Vol.2015-DBS-162 No.20
2015/11/26

ファイル格納位置の動的制御による Hadoop MapReduce ジョブの性能向上

藤島永太† 山口実靖†

膨大な情報を処理する方法として，Hadoop MapReduce が注目されている．MapReduce 処理では，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において，Reduce 処理が I/O バウンドとなる場合がある．本稿では，単一 Reduce 処理となる MapReduce ジョブのボトルネックが Reduce 処理ノードの I/O 処理にあること，その I/O 処理が主にシーケンシャルアクセスであることを示す．そして，ファイルシステムのファイル格納位置の制御によるシーケンシャルアクセス性能向上手法を改善し，格納位置を動的に制御する手法を提案する．また，通常手法，既存の静的制御手法，動的制御手法の性能評価によりその有効性を示す．

Improving the Performance of Hadoop MapReduce Jobs with the Dynamic Control of File Storage Location

EITA FUJISHIMA† SANEYASU YAMAGUCHI†

Hadoop MapReduce has been attracting attention as a method of processing an enormous amount of data. In cases of jobs wherein all the output files of all the relevant Map tasks are transmitted and consolidated into a single Reduce task, such as in TeraSort, the single Reduce task is the bottleneck task and is I/O bounded for processing many large output files. In this paper, we indicate that the bottleneck of MapReduce job to be single Reduce processing is in the I/O processing of Reduce processing node and that the I/O processing is mainly sequential access. Then, we propose to improve the existing method for increasing a sequential access performance by dynamic controlling of file location of filesystem. We present performance evaluation of the normal method, the existing static method, and the proposed dynamic controlling method, and then demonstrate that our method improves the performance.

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[1] 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静的制御により 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 速度を向上させる既存手法[2] を動的制御により改善し，評価によりその有効性を示す[a].

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 章で提案手法について考察を行い，9 章にて本稿をまとめる．

2. MapReduce 処理の流れ

MapReduce[3] 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

† 工学院大学大学院工学研究科電気・電子工学専攻
Electrical Engineering and Electronics, Kogakuin University Graduate School.

a 本稿は，文献[2] を元に発展させたものである．

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[4] [5] ．

このような単一 Reduce 処理となってしまう MapReduce ジョブの例として，TeraSort などのソート処理，WordCount などの集計処理が挙げられる．これらのジョブでは，最終的な結果を出力する Reduce タスクが単一のノードにまとめられるため，負荷が分散されずに集中してしまう．そして, この単一 Reduce 処理ノード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

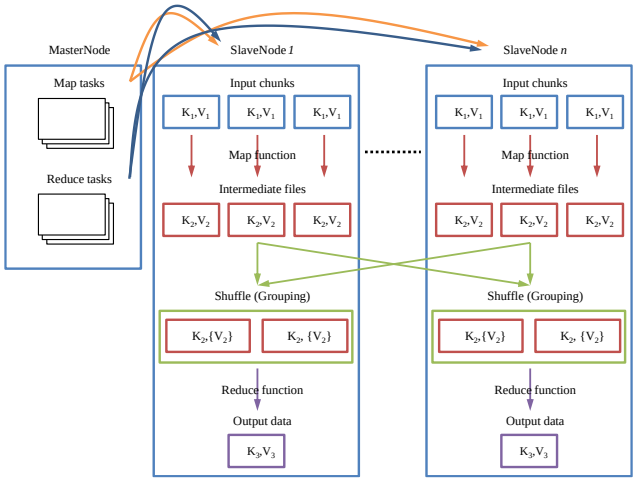


図 1. MapReduce 処理の流れ

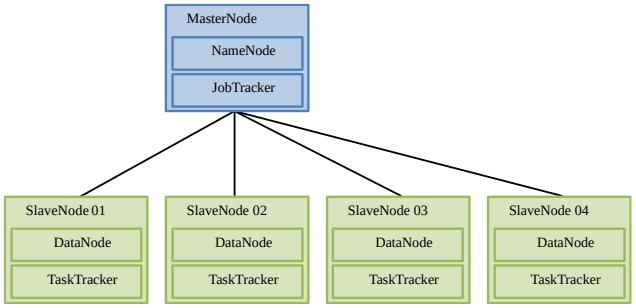


図 2. 測定用 Hadoop クラスタ概略図

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 ブロックサイズはデフォルトの 64 MB，複製数は 3 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 ext3 ファイルシステムで動作する 500GB の HDD 内におかれ，本 HDD の仕様の詳細は表 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 TeraSort を開始してからの経過時間[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，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 処理ノードにおける I/O 処理は，主に Map 処理

