Vol.2016-HPC-155 No.1
2016/8/8

高性能分散ファイルシステムのための分散メタデータサーバ PPMDSの評価

鷹津 冬将[1,a)]　平賀 弘平[1]　建部 修見[2]

概要：ハイパフォーマンスコンピューティングの分野で広く利用されている分散ファイルシステムのボトルネックとなっているメタデータ処理について高性能化を図った PPMDS について，低レイテンシのネットワークを用いた環境において性能を評価し，既存の分散ファイルシステムのメタデータサーバである IndexFS と比較する．また，分散ファイルシステムはメタデータサーバだけではファイルの読み書きができない．筆者らがこれまで提案してきた OpenNVM を用いた高性能ローカルオブジェクトストレージと組み合わせ，その場合の性能を評価した．さらに，ファイルの作成リクエストのたびにオブジェクトストレージでオブジェクトを作成した場合の性能劣化を最小限に抑え，PPMDS 本来の性能を引き出すために複数のオブジェクトを予め作成する Bulk Creation を提案し，その性能を評価した．評価においては，メタデータサーバを 5 ノードで実行し，128 プロセスのクライアントからのアクセス時に，PPMDS が IndexFS に比べて 2.60 倍の性能となった．また，ローカルオブジェクトストレージと組み合わせた場合には，PPMDS 単体の性能に比べて，128 プロセスのクライアントからのアクセス時に 62.8 %，16 プロセスの場合に 11.5 %の性能となったが，Bulk Creation を適用させることで，それぞれ 83.7 %および 65.2 %の性能となり，適用前に比べ，それぞれ 1.34 倍，5.69 倍となった．

1. はじめに

ハイパフォーマンスコンピューティング分野とビッグデータ処理の融合したエクストリームビッグデータ処理の分野における課題は分散ファイルシステムなどのストレージシステムである．これらの分野では，チェックポイントとしてプロセスごとにファイルを作成するなど多数のファイルを同時に作成するワークロードがあり，その高性能化が課題のひとつとなっている．多数のファイルを作成するワークロードとしては他にも，ゲノムシーケンスやイメージプロセッシング，電話やビデオの記録などがある．こうした課題に対して，これまで PLFS [1] や GIGA+ [2], IndexFS [3] などの多数の研究がなされてきた．平賀らは，分散ファイルシステムにおいてはメタデータ管理機構がボトルネックとなっていることに着目し，これまでにメタデータ管理サーバを複数ノードへ分散化を行いスケーラビリティを向上させたメタデータサーバ PPMDS [4] を提案し，10Gb Ethernet と Gigabit Ethernet の混在する環境において性能を評価した．桐井らは PPMDS を大規模実行した際の性能をシミュレーションにより評価を行い，ネットワークの遅延の大きさが PPMDS の性能を左右すると考察を行っている [5]．現在のハイパフォーマンスコンピューティングの分野においては Infiniband など Ethernet に比べて遅延の小さなネットワークが用いられるため，そのような低遅延のネットワーク環境での性能評価が重要である．

また，筆者らはこれまでに OpenNVM [6] を用いた高性能分散ファイルシステムのためのローカルオブジェクトストレージを提案してきた [7]．これはストレージサーバにおいて，ストレージデバイス上で，ファイルのデータをオブジェクトとして管理を行うバックエンドとして用いる．このローカルオブジェクトストレージは，オブジェクト作成においてスレッド数に比例し高い性能を示すことを目的としている．しかしローカルオブジェクトストレージだけでは，分散ファイルシステムのストレージサーバとして利用することが出来ない．さらに PPMDS は，分散ファイルシステムが提供するべき機能のうち，メタデータ操作に関する機能のみしか提供していない．分散ファイルシステムにおいては，ファイルのデータの管理が必要であり，そのためにはメタデータサーバとストレージサーバを組み合わせる必要がある．その際には，ストレージサーバ上でのオブジェクトの ID をメタデータの一つとしてメタデータサーバで管理をすることになり，メタデータサーバの拡張

[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] 筑波大学計算科学研究センター
[a)] takatsu@hpcs.cs.tsukuba.ac.jp

が必要となる．ファイルを作成する際の単純な手法としては，ファイル作成のリクエストがメタデータサーバに発行された際に，ストレージサーバにおいてもファイルのデータを管理するオブジェクトのエントリを作成し，そのオブジェクトの ID をメタデータの一つとしてメタデータサーバで管理を行うが，メタデータサーバ単体の性能に比べてストレージサーバと組み合わせたことによる性能が変化を明らかにする必要がある．

本稿での貢献は，以下の 3 点である．

- PPMDS について低遅延のネットワークである Infiniband を用いた環境で評価を行い，PPMDS と同様にキーバリューストアでメタデータの管理を行う IndexFS [3] と性能を比較し，メタデータサーバのノード数が 5 でクライアントが 128 プロセスの場合に，PPMDS は IndexFS の 2.60 倍の性能となっていることを示した．
- OpenNVM を用いた高性能分散ファイルシステム向けローカルオブジェクトストレージ [7] をベースとした分散ファイルシステムのストレージサーバ PPOSS の提案，及び PPMDS と PPOSS の組み合わせた分散ファイルシステムの設計の提案し，初期評価を行った．
- メタデータサーバである PPMDS とストレージサーバ PPOSS を組み合わせた場合においても，メタデータサーバ本来の性能を引き出すための最適化 Bulk Creation を提案し，最適化を適用しない場合に比べて，メタデータサーバが 5 ノードでクライアントが 128 プロセスの場合に 1.34 倍，16 プロセスの場合に 5.69 倍の性能を示した．

本稿は 7 章で構成される．第 2 章では，PPMDS と筆者らがこれまでに設計開発してきた OpenNVM を用いた高性能分散ファイルシステムためのローカルオブジェクトストレージについて詳述する．第 3 章では，ローカルオブジェクトストレージをストレージサーバとし，PPMDS と組み合わせた分散ファイルシステムの設計について示す．第 4 章では，PPMDS とストレージサーバを組み合わせた分散ファイルシステムの実装について詳述し，ファイルの作成時に性能を高める最適化 Bulk Creation 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．第 6 章では，関連研究について示す．そして第 7 章で，結論と今後の課題及び展望について示す．

2. 背景

この章では，高性能分散ファイルシステムのための分散メタデータサーバ PPMDS と，ファイルのデータを保持するオブジェクトストレージである PPOSS について述べる．

2.1 PPMDS

一般的なファイルシステムにおいて，ファイルやディレクトリには実データの他にも名前空間や inode などのメタデータによって管理を行っている．メタデータにはその要素の番号である inode 番号のほか，属性やサイズ，更新時刻などが含まれる．従来のローカルファイルシステムにおけるこれらのメタデータの管理手法は，ローカルのブロックデバイス上で管理するために最適化されており，分散ファイルシステムのように複数のノード上で管理することには適していない．inode などのファイルに関するメタデータについては，ファイルと 1 対 1 対応しているため，単純な構造で管理できる．一方で，パスなどの名前空間についてはディレクトリのツリー構造で表わされるため，特定のディレクトリに多数のファイルを作成する場合などにおいてアクセスが集中し，並列性が阻害される可能性があり，分散ファイルシステムのメタデータサーバを設計する際の課題となっている．

