -m /mnt/vdo-test

# vdostats --verbose | grep "blocks used"
```

fstrim ユーティリティーは、ファイルシステムの空きブロックを検索し、未使用のアドレスの VDO ボリュームに **TRIM** コマンドを送信します。これにより、関連する論理ブロックが解放されます。VDO は **TRIM** コマンドを処理し、基礎となる物理ブロックを解放します。

関連情報

- **TRIM** コマンド、**DISCARD** コマンド、**fstrim** ユーティリティー、および **discard** マウントオプションの詳細は、5章*未使用ブロックの破棄* を参照してください。

第4章 VDO パフォーマンスのテスト

一連のテストを実行して、VDO パフォーマンスを測定し、VDO でシステムのパフォーマンスプロファイルを取得し、VDO で適切に動作するアプリケーションを特定できます。

前提条件

- 1 つ以上の Linux 物理ブロックデバイスが使用できる。
- ターゲットブロックデバイス (例: **/dev/sdb**) が 512 GiB を超える。
- 柔軟な I/O テスター (**fio**) がインストールされている。
- VDO がインストールされている。

4.1. VDO パフォーマンステスト用に環境の準備

VDO パフォーマンスをテストする前に、テスト時に使用するホストシステムの設定、VDO 設定、およびワークロードを考慮する必要があります。この選択肢は、領域効率、帯域幅、およびレイテンシーのベンチマークに影響します。

あるテストが別のテストの結果に影響を及ぼさないようにするには、各テストの反復ごとに新しい VDO ボリュームを作成する必要があります。

4.1.1. VDO パフォーマンスをテストする前に考慮すべき点

以下の条件と設定は、VDO テストの結果に影響します。

システムの設定

- 利用可能な CPU コアの数およびタイプ。この情報は、**taskset** ユーティリティーを使用してリスト表示できます。
- 利用可能なメモリーと、インストールされているメモリーの合計
- ストレージデバイスの設定
- アクティブディスクスケジューラー
- Linux カーネルバージョン
- インストールされているパッケージ

VDO の設定

- パーティション設定スキーム
- VDO ボリュームで使用しているファイルシステム
- VDO ボリュームに割り当てられた物理ストレージのサイズ
- 作成した論理 VDO ボリュームのサイズ
- UDS インデックス sparse または dense

- メモリーサイズの UDS インデックス
- VDO スレッド設定

ワークロード

- テストデータを生成するのに使用するツールの種類
- 同時クライアントの数
- 書き込まれたデータで重複する 4 KiB ブロックの量
- 読み書きのパターン
- ワーキングセットのサイズ

4.1.2. VDO 読み込みパフォーマンスのテストに関する特別な考慮事項

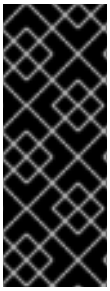
VDO 読み込みパフォーマンスをテストする前に、以下の追加要素を考慮する必要があります。

- 4 KiB ブロックが書き込まれていないと、VDO はストレージから読み込みを行わず、ゼロブロックですぐに応答します。
- 4 KiB ブロックが書き込まれていても、含まれているのがすべてのゼロの場合、VDO はストレージから読み込みを行わず、ゼロブロックですぐに応答します。

この動作により、読み込むデータがない場合に、読み込みのパフォーマンスが非常に高速になります。このため、読み取りテストは実際のデータでボリュームを事前に入力する必要があります。

4.1.3. VDO パフォーマンスをテストするシステムの準備

この手順では、テスト中に VDO パフォーマンスを最適化するためのシステム設定を行います。



重要

特定のテストに挙げられている境界を超えたテストを行うと、異常な結果により、テスト時間が損失する可能性があります。

たとえば、VDO テストは、100 GiB アドレス範囲で無作為な読み込みを実行するテストを記述します。500 GiB のワーキングセットをテストするには、VDO ブロックマップキャッシュに割り当てられる RAM の容量を適宜増やす必要があります。

手順

1. CPU が最も高いパフォーマンス設定で実行していることを確認します。
2. 可能であれば、BIOS 設定または Linux の **cpupower** ユーティリティーを使用して CPU 周波数のスケーリングを無効にします。
3. 可能であれば、CPU の動的プロセッサーの周波数調整 (Turbo Boost または Turbo Core) を有効にします。この機能により、テスト結果に多少のばらつきが生じますが、全体的なパフォーマンスは向上します。

4. ファイルシステムは、パフォーマンスに固有の影響を与える可能性があります。多くの場合、パフォーマンス測定値にずれが生じ、結果に対する VDO の影響を特定することが困難になります。
   妥当な場合には、raw ブロックデバイスのパフォーマンスを測定します。これができない場合は、VDO がターゲット実装で使用するファイルシステムを使用してデバイスをフォーマットします。

4.2. パフォーマンステスト用の VDO ボリュームの作成

この手順では、VDO パフォーマンスをテストするために、512 GiB の物理ボリュームに論理サイズ 1 TiB の VDO ボリュームを作成します。

手順

- VDO ボリュームを作成します。

```
# vdo create --name=vdo-test \
        --device=/dev/sdb \
        --vdoLogicalSize=1T \
        --writePolicy=policy \
        --verbose
```

  - **/dev/sdb** を、ブロックデバイスへのパスに置き換えます。
  - 非同期ストレージで VDO の **async** モードをテストするには、**--writePolicy=async** オプションで非同期ボリュームを作成します。
  - 同期ストレージで VDO の **sync** モードをテストするには、**--writePolicy=sync** オプションを使用して同期ボリュームを作成します。

4.3. VDO パフォーマンステストボリュームのクリーンアップ

この手順では、システムから VDO パフォーマンスをテストするのに使用する VDO ボリュームを削除します。

前提条件

- システムに VDO テストボリュームが存在する。

手順

- システムから VDO テストボリュームを削除します。

```
# vdo remove --name=vdo-test
```

検証手順

- ボリュームが削除されたことを確認します。

```
# vdo list --all | grep vdo-test
```

  このコマンドは、VDO テストパーティションのリストを表示しません。

4.4. VDO パフォーマンスに対する I/O 深度の影響のテスト

これらのテストは、VDO 設定の最適なスループットと最小のレイテンシーを生成する I/O 深度を決定します。I/O 深度は、**fio** ツールが一度に送信する I/O 要求の数を表します。

VDO は 4 KiB セクターサイズを使用するため、このテストは 4 KiB I/O 操作および 1、8、16、32、64、128、256、512、および 1024 の I/O 深度で全領域テストを実行します。

4.4.1. VDO での連続の 100% 読み込みに対する I/O 深度効果のテスト

このテストは、異なる I/O 深度の値で VDO ボリュームで連続の 100% 読み込み操作を実行する方法を決定します。

手順

1. 新しい VDO ボリュームを作成します。
   詳細は、「パフォーマンステスト用の VDO ボリュームの作成」 を参照してください。

2. テストボリュームで書き込み **fio** ジョブを実行して、テストがアクセスする領域を事前に入力します。

```
# fio --rw=write \
      --bs=8M \
      --name=vdo \
      --filename=/dev/mapper/vdo-test \
      --ioengine=libaio \
      --thread \
      --direct=1 \
      --scramble_buffers=1
```

3. 連続の 100% 読み込みについて報告されたスループットおよびレイテンシーを記録します。

```
# for depth in 1 2 4 8 16 32 64 128 256 512 1024 2048; do
  fio --rw=read \
      --bs=4096 \
      --name=vdo \
      --filename=/dev/mapper/vdo-test \
      --ioengine=libaio \
      --numjobs=1 \
      --thread \
      --norandommap \
      --runtime=300 \
      --direct=1 \
      --iodepth=$depth \
      --scramble_buffers=1 \
      --offset=0 \
      --size=100g
  done
```

4. VDO テストボリュームを削除します。
   詳細は、「VDO パフォーマンステストボリュームのクリーンアップ」 を参照してください。

4.4.2. VDO での連続の 100% 書き込みに対する I/O 深度効果のテスト

このテストは、異なる I/O 深度値で、VDO ボリュームに対して連続した 100% 書き込み操作を実行する方法を決定します。

手順

1. 新しい VDO テストボリュームを作成します。
   詳細は、「パフォーマンステスト用の VDO ボリュームの作成」 を参照してください。
2. 書き込み **fio** ジョブを実行して、テストがアクセスする可能性のある領域を事前に入力します。

```
# fio --rw=write \
      --bs=8M \
      --name=vdo \
      --filename=/dev/mapper/vdo-test \
      --ioengine=libaio \
      --thread \
      --direct=1 \
      --scramble_buffers=1
```

3. 連続した 100% 書き込みについて報告されたスループットとレイテンシーを記録します。

```
# for depth in 1 2 4 8 16 32 64 128 256 512 1024 2048; do
  fio --rw=write \
      --bs=4096 \
      --name=vdo \
      --filename=/dev/mapper/vdo-test \
      --ioengine=libaio \
      --numjobs=1 \
      --thread \
      --norandommap \
      --runtime=300 \
      --direct=1 \
      --iodepth=$depth \
      --scramble_buffers=1 \
      --offset=0 \
      --size=100g
  done
```

4. VDO テストボリュームを削除します。
   詳細は、「VDO パフォーマンステストボリュームのクリーンアップ」 を参照してください。

4.4.3. VDO での無作為の 100% 読み込みに対する I/O 深度効果のテスト

このテストは、異なる I/O 深度値の VDO ボリュームで無作為の 100% 読み込み操作を実行する方法を決定します。

手順

1. 新しい VDO テストボリュームを作成します。
   詳細は、「パフォーマンステスト用の VDO ボリュームの作成」 を参照してください。
2. 書き込み **fio** ジョブを実行して、テストがアクセスする可能性のある領域を事前に入力します。

```
# fio --rw=write \
      --bs=8M \
      --name=vdo \
      --filename=/dev/mapper/vdo-test \
      --ioengine=libaio \
      --thread \
      --direct=1 \
      --scramble_buffers=1
```

3. 無作為の 100% 読み込みについて報告されたスループットとレイテンシーを記録します。

