Vol.2020-HPC-176 No.8
2020/9/25

ストレージコネクタ spark-ceph-connector の書き込み性能の改善

高橋 宗史[1,a)]　建部修見[2]

概要：筆者らは，分散オブジェクトストレージ Ceph を Apache Spark から有効に活用するために，ストレージコネクタ `spark-ceph-connector` を開発した．しかし，読み込み性能と比較すると，十分な書き込み性能が得られていなかった．本研究では，`spark-ceph-connector` の書き込み性能が低い理由を分析し，いくつかの改良を施すとともに，`spark-ceph-connector` 固有の問題に対処できる適切な書き込み手法を適用した．これらの改善により，128 MiB のオブジェクトサイズに対する書き込み性能が，約 0.6 MiB/s から約 20.8 MiB/s に向上し，1,024 MiB のオブジェクトに対しては，最大で約 66.1 MiB/s まで大幅に改善され，実用的な書き込み性能が発揮できるようになった．

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. 背景

このセクションでは，分散オブジェクトストレージ Ceph と，ビッグデータ処理フレームワーク Apache Spark に関する近年の概況を述べる．

続くセクションでは，Apache Spark から Ceph を利用するときの問題と，ストレージコネクタ `spark-ceph-connector` の概要と，解決しようとしている目的を説明する．

1.1 Apache Spark

Google が大規模なデータ処理の基盤システムとして MapReduce [1] を開発・発表すると，その設計をリファレンスにしたオープンソースソフトウェアとして，データ処理フレームワークの Apache Hadoop [2] が開発・公開された．

また，MapReduce のデータ基盤として開発された分散ファイルシステム Google File System [3] をリファレンスとして，Apache Hadoop のデータ基盤となる Hadoop File System (HDFS) [4] もオープンソースソフトウェアとして開発された．Google File System や HDFS は，信頼性の低い多数のコモディティなハードウェアを用いて構築されたクラスタ上で利用することが想定されており，高いスケーラビリティと高い耐障害性という特徴を持っている．

Apache Spark [5] は，オープンソースで開発が行われているインメモリのビッグデータ処理フレームワークである．Apache Hadoop が広い分野に普及するにつれ，特に，頻繁なデータアクセスを必要とする，リアルタイムのデータ解析や機械学習などのワークロードにおいて，Apache Hadoop にはストレージアクセスやメタデータサーバーなどにボトルネックが存在することが明らかになってきた [6][7]．Apache Spark は, Resilient Distributed Datasets (RDD) [8] をベースとしたスマートなインメモリのデータ処理を行うことにより，こうした問題を解消し，高いデータ処理性能を発揮している [9].

Apache Spark は，従来のビッグデータ処理のみならず，広範囲の機械学習にも対応する MLlib [10] などが標準ライブラリとして整備されているため，大規模なデータセットを利用した機械学習のワークロードにも利用されている．幅広い分野での利用が広がっているため，今後もビッグデータ処理フレームワークとして利用され続けることが期待される．

1.2 HPC におけるビッグデータアプリケーションの利用

現在まで，Apache Hadoop や Apache Spark などのビッグデータ処理フレームワークを，HPC の高性能なクラスタにおいて活用しようとする試みが多くなされてきている．

[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

しかし，HPC 環境におけるデータインテンシブなアプリケーションでは，特に，共有ストレージシステムやそのメタデータサーバーなどが主なボトルネックとなることが知られている [11][12][13].

Apache Hadoop によって広く利用されるようになったビッグデータ処理フレームワークは，インメモリの高速な処理が行われる Apache Spark の登場によって，HPC のハードウェアを十分に活かしきれない問題が明らかになってきた．そうした問題を解決する試みとして，RDMA を利用したデータ処理の高速化などが行われている [14].

バイオインフォマティックスなどの科学分野では，大規模な計算のために Apache Hadoop や Apache Spark などのビッグデータ処理フレームワークが広く活用されている．一方で，データサイズの継続的な増加に伴って，コモディティベースのクラスタでは，データ処理に多大な時間がかかってしまうという問題も現れてきている．

これまでも，HPC 環境とビックデータ処理との間のギャップを埋めるために，ビッグデータ処理フレームワークを HPC 環境で活用するなどのさまざまな試みが行われてきており [15][16][17][18][19]，フレームワークとしては，Apache Spark が利用されることも多い [20][21]．今後も，より広い科学コミュニティで利用されるように，高性能なハードウェアをベースとした HPC 環境を有効に活用する方法を探すことは重要であり，そのためには既存の両環境のギャップを埋める取り組みを続けてゆく必要がある．

HPC 環境上で Apache Spark の性能を評価した研究では，特に，共有ファイルシステムにおけるメタデータ性能がボトルネックとなることが明らかにされており [22]，このような側面からも，規模が拡大し続けている HPC 環境におけるストレージシステムのスケーラビリティは，重要な課題となっていると言える．

1.3 コンテナ技術の普及とコンテナオーケストレーションシステム Kubernetes への適応

さらに，近年では，従来のオペレーティングシステムレベルの仮想化に比べてオーバーヘッドが非常に少ないコンテナ技術が活用されるようになってきている．

LXC コンテナや，namespaces や cgroup などの新しい Linux カーネルの機能をベースにして構築された Docker [23] の登場により，コンテナ技術は一般に普及し，さまざまな科学技術分野でも多く利用されるようになっている．さらに，規模の大きなコンピュータクラスタの資源を効率よく利用したり，自動スケーリングにより大規模なクラスタの管理運用負担を軽減できるなどの大きな利点が得られる，さまざまなコンテナオーケストレーションシステムも開発されてきた．パブリッククラウドの環境では，Google が Borg [24] および Omega [25] をベースとして開発し，オープンソースで公開した Kubernetes [26] が多くのプロバイダでデファクトスタンダードとなっている．Slurm などのジョブスケジューラを拡張することで，Kubernetes クラスターを，HPC 環境を含むオンプレミスの共有計算機環境にデプロイする試みも多くなされている [27][28][29].

Ceph や Apache Spark は，特に早い段階で Kubernetes に適応するための開発が活発に行われてきたソフトウェアであり，すでに多くのプロダクション環境での利用実績がある．

Ceph は，最新のバージョン Octopus から，デフォルトですべてのプロセスをコンテナで実行するようになり [30]，Kubernetes のオペレータである Rook [31] を利用することで，Kubernetes 上で自動的にストレージクラスタを構築することが可能になっている [32]．また，その他の分散ストレージとしては，HPC 環境で人気のある Lustre でも，Kubernetes 上でストレージクラスタを自動的に構築する試みが行われており [33]，Ceph と同様に Rook が利用される可能性がある．

Kubernetes ネイティブのバッチジョブスケジューラとして開発が行われているフレームワークとしては，Volcano [34] や，Volcano をベースに構築された Kubeflow [35] があり，これらを利用することで，多人数でクラスタを共有して，長時間のジョブをフェアに実行できる環境が構築できる．Apache Spark は，`spark-operator` [36] を利用することで，Apache Spark application の実行環境を自動的に構築することが可能になっており，ジョブスケジューラフレームワーク上で大規模な計算ジョブを実行することができる．

