POSIX 準拠の広域分散ファイルシステム Gfarm 上での Hadoop MapReduce アプリケーション

三上俊輔[†] 太田一樹[††] 建部修見[†]

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，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．評価の結果、様々なベンチマークにおいて Gfarm は HDFS と同等以上の性能を示した．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．

Hadoop MpaReduce on the Gfarm POSIX Compatible Grid File System

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

Google File System and Hadoop Distributed File System(HDFS) are used for storage of MapReduce.These file systems relax some POSIX requirements to enable high thoughput streaming access to these file systems. Due to a lack of of POSIX compatibility, it is difficult for existing software to directly access data stored in HDFS. Therefore, it is not possible to share storage among existing software and MapReduce applications. In order to external applications to process data using MapReduce, we must first import the data, process it, then export the output data into a POSIX compatible file system. This results in a large number of redundant file operations. In order to solve this problem we design and implement of Hadoop-Gfarm plug-in to use the Gfarm file system instead of HDFS. Our evaluation shows that the Hadoop-Gfarm plugin performs just as well as Hadoop's native HDFS. The Gfarm file system has an advantage since it supports not only MapReduce applications but also POSIX and MPI applications. If you use the Gfarm file system, you can reduce the redundant copy and storage operations without any performance degradation.

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[1] 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．[2]

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

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[3] 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[4] では，Google File System のオープンソース実装である HDFS(Hadoop Distributed File System)[5] をデータの入出力に利用している．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 の要件を緩和しており，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．

これらの機能がないことにより，MapReduce 以外のアプリケーションからそれらのファイルシステムに

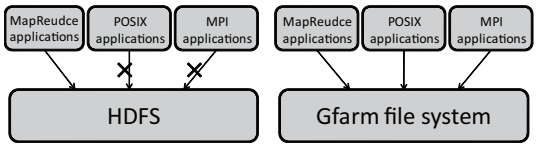

図 1 HDFS 上では Haodop MapReduce アプリケーションしか実行出来ないが、Gfarm 上では通常の POSIX アプリケーションや MPI のアプリケーションを実行できる

アクセスすることが困難となっている．一方で，科学技術計算においては MATLAB などの既存のソフトウェアを使用する必要がある場合や，MPI を使用する方が遥かに高速に処理できる計算もある．それらにもファイルを共有するために分散ファイルシステムが必要であるが，先述の理由により MapReduce のためのファイルシステムとは別になってしまう．

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．この場合，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 **図 1** に示すように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[6] 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[7] 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[8] 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．

本研究の貢献は POSIX 準拠の分散ファイルシステムでも HDFS と同等以上の性能を出すことが可能であることを示したことである．我々の性能評価では様々な種類のベンチマークにおいて Gfarm は HDFS と同等以上の性能を示し，性能を犠牲にすることなく，Gfarm 上のデータを MapReduce や POSIX や MPI などのアプリケーション間で共有出来ることがわかった．

2. MapReduce

MapReduce 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，

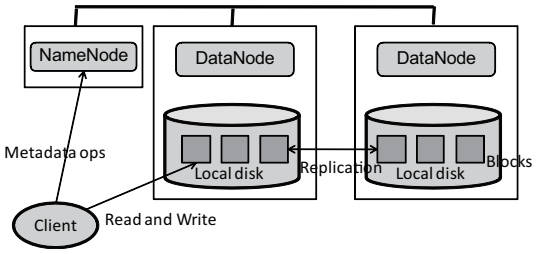

図 2 HDFS のアーキテクチャ

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[9] 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, BigTable にそれぞれ HDFS, Hadoop MapReduce, HBase[10] 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[11] や Hive[12] 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．両方に共通する特徴も多いが，重要な違いもある．

3.1 HDFS

HDFS は大量のデータを保存して，高スループットでアクセス出来るように設計された分散ファイルシステムである．ファイルはチャンクと呼ばれる単位に分割され、クラスタ内の別々のノードに保存される．一