```
# for depth in 1 2 4 8 16 32 64 128 256 512 1024 2048; do
  fio --rw=randread \
      --bs=4096 \
      --name=vdo \
      --filename=/dev/mapper/vdo-test \
      --ioengine=libaio \
      --numjobs=1 \
      --thread \
      --norandommap \
      --runtime=300 \
      --direct=1 \
      --iodepth=$depth \
      --scramble_buffers=1 \
      --offset=0 \
      --size=100g
  done
```

4. VDO テストボリュームを削除します。
   詳細は、「VDO パフォーマンステストボリュームのクリーンアップ」 を参照してください。

4.4.4. VDO での無作為の 100% 書き込みに対する I/O 深度効果のテスト

このテストは、異なる I/O 深度値で VDO ボリュームに対して無作為な 100% 書き込み操作を実行する方法を決定します。

重要

各 I/O 深度テストの実行間で VDO ボリュームを再作成する必要があります。

手順

以下の手順を、I/O 深度値 1、2、4、8、16、32、64、128、256、512、1024、および 2048 に対して個別に実行します。

1. 新しい VDO テストボリュームを作成します。
   詳細は、「パフォーマンステスト用の VDO ボリュームの作成」 を参照してください。

2. 書き込み **fio** ジョブを実行して、テストがアクセスする可能性のある領域を事前に入力します。

```
# fio --rw=write \
      --bs=8M \
      --name=vdo \
```

```
    --filename=/dev/mapper/vdo-test \
    --ioengine=libaio \
    --thread \
    --direct=1 \
    --scramble_buffers=1
```

3. 無作為な 100% 書き込みについて報告されたスループットとレイテンシーを記録します。

```
# fio --rw=randwrite \
    --bs=4096 \
    --name=vdo \
    --filename=/dev/mapper/vdo-test \
    --ioengine=libaio \
    --numjobs=1 \
    --thread \
    --norandommap \
    --runtime=300 \
    --direct=1 \
    --iodepth=depth-value
    --scramble_buffers=1 \
    --offset=0 \
    --size=100g
  done
```

4. VDO テストボリュームを削除します。
   詳細は、「VDO パフォーマンステストボリュームのクリーンアップ」 を参照してください。

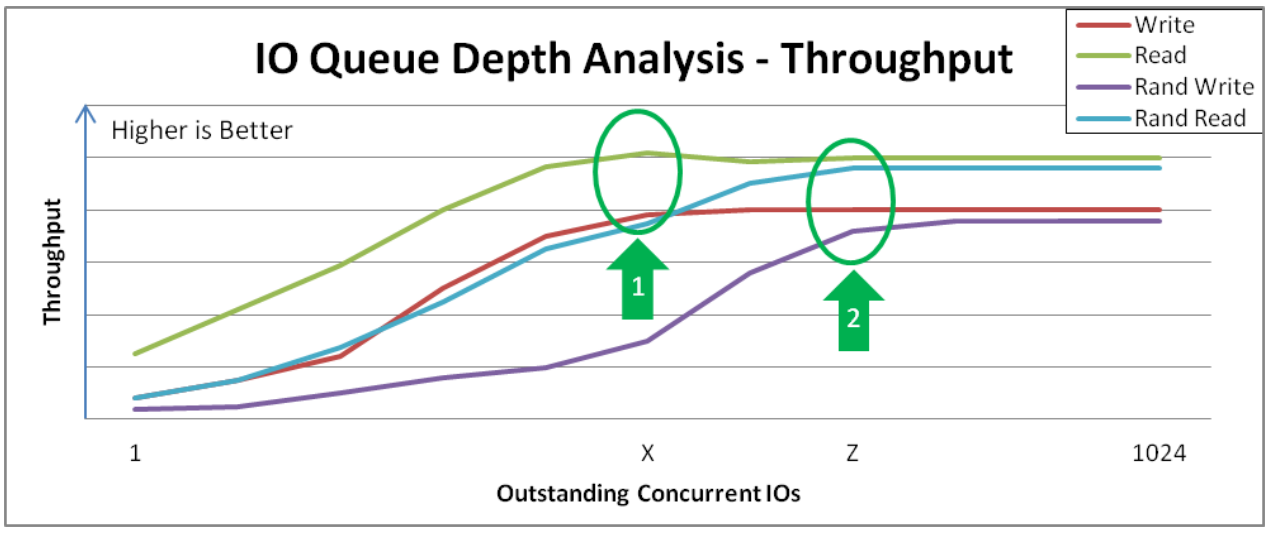
4.4.5. 異なる I/O 深度での VDO パフォーマンス分析

以下の例は、異なる I/O 深度値で記録された VDO スループットとレイテンシーを示しています。

I/O 深度が増えるとスループットの向上が低下する範囲と変曲点全体の動作を観察します。順次アクセスと無作為なアクセスのピークはおそらく異なる値になりますが、ピークはすべてのタイプのストレージ設定で異なる場合があります。

例4.1 I/O 深度分析

図4.1 VDO スループット分析



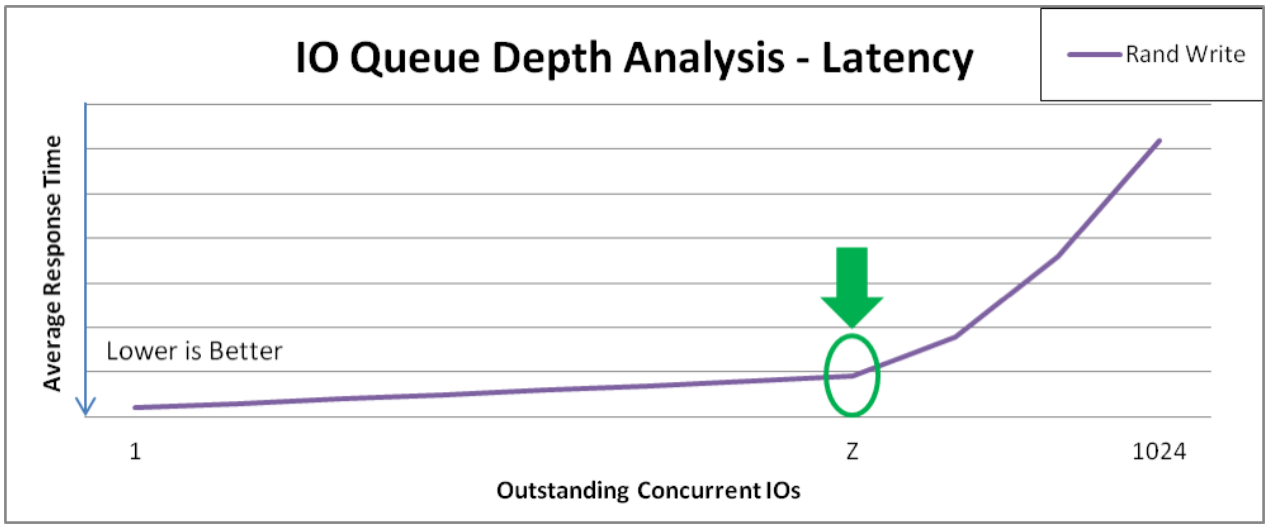
各パフォーマンス曲線の急な折れ曲がりに注意してください。

- マーカー 1 は、X 地点でピークシーケンススループットを識別します。この設定には、X よりも連続的な 4 KiB の I/O 深度による利点がありません。
- マーカー 2 は、Z 地点でピークの無作為な 4 KiB スループットを識別します。この設定には、Z よりも無作為な 4 KiB の I/O 深度による利点がありません。

X 地点および Z 地点の I/O 深度を超えると、帯域幅の上昇が減少し、平均要求のレイテンシーは、追加の I/O 要求ごとに 1:1 に増加します。

以下の図は、直前のグラフで曲線が急に折れ曲がった後の無作為な書き込みレイテンシーの例を示しています。このような地点でテストして、応答時間のペナルティが最小になる最大スループットを確認する必要があります。

図4.2 VDO レイテンシー分析



最適な I/O 深度

Z 地点は、最適な I/O 深度を示します。テスト計画は、Z と同等の I/O 深度で追加のデータを収集します。

4.5. VDO パフォーマンスにおける I/O リクエストサイズの影響のテスト

このテストを使用して、最適な I/O 深度で VDO のパフォーマンスを最適化するブロックサイズを指定できます。

テストでは、8 KiB から 1 MiB の範囲でさまざまなブロックサイズを使用して、固定 I/O 深度で全領域テストを実行します。

前提条件

- 最適な I/O 深度値を特定している。詳細は、「VDO パフォーマンスに対する I/O 深度の影響のテスト」 を参照してください。
  以下のテストでは、**optimal-depth** を、最適な I/O 深度値に置き換えます。

4.5.1. VDO での連続書き込みに対する I/O 要求サイズの影響のテスト

このテストは、異なる I/O 要求サイズで VDO ボリュームへの連続書き込み操作を実行する方法を決定します。

手順

1. 新しい VDO ボリュームを作成します。
   詳細は、「パフォーマンステスト用の VDO ボリュームの作成」 を参照してください。
2. テストボリュームで書き込み **fio** ジョブを実行して、テストがアクセスする領域を事前に入力します。

```
# fio --rw=write \
      --bs=8M \
      --name=vdo \
      --filename=/dev/mapper/vdo-test \
      --ioengine=libaio \
      --thread \
      --direct=1 \
      --scramble_buffers=1
```

3. 連続書き込みテストについて報告されたスループットとレイテンシーを記録します。

