



Red Hat Ceph Storage 3

アーキテクチャーガイド

Red Hat Ceph Storage アーキテクチャーガイド

Red Hat Ceph Storage 3 アーキテクチャーガイド

Red Hat Ceph Storage アーキテクチャーガイド

Enter your first name here. Enter your surname here.

Enter your organisation's name here. Enter your organisational division here.

Enter your email address here.

法律上の通知

Copyright © 2022 | You need to change the HOLDER entity in the en-US/Architecture_Guide.ent file |.

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

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

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

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

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

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

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

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

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

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

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

All other trademarks are the property of their respective owners.

概要

本書では、Ceph Storage クラスタとそのクライアントのアーキテクチャー情報を提供します。

目次

第1章 概要	3
第2章 ストレージクラスターアーキテクチャー	5
2.1. プール	5
2.2. 認証	6
2.3. 配置グループ (PG)	7
2.4. CRUSH	8
2.5. I/O 操作	8
2.5.1. レプリケートされた I/O	9
2.5.2. イレイジャーコーディング I/O	10
2.6. OBJECTSTORE インターフェース	12
2.6.1. FileStore	12
2.6.2. BlueStore	13
2.7. 自己管理操作	14
2.7.1. ハートビート	15
2.7.2. ピアリング	15
2.7.3. リバランスおよびリカバリー	15
2.7.4. データの整合性の確保	16
2.8. 高可用性	16
2.8.1. Data Copies	16
2.8.2. クラスターの監視	17
2.8.3. CephX	17
第3章 クライアントアーキテクチャー	19
3.1. ネイティブプロトコルおよび LIBRADOS	19
3.2. オブジェクト監視/非表示	19
3.3. 必須排他的ロック	20
3.4. オブジェクトマップ	21
3.5. データストライピング	22
第4章 暗号化	26

第1章 概要

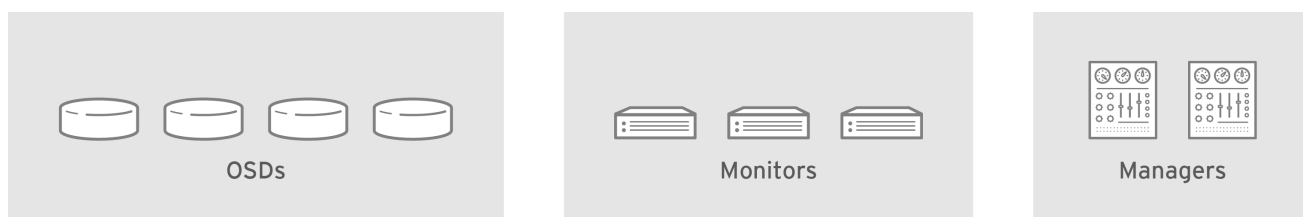
Red Hat Ceph Storage は、優れたパフォーマンス、信頼性、スケーラビリティを提供するように設計された、分散型のデータオブジェクトストアです。分散オブジェクトストアは、非構造化データに対応し、クライアントが最新のオブジェクトインターフェースとレガシーのインターフェースを同時に使用できるため、ストレージの未来と言えます。以下に例を示します。

- 多くの言語の API (C/C++、Java、Python)
- RESTful インターフェース (S3/Swift)
- ブロックデバイスインターフェース
- ファイルシステムインターフェース

Red Hat Ceph Storage の力により、組織の IT インフラストラクチャーを変換でき、特に Red Hat OpenStack Platform などのクラウドコンピューティングプラットフォーム用に大量のデータを管理することができます。Red Hat Ceph Storage は、ペタバイトからエクサバイト以上のデータにアクセスする数千のクライアント向けの追加のスケーラビリティを提供します。

すべての Ceph デプロイメントの中心となるのは、「Ceph Storage Cluster」です。これは、3 種類のデーモンで構成されます。

- **Ceph OSD デーモン:** Ceph OSD は、Ceph クライアントの代わりにデータを格納します。また、Ceph OSD は Ceph ノードの CPU、メモリー、ネットワークを使用して、データの複製、イレイジャーコーディング、リバランス、復旧、監視、レポート作成などの機能を実行します。
- **Ceph Monitor:** Ceph monitor は、ストレージクラスターの現在の状態を備えた Ceph Storage クラスターマッピングのマスターコピーを維持します。モニターには高い一貫性が必要であり、Paxos を使用して Ceph Storage クラスターの状態に関する合意を確保します。
- **Ceph Manager:** RHCS 3 の新機能で、Ceph Manager は Ceph Monitor の代わりに配置グループ、プロセスメタデータ、およびホストメタデータに関する詳細情報を維持します。また、大規模なパフォーマンスが大幅に向上します。Ceph Manager は、配置グループの統計など、読み取り専用の Ceph CLI クエリーの多くの実行を処理します。Ceph Manager は RESTful モニタリング API も提供します。



CEPH_378927_1017

Ceph クライアントインターフェースは、Ceph Storage クラスターとの間でデータの読み取りおよび書き込みを行います。Ceph Storage クラスターとの通信には、クライアントに以下のデータが必要です。

- Ceph 設定ファイル、またはクラスター名 (通常は **ceph**) およびモニターアドレス
- プール名
- ユーザー名およびシークレットキーへのパス。

Ceph クライアントは、オブジェクトを保存するオブジェクト ID とプール名を維持します。ただし、Object-to-OSD インデックスを維持したり、オブジェクトの位置を検索するために集中化オブジェクトインデックスと通信したりする必要はありません。データの保存や取得のために、Ceph クライアントは Ceph monitor にアクセスし、ストレージクラスターマップの最新コピーを取得します。次に、Ceph クライアントは **librados** にオブジェクト名とプール名を提供します。これは、CRUSH (Controlled Replication Under Scalable Hashing) アルゴリズムを使用して、オブジェクトの配置グループと、データの保存と取得のための主要な OSD を計算します。Ceph クライアントは、読み取りおよび書き込み操作を実行することができるプライマリー OSD に接続します。クライアントと OSD には、中間サーバー、ブローカー、またはバスがありません。

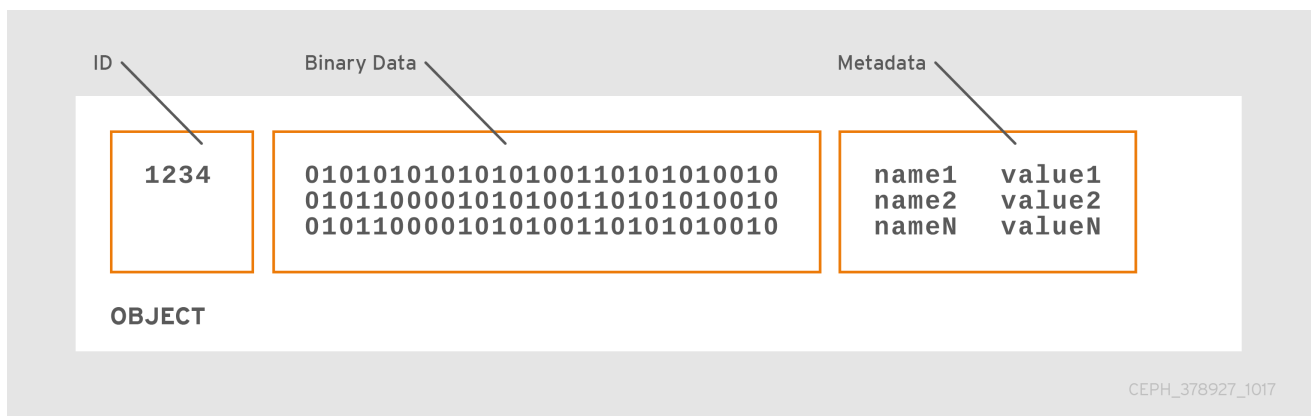
OSD がデータを保存すると、クライアントが Ceph Block Device、Ceph Object Gateway、Ceph Filesystem または別のインターフェースであるかどうかに関わらず、Ceph クライアントからデータを受信し、データをオブジェクトとして格納します。



注記

オブジェクト ID は、OSD のストレージメディアだけでなく、クラスター全体で一意です。

Ceph OSD は、すべてのデータをオブジェクトとしてフラットな namespace に格納します。ディレクトリーの階層はありません。オブジェクトには、クラスター全体で一意的な ID、バイナリーデータ、および名前/値のペアで構成されるメタデータがあります。



Ceph クライアントは、クライアントのデータフォーマットのセマンティクスを定義します。たとえば、Ceph ブロックデバイスはブロックデバイスイメージを、クラスター全体で保存した一連のオブジェクトにマッピングします。



注記

一意的な ID、データ、および名前と値のペアのメタデータで構成されるオブジェクトは、構造化と非構造化のデータの両方だけでなく、レガシーと最新のデータストレージインターフェースを表すことができます。

第2章 ストレージクラスターアーキテクチャー

Ceph Storage クラスターは、制限のないスケーラビリティ、高可用性、およびパフォーマンスのために多数の Ceph ノードを持つことができます。各ノードは、相互に通信する非独占的なハードウェアとインテリジェントな Ceph デモンを活用して、以下を行います。

- データの書き込みと読み取り
- データの圧縮
- データの複製またはイレイジャーコーディングによる持続性の確保
- クラスターの健全性の監視およびレポート (別名「ハートビート」)
- データの動的な再配布 (別名「バックフィル」)
- データの整合性の確認
- 障害からの回復

データの読み取りおよび書き込みを行う Ceph クライアントインターフェースでは、Ceph Storage クラスターはデータを格納する単純なプールのように見えます。ただし、**librados** およびストレージクラスターは、クライアントインターフェースから完全に透過的な方法で多くの複雑な操作を実行します。Ceph クライアントおよび Ceph OSD はどちらも CRUSH (Controlled Replication Under Scalable Hashing) アルゴリズムを使用します。以下のセクションでは、CRUSH がこれらの操作をシームレスに実行できるようにする方法について詳しく説明します。

2.1. プール

Ceph Storage クラスターは、データオブジェクトを「プール」と呼ばれる論理パーティションに格納します。Ceph 管理者は、ブロックデバイス、オブジェクトゲートウェイなどの特定タイプのデータ用に、または単にあるユーザーのグループを別のグループから分離するために、プールを作成することができます。

Ceph クライアントの視点からは、ストレージクラスターは非常に簡単です。Ceph クライアントが i/o コンテキスト経由でデータの読み取りまたは書き込みを行う場合は、常に Ceph Storage クラスターのストレージプールに接続します。クライアントはプール名、ユーザーおよびシークレットキーを指定するため、プールはデータオブジェクトへのアクセス制御が設定された論理パーティションとして機能します。

実際に、Ceph プールは、オブジェクトデータを格納するための論理パーティションのみではありません。プールは、Ceph Storage クラスターがデータを分散し、格納する方法において重要な役割を果たします。ただし、これらの複雑な操作は、Ceph クライアントに対して完全に透過的です。Ceph プールは、以下を定義します。