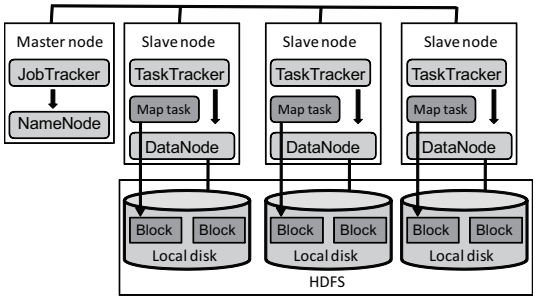

図 3 Hadoop MapReduce と HDFS の物理構成

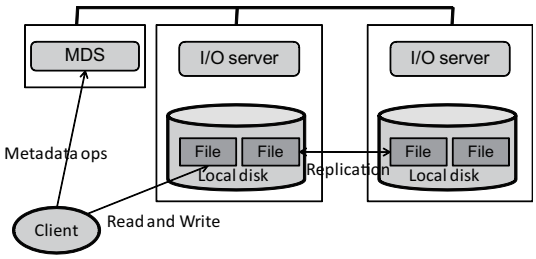

図 4 Gfarm のアーキテクチャ

つのチャンクサイズはデフォルトでは 64MB で，複数ノードに複製を保存し，マシンが故障してもデータを失わないように設計されている．HDFS は MapReduce に特化しており，MapReduce ジョブを実行するのには不必要だが，それ以外のジョブを実行させる時に必要な機能に欠けている．具体的には，HDFS はファイルの任意の位置の修正や複数ライターからの同時書き込みができない．これらの機能がないため，MapReduce 以外のアプリケーションが HDFS に直接アクセスすることは困難になっている．

HDFS は**図 2** に示すようにマスター・スレーブアーキテクチャである．NameNode と呼ばれるメタデータサーバがファイルシステムのメタデータを管理し，クライアントからのメタデータへのアクセスを受け付ける．DataNode と呼ばれるストレージサーバが実際のデータを保持し，クライアントからのデータの読み書きを受け付ける．

HDFS は Hadoop MapReduce と連携して動作可能である．通常は**図 3** の様に，HDFS のストレージサーバの DataNode と Hadoop MapReduce の計算サーバである Task Tracker は同一のノードにに配置する．そして，Hadoop MapReduce のジョブ管理サーバの JobTracker が NameNode にアクセスしてデータが実際にどこのノードに配置されているかを取得し，データに近い位置にある Task Tracker にタスクを配置する．これにより無駄なネットワークの使用を抑え，効率の良い I/O を実現している．

3.2 Gfarm

Gfarm は NFS の代わりとなり得る汎用的な分散

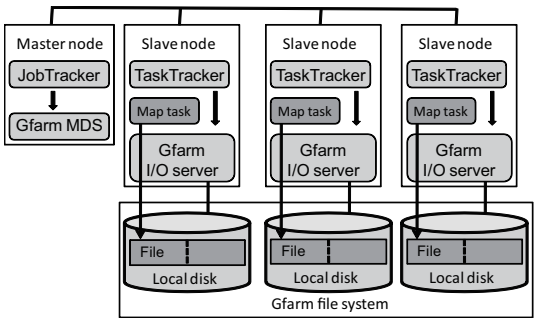

図 5 Hadoop MapReduce と Gfarm の物理構成

ファイルシステムである．HDFS 同様に計算ノードのローカルディスクを束ねて一つの仮想的なファイルシステムを提供する．**図 4** に示すように，メタデータサーバ (MDS) がファイルシステムのメタデータを管理し，ストレージサーバ (I/O server) が実際のデータへのアクセスを提供するのも HDFS と類似する特長である．クライアントは Gfarm のクライアントライブラリ，または FUSE[7] を使って Gfarm 上のファイルへアクセス出来る．FUSE を使えば POSIX アプリケーションは全て透過に Gfarm 上のファイルへアクセス可能である．

HDFS とのアーキテクチャ上の大きな違いとして，HDFS はファイルをブロック分割するが，Gfarm はファイルをブロック分割しないことが挙げられる．ブロック分割によって大きいサイズのファイルも分割して保存され，分散して処理することが可能となるが，実際のデータセットはデータの一部だけを取り出す場合も多く，複数のファイルに分割して保存されることが多い．ファイル分割をし，データセットはワイルドカードで指定するなどして管理すれば，ブロック分割をしなくても分散して処理が可能である．一方でブロック分割をする場合はファイルのメタデータに加えてブロックの情報を管理するため，メタデータサーバに必要なメモリ量が多くなるなどの欠点が考えられる．