また，Docker デーモンなどのコンテナランタイムが特権モードで動作することは，HPC 環境のみの問題ではなく，コンテナ技術一般においてもセキュリティ上の問題点であると認識されており，ルート権限を必要とせず，ユーザー権限のみでコンテナのライフサイクルをカバーできるソフトウェアが活発に開発されている．ユーザー権限のみでのコンテナのビルドを行えるオープンソースのソフトウェアとしては，Google を中心に開発されている Kaniko [37] や，Red Hat を中心に開発されている Buildah [38] があり，ユーザー権限のみで実行できるコンテナランタイムとしては，Docker の rootless mode [39][40] や，Red Hat による Podman [41] が開発されている．さらにオペレーティングシステムレベルで環境の安全性を高めるためには，コンテナ向けのセキュアで軽量なサンドボックス環境を提供する Kata Containers[42] や gVisor [43][44] なども活用できる．

HPC 環境においても，ABCI，TSUBAME3.0 などのスーパーコンピューターでは，HPC 向けに開発されたユーザー権限で実行できる Singularity コンテナや，一部の Docker コンテナを実行できるようになってきている [45][46]．現在では，特に，root 特権を要求するストレージシステムに

関しては，HPC 環境におけるコンテナの利用にはまだ課題が残っているものの，Apache Spark などのユーザー空間で動作するソフトウェアに関しては，ユーザー権限で実行できるセキュアなコンテナ技術の普及により，より多くの共有計算機環境で，さまざまな科学技術計算のためにコンテナが活用されるようになってゆくことが期待される．spark-ceph-connector は，こうしたコンテナ環境での利用も念頭に置いて開発されている．

2. spark-ceph-connector

新しい環境に適応し，幅広い分野で利用が広がっている Apache Spark と Ceph は，今度も，ビッグデータ処理フレームワークや，大規模なデータを格納することができるスケーラブルな分散ストレージシステムとして，さまざまな領域で利用されることが期待される．

これまでに述べたように，Apache Spark では，HPC 環境では共有ファイルシステムにおけるメタデータ性能がボトルネックになることが明らかにされている．また，ビッグデータ処理フレームワークでは，必要なデータ量が増大し続けており，大規模なデータを格納することができ，ータ量によらずにスケーラブルな性能が発揮できるストレージシステムが求められている．こうした問題に対して，メタデータサーバーが不要である EB までのスケーラビリティを持つ分散オブジェクトストレージ Ceph を利用することが解決策となる可能性がある．そのためには，Apache Spark から効率よく Ceph のデータを効率的に利用できるストレージコネクタが必要とされている．

そこで，筆者らは，こうした問題を解決することを目的として，ストレージコネクタ spark-ceph-connector を開発した [47]．spark-ceph-connector は，Apache Spark で利用されている他のストレージコネクタと同様に，Apache Spark と互換性のある HDFS の Filesystem API を利用することによって実装を行っている．実装にあたっては，最も高い性能が発揮できるように，Ceph の RADOS Gateway，RBD，CephFS などのインターフェイスの実装基盤としても使われている，librados を選択し，JNA を利用した Java バインディングのライブラリを Scala 経由で利用した．

しかし，spark-ceph-connector はデータの読み込みに対しては高い性能を発揮できた一方で，データの書き込み性能は非常に低い性能しか発揮することができなかった．本研究では，5 節に示すさまざまな観点から，spark-ceph-connector の書き込み性能が低い理由を分析する．それによって，いくつかの改良を施すとともに，spark-ceph-connector 固有の問題に対処できる適切な書き込み手法を適用することで，書き込み性能を改善する．

なお，spark-ceph-connector は，Spark と同じ Apache License 2.0 ライセンスのもと，GitHub 上でオープンソースで公開している [48]．

3. 関連研究

3.1 cephfs-hadoop

Ceph で利用できるその他のストレージコネクタとしては，Ceph の追加レイヤーである CephFS に対して，spark-ceph-connector と同様に HDFS の Filesystem API を実装したストレージコネクタ cephfs-hadoop が存在する [49]．

しかし，このストレージコネクタの実装は非常に古く，2012 年にリリースされた Hadoop バージョン 1.1 系列を必要とするため，現在一般に使われている Hadoop バージョン 2 やバージョン 3 系列では，まったく利用することができない．

さらに，このストレージコネクタは，Ceph のインターフェイスの中でも最もオーバーヘッドが大きい，POSIX 準拠のインターフェイスである CephFS の上に構築されている．そのため，Ceph ネイティブの librados の上に構築された spark-ceph-connector と比較すると，性能面での優位性も低いと言える．

3.2 zero-rename committer

zero-rename committer [50] は，Amazon S3 で利用できる S3A Connector のための Committer の改良に関する研究である．S3A コネクタと密接に関係する実装であるため，spark-ceph-connector から修正せずに直接利用できるかどうかは明らかではない．

本研究の spark-ceph-connector の利用では，zero rename を実現することまではできなかったが，この改良の中心である rename の回数を減らすという同様の方針に基づき，現在のストレージコネクタでも直接利用できる fileOutputCommitter の改善されたアルゴリズムを利用した．この研究により，今度，zero-rename committer と同等の Committer を利用できるようにすることで，さらなる改善が見込めることがわかる．

その他にも，Amazon や Amazon S3 向けの Committer の改善を行っているが [51]，AWS のプラットフォーム上でしか利用できないプロプライエタリなソフトウェアであるため，spark-ceph-connector やオンプレミスの Ceph クラスタのためには利用することはできない．

3.3 Stocator

Stocator [52] は，zero-rename committer と同様の改善を独自のアプローチで実装したストレージコネクタである．

しかし，Amazon S3 や Swift での利用は考慮されているが，Ceph 向けには開発されておらず，librados ライブラリを用いたデータアクセスを行うためには，Stocator 用のストレージプラグインを追加で開発する必要がある．

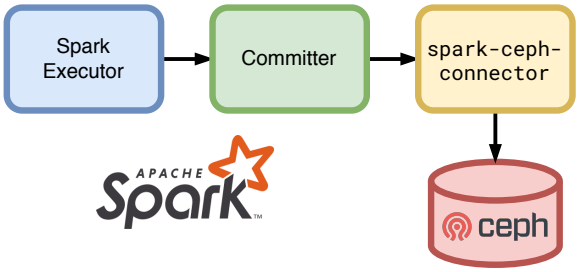

図 1 Apache Spark 上のデータの書き込みに関わる 3 つのコンポーネント

spark-ceph-connector は，追加のライブラリを必要とせず，Apache Spark のプラグインシステムを有効活用し，コードベースに修正を加えずにそのまま利用することが可能である．

4. 準備

このセクションでは，Apache Spark と，筆者らが実装した spark-ceph-connector の実装のうち，本研究が対象とする書き込みに関するコンポーネントの動作と，それらコンポーネント間の関係について簡単に説明する．

4.1 Apache Spark 上のデータの書き込みに関わる 3 つのコンポーネント

Apache Spark では，図 1 に示すように，Spark Executor，Committer，ストレージコネクターという 3 つのコンポーネントがデータのやりとりに関係する．

Spark Executor は，Spark Driver でデータの処理方法が決定された後，ストレージに書き込むべきデータのチャンクを生成する．Committer は，ストレージ上のデータの書き込み場所と，書き込みのステップを規定する．ここで，HDFS インターフェイスを利用したストレージコネクタでデータを書き込む際には，FileOutputCommitter という Committer が利用される．ストレージコネクタは，Committer が指定した書き込み場所とデータのチャンクを受け取り，実際のデータの書き込みを実行する．

spark-ceph-connector では，このとき，データチャンクの 1 つごとにコネクタ内部のバッファでデータを受け取った後，librados を用いて Ceph クラスタと通信を行い，実際のデータの書き込みを実施する．書き込まれたデータは，RADOS オブジェクトを構成し，Ceph クラスタ上に分散して格納される．

