Vol.2010-HPC-126 No.4
2010/8/3

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

三上俊輔[†1]　太田一樹[†2]　建部修見[†1]

MapReduce のための分散ファイルシステムとして Google File System や HDFS (Hadoop Distributed File System) が使われているが，それらのファイルシステムは特定 API によるストリーミングアクセスを前提とし，POSIX の要件を緩和している．このため MapReduce 以外のプログラムからそれらのファイルシステムを直接使用することが困難であり，MapReduce 処理をするためにそれらのファイルシステムにインポートして，他のプログラムで利用するために結果をエクスポートするなど，コピーが必要なことが多い．この問題を解決するために本研究では HDFS の代わりに POSIX 準拠の API を持った広域分散ファイルシステム Gfarm を使うことを提案し，Gfarm 上で MapReduce 処理を可能にするための Hadoop-Gfarm プラグインを設計し評価する．マイクロベンチマークにおいて，Gfarm は HDFS より約 30%高い書き込み性能を示し，読み込みはほぼ同程度の性能であった．また，grep，sort などの単純な MapReduce アプリケーションにおいて Gfarm と HDFS は同程度の性能であった．提案手法を使えば性能を低下させることなく，POSIX 準拠の API を使え，無駄なデータの移動やコピーを減らすことができる．

Data Intensive distributed computing using MapReduce on Gfarm file system

Shunsuke Mikami,[†1] Ohta Kazuki[†2]
and Osamu Tatebe[†1]

Distributed filesystems that have been designed to use MapReduce, such as Google file system and HDFS (Hadoop Distributed File System), suppose streaming access and they relax a couple of POSIX requirements. This makes it difficult for programs other than MapReduce to access these filesystems. It is often need to import files to these file system for MapReduce and export for other programs. In order to solve this problem, we are proposing using Gfarm, a globally distributed file system, to bridge the gap. We designed and implemented Hadoop-Gfarm plugin to allow Hadoop MapReduce applications access to the files stored on the Gfarm file system. Micro benchmarks show that 30% higher performance than HDFS in write throughput, and similar performance in read throughput. The grep and sort applications show similar performance. Overall using the Gfarm file system in place of HDFS allows users to use a POSIX-compliant API and reduces redundant copy without sacrificing performance.

1. はじめに

近年，データが大規模化しそのデータを妥当な時間で処理するためには数百台のマシンで分散処理をする必要がある．効率よく分散処理するには並列計算，エラー処理，データ分散など複雑な処理が必要となる．MapReduce[7] のモデルを使えば，そういった複雑な処理を MapReduce が行い，プログラマは実際の計算のための処理だけを記述して分散処理を実行できる．

現在 MapReduce は様々な用途に使われており，当初 Google で利用されてきた用途，転置インデックスの作成やログ解析などだけではなく，ゲノム解析など科学研究の分野でも使われるようになってきている．[5]

MapReduce は大規模データを処理するため，データの入出力に分散ファイルシステムを利用する．Google では Google File System[12] を MapReduce の入出力として使用している．MapReduce のオープンソース実装である Hadoop MapReduce では，Google File System のオープンソース実装である HDFS(Hadoop Distributed File System)[2] がデータの入出力に利用されている．しかし，これらの現在 MapReduce に使われている分散ファイルシステムは汎用的なファイルシステムではなく MapReduce プログラムや特定の API からのアクセスを基本としている．また POSIX の要件を弱めており，ファイルの任意の位置の修正や複数ライターからの単一ファイルへの書き込みは出来ない．

MapReduce が強力な分散処理フレームワークであるが，アプリケーションによっては MapReduce の処理が向いていない場合や，既存のソフトウェアを使いたい場合などもある．また，MapReduce によって処理した結果を他のデータベースやソフトウェアに渡すこともある．このため，元データは他のファイルシステムにおいて，MapReduce プログラム使用

†1 筑波大学
University of Tsukuba
†2 Preferred Infrastructure, Inc.