PPMDS [4] は高性能ファイルシステムのための分散メタデータサーバである．これまでの並列ファイルシステムでは単一ディレクトリへの大量のファイル作成などのファイルシステムに対する操作のスケーラビリティが十分で無いという問題があった．ファイルには親の要素であるディレクトリがあるが，PPMDS では，この親のディレクトリの inode 番号と，自身のファイル名（あるいはディレクトリ名）のペアを Key とし，自身のメタデータを Value とした Key-Value 形式でメタデータのデータ構造を表現し，スケーラブルなキーバリューストア上で管理を行う．こうすることで，特定のディレクトリ内に含まれるファイルを参照する際には，このディレクトリの inode 番号を用いて Key を範囲検索することで可能となる．PPMDS では複数のサーバにメタデータを分散させるために，このキーバリューストアに先に示したデータ以外にも inode 等のメタデータのエントリの分散先サーバのリストを格納する．この分散先サーバのリストはそのディレクトリのメタデータを管理するすべてのメタデータサーバ上のキーバリューストアで管理される．クライアントがファイルを作成する際には，そのファイルの親ディレクトリの inode 番号を取得，任意のメタデータサーバに対してファイルの作成リクエストを発行し，サーバは，ファイル作成のトランザクションを開始して，そのディレクトリのエントリの分散先サーバのリストの中からファイルのメタデータを格納するノードを決定，作成し，トランザクションをコミットする．このようにすることで，PPMDS は同一ディレクトリ内においても多数のファイルを複数のメタデータサーバ間で分散して作成することができるため，高い並列性を実現した．

しかし，PPMDS は分散ファイルシステムが提供する機能のうち，メタデータ操作に関する機能のみしか提供していない．分散ファイルシステムにおいては，ファイルのデータの管理が必要であり，そのためにはストレージサーバと組み合わせる必要がある．その際には，ストレージ

サーバ上でのオブジェクトの ID をメタデータの一つとしてメタデータサーバで管理をすることになり，メタデータサーバの拡張が必要となる．

2.2 OpenNVM を用いた分散ファイルシステムのためのローカルオブジェクトストレージ

筆者らは，これまでに分散ファイルシステムのストレージノードにおいて，ストレージデバイス上でデータをオブジェクトとして管理するために，OpenNVM [6] を用いた分散ファイルシステムのためのローカルオブジェクトストレージを開発してきた [7]．OpenNVM は，48bit のアドレス空間や，複数のアドレスへの書き込みやトリムをアトミックに行う機能を提供している．このローカルオブジェクトストレージでは，物理デバイスの容量を超えて利用することのできるこの仮想アドレス空間を固定長の Region の配列に分割し，各 Region と各オブジェクトを一対一対応させている．そうすることで，オブジェクトの ID として Region の先頭のセクタ番号を利用することができ，オブジェクトの作成や参照時に間接参照を減らし，性能を高めている．また，各 Region 内ではデータのバージョンを管理する Version Mode とアクセス性能を重視する Direct Mode があり，Direct Mode ではサイズの管理を行うものと行わないもののふたつのモードを提供している．オブジェクトの作成の際には，各 Region の先頭のセクタへ書き込みを行い，オブジェクトストレージ全体のメタデータを管理している先頭の Region の更新を行う．その際に，複数のオブジェクトをまとめて初期化を行う Bulk Initialization や先頭の Region の更新回数を減らす Bulk Reservation の二つの最適化を行うことで，最大で 1 秒間に 74 万個のオブジェクトを作成することのできる性能が示した．

しかしローカルオブジェクトストレージだけでは，分散ファイルシステムのストレージサーバとして利用することは出来ない．ストレージサーバとしてネットワークを操作する部分について，メタデータサーバと組み合わせた際に性能の劣化が最小限に抑えられるように設計・実装する必要がある．

3. 設計

この章では，ローカルオブジェクトストレージをバックエンドとしたストレージサーバと PPMDS と組み合わせた分散ファイルシステムの設計について示す．

3.1 PPOSS

OpenNVM を用いた高性能分散ファイルシステムのためのローカルオブジェクトストレージ [7] は，単体ではストレージサーバとして用いることは出来ない．このローカルオブジェクトストレージをバックエンドとした，ストレージサーバを設計し PPOSS とする．PPOSS も OpenNVM を使うため，48bit の仮想アドレス空間を利用できる．文献 [7] と同様にこの仮想アドレス空間を固定長の Region の配列として分割する．各 Region をひとつのオブジェクトとして用いることで，オブジェクトへのアクセス時にはオブジェクトの ID から直接 Region を特定できる．文献 [7] では，Region 内でデータを管理する方法としてすべてのバージョンを保持する Version Mode と，アクセス性能を高める Direct Mode があった．この Direct Mode を用いる場合，データへのアクセスにはオブジェクトの ID とオフセットからデバイス上のセクタ番号を計算で求めることができ，高いアクセス性能が期待できる．クライアントから作成リクエストが発行された際には，新しい Region の先頭のセクタを初期化し，その Region の先頭のセクタ番号を Region の大きさで割った値をオブジェクトの ID としてクライアントに返す．複数のクライアントからのアクセスのため PPOSS も複数スレッドで動作するが，新しい Region の番号を複数のスレッドで共有すると衝突により性能が劣化する．それを回避するために，スレッド毎にその値をそれぞれ保有する．

PPOSS はクライアント・サーバ方式の分散オブジェクトストレージであるが，分散ファイルシステムのためのストレージサーバでもある．ファイルの実データをひとつのオブジェクトに対応させることで，オブジェクトのサイズなどのメタデータはすべてメタデータサーバで管理することができるため，PPOSS ではメタデータの管理を行わない．また，サーバ同士は独立しており，サーバ間の通信は行わない．

PPOSS が提供する機能は，オブジェクトの作成と読み書き，削除である．クライアントは，PPOSS のサーバのリストの中から任意の PPOSS のサーバに対して作成リクエストを発行し，そこでのオブジェクトの ID とサーバの ID から PPOSS 全体のオブジェクトの ID を発行する．読み書きや削除の際には，そのオブジェクトの ID から特定の PPOSS のサーバに対してオブジェクトの操作を行う．

3.2 PPOSS と PPMDS を組み合わせた分散ファイルシステムの設計

PPMDS はクライアントである ppfuse とメタデータサーバである ppmds で構成され，これらとストレージサーバである PPOSS を組み合わせ，分散ファイルシステムを実現する．分散ファイルシステム全体の構成を**図 1** に示す．図 1 に示されるように，複数のメタデータサーバ，ストレージサーバ，クライアントが同一のネットワーク上に存在し，ストレージサーバである PPOSS はローカルオブジェクトストレージを用いてデータを管理する．

作成時の処理の手順を**図 2** に示す．ファイルを作成する際は，図 2 中の (1) の様にクライアントよりメタデータサーバへリクエストが発行される．メタデータサーバで

