Vol.2014-HPC-143 No.1
2014/3/3

不揮発性デバイス向けObject Storageの実装と評価

鷹津 冬将[1,3,a)]　平賀 弘平[1,3]　建部 修見[2,3]

概要：膨大なデータを管理するために分散ファイルシステムが注目されている．分散ファイルシステムのストレージノードでは一般にオブジェクトストレージを使うことによりデータをデバイス上で管理する．ハードディスクよりも高速で汎用的な不揮発性デバイスが登場した今日，オブジェクトストレージにおいてもこのような不揮発性デバイスに適した設計が求められている．本稿では，これまでに開発してきたオブジェクトストレージにおける課題と，アクセス性能を高めたオブジェクトストレージの設計と実装，評価について述べる．この評価においては，提案するオブジェクトストレージがオブジェクトの書き込みだけでなく読み込みにおいても高いアクセス性能があることを示した．

1. はじめに

近年，天文学や生命科学などの科学技術計算分野におけるデータインテンシブコンピューティングやe-サイエンスの分野では，実験によって得られたデータの解析や実験と並行してシミュレーションのために高性能計算機を用いる．このような高性能計算機を用いた解析やシミュレーションにおいては，実験装置の発展やシミュレーション規模の拡大などにより扱うデータ量が増加の一途をたどっている．たとえば，X線自由電子レーザー施設のSACLA [1]では，生成される実験データが最大で1秒間に5.8G bit生成される [2]．SACLAを用いたXFEL照射実験は数日間連続して実施される [3]が，先ほどの値を用いると最大で一日あたり約60TBのデータが生成されることになり，一度の実験で膨大なデータが出力されることが推測される．実験やシミュレーションで生成されたデータはストレージシステムによって管理されるが，ストレージシステムは現状においても先の例のようにペタバイト級のデータを処理する必要がある．データ量は年々増加しているため数年後にはポストペタ，つまりエクサバイト級やゼタバイト級といった膨大なデータ量を処理する必要がでてくることが予想される．このような巨大なデータは1台のストレージデバイスはもちろん，一台の計算機で管理することは不可能なため，現在はGfarm [4]などの分散ファイルシステムを使い複数の計算機上にある複数のストレージデバイスによって管理することが注目されている．

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
2 筑波大学システム情報系
3 独立行政法人科学技術振興機構 CREST
a) takatsu@hpcs.cs.tsukuba.ac.jp

このような科学技術計算分野で用いられる高性能計算機向けの分散ファイルシステムのストレージノードで用いられるストレージデバイスがハードディスクからフラッシュを利用したデバイスなど読み書きが高速なストレージデバイスに移行しつつある．フラッシュを利用したデバイスには書き込みをアトミックに行ったりデータを保持する空間を物理的な容量を遙かに超えて利用できたりするなど，過去のハードディスクなどに比べより柔軟に利用できるデバイスが登場した．これらの機能を使うことでよりこれまでに比べて高い性能が示されることが期待されており，実際にこのような機能を使うことで性能向上を図ったファイルシステムも登場した [5]．しかし，分散ファイルシステムのストレージノードでデータをストレージデバイスに対して保存するバックエンドのシステムとしてファイルシステムを用いると分散ファイルシステムのストレージノードとしては不要なメタデータなどの管理などが行われ，性能低下を招く恐れがある．そのため分散ファイルシステムではバックエンドのシステムとしてデータをファイルとして保存するのではなく，オブジェクトとして保存するオブジェクトストレージが使われることがある．しかしながら，現在のストレージノードで利用されているオブジェクトストレージは利用するストレージデバイスとして従来のハードディスクなどのデバイスを用いることを想定した設計となっており，新しいデバイスに最適化されていないため，デバイスの最大性能を引き出すことができていない．

このような問題に対し筆者らはこれまでに不揮発性デバイス向けのオブジェクトストレージを設計し発表 [6]している．その設計では，書き込み単位でオブジェクトのすべてのバージョンを保存していたが，この設計では読み込み