の際に入力データを HDFS へインポートしたり，計算結果をエクスポートする作業が必要となる．これは利便性が悪いだけではなく，ストレージの無駄遣いとなる．本研究では一つのファイルシステムでデータを管理し，様々なアプリケーションを実行するために，分散ファイルシステム Gfarm[9] を使用することを提案する．Gfarm は POSIX 準拠の API を持った広域分散ファイルシステムである．Gfarm はその並列入出力 API だけでなく FUSE[14] を使って Linux クライアントからマウント可能で，既存のプログラムから完全に透過的にアクセスできる．また，MPI-IO による MPI プログラムからのアクセスも可能である．Gfarm を HDFS の代わりに使うことで，無駄なデータの移動やコピーを減らし，ストレージの管理を Gfarm に一元化することができる．

本研究では Hadoop MapReduce から直接 Gfarm 上のファイルへアクセス可能にするための Hadoop-Gfarm プラグインの設計し，実装した．本稿では，実装したプラグインの設計について述べ，実環境において評価を行なった．

本稿の構成を次に示す．まずは研究の背景として 2 章では MapReudce の説明をし，3 章で HDFS と Gfarm の特徴について述べる．4 章では本研究において実装した Hadoop-Gfarm プラグインの説明をし，5 章で性能の評価を行う．6 章で関連研究と比較を行い，7 章で全体をまとめを行う．

2. MapReduce

MapReduce[7] とは Google によって 2004 年に提案された大規模データを並列分散処理するためのフレームワークである．MapReduce のプログラミングモデルでは処理を Map, Shuffle，Reduce の各フェーズに分解する．Map では key/value のペアを入力データとして受け取り，ユーザー定義の map 関数を実行し，中間 key/value を生成する．Shuffule フェーズでは，中間 key/value を受け取り同じ key に対して value のリストを生成し，ソートして Reduce に渡す．Reduce フェーズでは，key と対応する value のリストを受け取り，ユーザー定義の reduce 処理を行い，最終出力となる key/value データを生成する．各 Map タスク，Reduce タスクは副作用がなく，完全に並列して計算可能である．ジョブマスタがこの一連の MapReduce ジョブを管理し，ワーカーが実際の Map タスクと Reduce タスクを実行する．ジョブを実行すると，ジョブマスタが入力データを自動的に 16MB から 64MB 程度に分割し，タスクを生成し，各ワーカノードに割り当てる．タスクの割り当ての際に入力データの近くに割り当てることによってネットワークのデータ転送量を抑え，効率的な I/O を実現している．

2.1 Hadoop

Google は Google file system, MapReduce, BigTable などの様々な分散処理技術を発表したが，それらの技術をオープンソースで開発を進めているのが Apache Hadoop プロジェクトである．Hadoop には様々なサブプロジェクトがあり，Google の Google file system, MapReduce, Big table[6] にそれぞれ HDSF, Hadoop MapReduce, HBase[1] が対応する．また，最近では周辺のサブプロジェクトの開発も進んでおり，MapRedue のプログラムを書くよりも簡単にデータ処理を記述できるメタ言語実行環境として Pig[4] や Hive[3] が注目されている．Pig や Hive はそれぞれ独自の記法で記述したプログラムを MapReduce のジョブに変換して実行する．MapReduce プログラムを直接記述するよりも自由度や性能面で劣るが，プログラマの負担を減らす事ができるという大きなメリットを持つ．

3. 分散ファイルシステム

この章では HDFS と Gfarm のアーキテクチャについて述べる．いづれもマスタースレーブ型の分散ファイルシステムで，ファイルシステムのメタデータを管理するマスターサーバと，実際のデータへのアクセスを提供するスレーブサーバを持つ．データの書き込み，読み込みなどはスレーブサーバへ直接行われるため，ノード数に対してスケーラブルな性能を持つ．また，複製を作成し分散配置することにより信頼性を確保することができる．次節では各ファイルシステムの特徴について述べる．

3.1 HDFS

HDFS は大きいファイルのストリーミング型のデータアクセスに特化しており，書き込みを一度だけ行い，読み込みを何度も行うデータ処理のパターンに向いている．ファイルはブロック（デフォルトでは 64MB）に分割され，各スレーブノードに分散して配置される．これらの特徴は MapReduce プログラムを実行することを想定されており，MapReduce のプログラムには適している．しかし，MapReduce プログラムでは必要にならないが，他のアプリケーションでは必要になる機能に関して足りない部分がある．HDFS は複数のライターからの書き込みやファイルの任意の位置の修正などをサポートしていない．FUSE でマウントすることは可能だが，マウントしたとしてもこれらの操作はサポートしておらず，既存のソフトウェアから HDFS 上のファイルを処理することは出来ない．以上の理由により，汎用的なファイルシステムとして用いることは難しい．