- **プールタイプ**: Ceph の初期バージョンでは、1つのプールがオブジェクトの複数のディープコピーを保持するだけです。現在、Ceph はオブジェクトの複数のコピーを維持することも、イレイジャーコーディングを使用して耐久性を確保することもできます。データの耐久性の方法はプール全体のものであり、プールの作成後も変更されません。プールタイプは、プールの作成時にデータの持続性メソッドを定義します。プールタイプは、クライアントに対して完全に透過的です。
- **配置グループ**: エクサバイトスケールストレージクラスターでは、Ceph プールには、数百万以上のデータオブジェクトが格納される可能性があります。Ceph は、レプリカまたはイレイジャーコードチャUNKによるデータの耐久性、スクラブまたは CRC チェックによるデータの整合性、レプリケーション、リバランス、リカバリーなど、さまざまな種類の操作を処理する必

必要があります。そのため、オブジェクトごとにデータを管理することは、スケーラビリティとパフォーマンスのボトルネックを示します。Ceph は、プールを配置グループにシャードリングして、このボトルネックに対応します。CRUSH アルゴリズムは、オブジェクトを格納するために配置グループを計算し、配置グループに対して OSD の動作セットを計算します。CRUSH は各オブジェクトを配置グループに配置します。次に、CRUSH は各配置グループを OSD のセットに保存します。システム管理者は、プールを作成または変更する際に配置グループ数を設定します。

- **CRUSH Ruleset:** CRUSH は別の重要な役割を果たします。CRUSH は障害ドメインおよびパフォーマンスドメインを検出できます。CRUSH はストレージメディアタイプで OSD を特定し、OSD をノード、ラック、および行に階層的に編成できます。CRUSH により、Ceph OSD は障害ドメイン全体でオブジェクトのコピーを格納できます。たとえば、オブジェクトのコピーは、異なるサーバーラック、アイル、ラック、およびノードに保存することができます。ラックなど、クラスターの大部分に障害が発生した場合でも、クラスターが回復するまで、クラスターはデグレード状態で動作できます。

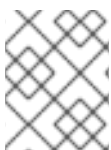
さらに、CRUSH を使用すると、クライアントは、SSD、SSD ジャーナルを備えたハードドライブ、またはデータと同じドライブにジャーナルを備えたハードドライブなど、特定のタイプのハードウェアにデータを書き込むことができます。CRUSH ルールセットは、プールの障害ドメインとパフォーマンスドメインを決定します。管理者は、プールの作成時に CRUSH ルールセットを設定します。注記：管理者は、**プールの作成後にプールのルールセットを変更できません**。

- **耐久性:** エクサバイトスケールストレージクラスターでは、ハードウェア障害は予想され、例外ではありません。データオブジェクトを使用して、ブロックデバイスなどの非常に粒度の高いストレージインターフェースを表す場合に、その非常に粒度の高いインターフェース用に1つ以上のデータオブジェクトが失われると、非常に粒度の高いストレージエンティティの整合性が損なわれ、使用できなくなる可能性があります。したがって、データの損失は許容できません。Ceph は、2つの方法でデータの持続性を提供します。まず、レプリカプールは CRUSH 障害ドメインを使用してオブジェクトの複数のディープコピーを保存し、別のデータオブジェクトのコピーを物理的に分離します。つまり、コピーは別の物理ハードウェアに分散されます。これにより、ハードウェアの障害時の持続性が向上します。2つ目は、イレイジャーコーディングされたプールは各オブジェクトを **K+M** チャンクとして格納します。**K** はデータチャンクを表し、**M** はコーディングチャンクを表します。合計は、オブジェクトを保存するために使用される OSD 数を表し、**M** の値は失敗する可能性がある OSD 数を表します。**M** の値は、OSD が失敗するとデータが復元する回数を表します。

クライアントの観点からは、Ceph はエレガントでシンプルです。クライアントは、プールからの読み取りとプールへの書き込みを行うだけです。ただし、プールは、データの耐久性、パフォーマンス、および高可用性において重要な役割を果たします。

2.2. 認証

ユーザーを特定し、中間者攻撃から保護するために、Ceph は **cephx** 認証システムを提供し、ユーザーおよびデーモンを認証します。



注記

cephx プロトコルは、ネットワーク経由で転送されるデータや OSD に保存されるデータの暗号化には対応しません。

Cephx は共有シークレットキーを使用して認証を行います。つまり、クライアントとモニタークラスターの両方にはクライアントの秘密鍵のコピーがあります。認証プロトコルにより、両当事者は、実際にキーを公開することなく、キーのコピーを持っていることを互いに証明できます。これは相互認証を提供します。つまり、ユーザーがシークレットキーを所有し、ユーザーにはシークレットキーのコピーがあることを確認します。

2.3. 配置グループ (PG)

何百万ものオブジェクトをクラスターに格納し、それらを個別に管理することは、リソースを大量に消費します。そのため、Ceph は配置グループ(PG)を使用して多数のオブジェクトをより効率的に管理します。

PG は、オブジェクトのコレクションを含むために機能するプールのサブセットです。Ceph は、プールを一連の PG にシャードします。次に、CRUSH アルゴリズムはクラスターマップとクラスターのステータスを考慮し、PG をクラスター内の OSD に均等に、かつ疑似ランダムに分散します。

以下は、その仕組みです。

システム管理者はプールを作成すると、CRUSH はプールのユーザー定義の PG を作成します。通常 PG の数は、データの合理的に細分化されたサブセットになります。たとえば、1プールの1OSD あたり 100 PG は、各 PG にプールのデータの約 1% が含まれていることを意味します。

Ceph が PG をある OSD から別の OSD に移動する必要がある場合に、PG の数はパフォーマンスに影響を与えます。プールの PG が少なすぎると、Ceph はデータの大部分を同時に移動し、ネットワークの負荷がクラスターのパフォーマンスに悪影響を及ぼします。プールに PG が多すぎると、Ceph は、データのごく一部を移動する際に CPU および RAM を過剰に使用するので、クラスターのパフォーマンスに悪影響を与えます。パフォーマンスを最適化するために PG 数を計算する方法は、「[PG Count](#)」を参照してください。

Ceph は、オブジェクトのレプリカを保存するか、オブジェクトのイレイジャーコードチャンクを保存することで、データの損失を防ぎます。Ceph は、オブジェクトまたはオブジェクトのイレイジャーコードチャンクを PG 内に格納するため、Ceph は、オブジェクトのコピーごとまたはオブジェクトのイレイジャーコードチャンクごとに、「動作セット」と呼ばれる OSD のセットに各 PG を複製します。システム管理者は、プール内の PG 数と、レプリカまたは消去コードチャンクの数を見ることができます。ただし、CRUSH アルゴリズムは、特定の PG の動作セットに含まれる OSD を計算します。

CRUSH アルゴリズムおよび PGs は Ceph を動的にします。クラスターマップまたはクラスターの状態の変更により、Ceph が PG をある OSD から別の OSD に自動的に移動させる可能性があります。以下にいくつかの例を示します。

- **クラスターの拡張:** 新規ホストとその OSD をクラスターに追加すると、クラスターマップが変更されます。CRUSH は PG を OSD にクラスター全体に疑似ランダムに分散するため、新規ホストおよびその OSD を追加すると、CRUSH はプールの配置グループの一部をそれらの新規 OSD に再度割り当てます。つまり、システム管理者は、クラスターを手動でリバランスする必要はありません。また、新規の OSD には、他の OSD とほぼ同じ量のデータが含まれていることを意味します。これは、新規の OSD に新規作成の OSD が含まれていないことも意味し、クラスター内の「ホットスポット」を防ぎます。
- **OSD 失敗:** OSD が失敗すると、クラスターの状態が変わります。Ceph は、レプリカまたはイレイジャーコードのチャンクの1つを一時的に失い、別のコピーを作成する必要があります。動作セットのプライマリー OSD が失敗すると、動作セットの次の OSD がプライマリーになり、CRUSH は新規の OSD を計算して、追加のコピーまたはイレイジャーコードチャンクを格納します。

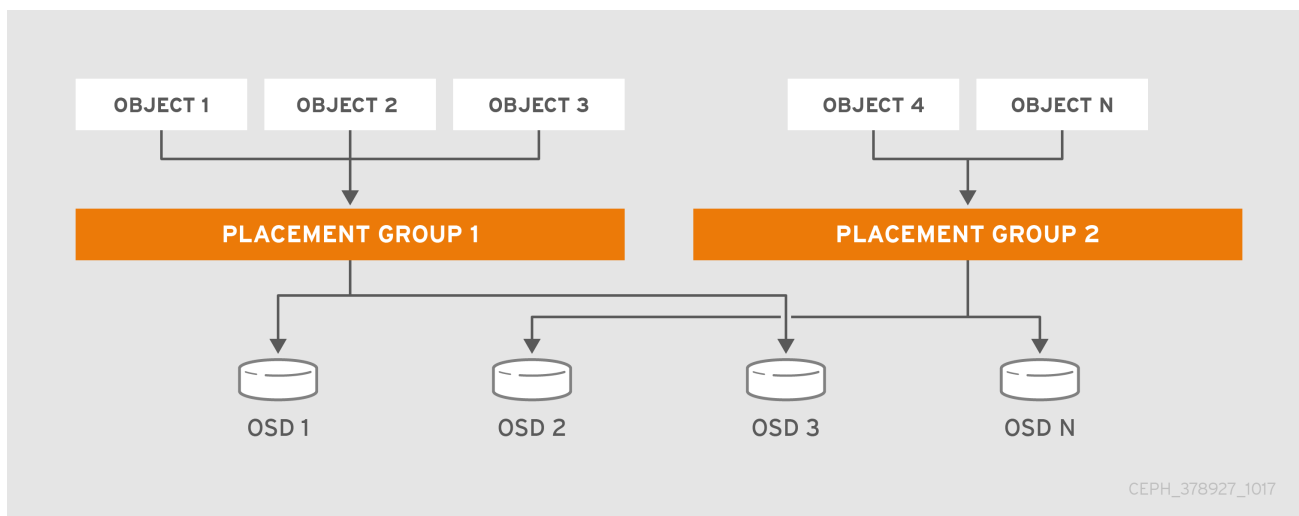
何十万もの PG のコンテキスト内で何百万ものオブジェクトを管理することで、Ceph Storage クラスターは効率的に拡張、縮小、および障害からの回復を行うことができます。

Ceph クライアントの場合は、**librados** を介した CRUSH アルゴリズムにより、オブジェクトの読み取りおよび書き込みプロセスが非常にシンプルになります。Ceph クライアントは単にオブジェクトをプールに書き込むか、プールからオブジェクトを読み取ります。動作セットのプライマリー OSD は、Ceph クライアントに代わって、オブジェクトのレプリカまたはオブジェクトのイレイジャーコードチャンクを動作セットのセカンダリー OSD に書き込むことができます。

クラスターマップまたはクラスターの状態が変更されると、OSD が PG を格納する CRUSH 計算も変更されます。たとえば、Ceph クライアントはオブジェクト **foo** をプール **bar** に書き込むことができます。CRUSH はオブジェクトを PG **1.a** に割り当て、これを **OSD 5** に保存します。これにより、**OSD 10** および **OSD 15** のレプリカがそれぞれ作成されます。**OSD 5** が失敗すると、クラスターの状態が変わります。Ceph クライアントがプール **bar** からオブジェクト **foo** を読み込むと、**librados** 経由のクライアントは新しいプライマリー OSD として **OSD 10** から自動的に取得されます。

librados を使用する Ceph クライアントは、オブジェクトの書き込みおよび読み取り時に、動作セット内でプライマリー OSD に直接接続します。I/O 操作は集中化ブローカーを使用しないため、ネットワークのオーバーサブスクリプションは通常、Ceph の問題ではありません。

