Vol.2019-HPC-171 No.1
2019/9/20

分散オブジェクトストレージ Ceph のための Spark ストレージコネクタの設計

高橋 宗史[1,a)]　建部修見[2]

概要：高いスケーラビリティを特徴とする分散オブジェクトストレージ Ceph は，大規模データを蓄積・分析に利用するデータレイクとしての活用が広まっている．大規模化するデータを効率的に処理するためには，大容量ストレージとデータ処理アプリケーションとの高速なデータ転送を実現することが重要である．本研究では，スケーラブルな分散オブジェクトストレージ Ceph に蓄積されたデータを効率的に利用することを目的として，リアルタイムの大規模データ処理基盤として広く使用されている Apache Spark や Apache Hadoop からデータの利用を可能にするストレージコネクタを設計・実装した．

キーワード：分散オブジェクトストレージ，Ceph，Apache Spark，Apache Hadoop，ストレージコネクタ

1. はじめに

1.1 オブジェクトストレージ

科学と産業の様々な分野において，大規模なデータを蓄積して，大規模なデータ処理を行うことによって，価値のあるデータを発見しようとするこ込みが盛んに行わている．科学計算の大規模化・産業でのデータ収集の拡大により，データ処理のために蓄積されるデータの容量は増加の一途をたどっており，スケーラビリティが高く，大容量のデータをより少ないコストで長期間保管することができるオブジェクトストレージの重要性は高まっている．

オブジェクトストレージには，パブリッククラウドプロバイダが提供する Amazon S3 [1] や Google Cloud Storage [2] などのサービスが存在する．パブリッククラウドのストレージのネットワーク性能は改善が続けられており [3]，コストや要求性能が満たされる場合には選択肢として有力である。しかし，特に，インタラクティブなデータ解析を高速に行ったり，セキュリティ上の制限，遠隔地とのデータ転送に伴うレイテンシやバンド幅の制限などの理由により，オンサイトでオブジェクトストレージシステムを保有する要求も大きい．

スーパーコンピューティングの分野では，エクサスケール・コンピューティングへ向けて，ストレージを含む各コンポーネントが発展を続けている．計算ノードの性能向上やスケーラビリティの向上に伴い，しかし，コンピューティングとストレージの性能差は広がり続けており，ストレージシステムに対しては，多数の計算ノードからのアクセス要求に応えられる高い性能が求められている．POSIX 互換ファイルシステムでは，メタデータへのアクセスが性能のボトルネックになることが多い．この問題を解消するために，POSIX の制限を緩和した，スケーラビリティの高いオブジェクトストレージも補完的に活用されている．

1.2 分散オブジェクトストレージ Ceph

Ceph [4] は，エクサスケールレベルの優れたスケーラビリティを持つ，ソフトウェア定義型の分散オブジェクトストレージである．Ceph のストレージクラスタは，コモディティなハードウェアを用いて構築することができ，メタデータサーバーが存在せず，クラスタを構成するコンポーネントに単一障害点がなく，可用性が高いことが特徴である．クラスタのダウンタイムなしでのキャパシティの拡張，ストレージシステムのローリングアップデートなどが可能となっている．2006 年に基本設計が発表されて以来，フリーソフトウェアとして継続的に開発が進められており，現在でも活発に改良が続けられている．

Ceph には，RADOS Block Device (RBD)，CephFS，RADOS Gateway (RGW)，`librados` などのさまざまなインターフェイスがある．

[1] 筑波大学 大学院 システム情報工学研究科 コンピュータサイエンス専攻
Department of Computer Science, Graduate School of Systems and Information Engineering, University of Tsukuba
[2] 筑波大学 計算科学研究センター
Center for Computational Sciences, University of Tsukuba
[a)] shuuji3@hpcs.cs.tsukuba.ac.jp

RBD は，RADOS オブジェクトストアをブロックデバイスとして抽象化するインターフェイスである．用途としては，OpenStack や CloudStack などを用いて構築されたプライベートクラウドにおいて，仮想ディスクイメージの格納先や，オンデマンドの仮想ブロックデバイスを提供するストレージ基盤として利用されている [5].

CephFS は，RADOS オブジェクトストアを POSIX 準拠のファイルシステムとして抽象化するインターフェイスである．しかし，他のインターフェイスと異なり，ファイルの情報を管理するためのメタデータサーバーを必要としている．

RGW は，RADOS オブジェクトストアを REST API によって利用する，S3 互換のオブジェクトとして抽象化するインターフェイスである．パブリッククラウドの S3 互換のオブジェクトストレージサービスのバックエンド [6] や OpenStack などのプライベートクラウドのストレージ基盤としても広く活用されている [5], [7]．また，オーバーヘッドは大きくなるものの，プライベートクラウドのファイルストレージとして，NFS 経由で利用することも可能である [7], [8], [9].

`librados` は，RADOS オブジェクトストアに直接アクセスすることが可能なライブラリであり，RBD や RGW は，`librados` を基盤として構築されている．

その他にも，15 万台もの大規模な仮想マシン群のバックエンドとしての利用や，数 10 GiB/s 規模の大規模な科学実験データを保存するためのストレージのバックエンドとしての利用も検討されているなど，高いパフォーマンスやスケーラビリティが求められる環境で実用されている [10].