4.2 rename によるオブジェクトの移動

ファイルの場所をメタデータで管理する一般的なファイルシステムにとっては，ファイルの rename は，ファイルの path を保存するメタデータの変更を行うだけの非常にコストの小さい処理である．しかし，Ceph や Amazon S3 などのオブジェクトストレージでファイルの rename を実行する場合には，まず，変更前のオブジェクトを変更後の場所に複製し，次に，変更前のオブジェクトを削除する，というコストの高い処理が必要になる．これは，POSIX のファイルシステムと NFS の間でのファイルの移動と似ている．このように， rename に大きなコストが必要になるのは，POSIX に準拠しないオブジェクトストレージ一般に当てはまる特徴である．

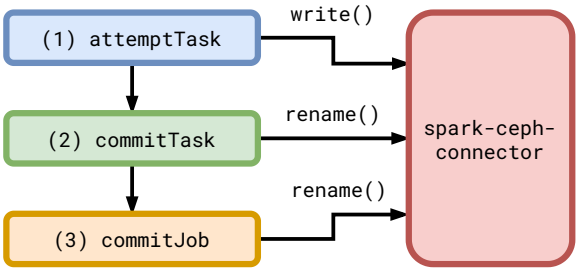

図 2 Committer の動作

Amazon S3 は，プロプライエタリなソフトウェアで構築されたサービスであるため，実際の実装を知ることはできず，ファイル移動が必要な理由はわからなかった．

一方，Ceph の基盤である RADOS オブジェクトストアは，CRUSH というアルゴリズムをベースとして実装されている．Ceph では，CRUSH アルゴリズムを利用することで，クラスタの構成情報およびオブジェクトの名前をもとにして，オブジェクトを保存する場所をアルゴリズムの計算で一意に求める．そのため，オブジェクトの名前の変更を行うと，オブジェクトの保存場所の再計算必要になり，結果として，そのオブジェクトの Ceph クラスタ上のデータの格納場所も変化し，データの移動を避けることができない．

Ceph では，クライアント上でオブジェクトの場所を求めることができるおかげで，ストレージクラスタ上のメタデータサーバーを排除できるようになり，メタデータサーバーが性能のボトルネックとなる HDFS などとは異なり，高いスケーラビリティが実現している．一方で，この特性は，今回のように rename を実行する場合には不利に働き，望ましくないデータ移動が多く発生してしまう．さらに，CRUSH では，データが Ceph クラスタ上で確率的に十分に分散されるように設計されているため，離れたノード上の OSD にデータを移動しなければならない可能性も高くなってしまう．

4.3 Committer の動作

4.1 節で説明したように，ストレージコネクタは，Committer で定義された動作に従ってデータの書き込みを行う．Apache Spark では，デフォルトでは，Hadoop でネイティブに利用される HDFS 向けに実装された fileOutputCommitter が使用され，Version 1 というアルゴリズムが利用される．

このアルゴリズムに基づいてファイルの書き込みが行われるとき，Committer は，図 2 に示すように，次のような 3 つのステップでデータの書き込みを計画し，ストレージコネクターに書き込み指示を行う．

(1) attemptTask: データを`$outputPath/_temporary/$appAttemptID/_temporary/$taskAttemptID/$dataPart` に書き込む

(2) commitTask: ファイルを`$outputPath/_temporary/$appAttemptID/$taskID/$dataPart` に rename する

(3) commitJob: ファイルを`$outputPath/$dataPart` に rename する

しかし，このような複数のステップを踏んだデータの書き込み処理は，Hadoop がネイティブで使用する分散ファイルシステム HDFS をバックエンドとしていることを前提としている．HDFS では，独立したメタデータサーバーが存在するため，rename の処理はそのメタデータサーバーに rename 操作をリクエストし，コストの低い処理を実行してもらえば完了する．一方，Ceph や Amazon S3 のようなオブジェクトストレージでは，複数回の rename が発生することがパフォーマンス上の大きな問題となる．

これは，4.2 節で説明したように，Ceph などのオブジェクトストレージでは，rename が発行されるたびに実際にオブジェクトを移動することが避けられず，それに伴い，ストレージコネクタで不必要な書き込みが発生し，全体としての書き込み性能が劣化してしまうためである．

5. 問題の分析と改善のための提案手法

5.1 複数回の rename の発生による性能劣化

実際の書き込み時のメソッドコールをトレースすることで，1 回の書き込みの他に，rename() が 2 回行われていることを確認した．

この問題の影響を軽減するために, Version 2 のアルゴリズムを使用した `fileOutputCommitter` を利用する [53][54]. このアルゴリズムは，もともとは Hadoop v2 へのアップグレード時のアーキテクチャの変更により，Job から多数のファイルが Commit されたときに，Hadoop v1 と比較して性能の劣化が生じるのを回避する目的で実装されものであるが，オブジェクトストレージにおける rename の問題を軽減するために利用することができる．

7.1 節で行う実験 1 では，このアルゴリズムの変更による影響を確認する．

5.2 Ceph のレプリケーションによる影響

Ceph では，デフォルトでレプリケーションファクターが 3 に設定されており，RADOS オブジェクトストアに格納されたオブジェクトデータは，格納直後に自動的にクラスタ上に分散して複製されるように設計されている．そのため，Commit のプロセスの中でオブジェクトストレージにとっては，一時的にしか配置されないデータに対しても，複製が発生してしまう．これにより，オブジェクトの読み込みと書き込みが不必要に起こり，書き込み性能に影響があることがわかっている [47]． 7.1 節で行う実験 1 では，`fileOutputCommitter` のアルゴリズムの変更に加えて，このレプリケーションによる影響の大きさを確認し，両者の性能を比較する．

5.3 Spark Executor が生成するデータチャンクのサイズ

書き込み性能が非常に低い原因を分析するために，ストレージコネクタのコード内にデバックコードを追加し，データ転送時に実際にストレージコネクタが Committer と Ceph との間で転送しているデータのサイズを確認した．

その結果，Committer から送られてくるデータチャンクの大部分が，約 1 KiB のサイズと 1 Byte のサイズのものに限られており，サイズが非常に小さいことが明らかになった．

そこで， 7.2 節で行う実験 2 では，チャンクサイズの改善を行うために，Java ネイティブのオブジェクトのシリアライザを利用する `saveAsObjectFiles()` による書き込みを試みる．

さらに，`saveAsObjectFiles()` によるチャンクサイズの増加によって不足するストレージコネクタのバッファを拡大し，バッファが性能に与える影響を確認する．

5.4 Spark の DataFrame と効率的なシリアライザの利用

Apache Spark では，データの分散コレクションを内部で保持するためのコアとなるデータ構造として，`RDD` が利用されている．さらに，Apache Spark バージョン 2.0 以降では，`RDD` をベースに拡張された `DataFrame` を利用できるようになった．`DataFrame` により，データセットを構造化し，大規模なデータセットをより容易に効率よく処理できるようになる．

また，`DataFrame` を利用すると，高度なデータのシリアライズが容易に実行できるようになる．そこで， 7.3 節で行う実験 3 では，text，JSON，OCR，Apache Parquet の 4 種類のフォーマットでシリアライズすることで，ストレージコネクタのバッファを効率的に利用することを試みる．

6. 実験環境と予備実験

6.1 実験環境

実験では，SSD を搭載した 5 ノードからなる Ceph クラスタを構成し，その上で実験を行った．表 1 に，クラスタを構成するノードの情報を示す．1 ノードは，管理用の `ceph-mon` コンテナを実行するノードとして使用し，4 ノードは，それぞれ 4 つのストレージ I/O 用の `ceph-osd` コンテナを実行することで Ceph クラスタを構成した．