以下の図は、CRUSH がオブジェクトを PG に割り当てる方法、および PG を OSD に割り当てる方法を示しています。CRUSH アルゴリズムでは、動作セットの各 OSD が別の障害ドメインに置かれるように PG を OSD に割り当てます。これは通常、OSD が常に別々のサーバーホストに置かれ、別のラックに置かれることを意味します。



2.4. CRUSH

Ceph は CRUSH ルールセットをプールに割り当てます。Ceph クライアントがプールにデータを保存または取得する場合、Ceph は CRUSH ルールセット、ルールセット内のルール、およびデータを保存および取得するためのルールの最上位バケットを特定します。Ceph が CRUSH ルールを処理する際に、オブジェクトの配置グループが含まれるプライマリー OSD を特定します。これにより、クライアントは OSD に直接接続し、配置グループにアクセスして、オブジェクトデータの読み取りまたは書き込みを行うことができます。

配置グループを OSD にマッピングするには、CRUSH マップはバケットタイプの階層リストを定義します。バケットタイプの一覧は、生成された CRUSH マップの **types** の下にあります。バケット階層を作成する目的は、ドライブタイプ、ホスト、シャーシ、ラック、電源分散ユニット、Pod、行、部屋、およびデータセンターなどの障害ドメインやパフォーマンスドメインによって、リーフノードを分離することです。

OSD を表すリーフノードを除き、残りの階層は任意になります。デフォルトのタイプが要件に適さない場合、管理者は独自のニーズに合わせて定義することができます。CRUSH は、通常は階層内の Ceph OSD ノードをモデル化する有向非巡回グラフをサポートします。そのため、Ceph 管理者は、単一の CRUSH マップで複数の root ノードを持つ複数の階層をサポートできます。たとえば、管理者は、高パフォーマンス用に高コストの SSD を表す階層を作成したり、中程度のパフォーマンス用に SSD ジャーナルを備えた低コストのハードドライブの別の階層を作成したりできます。

2.5. I/O 操作

Ceph クライアントは、Ceph モニターから「クラスターマップ」を取得し、プールにバインドし、プール内の配置グループ内のオブジェクトで i/o を実行します。プールの CRUSH ルールセットと配置グループの数は、Ceph がデータを配置する方法を決定する主要な要因です。クラスターマップの最新バージョンでは、クライアントは、クラスター内のすべてのモニターおよび OSD、およびそれらの現在の状態を認識します。ただし、クライアントはオブジェクトの場所について何も知りません。

クライアントが必要とする入力、オブジェクト ID とプール名のみです。これは単純な方法です。Ceph はデータを名前付きプールに保管します。クライアントがプールに名前付きのオブジェクトを保存する場合は、オブジェクト名、ハッシュコード、プール名の PG 数およびプール名を入力として取り、次に CRUSH (Controlled Replication Under Scalable Hashing) は配置グループの ID および配置グループのプライマリー OSD を計算します。

Ceph クライアントは、以下の手順を使用して PG ID を計算します。

1. クライアントはプール ID とオブジェクト ID を入力します。たとえば、**pool = liverpool** および **object-id = john** です。
2. CRUSH はオブジェクト ID を取得し、それをハッシュします。
3. CRUSH は PG 数のハッシュモジュロを計算し、PG ID を取得します。たとえば **58** です。
4. CRUSH は PG ID に対応するプライマリー OSD を計算します。
5. クライアントはプール名が指定されたプール ID を取得します。たとえば、プール「liverpool」はプール番号 **4** です。
6. クライアントはプール ID を PG ID の前に追加します。たとえば **4.58** です。
7. クライアントは、動作セットのプライマリー OSD と直接通信することにより、書き込み、読み取り、削除などのオブジェクト操作を実行します。

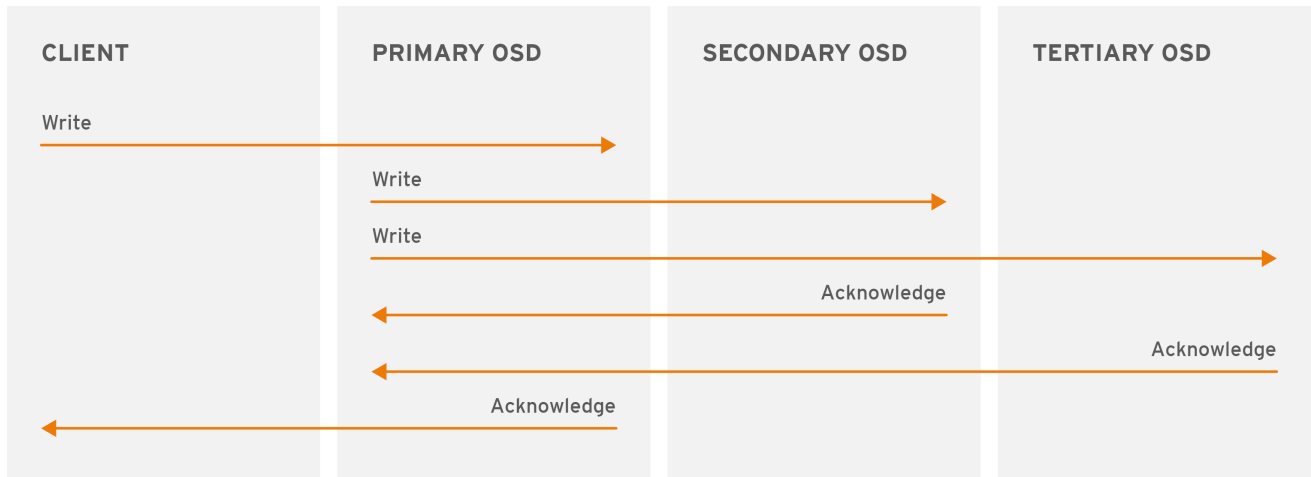
Ceph Storage クラスターのトポロジおよび状態は、セッション時に比較的安定しています。librados を介して Ceph クライアントにオブジェクトの場所を計算する権限を付与することは、クライアントが読み取り/書き込み操作ごとにチャットセッションを介してストレージクラスターにクエリーを実行することを要求するよりもはるかに高速です。CRUSH アルゴリズムにより、クライアントはオブジェクトが保存される場所を計算でき、クライアントが動作セットのプライマリー OSD に直接アクセスできるようにし、オブジェクト内のデータを保存または取得できるようにします。エクサバイト規模のクラスターには数千もの OSD があるため、クライアントと Ceph OSD 間のネットワークオーバーサブスクリプションは重大な問題ではありません。クラスターの状態が変更された場合は、クライアントは単に Ceph モニターからクラスターマップへ更新をリクエストすることができます。

RHCS 2 以前のリリースでは、非常に大規模なクラスターのデーモンは、クラスターマップのサイズが大きすぎるとパフォーマンスが遅くなる可能性があります。たとえば、OSD が 10k のクラスターには OSD ごとに 100 PG がある場合があり、これにより、データを効率的に分散するために ~1M PG が使用され、クラスターマップ用に多数のエポックが分散されます。そのため、デーモンは、非常に大きなクラスターを持つ RHCS 2 でより多くの CPU および RAM を使用します。RHCS 3 以降のリリースでは、デーモンは RHCS 2 以前のリリースのようにクラスターの現在の状態を受け取ります。ただし、Ceph Manager (**ceph-mgr**) デーモンは PG のクエリーを処理するようになり、大規模でパフォーマンスが大幅に改善されました。Red Hat は、数千の OSD を持つ非常に大規模なクラスターに RHCS 3 以降のリリースを使用することを推奨します。

2.5.1. レプリケートされた I/O

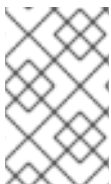
Ceph クライアントと同様に、Ceph OSD は Ceph モニターに接続してクラスターマップの最新コピーを取得できます。Ceph OSD も CRUSH アルゴリズムを使用しますが、これを使用してオブジェクトのレプリカを保存する場所を計算します。標準的な書き込みシナリオでは、Ceph クライアントは CRUSH アルゴリズムを使用して、オブジェクトの動作セット内の配置グループ ID とプライマリー

OSD を計算します。クライアントがプライマリー OSD にオブジェクトを書き込む際に、プライマリー OSD は格納する必要のあるレプリカ数を見つけます。値は `osd_pool_default_size` 設定にあります。次に、プライマリー OSD は、オブジェクト ID、プール名、およびクラスターマップを取得し、CRUSH アルゴリズムを使用して動作セットのセカンダリー OSD の ID を計算します。プライマリー OSD はオブジェクトをセカンダリー OSD に書き込みます。プライマリー OSD がセカンダリー OSD から確認応答を受信し、プライマリー OSD 自体がその書き込み操作を完了すると、Ceph クライアントへの書き込み操作が成功したことを確認します。



CEPH_378927_1017

Ceph クライアントの代わりにデータレプリケーションを実行する機能を使用すると、Ceph OSD デモンは、高いデータ可用性とデータの安全性を確保しながら、Ceph クライアントをその責務から解放します。



注記

プライマリー OSD とセカンダリー OSD は通常、別の障害ドメインに配置するように設定されています。CRUSH は、障害ドメインについて考慮し、セカンダリー OSD の ID を計算します。

2.5.2. イレイジャーコーディング I/O

Ceph は、多くのイレイジャーコードアルゴリズムのいずれかを読み込むことができます。**Reed-Solomon** アルゴリズムが最も早く一般的に使用されるものになります。イレイジャーコードは、実際には forward error correction(FEC)コードで、**K** チャンクのメッセージを **N** チャンクの「コード単語」と呼ばれる長いメッセージに変換します。これにより、Ceph は **N** チャンクのサブセットから元のメッセージを復元できます。

