Vol.2015-DPS-163 No.17
Vol.2015-MBL-75 No.17
2015/5/28

Hadoop MapReduce の Reduce 処理の I/O 高速化

藤島 永太[†]　　山口 実靖[‡]
工学院大学大学院工学研究科電気・電子工学専攻[†]
工学院大学工学部情報通信工学科[‡]

近年，膨大な情報を収集・蓄積・分析する方法として，ASF(Apache Software Foundation)が開発・公開している Hadoop が注目されている．
一般に Hadoop MapReduce は，Map 処理と Reduce 処理の二段階でデータの処理を行う．Map 処理は，データの断片の加工や必要な情報の抽出を行う．Reduce 処理は，Map 処理の出力をまとめ，最終的な結果の出力を行う．よって，多数の Map 処理の出力を単一の Reduce 処理に入力として与えるようなワークロードでは，Reduce 処理が I/O バウンドとなり，全体の処理時間が長くなる場合がある．
本研究では，巨大なファイルをシーケンシャルに読み書きする Hadoop の特性と定記録密度方式 HDD の特性に着目し，MapReduce 処理の I/O 高速化手法を提案し，処理時間の比較によりその有効性を示す．

1. はじめに

近年，世界中の情報量が爆発的に増加しており，その情報を収集・蓄積・分析して有効に活用することに注目が集まっている．その膨大な情報を扱う方法として，ASF(Apache Software Foundation)が開発・公開している Hadoop に注目が集まっている．

一般に Hadoop MapReduce は，Map 処理と Reduce 処理の二段階でデータの処理を行う．Map 処理では，分割されているデータの断片の加工や，必要な情報の抽出を行う．Reduce 処理は，Map 処理の出力をまとめ，最終的な結果を出力する．よって，多数の Map 処理の出力を単一の Reduce 処理に入力として与えるようなジョブでは，Reduce 処理が I/O バウンドとなる場合がある．

本研究では，Hadoop MapReduce の中間データが HDD 上でシーケンシャルに読み書きされることと，定記録密度方式の HDD においてシーケンシャル I/O 速度がゾーン毎に異なることに着目し，上記の単一 Reduce 処理となるジョブの性能の改善を行う．具体的には，ファイルシステムの制御により MapReduce 処理の中間データを HDD の高速部(HDD の外周)に作成させ，Reduce 処理のシーケンシャル I/O 速度を向上させる手法を提案し，評価によりその有効性を示す．

本稿の構成は以下の通りである．2 章で MapReduce 処理の流れを示し，3 章で単一 Reduce 処理となるジョブにおけるボトルネックが Reduce 処理ノードの I/O 処理にあることと，その I/O 処理がシーケンシャル I/O であることを示す．4 章で HDD のゾーンとシーケンシャル I/O 速度について述べ，5 章で Reduce 処理の I/O 高速化手法を提案し，6 章で性能評価を行う．7 章で関連研究を紹介し，8 章にて本稿をまとめる．

Performance Improvement of Reduce Phase of Hadoop MapReduce,
† Eita Fujishima, Electrical Engineering and Electronics, Kogakuin University Graduate School
‡Saneyasu Yamaguchi, Information and Communication Technology, Kogakuin University

2. MapReduce 処理の流れ

MapReduce は，Map 処理と Reduce 処理の二段階でデータの処理を行う．

Map 処理では，JobTracker が HDFS 上の入力データを分割して各 Map タスクに割り当て，それらの Map タスクを TaskTracker に割り振る．Map タスクを割り振られた TaskTracker は，処理対象データから Key/Value ペア群を生成する．そして，各ペアに対して Map 関数を実行し，新たな Key/Value ペア群を中間データとして出力する．

一方 Reduce 処理では，JobTracker が中間データを Key 毎にグルーピング(Shuffle)して各 Reduce タスクに割り当て，それらの Reduce タスクを TaskTracker に割り振る．Reduce タスクを割り振られた TaskTracker は，Key 毎に集められた中間データに対して Reduce 関数を実行し，最終的な結果となる Key/Value ペアを出力する．

よって，Reduce 処理を行う TaskTracker が一つしか存在しないような MapReduce ジョブを処理する場合，複数の Map タスクの中間データが単一の Reduce 処理ノードに集中して転送され，Reduce 処理ノードにおける I/O 処理がボトルネックとなる[1].

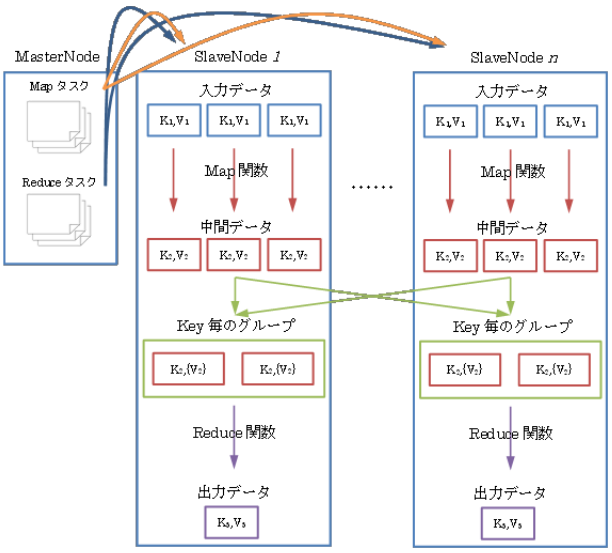

図 1. MapReduce 処理の流れ

Reduce 処理では，大量のデータの受信およびそれらのファイルの書き込みと，ファイルからの大量のデータの読み込みが主な I/O 処理となるため，ディスクのシーケンシャルリード/ライト速度が Reduce 処理時間にとって重要な性能になると考えられる．