の際に性能が大きく低下する場合があることが実装の結果判明した．分散ファイルシステムを利用するワークロードとして，バージョンを保存する必要はなく，さらに読み込みを多く行う場合も想定される．このようなワークロード向けには先の設計は不適切であり，別の管理手法が必要とされていた．そのようなために本研究では，バージョンを管理しないことにより間接参照を減らし，アクセス性能を高めた設計とその実装，評価を行った．本稿は 5 章で構成される．第 2 章では，関連研究について述べる．第 3 章では，本研究で行ったアクセス性能を高めたオブジェクトストレージの設計について示す．第 4 章では，オブジェクトストレージの実装について詳述する．第 5 章では，評価実験の詳細及び評価結果を示す．そして第 6 章で，結論と今後の課題及び展望について示す．

2. 関連研究

本稿では，オブジェクトストレージを対象としている．この関連研究としては OBFS [7] や BlobSeer [8] など既存例が存在する．

OBFS は Object-based Storage Device(OSD) 向けに設計されたファイルシステムであり，ストレージの空間を Region と呼ばれる単位で区切り，Region にオブジェクトを格納する点は，これまでに筆者らが提案してきた設計と共通している．一方でバージョン管理を行っておらず，また仮想アドレス空間を利用できる新しいデバイスに対応していない点において，本稿で実装したオブジェクトストレージと異なる．

BlobSeer はデータをオブジェクトとして保存し，バージョン管理を行う分散オブジェクトストレージであるが，BlobSeer はブロックデバイスに対して直接読み書きを行うわけではない．この点において本稿で提案するオブジェクトストレージと異なる．

また，分散ファイルシステムにおいて，ext3 [9]/ext4 [10] や XFS，ZFS，Btrfs [11] などといったファイルシステムをデータをストアするストレージノードのバックエンドとして用いることがある．具体例では，Ceph [12] では Btrfs ベースのオブジェクトストレージが利用されており，Lustre [13] では ext4 や ZFS などのファイルシステムをベースとしたオブジェクトストレージを利用されている．

さらに，仮想アドレス空間を効率的に用いるファイルシステムとしては，DFS(Direct File System) [5] があり，DFS の設計をベースとしたファイルシステムとして directFS がある．しかしながら，このような通常のファイルシステムにおいては，実際のデータ以外にも名前や属性などのメタデータもあわせて管理を行っている．メタデータを管理するために様々な処理を行っていることによるオーバーヘッドにより，分散ファイルシステムのバックエンドとして性能低下を招く可能性がある．

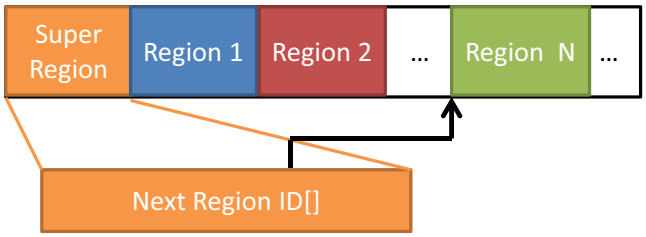


図 1 VSL による空間を Region に分割
[6] より引用

また，不揮発性メモリをストレージデバイスとして使うためのファイルシステムとして，SCMFS [14] や NVMFS [15] などがある．SCMFS はストレージクラスメモリである不揮発性メモリ向けのファイルシステムである．デバイスをメモリを使うファイルシステムとして設計されおり，この点において本研究と異なる．NVMFS は不揮発性メモリと SSD を組み合わせることでアクセス性能を高めたファイルシステムという点が本研究と大きく異なる．

3. オブジェクトストレージの設計

本稿では，不揮発性デバイスとして Fusion-io 社の ioDrive をターゲットとする．ioDrive では SDK を利用することにより，仮想アドレス空間を利用することやアトミックなデータの更新を行うことができる．

SDK によって提供される仮想アドレス空間は実際の物理的な容量以上のアドレスが提供されている．この仮想的なアドレスからデバイス上の物理的なアドレスへのマッピングは ioDrive のドライバが行っており，ユーザーアプリケーションはデバイス上の物理的なアドレスを意識することなくデータを読み書きすることができる．

さらに，ioDrive などのフラッシュを利用した不揮発性メモリでは複数のスレッドからの書き込みリクエストを処理することができ，単一スレッドに比べ高い性能を示す特性がある．

また，本稿が提案するオブジェクトストレージは分散ファイルシステムにおいて利用されることを目的とする．そのため，[6] で提案したオブジェクトストレージではストレージノードマイグレーションを効率的に行うためにバージョン管理を行う設計となっていた．しかしながら，このようにバージョン管理機構を提供すると，I/O 性能が低下する恐れがある．分散ファイルシステムによっては，ストレージノードのマイグレーションのためのバージョン管理をストレージノード以外のサービスによって行うものもある．そのような場合においては，I/O 性能を低下させないためにバージョン管理を行わないオブジェクトストレージが必要となる．