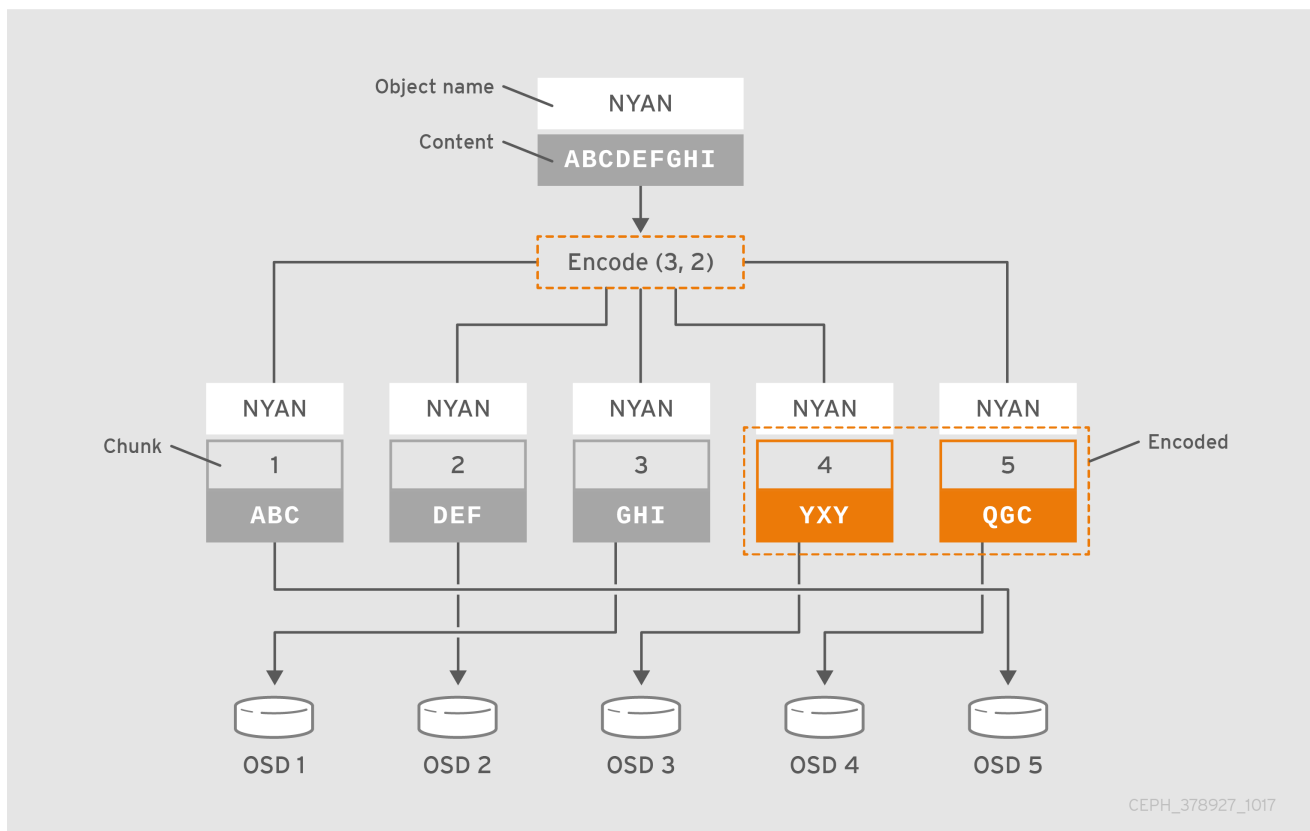
具体的には、変数 **K** が元のデータチャンク量である $N = K + M$ です。変数 **M** は追加または冗長チャンクを表し、イレイジャーコードアルゴリズムが障害から保護されるようにし、変数 **N** はイレイジャーコーディングプロセス後に作成されたチャンクの合計数です。**M** の値は $N - K$ です。これは、アルゴリズムが **K** の元のデータチャンクから $N - K$ 冗長チャンクを計算することを意味します。このアプローチにより、Ceph が元のデータすべてにアクセスできることを保証します。システムが任意の $N - K$ の障害に対して回復性があります。たとえば、16 の **N** 設定のうち 10 **K**、またはイレイジャーコーディング **10/16** の場合、イレイジャーコードアルゴリズムは 10 ベースチャンク **K** に 6 つの追加チャンクを追加します。たとえば、 $M = K - N$ または $16 - 10 = 6$ の設定では、Ceph は 16 チャンク **N** を 16 OSD に分散します。元のファイルは、6 OSD が失敗しても 10 つの検証された **N** チャンクから再構築でき、Ceph クラスタはデータが失われないようにし、非常に高いフォールトトレランスを確保します。

複製されたプールと同様に、イレイジャーコーディングされたプールでは、セットアップ内のプライマリー OSD がすべての書き込み操作を受け取ります。複製されたプールでは、Ceph はセットのセカンダリー OSD 上の配置グループで各オブジェクトのディープコピーを作成します。イレイジャーコー

ディングの場合、プロセスは少し異なります。コード化されたプールは各オブジェクトを **K+M** チャンクとして格納します。これは **K** データチャンクと **M** コーディングチャンクに分割されます。プールには **K+M** のサイズが設定され、これにより Ceph が各チャンクを動作セットの OSD に保管することができます。Ceph は、チャンクのランクをオブジェクトの属性として保存します。プライマリー OSD はペイロードを **K+M** チャンクにエンコードし、それらを他の OSD に送信します。プライマリー OSD は、配置グループログの権威バージョンを維持する役割も果たします。

たとえば、一般的な構成では、システム管理者は 5 つの OSD を使用し、そのうちの 2 つの OSD の損失を維持するために、イレイジャーコード化されたプールを作成します。つまり、**(K+M = 5)** であり、**(M = 2)** になります。

Ceph が **ABCDEFGHI** を含むオブジェクト **NYAN** をプールに書き込むと、イレイジャーエンコーディングアルゴリズムは、コンテンツを 3 つに分割するだけで、コンテンツを 3 つのデータチャンクに分割します。1 つ目は **ABC**、2 番目の **DEF**、および最後の **GHI** が含まれます。コンテンツの長さが **K** の倍数でない場合は、アルゴリズムによりコンテンツをパディングします。この関数は、2 つのコーディングチャンクも作成します。4 つ目は **YXY** で、5 つ目は **QGC** が付きます。Ceph は、動作セット内の OSD 上にそれぞれのチャンクを保存します。ここで、**NYAN** の名前を持つオブジェクトにチャンクが保管されますが、異なる OSD にあります。アルゴリズムは、名前に加えて、チャンクをオブジェクト **shard_t** の属性として作成した順番を保持する必要があります。たとえば、チャンク 1 には **ABC** が含まれ、Ceph はこれを **OSD5** に格納されます。一方、チャンク 4 には **YXY** が含まれ、**OSD3** に格納されます。



リカバリーのシナリオでは、クライアントはチャンク 1 から 5 を読み取ることで、イレイジャーコード化されたプールからオブジェクト **NYAN** を読み込もうとします。OSD は、2 と 5 のチャンクがないことをアルゴリズムに通知します。これらのチャンクは「イレイジャー」と呼ばれます。たとえば、**OSD4** が除外されているため、プライマリー OSD はチャンク 5 を読み取ることができませんでした。また、**OSD2** は最も遅く、そのチャンクを考慮していなかったため、チャンク 2 を読み取ることができませんでした。ただし、アルゴリズムにチャンクが 3 つあると、すぐに 3 つのチャンク (**ABC** を含むチャンク 1、**GHI** を含むチャンク 3、**YXY** を含むチャンク 4) を読み取ります。次に、オブジェクト **ABCDEFGHI** の元のコンテンツと、**QGC** を含むチャンク 5 の元のコンテンツを再構築します。

データをチャンクに分割することは、オブジェクトの配置とは無関係です。CRUSH ルールセットとイ

レイジャーコーディングされたプールプロファイルにより、OSD 上のチャンクの配置が決定されます。たとえば、イレイジャーコードプロファイルで Locally Repairable Code (**lrc**) プラグインを使用すると、追加のチャンクが作成され、回復に必要な OSD が少なくなります。たとえば、**lrc** プロファイル構成 **K=4 M=2 L=3** では、アルゴリズムは、**jerasure** プラグインと同じように 6 つのチャンク (**K+M**) を作成しますが、局所性の値 (**L=3**) は、ローカルにさらに 2 つのアルゴリズムを作成する必要があります。アルゴリズムは、(**K+M**)/**L** などの追加チャンクを作成します。チャンク 0 を含む OSD が失敗すると、チャンク 1、2、および最初のローカルチャンクを使用してこのチャンクを回復できます。この場合、アルゴリズムは 5 つではなく 3 つのチャンクのみを回復に必要とします。CRUSH、イレイジャーコーディングプロファイル、およびプラグインの詳細は、Red Hat Ceph Storage 3 の『ストレージ戦略』を参照してください。



注記

イレイジャーコーディングされたプールを使用すると、オブジェクトマップが無効になります。オブジェクトマップの詳細は、「オブジェクトマップ」セクションを参照してください。

2.6. OBJECTSTORE インターフェース

Objectstore は、OSD の raw ブロックデバイスに低レベルのインターフェースを提供します。クライアントがデータの読み取りまたは書き込みを行うと、**ObjectStore** インターフェースと対話します。Ceph 書き込み操作は、基本的に ACID トランザクションです。つまり、**原子性**、**一貫性**、**分離**、および **耐久性** を提供します。**ObjectStore** は、**トランザクション** が **原子性** を提供するために、オールオアナッシングになるようにします。**ObjectStore** は、オブジェクトセマンティクスも処理します。Overview セクションにあるように、ストレージクラスターに保存されているオブジェクトには、一意の識別子、オブジェクトデータ、およびメタデータがあります。したがって、**ObjectStore** は Ceph オブジェクトセマンティクスが正しいことを確認して **整合性** を提供します。**ObjectStore** は、書き込み操作で **Sequencer** を呼び出して ACID トランザクションの **分離** 部分も提供し、Ceph 書き込み操作が順番に実行されるようにします。これとは対照的に、OSD のレプリケーションまたはイレイジャーコーディング機能は ACID トランザクションの **Durability** コンポーネントを提供します。**ObjectStore** は、ストレージメディアへの低レベルのインターフェースであるため、パフォーマンスの統計も提供します。

Ceph は、データを保管するための具体的な方法を実装します。

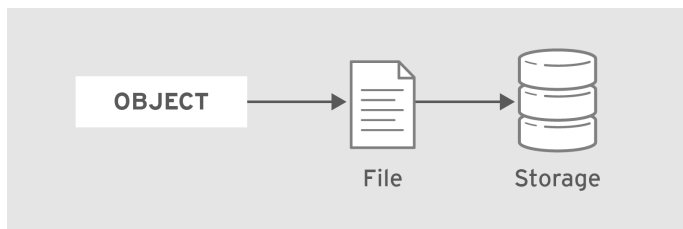
- **FileStore**: オブジェクトデータを保存するファイルシステムを使用した実稼働グレードの実装。
- **BlueStore**: オブジェクトデータを保存するために raw ブロックデバイスを使用した実稼働クラスの実装。
- **Memstore**: RAM で読み取り/書き込み操作を直接テストするための開発者の実装。
- **K/V Store**: Ceph がキー/値データベースを使用する内部実装。

通常、管理者は **FileStore** および **BlueStore** にのみ対応するため、以下のセクションではこれらの実装についてのみ詳しく説明します。

2.6.1. FileStore

FileStore は Ceph の元のストレージ実装の 1 つで、最も広く使用されている実装です。Ceph プロジェクトは 2004 年に開始した場合、Ceph はストレージのハードディスクドライブのみに依存します。これは、ソリッドステートドライブまたは PCI 表現を介した不揮発性メモリーへの有線性ありませんでした。**FileStore** は、raw ブロックデバイスと直接対話するのではなく、ファイルシステム（通常は **xfs**

）と対話します。**ObjectStore** がオブジェクトのセマンティクスを処理して **FileStore** に渡すと、**FileStore** は配置グループをディレクトリー、オブジェクトをファイルとして扱い、メタデータを XATTR または **omap** エントリーとして扱います。



CEPH 378927 1017

FileStore は、オープンソースのファイルシステムのセマンティクスを活用し、別のドライブでジャーナルトランザクションを使用する利点を提供します。**FileStore** には 不利な点もあります。**FileStore** の欠点の1つは、Ceph 書き込み操作は基本的に ACID トランザクションであることです。**原子性** を実現するため、Ceph **FileStore** はデータを書き込む前に **すべて** のトランザクションをジャーナル化します。データのジャーナリングと書き込みに同じドライブを使用する場合は、書き込みレイテンシーが大きくなり、書き込みペナルティーが発生します。開発者は、トランザクションがあるのにコピーオンライトセマンティクスがあり、データを同時に書き込む可能性があるため、**btrfs** ファイルシステムのフォーマットが最終的にデフォルトのファイルシステムフォーマットであると見なしています。ただし、**btrfs** は、実稼働システムの Ceph プロジェクトの信頼性要件を満たさない。したがって、**FileStore** は通常 **btrfs** の代わりに **ext4** ファイルシステムおよび **xfs** ファイルシステムを使用します。

ext4 の短所の1つは、XATTRs (約 4k) のストレージが非常に制限されていることです。したがって、**xfs** は **FileStore** の推奨ファイルシステムになりました。これは、特に約 64k の XATTR のストレージが多いためです。ただし、64k は、大規模なサムネイル画像を持つ可能性のあるムービルの図など、大規模なメタデータを必要とするオブジェクトの制限です。**ObjectStore** および **FileStore** は **ObjectMap** または **omap** で拡張されました。

