Red Hat Enterprise Linux 9

RHEL での論理ボリュームの重複排除および圧縮

LVM に VDO をデプロイしてストレージ容量を増やす

Last Updated: 2024-05-02

Red Hat Enterprise Linux 9 RHEL での論理ボリュームの重複排除および圧縮

LVM に VDO をデプロイしてストレージ容量を増やす

法律上の通知

概要

Logical Volume Manager (LVM) の Virtual Data Optimizer (VDO) 機能を使用して、重複排除および圧縮された論理ボリュームを管理します。VDO は、LVM のシンプロビジョニングボリュームと同様に、LVM の論理ボリューム (LV) の一種として管理できます。 VDO on LVM (LVM-VDO) をデプロイして、ブロックアクセス、ファイルアクセス、ローカルストレージ、およびリモートストレージ用の重複排除ストレージを提供できます。VDO ボリュームの物理スペースが 100% 使用されることを避けるために、シンプロビジョニングされた VDO ボリュームを設定することもできます。

目次

多様性を受け入れるオープンソースの強化

Red Hat では、コード、ドキュメント、Web プロパティーにおける配慮に欠ける用語の置き換えに取り組んでいます。まずは、マスター (master)、スレーブ (slave)、ブラックリスト (blacklist)、ホワイトリスト (whitelist) の 4 つの用語の置き換えから始めます。この取り組みは膨大な作業を要するため、今後の複数のリリースで段階的に用語の置き換えを実施して参ります。詳細は、Red Hat CTO である Chris Wright のメッセージ をご覧ください。

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 LVM 上の VDO の概要

Virtual Data Optimizer (VDO) 機能は、ストレージ用にインラインのブロックレベルの重複排除、圧縮、およびシンプロビジョニングを提供します。VDO は、LVM シンプロビジョニングボリュームと同様に、論理ボリュームマネージャー (LVM) 論理ボリューム (LV) の一種として管理できます。

LVM 上の VDO ボリューム (LVM-VDO) には、次のコンポーネントが含まれます。

VDO プール LV

- これは、VDO LV のデータを保存、重複排除、および圧縮するバッキング物理デバイスです。VDO プール LV は、VDO ボリュームの物理サイズを設定します。これは、VDO がディスクに保存できるデータ量です。
- 現在、各 VDO プール LV は 1 つの VDO LV のみを保持できます。そのため、VDO は VDO LV ごとに重複排除と圧縮を行います。別々の LV に保存されている重複データは、同じ VDO ボリュームのデータ最適化の恩恵を受けません。

VDO LV

- これは、VDO プール LV 上にプロビジョニングされた仮想デバイスです。VDO LV は、VDO ボリュームのプロビジョニングされた論理サイズを設定します。これは、重複排除と圧縮が行われる前にアプリケーションがボリュームに書き込みできるデータ量です。

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、圧縮、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、VDO プール LV が VDO LV を作成するために使用するブロックデバイスを公開します。その後、VDO LV がシステムによって使用されます。
- **kvdo** は、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って返します。
- **kvdo** は、データのブロックを VDO ボリュームに書き込む要求を受信すると、最初にその要求が DISCARD または TRIM 要求であるかどうか、またはデータが均一にゼロであるかどうかを確認します。これらの条件のいずれかが満たされた場合、**kvdo** はブロックマップを更新し、要求を承認します。そうでない場合は、VDO はデータを処理して最適化します。
- kvdo モジュールは、ボリュームの Universal Deduplication Service (UDS) インデックスを内部的に利用し、受信したデータの重複を分析します。UDS は、新しいデータごとに、そのデータが以前に保存されたデータと同一であるかどうかを確認します。インデックスで一致が見つかった場合、ストレージシステムはその一致の正確性を検証し、内部参照を更新して、同じ情報を複数回保存しないようにすることができます。

LVM シンプロビジョニング実装の構造をすでによく理解している方は、表 1.1 を参照してください。VDO のさまざまな要素がシステムにどのように提示されるかを確認できます。

表1.1 LVM 上の VDO と LVM シンプロビジョニングのコンポーネントの比較

	物理デバイス	プロビジョニングされるデバイス
LVM 上の VDO	VDO プール LV	VDO LV
LVM シンプロビジョニング	シンプール	シンボリューム

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のみを認識し、実際に使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、使用可能な物理領域を監視し、使用量がしきい値を超えた場合にアラートを生成します。利用可能な VDO スペースの監視については、VDO の監視 セクションを参照してください。

第2章 LVM-VDO の要件

LVM 上の VDO には、配置とシステムリソースに関する特定の要件があります。

2.1. 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

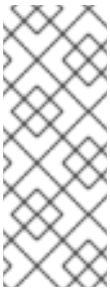
VDO には、固定メモリー 38 MB と変動用に容量を確保する必要があります。

- 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

2.2. 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

2.3. 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2.1 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	RAM の使用量:UDS	RAM の使用量:VDO	ディスク使用量	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

物理サイズ	RAM の使用量:UDS	RAM の使用量:VDO	ディスク使用量	インデックスタイプ
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2.2 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	RAM の使用量:UDS	RAM の使用量:VDO	ディスク使用量	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

2.4. ストレージスタックでの LVM-VDO の配置

VDO 論理ボリュームの下に配置する必要があるストレージ層と、その上に配置する必要があるストレージ層があります。

シックプロビジョニングの層を VDO の上に配置することもできますが、その場合はシックプロビジョニングの保証に頼ることができません。VDO 層はシンプロビジョニングされているため、シンプロビジョニングの影響は、上記のすべての層に及びます。VDO ボリューム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、物理領域が不足する可能性があります。

以下の層は、VDO の配下にある場合にサポートされます。VDO 層の上に配置しないでください。

- DM Multipath
- DM Crypt
- ソフトウェア RAID (LVM または MD RAID)

以下の設定には **対応していません**。

- ループバックデバイスの上に VDO を配置する

- VDO の上に暗号化されたボリュームを配置する
- VDO ボリュームにパーティションを作成する
- VDO ボリュームの上に RAID (LVM RAID、MD RAID、またはその他のタイプ) を配置する
- LVM-VDO に Ceph ストレージをデプロイする

関連情報

- LVM ボリュームのスタッキング (ナレッジベース記事)

第3章 重複排除および圧縮された論理ボリュームの作成

VDO 機能を使用してデータの重複排除と圧縮を行う LVM 論理ボリュームを作成できます。

3.1. LVM-VDO のデプロイメントシナリオ

LVM 上の VDO (LVM-VDO) は、さまざま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。