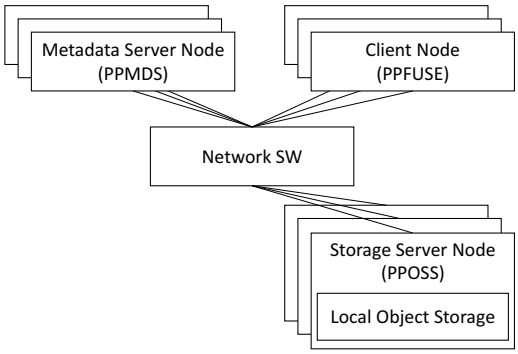


図 1 分散ファイルシステム全体の構成

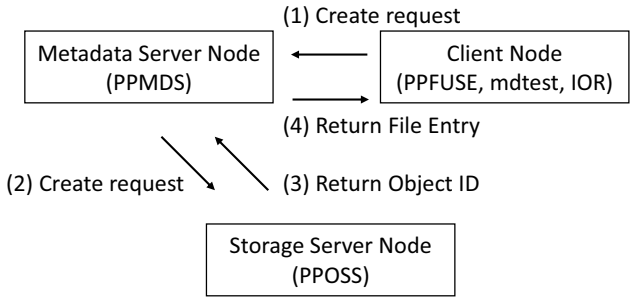


図 2 作成時の手順

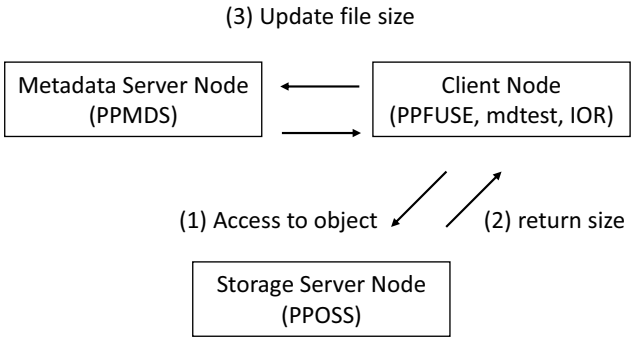


図 3 読み書きの際の手順

は，リクエストを受け，図 2 中 (2) のようにストレージサーバへファイルを格納するオブジェクトのエントリを作成する．ストレージサーバは，オブジェクトを作成しそのオブジェクトの ID を返す (図 2 中 (3))．メタデータサーバでは，そのオブジェクトの ID を含むファイルのエントリを作成し，そのファイルのエントリを返す (図 2 中 (4))．その際に，ファイルのオブジェクトの ID が含まれているため，クライアントはこの ID を用いてファイルの読み書きが可能となる．

読み書きの際の処理の手順を**図 3** に示す．ファイルの作成時に，メタデータサーバより，ストレージサーバでのオブジェクトの ID を取得できているため，このオブジェクトの ID を用いてファイルの読み書きを行う (図 3 中 (1))．読み書きを行った際には，オブジェクトサーバより読み書きすることの出来たデータのサイズがクライアントに返される (図 3 中 (2))．また，追記書き込みなど，ファイルサイズが更新された場合には図 3 中 (3) のように，ファイルのクローズ時にメタデータサーバにファイルサイズの更新をリクエストし，メタデータを更新する．

4. 実装

先の章でも示したように，PPOSS は OpenNVM を用いた高性能分散ファイルシステムのためのローカルオブジェクトストレージをバックエンドとしたストレージサーバデーモンであり，通信部分には msgpack-rpc [8] を用いて実装を行った．また，本稿ではプロトタイプ実装のため，作成と読み書きの部分についてのみ実装を行った．ローカルオブジェクトストレージでのデータの管理を行うモードについてはアクセス性能を重視する Direct Mode without Size を用いた．また，ローカルオブジェクトストレージでは複数のオブジェクトをまとめて初期化を行う Bulk Initialization や先頭の Region の更新回数を減らす Bulk Reservation の二つの最適化がある．これらのパラメータはともに 64 とした．そのため，ストレージサーバに作成リクエストが発行されても実際にストレージデバイスへの書き込みが行われるのは 64 回に 1 回となる．オブジェクトの ID は符号なし 64 ビット整数とし，先頭 32 ビットをストレージサーバの ID，残りの 32 ビットをそのストレージサーバ上でのオブジェクトの ID とした．

PPOSS を含むファイルシステム全体におけるファイル作成時の手続きとしては以下のようになる．

(1) クライアントがメタデータサーバに対してファイルの作成をリクエストする
(2) メタデータサーバがそのファイルの inode 番号を発行する．
(3) その inode 番号を Key とした Jump Consistet Hash 法 [9] でオブジェクトを発行するストレージサーバを決定する．
(4) そのストレージサーバに対して，オブジェクトの作成リクエストを発行する．
(5) inode エントリやオブジェクトの ID などのメタデータのエントリを格納する．

4.1 Bulk Creation

先の実装において，メタデータサーバがストレージサーバのクライアントとなり，ファイルの作成リクエストが発行されるたびにストレージサーバに対してオブジェクトの作成リクエストを発行し，ストレージサーバからオブジェクトの ID が返ってくるまでブロックする．メタデータサーバとストレージサーバが同一ノードにない場合，オブジェクトの作成のためにネットワークをまたぐため性能が低下するおそれがある．PPOSS のバックエンドとなるローカルオブジェクトストレージでは，オブジェクト作成の処理で各オブジェクトの先頭のセクタを初期化するが，

作成リクエストが発行されるたびに初期化を行うのではなく，複数回に一度まとめて複数のオブジェクトの初期化を行うことで性能を高める最適化 Bulk Initialization がある．我々の実装する分散ファイルシステムのプロトタイプにおいてもこの最適化手法の概念を踏襲し，メタデータサーバがストレージサーバにオブジェクトの作成リクエストを発行する回数を減らすために，N 回に一度，まとめて N 個分のオブジェクトの作成リクエストを発行する最適化 Bulk Creation を行う．この最適化を適用した場合の全体のファイル作成時の手続きとしては以下のようになる．

(1) メタデータサーバは予めすべてのストレージサーバに対して，それぞれ N 個分のオブジェクトの作成リクエストを発行し，ストレージサーバの ID をキーとした連想配列に追加する．
(2) クライアントがメタデータサーバに対してファイルの作成をリクエストする
(3) メタデータサーバがそのファイルの inode 番号を発行する．
(4) その inode 番号を Key とした Jump Consistet Hash 法でオブジェクトを発行するストレージサーバを決定する．
(5) そのストレージサーバで予め作成してあるオブジェクトの ID のリストが空の場合は，新たに N 個分のオブジェクトの作成リクエストを発行し，(1) の連想配列に追加する．
(6) そのストレージサーバで予め作成してあるオブジェクトの ID をメタデータサーバにひも付け，連想配列から使用したキーを削除する．
(7) inode エントリやオブジェクトの ID などのメタデータのエントリを格納する．

(1) はメタデータサーバの起動時にのみ実行され，(2)〜(7) が作成リクエストが発行されるたびに処理される．このような手続きにすることで，メタデータとストレージサーバの間の通信は N 回に 1 回になる．

5. 評価

PPOSS と PPMDS を組み合わせた分散ファイルシステムのプロトタイプ実装についてファイルの作成性能，およびアクセス性能の評価を行った．

5.1 評価環境

評価に利用した計算機の環境をメタデータサーバを実行した計算機については**表 1** に，クライアントを実行した計算機については**表 2** に，ストレージサーバを実行した計算機については**表 3** にそれぞれ示す．また，これらの計算機を含む評価環境全体の概略図を**図 4** に示す．図 4 に示されるように，メタデータサーバノードは 1 から 5 ノードまでノード数を変化させて用いた．また，クライアントノードも最大で 8 ノードを用いプロセス数を 1 から 128 まで変化させて用いた．データを保存するストレージサーバノードはノード数を 10 ノードに固定した．これらの計算機はすべて Infiniband によるネットワークで結ばれているが，通信は IPoIB(IP over Infiniband) 上で行った．