3. Reduce 処理のボトルネック

単一 Reduce 処理となるジョブである TeraSort を実行して，Reduce 処理中のボトルネック処理を Linux の vmstat コマンドおよび iostat コマンドを使用して調査した．測定環境としては図 2 のようにマスターノード 1 台およびスレーブノード 4 台を用い，TeraSort の入力データサイズは 16 GB，HDFS のブロックサイズはデフォルトの 64MB，複製数は 3 つとした．マスターノードの仕様は，CPU が Celeron 2.27GHz，HDD が 640GB，メモリが 16GB，OS が CentOS 6.5 x86_64 (Linux 2.6.32)，ファイルシステムが ext4 である．スレーブノードの仕様は，CPU が AMD Athlon II X2 220 Processor 2.7 GHz，HDD が 250 GB と 500 GB，メモリが 2 GB，OS が CentOS 6.5 x86_64 (Linux 2.6.32)，ファイルシステムは 250GB の HDD が ext4 であり 500GB の HDD が ext3 である．中間データを含む全ての Hadoop データは 500GB(ext3)の HDD 内におかれ，本 HDD 仕様の詳細は表 1 の通りである．Hadoop のバージョンは 2.0.0-cdh4.2.1 である．

表 1. 測定用 HDD の仕様

型番	DT01ACA050
インタフェース	SATA 3.0
インタフェーススピード	6.0 Gbps
容量	500 GB
バッファサイズ	32 MB
回転数	7,200 rpm
平均回転待ち時間	4.17 ms

単一の Reduce 処理ノードにおける CPU 使用率，CPU の I/O 待ち率，I/O 使用率の測定結果を図 3 および図 4 に示す．非 Reduce 処理ノードの一つにおける使用率の測定結果を図 5 および図 6 に示す．各グラフの横軸は実行時間[sec]，縦軸は CPU 使用率，I/O 待ち率，I/O 使用率[%]を示している．本測定では，MapReduce 処理開始の 506 秒後に全ての Map 処理が終了し，それ以降は Reduce 処理のみが行われている．図 3，図 4 より，単一の Reduce 処理ノードにおいて I/O 使用率がほぼ全ての時間帯において 100%となっており，

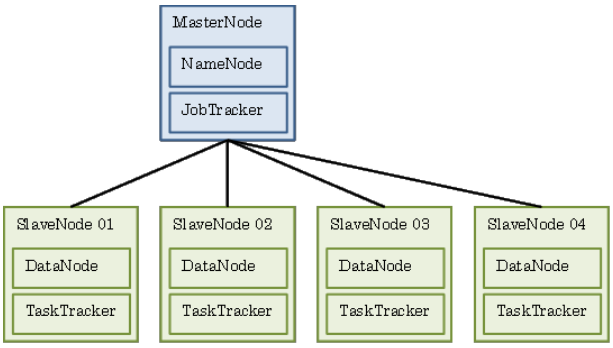

図 2. 測定用 Hadoop クラスタ概略図

本ジョブのボトルネック処理が Reduce 処理ノードの I/O 処理であることが分かる．これは，非 Reduce 処理ノードから送信されてくる Map 処理の出力(中間データ)を単一の Reduce 処理ノードが受信し HDD に書き込んだり，それら中間データを Reduce 処理のために読み込んだりしているためである．

図 5，図 6 より，非 Reduce 処理ノードでは Map 処理終了後は負荷が低く，TeraSort ジョブ全体でみると，ほとんどの時間帯において非 Reduce 処理ノードがボトルネックとなっていないことが分かる．Map 処理中(処理開始から 506 秒まで)に着目しても，I/O 使用率は高くないことが多く，CPU 使用率も図 5 と比較すると高くないことが分かる．非 Reduce 処理ノードにおいて I/O 処理は主に Map 処理の入力データの読み込みのために行われるが，その負荷は比較的低いことが分かる．

以上より，本ジョブの処理時間において単一 Reduce 処理ノードにおける Reduce 処理の時間が大きな比率を占めていること，当該 Reduce 処理が I/O バウンドであることが分かる．

次に，Reduce 処理ノードにおいて HDD に発行された I/O 要求のサイズについて考察する．Linux OS の SCSI 層にて HDD に対して発行された I/O 要求を観察し，I/O 要求の対象アドレスとサイズを調査した．発行 I/O 要求のサイズの分布を図 7 に示す．図における