```
# for iosize in 4 8 16 32 64 128 256 512 1024; do
  fio --rw=write \
      --bs=${iosize}k \
      --name=vdo \
      --filename=/dev/mapper/vdo-test \
      --ioengine=libaio \
      --numjobs=1 \
      --thread \
      --norandommap \
      --runtime=300 \
      --direct=1 \
      --iodepth=optimal-depth \
      --scramble_buffers=1 \
      --offset=0 \
      --size=100g
  done
```

4. VDO テストボリュームを削除します。
   詳細は、「VDO パフォーマンステストボリュームのクリーンアップ」 を参照してください。

4.5.2. VDO での無作為な書き込みに対する I/O 要求サイズの影響のテスト

このテストは、異なる I/O 要求サイズで VDO ボリュームへの無作為な書き込み操作を実行する方法を決定します。

重要

各 I/O 要求サイズのテスト実行の間に VDO ボリュームを再作成する必要があります。

手順

I/O 要求サイズが **4k**、**8k**、**16k**、**32k**、**64k**、**128k**、**256k**、**512k**、および **1024k** の場合は、以下の手順を個別に実行します。

1. 新しい VDO ボリュームを作成します。
   詳細は、「パフォーマンステスト用の VDO ボリュームの作成」 を参照してください。

2. テストボリュームで書き込み **fio** ジョブを実行して、テストがアクセスする領域を事前に入力します。

```
# fio --rw=write \
      --bs=8M \
      --name=vdo \
      --filename=/dev/mapper/vdo-test \
      --ioengine=libaio \
      --thread \
      --direct=1 \
      --scramble_buffers=1
```

3. 無作為な書き込みテストについて報告されたスループットとレイテンシーを記録します。

```
# fio --rw=randwrite \
      --bs=request-size \
      --name=vdo \
      --filename=/dev/mapper/vdo-test \
      --ioengine=libaio \
      --numjobs=1 \
      --thread \
      --norandommap \
      --runtime=300 \
      --direct=1 \
      --iodepth=optimal-depth \
      --scramble_buffers=1 \
      --offset=0 \
      --size=100g
  done
```

4. VDO テストボリュームを削除します。
詳細は、「VDO パフォーマンステストボリュームのクリーンアップ」 を参照してください。

4.5.3. VDO での連続読み込みに対する I/O 要求サイズの影響のテスト

このテストは、異なる I/O 要求サイズで VDO ボリュームへの連続読み取り操作を実行する方法を決定します。

手順

1. 新しい VDO ボリュームを作成します。
詳細は、「パフォーマンステスト用の VDO ボリュームの作成」 を参照してください。

2. テストボリュームで書き込み **fio** ジョブを実行して、テストがアクセスする領域を事前に入力します。

```
# fio --rw=write \
      --bs=8M \
      --name=vdo \
      --filename=/dev/mapper/vdo-test \
      --ioengine=libaio \
      --thread \
      --direct=1 \
      --scramble_buffers=1
```

3. 連続読み取りテストについて報告されたスループットとレイテンシーを記録します。

```
# for iosize in 4 8 16 32 64 128 256 512 1024; do
  fio --rw=read \
      --bs=${iosize}k \
      --name=vdo \
      --filename=/dev/mapper/vdo-test \
      --ioengine=libaio \
      --numjobs=1 \
      --thread \
      --norandommap \
      --runtime=300 \
      --direct=1 \
      --iodepth=optimal-depth \
      --scramble_buffers=1 \
      --offset=0 \
      --size=100g
  done
```

4. VDO テストボリュームを削除します。
   詳細は、「VDO パフォーマンステストボリュームのクリーンアップ」 を参照してください。

4.5.4. VDO での無作為な読み込みに対する I/O 要求サイズの影響のテスト

このテストは、異なる I/O 要求サイズで VDO ボリュームへの無作為な読み取り操作を実行する方法を決定します。

手順

1. 新しい VDO ボリュームを作成します。
   詳細は、「パフォーマンステスト用の VDO ボリュームの作成」 を参照してください。

2. テストボリュームで書き込み **fio** ジョブを実行して、テストがアクセスする領域を事前に入力します。

```
# fio --rw=write \
      --bs=8M \
      --name=vdo \
      --filename=/dev/mapper/vdo-test \
      --ioengine=libaio \
      --thread \
      --direct=1 \
      --scramble_buffers=1
```

3. 無作為な読み取りテストで報告されたスループットとレイテンシーを記録します。

```
# for iosize in 4 8 16 32 64 128 256 512 1024; do
  fio --rw=read \
      --bs=${iosize}k \
      --name=vdo \
      --filename=/dev/mapper/vdo-test \
      --ioengine=libaio \
      --numjobs=1 \
      --thread \
```

```
    --norandommap \
    --runtime=300 \
    --direct=1 \
    --iodepth=optimal-depth \
    --scramble_buffers=1 \
    --offset=0 \
    --size=100g
done
```

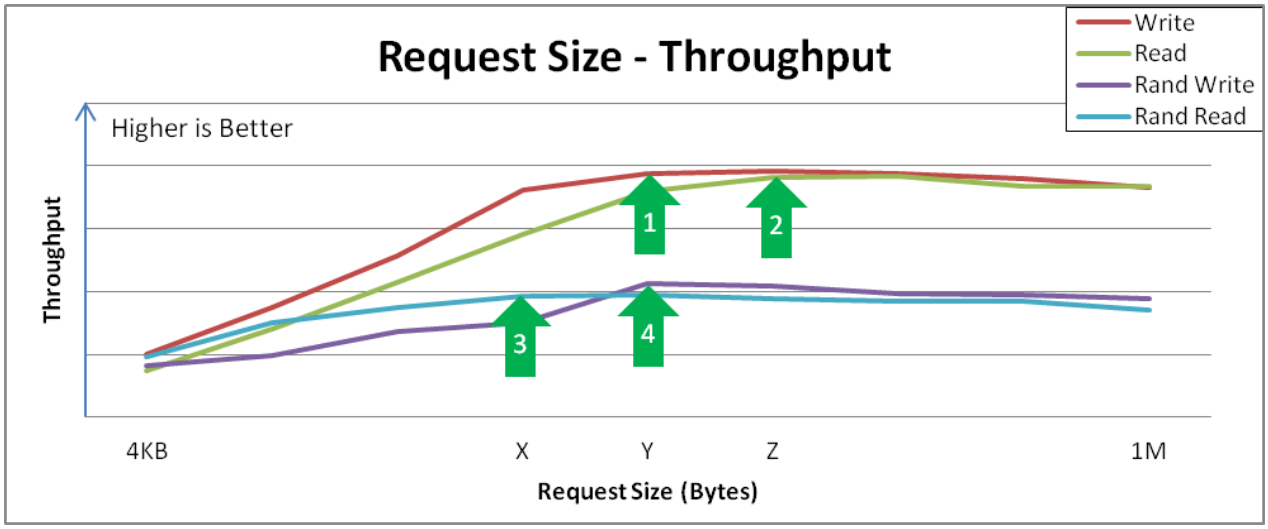
4. VDO テストボリュームを削除します。
   詳細は、「VDO パフォーマンステストボリュームのクリーンアップ」 を参照してください。

4.5.5. 異なる I/O 要求サイズでの VDO パフォーマンスの分析

以下の例は、異なる I/O 要求サイズで記録された VDO スループットとレイテンシーを示しています。

例4.2 I/O リクエストサイズ分析

図4.3 要求サイズ対スループットの分析と重要な変曲点



サンプルの結果を分析します。

- 連続書き込みは、要求サイズ **Y** でピークスループットに達します。
  この曲線は、設定可能なアプリケーション、特定の要求サイズによって自然に支配されるアプリケーションが、パフォーマンスをどのように認識できるかを示しています。多くの場合、4 KiB の I/O 操作では結合が有効になるため、要求サイズが大きいほどスループットが向上します。

- 連続読み込みは、**Z** 地点で同様のピークスループットに達します。
  これらのピーク後、I/O 操作が完了するまでの全体的なレイテンシーは、スループットを追加することなく増加します。このサイズを超える I/O 操作を受け入れないようにデバイスを調整する必要があります。

- 無作為な読み込みは、**X** 地点でピークスループットに達します。
  特定のデバイスは、要求サイズの無作為アクセスで、ほぼ連続的なスループット率を実現する場合がありますが、純粋な連続アクセスと異なる場合は、より多くのペナルティーを受けることになります。

- 無作為な書き込みは、**Y** 地点でピークスループットに達します。

無作為な書き込みは、重複排除デバイスの相互作用が最も多く、VDO は、特に要求サイズまたは I/O 深度が大きい場合に高いパフォーマンスを実現します。

4.6. VDO パフォーマンスにおける混合 I/O 負荷の影響のテスト

このテストでは、読み取りと書き込みが混合した I/O 負荷で VDO 設定がどのように動作するかを判断し、最適で無作為なキューの深さと、4 KB から 1 MB の要求サイズでの読み取りと書き込みの混合の影響を分析します。

この手順では、固定の I/O 深度で全領域テストを実行し、8 KB から 256 KB の範囲で可変ブロックサイズを設定し、0% から初めて 10 % 刻みで読み取り率を設定します。

前提条件

- 最適な I/O 深度値を特定している。詳細は、「VDO パフォーマンスに対する I/O 深度の影響のテスト」 を参照してください。
  以下の手順では、**optimal-depth** を。最適な I/O 深度値に置き換えます。

手順

1. 新しい VDO ボリュームを作成します。
   詳細は、「パフォーマンステスト用の VDO ボリュームの作成」 を参照してください。

2. テストボリュームで書き込み **fio** ジョブを実行して、テストがアクセスする領域を事前に入力します。

```
# fio --rw=write \
      --bs=8M \
      --name=vdo \
      --filename=/dev/mapper/vdo-test \
      --ioengine=libaio \
      --thread \
      --direct=1 \
      --scramble_buffers=1
```

3. 読み書き入力要因ついて報告されたスループットとレイテンシーを記録します。

```
# for readmix in 0 10 20 30 40 50 60 70 80 90 100; do
    for iosize in 4 8 16 32 64 128 256 512 1024; do
      fio --rw=rw \
          --rwmixread=$readmix \
          --bs=${iosize}k \
          --name=vdo \
          --filename=/dev/mapper/vdo-test \
          --ioengine=libaio \
          --numjobs=1 \
          --thread \
          --norandommap \
          --runtime=300 \
          --direct=0 \
          --iodepth=optimal-depth \
          --scramble_buffers=1 \
          --offset=0 \
```