3.2 Gfarm file system

Gfarm はより広範囲なアプリケーションへの適応，日常的な利用を可能にすることを目

標として，NFS の代用となりうる，POSIX 標準 API をサポートする頑強で信頼性のある広域分散ファイルシステムである．HDFS とは違って，一つのファイルはブロック分割されず一台のスレーブノードに格納される．MapReduce プログラムにおいてスケーラブルな性能を達成するには，ファイルがノードに分散されて配置されている必要がある．ファイルを分割することがユーザーの負担になるという意見もあるだろうが，実際は大きいデータセットは複数ファイルに分かれてることが多いため，ユーザの負担は小さい．FUSE で通常のファイルシステムへマウントすることが可能で，既存のソフトウェアから完全に透過に利用できる．また，MPI-IO による MPI プログラムからの並列 I/O もサポートしている．

4. Hadoop-Gfarm プラグイン

Hadoop MapReduce プログラムから Gfarm 上のファイルにアクセスするには FUSE を使ってアクセスすることも可能である．しかし，その場合，FUSE のオーバーヘッドの影響を受けることと，Hadoop MapReduce によるデータの位置を考慮したスケジューリングができないため，本研究において Hadoop から Gfarm 上のファイルへ直接アクセスするための Hadaop-Gfarm プラグインを開発した．Hadoop は元々バックエンドの分散ファイルシステムを HDFS 以外でも利用できるように抽象化された FileSystem API (org.apache.hadoop.fs.FileSystem) を備えており，Hadoop-Gfarm プラグインでもこのファイルシステム API を継承して Gfarm 用に実装している．同様の実装として KFS[8] や S3[13] などもこの API を使用して Hadoop MapReduce プログラムから直接アクセス可能となっている．**図 1** に Hadoop-Gfarm のソフトウェアスタックを示す．Hadoop-Gfarm プラグインは JNI(Java Native Interface) の shim layer であり，Hadoop や Gfarm のソースコードに変更を加えることなく Hadoop MapReduce のアプリケーションを実行できる．また Hive や Pig は Hadoop MapReduce の上位のレイヤーなので，これらのプログラムも実行可能である．

この FileSystem API が含む関数は，通常の POSIX ライクの *read()*, *write()*, *open()*, *mkdir()* などのファイルシステム操作である．さらに Hadoop がデータの位置を知るための関数 *getFileBlockLocations()* も含んでおり，HDFS 以外のファイルシステムでもこの関数を実装することで Hadoop のジョブを管理するマスターサーバがデータが実際に保存されているホスト名を知ることができ，データの位置を考慮したスケジューリングが可能となる．**図 2** に Hadoop と Gfarm の関係を示す．スレーブには Hadoop の計算サーバと Gfarm のストレージサーバ両方を起動する．ジョブが実行されると，Haodop のジョブマスタが *getFileBlockLocations()* 関数を使用して，Gfarm のメタデータサーバに各タスクのデータの位置を取得し，データに近い計算サーバに優先してタスクを割り当てる．Gfarm はファイルがブロック分割されてないが，MapReduce プログラムを実行する際はこちらが設定ファイルで指定したサイズで入力データの分割も自動的に行われ，Map タスクの粒度がファイルサイズと同じになることはない．スケジューリングの方法は HDFS の場合とまったく同じ方法で行われる．

Hadoop-Gfarm ではこれらの関数を JNI(Java Native Interface) を使って実装し，Hadoop から Gfarm のクライアントライブラリを直接利用している．Java と C++のデータのやり取りの際にはコピーが発生しないように直接バッファに参照されるメモリ領域の開始アドレスを渡している．まず，JNI の shim layer に関するオーバーヘッドの評価を性能評価を実施した．**表 1** に使用したマシンの性能を示す．結果を**図 3** に示す．read と write が Gfarm の C 言語の API を直接使用した場合の結果で，read(JNI) と write(JNI) はその API を Java のコードから JNI を通して使用した場合の結果である．書き込み読み込み共に大きな性能の差は見られず，プラグインを実装するにあたって JNI のオーバーヘッドは小さいことが確認できた．

5. 性 能 評 価

HDFS と Gfarm の比較をするために，マイクロベンチマークによる読み込み，書き込みの性能評価，grep と sort の MapReduce アプリケーションによる評価を行った．表 1 に実験に使用したマシンの性能を示す．今回の評価では 15 ノードまで使用し，15 ノードの際は一つはマスターとスレーブを兼ねており，それ以外の場合は別にマスターノードを使用している．また，どちらのファイルシステムも複製の作成が可能ではあるが今回は全て複製なしで評価を行った．