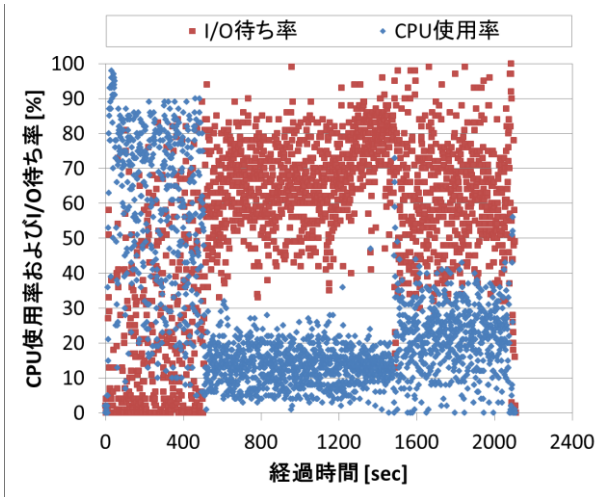

図 3. 単一 Reduce 処理ノードの CPU 使用率および I/O 待ち率

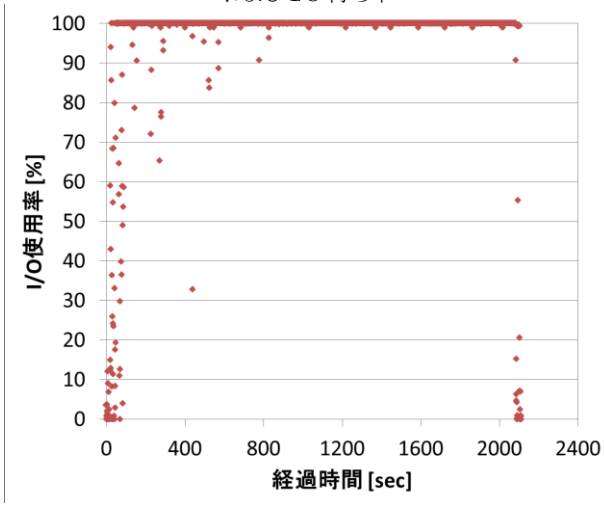

図 4. 単一 Reduce 処理ノードの I/O 使用率

Vol.2015-DPS-163 No.17
Vol.2015-MBL-75 No.17
2015/5/28

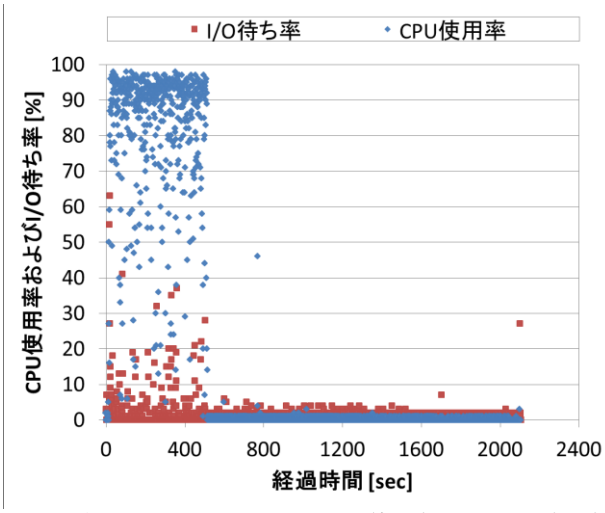

図 5. 非 Reduce 処理ノードの CPU 使用率および I/O 待ち率

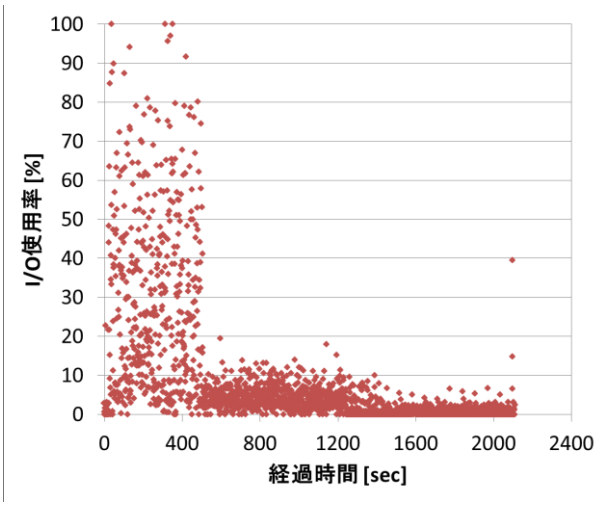

図 6. 非 Reduce 処理ノードの I/O 使用率

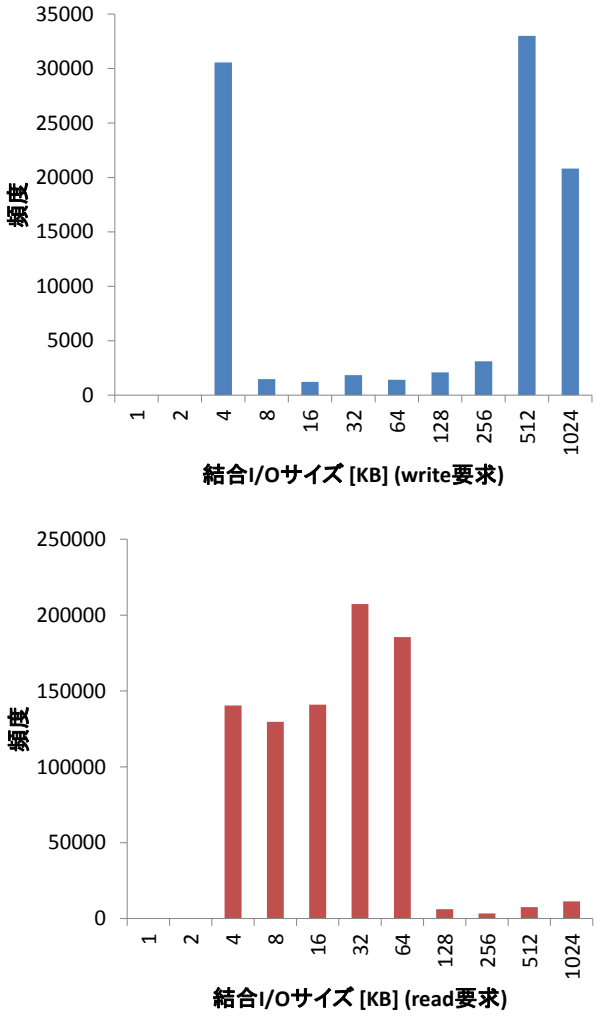

図 7. Reduce 処理ノードの結合 I/O サイズ頻度分布

“結合 I/O サイズ”とは，時間的に連続して発行された I/O 要求群の対象アドレスが空間的に連続している場合は同一の I/O 要求である見なし，それらを結合した I/O 要求サイズである．アプリケーションが巨大な I/O 要求を OS に対して発行すると，OS がこの I/O 要求を複数の細かい I/O 要求に分割してから HDD に対して要求を発行する．上記の“結合 I/O サイズ”はこれらを結合して集計したものである．HDD にとっては，この“結合 I/O サイズ”の大小が HDD がどの程度のシーケンシャル性を持って動作するかを定めるものとなる．

図より，Reduce 処理中には大きなサイズの I/O 要求が多く発行されており，Reduce 処理中は高いシーケンシャル性で HDD が動作していることが分かる．よって，本ジョブの性能を向上させるにはシーケンシャル I/O 速度の向上が重要であると予想できる．