表 1　Ceph クラスタの実験ノードの環境

Component	Description
Operating System	Ubuntu 20.04 LTS
Kernel	Linux 5.4.0-42-generic (x86-64)
CPU	Intel Xeon CPU E5620 @ 2.40GHz x2
Memory	DDR3 ECC PC3-10600 4 GB x6 (24 GB)
Network	10 Gb Ethernet
SSD	RevoDrive 3 X2 PCIe SSD 240 GB (60GB x4)
Ceph	ceph version 15.2.3 (d289bbd) octopus (stable)

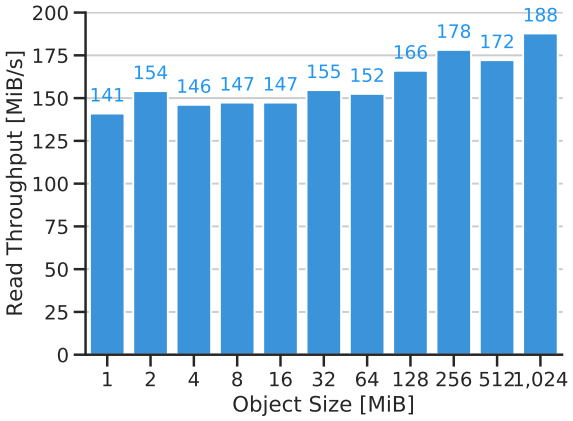

図 3　rados-bench を用いて測定した読み込みのベース性能

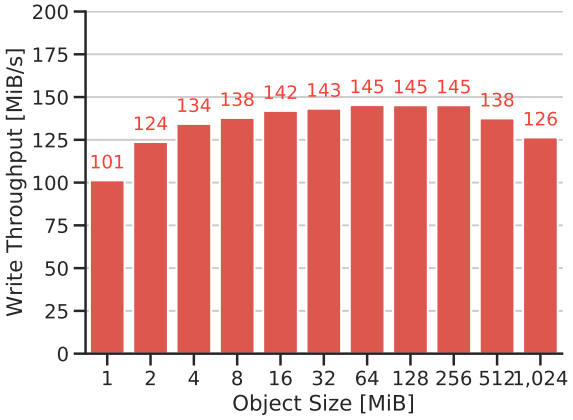

図 4　rados-bench を用いて測定した書き込みのベース性能

6.2　予備実験: rados-bench を用いたベース性能の測定

最新バージョンの Octopus を使用して構築した Ceph クラスタにおけるベース性能を，Ceph に付属する rados-bench を用いて，オブジェクトサイズを変化させて測定した．読み込みのベース性能の測定結果を図 3 に，読み込みのベース性能の測定結果を図 4 にそれぞれ示す．

実験で使用した Ceph クラスタでは，読み込み・書き込みともに，100 MiB/s を超える良好な性能が発揮されていることがわかる．本研究では，spark-ceph-connector の書き込み性能を，図 4 に示されている，書き込みの最大実効性能に近づけることを目指す．

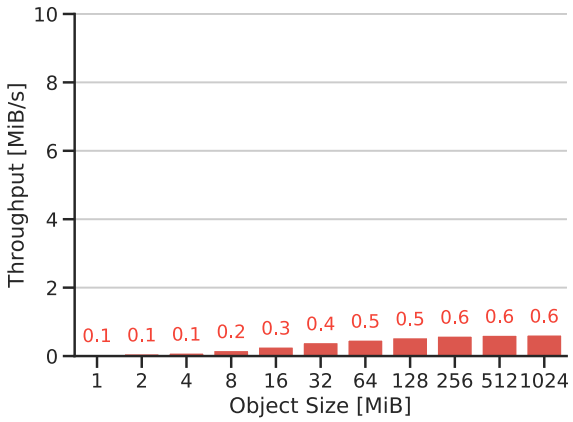

図 5　Committer Version 1 / レプリケーションファクター 3

7.　実験と評価

7.1　実験 1: レプリケーションファクターと Committer のアルゴリズムの変更

実験

実験 1 では，レプリケーションファクターと Committer のアルゴリズムの変更による影響を見るために，レプリケーションファクターが 3 の場合と 1 の場合の 2 通り，fileOutputCommitter アルゴリズムが Version 1 の場合と Version 2 の場合の 2 通り，合わせて 4 通りのパラメータを設定して実験を行う．

結果

4 通りの実験結果を，図 5，図 6，図 7，図 8 に示す．図 5 は，2 つのパラメータを変更する前のもので，図 6 ではレプリケーションファクターを 1 に，図 7 では Committer を Version 2 に，それぞれ変更したもので，図 8 は，両パラメータを変更して改善を加えたものである．

図 5 と 図 6 を比較すると，レプリケーションファクターの変更により，全体的に約 30% の性能が向上したことがわかる．図 5 と 図 7 を比較すると，fileOutputCommitter の Version の変更により，大きく性能が向上し，全体的に約 4 倍の性能向上が見られた．

評価

レプリケーションファクターと fileOutputCommitter のアルゴリズムの変更は，ともにストレージコネクタの性能を向上させることが確認できた．特に，両者のうち，Committer の動作の変更が性能向上に大きく寄与することがわかった．このことから，spark-ceph-connector では，複数回の rename が発生することによる性能劣化の影響が大きかったことが確認できた．

一方，これまで，データの書き込みには，RDD の saveAsText() メソッドを使用していた．RDD は，Apache Spark のベースとなるデータ構造で，テキストデータの保存には saveAsText() を使用するのが基本的な方法である

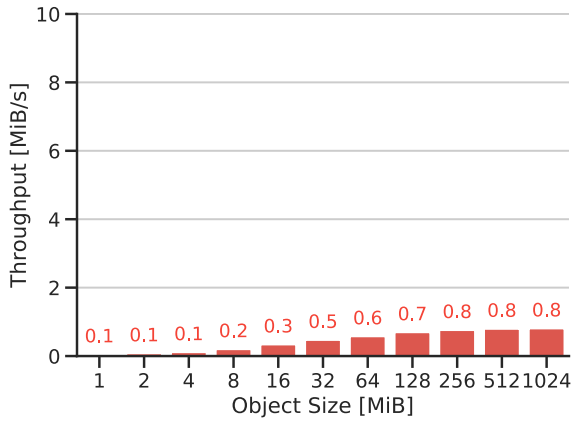

図 6 Committer Version 1 / レプリケーションファクター 1

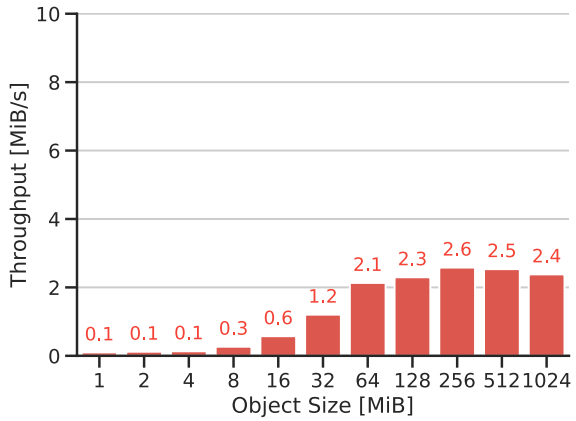

図 7 Committer Version 2 / レプリケーションファクター 3

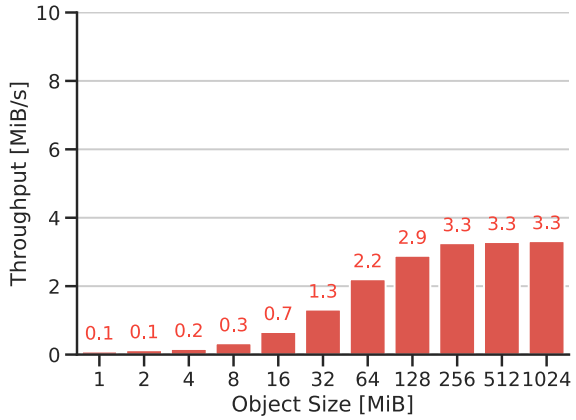

図 8 Committer Version 2 / レプリケーションファクター 1

ためである．

しかし，5.3 節で述べたように，実験 1 の書き込み中のデータを分析すると，実際にストレージコネクターに送られる書き込みデータのチャンクサイズが，1 KiB と非常に小さいことが明らかになった．小さなサイズのデータチャンクでは，ストレージコネクタのバッファの一部しか利用できないため，データチャンクのサイズを増加させる必要がある．