表 1 評価を行ったノードの性能（メタデータサーバ）

CPU	Intel(R) Xeon(R) CPU E5-2695 v2 2.40GHz (12 コア 12 スレッド) x2
メモリ	64GB
OS	CentOS 6
ネットワーク	Mellanox Technologies MT27500 4x FDR (56Gbps)

表 2 評価を行ったノードの性能（クライアント）

CPU	Intel(R) Xeon(R) CPU CPU E5-2665 2.40GHz (8 コア 8 スレッド) x2
メモリ	64GB
OS	CentOS 6
ネットワーク	Mellanox Technologies MT27500 4x FDR (56Gbps)

表 3 評価を行ったノードの性能（ストレージサーバ）

CPU	Intel(R) Xeon(R) E5620 CPU 2.40 GHz (4 コア 8 スレッド) x2
メモリ	24GB
OS	CentOS 6
ネットワーク	Mellanox Technologies MT26428 4x QDR (32Gbps)
ストレージ	Fusion-io ioDrive 160GB SLC
SDK	OpenNVM (Version 0.7)

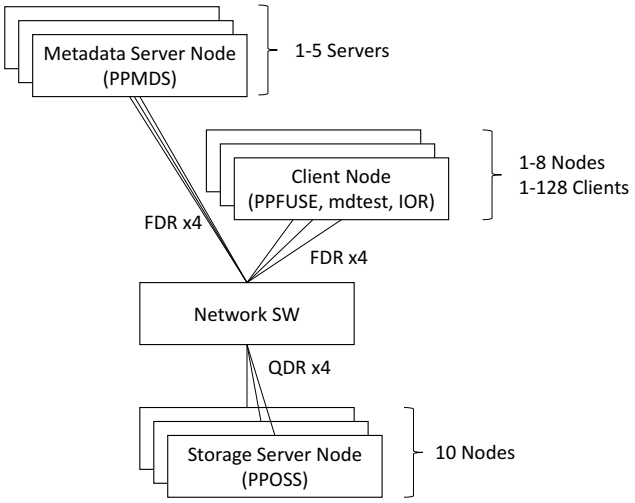


図 4 評価環境における各ノードの構成

5.2 PPMDS 単体性能の評価

まず，PPMDS 単体の性能について評価を行った．評価には mdtest HPC benchmark [10] のバージョン 1.9.3 を用

いた．mdtest は MPI を利用し，並列メタデータ操作性能を評価するベンチマークソフトウェアで，指定したディレクトリ直下に並列に指定された数のファイルとディレクトリの create 等を行う．今回の評価では，各クライントプロセスが異なるマウントポイントを利用できるように，mdtest に拡張を行った．

この評価では，mdtest 1 プロセス当たりそれぞれ 5,000 個のファイル，ディレクトリの作成を行った．クライアントプロセスの数は 1 から 128 まで変化させたため，128 プロセス時には合計 640,000 個のファイル，ディレクトリがそれぞれ生成される．

結果を**図 5** に示す．図 5 中の**図 5(a)** がディレクトリ作成性能，**図 5(b)** がファイル作成性能をそれぞれ示す．図 5 中のそれぞれのグラフにおいて，縦軸は 1 秒辺りに作成することのできるファイル・ディレクトリの数であり，数値が大きいほど高い性能であることを示している．横軸はクライアントのプロセス数であり，1 から 128 まで変化させて計測した．

図 5 のうちディレクトリ作成性能を示す図 5(a) においては，メタデータサーバのノード数が 1 ノードで，クライアント数が 16 の場合に 50,841.7 ops/s と最も高い性能を示しており，メタデータサーバが 2 ノード以上となった場合では少ないクライアント数の場合はノード数が少ないほうが，クライアント数が多い場合にはノード数が多いほうが高い性能を示している．1 ノードの場合に最も高い性能を示したのは，ディレクトリの作成時はメタデータサーバがディレクトリのエントリを分散させるサーバのリストを関係するすべてのサーバに対して作成するためてある．一方で，ファイル作成性能を示す図 5(b) においては，ノード数の増加にともなって性能がスケールしており，メタデータサーバのノード数が 5 ノードでクライアントが 128 プロセスだった場合に 142,564.4 ops/s となっている．また，ディレクトリ作成時の性能に比べてファイル作成時の性能が高いが，ファイル作成時にはディレクトリのメタデータのひとつであるディレクトリエントリの分散させるサーバのリストが存在せず，ディレクトリ作成時にあったそのリストを関係する全てのサーバで作成するコストがファイル作成時にはないためである．

また，比較のために PPMDS と同様にメタデータをキーバリューストアで管理を行う IndexFS [3] についてもあわせて評価を行った．IndexFS は，これまでの多くの分散ファイルシステムがデータアクセス性能の高性能化を測ることに注目しており，メタデータアクセスの高性能化が欠如していたことから，PVFS [11] や Lustre [12]，HDFS [13] など既存の分散ファイルシステムのメタデータアクセス性能を向上させるためのメタデータサーバのミドルウェアである．メタデータを管理するキーバリューストアではファイルの親ディレクトリの inode 番号とファイル名のハッシュ値のペアをキーとしてメタデータを管理している．文献 [3] によると IndexFS は，メタデータサーバの台数が 128 ノードまで性能がリニアにスケールし，メタデータアクセスが重要なワークロードでは，ベースとなる分散ファイルシステムに比べ大幅に性能を向上させる．これらの理由から IndexFS と性能を比較する．

IndexFS におけるファイルの作成性能の結果を**図 6** に示す．図 6 中の**図 6(a)** がディレクトリ作成性能，**図 6(b)** がファイル作成性能をそれぞれ示す．

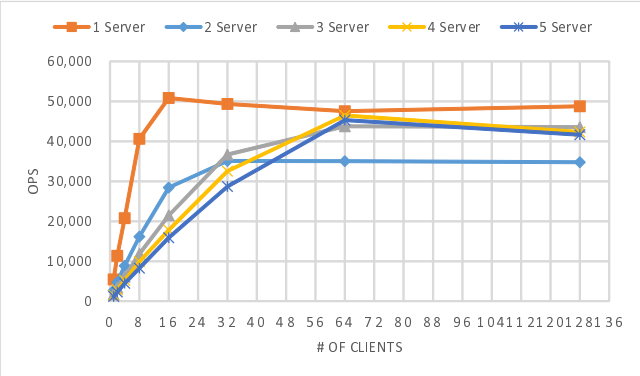
図 6 のうちディレクトリ作成性能を示す図 6(a) においても，PPMDS と同様にメタデータサーバのノード数が 1 ノードで，クライアント数が 16 の際に 18,987.9 ops/s と最も高い性能を示している．これはメタデータサーバのノード数が 2 ノード以上になった場合に，メタデータサーバ間の通信が発生するためと考えられる．一方で，ファイル作成性能を示す図 5(b) においては，メタデータサーバのノード数が 2 ノードと 3 ノード，および 4 ノードと 5 ノードの場合において性能差がほぼない．これは，ファイルが増えた場合のディレクトリ分割を 2 のべき乗になるようにしているためだと考えられる．また，PPMDS と IndexFS のファイル作成性能を比較するとメタデータサーバのノード数が 5 でクライアントが 128 プロセスの場合に，PPMDS は IndexFS の 2.60 倍の性能となっている．PPMDS においてファイルの作成時には，親ディレクトリの分散先サーバーリストの読み込みと格納先のサーバーに対してメタデータの格納に伴う書き込みが行われるが，これらの読み込みと書き込みは，多数のファイル作成リクエストが発行されても衝突しないため，処理に伴う同期待ちが発生しないため，IndexFS に比べて高い性能を示したものと考えられる．