4. HDD のゾーンとシーケンシャル I/O 速度

定記録密度方式 HDD のシーケンシャル I/O 速度はディスクの外周側と内周側のゾーンにより異なる．本章にて，本稿の実験で用いた HDD のゾーンごとのシーケンシャルリード/ライト速度の測定結果を示す．

4.1. 測定方法

Hadoop 用にマウントしている記録容量 500 GB の HDD に対して 64 MB の読込/書込を 7327 回行うプログラムを実行し，本 HDD のシーケンシャルリード/ライト速度を測定した．測定は，ファイルシステムを介さずにデバイスのスペシャルファイルに対して直接読込/書込を行うものと，ファイルシステム(ext2, ext3, ext4)を介して行うものの両方を行い，それらの読込/書込にかかった時間を測定した．測定環境は，3 章で用いたスレーブノードの 1 台である．測定に使用した HDD の仕様は表 1 の通りである．

4.2. 測定結果

測定結果を図 8～図 11 に示す．各グラフの横軸は HDD 内のディスクアドレス[GB]，縦軸は読込/書込時間[sec]を示す．各プロットは 64 MB の読込/書込一回に掛かった時間を示す．図 8 はファイルシステムを介さずにデバイススペシャルファイルに直接読込/書込を行った場合の測定結果，図 9～11 はファイルシステム(ext2,ext3,ext4)を介して読込/書込を行った場合の測定結果である．図 8 において，HDD の最初のゾーンでの読み込みでは平均 0.339 秒，最後から二番目のゾーンでの読み込みでは平均 0.660 秒かかっている．また HDD に直接書込を行う計測では，最初のゾーンでの

読み込みでは平均 0.343 秒，最後から二番目のゾーンでの読み込みでは平均 0.667 秒かかっている．

同様に，図 9～11 のファイルシステムを介して読込/書込を行う測定でも，アドレスの小さい方が速く，アドレスの大きい方が遅いことがわかる．

以上の結果より，ディスクアドレスの小さい外周側のゾーンとディスクアドレスの大きい内周側のゾーンでは性能が大きく異なり，本稿で用いた HDD の例では約 2 倍異なることが分かる．

4.3. 考察

前節の測定より，ファイルに対するシーケンシャルアクセス性能はディスク内の位置により大きく異なることが確認された．ファイルの格納位置はファイルシステムが決定するが，近年のファイルシステムはファイルごとのアクセスパターンに関する情報を保持しておらず，ファイルごとに最適な配置を行うことができない．よって，ファイルシステムの動作を制御し，ファイル格納位置を制御することにより TeraSort の様なシーケンシャル I/O がボトルネックとなるアプリケーションの性能を大幅に改善できると考えられる．

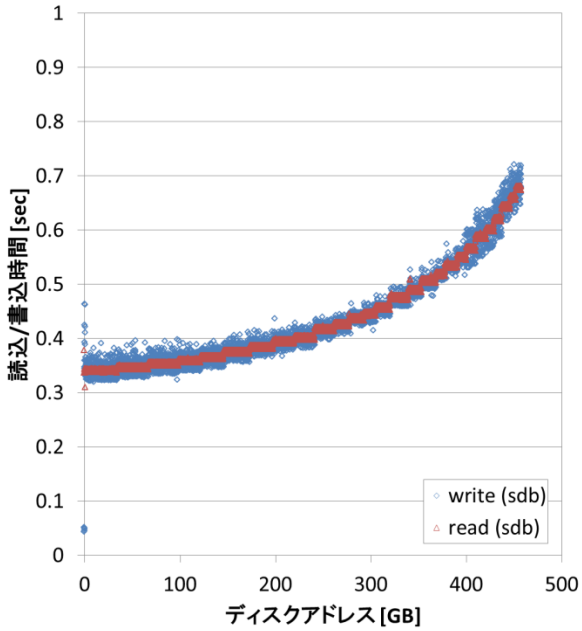

図 8. HDD のゾーン毎のシーケンシャル I/O 時間
(ファイルシステム介さず)

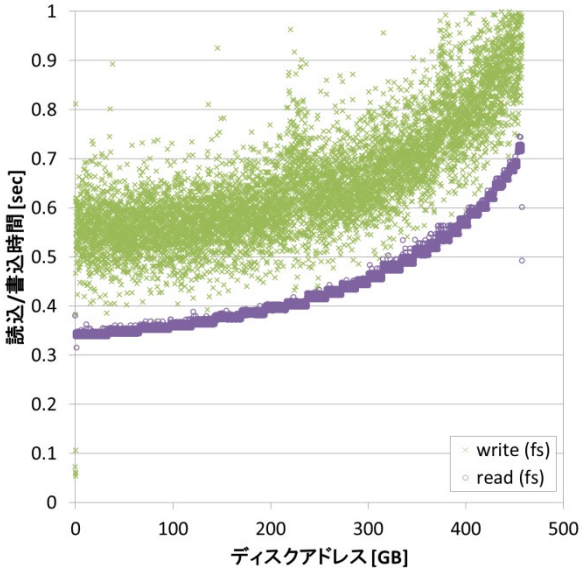

図 9. HDD のゾーン毎のシーケンシャル I/O 時間(ext2)

5. 提案手法

5.1. ファイル格納位置の制御

Hadoop MapReduce で作成される中間データや一時ファイルは，ディスクに十分な連続空き領域がありフラグメンテーションが起きない状況であれば，通常はディスク内の連続領域に書き込まれる．よって，ディスクアドレスの大小に関わらずシーケンシャルに近い形で書き込まれる．従って，ディスクの外周側のゾーンに書き込まれると実行時間が短くなり，逆に内周側のゾーンに書き込まれると