- ブロックアクセス
- ファイルアクセス
- ローカルストレージ
- リモートストレージ

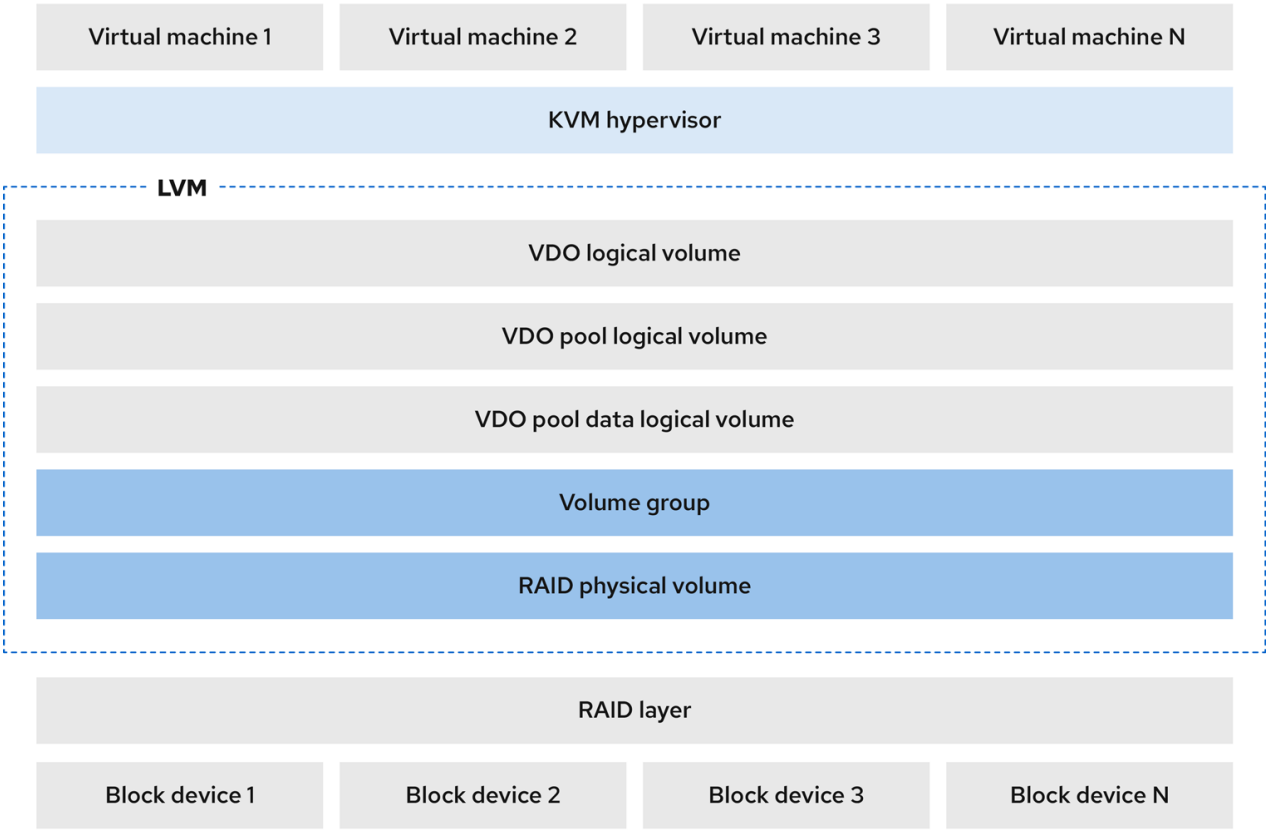
LVM-VDO は、通常の論理ボリューム (LV) として重複排除したストレージを公開するため、そのストレージを標準ファイルシステム、iSCSI および FC のターゲットドライバーとともに、または統合ストレージとして使用できます。