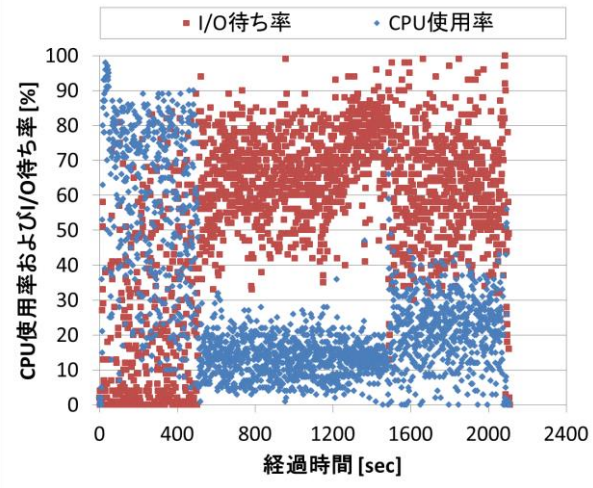


図 3. 単一 Reduce 処理ノードの CPU 使用率および I/O 待ち率

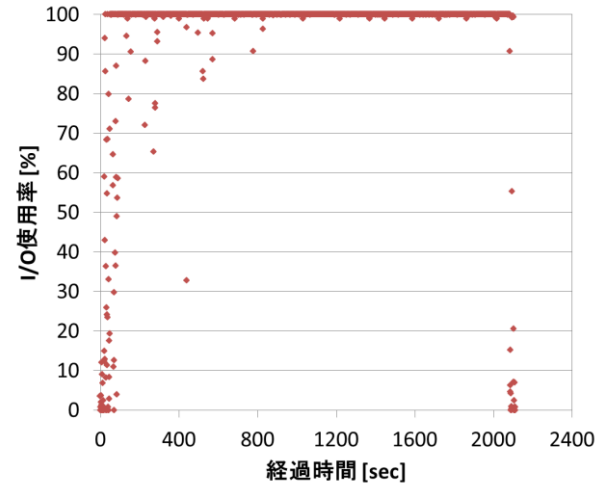


図 4. 単一 Reduce 処理ノードの I/O 使用率

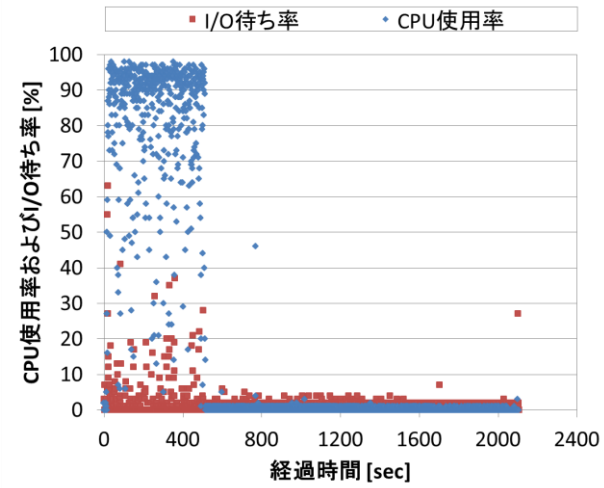


図 5. 非 Reduce 処理ノードの CPU 使用率および I/O 待ち率

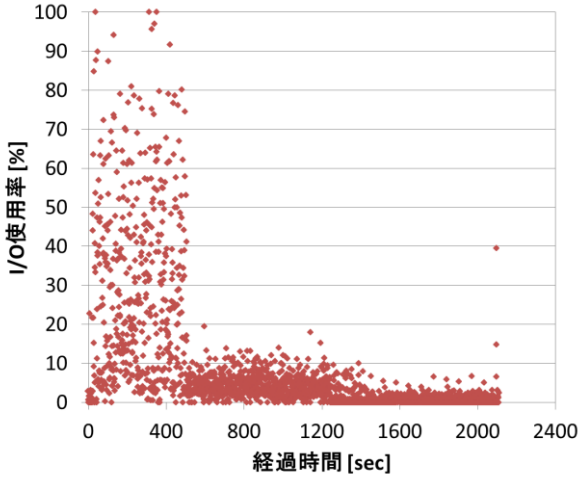


図 6. 非 Reduce 処理ノードの I/O 使用率

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“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 要求であると見なし，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．この“結合 I/O サイズ”の大小は，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．