```
        --size=100g
  done
done
```

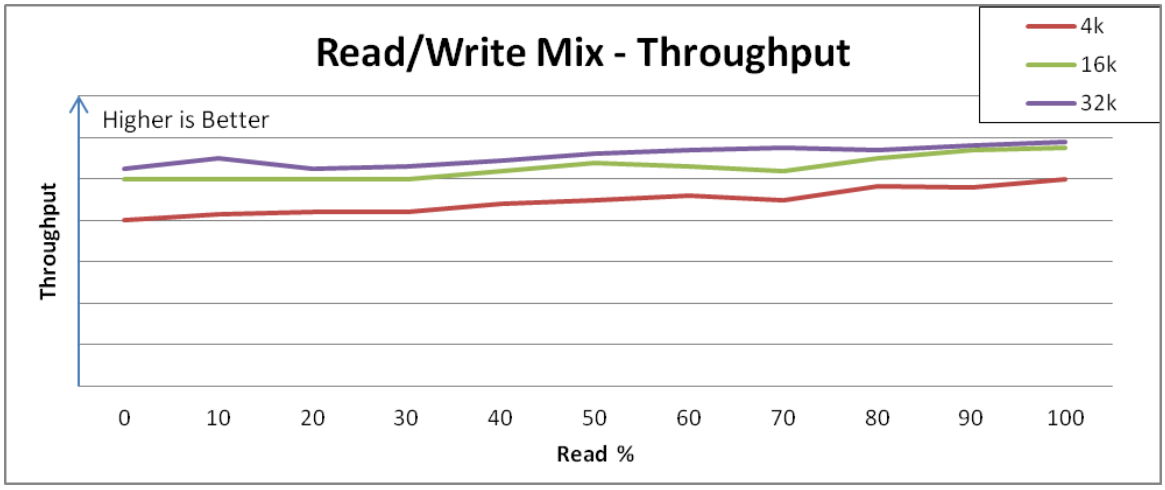
4. VDO テストボリュームを削除します。
   詳細は、「VDO パフォーマンステストボリュームのクリーンアップ」 を参照してください。

5. テスト結果をグラフで表示します。

例4.3 混合 I/O 負荷分析

以下の図は、VDO が混合 I/O 負荷にどのように応答するかの例を示しています。

図4.4 パフォーマンスは、さまざまな読み取りと書き込みの組み合わせ混合で一貫しています。



集約パフォーマンスと集約レイテンシーは、混合する書き込みと書き込みの範囲全体で比較的一貫性があり、より低い最大書き込みスループットからより高い最大書き込みスループットへと移ります。

この動作はストレージによって異なる場合がありますが、負荷が変化してもパフォーマンスが一貫していること、または特定の読み取りと書き込みの組み合わせを示すアプリケーションのパフォーマンス期待値を理解できることが重要となります。

注記

システムが同様の応答の一貫性を示さない場合は、設定が最適ではない可能性があります。この場合は、Red Hat セールスエンジニアにお問い合わせください。

4.7. VDO パフォーマンスに対するアプリケーション環境の影響のテスト

これらのテストは、実際のアプリケーションが混在する環境にデプロイされたときの VDO 設定の動作を決定します。予想される環境に関する詳細が分かっている場合は、それもテストしてください。

前提条件

- 設定で許容されるキューの深度を制限することを検討してください。

- 可能であれば、VDO パフォーマンスに最も有益なブロックサイズで要求を発行するようにアプリケーションを調整します。

手順

1. 新しい VDO ボリュームを作成します。
   詳細は、「パフォーマンステスト用の VDO ボリュームの作成」 を参照してください。

2. テストボリュームで書き込み **fio** ジョブを実行して、テストがアクセスする領域を事前に入力します。

```
# fio --rw=write \
      --bs=8M \
      --name=vdo \
      --filename=/dev/mapper/vdo-test \
      --ioengine=libaio \
      --thread \
      --direct=1 \
      --scramble_buffers=1
```

3. 読み書き入力要因ついて報告されたスループットとレイテンシーを記録します。

```
# for readmix in 20 50 80; do
    for iosize in 4 8 16 32 64 128 256 512 1024; do
      fio --rw=rw \
          --rwmixread=$readmix \
          --bsrange=4k-256k \
          --name=vdo \
          --filename=/dev/mapper/vdo-name \
          --ioengine=libaio \
          --numjobs=1 \
          --thread \
          --norandommap \
          --runtime=300 \
          --direct=0 \
          --iodepth=$iosize \
          --scramble_buffers=1 \
          --offset=0 \
          --size=100g
    done
  done
```

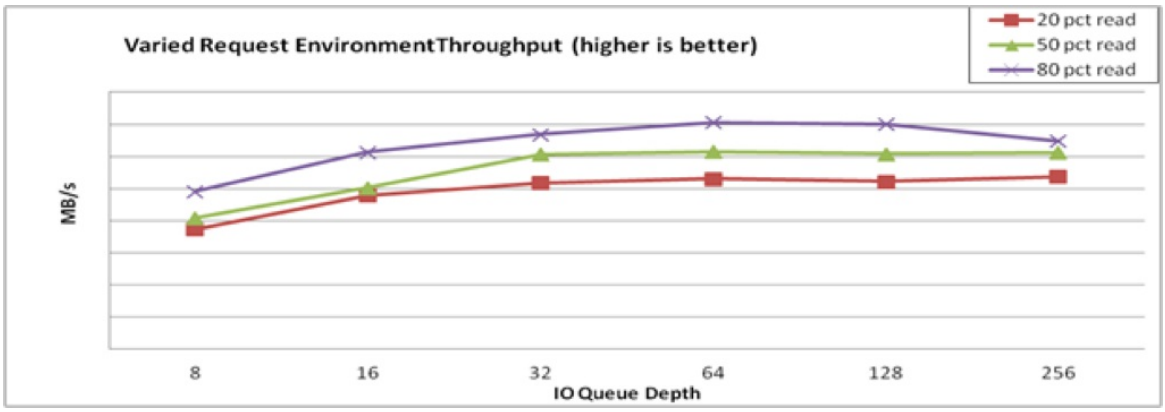
4. VDO テストボリュームを削除します。
   詳細は、「VDO パフォーマンステストボリュームのクリーンアップ」 を参照してください。

5. テスト結果をグラフで表示します。

例4.4 アプリケーション環境分析

以下の図は、VDO が混合 I/O 負荷にどのように応答するかの例を示しています。

図4.5 混合環境パフォーマンス



4.8. FIO を使用した VDO パフォーマンスのテストに使用するオプション

VDO テストは、**fio** ユーティリティーを使用して、反復可能な特性を持つデータを合成して生成します。テストで実際のワークロードをシミュレートするには、以下の **fio** オプションが必要です。

表4.1 使用されている fio オプション

引数	説明	テストで使用される値
--size	**fio** がジョブごとにターゲットに送信するデータ量。 **--numjobs** オプションも参照してください。	100 GiB
--bs	**fio** によって生成された各読み書き要求のブロックサイズ。 Red Hat は、4 KiB のデフォルトの VDO と一致する 4 KiB のブロックサイズを推奨しています。	4k
--numjobs	**fio** がベンチマーク用に作成するジョブの数。 各ジョブは、**--size** オプションで指定したデータ量を送信します。最初のジョブは、**--offset** オプションで指定したオフセットでデバイスにデータを送信します。その後のジョブは、**--offset_incremen** オプションを指定しない限り、ディスクの同じ領域を上書きします。これは、前のジョブがその値で開始した場所からそれぞれのジョブをオフセットします。 フラッシュディスク (SSD) で最高のパフォーマンスを実現するには、Red Hat では少なくとも 2 つのジョブを推奨します。通常、1 つのジョブで、ローテーションを行うディスク (HDD) のスループットをいっぱいにできます。	HDD の場合は 1、SSD の場合は 2

引数	説明	テストで使用される値
--thread	**fio** ジョブに対して、フォークではなくスレッドで実行するように指示します。これにより、コンテキストスイッチを制限することでパフォーマンスが向上する可能性があります。	none
--ioengine	**fio** がベンチマークに使用する I/O エンジン。 Red Hat のテストは、**libaio** と呼ばれる非同期で非バッファーのエンジンを使用して、複数のプロセスが同時に無作為な要求を行うワークロードをテストします。**libaio** エンジンにより、1 つのスレッドが、データを取得する前に複数のリクエストを非同期に作成できます。これにより、同期エンジンが多くのスレッドからのリクエストを提供した場合に、同期エンジンが必要とするコンテキストスイッチの数が制限されます。	**libaio**
--direct	このオプションは、デバイスに送信された要求がカーネルページキャッシュを回避できるようにします。 **libaio** エンジンを **--direct** オプションと併用する必要があります。そうでない場合は、カーネルはすべての I/O 要求に対して同期 API を使用します。	1 (**libaio**)
--iodepth	任意の時点で実行している I/O バッファーの数。 高い値を設定すると、通常、無作為な読み取りまたは書き込みのパフォーマンスが向上します。高い値を設定すると、コントローラーは常にバッチ処理を要求できます。ただし、設定値が高すぎる (通常は 1K を超える) と、望ましくないレイテンシーが発生する可能性があります。 Red Hat では、128 から 512 までの値を推奨します。最後の値はトレードオフであり、アプリケーションがレイテンシーを許容する方法によって異なります。	最小 128
--iodepth_batch_submit	I/O 深度のバッファープールが空になり始めたときに作成する I/O 要求の数。 このオプションは、I/O 操作からのタスク切り替えを、テスト中のバッファー作成に制限します。	16
--iodepth_batch_complete	バッチ送信前に完了する I/O 操作の数。 このオプションは、I/O 操作からのタスク切り替えを、テスト中のバッファー作成に制限します。	16

引数	説明	テストで使用される値
--gtod_reduce	レイテンシーを計算する時刻の呼び出しを無効にします。 この設定を有効にすると、スループットが低下します。レイテンシー測定が必要でない限り、オプションを有効にします。	1

第5章 未使用ブロックの破棄

破棄操作に対応するブロックデバイスで破棄操作を実行するか、そのスケジュールを設定できます。ブロック破棄操作は、マウントされたファイルシステムでファイルシステムブロックが使用されなくなった基礎となるストレージと通信します。ブロック破棄操作により、SSD はガベージコレクションルーチンを最適化でき、シンプロビジョニングされたストレージに未使用の物理ブロックを再利用するように通知できます。

要件

- ファイルシステムの基礎となるブロックデバイスは、物理的な破棄操作に対応している必要があります。
  /sys/block/<device>/queue/discard_max_bytes ファイルの値がゼロではない場合は、物理的な破棄操作はサポートされます。

5.1. ブロック破棄操作のタイプ

以下のような、さまざまな方法で破棄操作を実行できます。

バッチ破棄

これは、ユーザーによって明示的にトリガーされ、選択したファイルシステム内の未使用のブロックをすべて破棄します。

オンライン破棄

これは、マウント時に指定され、ユーザーの介入なしにリアルタイムでトリガーされます。オンライン破棄操作は、**used** から **free** 状態に移行中のブロックのみを破棄します。

定期的な破棄

systemd サービスが定期的に実行するバッチ操作です。

すべてのタイプは、XFS ファイルシステムおよび ext4 ファイルシステムでサポートされます。

推奨事項
Red Hat は、バッチ破棄または周期破棄を使用することを推奨します。

以下の場合にのみ、オンライン破棄を使用してください。

- システムのワークロードでバッチ破棄が実行できない場合
- パフォーマンス維持にオンライン破棄操作が必要な場合

5.2. バッチブロック破棄の実行

バッチブロック破棄操作を実行して、マウントされたファイルシステムの未使用ブロックを破棄することができます。

前提条件

- ファイルシステムがマウントされている。
- ファイルシステムの基礎となるブロックデバイスが物理的な破棄操作に対応している。

手順

- **fstrim** ユーティリティーを使用します。

- 選択したファイルシステムでのみ破棄を実行するには、次のコマンドを使用します。