オブジェクトストレージを実現するために，[6] では，SDK によって提供される膨大な仮想アドレス空間を図 1 に示されるように固定長で分割し，この分割された各領域の

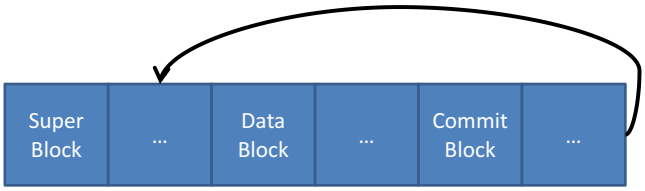


図 2 Version Mode における Region の詳細 [6] より引用

(a) スーパーブロックあり

(b) スーパーブロックなし

図 3 Direct Mode における Region の詳細

ことを Region と定義した．本稿においてもこの設計を踏襲し，仮想アドレス空間を固定長で Region に分割し，ひとつのオブジェクトはひとつの Region を使う．これによって各オブジェクトにアクセスする場合においてアドレスの計算を行う必要をなくしオーバーヘッドを低く抑えたり，多数のオブジェクトの数を保証したりすることができる．

各 Region においてデータを保存する手法として，[6] においては図 2 に示されるように書き込み単位で Commit Block を生成しすべてのバージョンを保存していた．こうすることで，書き込みを高速に行うことや，すべてのバージョンを保存することが可能となっていた．各 Region においてこのようにデータを管理する方法を本稿では Version Mode と定義する．

Version Mode はすべてのバージョンを保持するが，読み込みの際には間接参照を繰り返し行うため，読み込み性能が低下する．分散ファイルシステムのストレージノードではマイグレーションを行うが，マイグレーションのための機能をストレージノード側でサポートしオブジェクトストレージでマイグレーションのための機能をサポートする必要がない場合がある．そのような場合では，より高速に読み書きを行いたいという需要があるため，本稿では高速に読み書きを行う Direct Mode を提案する．この Direct Mode では，各 Region においてデータに対して一切加工せずに書き込みを行う．

Direct Mode の各 Region においてデータを管理する方法を図 3 に示す．図 3 では Region の先頭に Super Block がある図 3(a) に示されるタイプと，Super Block のない図 3(b) に示されるタイプの 2 種類がある．このスーパーブロックはオブジェクトのサイズなどのメターデータを保持する．つまりメタデータを保持する場合は図 3(a) に示されるタイプを用いる．しかしながら，今回ターゲットとしている分散ファイルシステムではオブジェクトのサイズはメタデータサーバによって行われるため，オブジェクトのサイズが必要ない場合がある．その場合は Super Block は必要ないので，図 3(b) に示されるように Region の中は Data Block のみとなる．二つのタイプとも，Data Block を管理する Commit Block の様なブロックが存在しない．これは，すべての書き込みリクエストにおいてデータを一切加工せずに行うことで，書き込み時にはデバイスに対してオブジェクトの ID とオブジェクト中のオフセットを加算することによって得られたアドレスに対してデータをダイレクトに書き込むからである．このように，間接参照をせずにデータをダイレクトに書き込むことが出来るため，高速に書き込みを行うことが可能となっている．読み込み時においても同様で，オブジェクトの ID とオブジェクト中のオフセットからデバイス上でのアドレスが計算によって求められるため間接参照を経ることなく直接読み込みを行うことができるため，読み込み性能が向上すると推定される．

このような設計のため各オペレーションに対応する動作は以下のようになる．

- 作成
  作成処理は基本的には Version Mode と変わらない．各 Region の初期化を行う際に先頭のブロックのみ初期化を行うが，サイズを保持する場合は SuperBlock を書き込み，サイズを保持しない場合はゼロを書き込む．
- 書き込み
  書き込みを行う際には，先述の通りデバイスに対してオブジェクトの ID とオブジェクト中のオフセットを加算することによって得られたアドレスに対してデータをダイレクトに書き込むことができる．
  サイズを保持する必要がある場合には，オフセットに Super Block のサイズ分をさらに加算する．
- 読み込み
  読み込みにおいても書き込みと同様に，オブジェクトの ID とオブジェクト中のオフセットからデバイス上でのアドレスが計算によって求められるため間接参照を経ることなく直接読み込みを行う．