HPC 向けの並列ファイルシステムでは PVFS[13] の様にストライピングを使うことが多いが，Gfarm ではファイルをストライピングしない．ファイルストライピングはクライアントが少数の場合はディスクを有効に使い性能を向上させることが可能であるが，多くのクライアントから同時に書きこむ場合，ネットワークがボトルネックとなる可能性がある．

他の Ceph[14] や Lustre[15] などの汎用的な分散ファイルシステムと比較すると，それらは計算ノードとは別に専用のストレージを必要とする．また，そのストレージと計算ノード間に十分なネットワークバンド幅がないとそこがボトルネックとなってしまう．Gfarm は計算ノードのディスクを利用し，追加のストレージも必要なく，ローカリティを利用し，ネットワークの I/O を減らすことが出来る．

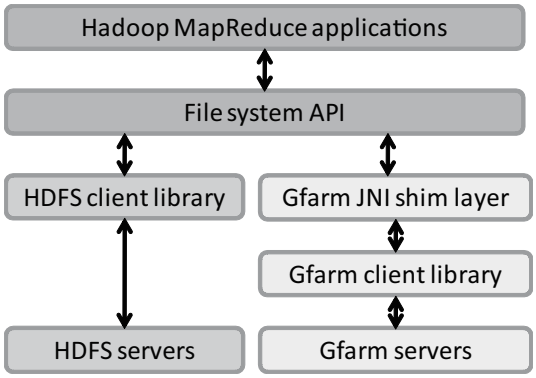

図 6 Hadoop-Gfarm ソフトウェアスタック

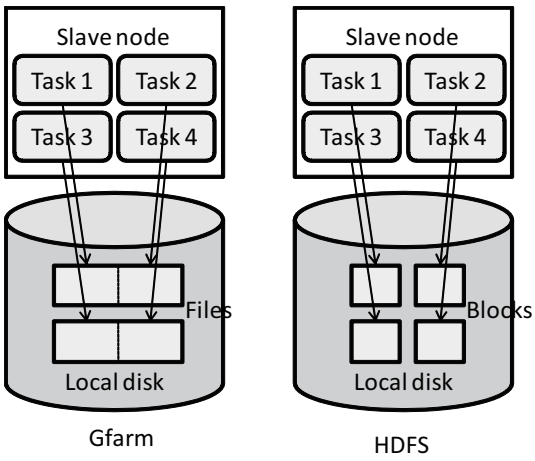

図 7 両ファイルシステムにおけるディスクアクセスパターン

Gfarm では二つの手法によりディスクアクセスの集中を防いでいる．一つ目はファイルにアクセスする際に適切な複製を選択することである．クライアントは MDS からアクセスするファイルの複製を持つノードの情報を受け取り，容量が空いていて CPU 利用率が閾値よりも下のノードのうち，RTT が最も小さいノードを選択する．二つ目はプロセスを配置する際にデータと近い場所に配置し，ネットワーク I/O を減らすことである．これにはスケジューラの助けが必要であるが，Gfarm はファイルの場所を取得するための API やコマンドがあり，それらを利用してスケジューリングすることが出来る．実際に Pwrake[16] という並列分散ワークフローシステムではローカリティを考慮したスケジューラを実装しており，Gfarm 上にてファイルが存在するノードに優先的にタスクを割り当て，効率の良い処理が可能となっている．

4. Hadoop-Gfarm プラグインの設計と実装

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[17] を開発した．

3.1 節と 3.2 節で説明した様に，HDFS と Gfarm の物理的な構成は似ている．HDFS の DataNode と Gfarm の I/O server, HDFS の NameNode と Gfarm の MDS がそれぞれ対応し，いずれも計算ノードのローカルディスクを利用してファイルシステムを提供する．Hadoop MapRdcue と HDFS をクラスタに配置する場合の構成は図 3 の様に DataNode と TaskTracker を同じノードに配置するが，Gfarm の場合でも**図 5** の様に I/O server と TaskTracker を同じノードに配置する構成をとることが可能である．