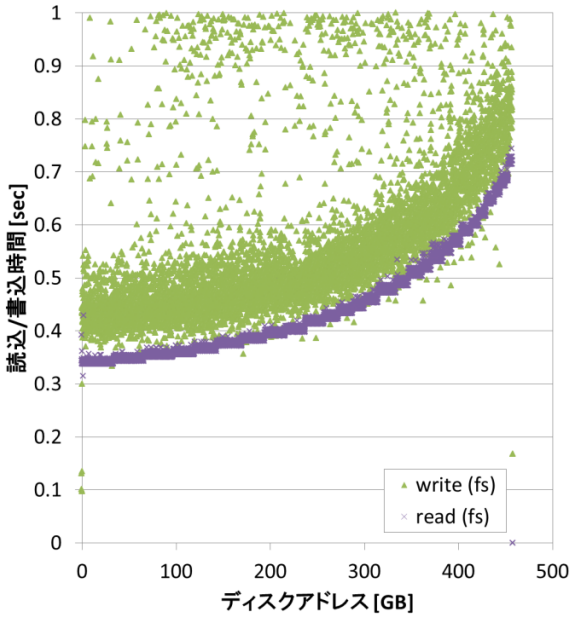

図 10. HDD のゾーン毎のシーケンシャル I/O 時間(ext3)

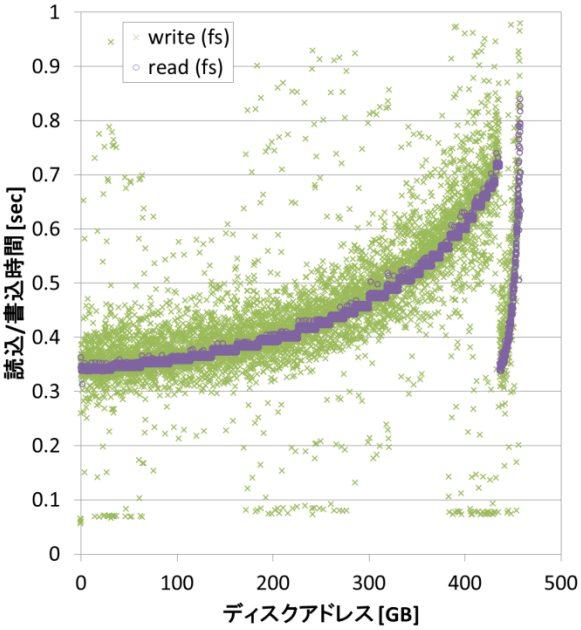

図 11. HDD のゾーン毎のシーケンシャル I/O 時間(ext4)

実行時間が長くなると予想される.

よって, ファイルシステムのデータブロックビットマップを書き換え, ファイルが空き領域内におけるディスクの外周側(アドレスの小さい位置)に作成されるように制御すれば単一 Reduce 処理のジョブの性能を向上させることが可能であると考えられる.

本章にて, ファイルシステムのデータブロックビットマップを書き換え, 常に空き領域内の低アドレス部のみを使用可能とし, 低アドレス部を優先的に使用させることにより Hadoop MapReduce の単一 Reduce 処理となるジョブの性能を向上させる手法を提案する.

5.2. 提案手法の実装

本稿では, 著名なオープンソースファイルシステムである ext2/ext3/ext4 を用いて提案手法を実装した. これらのファイルシステムでは, ディスクは 4KB のブロックを単位に管理され, 複数のブロックを集めてブロックグループを構成する. そして, 各ブロックグループ用にブロックビットマップ, inode ビットマップ, inode テーブルが用意されている. ブロックビットマップはブロックグループ内の各ブロックが使用中であるか未使用であるかを管理するビットマップであり, inode ビットマップは各 inode 番号が使用中であるか否かを管理するビットマップであり, inode テーブルは各ファイルの inode 情報(ファイルの保存位置, アクセス権限, 更新日時など)を管理している.

本稿の実装では, 上記ファイルシステムのデータブロックビットマップの未使用ビットのうち, 低アドレス部以外のビットを 1 とし(使用中とし), 高アドレス部にファイルが配置されることを回避する.

6. 性能評価

本章にて, 提案手法の性能を評価する.

6.1. 測定方法

通常状態と提案手法適用状態で TeraSort を実行し, 両状態における性能を比較した. 測定は 10 回ずつ実行した. 測定環境は, 入力データサイズが 4GB, 8GB, 16GB, 32GB の 4 パターンで行うこと以外は, 3 章と同様である.

また, 上記の入力データサイズ 16GB の測定環境の内, HDFS ブロックサイズを 64MB から 128MB に変更した場合についても, 同様の方法で性能の比較を行った. 測定環境は, HDFS ブロックサイズ以外は 3 章と同様である.

ここで, 通常状態とは Hadoop データ配置用 HDD の全ブロックが空き領域の状態のことを表し, 提案手法適用状態とは提案手法により外周部 60GB 以外へのファイルの配置を禁止している状況のことを表す.

6.2. 測定結果

入力データサイズ 4GB, 8GB, 16GB, 32GB, HDFS ブロックサイズ 64MB における通常状態と提案手法適用状態における性能を図 12～図 15 に示す. そして, 入力データサイズ 16GB, HDFS ブロックサイズ 128MB における性能を図 16 に示す. 各グラフの横軸は通常状態か提案手法適用状態かを表し, 縦軸は平均実行時間[sec]

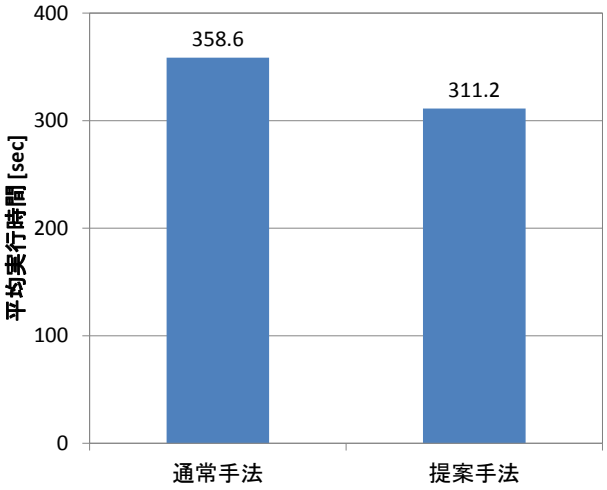

図 12. 平均実行時間(入力データサイズ 4GB, HDFS ブロックサイズ 64MB)