Ceph の商用サポートが 2012 で起動されると、SSD は引き続き高価でした。ただし、SSD ドライブでジャーナル書き込み操作を実行し、高速 SAS ハードディスクドライブにオブジェクトデータを保存する機能を使用すると、高スループットおよび高 I/O ワークロードのパフォーマンス特性が提供されます。SSD へのジャーナリングによりパフォーマンスが大幅に向上しましたが、**xfs** は引き続きトランザクションを書き込んでから書き込んでいました。そのため、SSD でペナルティーが低くしても、**FileStore** では 2 回書き込みペナルティーが維持されます。

FileStore は配置グループをディレクトリーとして処理します。初期クラスターの場合、問題の有無はほとんど表示されません。ただし、Ceph クラスターは拡張する傾向があります。管理者は新規ノードおよび OSD を追加する際に、配置グループの数を増やす必要があります。**FileStore** では、オブジェクトデータはディレクトリー内のファイルとして、配置グループを表すディレクトリーに置かれます。したがって、配置グループの数を増やす場合は、ファイル内のオブジェクトデータは異なるディレクトリーに移動する必要があります。Ceph は 32 ビットのハッシュアルゴリズムを使用してオブジェクトをランダムに分散します。その一部はオブジェクトのファイル名に組み込まれています。これは、データをアドレス指定および再分配するための非効率的なアプローチです。

2.6.2. BlueStore

BlueStore は Ceph の次世代ストレージ実装です。ストレージデバイスの市場にはソリッドステートドライブ、SSD、PCI Express または NVMe 経由の不揮発性メモリーが含まれるようになったため、Ceph での使用により、**FileStore** ストレージの実装の制限の一部が示されます。**FileStore** には SSD および NVMe ストレージを容易にするための多くの改良がありますが、その他の制限は残ります。また、配置グループを増やすと、計算コストが続き、二重書き込みペナルティーが残ります。**FileStore** はブロックデバイスのファイルシステムと対話し、**BlueStore** はその間接状態をなくし、オブジェクトストレージ用に raw ブロックデバイスを直接使用します。**BlueStore** は、k/v データベース用に小規模なパーティションで非常に軽量の **BlueFS** ファイルシステムを使用します。**BlueStore** では、配置グループを表すディレクトリー (オブジェクトを表すファイルおよびメタデータを表すファイルの) の

XATTR がなくなります。**BlueStore** では **FileStore** の 2 回の書き込みペナルティーが解消されるため、ほとんどのワークロードで **BlueStore** を使用する場合に、書き込み操作はほぼ 2 倍速くなります。

BlueStore は以下のようにデータを保存します。

- オブジェクトデータ:** **BlueStore** では、Ceph はオブジェクトを raw ブロックデバイスに直接ブロックとして保存します。オブジェクトデータを格納する raw ブロックデバイスの部分にはファイルシステムが含まれません。ファイルシステム省略により間接的な層が排除され、パフォーマンスが向上します。ただし、**BlueStore** のパフォーマンスのほとんどは、ブロックデータベースと write-ahead ログにより向上されます。
- ブロックデータベース:** **BlueStore** では、**整合性** を保証するために、ブロックデータベースがオブジェクトのセマンティクスを処理します。オブジェクトの一意の ID はブロックデータベースのキーです。ブロックデータベースの値は、格納したオブジェクトデータ、オブジェクトの配置グループ、およびオブジェクトメタデータを参照する一連のブロックアドレスで構成されます。ブロックデータベースは、オブジェクトデータを格納する同じ raw ブロックデバイス上の **BlueFS** パーティションに存在する場合もあれば、別のブロックデバイスに存在する場合もあります。通常、ハードディスクドライブがプライマリーブロックデバイスであり、SSD または NVMe によってパフォーマンスが向上します。ブロックデータベースは、**FileStore** に対して多くの改善を提供します。つまり、**BlueStore** のキー/値のセマンティクスは、ファイルシステム XATTR の制限に悪影響を与えません。また、**BlueStore** は、**FileStore** の場合のように、あるディレクトリーから別のディレクトリーにファイルを移動するオーバーヘッドなしに、ブロックデータベース内の他の配置グループにオブジェクトをすぐに割り当てることができます。**BlueStore** には新機能も導入されています。ブロックデータベースは、保存されたオブジェクトデータとそのメタデータのチェックサムを保存できるため、読み取りごとに完全なデータチェックサム操作が可能になります。これは、ビットロットを検出するための定期的なスクラブよりも効率的です。**BlueStore** はオブジェクトを圧縮でき、ブロックデータベースはオブジェクトの圧縮に使用されるアルゴリズムを保存できます。これにより、読み取り操作で解凍に適切なアルゴリズムが選択されます。
- 先行書き込みログ:** **BlueStore** では、**FileStore** のジャーナリング機能と同様に、先行書き込みログによって **原子性** が保証されます。**FileStore** と同様に、**BlueStore** は各トランザクションのすべての要素をログに記録します。ただし、**BlueStore** の先行書き込みログまたは WAL は、この機能を同時に実行できるため、**FileStore** の二重書き込みペナルティーがなくなります。その結果、**BlueStore** は、ほとんどのワークロードの書き込み操作で **FileStore** のほぼ 2 倍の速度になります。**BlueStore** は、オブジェクトデータを保存するために同じデバイスに WAL をデプロイできます。または、通常、プライマリーブロックデバイスがハードディスクドライブである場合や、SSD または NVMe によってパフォーマンスが向上する場合は、WAL を別のデバイスにデプロイできます。



注記

個別のデバイスがプライマリーストレージデバイスよりも高速である場合にのみ、ブロックデータベースまたは先行書き込みログを別のブロックデバイスに保存すると便利です。たとえば、SSD デバイスおよび NVMe デバイスは通常 HDD よりも高速です。ブロックデータベースと WAL を別のデバイスに配置すると、ワークロードの違いにより、パフォーマンス上の利点があります。

2.7. 自己管理操作

Ceph クラスターは、多くの自己監視および管理操作を自動的に実行します。たとえば、Ceph OSD はクラスターの正常性を確認し、Ceph モニターに報告することができます。CRUSH を使用してオブジェクトを配置グループに割り当て、配置グループを OSD のセットに割り当てることにより、Ceph

OSD は CRUSH アルゴリズムを使用して、クラスターのバランスを取り直したり、OSD 障害から動的に回復したりできます。以下のセクションでは、Ceph が実行する操作の一部を説明します。

2.7.1. ハートビート

Ceph OSD はクラスターに参加し、Ceph モニターのステータスを報告します。最下位レベルでは、Ceph OSD ステータスは **up** または **down** で、実行中で Ceph クライアント要求を処理できるかどうかを反映しています。Ceph OSD が **down** で、Ceph Storage クラスターの **in** (内) にある場合、このステータスは、Ceph OSD の失敗を示す可能性があります。Ceph OSD が実行していない場合は、クラッシュします。Ceph OSD は、Ceph モニターが **ダウン** していることを通知しません。Ceph モニターは Ceph OSD デーモンを定期的に ping して、実行中であることを確認できます。ただし、ハートビートにより、Ceph OSD は、隣接 OSD が **ダウン** しているかどうかを判断し、クラスターマップを更新して Ceph モニターに報告します。これは、Ceph モニターが軽量の重みプロセスを維持できることを意味します。

2.7.2. ピアリング

Ceph は、複数の OSD 上の配置グループのコピーを保存します。配置グループのコピーごとにステータスがあります。これらの OSD は「ピア」または相互にチェックし、PG の各コピーのステータスに同意していることを確認します。通常、ピアリングの問題は自己解決します。



注記

Ceph モニターが配置グループを保存する OSD の状態に同意する場合、配置グループが最新のコンテンツを持っていることを意味しません。

Ceph が配置グループを有効な OSD セットに保存する場合は、それらを **プライマリー**、**セカンダリー** などとして参照します。通常、**プライマリー** は **動作セット** の最初の OSD になります。配置グループの最初のコピーを保存する **プライマリー** は、その配置グループのピアプロセスを調整する役割を果たします。**プライマリー** は、指定した配置グループのオブジェクトに対してクライアントが開始する書き込みを許可する唯一の OSD で、**プライマリー** として機能します。

動作セット は、配置グループを格納する一連の OSD です。**動作セット** は、現在配置グループを担当している Ceph OSD デーモンや、あるエポックの時点で特定の配置グループを担当していた Ceph OSD デーモンを指す場合があります。

動作セット に含まれる Ceph OSD デーモンは常に **up** であるとは限りません。**動作セット** の OSD が **up** であると、これは Up Set の一部になります。OSD に障害が発生した場合に Ceph は PG を他の Ceph OSD に再マッピングできるため、Up Set は重要な違いです。



注記

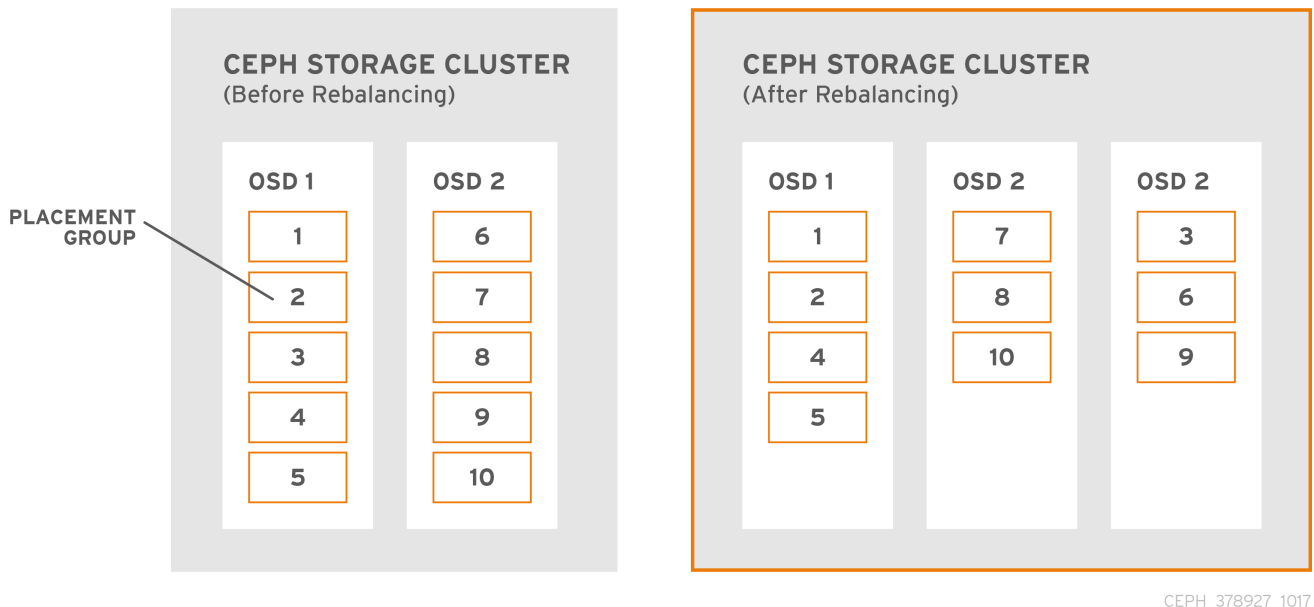
osd.25、**osd.32**、および **osd.61** を含む PG の Acting Set では、最初の OSD **osd.25** は **プライマリー** です。OSD が失敗すると、**セカンダリー** では、**osd.32** が **プライマリー** になり、Ceph は Up Set から **osd.25** を削除します。