特に，大規模なデータを格納するような場合に，HDFS に存在するストレージクラスタ管理の煩雑さの問題，スケーラビリティの制限，MapReduce のみではなく，多様なデータ処理技術が活用されるようになってきたことにより，HDFS から Ceph への移行が進んでいる [11], [12].

さらに，ストレージを高効率に利用できるようにするために，HDFS・Ceph ともに Erasure Coding が実装されており，実用化されている [13], [14].

1.3 Apache Spark

Apache Spark [15], [16], [17] は，大規模分散処理が可能なデータ処理基盤ソフトウェアである．

ワークロードによっては現在でも Apache Hadoop は有用だが，特に，特定のデータに対して反復的な処理が必要なインタラクティブなデータ解析や機械学習などイテレーション処理が重要なワークロードには Apache Spark が適しており，こうしたデータ処理では広く普及している．

Apache Spark では，標準でさまざまなライブラリが提供されており，大規模ストレージに格納されたデータに対して，MLlib による機械学習，Spark Streaming によるストリーム処理，GraphX によるグラフ処理などを行うシステムを構築することができる．

また，Apache Spark はパブリッククラウド上のデータ処理基盤としても採用が広がっており，Google Cloud Platform の Dataproc [18] や，Amazon Web Service の Amazon EMR [19] などは，Apache Spark をマネージドサービスとして提供している．こうしたサービスでは，各クラウドが提供するオブジェクトストレージと Apache Spark との間でオブジェクトデータの I/O を行うためのコネクタが提供されており，広く活用されている．

1.4 既存の手法の問題点と提案手法の利点

現在，Apache Spark から Ceph に格納されたデータを利用する場合には，もっぱら，AWS の S3 オブジェクトストレージ用に開発された S3A コネクタが利用されている．S3A コネクタは，Hadoop FileSystem API の実装の一つであり，Spark から S3 のオブジェクトデータを読み書きすることを可能にする．Ceph のインターフェイスの 1 つである RADOS Gateway を利用することで，S3 オブジェクト互換の形式でデータを Ceph のストレージクラスタに格納することができるため，Ceph を Spark のデータストアとして利用することができるようになる．

しかしながら，RADOS Gateway をこのように用いる場合，`librados` から取得した Ceph ネイティブの RADOS オブジェクトを，一度 S3 オブジェクトの形式に変換した上で，さらに S3A からデータを読み書きする必要があり，データ形式の変換が 2 重に行われることになる．また，以前より，RADOS Gateway 自体が Ceph の I/O 性能のボトルネックの原因となることが知られてきた．

図 1 に，既存の S3A コネクタと本研究で実装した Ceph コネクタにおける，オブジェクトの変換の違いを示す．

そこで，本研究では，Spark から直接 RADOS オブジェクトを読み書きを行うことができる Ceph コネクタの設計・実装を試みた．ボトルネックの原因となっていた Rados Gateway をバイパスすることで，オーバーヘッドが削減され，I/O の性能の向上が期待される．

2. 関連研究

2.1 GCS コネクタ / S3 コネクタ (S3A)

Google Cloud Storage (GCS) コネクタ [20], [21] は, GCS のための実装である．

S3A コネクタは，Hadoop コミュニティによって提供されている Hadoop FileSystem API の実装であり，AWS S3 オブジェクトストレージとデータの読み書きをするためのコネクタである [22], [23].

いずれも Hadoop FileSystem API を実装しているため，オブジェクトストレージに格納したデータを，Apache Spark や Apache Hadoop から透過的に利用することがで

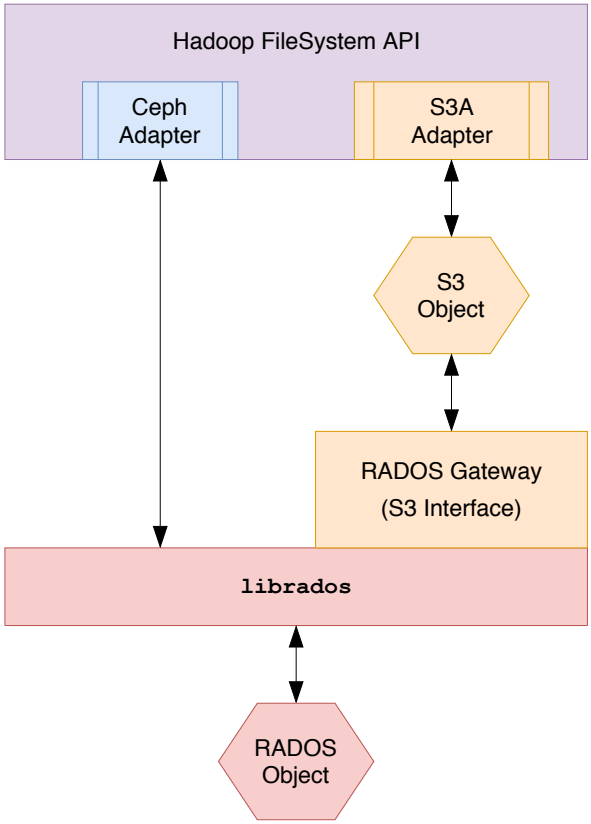

図 1 既存の S3A コネクタと Ceph コネクタにおけるオブジェクトの変換の違い

きる．これらのコネクタは，パブリッククラウドのサービスを利用することを前提としており，プライベートクラウドや，自身で管理されている高性能なストレージクラスタで利用するためには，パブリッククラウドの仕様に合わせた実装を開発する必要がある．