注記

現在、LVM-VDO への Ceph Storage のデプロイはサポートされていません。

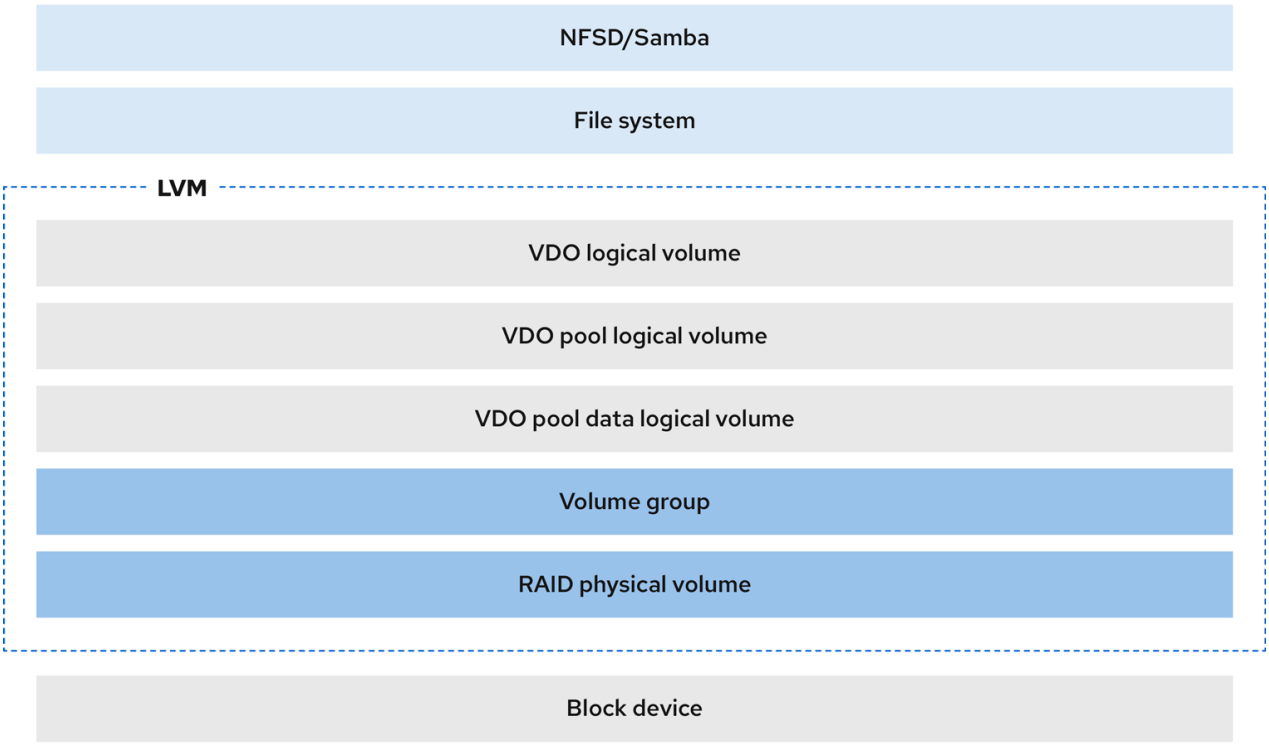
KVM

LVM-VDO は、Direct Attached Storage を使用して設定された KVM サーバーにデプロイできます。



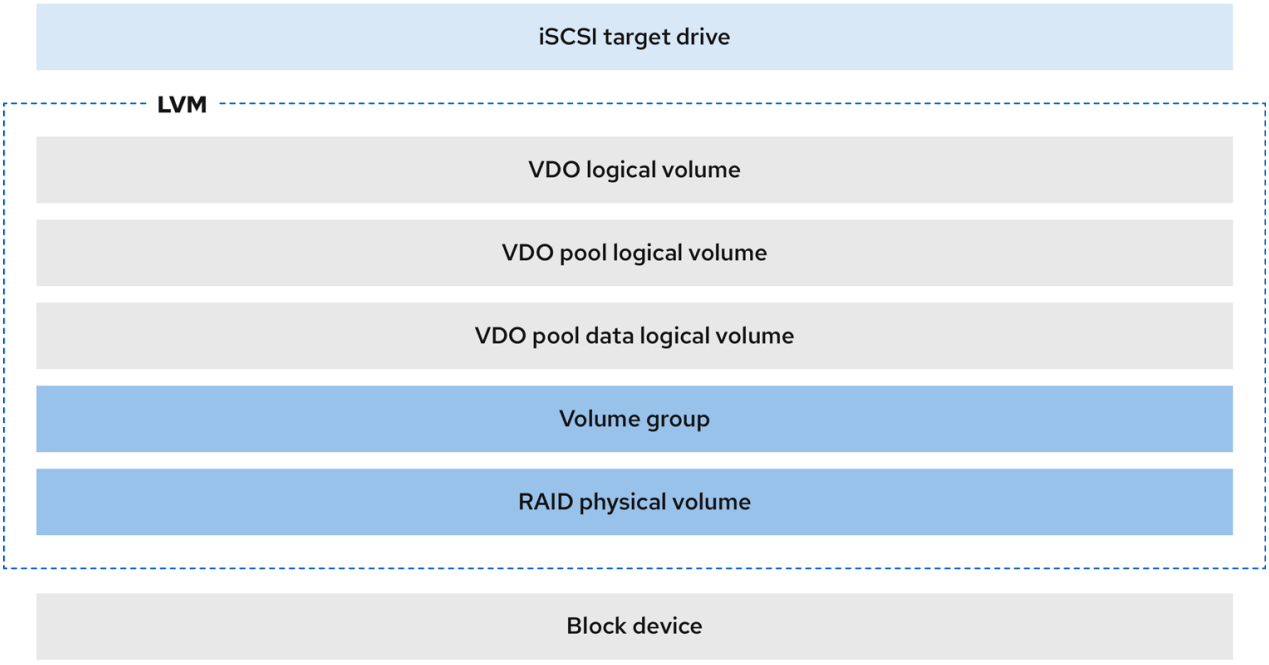
ファイルシステム

VDO LV 上にファイルシステムを作成し、NFS サーバーまたは Samba を使用して NFS または CIFS ユーザーに公開できます。



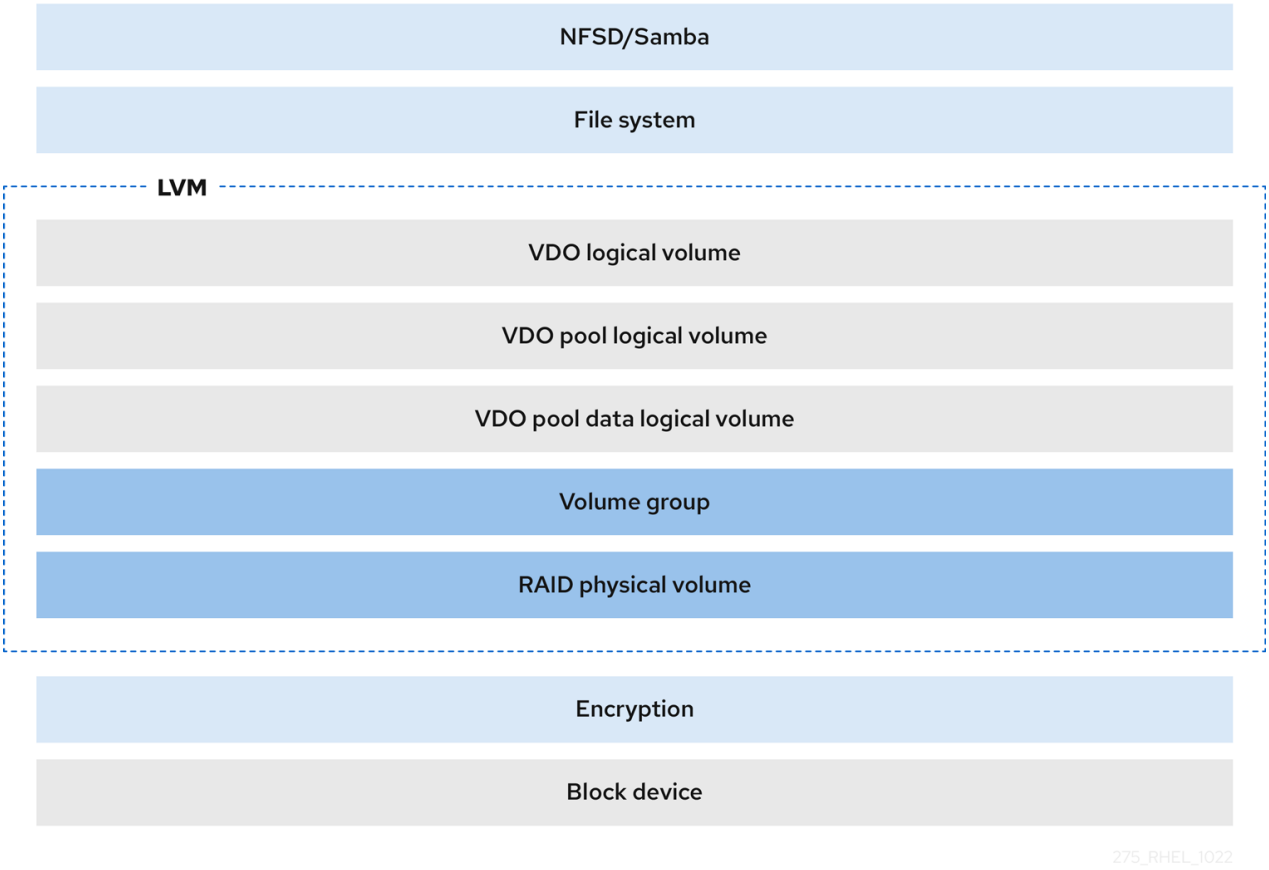
iSCSI ターゲット

VDO LV 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。



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 LV ボリュームを暗号化すると、データのセキュリティーを確保しながら、VDO LV より上のファイルシステムを重複排除できます。