5.3 PPMDS/PPOSS の作成性能の評価

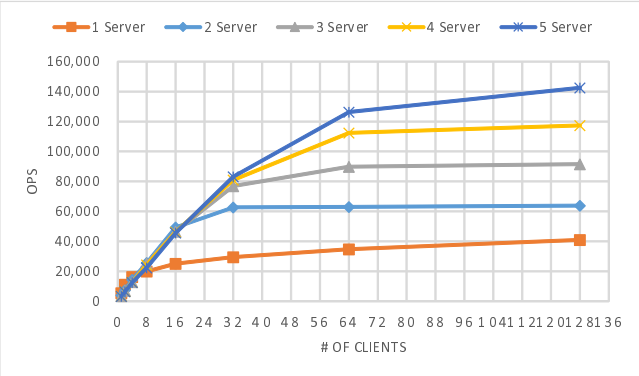
次に PPMDS に PPOSS を組み合わせた場合のファイル，ディレクトリの作成性能について評価を行った．ここでも評価には mdtest HPC benchmark [10] のバージョン 1.9.3 を用いた．先の評価と同様に，mdtest 1 プロセス当たりそれぞれ 5,000 個のファイル，ディレクトリの作成を行う．クライアントプロセスの数は 1 から 128 まで変化させたため，128 プロセス時には合計 640,000 個のファイル，ディレクトリがそれぞれ生成される．

結果を**図 7** に示す．図 7 中の**図 7(a)** がディレクトリ作成性能，**図 7(b)** がファイル作成性能をそれぞれ示す．図 7 中のそれぞれのグラフにおいて，縦軸は 1 秒辺りに作成することのできるファイル・ディレクトリの数であり，数値が大きいほど高い性能であることを示している．横軸はクライアントのプロセス数であり，1 から 128 まで変化させて計測した．

図 7 のうちディレクトリ作成性能を示す図 7(a) は，PP-

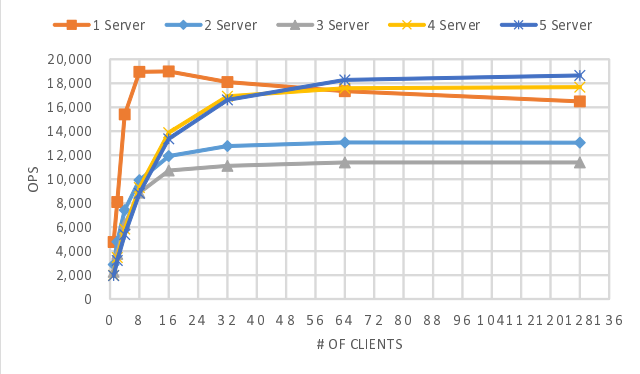


(a) ディレクトリ作成性能

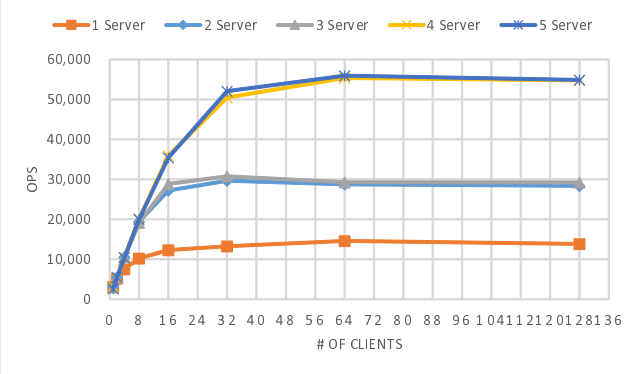


(b) ファイル作成性能

図 5 PPMDS の単一ディレクトリに対するメタデータ操作の性能

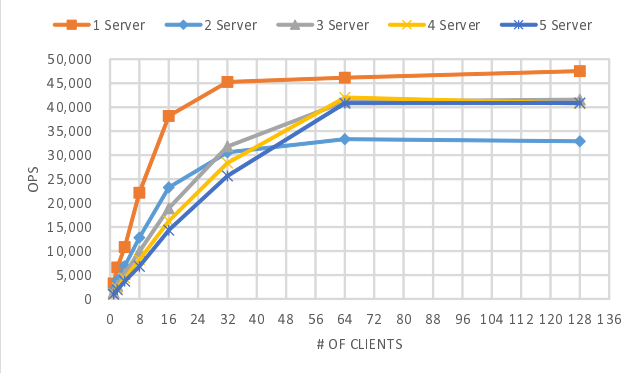


(a) ディレクトリ作成性能

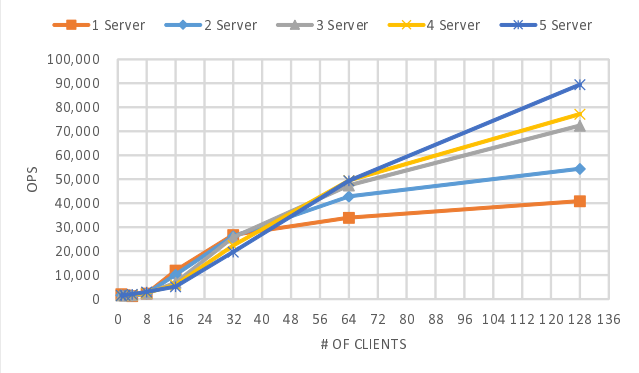


(b) ファイル作成性能

図 6 IndexFS の単一ディレクトリに対するメタデータ操作の性能



(a) ディレクトリ作成性能



(b) ファイル作成性能

図 7 PPOSS と組み合わせた PPMDS の単一ディレクトリに対するメタデータ操作の性能

MDS 単体のディレクトリ作成性能を示す図 5(a) と異なり，メタデータサーバが 1 ノードでクライアントが 128 プロセスの場合に最も高い性能である 48,784.3 ops/s を示した．PPMDS 単体時に 1 ノードで最も高い性能を示したクライアントが 16 プロセスの場合では，単体時の性能に比べて 75.0 %の性能となっている．ディレクトリ作成時の処理については，PPOSS と組み合わせた場合も単体時と変更しておらず，このような結果になった原因を現在調査中である．一方で，図 7(a) が示すファイル作成性能を比較すると，メタデータサーバが 5 ノードでクライアント数が

128 プロセスの場合に，PPMDS 単体の場合に比べて 62.8 %となり，クライアント数が少ない 16 プロセスの場合では 11.5 %となった．これは，ファイル作成のたびに PPOSS でオブジェクトの発行を行い，それにより処理がブロックされているためと考えられる．