2.7.3. リバランスおよびリカバリー

管理者が Ceph OSD を Ceph Storage クラスターに追加する際に、Ceph はクラスターマップを更新します。変更されたクラスターマップは CRUSH 計算の入力を変更するため、クラスターマップへのこの変更は、オブジェクトの配置も変更します。CRUSH はデータを均等に配置しますが、疑似ランダムに配置します。そのため、管理者が新しい OSD を追加すると、移動するデータはごくわずかになります。

す。通常、データ量は、クラスター内のデータの合計量で分割される新規 OSD 数です。たとえば、50 個の OSD があるクラスターでは、OSD を追加すると、データの 1/50 または 2% が移動する可能性があります。

以下の図は、一部の PG が既存の OSD (OSD 1 および OSD 2) から新規 OSD (OSD 3) に移行する一部のリバランスプロセスを示しています。リバランスを行う場合でも、CRUSH は安定しています。配置グループの多くは元の設定のままであり、各 OSD には容量が追加されるため、クラスターのリバランス後に新しい OSD に負荷が急増することはありません。



2.7.4. データの整合性の確保

データの整合性を維持する一環として、Ceph は、不良ディスクセクターやビットロットを防ぐための多数のメカニズムを提供します。

- **スクラブ:** Ceph OSD デーモンは配置グループ内のオブジェクトをスクラブすることができます。つまり、Ceph OSD デーモンは、1つの配置グループのオブジェクトメタデータを、他の OSD に保存されている配置グループのレプリカと比較できます。スクラビング (通常は毎日実行) は、バグやストレージエラーをキャッチします。Ceph OSD デーモンは、オブジェクト内のデータをビットごとに比較することにより、より深いスクラビングも実行します。ディープスクラビング (通常は毎週実行) は、軽量スクラブでは見られなかったドライブ上の不良セクターを見つけます。
- **CRC チェック:** BlueStore を使用する場合の RHCS 3 では、Ceph は書き込み操作で CRC(cyclical redundancy check)を実行することにより、データの整合性を確保できます。次に、CRC 値をブロックデータベースに保存します。読み取り操作では、Ceph はブロックデータベースから CRC 値を取得し、取得したデータの生成された CRC と比較して、データの整合性を即時に確保できます。

2.8. 高可用性

CRUSH アルゴリズムで有効化されたスケラビリティに加え、Ceph も高可用性を維持する必要があります。つまり、Ceph クライアントは、クラスターのパフォーマンスが低下した状態にある場合や、モニターに障害が発生した場合でも、データの読み取りと書き込みができる必要があります。

2.8.1. Data Copies

複製されたストレージプールでは、Ceph では、パフォーマンスが低下した状態で動作するためにオブ

ジェットの複数のコピーが必要になります。理想的には、Ceph Storage クラスターは、動作セットの OSD のいずれかが失敗しても、クライアントがデータの読み取りと書き込みを行えるようにします。このため、Ceph はオブジェクトの 3 つのコピーを作成するようにデフォルトで設定されます (書き込み操作に少なくとも 2 つのコピーがクリーンであること)。Ceph は、2 つの OSD が失敗してもデータを保存しますが、書き込み操作が中断されます。

レイジャーコーディングされたプールでは、Ceph はパフォーマンスが低下した状態で操作できるように、複数の OSD にまたがるオブジェクトのチャンクを格納する必要があります。複製されたプールと同様に、理想としては、レイジャーコーディングされたプールにより、Ceph クライアントはパフォーマンスが低下した状態で読み書きできるようになります。このため、Red Hat では、 $M=2$ が設定された 5 つの OSD にチャンクを保存し、2 つの OSD の失敗を許可し、データを復元する機能を維持するように $K+M=5$ を推奨しています。

2.8.2. クラスターの監視

Ceph クライアントがデータの読み取りおよび書き込みが可能になる前に、Ceph Monitor に問い合わせでクラスターマップの最新コピーを取得する必要があります。Ceph Storage クラスターは単一のモニターで操作できますが、単一障害点が生じます。つまり、モニターがダウンしても、Ceph クライアントはデータの読み取りまたは書き込みができなくなります。

信頼性とフォールトトレランスを強化するために、Ceph はモニターのクラスターをサポートしています。モニターのクラスターでは、レイテンシーおよびその他の障害により、1 つ以上のモニターがクラスターの現在の状態のままになる可能性があります。このため、Ceph にはクラスターの状態に関するさまざまなモニターインスタンス間の合意が必要です。Ceph は常にモニターおよび Paxos アルゴリズムを使用して、クラスターの現在の状態に関するモニター間で合意を確立します。モニターホストには、クロックのドリフトを防ぐために NTP が必要です。

管理者は通常、過半数のモニターで Ceph をデプロイし、過半数を判断できるようにします。たとえば、過半数は 1、2:3、3:5、4:6 などになります。

2.8.3. CephX

cephx 認証プロトコルは、Kerberos と同様に動作します。

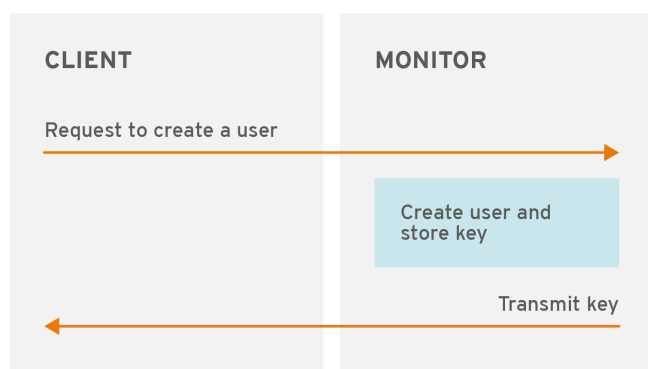
ユーザー/アクターが Ceph クライアントを呼び出してモニターに接続します。Kerberos とは異なり、各モニターはユーザーを認証して鍵を配布できるため、cephx を使用する際に単一障害点やボトルネックはありません。このモニターは、Ceph サービスの取得に使用するセッションキーが含まれる Kerberos チケットと同様の認証データ構造を返します。このセッションキー自体はユーザーの永続的なシークレットキーで暗号化されるため、ユーザーだけが Ceph モニターからサービスを要求することができます。次に、クライアントはセッションキーを使用してモニターから必要なサービスを要求し、モニターはクライアントに実際にデータを処理する OSD に対してクライアントを認証するチケットと共にクライアントを提供します。Ceph モニターおよび OSD はシークレットを共有しているため、クライアントはモニターが提供するチケットをクラスター内の任意の OSD またはメタデータサーバーと共に使用することができます。攻撃者は、Kerberos のように cephx チケットの有効期限が切れるため、攻撃者は取得した期限切れのチケットまたはセッションキーを誤って使用できません。この形式の認証は、通信媒体にアクセスできる攻撃者が、ユーザーの秘密鍵が期限切れになる前に漏えいしない限り、別のユーザーの ID で偽のメッセージを作成したり、別のユーザーの正当なメッセージを変更したりすることを防ぎます。

cephx を使用するには、管理者は最初にユーザーを設定する必要があります。以下の図では、client.admin ユーザーはコマンドラインから `ceph auth get-or-create-key` を呼び出して、ユーザー名およびシークレットキーを生成します。Ceph の auth サブシステムはユーザー名およびキーを生成し、モニターでコピーを保存し、ユーザーのシークレットを client.admin ユーザーに送り返します。つまり、クライアントとモニターが秘密鍵を共有していることを意味します。



注記

client.admin ユーザーは、安全にユーザー ID とシークレットキーを提供する必要があります。



CEPH_378927_0917

第3章 クライアントアーキテクチャー

Ceph クライアントは、データストレージインターフェースの表示方法が大きく異なります。Ceph ブロックデバイスは、物理ストレージドライブと同様にマウントするブロックストレージを表示します。Ceph ゲートウェイは、S3 準拠のオブジェクトストレージサービスと Swift 準拠の RESTful インターフェースを独自のユーザー管理と共に表示します。ただし、すべての Ceph クライアントは Reliable Autonomic Distributed Object Store(RADOS)プロトコルを使用して Ceph ストレージクラスターと対話し、全く同じ基本的なニーズがあります。

- Ceph 設定ファイル、またはクラスター名（通常は `ceph`）およびモニターアドレス
- プール名
- ユーザー名およびシークレットキーへのパス。

Ceph クライアントは、object-watch-notify や striping などのいくつかの同様のパターンに従う傾向があります。以下のセクションでは、RADOS、librados、および Ceph クライアントで使用される一般的なパターンについて、もう少し説明します。