重要

VDO LV の上に暗号化層を適用しても、データの重複排除はほとんど行われません。VDO が重複ブロックを排除する前に、暗号化によって重複ブロックが変更されます。

暗号化層は常に VDO LV の下に配置してください。

3.2. LVM-VDO ボリュームの物理サイズおよび論理サイズ

このセクションでは、VDO が使用できる物理サイズ、利用可能な物理サイズ、論理サイズを説明します。

物理サイズ

これは、VDO プール LV に割り当てられた物理エクステントと同じサイズです。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、物理サイズからメタデータのサイズを引いたものに相当し、スラブサイズの倍数に切り捨てられます。

論理サイズ

これは、VDO LV 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。VDO は現在、絶対最大論理サイズ 4 PB 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。
VDO 論理ボリューム (LV) をセットアップする際には、VDO LV が提示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、論理と物理の割合が 10 対 1 のストレージをプロビジョニングすることを推奨します。つまり、物理ストレージを 1 TB 使用している場合は、論理ストレージを 10 TB にします。

--virtualsize オプションを指定しないと、VDO はボリュームを **1:1** の比率でプロビジョニングします。たとえば、VDO LV を 20 GB の VDO プール LV 上に配置すると、VDO はデフォルトのインデックスサイズが使用されている場合に UDS インデックス用に 2.5 GB を確保します。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、使用可能なストレージは 17.5 GB 以下になります。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。

関連情報

- 物理サイズ別の VDO 要件の例

3.3. VDO のスラブサイズ

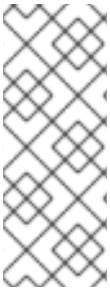
VDO ボリュームの物理ストレージは、複数のスラブに分割されます。各スラブは、物理領域における連続した領域です。特定のボリュームのスラブはすべて同じサイズで、128 MB の 2 のべき乗倍のサイズ (最大 32 GB) になります。

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3.1 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。

3.4. 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。

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

3.5. LVM-VDO ボリュームの作成

この手順では、VDO プール LV に VDO 論理ボリューム (LV) 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。詳細は、VDO のインストール を参照してください。
- 空きストレージ容量を持つ LVM ボリュームグループがシステムに存在する。

手順

1. **vdo1** などの VDO LV の名前を選択します。システムの VDO LV ごとに、異なる名前とデバイスを使用する必要があります。
   以下のすべてのステップで、**vdo-name** をその名前に置き換えてください。
2. VDO LV を作成します。

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

   - **vg-name** を、VDO LV を配置する既存の LVM ボリュームグループの名前に置き換えます。
   - **logical-size** を、VDO LV が提供する論理ストレージのサイズに置き換えます。

- 物理サイズが 16 TiB を超える場合は、以下のオプションを指定して、ボリューム上のスラブサイズを 32 GiB に増やします。

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

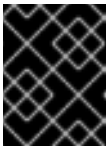
物理サイズが 16 TiB を超える場合にデフォルトのスラブサイズ 2 GiB を使用すると、**lvcreate** コマンドは次のエラーで失敗します。

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

例3.1 コンテナーストレージ用の VDO LV の作成

たとえば、1 TB の VDO プール LV で、コンテナーストレージ用に VDO LV を作成するには、次のコマンドを実行します。

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



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てください。

3. VDO LV 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/vg-name/vdo-name
```

- ext4 ファイルシステムの場合:

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

関連情報

- **lvmvdo(7)** man ページ

3.6. LVM-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、LVM-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

- LVM-VDO ボリュームがシステム上にある。詳細は、LVM-VDO ボリュームの作成 を参照してください。

手順

- LVM-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/vg-name/vdo-name mount-point
```

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

  - ext4 ファイルシステムの場合:

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

  LVM-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。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。

関連情報

- **systemd.mount(5)** man ページ

3.7. LVM-VDO ボリュームでの圧縮および重複排除設定の変更

この手順では、VDO プール論理ボリューム (LV) の圧縮および重複排除を有効または無効にします。

注記

圧縮と重複排除はデフォルトで有効になっています。

前提条件

- LVM-VDO ボリュームがシステム上にある。

手順

1. 論理ボリュームで圧縮および重複排除が有効かどうかを調べるには、次のコマンドを実行します。

```
# lvs -o+vdo_compression,vdo_deduplication
```

2. 稼働中の VDOPoolLV の重複排除インデックスと圧縮のステータスを確認します。

```
# lvs -o+vdo_compression_state,vdo_index_state
```

vdo_index_state は、**error**、**close**、**opening**、**closing**、**online**、および **offline** として表示されます。

3. VDOPoolLV の圧縮を有効または無効にするには、次のコマンドを実行します。

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

4. VDOPoolLV の重複排除を有効または無効にするには、次のコマンドを実行します。

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

関連情報

- **lvmvdo(7)** man ページ

3.8. VIRTUAL DATA OPTIMIZER を使用したシンプロビジョニングの管理

VDO ボリュームの物理領域の使用率が 100% に近づいている場合、その状況に対処するために、シンプロビジョニングされた VDO ボリュームを設定し、今後の物理領域の拡張に備えることができます。たとえば、**lvcreate** 操作で **-l 100%FREE** を 使用する代わりに、'95%FREE' を使用して、必要に応じて後で回復できるように予約スペースを確保します。この手順では、以下の問題を解決する方法を説明します。

- ボリュームの領域が不足している
- ファイルシステムが読み取り専用モードになる
- ボリュームにより ENOSPC が報告される

注記

VDO ボリュームの物理領域の高使用率に対処する最善の方法は、未使用のファイルを削除し、オンライン廃棄または **fstrim** を使用して、その未使用ファイルが使用しているブロックを破棄することです。VDO ボリュームの物理領域は、8192 スラブまでしか拡張できません。これは、デフォルトのスラブサイズが 2 GB の VDO ボリュームの場合は 16 TB、最大スラブサイズが 32 GB の VDO ボリュームの場合は 256 TB に相当します。

以下のすべての手順で、**myvg** と **myvdo** を、それぞれボリュームグループ名および VDO 名に置き換えます。

前提条件

1. VDO ソフトウェアをインストールしている。詳細は、VDO のインストール を参照してください。
2. 空きストレージ容量を持つ LVM ボリュームグループがシステムに存在する。
3. **lvcreate --type vdo --name myvdo myvg -L logical-size-of-pool --virtualsize virtual-size-of-vdo** コマンドを使用した、シンプロビジョニングされた VDO ボリューム。詳細は、LVM-VDO ボリュームの作成 を参照してください。