5.4 Bulk Creation の影響

次に PPMDS に PPOSS を組み合わせた際のファイル作成時のストレージサーバとの通信回数を減らす Bulk Creation の影響について評価を行った．この評価では，PPOSS と通信を行うのは 64 回に 1 回で，その際に 64 個のオブジェクトを作成するように設定した．ここでも評価には mdtest HPC benchmark [10] のバージョン 1.9.3 を用いた．先の評価と同様に，mdtest 1 プロセス当たりそれぞれ 5,000 個のファイル，ディレクトリの作成を行う．クライアントプロセスの数は 1 から 128 まで変化させたため，128 プロセス時には合計 640,000 個のファイル，ディレクトリがそれぞれ生成される．

結果を**図 8** に示す．図 8 中の**図 8(a)** がディレクトリ作成性能，**図 8(b)** がファイル作成性能をそれぞれ示す．図 8 中のそれぞれのグラフにおいて，縦軸は 1 秒辺りに作成することのできるファイル・ディレクトリの数であり，数値が大きいほど高い性能であることを示している．横軸はクライアントのプロセス数であり，1 から 128 まで変化させて計測した．

図 8 のうちディレクトリ作成性能を示す図 8(a) も，PPMDS 単体のディレクトリ作成性能を示す図 5(a) と異なり，メタデータサーバが 1 ノードでクライアントが 128 プロセスの場合に最も高い性能である 47,064.7 ops/s を示した．PPMDS 単体時に 1 ノードで最も高い性能を示したクライアントが 16 プロセスの場合では，単体時の性能に比べて 75.2%の性能となっている．ディレクトリ作成時の処理については，最適化の有無を含め PPOSS と組み合わせた場合も単体時と変更しておらず，このような結果になった原因を現在調査中である．一方で，図 8(a) が示すファイル作成性能を比較すると，メタデータサーバが 5 ノードでクライアント数が 128 プロセスの場合に，PPMDS 単体の場合に比べて 83.7%となり，クライアント数が少ない 16 プロセスの場合では 65.2%となり，最適化を適用しない場合に比べて，クライアントが 128 プロセスの場合に 1.34 倍，16 プロセスの場合に 5.69 倍にそれぞれなった．これは，ファイル作成のたびに PPOSS でオブジェクトの発行を行うのではなく，64 回に 1 回まとめてオブジェクトの作成をおこなうようにしたためと考えられる．

5.5 PPMDS/PPOSS の書き込み性能の評価

この評価では，PPMDS を PPOSS を組み合わせた分散ファイルシステムのプロトタイプ実装の書き込み性能について評価を行う．

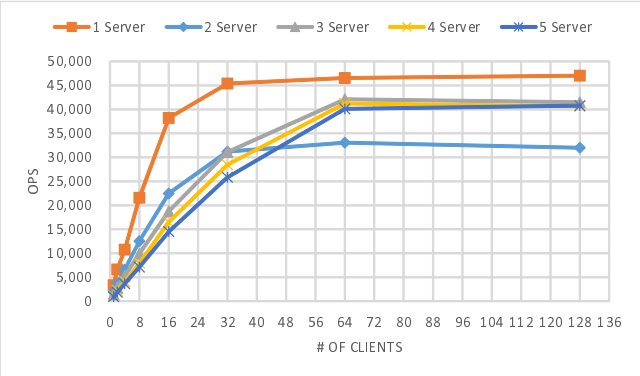
この評価の前に予備評価として PPOSS の書き込み性能について評価を行った．この予備評価では，複数のクライアントプロセスからから単一の PPOSS サーバに対して同時に書き込みを行う．各クライアントプロセスは 128 MiB の書き込みを行い，クライアントプロセス数とブロックサイズを変化させ，それぞれ 5 回の評価を行い平均を求めた．クライアントプロセス数を 1 から 128 まで変化させたため，128 プロセス時には合計 16GiB のデータが書き込まれる．

結果を**図 9** に示す．縦軸は 1 秒辺りに書き込むことの出来たデータサイズを示しており，数値が高いほど高い性能を示す．横軸は各評価の際の書き込み時のブロックサイズを示している．図中の各線は，クライアントプロセス数である．クライアントプロセスの増加にしたがって性能が向上しているが，32 プロセス以上の際に性能が限界となっている．これはストレージデバイスの性能によるものである．128 プロセスの結果をベースラインとして PPMDS と PPOSS を組み合わせた分散ファイルシステムのプロトタイプ実装の評価を行う．

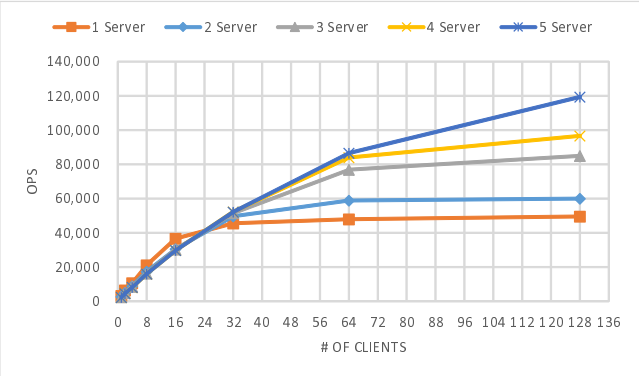
次に PPMDS を PPOSS を組み合わせた分散ファイルシステムのプロトタイプ実装の書き込み性能について評価を行った．評価には IOR HPC Benchmark [14] のバージョン 2.10.3 を用いた．IOR は MPI を利用し，並列ファイルアクセス性能を評価するベンチマークソフトウェアで，指定したディレクトリ直下で，各プロセスが並列に指定されたブロックサイズで指定されたファイルサイズのファイルの読み書きを行う．この評価では書き込み性能について，各プロセス数ごとに 128 MiB のファイルをブロックサイズを変化させ，5 回評価を取り平均をもとめた．また，メタデータサーバのノード数の変化による影響について評価するために，ノード数も 1 から 5 まで変化させ評価を取った．IOR のプロセス数は 1 から 128 まで変化させた．ファイルシステムのクライアントである ppfuse を 8 台のノードで合わせて 128 個のマウントポイントでマウントを行い，各 IOR のプロセスはそれぞれ異なるマウントポイントで評価を行う．

結果を**図 10** に示す．図 10 中には**図 10(a)** から**図 10(e)** までの 5 つのグラフがある．これらはそれぞれの評価の際のメタデータサーバの数である．各グラフ中の縦軸は 1 秒辺りに書き込むことの出来たデータサイズを示しており，数値が高いほど高い性能を示す．横軸は各評価の際の書き込み時のブロックサイズを示している．各グラフ中の各線は，IOR のプロセス数である．

図 10(a) から図 10(e) の各グラフを比較するとメタデータサーバのノード数を変化させた場合の性能差は大きくない．これはファイルへの書き込み時のメタデータサーバへのアクセスが，open(), close() の際にしか発生しないため