4. 実装

これらの設計に基づきオブジェクトの作成及びデータの読み書きを行うことのできる実装を行った．実装はライブラリの形として実装され，ユーザーがアプリケーションを開発する場合はこのライブラリに含まれる API を利用する．ユーザーアプリケーションが提案するオブジェクトストレージをロードする際には Super Region から各スレッ

ドが作成する次のオブジェクトの ID の一覧をメモリに読み込むことで初期化を行い，設計に基づきオブジェクトの作成やデータの書き込みなどの処理を行う．

4.1 Version Mode

実装における Region の Version Mode の詳細な実装を図 4 に示す．図 4 に示されるように，実装における Version Mode の実装は [6] と変わらない．各 Region において Super Block は，最後の Commit Block のアドレスの他，Region を循環ログとして扱うために Region 内のデータがストアされている先頭と末尾のブロックのアドレスを保持している．Commit Block は，ひとつ前の Commit Block のアドレスと書き込みが行われたタイムスタンプ，書き込まれたデータが保持されている Data Block のアドレスが保持されている．Data Block は，書き込みが行われたデータが保持されている．書き込みリクエストごとに Commit Block を作成し，線形リストとして扱うことでバージョンを管理する．

オブジェクトを作成する際においては，新しく作成されるオブジェクトの ID は，作成を行うスレッドごとに異なり，この ID は Super Region に含まれメモリ上に保持される．そのため，オブジェクトを作成する際には，メモリ上から作成を行うスレッドに対応する ID を取得する実装となっている．このような実装にすることにより作成オペレーションを行う複数のスレッド間でロックを行わなくて済むため性能の向上につながる．作成オペレーションそのものは Super Block の初期化を行う．

Version Mode での書き込みでは，オブジェクトの ID から Super Block のアドレスが示されるため，Super Block のデータを読み込むことで，最後の Commit Block のアドレスを求める．このアドレスを Parent Commit Block ID とする新しい Commit Block となるデータを作成し，書き込みリクエストのデータ及び新しい Super Block のデータとともに書き込みユーザーに結果を返す実装である．

4.2 Direct Mode

一方で Direct Mode では，前述したように各 Region においてデータを管理する方法が Version Mode と異なる．また，サイズを保持する場合と保持しない場合の二通りのパターンが存在し，それぞれで実装が異なる．

サイズを保持する場合の作成オペレーションは，Super Block の初期化を行う．また，オブジェクトの存在確認のために，先頭のセクターになんらかのデータが書き込まれている必要があるため，サイズを保持しない場合でも先頭に 1 セクター分のデータを書き込む．

読み書きを行う各オペレーションは，先述の通りデバイスに対してオブジェクトの ID とオブジェクト中のオフセットを加算することによって得られたアドレスに対してデータをダイレクトに読み書きを行う．サイズを保持する必要がある場合には，オフセットに Super Block のサイズ分をさらに加算することでデータにアクセスを行う．

表 1 評価を行ったノードの性能

CPU	Intel(R) Xeon(TM) E5620 CPU 2.40 GHz (4 コア 8 スレッド) x2
メモリ	24GB
OS (Kernel)	CentOS 6.4 (3.4.4)
ストレージ	Fusion-io ioDrive 160GB

4.3 オブジェクト作成における最適化

Direct Mode においてオブジェクトを作成する際の手続きとして Version Mode と異なる点は，先頭のブロックの初期化内容が異なるのみであり，Super Region の更新については変わらない．

Super Region の更新を行う際に最適化として [6] では，作成のオペレーションごとに Super Region の更新を行うのではなく，まとまった数ごとに Super Region の更新を行う様に実装した．本稿における実装においてもこの最適化を踏襲し，1024 個のオブジェクトを作成するごとに Super Region を更新するように実装した．

この最適化は Version Mode，Direct Mode の両方に適用される．

5. 評価

プロトタイプ実装について複数の手法で評価を行った．

5.1 評価環境

提案するオブジェクトストレージと既存の各ファイルシステムの評価を行ったノードの性能を表 1 に示す．

5.2 比較対象

本章の提案手法との比較対象として XFS, directFS を取り上げる．これらは通常の POSIX ファイルシステムであるが，さまざまな分散ファイルシステムのストレージノードのバックエンドシステムとして POSIX ファイルシステムを使うためこれらを比較対象として選んだ．directFS 以外のファイルシステムは ioDrive を通常のブロックデバイスとして利用し，directFS は SDK が提供するインターフェースを利用するファイルシステムとなっている．