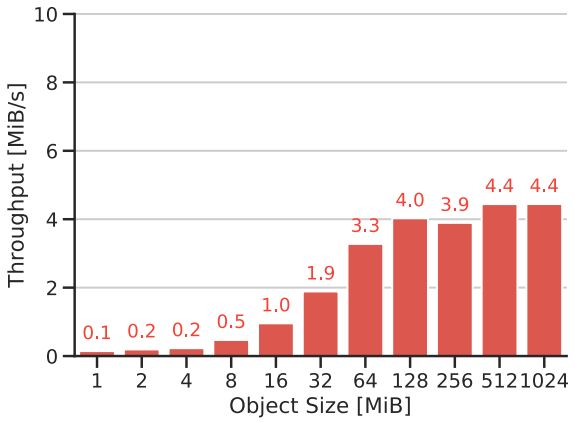

図 9 `saveAsObjectFiles()` / バッファサイズ 4 KiB

また，4 つの実験すべてにおいて，Throughput はオブジェクトサイズ 128 MiB あたりで飽和している．これは，1 度に送られるチャンクサイズが 1 KiB しかないため，Ceph クラスタへの書き込みの待ち時間などのオーバーヘッドが支配的となってしまったためだと考えられる．

7.2 実験 2: ストレージコネクタのバッファサイズ変更

実験

実験 1 の結果を踏まえ，実験 2 では，Version 2 の `fileOutputCommitter` を使用し，Ceph のレプリケーションファクターは 1 に設定して実験を行う．

実験 2 では，RDD の `saveAsObjectFiles()` メソッドの使用を試みた．これにより，初めに Java ネイティブのオブジェクトのシリアライザを利用してデータをシリアライズした後に，データを書き込む処理が行われるようになり，データのチャンクサイズが大きくなることが期待される．

また，チャンクが大きくなった場合に合わせて，ストレージコネクタのバッファを増加させる．

結果

実験 1 と同じバッファサイズで，`saveAsObjectFiles()` を使用して行った書き込みの測定結果を図 9 に，そして，`spark-ceph-connector` のバッファサイズを 4 MiB に増加させて，同様に `saveAsObjectFiles()` を使用した書き込みの測定結果を図 10 に，それぞれ示す．

まず，図 9 を見ると，オブジェクトサイズ 128 MiB 以上では 4.0 MiB/s 以上の Throughput が出ており，図 8 の結果と比較して，同じバッファサイズを使用していても，約 x1.3 - x1.5 の性能向上が見られた．

次に，`spark-ceph-connector` のバッファサイズを 4 MiB に増加させると，さらに性能が改善され，最大 5.9 MiB/s の Throughput が出ており，図 8 の結果と比較して，約 x1.6 - x1.7 の性能向上が見られた．

評価

実験 2 では，書き込みに使用した `saveAsTextFiles()`

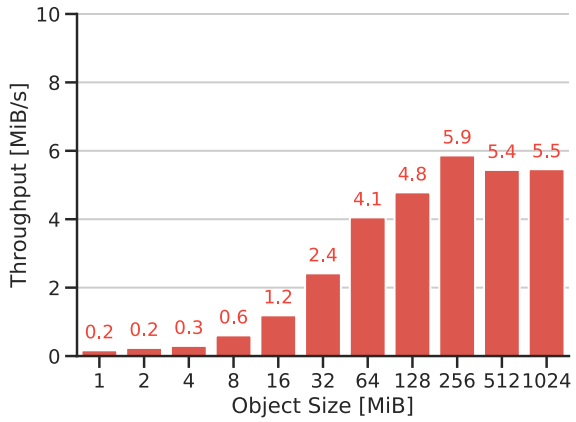

図 10 `saveAsObjectFiles()` / バッファサイズ 4 MiB

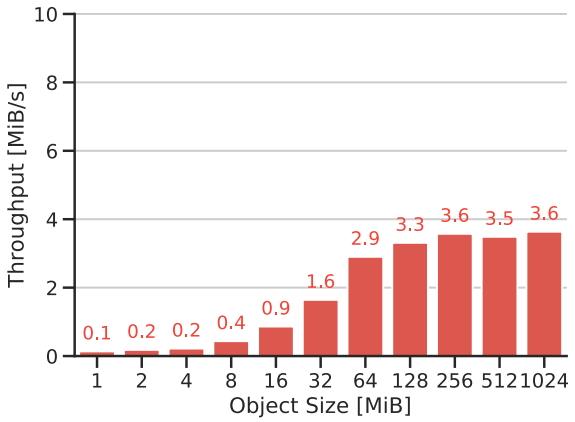

図 11 DataFrame を利用した text フォーマットによるシリアライズ

の代わりに `saveAsObjectFiles()` を利用することにより，性能が向上したこれは，`saveAsTextFiles()` がデータをシリアライズすることにより，ストレージコネクタに送信されるデータの断片化が軽減されたためである．しかし，Java ネイティブのシリアライズ方式は，Apache Spark 内部のデータ構造向けに最適されているわけではないため，実験 1 で見られた約 1 KiB よりは大きいものの，1 Byte から約 15 KiB までの比較的小さい不規則な異なるサイズのチャンクに分割されてしまうことがわかった．そのため，書き込みデータのチャンクが小さいことによる不効率なデータ転送の問題は十分に解決されていない．

実験 2 の傾向として，バッファサイズにかかわらず，オブジェクトサイズ 128MiB 以上では性能がともに飽和しており，実験 1 と同様の傾向を示している．これも，データのチャンクサイズが小さいため，一度に書き込むことができるデータ量に上限ができてしまっているためだと考えられる．

7.3 実験 3: DataFrame を利用したシリアライズフォーマットの変更

実験

実験 3 では，`DataFrame` を利用し，シリアライズフォーマットとして，text，JSON，ORC，Parquet の 4 種類を用いたデータの書き込みの実験を行う．これにより，実験 2 までに明らかになった，ストレージコネクタ内のバッファが十分に使い切れていない問題を軽減することを試みる．

結果

text，JSON，ORC，Parquet の 4 種類のフォーマットによるシリアライズを利用した書き込みの測定結果を，それぞれ図 11，図 12，図 13，図 14 に示す．

まず，図 11 の text でのシリアライズの結果を，図 8 に示された実験 1 における Committer Version 2 とレプリケーションファクター 1 の場合の結果と比較する．text でのシリアライズの場合は，`DataFrame` を利用したとしても，`RDD` より大幅な性能改善は見られなかった．また，実験 2 における Java ネイティブのシリアライズ方式に比べても，性能は高くならなかった．また，実験 1 と実験 2 と同様に，オブジェクトサイズ 128 MiB 以上では性能が飽和してしまった．