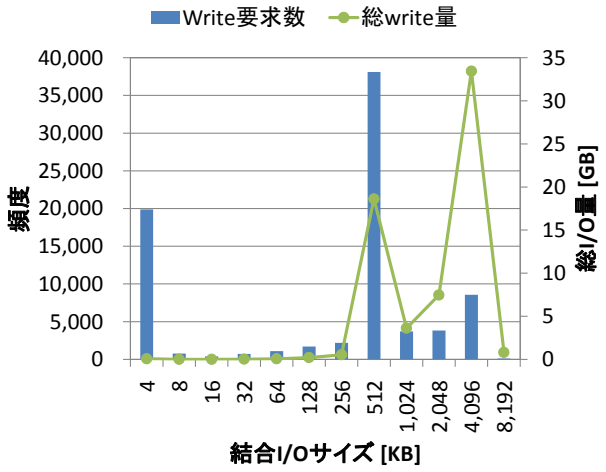
図より，Reduce 処理中には大きなサイズの I/O 要求が多く発行されており，Reduce 処理中は高いシーケンシャル性で HDD が動作していることが分かる．よって，本ジョブの性能を向上させるにはシーケンシャル I/O 速度の向上が重要であると予想できる．

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し，アプリケーション実行中の I/O について考察する．

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



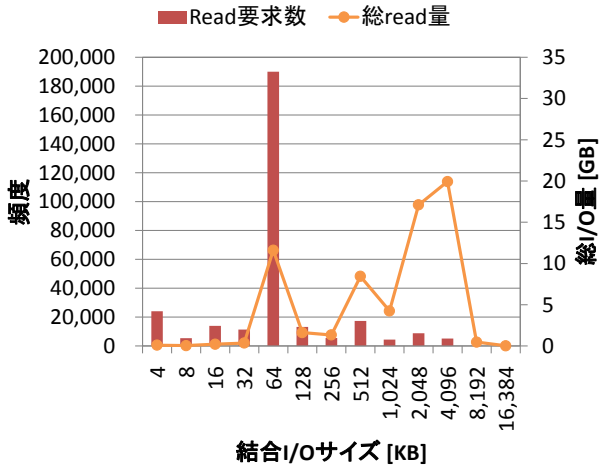


図 7. Reduce 処理ノードの結合 I/O サイズ頻度分布

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中から 1 台を使用した. 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，最初のゾーンでの書き込みでは平均 0.343 秒，最後のゾーンでの読み込みでは平均 0.667 秒かかっている．