また，これらファイルシステムはユーザーアプリケーションからは VFS を経由してアクセスすることになる．一方で提案手法においては，ユーザー空間で動作するライブラリとして実装してあるため，ユーザーアプリケーションからは，直接 API を呼び出す設計となっている．

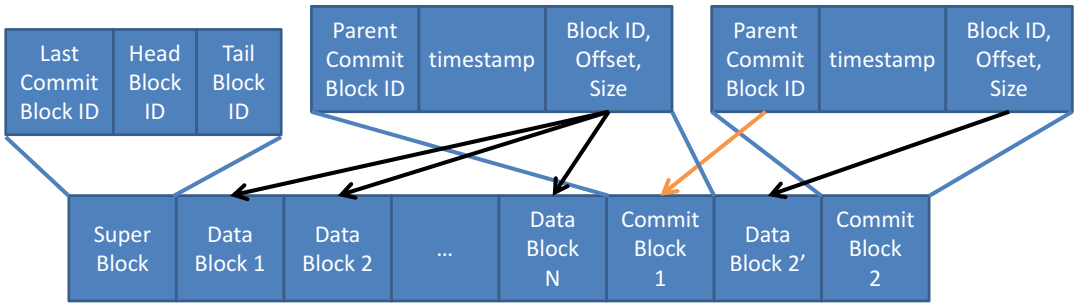


図 4 実装における Version Mode の Region の詳細
[6] より引用

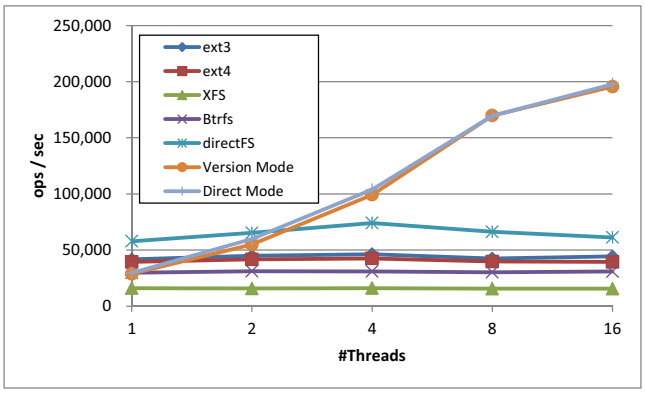


図 5 単位時間あたりのオブジェクト作成性能の評価結果

5.3 オブジェクト作成性能評価

この評価では，単位時間に作成することのできるオブジェクトの数を各ファイルシステムと提案手法について性能を評価を行う．分散ファイルシステムにおいては，複数のクライアントから同時にファイルを作成するリクエストが届く．そのため，この評価においては実際にオブジェクトを生成するワーカースレッドの数を変化させて評価を行う．

本評価では各ファイルシステムと提案手法についてスレッド別に 10 回ずつ性能を評価し，平均を算出することで評価を行う．

評価結果を図 5 に示す．この図 5 においては，横軸がワーカースレッドの数であり，縦軸が単位時間あたりに作成したオブジェクトの数を表している．そのため，縦軸の数値が高い方が高い性能を示している．

この評価では，既存のファイルシステムがスレッド数にかかわらずほぼ一定の性能を示すのに比べ，提案手法が最適化の有無にかかわらずスレッド数の増加に合わせて単位時間あたりに作成されるオブジェクトの数が増加することを示している．特に 16 スレッドの場合においては最適化を行っている場合の提案手法が既存のファイルシステムの中で最も高い性能を示している directFS に比べると 3.34 倍と高い性能を示している．これは，提案手法が既存のファイルシステムとは違い，新しく作成されるオブジェクトの ID やブロックの位置を他のスレッドと競合しないような設計になっているためと考えられる．