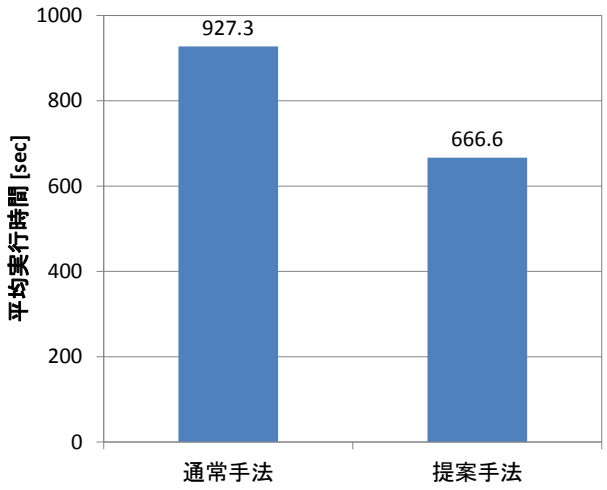

図 13. 平均実行時間(入力データサイズ 8GB, HDFS ブロックサイズ 64MB)

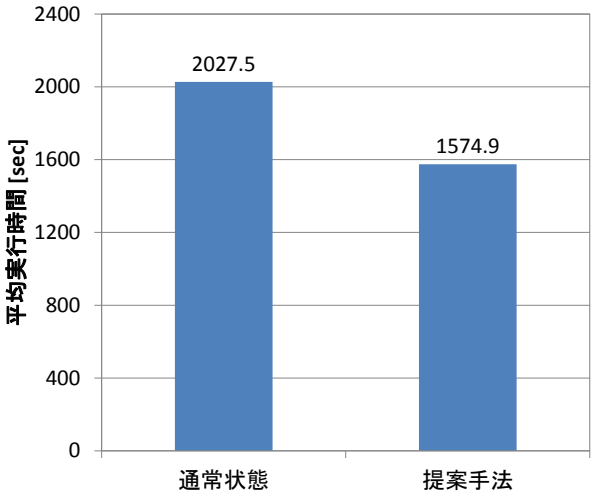

図 14. 平均実行時間(入力データサイズ 16GB, HDFS ブロックサイズ 64MB)

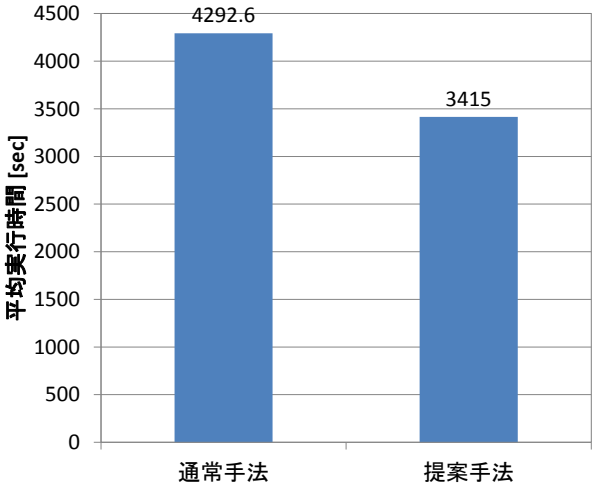

図 15. 平均実行時間(入力データサイズ 32GB, HDFS ブロックサイズ 64MB)

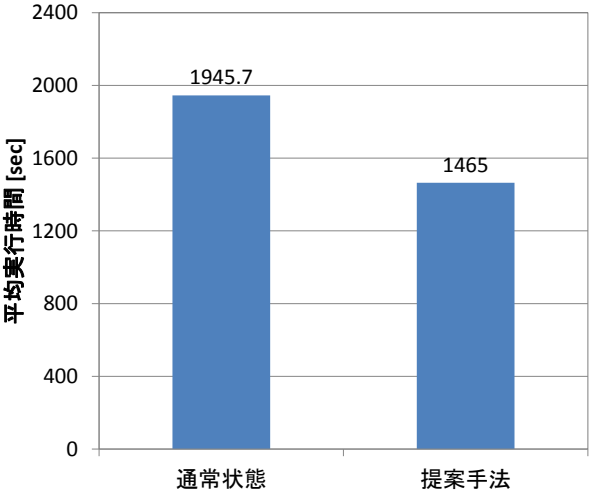

図 16. 平均実行時間(入力データサイズ 16GB, HDFS ブロックサイズ 128MB)

を示す．図 12 では提案手法の平均実行時間は通常状態の平均実行時間よりも 13.2%，図 13 では 28.1%，図 14 では 22.3%，図 15 では 20.4%，図 16 では 24.7%短縮できたことが確認できる．また，各測定における実行時間の分布を図 17～図 26 に示す．図 17 と図 18，図 19 と図 20，図 21 と図 22，図 23 と図 24，図 25 と図 26 をそれぞれ比較すると，通常状態では性能が高い場合と低い場合が混在するが，提案手法では常に高い性能で安定していることが分かる．これにより，平均性能においては提案手法が通常手法を大きく上回る結果となっている．これは，通常状態ではファイルがディスク外周部(高性能部)に配置される場合と内周部(低性能部)に配置される場合の両方があるが，提案手法では必ず外周部に配置されることが原因である．

また，図 12～図 16 の全ての評価において提案手法の性能が通常手法の性能を上回っており，提案手法の優位性は入力データサイズおよび HDFS ブロックサイズに依ないことも確認できる．