5.1 書き込み性能

Hadoop 付属の TestDFSIO ベンチマークを使って書き込みの性能を計測した．各ノード

表 1 マシン性能

CPU	2.33GHz Quadcore Xeon E5410 (2 sockets)
Memmory	32GB
OS	Linux 2.6.18-6-amd64 SMP
Disk	Hitachi HUA72101 1TB
Hadoop Version	0.20.2
Gfarm Version	2.30

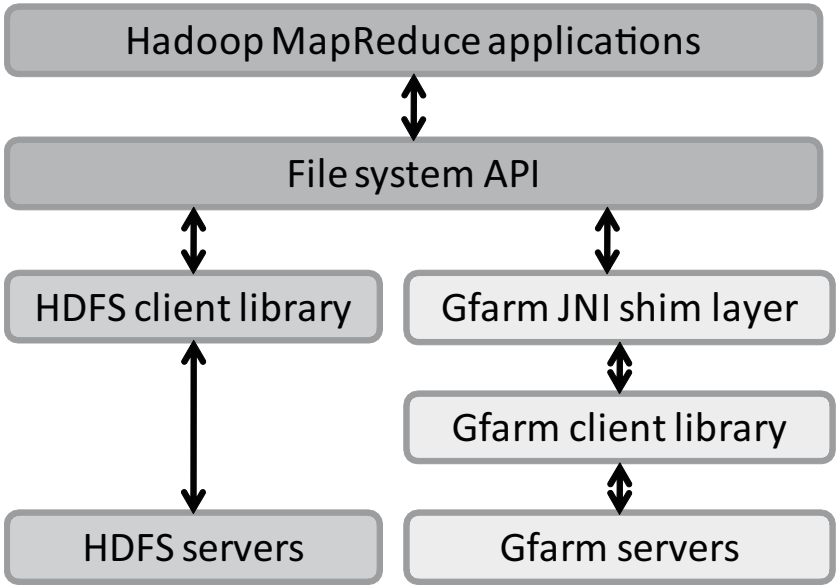

図 1 Hadoop-Gfarm ソフトウェアスタック

毎異なるファイルを同時に書き込んで，各ノード書き込みに要した時間を合計してスループットを計算している．メモリサイズが 32GB あるため，それよりも大きいサイズである 50GB を各ノードで書き込んでいる．**図 4** に 1 ノードから 15 ノードまでノード数を変えて書き込んだ結果を示す．どちらもノード数に対して線形にスケールしているが，Gfarm が HDFS より 30% 近く高い性能を示している．

5.2 読み込み性能

書き込み性能の評価と同様に TestDFSIO ベンチマークを使用して性能を計測した．本評価ではプログラムに一部変更を加えた．TestDFSIO の読み込みベンチマークではデータの配置を考慮せずにデータを読み込む．しかし，実際の MapReduce プログラムではデータの配置を考慮出来るように修正した．このプログラムでは各ノードに 5GB のファイルが生成されている状態で，各ノードで異なるのファイルを同時に読み込んでいる．生成した後，メモリから追い出してから読み込みを行っている．また，読み込みにおいてタスクをデータの近くに配置することの性能への影響を調べるためにデータの位置を考慮せずにタスクを