(a) ディレクトリ作成性能



(b) ファイル作成性能

図 8 PPMDS の単一ディレクトリに対するメタデータ操作の性能

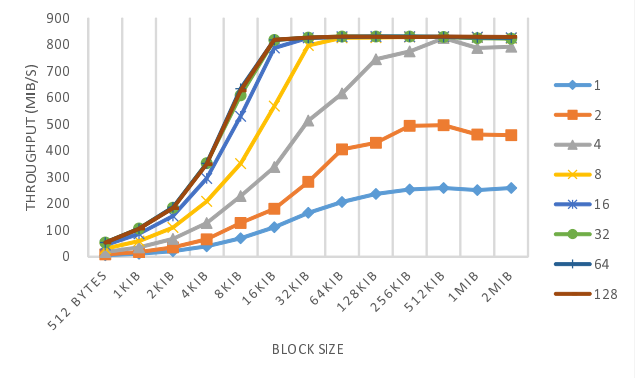


図 9 複数のクライアントから PPOSS への書き込み性能に関する予備評価の結果

で，各 write() の際には open() 時に取得したファイルに対応するオブジェクトのオブジェクト ID を用いてストレージサーバとのやり取りしか発生しないためである．

また，全体の最大性能は，メタデータサーバ数が 2，クライアントプロセス数が 128，ブロックサイズが 1 MiB の時に 6,158.22 MiB/s となっている．この条件では各ストレージサーバに平均して，615.8 MiB/s のアクセスが行われたことになる．図 9 に示される PPOSS 単一ノードの性能ではブロックサイズが 1 MiB の際の逐次書き込み性能はクライアントプロセス数が 128 の場合で 829.2 MiB/s であり，PPOSS 単体の評価に比べて 25.7 %の性能劣化となっている．

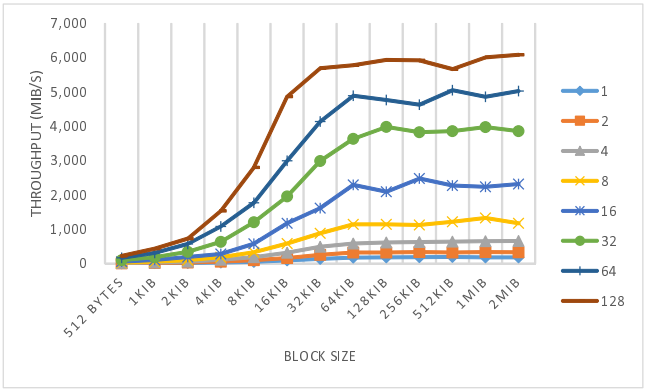
この原因を調査するために，PPMDS1 ノード，クライアントを 128 プロセスと固定し，PPOSS のノード数を変化させることで評価を行った．この評価においても IOR HPC Benchmark [14] のバージョン 2.10.3 を用い，各プロセスごとに 128 MiB のファイルをブロックサイズを変化させ，5 回評価を取り平均をもとめた．

結果を**図 11** に示す．**図 11(a)** は書き込み性能評価の結果を示している．縦軸は 1 秒辺りに書き込むことの出来たデータサイズを示しており，数値が高いほど高い性能を示す．横軸は各評価の際の書き込み時のブロックサイズを示している．各線は，PPOSS のノード数である．**図 11(b)** は，PPOSS 単一ノードに対する書き込み性能の予備評価における 128 プロセスの結果を 1 とした際の，PPOSS1 ノードあたりのプロトタイプ実装の性能の比率を示している．

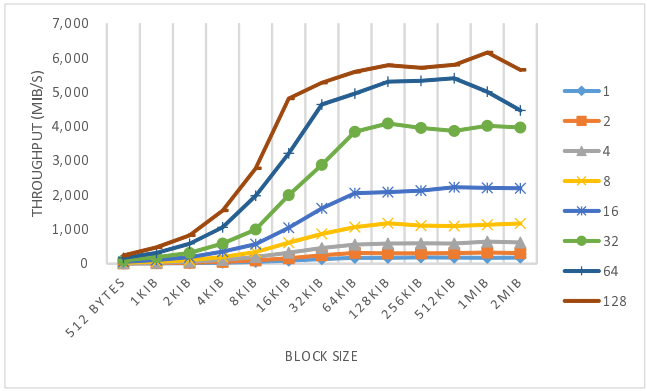
図 11(a) からは PPMDS 全体の書き込み性能が，PPOSS のノード数によってスケールすることが示されている．一方で，図 11(b) からは各ノードあたりの性能が，ブロックサイズが 64 KiB 以上の場合においては，PPOSS のノード数が増えるにしたがって性能が低下している．これは，ファイルが PPOSS の各ノードに均等に分散しておらず，全体でのクライアントプロセスの数が少ないため，一部の PPOSS にアクセスするプロセス数が少なくなったためと考えられる．また，ブロックサイズが低い場合に，性能が低下している．PPMDS のクライアント ppfuse は FUSE を利用したファイルシステムであり，アクセス時にはメモリコピーなどによるオーバーヘッドあり，そのためと考えられる．

6. 関連研究

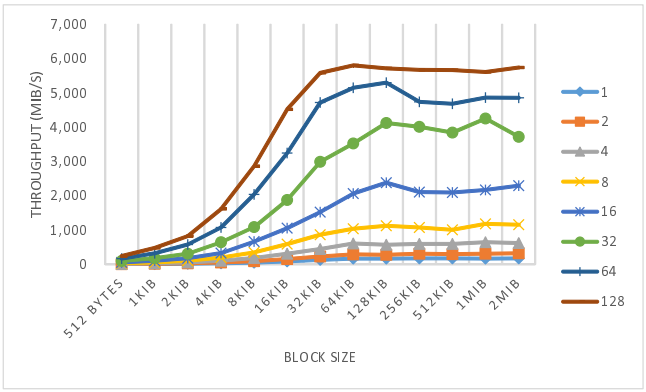
高性能計算機のための分散ファイルシステムにに関する研究は現在も活発に行われているため多岐にわたる．評価時に比較で用いた IndexFS [3] はそれらの中でも，メタデータをキーバリューストアで管理を行う点で PPMDS と共通している．IndexFS は，PVFS [11] や Lustre [12]，HDFS [13] など既存の分散ファイルシステムに，メタデータや小さなファイルの操作を高性能でスケールさせるメタデータサーバのミドルウェアである．GIGA+ [2] で用いられていたインクリメンタルなディレクトリ分割を IndexFS においても用いており，ディレクトリ内のファイル数がしきい値を超えた際にディレクトリを分割する．また，これは既存の分散ファイルシステムのメタデータ操作性能を