2.2 CephFS-Hadoop プラグイン

Ceph のストレージクラスタのデータに Spark や Hadoop からアクセスする手段としては，Ceph 公式の cephfs-hadoop プラグインが提供されている [24]．これは，CephFS インターフェイスを用いて，データを Hadoop FileSystem API を経由で利用することを可能にするものであり，HDFS を置換することを目的として開発されたプラグインである．

しかし，1.2 節で言及したように，CephFS は Ceph を基盤として POSIX 準拠のファイルシステムを構築することを目的として開発されているインターフェイスである．Ceph のインターフェイスの中では比較的性能が低く，また，完全な POSIX 準拠を必要としない Hadoop FileSystem API から利用するためには不要なインターフェイスが多く実装されているため，さまざまなオーバーヘッドが存在する．

2.3 分散ファイルシステム向けプラグイン

分散ファイルシステムからデータ処理基盤へ接続するための同様のプラグインとして，GlusterFS のための `glusterfs-hadoop` [25]，GPFS のための GPFS-hadoop [26]，Gfarm ?のための Hadoop-Gfarm [27] が開発されているが，これらはどれも特定の分散ファイルシステムに特化したものとなっており，オブジェクトストレージのデータ特性に合わせたオブジェクトデータへの直接アクセスを提供するものとはなっていない．

2.4 その他

Ceph に格納されたデータに Apache Hadoop からアクセスすることを目的に開発されていたプラグインとしては，RGW に対してロードバランシングを行う RGWFS や RGW-Proxy [28] がある．しかし，これらはオーバーヘッドの存在する RGW に対してさらに追加のレイヤーを加えるプラグインであり，また，Apache Spark などのデータ処理基盤からのアクセスには S3A の方がより性能が高いため，RADOS オブジェクトへの高性能なアクセスを提供するという本研究の目的には適さない．

3. 設計と実装

3.1 Spark のストレージインターフェイス

Spark のストレージインターフェイスは Hadoop のストレージインターフェイスと完全な互換性を持っており，Hadoop FileSystem API を実装したすべてのストレージコネクタは，Spark からも透過的に利用することができる．したがって，Ceph のオブジェクトにアクセスすることが可能な Hadoop FileSystem API の実装を行うことにより，Ceph のデータを Spark や Hadoop から直接利用することが可能になる．

3.2 `librados` を用いた RADOS オブジェクトへの直接アクセス

本研究では，Ceph の RADOS オブジェクトに直接アクセスして利用することができる `librados` ライブラリを使用して Ceph コネクタを実装する．`librados` 自体は C++ で書かれているが，Java，Python，Go，Haskell，Rust，Ruby，Node.js，PHP，D など，多数の言語用のバインディングが存在するため，さまざまなプログラムから利用することができる．

また，コネクタを実装するプログラミング言語としては，Scala を選択した．Scala は，Java のライブラリの大部分をそのまま利用することができ，コンパイル時に Java とバイトコードレベルで互換性があるプログラムを生成することができる．そのため，`librados` の Java バインディングである `rados-java` [29] を利用することで，Scala から Ceph クラスタ内の RADOS オブジェクトへアクセスする

ことができるようになる．rados-java バインディングは，JNA (Java Native Access) [30] を用いて実装されており，librados のネイティブコードが実行されるため，JVM で実行することによるオーバーヘッドはほぼ存在しない．さらに，Scala を利用したことで，Java に比べてリーダビリティの高い簡潔なコードで記述することが可能となった．

3.3 コネクタの構成

図 2 に，実装した Ceph コネクタの構成の概略を示す．

本研究では，Hadoop FileSystem API の抽象クラスである hadoop.fs.FileSystem を継承した新たなクラスとして，hadoop.fs.CephFileSystem を実装した．このクラスが Spark などのデータ処理基盤の I/O 処理のエントリーポイントとなり，データの I/O を行う責務を担うことになる．

CephFileSystem の実際の I/O 処理を行うのは，それぞれ CephInputStream クラスと CephOutputDataStream クラスである．

この中で，Java の New I/O ライブラリにより提供される高速な I/O 処理機能 (java.nio.ByteBuffer や java.nio.Channels) を活用することで，Ceph クラスタの RADOS オブジェクトのデータとの I/O 処理を高速化する工夫を行った．