手順

1. シンプロビジョニングの VDO ボリュームに最適な論理サイズを判断します。

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

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

領域の節約率を計算するには、以下の式を使用します。

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

この例では、次のようになります。

- 約 80 GB のデータセットでは、約 **3.22:1** の領域削減率が得られます。
- データセットのサイズにこの比率を乗算すると、潜在的な論理サイズが得られます。同じ領域節約率でより多くのデータが VDO ボリュームに書き込まれる場合、これは 256 GB になります。
- この数を 200 GB まで下げると、同じ領域節約率の場合、空き物理領域に安全なマージンを確保できる論理サイズになります。

2. VDO ボリュームの空き物理領域を監視します。

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

このコマンドを定期的に実行すると、VDO ボリュームの使用済み物理領域と空き物理領域を監視できます。

3. オプション: 利用可能な **/usr/share/doc/vdo/examples/monitor/monitor_check_vdostats_physicalSpace.pl** スクリプトを使用して、VDO ボリュームの物理領域の使用状況に関する警告を表示します。

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

4. VDO ボリュームを作成すると、**dmeventd** 監視サービスが VDO ボリュームの物理領域の使用状況を監視します。これは、VDO ボリュームの作成時または起動時にデフォルトで有効になります。
VDO ボリュームの監視中に、**journalctl** コマンドを使用して、ログ内の **dmeventd** の出力を表示します。

```
lvm[8331]: Monitoring VDO pool myvg-vpool0-vpool.
...

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

5. 利用可能な物理領域がほとんどない VDO ボリュームを修復します。VDO ボリュームに物理領域を追加できても、ボリュームを拡張する前に領域がいっぱいになった場合は、そのボリュームへの I/O を一時的に停止する必要があるかもしれません。
ボリュームへの I/O を一時的に停止するには、以下の手順を実行します。VDO ボリューム **myvdo** には、**/users/homeDir** パスにマウントされたファイルシステムが含まれています。

   a. ファイルシステムをフリーズします。

```
# xfs_freeze -f /users/homeDir

# vgextend myvg /dev/vdc2

# lvextend -l new_size myvg/vpool0-name

# xfs_freeze -u /users/homeDir
```

b. ファイルシステムをアンマウントします。

```
# umount /users/homeDir

# vgextend myvg /dev/vdc2

# lvextend -l new_size myvg/vpool0-name

# mount -o discard /dev/myvg/myvdo /users/homeDir
```

注記

キャッシュされたデータを含むファイルシステムをアンマウントまたはフリーズすると、キャッシュされたデータの書き込みが発生します。これにより、VDO ボリュームの物理領域がいっぱいになる可能性があります。VDO ボリュームの空き物理領域の監視しきい値を設定する場合は、ファイルシステムのデータキャッシュの最大量を考慮してください。

6. ファイルシステムが使用しなくなったブロックは、**fstrim** ユーティリティーを使用してクリーンアップできます。VDO ボリュームにマウントしたファイルシステムに対して **fstrim** を実行すると、そのボリュームの空き物理領域が増える可能性があります。**fstrim** ユーティリティーは、VDO ボリュームに破棄を送信します。これは、以前使用したブロックへの参照を削除するのに使用されます。これらのブロックのいずれかが単一参照されている場合は、物理領域が使用可能になります。

a. VDO の統計情報を確認して、現在の空き領域の量を確認します。

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

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

b. 未使用のブロックを破棄します。

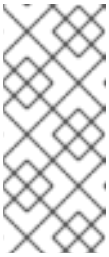
```
# fstrim /users/homeDir
```

c. VDO ボリュームの空き物理領域を表示します。

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

 Device              Size    Used   Available Use%  Space saving%
myvg-vpool0-vpool  100.0G   30.0G  70.0G     30%    43%
```

この例では、ファイルシステムで **fstrim** を実行した後、VDO ボリュームで使用する物理領域 65 G が破棄によって回収されました。



注記

重複排除と圧縮のレベルが低いボリュームを破棄すると、重複排除と圧縮のレベルが高いボリュームを破棄するよりも、物理領域を回収できます。重複排除と圧縮のレベルが高いボリュームでは、単に未使用のブロックを破棄する場合よりも、物理領域を確保するためにより詳細なクリーンアップが必要になる可能性があります。

第4章 LVM-VDO ボリュームの縮小オプション

ファイルシステムは、**discard** オプションを使用してマウントできます。これを使用すると、VDO ボリュームに未使用領域が通知されます。また、**fstrim** アプリケーション (オンデマンドでの破棄) を使用する方法や、**mount -o discard** コマンドを使用して即座に破棄する方法もあります。

fstrim アプリケーションを使用する場合、管理者は追加のプロセスをスケジュールして監視する必要がありますが、**mount -o Discard** コマンドを使用すると、可能な場合はすぐに領域を回復できます。

現在、未使用のブロックを破棄する場合は、**discard** マウントオプションではなく、**fstrim** アプリケーションを使用することが推奨されています。このオプションは、パフォーマンスに非常に大きな影響を及ぼす可能性があるためです。そのため、**nodiscard** がデフォルトになっています。

4.1. VDO での DISCARD マウントオプションの有効化

この手順では、VDO ボリュームで **discard** オプションを有効にします。

前提条件

- LVM-VDO ボリュームがシステム上にある。

手順

- ボリュームで **discard** を有効にします。

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

関連情報

- **xfs(5)**、**mount(8)**、および **lvmvdo(7)** man ページ

4.2. 定期的な TRIM 操作の設定

この手順では、システムで TRIM 操作のスケジューリングを有効にします。

前提条件

- LVM-VDO ボリュームがシステム上にある。

手順

- タイマーを有効にして起動します。

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

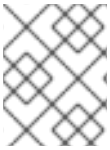
検証

- タイマーが有効化されていることを確認します。

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

例4.1 検証手順での出力例

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



注記

fstrim.timer は、マウントされている全ファイルシステムで実行されるため、VDO ボリュームへの参照は表示されません。

関連情報

- **fstrim(8)** man ページ

第5章 VDO パフォーマンスの最適化

VDO カーネルドライバーは、複数のスレッドを使用してタスクを高速化します。1つのスレッドが I/O リクエストのすべてを実行するのではなく、作業を小さく分割し、異なるスレッドに割り当てます。これらのスレッドは、リクエストを処理する際に相互に通信します。このようにして、1つのスレッドで、頻繁にロックやロック解除を行うことなく共有データを処理できます。

1つのスレッドがタスクを完了した時点で、VDO はすでに別のタスクを用意しています。これにより、スレッドがビジー状態に保たれ、タスクの切り替えにかかる時間が短縮されます。VDO は、キューへの I/O 操作の追加や重複排除インデックスへのメッセージの処理など、低速のタスクにも別のスレッドを使用します。