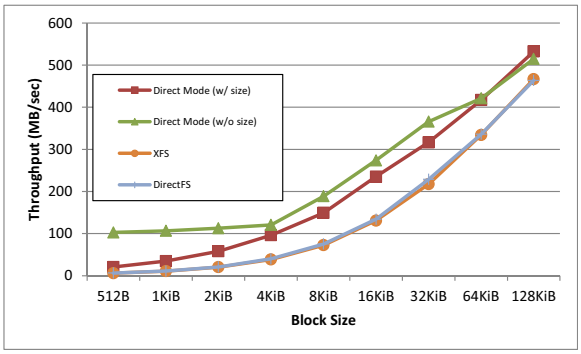
また，この評価結果からは 16 スレッドの場合において最適化を行うことで最適化を行わない場合に比べると 164%の性能が出ていることが示されている．これは最適化を行うことにより，作成オペレーションのたびに書き込まれるデータ量が減少したためと考えられる．

5.4 読み込みバンド幅の評価

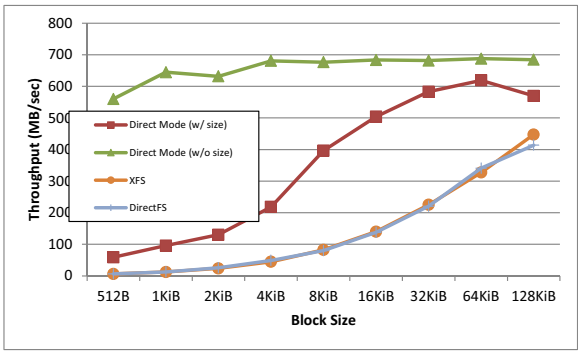
この評価では，単位時間あたりに読み込むことのできるデータサイズを読み込むブロックサイズ別に評価を行う．評価の際には，提案手法及び既存のファイルシステムにおいてあらかじめ作成されデータが書き込まれてあるオブジェクトやファイルに対して逐次，あるいはランダムに 1GiB のデータを読み込む．既存のファイルシステムにおいて，読み込むデータの書き込みを行う際には書き込みごと同期を行い確実にストレージデバイス側にデータを反映させる．読み込むデータの準備を行う際には，指定されたブロックサイズで合計 1GiB 分のデータを書き込むことで準備を行う．本評価では各ファイルシステムと提案手法についてブロックサイズ別に 10 回ずつ性能を評価し，平均を算出することで評価を行う．

評価結果を図 6 に示す．図 6(a) はランダムに読み込みを行った場合の評価結果を，図 6(b) は逐次的に読み込みを行った場合の評価結果をそれぞれ表す．図 6 においては，横軸がブロックサイズであり，縦軸が単位時間あたりに書き込むことのできたデータサイズを示している．そのため，縦軸の数値が高い方が高い性能を示している．図 6 の各グラフにおいて Version Mode の結果が描かれていないが，これは性能が低く評価にかかる時間が 1,000 秒以上かかり途中で評価を打ち切ったためである．

この評価結果からは，提案手法のうち Direct Mode が他のファイルシステムに比べブロックサイズが 1MiB 以下の場合に高い読み込み性能を示すことを表している．特にブロックサイズが 128KiB の場合の逐次読み込みの評価においてはサイズを保持する場合で XFS の 1.27 倍，サイズを保持しない場合で 1.53 倍の性能を示している．また，ブロックサイズが 128KiB の場合のランダム読み込みの評価

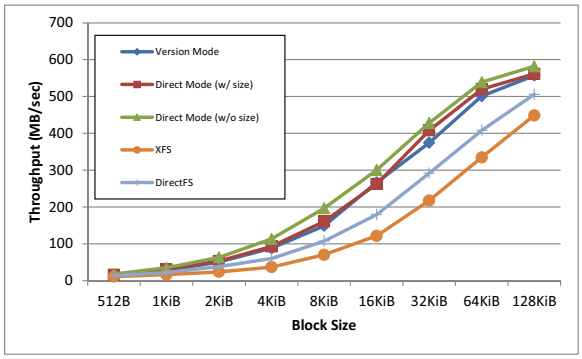


(a) ランダム読み込み

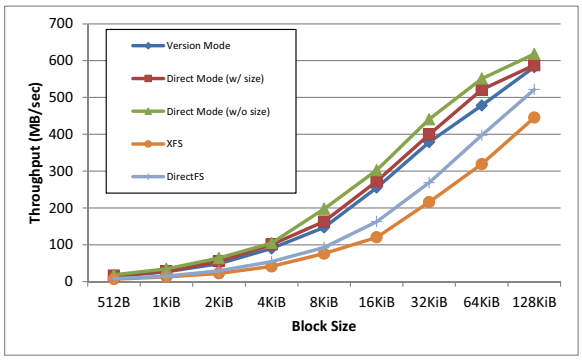


(b) 逐次読み込み

図 6 ブロックサイズ別の単位時間の読み込み性能の評価結果



(a) ランダム書き込み



(b) 逐次書き込み

図 7 ブロックサイズ別の単位時間の書き込み性能の評価結果

においてはサイズを保持する場合で XFS の 1.14 倍，サイズを保持しない場合で 1.10 倍の性能を示している．これは読み込みの際に間接参照が少ないためと考えられる．