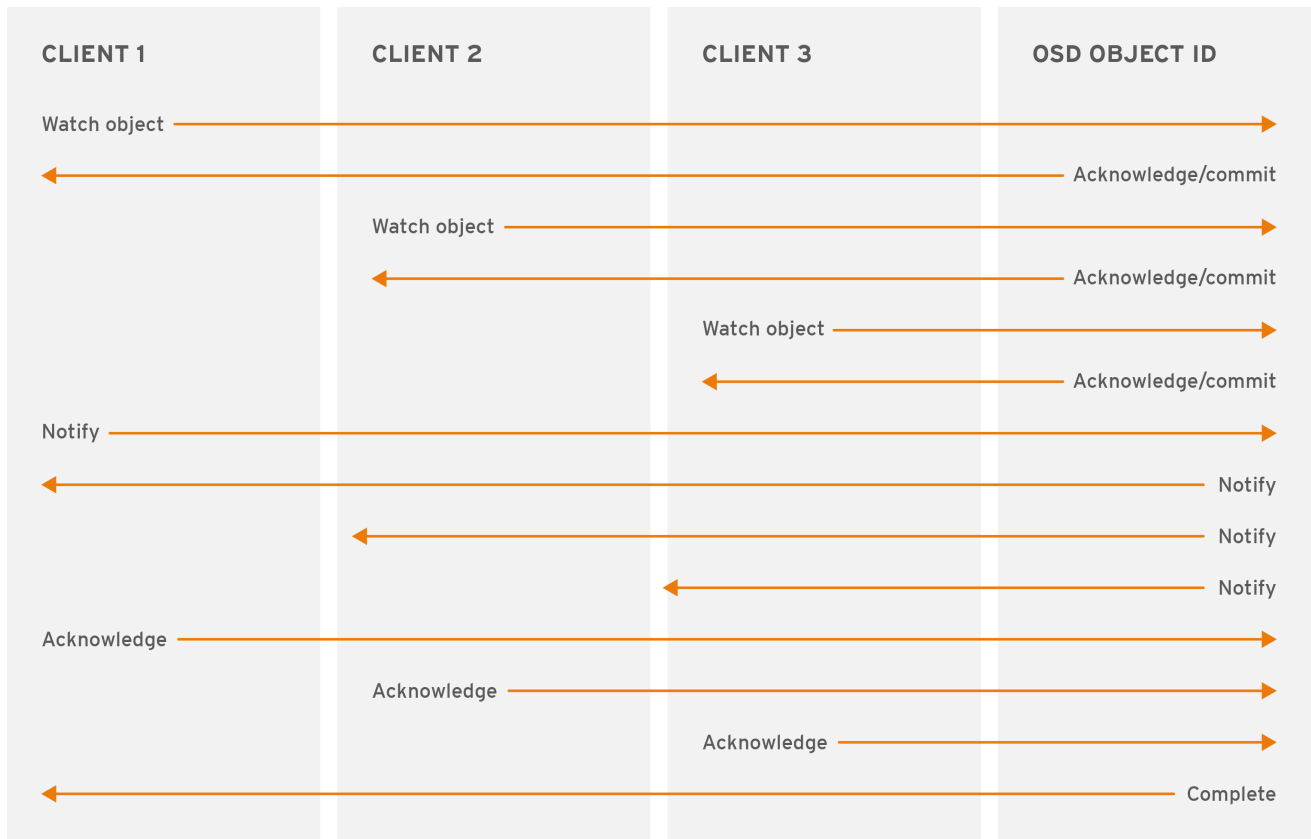
3.1. ネイティブプロトコルおよび LIBRADOS

最新のアプリケーションには、非同期通信機能を備えた単純なオブジェクトストレージインターフェースが必要です。Ceph Storage Cluster は、非同期通信機能を備えたシンプルなオブジェクトストレージインターフェースを提供します。このインターフェースは、クラスター全体でオブジェクトへの直接並行アクセスを提供します。

- プール操作
- スナップショット
- オブジェクトの読み取り/書き込み
 - 作成または削除
 - オブジェクト全体またはバイト範囲
 - 追加または切り捨て
- Create/Set/Get/Remove XATTRs
- Create/Set/Get/Remove Key/Value Pairs
- 複合操作および dual-ack セマンティクス

3.2. オブジェクト監視/非表示

Ceph クライアントは、永続的な関心をオブジェクトに登録し、プライマリ OSD へのセッションを開いたままにすることができます。クライアントは、通知メッセージおよびペイロードをすべてのウォッチャーに送信し、ウォッチャーが通知を受信したときに通知を受信できます。これにより、クライアントは同期/通信チャネルとして、任意のオブジェクトを使用できます。



CEPH_378927_1017

3.3. 必須排他的ロック

必須の排他的ロックは、複数のマウントが設定されている場合に、RBD を単一のクライアントにロックする機能です。これは、マウントされた複数のクライアントが同じオブジェクトに書き込もうとした場合の書き込みの競合状況に対処する上で役立ちます。この機能は、前セクションで説明した **object-watch-notify** で構築されます。したがって、書き込み時に、あるクライアントがオブジェクトに排他ロックを最初に確立すると、マウントされた別のクライアントは、書き込み前にピアがオブジェクトにロックを設定したかどうかを最初に確認します。

この機能を有効にすると、1つのクライアントのみが一度に RBD デバイスを変更することができます。特に、スナップショットの作成/削除などの操作中に内部 RBD 構造を変更する場合などです。また、障害が発生したクライアントをある程度保護します。たとえば、仮想マシンが応答していないように見え、他の場所で同じディスクを使用してそのコピーを開始した場合、最初の仮想マシンは Ceph でブラックリストに登録され、新しい仮想マシンを破損することはできません。

必須の排他的ロックは、デフォルトでは有効になっていません。イメージの作成時に、**--image-features** パラメーターを使用して明示的に有効にする必要があります。以下に例を示します。

```

rbd -p mypool create myimage --size 102400 --image-features 5

```

ここで、数値の 5 は、1 と 4 の合計であり、1 は階層化サポートを有効にし、4 は排他的ロックサポートを有効にします。したがって、上記のコマンドは 100 GB rbd イメージを作成し、階層化および排他的ロックを有効にします。

必須の排他的ロックも **object map** の前提条件となります。排他的なロックサポートを有効にしないと、オブジェクトマップのサポートを有効にすることはできません。

必須の排他的ロックは、ミラーリングのためのいくつかの基本的な作業も行います。

3.4. オブジェクトマップ

オブジェクトマップは、クライアントが rbd イメージに書き込む際にサポートする RADOS オブジェクトの存在を追跡する機能です。書き込みが発生すると、その書き込みはバッキング RADOS オブジェクト内のオフセットに変換されます。オブジェクトマップ機能が有効になっている場合、これらの RADOS オブジェクトの存在が追跡されます。したがって、オブジェクトが実際に存在するかどうかを知ることができます。オブジェクトマップは、librbd クライアントのメモリー内に保持されるため、存在しないことがわかっているオブジェクトを OSD に照会することを回避できます。つまり、オブジェクトマップは実際に存在するオブジェクトのインデックスです。

オブジェクトマップは、以下のような特定の操作に有益です。

- サイズ変更
- エクスポート
- コピー
- フラット化
- 削除
- 読み取り

縮小サイズ変更操作は、末尾のオブジェクトが削除される部分的な削除のようなものです。

エクスポート操作は、RADOS から要求されるオブジェクトを認識します。

コピー操作では、どのオブジェクトが存在し、コピーする必要があるかを認識します。潜在的に数百、数千の可能なオブジェクトを反復する必要はありません。

フラット化操作は、クローンへのすべての親オブジェクトのコピーアップを実行して、クローンを親から切り離すことができるようにします。つまり、子クローンから親スナップショットへの参照を削除できます。したがって、すべての潜在的なオブジェクトの代わりに、コピーアップは存在するオブジェクトに対してのみ行われます。

削除操作は、イメージに存在するオブジェクトのみを削除します。

読み取り操作は、存在しないことがわかっているオブジェクトの読み取りをスキップします。

そのため、サイズ変更（縮小のみ）、エクスポート、コピー、フラット化、および削除などの操作の場合、これらの操作は影響を受けた RADOS オブジェクトすべて（存在するかどうかに関係なく）に対して操作を発行する必要があります。オブジェクトマップを有効にすると、オブジェクトが存在しない場合は、操作を発行する必要はありません。

たとえば、1TB のスパース RBD イメージがある場合は、数百、数千というバッキングする RADOS オブジェクトが含まれる可能性があります。オブジェクトマップを有効にしない削除操作では、イメージの潜在的なオブジェクトごとに `remove object` 操作を実行する必要があります。ただし、オブジェクトマップが有効な場合は、存在するオブジェクトの `remove object` 操作のみを実行する必要があります。

オブジェクトマップは、実際のオブジェクトを持たないが、親からオブジェクトを取得するクローンに対して価値があります。クローンで作成されたイメージがある場合、クローンには最初にオブジェクトがなく、すべての読み取りは親にリダイレクトされます。したがって、オブジェクトマップは、オブジェクトマップがない場合と同様に読み取りを改善できます。最初に、クローンの OSD に対して読み取り操作を発行する必要があります。それが失敗すると、オブジェクトマップを有効にして、親に対して別の読み取りを発行します。存在しないことがわかっているオブジェクトの読み取りをスキップします。

オブジェクトマップは、デフォルトでは有効になっていません。イメージの作成時に、`--image-features` パラメーターを使用して明示的に有効にする必要があります。また、必須の排他的ロック（前のセクションで説明）は、オブジェクトマップの前提条件です。排他的なロックサポートを有効にしないと、オブジェクトマップのサポートを有効にすることはできません。イメージの作成時にオブジェクトマップのサポートを有効にするには、以下を実行します。

```
rbd -p mypool create myimage --size 102400 --image-features 13
```

ここで、数値の 13 は、1 と 4、8 の合計であり、1 は階層化サポートを有効にし、4 は排他的ロックサポートを有効にして、8 は、オブジェクトマップサポートを有効にします。したがって、上記のコマンドは 100 GB rbd イメージを作成し、階層化、排他ロックおよびオブジェクトマップを有効にします。

3.5. データストライピング

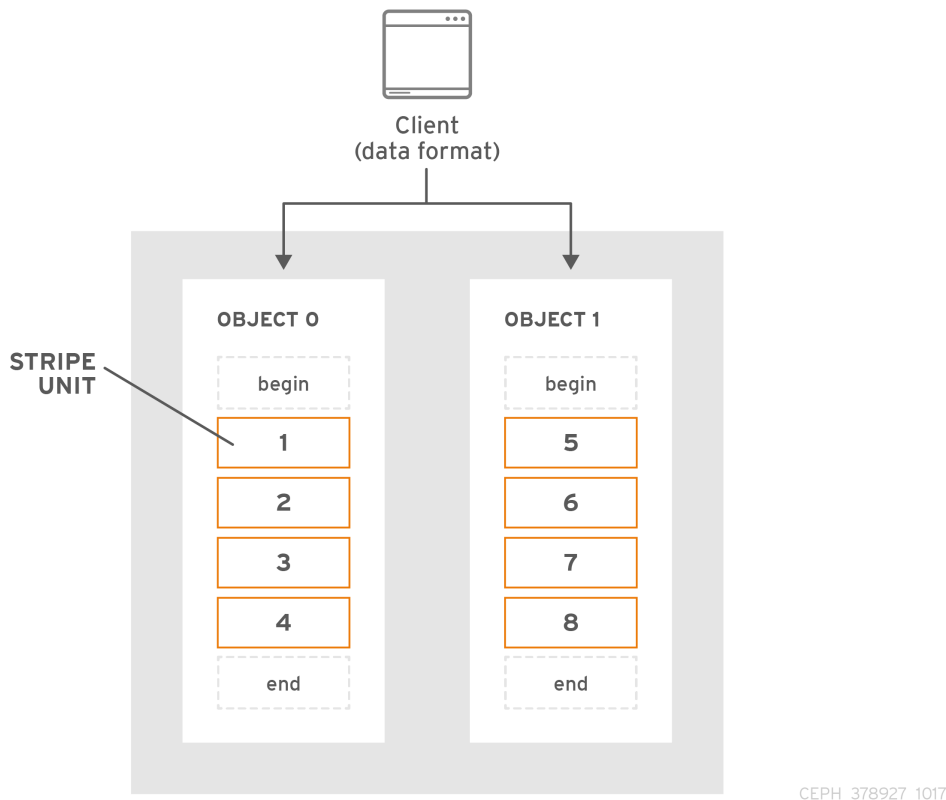
ストレージデバイスにはスループットの制限があり、パフォーマンスとスケーラビリティに影響を及ぼします。したがって、ストレージシステムは多くの場合、複数のストレージデバイス間で連続した情報の連続した情報プライピングに対応し、スループットとパフォーマンスを向上させることができます。データストライピングの最も一般的な形式は RAID から来ています。Ceph のストライピングに最も類似する RAID タイプは、RAID 0、つまり「ストライプ化ボリューム」です。Ceph のストライピングでは、RAID 0 ストライピングのスループット、n-way RAID ミラーリングの信頼性、および迅速なリカバリーを提供します。