3.4 オブジェクトストレージにおける仮想的なディレクトリの表現

オブジェクトストレージでは，POSIX 準拠ファイルシステムの制限を緩和するために，フラットな名前空間が採用されており，バケット内のすべてのオブジェクトはバケットルート直下に配置されている．POSIX 準拠ファイルシステムでは / の文字がディレクトリ階層を表現するのに使われるが，オブジェクトストレージでは，名前に / の文字を含めたとしても，単なるファイル名の文字の 1 つとして扱われるようになっている．

Hadoop FileSystem API は POSIX に完全に準拠しているわけではないが，POSIX ファイルシステムがベースとなっており，ディレクトリ階層の存在を前提として設計されている．そのため，オブジェクトストレージであっても，ファイルとディレクトリとを区別して処理を行う必要がある．ディレクトリ階層の存在を前提としている Hadoop FileSystem API のメソッドとしては，カレントディレクトリの 1 階層のみのファイル情報の一覧を取得する listStatus()，複数のディレクトリ階層を作成する mkdirs()，ファイルやディレクトリの再帰的な削除や移動を行う必要がある delete() や rename() が存在する．

本研究で実装した Ceph コネクタでは，こうした API に対して適切に処理を行うために，ファイル名の末尾が / であるサイズ 0 のオブジェクトを作成するという方法を採用した．これにより，その場所に仮想的なディレクトリが存在すると見做し，データ処理基盤に対してディレクトリ階層が存在しているものとして振る舞うことができるようにした．

表 1 Ceph クラスタの実験ノードの環境

コンポーネント	説明
CPU	Intel Xeon CPU E5-2630 v4 @ 2.20GHz x2
メモリ	DDR2 FB-DIMM 667 MHz 4GB x8 (32GB)
ネットワーク	10 GbaseT/Full
ストレージ	RevoDrive3 X2 SSD 240GB (60GB x4) x1
OS	CentOS Linux release 7.5.1804 (Core)
Linux	3.10.0-957.21.3.el7.x86_64
Ceph	v14.2.2 nautilus (stable)

4. 実験と手法

4.1 実験環境

実験では，合計 6 ノードを使用して Ceph ストレージクラスタを構成した．表 1 に，クラスタを構成する各ノードの情報を示す．6 ノードのうち，1 ノードは管理用のマスターノード，5 ノードはオブジェクトをストアするストレージノードの役割を担う．

マスターノードには，Ceph クラスタの管理用のデーモン ceph-mon および ceph-mgr をセットアップした．ストレージノードには，ストレージデーモンの ceph-osd を，RevoDrive3 X2 の 4 つのモジュール 1 つにごとに 1 プロセス，すなわち，1 ノードごとに 4 プロセスをセットアップした．RevoDrive3 X2 には 4 つの SSD モジュールが搭載されており，各モジュールごとに独立した I/O が可能であり，モジュールごとにデーモンを割り当てることで高速な並列 I/O 処理が可能となるためである．

Ceph のレプリケーションファクターは，デフォルトの 3 に設定した．また，Ceph v12 (Luminous) 以降のデフォルトの設定では，RADOS オブジェクトストアに格納することができるオブジェクトのサイズが 128 MiB までに制限されている．これは，サイズが大きすぎるオブジェクトが多数存在すると，クラスタ構成の変化によるオブジェクトの再配置や，障害回復時にパフォーマンスの低下を招く場合があるためである．実験では，より大きなサイズのオブジェクトを格納した場合の性能を確かめるために，オブジェクトサイズの上限を 1024 MiB まで増やして実験を行った．

4.2 Ceph コネクタの性能評価

Ceph クラスタと Apache Spark との間の，オブジェクトの Read と Write それぞれに対して，性能評価を行った．Read については，1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024 MiB のサイズのランダムなテキストデータを 1 GiB 分生成し，これらを RADOS オブジェクトとして Ceph ク