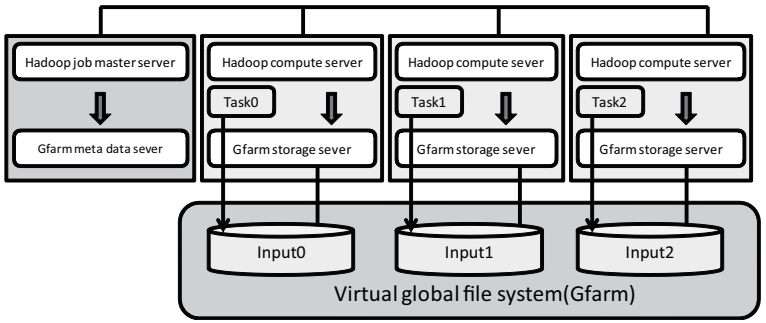

図 2 Hadoop と Gfarm の相互関係

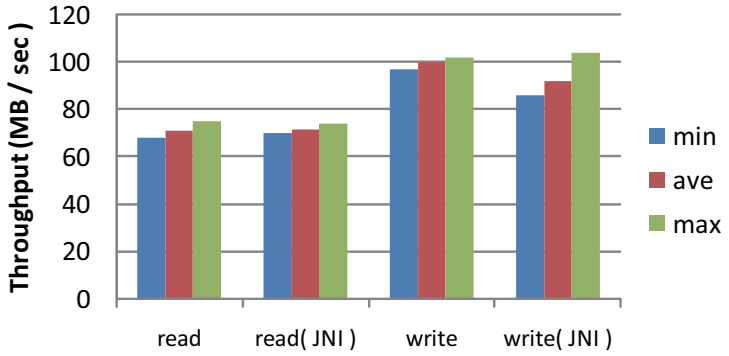

図 3 JNI の shim layer に関するオーバーヘッドの評価

配置した場合の性能を計測した．**図 5** では Gfarm w/ affinity がデータの配置を考慮した場合，Gfarm w/o affinity はデータの位置を考慮していない場合の結果を示す．HDFS と Gfarm はほぼ同等の性能を示している．また，Gfarm においてデータの位置を考慮した場合と考慮しない場合でも，ほとんど性能差は見られなかった．しかし，データの位置を考慮していない場合はネットワークのトラフィックが高くなっており，ノード数がさらに多い環境ではネットワークがボトルネックとなると考えられる．また，今回の読み込みでは各ノードが一つの別々のファイルを最後まで読み込むため，どのようにタスクを配置してもディスクアクセスの衝突が起こることはない．しかし，通常の MapReduce アプリケーションでは