HDFS はファイルをブロック分割し，複数ノードに配置するため，それぞれのノードに分散してタスクを配置することが可能である．一方で Gfarm ではファイルを分割しない．しかし，ブロック分割しなくても，複数ファイルが各ノードに分散して配置されていれば MapReduce を実行する際も HDFS と同じディスクアクセスパターンをとることが出来る．例えば，**図 7** の様に，256MB のファイルが二つあると仮定する．HDFS の場合はブロックサイズを 128MB とすると，合計で 4 つのブロックとなり，各 map タスクが 128MB のブロック全体を読み込む．Gfarm の場合はファイル自体は分割されないが，各 map タスクは図 7 で示されている様にファイルの前半 128MB か後半 128MB を処理する．この場合 HDFS と Gfarm におけるディスクへのアクセスパターンは同一となる．

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 CloudStore[18] や S3[19] 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 6** 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

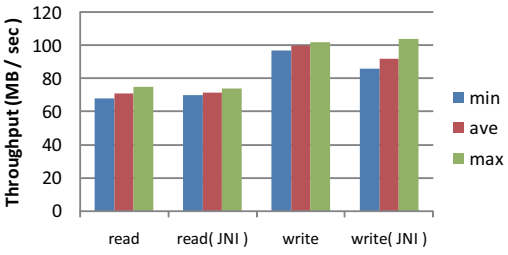

図 8 JNI の shim layer に関するオーバーヘッドの評価

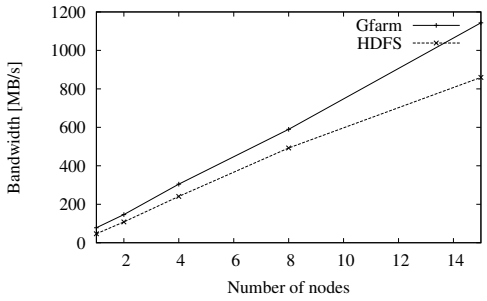

図 9 書き込み性能

の関数を実装することで Hadoop のジョブを管理する JobTracker 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．

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呼び出してしている．そこで実際の性能評価の前に，予備評価として JNI の shim layer に関するオーバーヘッドの計測をした．表 1 に使用したマシンの性能を示す．結果を**図 8** に示す．読み込みでは性能差は 1% 以下であり，書き込みにおいては 9% 程度のオーバーヘッドがあった．

表 1 マシン性能

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

5. 性能評価

HDFS と Gfarm の比較をするために，様々な種類のベンチマークを行った．表 1 に実験に使用したマシンの性能を示す．最後の StarCount の評価以外では 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 マイクロベンチマーク

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．書き込みの評価では各ノード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 9** 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．

読み込み性能の評価にも TestDFSIO ベンチマークを使用したが，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が，データの配置を考慮出来るようように修正した．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配置した場合の性能を計測した．**図 10** 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たスケジューリングがない場合の結果を示している．HDFS と Gfarm はほぼ同等の性能であった．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入力データを分割して読み込むため，データの位置がわからない場合はディスクアクセスが偏り，性能が低下すると考えられる．

5.2 MapReduce アプリケーションによる評価

MapReduce アプリケーションでの評価として Hadoop に付属の Grep プログラムと Terasort プログラムを使用した．Grep ベンチマークでは，ランダムデータを生成し，それに対して”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，Hadoop