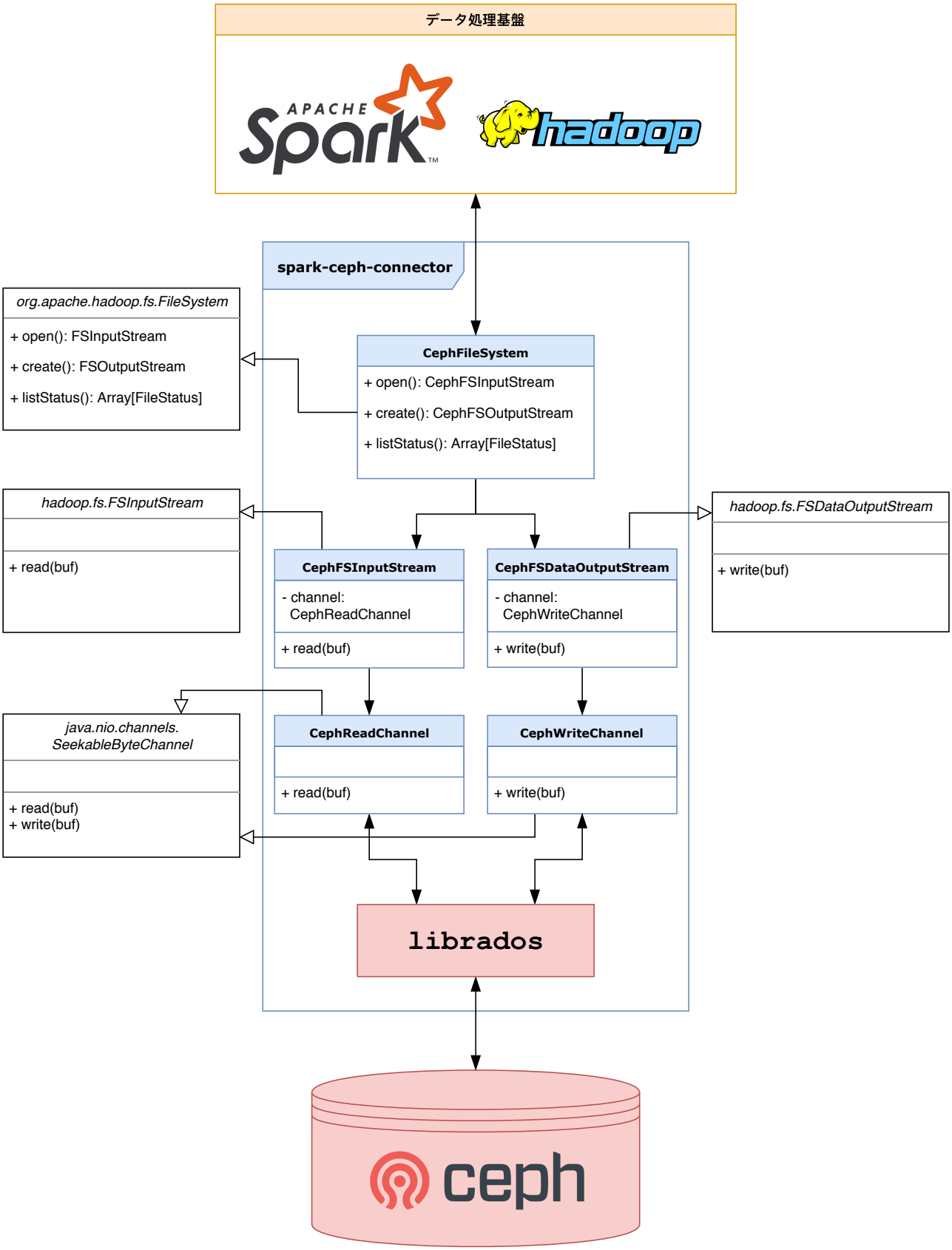

図 2 Ceph コネクタの構成

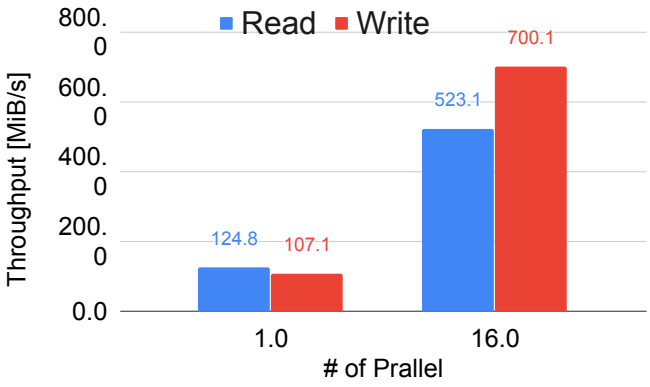

図 3 `rados-bench` の測定結果

ラスタに格納しておき，Apache Spark からの読み込むのスループットを測定した．

Write については，Apache Spark のキャッシュにあらかじめロードしておいた同様のサイズのオブジェクトデータを，それぞれ Ceph クラスタへ書き込むときのスループットを測定した．なお，オブジェクトデータの準備時の Ceph クラスタへのデータの書き込みには，GNU Parallel [31] を用いて並列化を行うことで，高速にデータを格納する工夫を行った．

4.3 性能のベースライン

まずは，Ceph ストレージクラスタの性能のベースラインを確かめるために，Ceph により提供されている `rados-bench` ベンチマークを用いて予備実験を行った．これにより，Ceph クラスタに対して，並列アクセスを含む直接のシーケンシャルアクセスの性能を測定することができる．並列アクセス数として 1 並列および 16 並列，オブジェクトサイズとして基本となる 4 MiB に対して，シーケンシャルの Read / Write 性能を測定した．

`rados-bench` の測定結果を図 3 に示す．

1 並列の Read/Write では，Read が 124.8 MiB/s，Write が 107.1 MiB/s であった．本稿では，1 並列での性能測定を行ったため，この値が性能の上限となると考えられる．

一方，16 並列の Read/Write では，Read が 1 並列のときの約 4 倍の 523.1 MiB/s，Write が約 7 倍の 700.1 MiB/s の高い性能を示した．