```
# fstrim mount-point
```

- マウントされているすべてのファイルシステムで破棄を実行するには、次のコマンドを使用します。

```
# fstrim --all
```

fstrim コマンドを以下のいずれかで実行している場合は、

- 破棄操作に対応していないデバイス
- 複数のデバイスから設定され、そのデバイスの 1 つが破棄操作に対応していない論理デバイス (LVM または MD)

次のメッセージが表示されます。

```
# fstrim /mnt/non_discard

fstrim: /mnt/non_discard: the discard operation is not supported
```

関連情報

- **fstrim(8)** man ページ。

5.3. オンラインブロック破棄の有効化

オンラインブロック破棄操作を実行して、サポートしているすべてのファイルシステムで未使用のブロックを自動的に破棄できます。

手順

- マウント時のオンライン破棄を有効にします。
  - ファイルシステムを手動でマウントするには、**-o discard** マウントオプションを追加します。

```
# mount -o discard device mount-point
```

  - ファイルシステムを永続的にマウントするには、**/etc/fstab** ファイルのマウントエントリーに **discard** オプションを追加します。

関連情報

- **mount(8)** man ページ。
- **fstab(5)** man ページ

5.4. 定期的なブロック破棄の有効化

systemd タイマーを有効にして、サポートしているすべてのファイルシステムで未使用ブロックを定期的に破棄できます。

手順

- **systemd** タイマーを有効にして起動します。

```
# systemctl enable --now fstrim.timer
Created symlink /etc/systemd/system/timers.target.wants/fstrim.timer →
/usr/lib/systemd/system/fstrim.timer.
```

検証

- タイマーのステータスを確認します。

```
# systemctl status fstrim.timer
fstrim.timer - Discard unused blocks once a week
   Loaded: loaded (/usr/lib/systemd/system/fstrim.timer; enabled; vendor preset: disabled)
   Active: active (waiting) since Wed 2023-05-17 13:24:41 CEST; 3min 15s ago
  Trigger: Mon 2023-05-22 01:20:46 CEST; 4 days left
     Docs: man:fstrim

May 17 13:24:41 localhost.localdomain systemd[1]: Started Discard unused blocks once a
week.
```

第6章 永続的な命名属性の概要

システム管理者は、永続的な命名属性を使用してストレージボリュームを参照し、再起動を何度も行っても信頼できるストレージ設定を構築する必要があります。

6.1. 非永続的な命名属性のデメリット

Red Hat Enterprise Linux では、ストレージデバイスを識別する方法が複数あります。特にドライブへのインストール時やドライブの再フォーマット時に誤ったデバイスにアクセスしないようにするため、適切なオプションを使用して各デバイスを識別することが重要になります。

従来、**/dev/sd(メジャー番号)(マイナー番号)** の形式の非永続的な名前は、ストレージデバイスを参照するために Linux 上で使用されます。メジャー番号とマイナー番号の範囲、および関連する **sd** 名は、検出されると各デバイスに割り当てられます。つまり、デバイスの検出順序が変わると、メジャー番号とマイナー番号の範囲、および関連する **sd** 名の関連付けが変わる可能性があります。

このような順序の変更は、以下の状況で発生する可能性があります。

- システム起動プロセスの並列化により、システム起動ごとに異なる順序でストレージデバイスが検出された場合。
- ディスクが起動しなかったり、SCSI コントローラーに応答しなかった場合。この場合は、通常のデバイスプローブにより検出されません。ディスクはシステムにアクセスできなくなり、後続のデバイスは関連する次の **sd** 名が含まれる、メジャー番号およびマイナー番号の範囲があります。たとえば、通常 **sdb** と呼ばれるディスクが検出されないと、**sdc** と呼ばれるディスクが **sdb** として代わりに表示されます。
- SCSI コントローラー (ホストバスアダプターまたは HBA) が初期化に失敗し、その HBA に接続されているすべてのディスクが検出されなかった場合。後続のプローブされた HBA に接続しているディスクは、別のメジャー番号およびマイナー番号の範囲、および関連する別の **sd** 名が割り当てられます。
- システムに異なるタイプの HBA が存在する場合は、ドライバー初期化の順序が変更する可能性があります。これにより、HBA に接続されているディスクが異なる順序で検出される可能性があります。また、HBA がシステムの他の PCI スロットに移動した場合でも発生する可能性があります。
- ストレージアレイや干渉するスイッチの電源が切れた場合など、ストレージデバイスがプローブされたときに、ファイバーチャネル、iSCSI、または FCoE アダプターを持つシステムに接続されたディスクがアクセスできなくなる可能性があります。システムが起動するまでの時間よりもストレージアレイがオンラインになるまでの時間の方が長い場合に、電源の障害後にシステムが再起動すると、この問題が発生する可能性があります。一部のファイバーチャネルドライバーは WWPN マッピングへの永続 SCSI ターゲット ID を指定するメカニズムをサポートしますが、メジャー番号およびマイナー番号の範囲や関連する **sd** 名は予約されず、一貫性のある SCSI ターゲット ID 番号のみが提供されます。

そのため、**/etc/fstab** ファイルなどにあるデバイスを参照するときにメジャー番号およびマイナー番号の範囲や関連する **sd** 名を使用することは望ましくありません。誤ったデバイスがマウントされ、データが破損する可能性があります。

しかし、場合によっては他のメカニズムが使用される場合でも **sd** 名の参照が必要になる場合もあります (デバイスによりエラーが報告される場合など)。これは、Linux カーネルはデバイスに関するカーネルメッセージで **sd** 名 (および SCSI ホスト、チャネル、ターゲット、LUN タプル) を使用するためです。

6.2. ファイルシステムおよびデバイスの識別子

このセクションでは、ファイルシステムおよびブロックデバイスを識別する永続的な属性の相違点を説明します。

ファイルシステムの識別子
ファイルシステムの識別子は、ブロックデバイス上に作成された特定のファイルシステムに関連付けられます。識別子はファイルシステムの一部としても格納されます。ファイルシステムを別のデバイスにコピーしても、ファイルシステム識別子は同じです。一方、**mkfs** ユーティリティーでフォーマットするなどしてデバイスを書き換えると、デバイスはその属性を失います。

ファイルシステムの識別子に含まれるものは、次のとおりです。

- 一意の ID (UUID)
- ラベル

デバイスの識別子
デバイス識別子は、ブロックデバイス (ディスクやパーティションなど) に関連付けられます。**mkfs** ユーティリティーでフォーマットするなどしてデバイスを書き換えた場合、デバイスはファイルシステムに格納されていないため、属性を保持します。

デバイスの識別子に含まれるものは、次のとおりです。

- World Wide Identifier (WWID)
- パーティション UUID
- シリアル番号

推奨事項

- 論理ボリュームなどの一部のファイルシステムは、複数のデバイスにまたがっています。Red Hat は、デバイスの識別子ではなくファイルシステムの識別子を使用してこのファイルシステムにアクセスすることを推奨します。

6.3. /DEV/DISK/ にある UDEV メカニズムにより管理されるデバイス名

udev メカニズムは、Linux のすべてのタイプのデバイスに使用され、ストレージデバイスだけに限定されません。**/dev/disk/** ディレクトリーにさまざまな種類の永続的な命名属性を提供します。ストレージデバイスの場合、Red Hat Enterprise Linux には **/dev/disk/** ディレクトリーにシンボリックリンクを作成する **udev** ルールが含まれています。これにより、次の方法でストレージデバイスを参照できます。

- ストレージデバイスのコンテンツ
- 一意の ID
- シリアル番号

udev の命名属性は永続的なものですが、システムを再起動しても自動的には変更されないため、設定可能なものもあります。

6.3.1. ファイルシステムの識別子

/dev/disk/by-uuid/ の UUID 属性

このディレクトリーのエントリーは、デバイスに格納されているコンテンツ (つまりデータ) 内の **一意の ID** (UUID) によりストレージデバイスを参照するシンボリック名を提供します。以下に例を示します。