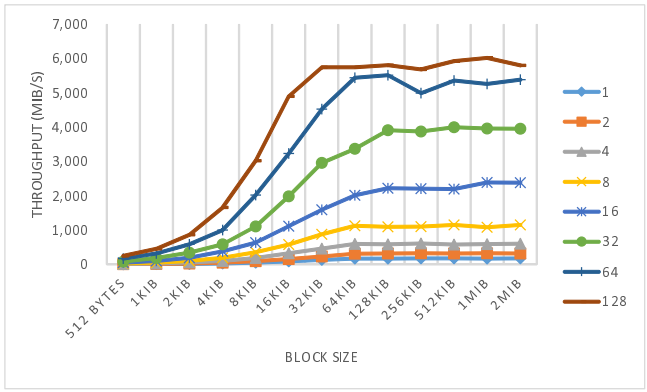
(a) 1 Server



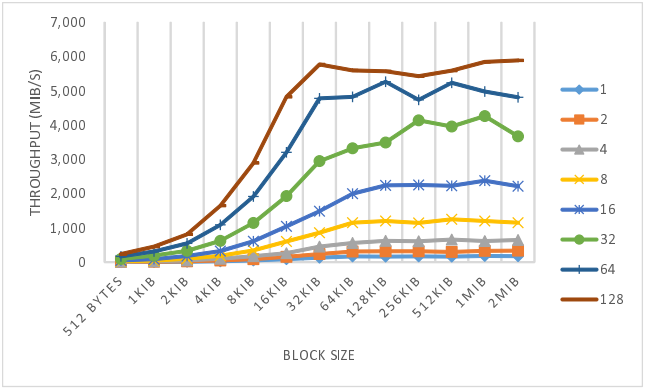
(b) 2 Servers



(c) 3 Servers



(d) 4 Servers



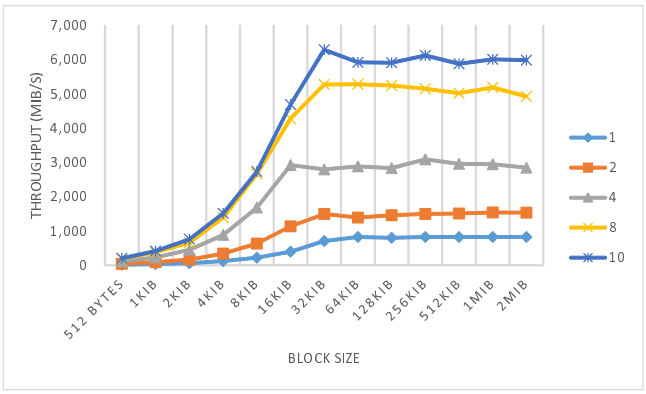
(e) 5 Servers

図 10 メタデータサーバの各ノード数におけるアクセス性能評価

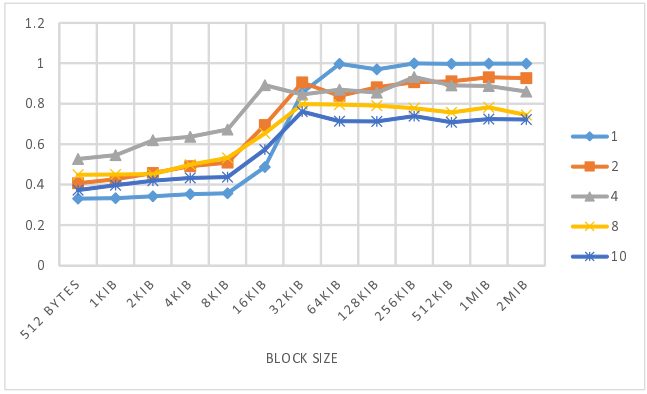
向上させる研究であり，その点においてオブジェクトストレージと組み合わせる本研究と異なる．

7. まとめ

本稿では，ハイパフォーマンスコンピューティングの分野で広く利用されている分散ファイルシステムのボトルネックとなっているメタデータ処理について高性能化を図った PPMDS について，低レイテンシのネットワークを用いた環境において性能を評価し，既存の分散ファイルシステムのメタデータサーバである IndexFS と比較した．また，ファイルを読み書きを行うために OpenNVM を用いた高性能分散ファイルシステムのためのローカルオブジェクトストレージをバックエンドとして利用するストレージサーバ PPOSS を提案し，PPMDS と組み合わせた分散



(a) PPOSS のノード数を変化させた場合の書き込み性能評価



(b) PPOSS 単一ノードの性能を 1 とした場合の比率

図 11 PPOSS のノード数を変化させた場合の書き込み性能評価

ファイルシステムを設計，プロトタイプ実装を行い，その際の性能を評価し，PPMDS 単体の性能と比較した．さらに，PPOSS と組み合わせた場合の性能劣化を最小限に抑えるために，複数のオブジェクトを予めまとめて作成しておく最適化手法 Bulk Creation を提案し，性能を評価した．

評価では，PPMDS 単体の性能が IndexFS に比べて，メタデータサーバが 5 ノード，クライアントが 8 ノード 128 プロセスからのファイル作成処理において，2.60 倍の性能となることを示した．また，ローカルオブジェクトストレージと組み合わせた場合には，PPMDS 単体の性能に比べて，128 プロセスのクライアントからのアクセス時に 62.8 %，16 プロセスの場合に 11.5 %の性能となったが，Bulk Creation を適用させることで，それぞれ 83.7 %および 65.2 %の性能となり，適用前に比べ，それぞれ 1.34 倍，5.69 倍となった．

アクセス性能評価では，アクセス性能がメタデータサーバのノード数によらないことを示した．一方で，ローカルオブジェクトストレージ単体の性能に比べて，PPOSS が 10 ノードでブロックサイズが 2MiB の場合に 72.2 %の性能となった．この性能を向上させることが今後の課題として考えられる．また，クライアントプロセスの数をさらに増やした場合の評価をシミュレーションで行うことも今後の課題である．

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] Bent, J., Gibson, G., Grider, G., McClelland, B., Nowoczynski, P., Nunez, J., Polte, M. and Wingate, M.: PLFS: A Checkpoint Filesystem for Parallel Applications, *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*, pp. 21:1–21:12 (2009).

[2] Patil, S. and Gibson, G. A.: Scale and Concurrency of GIGA+: File System Directories with Millions of Files., *FAST*, Vol. 11, pp. 177–190 (2011).

[3] Ren, K., Zheng, Q., Patil, S. and Gibson, G.: IndexFS: scaling file system metadata performance with stateless caching and bulk insertion, *SC14: International Conference for High Performance Computing, Networking, Storage and Analysis*, IEEE, pp. 237–248 (2014).

[4] 平賀弘平，建部修見：ノンブロッキングトランザクションに基づく分散ファイルシステムのための分散メタデータサーバの設計と実装，研究報告ハイパフォーマンスコンピューティング (HPC)， Vol. 2012, No. 28, pp. 1–9 (2012).

[5] 桐井祐樹，建部修見，Robert, R.：CODES/ROSS による分散ファイルシステムのための分散メタデータサーバ PPMDS の評価，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, No. 7, pp. 1–9 (2015).

[6] Fusion-io: NVM Primitives Library, http://opennvm.github.io/nvm-primitives-documents/ (2014).

[7] Takatsu, F., Hiraga, K. and Tatebe, O.: Design of object storage using OpenNVM for high-performance distributed file system, 情報処理学会論文誌コンピューティングシステム（ACS)， Vol. 9, No. 2 (2016). (to appear)

[8] : msgpack-rpc, https://github.com/msgpack/msgpack-rpc.

[9] Lamping, J. and Veach, E.: A fast, minimal memory, consistent hash algorithm, *arXiv preprint arXiv:1406.2294* (2014).

[10] : mdtest HPC Benchmark, http://mdtest.sourceforge.net/.

[11] Ross, R. B., Thakur, R. et al.: PVFS: A parallel file system for Linux clusters, *Proceedings of the 4th annual Linux Showcase and Conference*, pp. 391–430 (2000).

[12] Koutoupis, P.: The lustre distributed filesystem, *Linux J.*, Vol. 2011, No. 210 (2011).

[13] 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, pp. 1–10 (2010).

[14] : IOR HPC Benchmark , https://sourceforge.net/projects/ior-sio/.