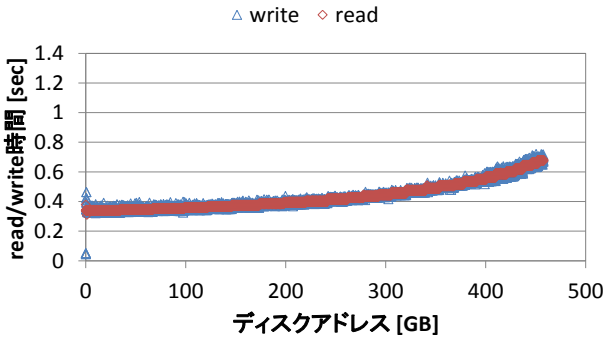


図 8. HDD のゾーン毎のシーケンシャル I/O 速度（ファイルシステム無し）

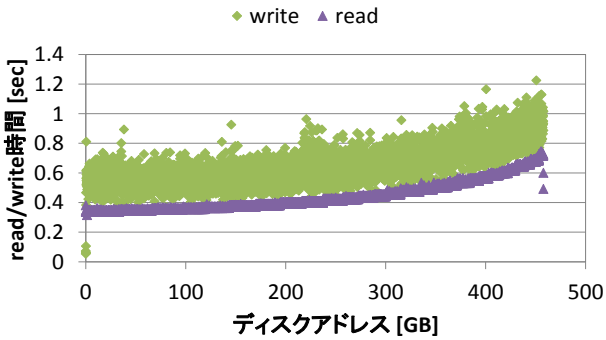


図 9. HDD のゾーン毎のシーケンシャル I/O 速度（ext2）

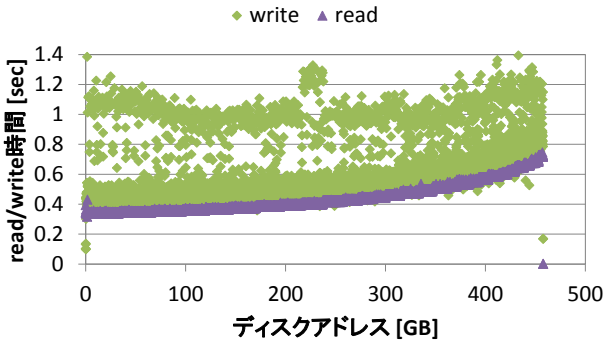


図 10. HDD のゾーン毎のシーケンシャル I/O 速度（ext3）

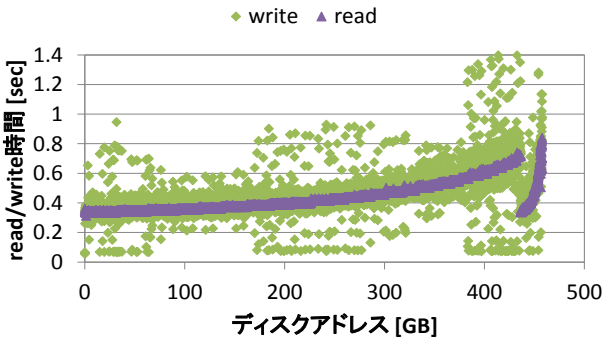


図 11. HDD のゾーン毎のシーケンシャル I/O 速度（ext4）

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．

5. 提案手法

本章では，既存手法であるファイル格納位置の静的制御［2］，本稿で提案するファイル格納位置の動的制御，および両手法の実装について述べる．

5.1. ファイル格納位置の静的制御

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，4 章の測定結果より，ディスクの外周側のゾーンにファイルが配置されると実行時間が短くなり，逆に内周側のゾーンに配置されると実行時間が長くなると予想される．

よって，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，当手法ではファイルシステムの静的な制御によりこれを実現する．

5.2. 静的制御手法の実装

静的制御手法の実装は，著名なオープンソースファイルシステムである ext2/3 を用いて行った．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，ブロックグループ毎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データ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．

本稿の実装は，上記ファイルシステムの各ブロックグループのデータブロックビットマップのうち，ディスクの外周に相当する低アドレス部以外のブロックグループのビットを使用中ビットへ変更し，内周に相当する高アドレス部にファイルが配置されることを回避する．本実装では，ジョブ実行前にジョブが使用するディスク容量を予測し，先頭ブロックグループからその容量に相当する範囲のブロックグループのみを使用可能領域，それ以外の範囲のブロックグループを使用禁止領域へと静的に変更する．ジョブ実行中に使用可能領域の変更は行わないため，静的な制御手法であるといえる．

5.3. ファイル格納位置の動的制御

上記の静的制御手法の実装は，ジョブ実行中に使用可能領域の拡大や縮小は行わない．例えば，HDFS の複製数が 3 である状態で，入力データサイズ 16 GB の TeraSort を実行した場合，単一 Reduce 処理ノードでは入力データの保存に約 12 GB，Map 処理に約 4 GB，他ノードから転送されてきた中間データの保存に約 12 GB，出力データの保存に約 16 GB の容量が必要となる．よって，最小でも約 44 GB のディスク領域を使用可能とする必要がある．静的制御手法では，時間的に変化する使用ディスク領域の最大サイズにあわせて使用可能領域を設定する必要があり，ジョブ実行中に使用するディスク領域が少ない時間帯は必要以上に大きな領域が空き状態となる．この場合，空き領域内で後方(相対的に性能が低い領域)にファイルが配置されてしまう可能性や，ファイル同士が離れて配置されてしまいシーク距離とシーク時間を増加させてしまう可能性が残されており，さらなる改善の余地があると考えられる．