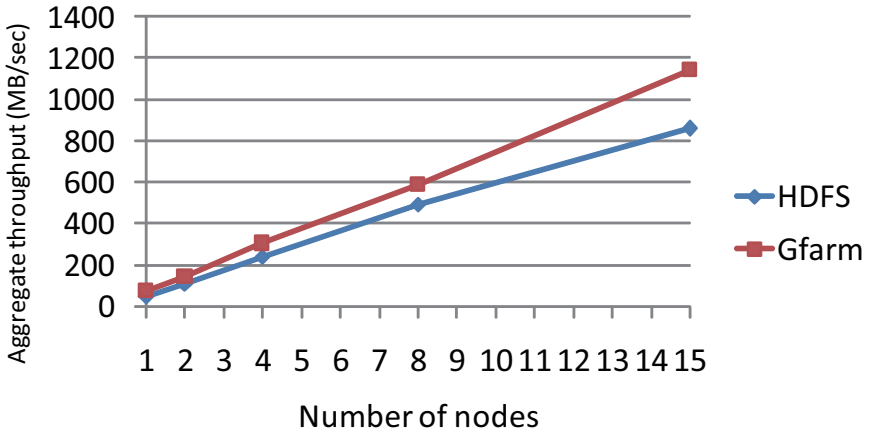

図 4 書き込み性能

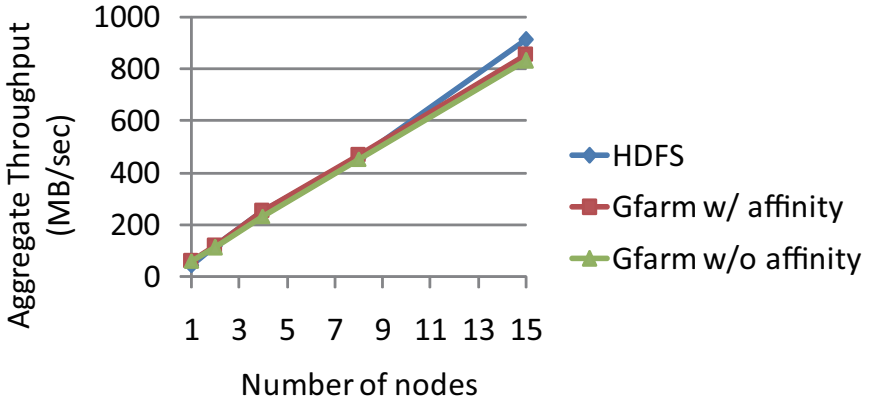

図 5 読み込み性能

入力データを分割して読み込むので，データの位置がわからない場合はスケジューリングに偏りが起こり，性能低下が起こると考えられる．

5.3 grep

Hadoop 付属の Teragen プログラムを使用して生成したデータに対して grep を行った．生成されたデータは key が 10byte, value が 90byte のテキストである．このデータに対して”AAA”の文字列で grep を行った．map タスクではテキストを検索し，マッチしたテキストを key，数字の 1 を value として出力する．reduce タスクでは map の出力の value を合計し，出現回数をカウントしている．また，combiner として reduce と同じ関数を使用し，map タスク側で結果をまとめることでデータの転送量を減少させている．データを生成してから grep を行う前に，データをメモリから読み込まないようにメモリをクリアしてから評価を行った．結果を**図 6** に示す．縦軸は入力データサイズに対する平均秒間処理バイト数である．Gfarm と HDFS はほぼ同程度の性能を示した．Gfarm においてデータの配置の考慮がある場合 (Gfarm w/ affinity) と無い場合 (Gfarm w/o affinity) を比較した結果，15 ノードでは約 5 倍の性能差が見られた．Hadoop のスケジューリングでは，ファイルの位置がわからない場合は 1 ファイル全て処理してから次のファイルを処理していくため，1 つのファイルに対して読み込みが衝突し，効率が低下していた．

5.4 sort

ソート性能の評価には Terasort プログラム[10] を使用した．Terasort プログラムは Teragen プログラムによって生成されたテキストを key によってソートし，そのまま出力するプログラムである．出力ファイルは別々のファイルへ書き出されるが，ファイルの順序は保たれている．つまり，出力ファイル N の全ての key が出力ファイル N+1 のどの key よりも小さいことを保証している．Hadoop MapReduce では自動的に shuffle フェーズでソートを行うため，map タスク，reduce タスクは何もせずに値をそのまま出力するだけである．実行結果を図**図 7** に示す．縦軸は入力データサイズに対する平均秒間処理バイト数である．ソートアプリケーションにおいても HDFS と Gfarm は同等の性能を示した．

6. 関 連 研 究

6.1 Cloudstore

Cloudstore[8] は Hadoop の利用を想定して作られたファイルシステムで，C++で実装されており，POSIX ライクの API を持ち，MapReduce 以外のアプリケーションからも利用可能である．Hadoop-Gfarm と同様に Java Native Interface を使って Hadoop MapReduce アプリケーションから Cloudstore 上のファイルへのアクセスを提供している．しかし，セキュリティやファイルのパーミッションなどの機能は備えていない．