5.1. VDO スレッドの種類

VDO は、さまざまなスレッドタイプを使用して特定の操作を処理します。

論理ゾーンスレッド (kvdo:logQ)

VDO デバイスのユーザーに提示される論理ブロック番号 (LBN) と、基盤となるストレージシステムの物理ブロック番号 (PBN) の間のマッピングを維持します。また、同じブロックへの同時書き込みも防止します。論理スレッドは、読み取り操作と書き込み操作の両方でアクティブになります。処理は通常均等に分散されますが、特定のアクセスパターンにより、1つのスレッドに作業が集中する場合があります。たとえば、特定のブロックマップページ内の LBN に頻繁にアクセスすると、1つの論理スレッドがそれらすべての操作を処理する可能性があります。

物理ゾーンスレッド (kvdo:physQ)

書き込み操作中にデータブロックの割り当てと参照カウントを処理します。

I/O 送信スレッド (kvdo:bioQ)

VDO からストレージシステムへのブロック I/O (**bio**) 操作の転送を処理します。これらのスレッドは、他の VDO スレッドからの I/O リクエストを処理し、基盤となるデバイスドライバーに渡します。これらのスレッドは、デバイス関連のデータ構造と対話し、デバイスドライバーカーネルスレッドのリクエストを作成し、デバイスリクエストキューがいっぱいになって I/O リクエストがブロックされた場合の遅延を防ぎます。

CPU 処理スレッド (kvdo:cpuQ)

他のスレッドタイプのデータ構造をブロックしない、またはデータ構造への排他的アクセスを必要としない、CPU 集中型のタスクを処理します。これらのタスクには、ハッシュ値の計算とデータブロックの圧縮が含まれます。

I/O 完了通知スレッド (kvdo:ackQ)

I/O リクエストの完了を、カーネルページキャッシュやダイレクト I/O を実行するアプリケーションスレッドなどの上位コンポーネントに通知します。メモリー競合への影響と CPU 使用率は、カーネルレベルのコードによる影響を受けます。

ハッシュゾーンスレッド (kvdo:hashQ)

一致するハッシュを使用して I/O リクエストを調整し、潜在的な重複排除タスクを処理します。重複排除リクエストを作成および管理しますが、重要な計算は実行しません。通常、ハッシュゾーンスレッドは1つで十分です。

重複排除スレッド (kvdo:dedupeQ)

I/O リクエストを処理し、重複排除インデックスと通信します。この作業はブロッキング状態を防ぐために別のスレッドで実行されます。また、インデックスがすぐに応答しない場合に重複排除をスキップするタイムアウトメカニズムも備えています。重複排除スレッドは VDO デバイスごとに1つだけ存在します。

ジャーナルスレッド (kvdo:journalQ)

リカバリージャーナルを更新し、ジャーナルブロックの書き込みをスケジュールします。このタスクを複数のスレッドに分割することはできません。ジャーナルスレッドは VDO デバイスごとに 1 つだけ存在します。

パッカースレッド (kvdo:packerQ)

圧縮が有効になっている場合、書き込み操作中に機能します。CPU スレッドから圧縮されたデータブロックを収集して、無駄な領域を削減します。パッカースレッドは VDO デバイスごとに 1 つだけ存在します。

5.2. パフォーマンスのボトルネックの特定

VDO パフォーマンスのボトルネックを特定することは、システム効率を最適化するために重要です。実行できる主要な手順の 1 つは、ボトルネックが CPU、メモリー、バッキングストレージの速度のうちどこにあるのかを判断することです。最も遅いコンポーネントを特定したら、パフォーマンスを向上させる戦略を立てることができます。

パフォーマンス低下の根本原因がハードウェアの問題ではないことを確認するには、ストレージスタックで VDO を使用した場合と使用しない場合のテストを実行します。

VDO の **journalQ** スレッドは、特に VDO ボリュームが書き込み操作を処理している場合には、必然的にボトルネックになります。別のスレッドタイプの使用率が **journalQ** スレッドよりも高いことが判明した場合、当該タイプのスレッドをさらに追加することでこの問題を修正できます。

5.2.1. top による VDO パフォーマンスの分析

top ユーティリティーを使用して、VDO スレッドのパフォーマンスを調べることができます。

手順

1. 個々のスレッドを表示します。

```
$ top -H
```

注記

top などのツールは、生産的な CPU サイクルと、キャッシュまたはメモリーの遅延により停止したサイクルを区別できません。このようなツールは、キャッシュの競合と、低速なメモリーアクセスを実際の動作として解釈します。ノード間でスレッドを移動すると、1 秒あたりの操作数が増加した一方で、CPU 使用率が低下したように見えることがあります。

2. **f** キーを押してフィールドマネージャーを表示します。
3. **(↓)** キーを使用して **P = Last Used Cpu (SMP)** フィールドに移動します。
4. スペースバーを押して、**P = Last Used Cpu (SMP)** フィールドを選択します。
5. **q** キーを押してフィールドマネージャーを閉じます。**top** ユーティリティーは、個々のコアの CPU 負荷を表示し、各プロセスまたはスレッドが最近使用した CPU を示します。**1** を押すと、CPU ごとの統計情報に切り替えることができます。

関連情報

- **top(1)** man ページ

- top の結果の解釈

5.2.2. top の結果の解釈

VDO スレッドのパフォーマンスを分析する際には、次の表を使用して **top** ユーティリティーの結果を解釈してください。

表5.1 top の結果の解釈

値	説明	提案
スレッドまたは CPU 使用率が 70% を超えている。	スレッドまたは CPU がオーバーロード状態です。実際の作業を伴わずに CPU 上でスケジュールされている VDO スレッドにより、使用率が高くなっている可能性があります。この状況は、過度のハードウェア割り込み、メモリーの競合、またはリソースの競合によって発生する可能性があります。	このコアを実行するタイプのスレッドの数を増やします。
%id 値と **%wa** 値が低い。	コアがタスクをアクティブに処理しています。	アクションは不要です。
%hi 値が低い。	コアが標準の処理作業を実行しています。	パフォーマンスを向上させるには、コアを追加します。NUMA の競合を回避します。
• **%hi** 値が高い。[a] • コアにスレッドが 1 つだけ割り当てられている。 • **%id** がゼロである。 • **%wa** 値がゼロである。	コアがオーバーコミットされています。	カーネルスレッドとデバイス割り込み処理を別のコアに再割り当てします。
• **kvdo:bioQ** スレッドが頻繁に **D** 状態になる。	VDO が、ストレージシステムを I/O リクエストで常にビジー状態に保っています。[b]	CPU 使用率が非常に低い場合は、I/O 送信スレッドの数を減らします。
kvdo:bioQ スレッドが頻繁に **S** 状態になる。	VDO に、必要以上の **kvdo:bioQ** スレッドがあります。	**kvdo:bioQ** スレッドの数を減らします。
I/O リクエストあたりの CPU 使用率が高い。	スレッドが増えると、I/O リクエストごとの CPU 使用率が増加します。	CPU、メモリー、またはロックの競合がないか確認します。