5.5 書き込みバンド幅の評価

この評価では，単位時間あたりに書き込むことのできるデータサイズを書き込むブロックサイズ別に評価を行う．評価の際には，提案手法及び既存のファイルシステムにおいてあらかじめ作成してあるオブジェクトやファイルに対して逐次，あるいはランダムに 1GiB のデータを書き込む．既存のファイルシステムにおいては，書き込みごとに同期を行い確実にストレージデバイス側にデータを反映させる．本評価では各ファイルシステムと提案手法についてブロックサイズ別に 10 回ずつ性能を評価し，平均を算出することで評価を行う．

評価結果を図 7 に示す．図 7(a) はランダムに書き込みを行った場合の評価結果を，図 7(b) は逐次的に書き込みを行った場合の評価結果をそれぞれ表す．図 7 においては，横軸がブロックサイズであり，縦軸が単位時間あたりに書き込むことのできたデータサイズを示している．そのため，縦軸の数値が高い方が高い性能を示している．

この評価結果からは，提案手法とすべてのファイルシステムにおいて書き込み性能が書き込まれるデータのブロックサイズの増加にあわせて転送速度が高くなっていることが示されている．

ダイレクトモードにおいてはブロックサイズが 128KiB の場合の逐次書き込みの評価においてはサイズを保持する場合で XFS の 1.13 倍，サイズを保持しない場合で 1.18 倍の性能を示している．また，ブロックサイズが 128KiB の場合のランダム書き込みの評価においてはサイズを保持する場合で XFS の 1.11 倍，サイズを保持しない場合で 1.15 倍の性能を示している．ログモードにおいては，ブロックサイズが 128KiB の場合における逐次書き込み評価で XFS の 1.12 倍，ランダム書き込み評価で XFS の 1.10 倍の性能を示している．これは書き込みの際に間接参照が少ないためと考えられる．

6. まとめ

本稿では，高速な分散ファイルシステムの実現に向けて，高速で高機能なストレージデバイスにおいて効率的にデータを保存することのできるオブジェクトストレージについて，これまでに行ってきた設計に対する課題を解決する設計の提案を行い，それを含むオブジェクトストレージを実装し，その評価をさまざまなアクセスパターンで行った結果について詳述した．

オブジェクト作成性能の評価では，オブジェクトを 1 秒間に作成することのできる性能をさまざまなスレッド数で評価を行った．この評価では，提案手法がスレッド数の増加に応じて高い性能を示した．特に 16 スレッドの場合においては，既存のファイルシステムの中で最も高い性能を示している directFS に比べると 3.34 倍と高い性能を示した．

読み込み性能の評価では，あらかじめ作成されているオブジェクトに対して，1 秒間に読み込むことの出来る性能をさまざまなブロックサイズで評価を行い，提案手法のうち Direct Mode がブロックサイズが 128KiB の逐次読み込みの際に 684MB/s に達した．これは XFS に比べると 1.5 倍の性能であり，Direct FS に比べ 1.6 倍の性能である．一方で Version Mode においては読み込み性能が他のファイルシステムなどに比べ，著しく低い結果となった．これは，データの読み込み以外にも Commit Block の読み込みなどのオーバーヘッドが存在するためと考えられる．

書き込み性能の評価では，あらかじめ作成されているオブジェクトに対して，1 秒間に書き込むことの出来る性能をさまざまなブロックサイズで評価を行い，提案手法のうち Direct Mode がブロックサイズが 128KiB のランダム書き込みの際に 581MB/s に達した．これは同じブロックサイズの Direct FS に比べ 1.1 倍の性能であり，XFS に比べると 1.2 倍の性能である．