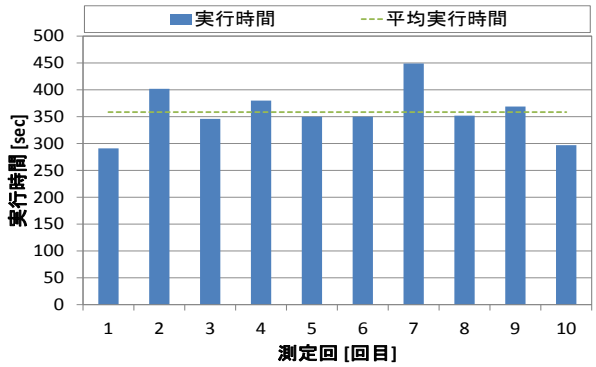

図 17. 実行時間分布(入力データサイズ 4GB, HDFS ブロックサイズ 64MB 通常手法)

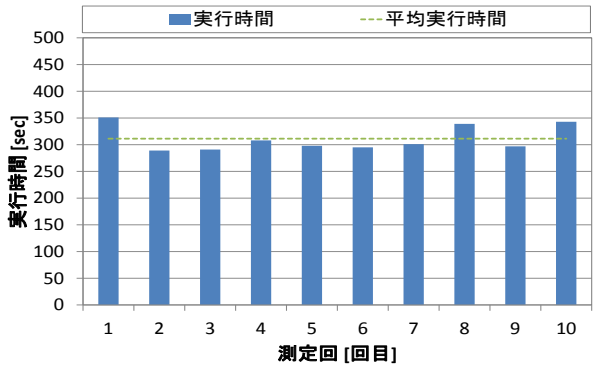

図 18. 実行時間分布(入力データサイズ 4GB, HDFS ブロックサイズ 64MB, 提案手法)

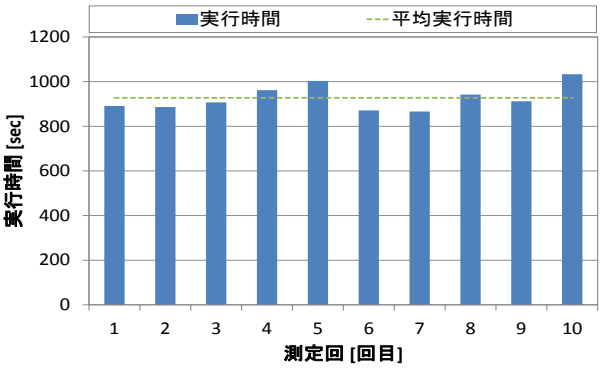

図 19. 実行時間分布(入力データサイズ 8GB, HDFS ブロックサイズ 64MB, 通常手法)

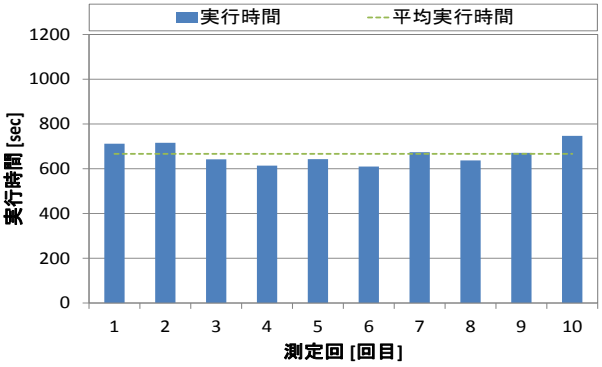

図 20. 実行時間分布(入力データサイズ 8GB, HDFS ブロックサイズ 64MB, 提案手法)

Vol.2015-DPS-163 No.17
Vol.2015-MBL-75 No.17
2015/5/28

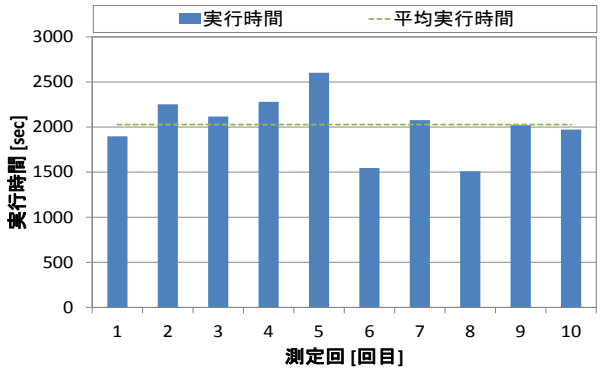

図 21. 実行時間分布(入力データサイズ 16GB, HDFS ブロックサイズ 64MB, 通常手法)

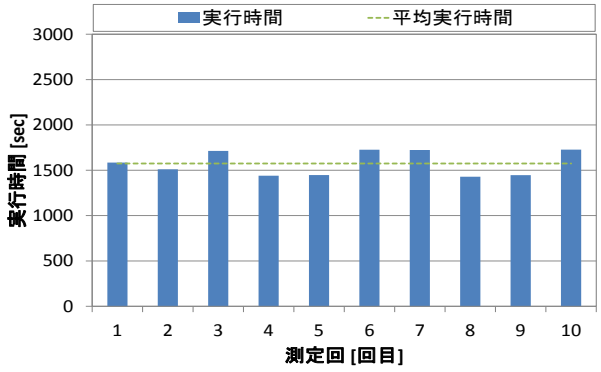

図 22. 実行時間分布(入力データサイズ 16GB, HDFS ブロックサイズ 64MB, 提案手法)

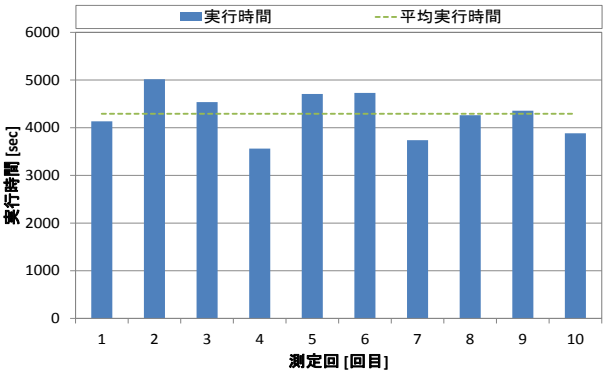

図 23. 実行時間分布(入力データサイズ 32GB, HDFS ブロックサイズ 64MB, 通常手法)