Ceph には、Ceph Block Device、Ceph Filesystem、および Ceph Object Storage の 3 種類のクライアントがあります。Ceph クライアントは、提供される表現形式からユーザー（ブロックデバイスイメージ、RESTful オブジェクト、CephFS ファイルシステムディレクトリー）から Ceph Storage クラスターのストレージのオブジェクトに変換します。

ヒント

Ceph Storage Cluster に Ceph が格納するオブジェクトはストライプ化されていません。Ceph Object Storage、Ceph Block Device、および Ceph Filesystem は、複数の Ceph Storage Cluster オブジェクトにデータのストライプ化を行います。librados 経由で Ceph Storage クラスターに直接書き込む Ceph クライアントは、これらの利点を得るためにストライピング（および並列 I/O）を実行する必要があります。

最も単純な Ceph のストライプ化形式には、1 つのオブジェクトのストライプ数が含まれます。Ceph クライアントは、オブジェクトが最大容量になるまで Ceph Storage Cluster オブジェクトにストライプユニットを書き込みます。その後、データの追加のストライプ用に別のオブジェクトを作成します。ストライプ化の最も単純な形式は、小さなブロックデバイスイメージ、S3 または Swift オブジェクトにとって十分と言えます。ただし、この単純な形式では、配置グループ全体にデータを分散する Ceph の機能を最大限に活用しないため、パフォーマンスはそれほど向上しません。以下の図は、ストライプ化の最も単純な形式を示しています。



CEPH_378927_1017

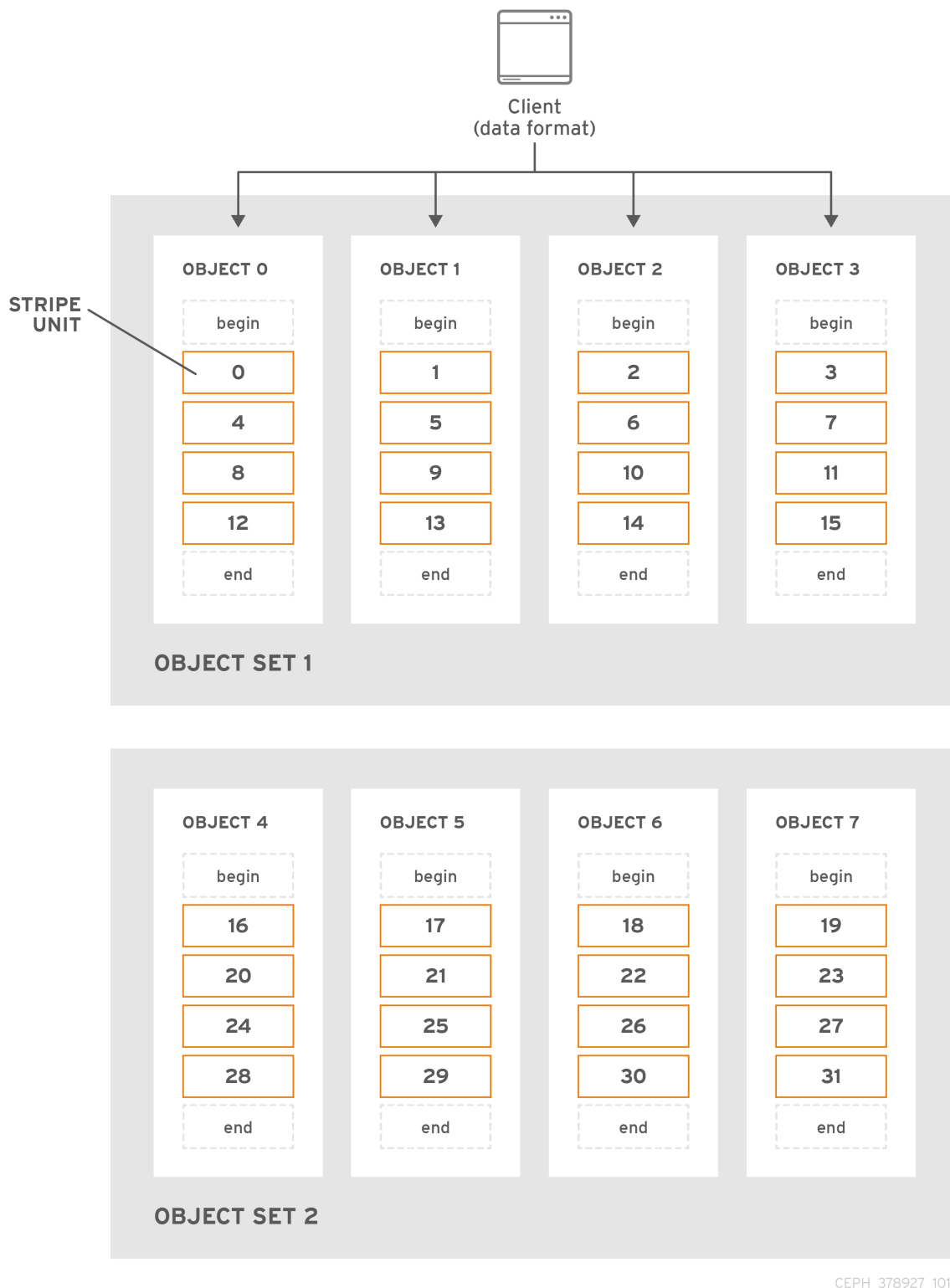
大規模なイメージサイズ、大規模な S3 または Swift オブジェクト（ビデオなど）を予測できる場合には、オブジェクトセット内の複数のオブジェクトにクライアントデータを取り除くことで、パフォーマンスを大幅に改善できる可能性があります。クライアントが対応するオブジェクトに並列にストライプユニットを書き込むと、書き込みパフォーマンスが大幅に向上します。オブジェクトは異なる配置グループにマッピングされ、さらに異なる OSD にマッピングされるため、それぞれの書き込みは最大書き込み速度で並行して行われます。1つのディスクへの書き込みは、ヘッドの移動（シークあたり 6 ミリ秒など）と1つのデバイスの帯域幅（例：100MB/s）によって制限されます。Ceph は、（異なる配置グループおよび OSD にマップする）複数のオブジェクトへの書き込みを分散することで、ドライブごとのシーク数を減らし、複数のドライブのスループットを組み合わせることで書き込み（または読み取り）の速度を大幅に向上させることができます。



注記

ストライプ化は、オブジェクトレプリカとは独立しています。CRUSH は OSD 間でオブジェクトを複製するため、ストライプは自動的に複製されます。

以下の図では、クライアントデータは、4つのオブジェクトで構成されるオブジェクトセット（以下の図の object set 1）でストライプ化されます。最初のストライプユニットは、object 0 の stripe unit 0 です。4番目のストライプユニットは、object 3 の stripe unit 3 です。4番目のストライプを作成したら、クライアントはオブジェクトセットがいっぱいかどうかを判断します。オブジェクトセットが満杯でない場合は、クライアントは最初のオブジェクトにストライプを書き始めます（以下の図のオブジェクト 0）。オブジェクトセットが満杯になると、クライアントは新しいオブジェクトセット（以下の図の object set 2）を作成し、新しいオブジェクトセット（以下の図のオブジェクト 4）の最初のオブジェクトで最初のストライプ（ストライプユニット 16）への書き込みを開始します。



CEPH_378927_1017

3つの重要な変数が、Ceph によるデータのストライプ化方法を決定します。

- **オブジェクトサイズ**：Ceph Storage クラスターのオブジェクトには、設定可能な最大サイズ（2MB、4MB など）があります。オブジェクトのサイズは、多くのストライプユニットを収容するのに十分な大きさであり、ストライプユニットの倍数である必要があります。Red Hat は、安全な最大値 16 MB を推奨します。
- **ストライプの幅**：Stripes には設定可能なユニットサイズ（64kb など）があります。Ceph クライアントは、オブジェクトに書き込むデータを、最後のストライプユニットを除いて、同じサイズのストライプユニットに分割します。ストライプ幅は、オブジェクトに多くのストライプユニットが含まれるように、オブジェクトサイズのごく一部にする必要があります。

- ストライプ数: Ceph クライアントは、ストライプ数で決定される一連のオブジェクトに一連のストライプユニットを書き込みます。一連のオブジェクトは、オブジェクトセットと呼ばれます。Ceph クライアントがオブジェクトセットの最後のオブジェクトに書き込みした後に、オブジェクトセットの最初のオブジェクトに戻ります。



重要

クラスターを実稼働環境に移行する前に、ストライプ化設定のパフォーマンスをテストします。データをストライプ化してオブジェクトに書き込んだ後は、これらのストライプ化パラメーターを変更することはできません。

Ceph クライアントがデータをストライプユニットにストライプ化し、ストライプユニットをオブジェクトにマッピングすると、Ceph の CRUSH アルゴリズムは、オブジェクトをストレージディスクにファイルとして保存する前に、オブジェクトを配置グループにマッピングし、配置グループを CephOSD デーモンにマッピングします。



注記

クライアントは単一のプールに書き込むため、オブジェクトにストライプ化されたすべてのデータは、同じプールの配置グループにマッピングされます。したがって、同じ CRUSH マップと同じアクセス制御を使用します。

第4章 暗号化

LUKS ディスク暗号化およびその利点について

Linux Unified Key Setup-on-disk-format (LUKS) メソッドを使用して、Linux システム上のパーティションを暗号化できます。LUKS は、ブロックデバイス全体を暗号化するため、脱着可能なストレージメディアやノート PC のディスクドライブといった、モバイルデバイスのコンテンツを保護するのに適しています。

ceph-ansible ユーティリティーを使用して、暗号化された OSD ノードを作成して、保存したデータを保護します。詳細は、Red Hat Enterprise Linux の『[Red Hat Ceph Storage 3 インストールガイド](#)』の「[Red Hat Ceph Storage クラスター のインストール](#)」セクションを参照してください。

LUKS の詳細は、Red Hat Enterprise Linux 7 の『[セキュリティガイド](#)』の「[Overview of LUKS](#)」セクションを参照してください。

ceph-ansible で暗号化したパーティションの作成方法

OSD のインストール時に、**ceph-ansible** は暗号化されたパーティションを作成する **ceph-disk** ユーティリティーを呼び出します。

ceph-disk ユーティリティーは、データ (ceph data) パーティションおよびジャーナル (ceph journal) パーティションに加えて、小規模な **ceph lockbox** パーティションを作成します。また、**ceph-disk** は、**cephx client.osd-lockbox** を作成します。**ceph lockbox** パーティションには、**client.osd-lockbox** が、暗号化された **ceph data** および **ceph journal** パーティションを復号化するのに必要な LUKS 秘密鍵を取得するのに使用する鍵ファイルが含まれています。

次に、**ceph-disk** は **cryptsetup** ユーティリティーを呼び出して、**ceph data** パーティションおよび **ceph journal** パーティション用に 2 つの **dm-crypt** デバイスを作成します。**dm-crypt** デバイスは、**ceph data** および **ceph journal** GUID を識別子として使用します。

ceph-ansible による LUKS 鍵の処理方法

ceph-ansible ユーティリティーは、LUKS 秘密鍵を Ceph Monitor の key-value ストアに保存します。各 OSD には、OSD データとジャーナルを含む **dm-crypt** デバイスを復号化する独自のキーがあります。暗号化したパーティションは、システムの起動時に自動的に復号化されます。