今後の課題としては，以下のことが挙げられる．

(1) 読み込み機能の最適化
Version Mode においては読み込み性能が著しく低い結果となった．これは線形リストの走査などによる多数の I/O リクエストが発行されたためと考えられる．そのため，読み込み機能の最適化が必要である．

(2) クリーナーの実装と評価
プロトタイプ実装では，クリーナーを実装していない．分散ファイルシステムとして利用するためには必要な機能と考えられるため実装する必要がある．

謝辞 本研究の一部は，JST CREST「ポストペタスケールデータインテンシブサイエンスのためのシステムソフトウェア」および，JST CREST「EBD：次世代の年ヨッタバイト処理に向けたエクストリームビッグデータの基盤技術」による．

参考文献

[1] 独立行政法人 理化学研究所 放射光科学総合研究センター X 線自由電子レーザー施設 SACLA, http://xfel.riken.jp/index.html.
[2] Sugimoto, T., Joti, Y., Ohata, T., Tanaka, R., Yamaga, M. and Hatsui, T.: Large-bandwidth data acquisition network for XFEL facility, SACLA, *Proceedings of ICALEPCS2011, Grenoble, France*, Vol. 626 (2011).
[3] 吉永一美，徳久淳師，大野善之，亀山豊久， 堀敦史，城地保昌，初井宇記， 石川裕：京での大量データの並列相関計算を支援するソフトウェアの提案，情報処理学会研究報告. [ハイパフォーマンスコンピューティング], Vol. 2013, No. 25, pp. 1–8 (2013).
[4] Tatebe, O., Hiraga, K. and Soda, N.: Gfarm Grid File System, *New Generation Comput.*, Vol. 28, No. 3, pp. 257–275 (2010).
[5] Josephson, W. K., Bongo, L. A., Li, K. and Flynn, D.: Dfs: A file system for virtualized flash storage, *ACM Transactions on Storage (TOS)*, Vol. 6, No. 3, p. 14 (2010).
[6] 鷹津冬将，平賀弘平，建部修見：不揮発性デバイス向けの Object Storage の設計，情報処理学会研究報告. [ハイパフォーマンスコンピューティング], Vol. 2013, No. 12, pp. 1–7（オンライン），入手先 ⟨http://ci.nii.ac.jp/naid/110009588132/⟩ (2013).
[7] Wang, F., Brandt, S. A., Miller, E. L. and Long, D. D. E.: OBFS: A File System for Object-based Storage Devices, *IN PROCEEDINGS OF THE 21ST IEEE / 12TH NASA GODDARD CONFERENCE ON MASS STORAGE SYSTEMS AND TECHNOLOGIES, COLLEGE PARK, MD*, pp. 283–300 (2004).
[8] Nicolae, B., Antoniu, G., Bougé, L., Moise, D. and Carpen-Amarie, A.: BlobSeer: Next Generation Data Management for Large Scale Infrastructures, *Journal of Parallel and Distributed Computing*, Vol. 71, No. 2, pp. 168–184 (2011).
[9] Ts'o, T. Y. and Tweedie, S.: Planned Extensions to the Linux Ext2/Ext3 Filesystem, *Proceedings of the FREENIX Track: 2002 USENIX Annual Technical Conference*, Berkeley, CA, USA, USENIX Association, pp. 235–243 (2002).
[10] Mathur, A., Cao, M., Bhattacharya, S., Dilger, A., Tomas, A. and Vivier, L.: The New ext4 Filesystem: Current Status and Future Plans, *Proceedings of the 2007 Linux Symposium*, pp. 21–33 (2007).
[11] Btrfs, https://btrfs.wiki.kernel.org/.
[12] 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*, OSDI '06, Berkeley, CA, USA, USENIX Association, pp. 307–320 (2006).
[13] Koutoupis, P.: The lustre distributed filesystem, *Linux J.*, Vol. 2011, No. 210 (2011).
[14] Wu, X. and Reddy, A. L. N.: SCMFS: A File System for Storage Class Memory, *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '11, New York, NY, USA, ACM, pp. 39:1–39:11 (online), DOI: 10.1145/2063384.2063436 (2011).
[15] Qiu, S. and Reddy, A.: NVMFS: A hybrid file system for improving random write in nand-flash SSD, *Mass Storage Systems and Technologies (MSST), 2013 IEEE 29th Symposium on*, pp. 1–5 (online), DOI: 10.1109/MSST.2013.6558434 (2013).