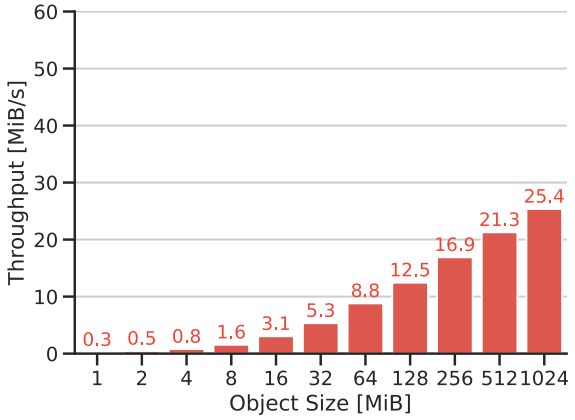

図 12 DataFrame を利用した JSON フォーマットによるシリアライズ

それに対して，図 12，図 13，図 14 の text 以外のシリアライズフォーマットを利用した書き込みの測定結果を見ると，これまでの実験とは異なり，性能の飽和が発生せず，オブジェクトサイズの増加に伴って性能も向上する傾向が現れた．

さらに，これまでは 6 MiB/s 以上の性能が発揮されなかったのとは対照的に，JSON フォーマットでは最大で 25 MiB/s，ORC と Parquet フォーマットでは，最大で 50 MiB/s を上回る大幅な性能向上が得られた．

評価

ORC や Parquet などの効率の良いバイナリのシリアライズフォーマットを利用することにより，ストレージコネクタの書き込み性能の大幅な向上を実現することができた．これは，これらのシリアライザによりストレージコネクタに送信されるデータチャンクのサイズが増加するとともに，Ceph クラスタへのデータの転送回数も減少し，Ceph

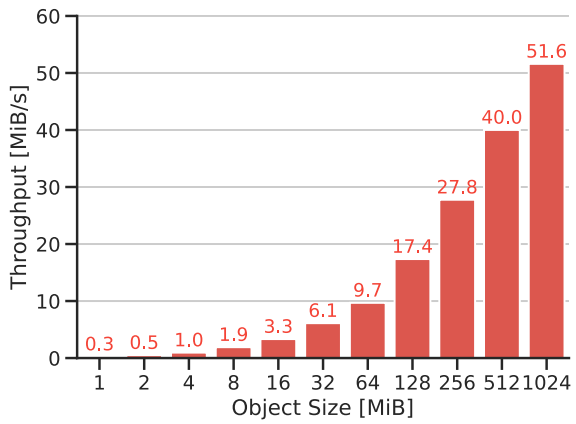

図 13 DataFrame を利用した ORC フォーマットによるシリアライズ

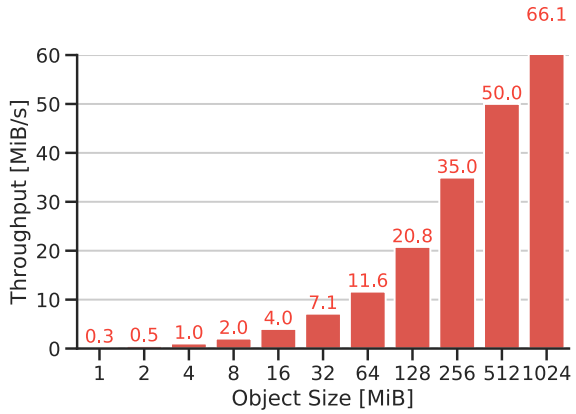

図 14 DataFrame を利用した Apache Parquet フォーマットによるシリアライズ

クラスタに効率的にデータを格納することができるようになったためだと考えられる．

テキスト形式の JSON フォーマットでも大きな性能向上が得られた．JSON フォーマットでシリアライズしたときにストレージコネクタに送られるデータチャンクのサイズを分析すると，比較的小さい 8 KiB のデータチャンクが利用されてした．しかし，`saveAsObjectFiles()` の場合とは異なり，データチャンクの転送の間，データチャンクが一定のサイズを保っていたため，効率的なデータ転送が実現したためであると考えられる．

8. 議論

8.1 Pool ごとのレプリケーションファクターの設定

Ceph のレプリケーションファクターを 1 に設定すると，データの損失を防ぐための冗長性が確保できなくなってしまうが，Ceph のレプリケーションファクターは，オブジェクトデータを格納するために RADOS 上に作成された Pool ごとに柔軟に設定することが可能である．

そのため，たとえば，実験中の一時的なデータを格納する Pool と，長期的にデータを保管しておく Pool を分割し，前者の Pool のレプリケーションファクターを 1 に，後者を 3 にそれぞれ設定し，実験前後にデータを保管する Pool を変更したり，あるいは，実験の前後で Pool のレプリケーションファクターを変更したりすることにより，実験中にストレージコネクタがレプリケーションの影響を受けないようにしながらも，実験前後のデータについては冗長性を保証することは可能である．

9. まとめと今後の課題

9.1 まとめ

筆者らは，分散オブジェクトストレージ Ceph を Apache Spark から利用できるストレージコネクタ `spark-ceph-connector` を開発したが，十分な書き込み性能が得られていなかった．本研究では，`spark-ceph-connector` の書き込み性能が低い理由を分析し，複数の改良を施した．また，`spark-ceph-connector` 固有の問題に対処できる適切な書き込み手法を適用した．

その結果，これらの改善により，128 MiB のオブジェクトサイズに対する書き込み性能が，約 0.6 MiB/s から約 20.8 MiB/s に向上し，1,024 MiB のオブジェクトに対しては，最大で約 66.1 MiB/s まで大幅に改善され，実用的な書き込み性能が発揮できるようになった．

9.2 今後の課題

これまで，`spark-ceph-connector` に対する性能評価としては，読み込みと書き込みの基本的な性能を評価するためのマイクロベンチマークしか行われてこなかった．今後は，より実用的なアプリケーションに近いワークロードを用いたベンチマークを実施することで，ストレージコネクタの実用的な性能を評価する必要がある．

また，`spark-ceph-connector` にデータを送信する Committer には，まだ改善の余地が残っている．近年，S3A ストレージコネクターには zero rename を実現する Committer が実装された．これと同様に，Ceph へのデータ書き込み時に性能向上の妨げとなる rename を回避できる Committer を利用または `spark-ceph-connector` 向けに実装することにより，書き込み性能をさらに改善できると予想される．

謝辞

本研究の一部は，筑波大学計算科学研究センターの学際共同利用プログラム（Cygnus），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）および富士通研究所との共同研究の助成を受けたものです．

本稿の作成にあたって，データのプロットには，`matplotlib` [55] をベースにしたデータ可視化ライブラリ `seaborn` [56][57] を利用しました．また，図の作成には，`mxGraph` [58] をベースにしたサービス diagrams.net [59]

を利用しました.

参考文献

[1] Dean, J. and Ghemawat, S.: MapReduce: Simplified Data Processing on Large Clusters, *Commun. ACM*, Vol. 51, No. 1, p. 107–113 (online), DOI: 10.1145/1327452.1327492 (2008).

[2] The Apache Software Foundation : Apache Hadoop , (online), available from ⟨https://hadoop.apache.org/ (2020-08-20)⟩.