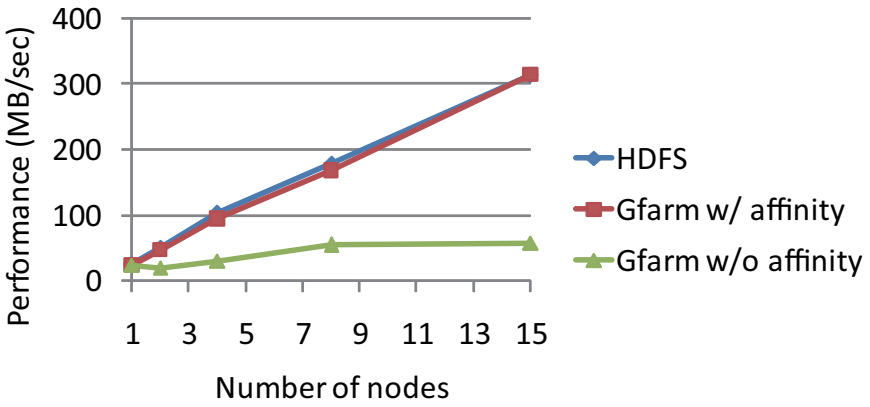

図 6 grep 性能

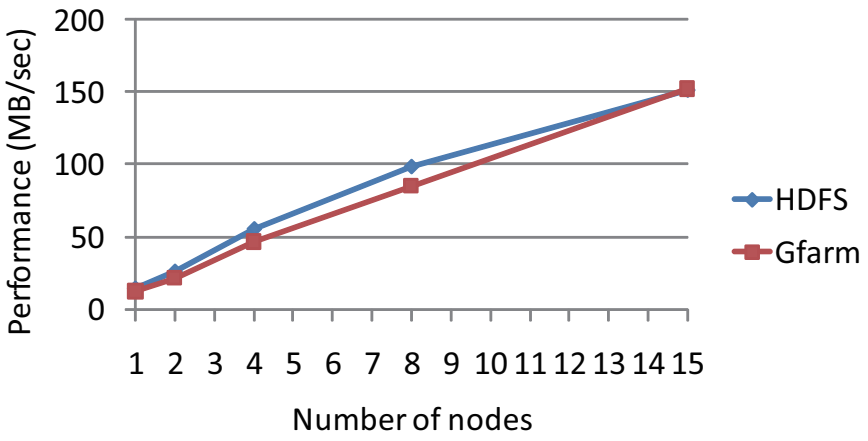

図 7 ソート性能

6.2 Hadoop-PVFS

PVFS[11] は並列アプリケーションによる高速な I/O を提供するために設計された並列ファイルシステムである.[15] では，16 台までの環境で HDFS と PVFS が同等の性能を達成したことを示している．また，PVFS 自体は複製の作成をサポートしていないが，Hadoop-PVFS 側で複製の作成をサポートしている．しかし，PVFS はファイルをストライピングするため，ネットワークへの負荷が多くなり，台数を増やした場合にスケールしない可能性がある．MapReduce のようにタスクを分散してデータの近くに配置する仕組みを持った並列アプリケーションにおいては，ストライピングはローカリティを有効に使うことが出来ないため適していない．

7. ま と め

本稿では HDFS の代わりに Gfarm を使用することを提案し，それを実現するための Hadoop-Gfarm プラグインを設計，実装し，HDFS と Gfarm の性能比較を行った．最大 15 ノードまでの環境において Gfarm は HDFS よりも約 30%高い書き込み性能，同等の読み込み性能を示し，grep や sort の MapReduce アプリケーションにおいても同等の性能を示した．機能面では，POSIX 準拠で汎用的なファイルシステムとして利用できる Gfarm が有利であり，Hadoop MapReduce のファイルシステムとして Gfarm を使用することは有望である．

本稿の残された課題としては，より大規模な広域環境での性能評価やより多様なアプリケーションによる評価，複製がある場合の評価など，実環境に近い状態での性能評価が挙げられる．

謝辞 本研究の一部は，情報爆発時代に向けた新しい IT 基盤技術の研究，文部科学省科学研究費補助金「特定領域研究」(課題番号 21013005) および文部科学省次世代 IT 基盤構築のための研究開発「e-サイエンス実現のためのシステム統合・連携ソフトウェアの研究開発」，研究コミュニティ形成のための資源連携技術に関する研究 (データ共有技術に関する研究) による．

参 考 文 献

1) Apache: HBase, *http://hbase.apache.org/*.
2) Apache: HDFS Architecture, *http://hadoop.apache.org*.
3) Apache: Hive, *http://wiki.apache.org/hadoop/Hive*.
4) Apache: Pig, *http://wiki.apache.org/hadoop/pig*.
5) Ben Langmead, Michael C Schatz, Jimmy Lin, Mihai Pop and Steven L Salzberg: Searching for SNPs with cloud computing, *Genome Biol 2009, 10:R134* (2009).

Vol.2010-HPC-126 No.4
2010/8/3

6) Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C. Hsieh, Deborah A. Wallach, Mike Burrows, Tushar Chandra, Andrew Fikes, and Robert E. Gruber: Bigtable: A Distributed Storage System for Structured Data, *OSDI'06* (2006).
7) Jeffrey Dean, Sanjay Ghemawat: MapReduce: Simplified Data Processing on Large Clusters, *OSDI04* (2004).
8) Kosmos: CloudStore, *http://kosmosfs.sourceforge.net*.
9) Osamu Tatebe, Noriyuki Soda, Youhei Morita, Satoshi Matsuoka, Satoshi Sekiguchi: Gfarm v2: A Grid file system, *Proceedings of CHEP 04* (2004).
10) Owen O Malley and Arun C. Murthy: Winning a 60 Second Dash with a Yellow Elephant (2009).
11) Philip H. Carns, Iii, Robert B. Ross, and Rajeev Thakur: Pvfs: a parallel file system for linux clusters, *In ALS 00: Proceedings of the 4th annual Linux Showcase and Conference* (2000).
12) Sanjay Ghemawat, Howard Gobioff, Shun-Tak Leung: The Google File System, *19th ACM Symposium on Operating Systems Principles* (2003).
13) Services, A.W.: S3, *https://s3.amazonaws.com/*.
14) Szeredi, M.: Filesystem in USEr space, *http://sourceforge.net/projects/avf* (2003).
15) Wittawat Tantisiriroj, Swapnil Patil, and Garth Gibson: Data-intensive file systems for internet services: A rose by any other, *CMU-PDL-08-114* (2008).