5. 結果と考察

5.1 Read 性能

オブジェクトサイズを変化させたときの Read の性能を，図 4 に示す．

オブジェクトサイズが大きくなるにつれて Read 性能も向上し，オブジェクトサイズ 32 MiB あたりから性能が徐々に飽和していることが確認できる．

最も大きなオブジェクトサイズ 1024 MiB のとき，Read 性能は最大となり，112.1 MiB/s であった．節 4.3 で測定

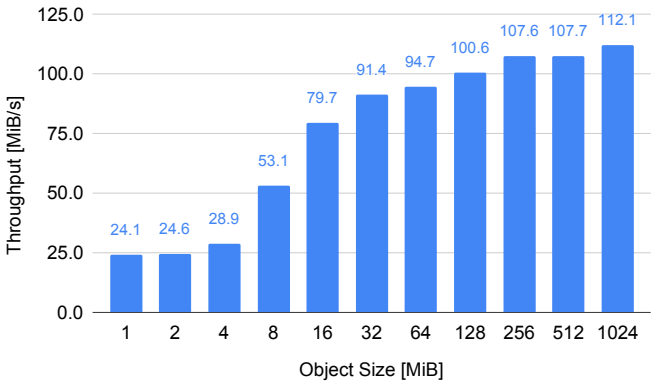

図 4 オブジェクトサイズを変化させたときの Read 性能

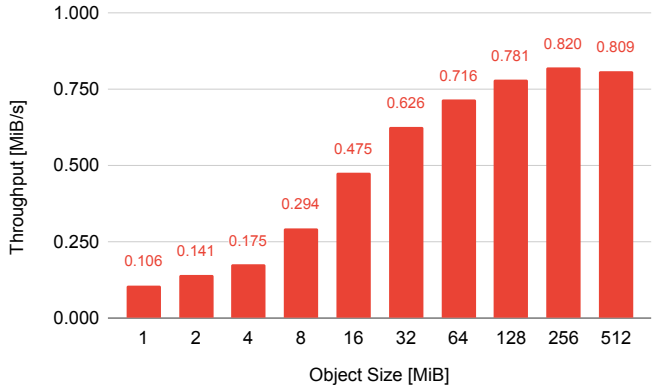

図 5 オブジェクトサイズを変化させたときの Write 性能

した Read のベースライン性能と比較すると，オブジェクトサイズ 1024 MiB で約 90 % の非常に高い性能が発揮された．Hadoop の HDFS ではデフォルトでブロックサイズ 128 MiB が利用されるが，Ceph で同じサイズの 128 MiB のサイズのオブジェクトを Read した場合でも，100.6 MiB の性能が発揮され，ベースライン性能の約 81 %の高い性能を発揮することができた．

5.2 Write 性能

オブジェクトサイズを変化させたときの Write の性能を，図 5 に示す．

オブジェクトサイズが大きくなるにつれて性能が向上し，32 MiB あたりから徐々に性能が飽和する傾向は，Read の場合とよく似ている．

一方，Read 性能がベースライン性能に対して非常に良い性能を発揮していたのに対して，Write の性能は著しく低く，Read の約 1/200 〜 1/100 という非常に低い性能を示している．最も高い 1024 MiB のオブジェクトサイズの場合であっても，1 MiB/s を下回る結果となっている．

5.2.1 性能劣化の原因

性能劣化の原因として考えられる理由の一つは，Apache Spark のデータ保存時のデータの書き込み方法であると考えられる．Apache Spark でのデータの保存時には，耐障

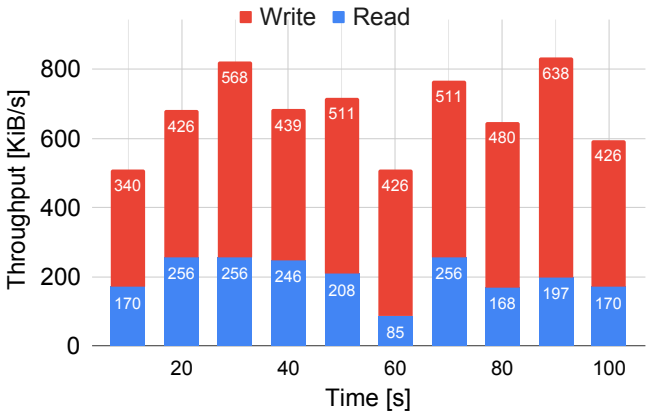

図 6 1 MiB のオブジェクト Write 時の Ceph クラスタにおける I/O の内訳

害性を高めるなどの理由により，次のような 3 ステップでのデータの書き込みが行われる．

(1) ファイルシステムに対して書き込みが成功するかどうかを確認するために，ジョブが割り当てられたタスクが `attempt` という prefix のついたファイルに実際にデータの書き込みを試みる．
(2) データの書き込みが成功したら，書き込まれたファイルを，当該タスクが生成したデータとしてファイルシステムの別の一時ファイルに `rename` する．
(3) すべてのタスクが成功したら，一時ファイルをファイルシステム上の最終的な書き込み場所に移動するために，さらに `rename` を実行する．