```
/dev/disk/by-uuid/3e6be9de-8139-11d1-9106-a43f08d823a6
```

次の構文を使用することで、UUID を使用して **/etc/fstab** ファイルのデバイスを参照できます。

```
UUID=3e6be9de-8139-11d1-9106-a43f08d823a6
```

ファイルシステムを作成する際に UUID 属性を設定できます。後で変更することもできます。

/dev/disk/by-label/ のラベル属性

このディレクトリーのエントリーは、デバイスに格納されているコンテンツ (つまりデータ) 内の **ラベル** により、ストレージデバイスを参照するシンボリック名を提供します。

以下に例を示します。

```
/dev/disk/by-label/Boot
```

次の構文を使用することで、ラベルを使用して **/etc/fstab** ファイルのデバイスを参照できます。

```
LABEL=Boot
```

ファイルシステムを作成するときにラベル属性を設定できます。また、後で変更することもできます。

6.3.2. デバイスの識別子

/dev/disk/by-id/ の WWID 属性

World Wide Identifier (WWID) は永続的で、SCSI 規格によりすべての SCSI デバイスが必要とする **システムに依存しない識別子** です。各ストレージデバイスの WWID 識別子は一意となることが保証され、デバイスのアクセスに使用されるパスに依存しません。この識別子はデバイスのプロパティーですが、デバイスのコンテンツ (つまりデータ) には格納されません。

この識別子は、SCSI Inquiry を発行して Device Identification Vital Product Data (**0x83** ページ) または Unit Serial Number (**0x80** ページ) を取得することにより獲得できます。

Red Hat Enterprise Linux では、WWID ベースのデバイス名から、そのシステムの現在の **/dev/sd** 名への正しいマッピングを自動的に維持します。デバイスへのパスが変更したり、別のシステムからそのデバイスへのアクセスがあった場合にも、アプリケーションはディスク上のデータ参照に **/dev/disk/by-id/** を使用できます。

例6.1 WWID マッピング

WWID シンボリックリンク	非永続的なデバイス	備考
/dev/disk/by-id/scsi-3600508b400105e210000900000490000	**/dev/sda**	ページ **0x83** の識別子を持つデバイス
/dev/disk/by-id/scsi-SSEAGATE_ST373453LW_3HW1RHM6	**/dev/sdb**	ページ **0x80** の識別子を持つデバイス

WWID シンボリックリンク	非永続的なデバイス	備考
/dev/disk/by-id/ata-SAMSUNG_MZNLN256HMHQ-000L7_S2WDNX0J336519-part3	**/dev/sdc3**	ディスクパーティション

システムにより提供される永続的な名前のほかに、**udev** ルールを使用して独自の永続的な名前を実装し、ストレージの WWID にマップすることもできます。

/dev/disk/by-partuuid のパーティション UUID 属性

パーティション UUID (PARTUUID) 属性は、GPT パーティションテーブルにより定義されているパーティションを識別します。

例6.2 パーティション UUID のマッピング

PARTUUID シンボリックリンク	非永続的なデバイス
/dev/disk/by-partuuid/4cd1448a-01	**/dev/sda1**
/dev/disk/by-partuuid/4cd1448a-02	**/dev/sda2**
/dev/disk/by-partuuid/4cd1448a-03	**/dev/sda3**

/dev/disk/by-path/ のパス属性

この属性は、デバイスへのアクセスに使用される **ハードウェアパス** がストレージデバイスを参照するシンボル名を提供します。

ハードウェアパス (PCI ID、ターゲットポート、LUN 番号など) の一部が変更されると、パス属性に失敗します。このため、パス属性は信頼性に欠けます。ただし、パス属性は以下のいずれかのシナリオで役に立ちます。

- 後で置き換える予定のディスクを特定する必要があります。
- 特定の場所にあるディスクにストレージサービスをインストールする予定です。

6.4. DM MULTIPATH を使用した WORLD WIDE IDENTIFIER

Device Mapper (DM) Multipath を設定して、World Wide Identifier (WWID) と非永続的なデバイス名をマッピングできます。

システムからデバイスへのパスが複数ある場合、DM Multipath はこれを検出するために WWID を使用します。その後、DM Multipath は **/dev/mapper/wwid** ディレクトリー (例: **/dev/mapper/3600508b400105df70000e00000ac0000**) に単一の "疑似デバイス" を表示します。

コマンド **multipath -l** は、非永続的な識別子へのマッピングを示します。

- **Host:Channel:Target:LUN**

- **/dev/sd** 名
- **major:minor** 数値

例6.3 マルチパス設定での WWID マッピング

multipath -l コマンドの出力例:

```
3600508b400105df70000e00000ac0000 dm-2 vendor,product
[size=20G][features=1 queue_if_no_path][hwhandler=0][rw]
\_ round-robin 0 [prio=0][active]
 \_ 5:0:1:1 sdc 8:32  [active][undef]
 \_ 6:0:1:1 sdg 8:96  [active][undef]
\_ round-robin 0 [prio=0][enabled]
 \_ 5:0:0:1 sdb 8:16  [active][undef]
 \_ 6:0:0:1 sdf 8:80  [active][undef]
```

DM Multipath は、各 WWID ベースのデバイス名から、システムで対応する **/dev/sd** 名への適切なマッピングを自動的に維持します。これらの名前は、パスが変更しても持続し、他のシステムからデバイスにアクセスする際に一貫性を保持します。

DM Multipath の **user_friendly_names** 機能を使用すると、WWID は **/dev/mapper/mpathN** 形式の名前にマップされます。デフォルトでは、このマッピングは **/etc/multipath/bindings** ファイルに保持されています。これらの **mpathN** 名は、そのファイルが維持されている限り永続的です。

重要

user_friendly_names を使用する場合は、クラスター内で一貫した名前を取得するために追加の手順が必要です。

6.5. UDEV デバイス命名規則の制約

udev 命名規則の制約の一部は次のとおりです。

- **udev** イベントに対して **udev** ルールが処理されるときに、**udev** メカニズムはストレージデバイスをクエリーする機能に依存する可能性があるため、クエリーの実行時にデバイスにアクセスできない可能性があります。これは、ファイバーチャネル、iSCSI、または FCoE ストレージデバイスといった、デバイスがサーバーシャーシにない場合に発生する可能性が高くなります。
- カーネルは **udev** イベントをいつでも送信する可能性があるため、デバイスにアクセスできない場合に **/dev/disk/by-*/** リンクが削除される可能性があります。
- **udev** イベントが生成されそのイベントが処理されるまでに遅延が生じる場合があります (大量のデバイスが検出され、ユーザー空間の **udev** サービスによる各デバイスのルールを処理するのにある程度の時間がかかる場合など)。これにより、カーネルがデバイスを検出してから、**/dev/disk/by-*/** の名前が利用できるようになるまでに遅延が生じる可能性があります。
- ルールに呼び出される **blkid** などの外部プログラムによってデバイスが短期間開き、他の目的でデバイスにアクセスできなくなる可能性があります。
- /dev/disk/ の **udev** メカニズムで管理されるデバイス名は、メジャーリリース間で変更される可能性があるため、リンクの更新が必要になる場合があります。

6.6. 永続的な命名属性のリスト表示

この手順では、非永続的なストレージデバイスの永続命名属性を確認する方法を説明します。

手順

- UUID 属性とラベル属性をリスト表示するには、**lsblk** ユーティリティーを使用します。

```
$ lsblk --fs storage-device
```

以下に例を示します。

例6.4 ファイルシステムの UUID とラベルの表示

```
$ lsblk --fs /dev/sda1

NAME FSTYPE LABEL UUID                                 MOUNTPOINT
sda1 xfs    Boot  afa5d5e3-9050-48c3-acc1-bb30095f3dc4 /boot
```

- PARTUUID 属性をリスト表示するには、**--output +PARTUUID** オプションを指定して **lsblk** ユーティリティーを使用します。

```
$ lsblk --output +PARTUUID
```

以下に例を示します。

例6.5 パーティションの PARTUUID 属性の表示

```
$ lsblk --output +PARTUUID /dev/sda1

NAME MAJ:MIN RM  SIZE RO TYPE MOUNTPOINT PARTUUID
sda1   8:1    0  512M  0 part /boot      4cd1448a-01
```

- WWID 属性をリスト表示するには、**/dev/disk/by-id/** ディレクトリーのシンボリックリンクのターゲットを調べます。以下に例を示します。

例6.6 システムにある全ストレージデバイスの WWID の表示

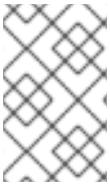
```
$ file /dev/disk/by-id/*

/dev/disk/by-id/ata-QEMU_HARDDISK_QM00001
symbolic link to ../../sda
/dev/disk/by-id/ata-QEMU_HARDDISK_QM00001-part1
symbolic link to ../../sda1
/dev/disk/by-id/ata-QEMU_HARDDISK_QM00001-part2
symbolic link to ../../sda2
/dev/disk/by-id/dm-name-rhel_rhel8-root
symbolic link to ../../dm-0
/dev/disk/by-id/dm-name-rhel_rhel8-swap
symbolic link to ../../dm-1
/dev/disk/by-id/dm-uuid-LVM-
QIWtEHtXGobe5bewlIUDivKOz5ofkgFhP0RMFsNyySVihqEl2cWWbR7MjXJolD6g
```