値	説明	提案

[a] 数パーセントを超えた値

[b] この状態は、ストレージシステムが複数のリクエストを処理できる場合、またはリクエストの処理が効率的である場合には適切です。

5.2.3. perf による VDO パフォーマンスの分析

perf ユーティリティーを使用して、VDO の CPU パフォーマンスを確認できます。

前提条件

- **perf** パッケージがインストールされている。

手順

1. パフォーマンスプロファイルを表示します。

```
# perf top
```

2. **perf** の結果を解釈して CPU パフォーマンスを分析します。

表5.2 perf の結果の解釈

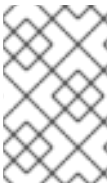
値	説明	提案
kvdo:bioQ スレッドがスピンロックを取得するために過度にサイクルを費やす。	VDO の下のデバイスドライバーで過度の競合が発生している可能性があります。	**kvdo:bioQ** スレッドの数を減らします。
CPU 使用率が高い。	NUMA ノード間で競合が生じています。 プロセッサーが対応している場合は、**stalled-cycles-backend**、**cache-misses**、および **node-load-misses** などのカウンターを確認します。ミス率が高いと、他のツールで CPU 使用率が高い場合と同様にストールが発生する場合があり、競合の可能性があります。	VDO カーネルスレッドの CPU アフィニティーまたは割り込みハンドラーの IRQ アフィニティーを実装して、処理作業を単一ノードに制限します。

関連情報

- man ページの **perf-top(1)**

5.2.4. sar による VDO パフォーマンスの分析

sar ユーティリティーを使用すると、VDO パフォーマンスに関する定期レポートを作成できます。



注記

すべてのブロックデバイスドライバーが **sar** ユーティリティーに必要なデータを提供できるわけではありません。たとえば、MD RAID などのデバイスは **%util** 値を報告しません。

前提条件

- **sysstat** ユーティリティーをインストールします。

```
# dnf install sysstat
```

手順

1. ディスク I/O 統計情報を 1 秒間隔で表示します。

```
$ sar -d 1
```

2. **sar** の結果を解釈して VDO パフォーマンスを分析します。

表5.3 sar の結果の解釈

値	説明	提案
• 基盤となるストレージデバイスの **%util** 値が 100% を大きく下回っている。 • VDO が 100% でビジー状態である。 • **bioQ** スレッドが大量の CPU 時間を使用している。	VDO の **bioQ** スレッドが高速デバイスに対して少なすぎます。	**bioQ** スレッドを追加します。 **bioQ** スレッドを追加すると、スピンロックの競合により、特定のストレージドライバーの速度が低下する可能性があることに注意してください。

関連情報

- **sar(1)** man ページ

5.3. VDO スレッドの再分配

VDO は、リクエストを処理するときに、さまざまなタスクにさまざまなスレッドプールを使用します。最適なパフォーマンスは、各プールのスレッド数を適切に設定することに依存します。適切なスレッド数は、利用可能なストレージ、CPU リソース、ワークロードの種類によって異なります。VDO の作業を複数のスレッドに分散して、VDO パフォーマンスを向上させることができます。

VDO は、並列処理を通じてパフォーマンスを最大化することを目的としています。利用可能な CPU リソースやボトルネックの根本原因などの要因に応じて、ボトルネックになっているタスクに対してより多くのスレッドを割り当てることで、パフォーマンスを向上できます。スレッド使用率が高い (70 -

80% 以上) と遅延が発生する可能性があります。したがって、スレッド数を増やすことが役立ちます。ただし、スレッド数が多すぎるとパフォーマンスが低下し、追加のコストが発生する可能性があります。

最適なパフォーマンスを得るには、次のアクションを実行します。

- 予想されるさまざまなワークロードで VDO をテストし、パフォーマンスを評価して最適化します。
- 使用率が 50% を超えるプールのスレッド数を増やします。
- 個々のスレッドの使用率が低くても、全体の使用率が 50% を超える場合は、VDO で使用できるコアの数を増やします。

5.3.1. NUMA ノード全体での VDO スレッドのグループ化

NUMA ノードをまたぐメモリーへのアクセスは、ローカルメモリーアクセスよりも遅くなります。コアがノード内の最終レベルのキャッシュを共有する Intel プロセッサーでは、単一ノード内でデータが共有される場合よりもノード間でデータが共有される場合に、キャッシュの問題が顕著になります。多くの VDO カーネルスレッドは排他的なデータ構造を管理する一方、しばしば I/O リクエストに関するメッセージを交換します。VDO スレッドが複数のノードに分散している場合や、スケジューラーがノード間でスレッドを再割り当てする場合、競合が発生する可能性があります。具体的には、複数のノードが同じリソースを求めて競合する可能性があります。

特定のスレッドを同じ NUMA ノードにグループ化することで、VDO のパフォーマンスを向上させることができます。

関連するスレッドの 1 つの NUMA ノードへのグループ化

- I/O 完了通知 (**ackQ**) スレッド
- 上位レベルの I/O 送信スレッド:
  - ダイレクト I/O を処理するユーザーモードスレッド
  - カーネルページキャッシュフラッシュスレッド

デバイスアクセスの最適化

- デバイスアクセスのタイミングが NUMA ノード間で異なる場合は、ストレージデバイスコントローラーに最も近いノードで **bioQ** スレッドを実行します。

競合の最小化

- I/O 送信とストレージデバイスの割り込み処理を、**logQ** または **physQ** スレッドと同じノードで実行します。
- 同じノードで他の VDO 関連の作業を実行します。
- 1 つのノードがすべての VDO 作業を処理できない場合は、メモリー競合を考慮してスレッドを他のノードに移動します。たとえば、処理に割り込むデバイスと **bioQ** スレッドを他のノードに移動します。

5.3.2. CPU アフィニティーの設定

VDO スレッドの CPU アフィニティーを調整すると、特定のストレージデバイスドライバーでの VDO パフォーマンスを向上させることができます。

ストレージデバイスドライバーの割り込み (IRQ) ハンドラーが大きな作業を実行し、ドライバーがスレッド化された IRQ ハンドラーを使用しない場合、VDO パフォーマンスを最適化するシステムスケジューラーの機能が制限される可能性があります。