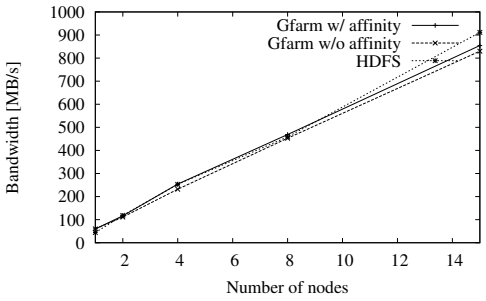

図 10 読み込み性能

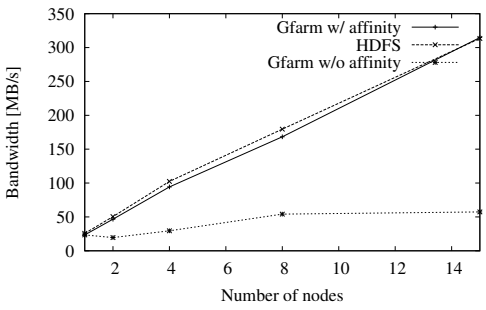

図 11 grep 性能

の機能である combiner を使用して，map タスク側で結果を一度集約す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 11** 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．Grep はデータの読み込みが処理の大部分を占めるため，マイクロベンチマークにおいて読み込み性能が互角であった結果と整合があると言える．

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．5.1 節における読み込みの評価ではデータ配置の考慮による性能差が小さかった事を考えると，Grep ではネットワークはボトルネックとなってはいないが，ディスクアクセスが集中したことにより性能が低下したと考えられる．

ソート性能の評価には Terasort プログラム[20] 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 12** 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．15 ノード使用した場合，Gfarm の方が約 10% 速かった．処理の内訳を調べた結果，Map フェーズでは性能差はないが，Reduce フェーズで Gfarm の方が速かった．Terasort において，Map フェーズでは読み込み，Reduce フェーズではデータの書き込みが処理の大部分を占めるが，5.1 節の結果において Gfarm の方が書き込みが速い結果と整合があると考えられる．

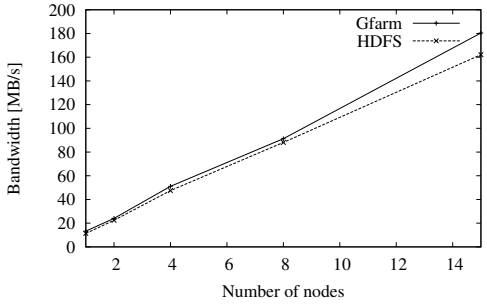

図 12 ソート性能

5.3 Gridmix

実運用環境の Hadoop クラスタは複数ユーザに使用され同時に様々な種類のジョブが実行される．そのような実際に近い状態をエミューレートするツールが Gridmix である．今回の評価では Gridmix2 を使用した．Gridmix はランダムデータを生成し，用意されているいくつかの種類のジョブをそのデータに対して実行する．それぞれのジョブの数は設定ファイルによって設定することが出来る．

今回の評価では JavaSort，Combiner，MonsterQuery という 3 種類のジョブを使用した．JavaSort は通常の Java によるソートプログラムで，Combiner は combiner という Hadoop の機能を使用したワードカウントのプログラムで，MonsterQuery は Hive や Pig のように最初の MapReudce ジョブの結果を次の MapReduce ジョブに渡して実行するような多段のジョブである．今回は 15 ノードを使用し，全部で 45 個のジョブを実行するように設定し，HDFS と Gfarm で全てのジョブが終わるまでにかかる時間を比較した．

図 13 は時間の経過と共に終了していないジョブが減っていく経過を示している．途中まではほぼ同じ速度でジョブを完了しているが，最終的には Gfarm の方が約 5% ほど速く全てのジョブを終了した．

5.4 StarCount

他のアプリケーションで生成したデータを Hadoop MapReudce で処理する場合，HDFS を使用する場合

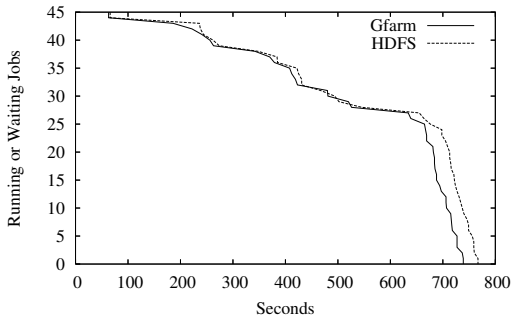

図 13 Gridmix ベンチマークの実行結果

はコピーする必要があるが，Gfarm を使用すれば直接処理することが可能である．無駄なコピーをなくすことによる速度向上を定量的に評価するために一つのユースケースを想定し，評価を行った．