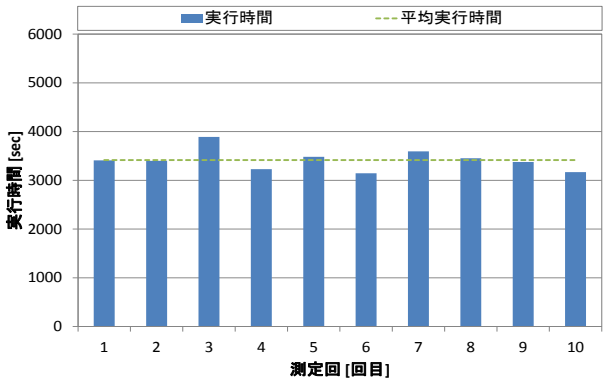

図 24. 実行時間分布(入力データサイズ 32GB, HDFS ブロックサイズ 64MB, 提案手法)

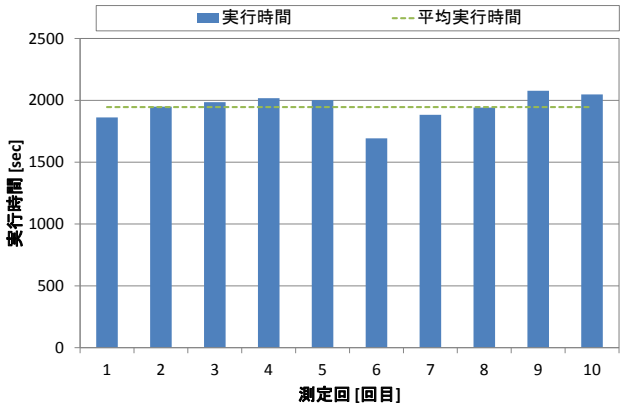

図 25. 実行時間分布(入力データサイズ 16GB, HDFS ブロックサイズ 128MB, 通常手法)

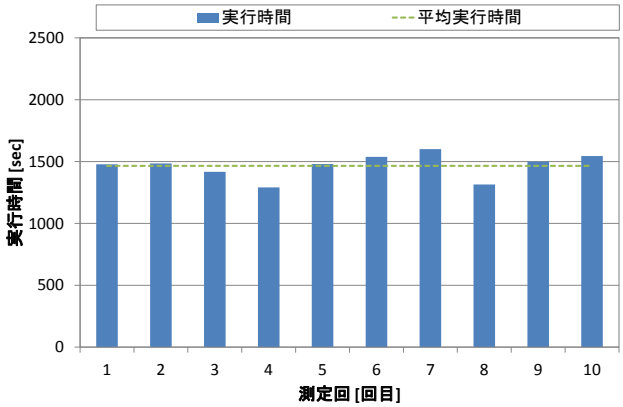

図 26. 実行時間分布(入力データサイズ 16GB, HDFS ブロックサイズ 128MB, 提案手法)

7. 関連研究

小沢らは, Hadoop の WordCount ジョブにて単一の Reduce 処理が I/O バウンドになることに着目し, データの圧縮によりその性能を向上させる手法を提案している[1]. また, 性能評価にて提案手法の有効性を示している. 単一 Reduce 処理の I/O 性能向上によるジョブ実行時間の短縮を目指す点において本研究と類似しているが, 小沢らの研究は HDD の特性には着目しておらず目標達成の方法は異なっている. また, 小沢らの手法と本稿の提案手法は排他的な関係にはないため, 両手法を併せて適用することにより両研究を補完できる関係にあると言える.

文献[2]において, ファイルシステムのデータブロックビットマップに着目してアプリケーション性能を向上させる手法が提案されているが, シーケンシャル I/O 性能の向上や, それによる Hadoop ジョブの性能向上には言及されておらず, 本稿とは研究の目的が大きく異なっている.

8. まとめ

本研究では, Hadoop MapReduce の単一 Reduce 処理となるジョブに着目し, そのボトルネックを示し, 性能向上手法の提案と性能評価による有効性の検証を行った. 具体的には, 単一 Reduce 処理である TeraSort の例を用いてその性能のボトルネックが単一 Reduce

Vol.2015-DPS-163 No.17
Vol.2015-MBL-75 No.17
2015/5/28

処理ノードの I/O 処理にあることを示し，そしてその I/O 処理が主にシーケンシャル I/O であることを示した．続いて，HDD のシーケンシャル I/O 速度がゾーンごとに異なること，ファイルの格納位置はファイルシステムが決定するが近年のファイルシステムはファイルのアクセス手法に関する情報を保持しておらず必ずしも適した位置にファイルを格納しないことに着目し，ファイルシステムのディスク使用状況管理データ(データブロックビットマップ)の修正によるファイル格納位置制御手法およびそれによる単一 Reduce 処理となるジョブの性能向上手法を提案した．そして，性能評価により通常手法においては必ずしも高い性能が得られないが，提案手法においては安定して高い性能が得られ，提案手法が有効であることを示した．

今後は，巨大なデータを扱うことに適したファイルシステムである xfs に着目し，今回の提案手法の適応について考察していく予定である．

参考文献

[1] 小沢健史, 及川一樹, 鬼塚真, 本庄利守 “列指向バッファ管理を用いた MapReduce の高速化”, DEIM Forum 2014 D1-3 (2014).
[2] 古野雄太・山口実靖, “ファイルシステムのデータレイアウト制御による I/O 性能の向上”, 電子情報通信学会 2015 年総合大会, D-4-27

謝辞 本研究は JSPS 科研費 25280022, 26730040 の助成を受けたものである．