[3] Ghemawat, S., Gobioff, H. and Leung, S.: The Google file system, *Proceedings of the nineteenth ACM symposium on Operating systems principles - SOSP '03*, ACM Press, (online), DOI: 10.1145/945445.945450 (2003).

[4] Shvachko, K., Kuang, H., Radia, S. and Chansler, R.: The Hadoop Distributed File System, *2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST)*, IEEE, (online), DOI: 10.1109/msst.2010.5496972 (2010).

[5] Zaharia, M., Chowdhury, M., Franklin, M. J., Shenker, S., Stoica, I. et al.: Spark: Cluster computing with working sets., *HotCloud*, Vol. 10, No. 10-10, p. 95 (2010).

[6] Shafer, J., Rixner, S. and Cox, A. L.: The Hadoop distributed filesystem: Balancing portability and performance, *2010 IEEE International Symposium on Performance Analysis of Systems & Software (ISPASS)*, IEEE, (online), DOI: 10.1109/ispass.2010.5452045 (2010).

[7] Vorapongkitipun, C. and Nupairoj, N.: Improving performance of small-file accessing in Hadoop, *2014 11th International Joint Conference on Computer Science and Software Engineering (JCSSE)*, IEEE, (online), DOI: 10.1109/jcsse.2014.6841867 (2014).

[8] 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, *Presented as part of the 9th USENIX Symposium on Networked Systems Design and Implementation NSDI 12)*, pp. 15–28 (2012).

[9] Saouabi, M. and Ezzati, A.: A comparative between hadoop mapreduce and apache Spark on HDFS, *Proceedings of the 1st International Conference on Internet of Things and Machine Learning - IML '17*, ACM Press, (online), DOI: 10.1145/3109761.3109775 (2017).

[10] Meng, X., Bradley, J., Yavuz, B., Sparks, E., Venkataraman, S., Liu, D., Freeman, J., Tsai, D., Amde, M., Owen, S., Xin, D., Xin, R., Franklin, M. J., Zadeh, R., Zaharia, M. and Talwalkar, A.: MLlib: Machine Learning in Apache Spark, *J. Mach. Learn. Res.*, Vol. 17, No. 1, p. 1235–1241 (online), DOI: 10.5555/2946645.2946679 (2016).

[11] Islam, N. S., ur Rahman, M. W., Lu, X., Shankar, D. and Panda, D. K.: Performance characterization and acceleration of in-memory file systems for Hadoop and Spark applications on HPC clusters, *2015 IEEE International Conference on Big Data (Big Data)*, IEEE, (online), DOI: 10.1109/bigdata.2015.7363761 (2015).

[12] Luckow, A., Paraskevakos, I., Chantzialexiou, G. and Jha, S.: Hadoop on HPC : Integrating Hadoop and Pilot-Based Dynamic Resource Management, *2016 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, IEEE, (online), DOI: 10.1109/ipdpsw.2016.166 (2016).

[13] Chaimov, N., Malony, A., Canon, S., Iancu, C., Ibrahim, K. Z. and Srinivasan, J.: Scaling Spark on HPC Systems, *Proceedings of the 25th ACM International Symposium on High-Performance Parallel and Distributed Computing - HPDC '16*, ACM Press, (online), DOI: 10.1145/2907294.2907310 (2016).

[14] Lu, X., Rahman, M. W. U., Islam, N., Shankar, D. and Panda, D. K.: Accelerating Spark with RDMA for Big Data Processing: Early Experiences, *2014 IEEE 22nd Annual Symposium on High-Performance Interconnects*, IEEE, (online), DOI: 10.1109/hoti.2014.15 (2014).

[15] Asaadi, H., Khaldi, D. and Chapman, B.: A Comparative Survey of the HPC and Big Data Paradigms: Analysis and Experiments, *2016 IEEE International Conference on Cluster Computing (CLUSTER)*, IEEE, (online), DOI: 10.1109/cluster.2016.21 (2016).

[16] Anderson, M., Smith, S., Sundaram, N., ă, M. C., Zhao, Z., Dulloor, S., Satish, N. and Willke, T. L.: Bridging the gap between HPC and big data frameworks, *Proc. the VLDB Endowment*, Vol. 10, No. 8, pp. 901–912 (online), DOI: 10.14778/3090163.3090168 (2017).

[17] Reuther, A., Byun, C., Arcand, W., Bestor, D., Bergeron, B., Hubbell, M., Jones, M., Michaleas, P., Prout, A., Rosa, A. and Kepner, J.: Scalable system scheduling for HPC and big data, *J. Parallel Distributed Computing*, Vol. 111, pp. 76–92 (online), DOI: 10.1016/j.jpdc.2017.06.009 (2018).

[18] Reuther, A., Byun, C., Arcand, W., Bestor, D., Bergeron, B., Hubbell, M., Jones, M., Michaleas, P., Prout, A., Rosa, A. and Kepner, J.: Scheduler technologies in support of high performance data analysis, *2016 IEEE High Performance Extreme Computing Conference (HPEC)*, IEEE, (online), DOI: 10.1109/hpec.2016.7761604 (2016).

[19] 伊東聰，矢留雅亮， 宮野悟：バイオインフォマティクス分野における HPC 的取り組みの紹介，研究報告ハイパフォーマンスコンピューティング (HPC)，Vol. 2019, No. 8, pp. 1–7 (2019).

[20] Caino-Lores, S., Carretero, J., Nicolae, B., Yildiz, O. and Peterka, T.: Toward High-Performance Computing and Big Data Analytics Convergence: The Case of Spark-DIY, *IEEE Access*, Vol. 7, pp. 156929–156955 (online), DOI: 10.1109/access.2019.2949836 (2019).

[21] Hayot-Sasson, V. and Glatard, T.: Evaluation of Pilot Jobs for Apache Spark Applications on HPC Clusters, *2019 15th International Conference on eScience (eScience)*, IEEE, (online), DOI: 10.1109/escience.2019.00023 (2019).

[22] Yildiz, O. and Ibrahim, S.: On the Performance of Spark on HPC Systems: Towards a Complete Picture, *Supercomputing Frontiers*, Springer International Publishing, pp. 70–89 (online), DOI: 10.1007/978-3-319-69953-0_5 (2018).

[23] Docker : Docker , (online), available from ⟨ https://www.docker.com/ (2020-08-20) ⟩.

[24] Verma, A., Pedrosa, L., Korupolu, M., Oppenheimer, D., Tune, E. and Wilkes, J.: Large-scale cluster management at Google with Borg, *Proceedings of the Tenth European Conference on Computer Systems - EuroSys '15*, ACM Press, (online), DOI: 10.1145/2741948.2741964 (2015).

[25] Schwarzkopf, M., Konwinski, A., Abd-El-Malek, M. and Wilkes, J.: Omega, *Proceedings of the 8th ACM European Conference on Computer Systems - EuroSys '13*, ACM Press, (online), DOI: 10.1145/2465351.2465386 (2013).

[26] The Linux Foundation, The Kubernetes Authors : Kubernetes: Production-Grade Container Orchestration , (online), available from ⟨https://kubernetes.io/ (2020-08-21)⟩.