この評価では，まず天体の画像データから星の明るさや位置などの情報を抜き出したテキストに変換し，それを MapReduce のジョブにて解析するというユースケースを想定した．入力データには 2MASS[21] が公開している FITS 形式の天体画像データを使用した．FITS データの解析には SExtractor[22] を使用し，それを Pwrake を使用したワークフローによって並列実行した．そしてその解析結果に対して等級毎に星の数を数える MapReduce アプリケーションを実行した．本論文中ではこの Pwrake によるワークフローのことを Source Extract と呼び，星を数える MapReduce アプリケーションのことを StarCount と呼ぶ．

評価には 30 ノード使用し，最初の Source Extract に対する入力データサイズは約 55GB であり，StarCount に対する入力データサイズは約 9GB であった．Source Extract はいずれも Gfarm 上で実行し，StarCount に HDFS を使う場合は Gfarm から HDFS へ Source Extract の結果をコピーしてから StarCount を実行した．実行結果を **図 14** に示す．StarCount のジョブ自体の速度は HDFS と Gfarm で同程度なので，Hadoop-Gfarm プラグインを使用することによって，Gfarm から HDFS へのデータのコピーにかかっている約 450 秒を節約することが出来た．これによる全体の実行時間の向上は約 20% であった．

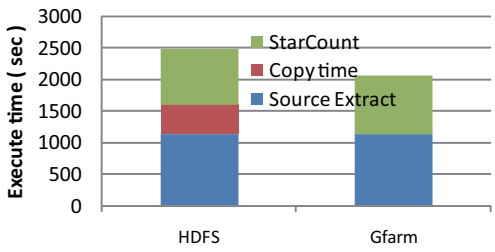

図 14 Souruce Extract と StarCount のワークフロー実行結果

6. 関連研究

6.1 Cloudstore

CloudStore は Hadoop で使用するために作られた C++で実装されている分散ファイルシステムで，特徴は HDFS とかなり似ているが，POSIX に準拠している．本研究の手法と同じく JNI を介して Hadoop MapReduce が CloudStore 上のファイルにアクセス出来る．しかし，汎用的なファイルシステムに必要な認証や権限などの機能がなく，複数ユーザで安全にファイルを共有することが出来ない．

6.2 Hadoop-PVFS と Hadoop on GPFS

PVFS と GPFS[23] は違う特徴も持つが，いずれも並列ストライピングファイルシステムである．それぞれ MapReduce アプリケーションにおいて HDFS と同等以上の性能を持つことが示されているが[24),25)]，性能を出すために MapReduce で処理するファイルには 128MB のブロックサイズを使用し，それ以外は 128KB 程度のブロックサイズを使用している．しかし，それでは MapReduce とそれ以外のアプリケーションでファイルを共有していることにならない．

6.3 Ceph

Ceph はカリフォルニア大学サンタクルーズ校 (UCSC) で開発が始まった分散ファイルシステムで，Linux カーネル 2.6.34 以降に組み込まれている．メタデータサーバを分散することができ，多くの分散ファイルシステムが抱える単一メタデータサーバによるの性能限界を超えることが可能とされている．公開されているパッチを Hadoop に当てることにより Ceph 上で Hadoop を実行することが可能である．しかし，Ceph はまだ試験段階であり，まだ本番環境で使用可能な状態ではない．

6.4 BlobSeer

BlobSeer[26] はバージョニング機能を持つ分散ファイルシステムである．HDFS 以上の性能を持つことが示されているが，BlobSeer はオンメモリのストレージである．

7. 結　　論

本研究では 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．マイクロベンチマークにおいては 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，その他の様々な種類のベンチマークにおいても Gfarm は HDFS と同等以上の性能を示した．HDFS は高スループットを実現するために POSIX の要件を緩和しているが，POSIX 準拠のファイルシステムでも HDFS と同等以上の性能を出すことが可能であることが分かった．