```
symbolic link to ../../dm-1
/dev/disk/by-id/dm-uuid-LVM-
QIWtEHtXGobe5bewlIUDivKOz5ofkgFhXqH2M45hD2H9nAf2qfWSrlRLhzfMyOKd
symbolic link to ../../dm-0
/dev/disk/by-id/lvm-pv-uuid-atlr2Y-vuMo-ueoH-CpMG-4JuH-AhEF-wu4QQm
symbolic link to ../../sda2
```

6.7. 永続的な命名属性の変更

この手順では、ファイルシステムの UUID またはラベルの永続的な命名属性を変更する方法を説明します。



注記

udev 属性の変更はバックグラウンドで行われ、時間がかかる場合があります。**udevadm settle** コマンドは変更が完全に登録されるまで待機します。これにより、次のコマンドが新しい属性を正しく利用できるようになります。

以下のコマンドでは、次を行います。

- new-uuid を、設定する UUID (例: **1cdfbc07-1c90-4984-b5ec-f61943f5ea50**) に置き換えます。**uuidgen** コマンドを使用して UUID を生成できます。
- new-label を、ラベル (例: **backup_data**) に置き換えます。

前提条件

- XFS ファイルシステムをアンマウントしている (XFS ファイルシステムの属性を変更する場合)。

手順

- **XFS** ファイルシステムの UUID またはラベル属性を変更するには、**xfs_admin** ユーティリティーを使用します。

```
# xfs_admin -U new-uuid -L new-label storage-device
# udevadm settle
```

- **ext4** ファイルシステム、**ext3** ファイルシステム、**ext2** ファイルシステムの UUID またはラベル属性を変更するには、**tune2fs** ユーティリティーを使用します。

```
# tune2fs -U new-uuid -L new-label storage-device
# udevadm settle
```

- スワップボリュームの UUID またはラベル属性を変更するには、**swaplabel** ユーティリティーを使用します。