このとき問題となるのは，オブジェクトストレージのフラットな名前空間とデータの実体とを結びつけている関係である．Ceph では，オブジェクトデータの格納先を決定するために CRUSH というデータの格納ストレージを決定的に計算可能なアルゴリズムを使用している．CRUSH は，オブジェクト名をキーとして，ファイルの格納先を決定するため，上のような 3 ステップの処理の中で `rename` が実行されるたびに，オブジェクト名が変更されるとともに，Ceph クラスタ上でのオブジェクトの格納先を変更することが必要とされる．そのため，rename が行われるたびにオブジェクトデータの読み込み・コピー・削除の作業が必要になる．本稿では，この問題の解決方法について提示することはできなかった．S3 などのオブジェクトストレージでも同様の問題が存在することが知られている．

5.2.2 Write 時の I/O の内訳

オブジェクトサイズを変化させたときの Write の性能を図 5 に，1 MiB のオブジェクト Write 時の Ceph クラスタにおける I/O の内訳を示す．この値は，Ceph クラスタ内外の I/O の統計情報を取得することを可能にする `iostat` モジュールを利用して取得した．

グラフより，オブジェクトデータの Write 処理の間に，Write の約 1/4 〜 1/3 もの Read 処理が同時に行われていることが確認できる．この Read/Write 処理の中には，単に 1 つのオブジェクトデータの Write 処理だけではなく，前述した 3 ステップの rename による同一オブジェクトの Read / Write が含まれていると考えられる．さらに，Ceph クラスタ内では，中間ファイルを含むすべてのオブジェクトデータに対して，3 つの OSD にレプリケーションする Read / Write 処理が自動的に発生するため，同一のオブジェクトデータに対して，レプリケーションのための Read 処理と Rename 処理が競合することが性能劣化の原因となっている可能性がある．データ書き込み時の I/O をより詳細に解析することが求められる．

6. まとめと今後の課題

本研究では，分散オブジェクトストレージ Ceph に格納された大規模データを，ビッグデータ処理基盤である Apache Spark や Apache Hadoop から高効率で利用できるようにすることを目的として，Hadoop FileSystem API をもとにした Ceph のストレージコネクタの設計と実装を行った．これにより，従来から使用されている，オブジェクトの冗長な変換が必要なコネクタを介さずに，RADOS オブジェクトのデータに直接アクセスすることが可能であることを示した．

今後の課題としては，性能劣化の大きかった Write の性能向上を改善するために，ノードローカルストレージを活用し，書き込みデータを一時的に保存することにより，Ceph クラスタとの I/O の通信を最適化することが考えられる．

また，図 3 で示した `rados-bench` の測定結果により，Ceph ストレージは並列アクセスにより高い性能を発揮することが示された，RADOS オブジェクトへのアクセスを効率化するために，同期アクセスによる 1 並列のアクセスだけではなく，`librados` の非同期アクセス API を活用したり，高い並列度のアクセスを行えるように改良することにより，ストレージコネクタを高性能化することが挙げられる．

さらに，今回の実験では，シングルノードからの I/O 性能のみを測定したが，多数ノードからなる Ceph クラスタおよび Spark クラスタを構成した場合のストレージアクセススケーラビリティを明らかにすることも重要である．

7. 謝辞

本研究の一部は，JSPS 科研費 17H01748，国立研究開発法人新エネルギー・産業技術総合開発機構（NEDO），JST CREST JPMJCR1414 および富士通研究所との共同研究による．