今後の課題としては複製がある場合の評価が挙げ

られる．両者とも複製機能を持っているため，ベンチマーク実行時に複製を作成すること自体は簡単であるが，両者の複製の作成方法が異なるため，比較が行えない．HDFS は書き込み時に同期的に複製を作成し，書き込みが終了した時には全ての複製が作成されている状態だが，Gfarm は最初の複製を書き込んだ後に非同期で複製を作成する．このような違いがあるため今回の評価では複製なしで測定を行ったが，実際の環境において複製は重要な機能である．

参 考 文 献

1) Jeffrey Dean, Sanjay Ghemawat: MapReduce: Simplified Data Processing on Large Clusters, *OSDI04* (2004).
2) 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).
3) Sanjay Ghemawat, Howard Gobioff, Shun-Tak Leung: The Google File System, *19th ACM Symposium on Operating Systems Principles* (2003).
4) Apache: Welcome to apache hadoop!
5) Apache: HDFS Architecture, *http://hadoop.apache.org/common/docs/c-urrent/hdfs_design.html*.
6) Osamu Tatebe, Kohei Hiraga, Noriyuki Soda: Gfarm Grid File System, *New Generation Computing, Ohmsha, Ltd. and Springer, Vol.28, No.3, pp.257-275, DOI: 10.1007/s00354-009-0089-5* (2010).
7) M. Szeredi: Filesystem in USEr space, *http://sourceforge.net/projects/avf* (2003).
8) 木村浩希: 広域分散ファイルシステム Gfarm の MPI-IO の実装と評価, *SWoPP 2010*.
9) 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).
10) Apache: HBase, *http://hbase.apache.org/*.
11) Apache: Pig, *http://wiki.apache.org/hadoop/pig*.
12) Apache: Hive, *http://wiki.apache.org/hadoop/Hive*.
13) 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).
14) Sage A. Weil. Scott A. Brandt. Ethan L. Miller. Darrell D. E. Long. Carlos Maltzahn: Ceph: A scalable,high-performance distributed file system, *in Proc. of the 7th USENIX Symposium on Operating Systems Design and Implementation (OSDI 06), pp.307.320* (2006).
15) Braam, P. J.: Lustre, *http://www.lustre.org/*.
16) Masahiro Tanaka, Osamu Tatebe: Pwrake: A parallel and distributed flexible workflow management tool for wide-area data intensive computing, *Proceedings of ACM International Symposium on High Performance Distributed Computing (HPDC), pp.356-359* (2010).
17) Kazuki Ohta, Shunsuke Mikami: Hadoop-Gfarm, *https://gfarm.svn.sourceforge.net/svnroot/gfarm/gfarm_hadoop*.
18) CloudStore: *http://kosmosfs.sourceforge.net*.
19) Amazon Web Services: S3, *https://s3.amazonaws.com/*.
20) Owen O Malley and Arun C. Murthy: Winning a 60 Second Dash with a Yellow Elephant (2009).
21) Cutri R.M., Skrutskie M.F., van Dyk S., et al.: 2MASS All Sky Catalog of Point Sources (Amherst: Univ. Massachusetts Press; Pasadena: IPAC), *http://www.ipac.caltech.edu/2mass/releases/allsky/* (2003).
22) E. Bertin and S. Arnouts: SExtractor: Software for source extraction, *Astronomy and Astrophysics Supplement Series, Vol. 117, 393-404* (1996).
23) FrankSchmuckand Roger Haskin: GPFS: A Shared-Disk File System for Large Computing Clusters, *Proceedings of the First USENIX Conference on File and Storage Technologies, pages 231-244* (2002).
24) Wittawat Tantisiriroj, Swapnil Patil, and Garth Gibson: Data-intensive file systems for internet services: A rose by any other, *CMU-PDL-08-114* (2008).
25) Karan Gupta, Reshu Jain, H. P. P. S. D. S.: Scaling Highly-Parallel Data-Intensive Supercomputing Applications on a Parallel Clustered File system, *The SC10 Storage Challenge* (2010).
26) B. Nicolae, G. Antoniu, L. Bouge, D. Moise, and A. Carpen-Amarie: BlobSeer: Next Generation Data Management for Large Scale Infrastructures, *Journal of Parallel and Distributed Computing* (2010).