最適なパフォーマンスを得るには、次のアクションを実行します。

- コアがオーバーロード状態になった場合は特定のコアを IRQ 処理専用にし、VDO スレッドアフィニティーを調整します。**%hi** 値が他のコアよりも数パーセント以上高い場合、コアはオーバーロード状態になっています。
- ビジーな IRQ コアでは、**kvdo:journalQ** スレッドなどのシングルトン VDO スレッドを実行しないようにします。
- 他のスレッドタイプも、IRQ でビジーなコアでは実行しないようにします (個々の CPU 使用率が高い場合のみ)。

注記

この設定は、システムを再起動すると元に戻ります。

手順

- CPU アフィニティーを設定します。

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

<cpu-numbers> は、プロセスを割り当てる CPU 番号のコンマ区切りのリストに置き換えます。**<process-id>** は、CPU アフィニティーを設定する実行中のプロセスの ID に置き換えます。

例5.1 CPU コア 1 および 2 にkvdo プロセスの CPU アフィニティーを設定する例

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

検証

- アフィニティーセットを表示します。

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

<cpu-numbers> は、プロセスを割り当てる CPU 番号のコンマ区切りのリストに置き換えます。**<process-id>** は、CPU アフィニティーを設定する実行中のプロセスの ID に置き換えます。

関連情報

- **taskset(1)** の man ページ

5.4. ブロックマップキャッシュサイズの増加

VDO ボリューム全体のキャッシュサイズを増やすことで、読み取りと書き込みの両方のパフォーマンスを向上させることができます。

読み取りおよび書き込みの遅延が増大した場合、またはストレージから読み取られた大量のデータがアプリケーション要件と一致しない場合は、キャッシュサイズの調整が必要な可能性があります。

警告

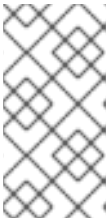
メモリーオーバーヘッドは 15% です。キャッシュが大きくなると、より多くの RAM が消費され、システム全体の安定性に影響を与える可能性があります。

手順

1. **/etc/lvm/profile/<filename>** 設定ファイルに次の行を追加します。

```
vdo_block_map_cache_size_mb=<cache_size>
```

<filename> は、設定ファイルの名前に置き換えます。**<cache_size>** は、キャッシュの新しいサイズに置き換えます。**10G** または **1T** などの接尾辞が指定されていない場合、値はメガバイトとして解釈されます。



注記

キャッシュサイズは 128 MB - 16 TB の範囲内の 4096 の倍数で、論理スレッドあたり少なくとも 16 MB である必要があります。変更は、次回 VDO デバイスが起動されたときに有効になります。すでに実行されているデバイスは影響を受けません。

2. VDO ボリュームを停止します。

```
# vdo stop --name=<volume_name>
```

<volume_name> は、VDO ボリュームの名前に置き換えます。

3. VDO ボリュームを起動します。

```
# vdo start --name=<volume_name>
```

<volume_name> は、VDO ボリュームの名前に置き換えます。

検証

- 現在の VDO ボリュームの設定を確認します。

```
# vdo status --name=<volume_name>
```

<volume_name> は、VDO ボリュームの名前に置き換えます。

関連情報

- **vdo(8)** man ページ

5.5. 破棄操作の高速化

VDO は、システム上のすべての VDO デバイスの DISCARD (TRIM) セクターの最大許容サイズを設定します。デフォルトのサイズは 8 セクターで、1 つの 4-KiB ブロックに相当します。DISCARD サイズを増やすと、破棄操作の速度が大幅に向上します。ただし、破棄のパフォーマンス向上は、他の書き込み操作の速度維持とトレードオフの関係にあります。

最適な DISCARD サイズは、ストレージスタックによって異なります。DISCARD セクターが非常に大きい場合でも非常に小さい場合でも、パフォーマンスが低下する可能性があります。さまざまな値で実験を行い、満足のいく結果が得られる値を見つけてください。

ローカルファイルシステムを保存する VDO ボリュームの場合、デフォルト設定である 8 セクターの DISCARD サイズを使用するのが最適です。SCSI ターゲットとして機能する VDO ボリュームの場合、2048 セクター (1 MB の破棄に相当) など、適度に大きな DISCARD サイズが最適に機能します。最大 DISCARD サイズは 10240 セクター (5 MB の破棄に相当) を超えないようにすることを推奨します。サイズを選択するときは、8 の倍数であることを確認してください。8 セクターより小さい場合、VDO は破棄を効果的に処理できない可能性があります。

手順

1. DISCARD セクターの新しい最大サイズを設定します。

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

   <number-of-sectors> はセクタ数に置き換えます。この設定は再起動するまで維持されます。

2. オプション: DISCARD セクターへの変更を再起動後も永続的に保持するには、カスタム **systemd** サービスを作成します。

   a. 次の内容で新しい **/etc/systemd/system/max_discard_sectors.service** ファイルを作成します。

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

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

   <number-of-sectors> はセクタ数に置き換えます。

   b. ファイルを保存して終了します。

   c. サービスファイルをリロードします。

```
# systemctl daemon-reload
```

   d. 新しいサービスを有効にします。

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

検証

- オプション: スケーリングガバナーの変更を永続化した場合は、**max_discard_sectors.service** が有効になっているかどうかを確認します。

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

5.6. CPU 周波数スケーリングの最適化

デフォルトでは、RHEL は CPU 周波数スケーリングを使用して、CPU に大きな負荷がかかっていないときに電力を節約し、発熱を軽減します。省電力よりもパフォーマンスを優先する場合は、最大クロック速度で動作するように CPU を設定できます。これにより、CPU はデータの重複排除と圧縮のプロセスを最大限の効率で処理できるようになります。CPU を最高周波数で実行することで、リソースを大量に消費する操作をより迅速に実行できるようになり、データ削減とストレージ最適化の点で VDO の全体的なパフォーマンスが向上する可能性があります。

警告

高パフォーマンス向けに CPU 周波数スケーリングをチューニングすると、消費電力と発熱が増加する可能性があります。システムの冷却が不十分な場合、これにより過熱が引き起こされてサーマルスロットルが発生し、パフォーマンスの向上が制限される可能性があります。

手順

1. 利用可能な CPU ガバナーを表示します。

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

2. パフォーマンスを優先するようにスケーリングガバナーを変更します。

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

この設定は再起動するまで維持されます。

3. オプション: スケーリングガバナーの変更を再起動後も永続的に保持するには、カスタム **systemd** サービスを作成します。

a. 次の内容で新しい **/etc/systemd/system/cpufreq.service** ファイルを作成します。

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

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

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

b. ファイルを保存して終了します。

c. サービスファイルをリロードします。

```
# systemctl daemon-reload
```

d. 新しいサービスを有効にします。

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

検証

- 現在使用されている CPU 周波数ポリシーを表示します。

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

- オプション: スケーリングガバナーの変更を永続化した場合は、**cpufreq.service** が有効になっているかどうかを確認します。

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