```
# swaplabel --uuid new-uuid --label new-label swap-device
# udevadm settle
```

第7章 WEB コンソールを使用した VIRTUAL DATA OPTIMIZER ボリュームの管理

RHEL 8 Web コンソールを使用して、VDO (Virtual Data Optimizer) を設定します。

以下の方法について説明します。

- VDO ボリュームの作成
- VDO ボリュームのフォーマット
- VDO ボリュームの拡張

前提条件

- RHEL 8 Web コンソールをインストールし、アクセスできる。詳細は、Web コンソールのインストールおよび有効化 を参照してください。
- **cockpit-storaged** パッケージがシステムにインストールされている。

7.1. WEB コンソールでの VDO ボリューム

Red Hat Enterprise Linux 8 では、Virtual Data Optimizer (VDO) がサポートされます。

VDO は、以下を組み合わせたブロック仮想化テクノロジーです。

圧縮
詳細は、Enabling or disabling compression in VDO を参照してください。

重複排除
詳細は、Enabling or disabling compression in VDO を参照してください。

シンプロビジョニング
詳細は、シンプロビジョニングボリューム (シンボリューム) の作成と管理 を参照してください。

このような技術を使用して、VDO は、以下を行います。

- ストレージ領域をインラインに保存します。
- ファイルを圧縮します。
- 重複を排除します。
- 物理ストレージまたは論理ストレージが提供するサイズよりも多くの仮想領域を割り当てることができます。
- 拡大して仮想ストレージを拡張できます。

VDO は、さまざまなタイプのストレージに作成できます。RHEL 8 Web コンソールでは、以下に VDO を設定できます。

- LVM

注記

シンプロビジョニングされたボリュームに VDO を設定することはできません。

- 物理ボリューム
- ソフトウェア RAID

ストレージスタックにおける VDO の配置の詳細は、System Requirements を参照してください。

関連情報

- VDO の詳細は、Deduplicating and compressing storage を参照してください。

7.2. WEB コンソールで VDO ボリュームの作成

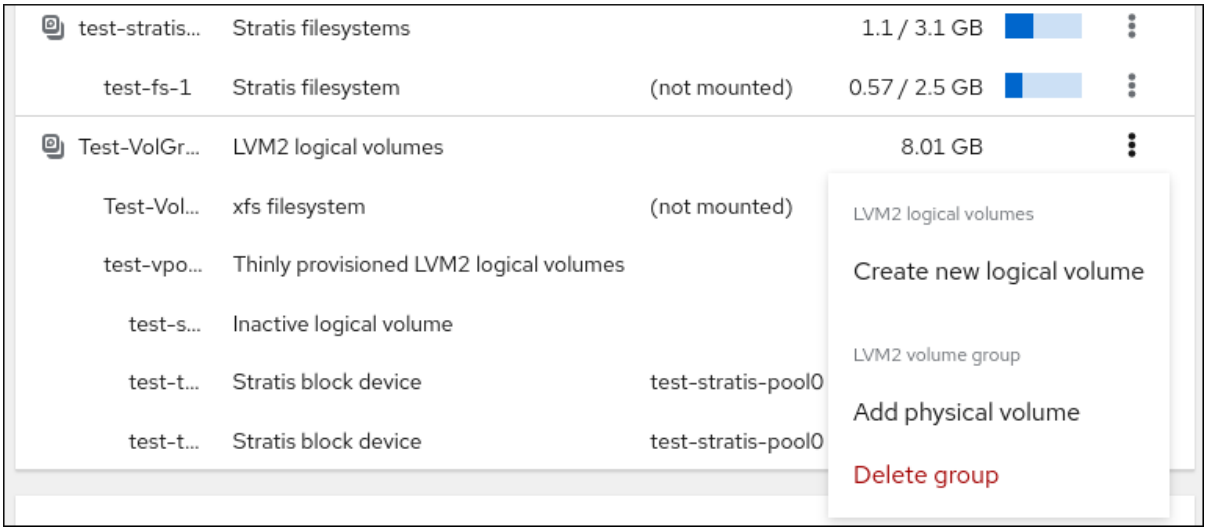
RHEL Web コンソールで VDO ボリュームを作成します。

前提条件

- VDO を作成する LVM2 グループ。

手順

1. RHEL 8 Web コンソールにログインします。
   詳細は、Web コンソールへのログイン を参照してください。
2. **Storage** をクリックします。
3. VDO ボリュームを作成する LVM2 グループの横にあるメニューボタン ⋮ をクリックします。



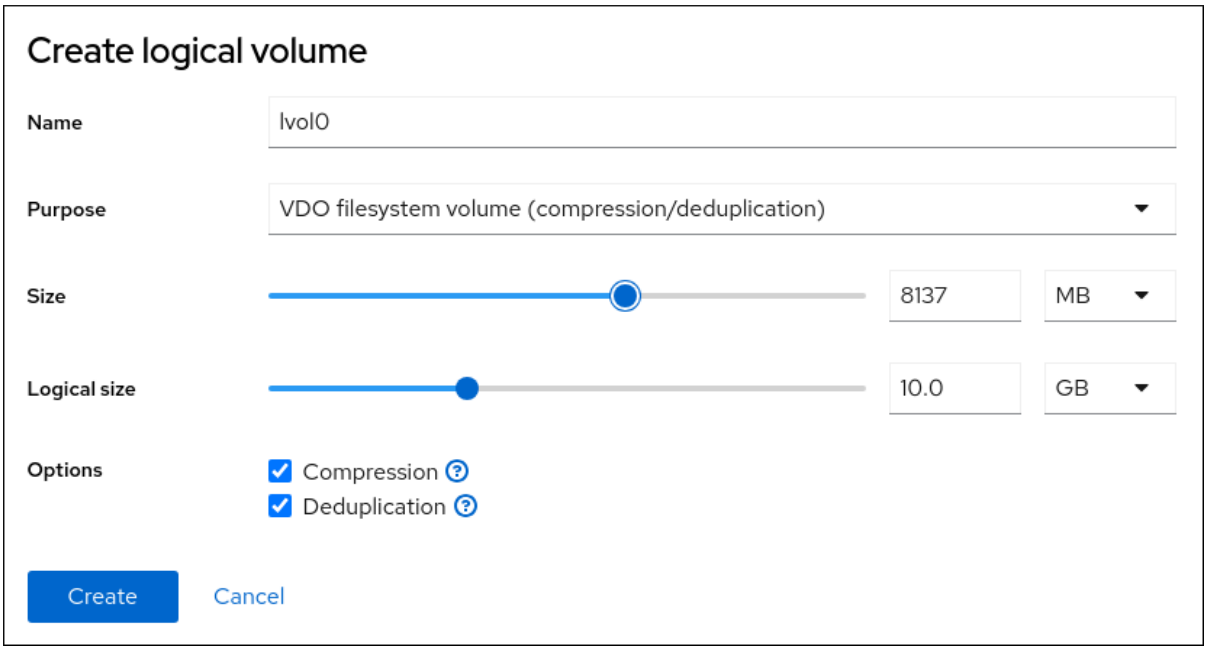
4. **Purpose** フィールドの横にあるドロップダウンメニューで **VDO filesystem volume** を選択します。
5. **Name** フィールドに、VDO ボリュームの名前 (スペースなし) を入力します。
6. **論理サイズ** バーに、VDO ボリュームのサイズを設定します。10 回以上拡張できますが、VDO ボリュームを作成する目的を検討してください。

- アクティブな仮想マシンまたはコンテナーストレージの場合は、使用する論理のサイズを、ボリュームの物理サイズの 10 倍になるようにします。
- オブジェクトストレージの場合は、使用する論理のサイズを、ボリュームの物理サイズの 3 倍になるようにします。

詳細は、Deploying VDO を参照してください。

7. **圧縮** オプションを選択します。このオプションを使用すると、さまざまなファイル形式を効率的に減らすことができます。
詳細は、Enabling or disabling compression in VDO を参照してください。

8. **重複排除** オプションを選択します。
このオプションは、重複ブロックのコピーを削除して、ストレージリソースが使用されなくなるようにします。詳細は、Enabling or disabling compression in VDO を参照してください。



検証手順

- **ストレージ** セクションに新しい VDO ボリュームが表示されることを確認します。そして、ファイルシステムでフォーマットすることができます。

7.3. WEB コンソールで VDO ボリュームのフォーマット

VDO ボリュームは物理ドライブとして動作します。これらを使用するには、ファイルシステムでフォーマットする必要があります。

警告

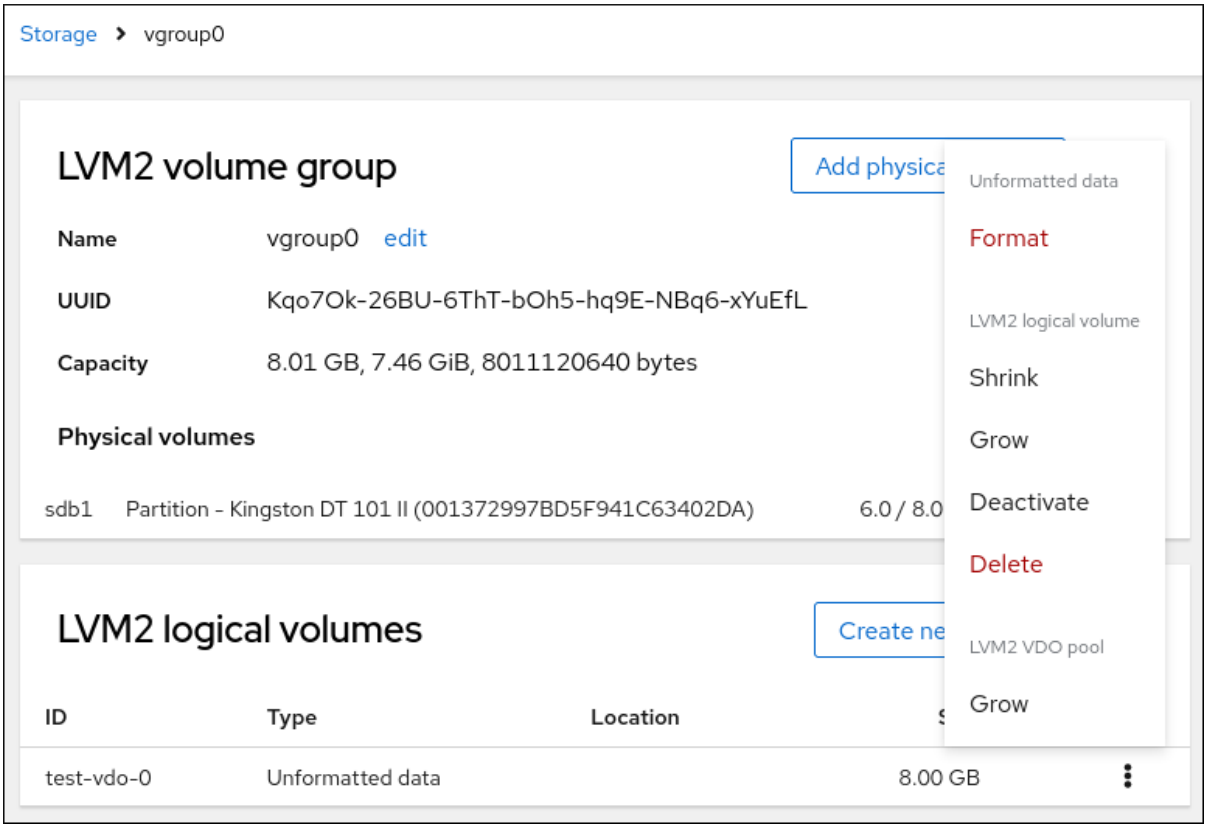
フォーマットするとボリューム上のすべてのデータが消去されます。

前提条件

- VDO ボリュームが作成されている。詳細については、Web コンソールでの VDO ボリュームの作成 を参照してください。

手順

1. RHEL 8 Web コンソールにログインします。詳細は、Web コンソールへのログイン を参照してください。
2. **Storage** をクリックします。
3. フォーマットする VDO ボリュームを含む LVM2 ボリュームグループをクリックします。
4. フォーマットする VDO ボリュームの行末にあるメニューボタン ⋮ をクリックします。
5. **Format** をクリックします。



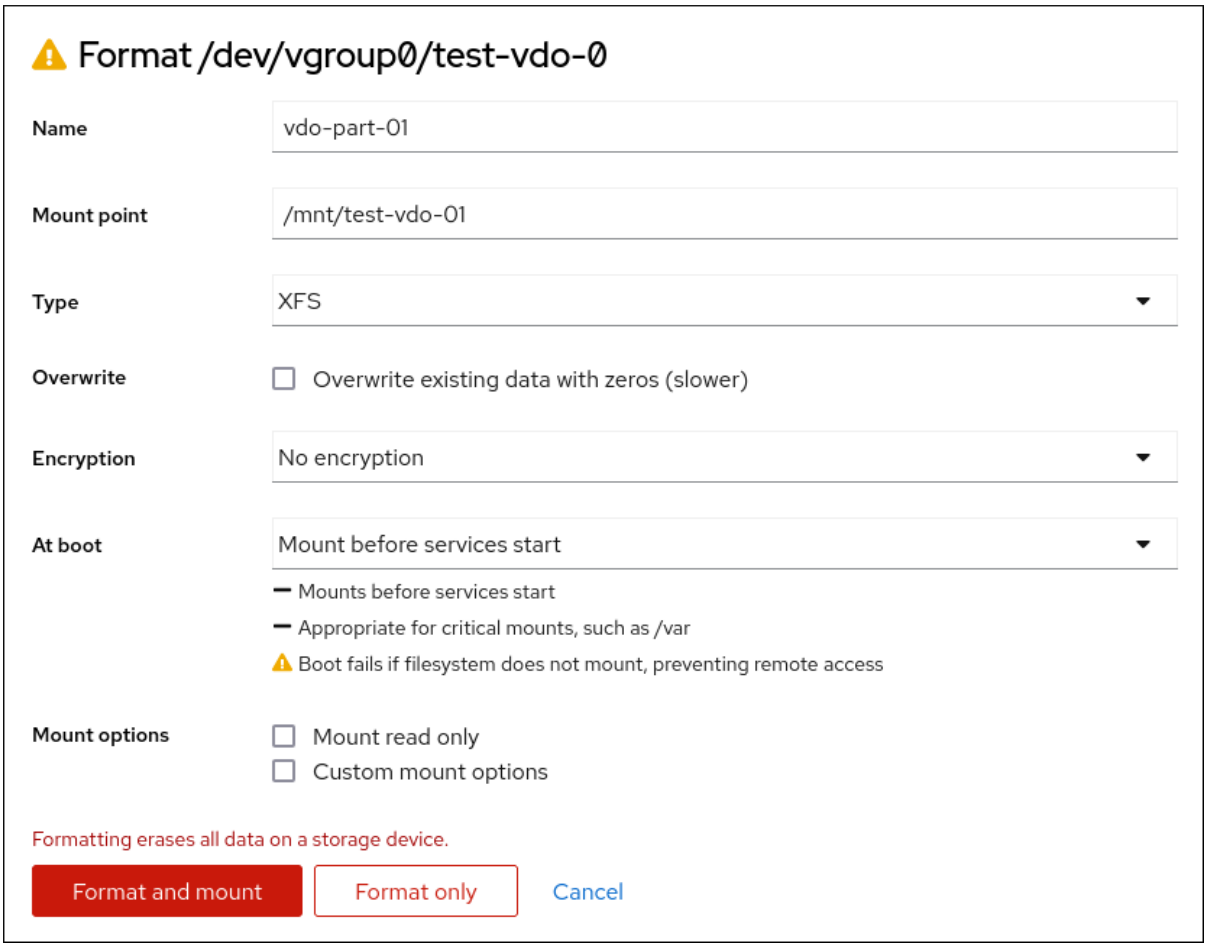
6. **名前** フィールドに、論理ボリューム名を入力します。
7. **マウントポイント** フィールドに、マウントパスを追加します。
8. デフォルトでは、このダイアログを完了すると、Web コンソールはディスクヘッダーのみを書き換えます。このオプションの利点は、フォーマットの速度です。**Overwrite existing data with zeros** オプションをオンにすると、Web コンソールはディスク全体をゼロで書き換えます。このプログラムはディスク全体を調べるため、このオプションを使用すると遅くなります。ディスクに機密データが含まれており、それを書き換えたい場合は、このオプションを使用します。
9. **Type** ドロップダウンメニューで、ファイルシステムを選択します。
   - デフォルトオプションの **XFS** ファイルシステムは、大規模な論理ボリュームをサポートし、物理ドライブを停止せずにオンラインで切り替え、拡張することができます。
     XFS は、ボリュームの縮小に対応していません。したがって、XFS でフォーマットされたボリュームのサイズを縮小することはできません。

- **ext4** ファイルシステムは論理ボリュームをサポートし、オンラインの物理ドライブを停止せずに、既存のファイルシステムの拡大および縮小を行うことができます。

LUKS (Linux Unified Key Setup) 暗号を使用したバージョンも選択できます。パスフレーズを使用してボリュームの暗号化を行えます。

10. **At boot** ドロップダウンメニューで、ボリュームをマウントするタイミングを選択します。

11. **Format and mount** または **Format only** をクリックします。
フォーマットに使用されるオプションや、ボリュームのサイズによって、フォーマットに数分かかることがあります。



検証

- 正常に完了すると、フォーマットされた VDO ボリュームの詳細が **Storage** タブと LVM2 ボリュームグループタブに表示されます。

7.4. WEB コンソールで VDO ボリュームの拡張

RHEL 8 Web コンソールで VDO ボリュームを拡張します。

前提条件

- **cockpit-storaged** パッケージがシステムにインストールされている。
- VDO ボリュームが作成されている。

手順

1. RHEL 8 Web コンソールにログインします。
   詳細は、Web コンソールへのログイン を参照してください。
2. **Storage** をクリックします。
3. **VDO デバイス** で、VDO ボリュームをクリックします。
4. VDO ボリュームの詳細で、**Grow** ボタンをクリックします。
5. **VDO の論理サイズを増加** ダイアログボックスで、VDO ボリュームの論理サイズを増やします。

1. **Grow** をクリックします。

検証手順

- 新しいサイズの VDO ボリュームの詳細を確認し、変更が正常に行われたことを確認します。