参考文献

[1] Amazon: Amazon S3（拡張性と耐久性を兼ね揃えたクラウドストレージ） | AWS.
[2] Google: Cloud Storage - オンライン データ ストレージ | Cloud Storage | Google Cloud.
[3] 吉田 浩， 合田 憲人， 上田 郁夫， 原 隆宣， 小杉 城治， 森田英輔， 中村 光志：クラウドコールドストレージに対する大規模実験データ格納のケーススタディ，研究報告ハイパフォーマンスコンピューティング（HPC），Vol. 2018-HPC-165, No. 8, pp. 1 – 10（オンライン），DOI: http://id.nii.ac.jp/1001/00190561/ (2018).
[4] Weil, S. A., Brandt, S. A., Miller, E. L., Long, D. D. E. and Maltzahn, C.: Ceph: A Scalable, High-Performance Distributed File System, *Proceedings of the 7th Symposium on Operating Systems Design and Implementation*, USA, USENIX Association, p. 307–320 (online), DOI: 10.5555/1298455.1298485 (2006).
[5] OpenStack Foundation : OpenStack User Survey Analytics and Data , (online), available from ⟨https://www.openstack.org/analytics⟩.
[6] DigitalOcean: Storage on DigitalOcean , (online), available from ⟨https://github.com/digitalocean/navigators-guide/blob/master/book/03-backup/ch07-storage-on-digitalocean.md⟩.
[7] Mascetti, L., Cano, E., Chan, B., Espinal, X., Fiorot, A., Labrador, H. G. a. l., Iven, J., Lamanna, M., Presti, G. L., Mo ś cicki, J. et al.: Disk storage at CERN, *Journal of Physics: Conference Series*, Vol. 664, No. 4, IOP Publishing, p. 042035 (2015).
[8] 高野了成, 谷村勇輔, 竹房あつ子, 広渕崇宏, 田中良夫： 高性能かつスケールアウト可能な HPC クラウド AIST Super Green Cloud , 研究報告ハイパフォーマンスコンピューティング（HPC），Vol. 2014-HPC-145, No. 4, pp. 1–6（オンライン），入手先 ⟨http://id.nii.ac.jp/1001/00102255/⟩ (2014).
[9] 谷村勇輔，浜西貴宏，高野了成，田中良夫： AIST Super Green Cloud におけるストレージシステムの構築と運用，研究報告ハイパフォーマンスコンピューティング（HPC），Vol. 2015-HPC-148, No. 30, pp. 1–6（オンライン），入手先 ⟨http://id.nii.ac.jp/1001/00113235/⟩ (2015).
[10] Lamanna, M.: Large-scale data services for science: Present and future challenges, *Physics of Particles and Nuclei Letters*, Vol. 13, No. 5, pp. 676–680 (2016).
[11] Hat, R.: Why Spark on Ceph? (Part 1 of 3).
[12] Mrowczynski, P.: Scaling cloud-native Apache Spark on Kubernetes for workloads in external storages, PhD Thesis (2018).
[13] James S. Plank : Erasure Codes for Storage Systems: A Brief Primer | USENIX , pp. 44 – 50 (online), available from ⟨https://www.usenix.org/publications/login/december-2013-volume-38-number-6/erasure-codes-storage-systems-brief-primer⟩ (2013).
[14] Arafa, Y., Barai, A., Zheng, M. and Badawy, A.-H. A.: Evaluating the Fault Tolerance Performance of HDFS and Ceph , *Proceedings of the Practice and Experience on Advanced Research Computing - PEARC '18*, New York, New York, USA, ACM Press, pp. 1–3 (online), DOI: 10.1145/3219104.3229269 (2018).
[15] Zaharia, M., Chowdhury, M., Franklin, M. J. and Shenker, S.: Spark: Cluster Computing with Working Sets , (online), available from ⟨https://www.usenix.org/conference/hotcloud-10/spark-cluster-computing-working-sets⟩.
[16] Zaharia, M., Chowdhury, M., Das, T., Dave, A., Ma, J., McCauly, M., Franklin, M. J., Shenker, S. and Stoica, I.: Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing , pp. 15–28 (online), available from ⟨https://www.usenix.org/node/162809⟩ (2012).
[17] The Apache Software Foundation : Apache Spark™ - Unified Analytics Engine for Big Data , (online), available from ⟨https://spark.apache.org/⟩.
[18] Google: Cloud Dataproc - クラウド ネイティブな Apache Hadoop と Apache Spark | Cloud Dataproc | Google Cloud .
[19] Amazon: Amazon EMR での Apache Spark | AWS .
[20] Google: bigdata-interop/gcs at master · GoogleCloudPlatform/bigdata-interop , (online), available from ⟨https://github.com/GoogleCloudPlatform/bigdata-interop/tree/master/gcs⟩.
[21] Google: Cloud Storage コネクタ | Cloud Dataproc ドキュメント | Google Cloud ,（オンライン），入手先 ⟨https://cloud.google.com/dataproc/docs/concepts/connectors/cloud-storage?hl=ja⟩.
[22] The Apache Software Foundation : Apache Hadoop Amazon Web Services support – Hadoop-AWS module: Integration with Amazon Web Services.
[23] The Apache Software Foundation : Hadoop/S3AFileSystem.java at master · sharvinbala/Hadoop.
[24] Ceph : ceph/cephfs-hadoop: cephfs-hadoop, (online), available from ⟨https://github.com/ceph/cephfs-hadoop⟩.
[25] : gluster/glusterfs-hadoop: GlusterFS plugin for Hadoop HCFS.
[26] Raghavendra, R., Dewan, P. and Srivatsa, M.: Unifying HDFS and GPFS: Enabling Analytics on Software-Defined Storage, *Proceedings of the 17th International Middleware Conference*, Middleware '16, New York, NY, USA, ACM, pp. 3:1–3:13 (online), DOI: 10.1145/2988336.2988339 (2016).
[27] 俊輔三上，一樹太田，修見建部：広域分散ファイルシステム Gfarm 上での MapReduce を用いた大規模分散データ処理，研究報告ハイパフォーマンスコンピューティング（HPC），Vol. 2010, No. 4, pp. 1–7（オンライン），入手先 ⟨https://ci.nii.ac.jp/naid/110007995492/⟩ (2010).
[28] : Big Data Analytics on Object Storage.
[29] Ceph: ceph/rados-java , (online), available from ⟨https://github.com/ceph/rados-java⟩.
[30] JNA : java-native-access/jna: Java Native Access, (online), available from ⟨https://github.com/java-native-access/jna⟩.
[31] Tange, O. et al.: Gnu parallel-the command-line power tool, *The USENIX Magazine*, Vol. 36, No. 1, pp. 42–47 (2011).