[27] Piras, M. E., Pireddu, L., Moro, M. and Zanetti, G.: Container Orchestration on HPC Clusters, *Lecture Notes in Computer Science*, Springer International Publishing, pp. 25–35 (online), DOI: 10.1007/978-3-030-34356-9_3 (2019).
[28] IBMSpectrumComputing : lsf-kubernetes: Home of the HPC Compatible Kubernetes Integration for IBM Spectrum LSF , (online), available from ⟨ https://github.com/IBMSpectrumComputing/lsf-kubernetes (2020-08-20) ⟩.
[29] sylabs : wlm-operator: Singularity implementation of k8s operator for interacting with SLURM. , (online), available from ⟨ https://github.com/sylabs/wlm-operator (2020-08-20) ⟩.
[30] Ceph : Ceph Introducing Cephadm , (online), available from ⟨ https://ceph.io/ceph-management/introducing-cephadm/ (2020-08-20) ⟩.
[31] Rook : Rook: Open-Source, Cloud-Native Storage for Kubernetes , (online), available from ⟨ https://rook.io/ (2020-08-20) ⟩.
[32] Mercl, L. and Pavlik, J.: Public Cloud Kubernetes Storage Performance Analysis, *Computational Collective Intelligence*, Springer International Publishing, pp. 649–660 (online), DOI: 10.1007/978-3-030-28374-2_56 (2019).
[33] Lambright, D., Ramesh, N. V., Bhat, A. and Kumar, A.: Lustre in Kubernetes, Santa Clara, CA, USENIX Association (2020).
[34] Volcano : Valcano: A Kuberenetes native system for High Performance Workload , (online), available from ⟨https://volcano.sh/ (2020-08-03)⟩.
[35] The Kubeflow Authors : Kubeflow: The Machine Learning Toolkit for Kubernetes , (online), available from ⟨https://www.kubeflow.org/ (2020-08-03)⟩.
[36] GoogleCloudPlatform : spark-on-k8s-operator: Kubernetes operator for managing the lifecycle of Apache Spark applications on Kubernetes. , (online), available from ⟨ https://github.com/GoogleCloudPlatform/spark-on-k8s-operator (2020-08-20) ⟩.
[37] GoogleContainerTools : kaniko: Build Container Images In Kubernetes , (online), available from ⟨https://github.com/GoogleContainerTools/kaniko (2020-08-01)⟩.
[38] containers : buildah: A tool that facilitates building OCI images , (online), available from ⟨https://github.com/containers/buildah (2020-08-01)⟩.
[39] Docker : Run the Docker daemon as a non-root user (Rootless mode) | Docker Documentation , (online), available from ⟨https://docs.docker.com/engine/security/rootless/ (2020-08-22)⟩.
[40] 畑中智之，建部修見：Cygnus 上での Docker Rootless Mode の利用の検討，研究報告ハイパフォーマンスコンピューティング (HPC)，Vol. 2020, No. 24, pp. 1–6 (2020).
[41] containers : Podman: A tool for managing OCI containers and pods , (online), available from ⟨https://github.com/containers/podman (2020-08-01)⟩.
[42] Randazzo, A. and Tinnirello, I.: Kata Containers: An Emerging Architecture for Enabling MEC Services in Fast and Secure Way, *2019 Sixth International Conference on Internet of Things: Systems, Management and Security (IOTSMS)*, IEEE, (online), DOI: 10.1109/iotsms48152.2019.8939164 (2019).
[43] google : gvisor: Application Kernel for Containers , (online), available from ⟨https://github.com/google/gviso (2020-08-01)⟩.
[44] Young, E. G., Zhu, P., Caraza-Harter, T., Arpaci-Dusseau, A. C. and Arpaci-Dusseau, R. H.: The True Cost of Containing: A gVisor Case Study, *11th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 19)*, Renton, WA, USENIX Association, (online), available from ⟨https://www.usenix.org/conference/hotcloud19/presentation/young⟩ (2019).
[45] 佐藤仁，小川宏高：AI クラウドでの Linux コンテナ利用に向けた性能評価，技術報告 23，国立研究開発法人産業技術総合研究所, 国立研究開発法人産業技術総合研究所 (2017).
[46] 松岡聡，遠藤敏夫， 額田彰，三浦信一，野村哲弘，佐藤仁，實本英之，Drozd, A.：HPC とビッグデータ・AI を融合するグリーン・クラウドスパコン TSUBAME3.0 の概要，技術報告 29，東京工業大学, 東京工業大学, 東京工業大学, 東京工業大学, 東京工業大学, 産業技術総合研究所, 東京工業大学, 東京工業大学 (2017).
[47] 高橋宗史，建部修見：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，研究報告ハイパフォーマンスコンピューティング (HPC)， Vol. 2019, No. 1, pp. 1–8 (2019).
[48] TAKAHASHI Shuuji (@shuuji3) : Spark Ceph Connector: Implemention of Hadoop Filesystem API for Ceph , (online), available from ⟨ https://github.com/shuuji3/spark-ceph-connector (2020-08-20) ⟩.
[49] Ceph : Using Hadoop with CephFS - Ceph Documentation , (online), available from ⟨ https://docs.ceph.com/docs/nautilus/cephfs/hadoop/ (2020-08-20) ⟩.
[50] Steve Loughran : A Zero-Rename Committer: Object-storage as a destination for Apache Hadoop and Spark , (online), available from ⟨ https://github.com/steveloughran/zero-rename-committer (2020-08-20) ⟩.
[51] Amazon : Improve Apache Spark write performance on Apache Parquet formats with the EMRFS S3-optimized committer , (online), available from ⟨ https://aws.amazon.com/jp/blogs/big-data/improve-apache-spark-write-performance-on-apache-parquet-formats-with-the-emrfs-s3-optimized-committer/ (2020-08-20) ⟩.
[52] Vernik, G., Factor, M., Kolodner, E. K., Michiardi, P., Ofer, E. and Pace, F.: Stocator: Providing High Performance and Fault Tolerance for Apache Spark Over Object Storage, *2018 18th IEEE / ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*, IEEE, (online), DOI: 10.1109/ccgrid.2018.00073 (2018).
[53] Hadoop Map/Reduce : [MAPREDUCE-4815] Speed up FileOutputCommitter#commitJob for many output files - ASF JIRA , (online), available from ⟨ https://issues.apache.org/jira/browse/MAPREDUCE-4815 (2020-08-20) ⟩.
[54] Hadoop Map/Reduce : [MAPREDUCE-6336] Enable v2 FileOutputCommitter by default - ASF JIRA , (online), available from ⟨ https://issues.apache.org/jira/browse/MAPREDUCE-6336 (2020-08-20) ⟩.
[55] Hunter, J. D.: Matplotlib: A 2D Graphics Environment, *Computing Sci. & Engineering*, Vol. 9, No. 3, pp. 90–95

(online), DOI: 10.1109/mcse.2007.55 (2007).
[56] Michael Waskom : seaborn: statistical data visualization — seaborn 0.10.1 documentation , (オンライン), 入手先 ⟨ https://seaborn.pydata.org/ (2020-08-20) ⟩.
[57] Michael, W., Olga, B., Joel, O., Maoz, G., Saulius, L., Paul, H., C, G. D., Tom, A., Yaroslav, H., B., C. J., Jordi, W., De, R. J., Cameron, P., Stephan, H., Jake, V., Santi, V., Gero, K., Eric, Q., Pete, B., Marcel, M., Kyle, M., C. Swain, Alistair, M., Thomas, B., Drew, O., Tal, Y., Lee, W. M., Constantine, E., Clark, F. and , Brian: mwaskom/seaborn: v0.10.1 (April 2020) (2020).
[58] jgraph : mxgraph: mxGraph is a fully client side JavaScript diagramming library , (online), available from ⟨ https://github.com/jgraph/mxgraph (2020-08-20) ⟩.
[59] jgraph : drawio: Source to app.diagrams.net , (online), available from ⟨ https://github.com/jgraph/drawio (2020-08-20) ⟩.