本節では，上記のような静的制御手法の欠点を排除し，ディスクの外周部の高速なシーケンシャル I/O 速度をより効率良く利用するために，各時点で必要とされる容量のみを使用可能とし，必要量の増加に応じて使用可能領域を動的に拡張するファイル格納位置の動的制御手法を提案する．

5.4. 動的制御手法の実装

動的制御手法の実装は，前述の静的制御手法の実装に定期的に使用可能領域量を監視する機能と，使用可能領域の容量が閾値を下回ったときに使用可能領域を動的に拡張する機能を追加したものとなっている．使用可能領域容量監視機能は定期的にファイルシステムの空きブロック数(本稿の実装で用いた ext2/3 において `s_free_blocks_count`)を調査し，提案手法が使用禁止としているブロック数から使用可能容量を算出する機能である．使用可能領域動的拡張機能は，監視機能により使用可能領域の容量がある閾値を下回った場合に起動され，使用禁止領域内のブロックをディスクの外周側から順に使用可能領域の容量が閾

値を上回るまで使用可能状態へと変更していく機能である．監視機能および動的拡張機能は独立したスレッドを持ち，Hadoop MapReduce ジョブと並列で動作する．

これらによりファイルが不必要にディスクの内周側に配置される可能性が排除され，静的制御手法よりもディスクの最外周を効率的に使用するようになる．また副次的な効果として，ディスクを低アドレス部から高アドレス部に順に使用させ，かつディスクの空き容量が少ない状況に保つことによりファイルが連続配置されることが増え，シーケンシャル性の向上(結合 I/O サイズの増加)が期待できる．

6. 性能評価

本章にて，前章で述べたファイル格納位置の静的制御手法，動的制御手法の性能を評価する．

6.1. 測定方法

通常手法, 静的制御手法, および動的制御手法で TeraSort を実行し, 各手法の性能を比較した. 測定は 10 回ずつ行い，平均実行時間を比較した．測定環境は，入力データサイズを 16 GB，HDFS ブロックサイズを 64 MB または 128 MB とし，それ以外の設定は 3 章と同様とした．

静的制御手法での測定は，Hadoop データ配置用 HDD の全データブロックが空き領域の状態で実行し，静的制御手法によりディスクの外周部 60GB 以外へのファイルの配置を禁止している状況で行った．

動的制御手法の定期監視間隔は 5 秒とし，動的拡張の閾値は 5 GB とした．

6.2. 測定結果

HDFS ブロックサイズが 64 MB で，入力データサイズが 16 GB の TeraSort の平均実行時間を図 12 に示す．また，HDFS ブロックサイズが 128 MB で，入力データサイズが 16 GB の TeraSort の平均実行時間を図 13 に示す. 各グラフの横軸は手法を表し，縦軸は平均実行時間[sec]を示す．図 12 では，静的制御手法の平均実行時間は通常手法よりも 22.3%，動的制御手法の平均実行時間は通常状態の平均実行時間よりも 41.2%, 図 13 では静的制御手法の平均実行時間は 24.7%，動的制御手法の平均実行時間は通常手法よりも 28.6%短縮できたことが確認できる．

図 12 において，各手法で 10 回ずつ測定を行った各測定の実行時間を図 14 に示す．図 14 より，通常状態では性能が高い場合と低い場合が存在するが，両提案手法では常に性能が高いことが分かる．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．

また静的手法と動的手法を比較すると，動的手法の実行時間の分散がより小さく，かつ全ての計測において動的手法の実行時間が短くなっていることが分かる．このことから，使用可能領域内の後方に配置される可能性の排除と，シーク距離の削減が効果的に機能していることが分かる．

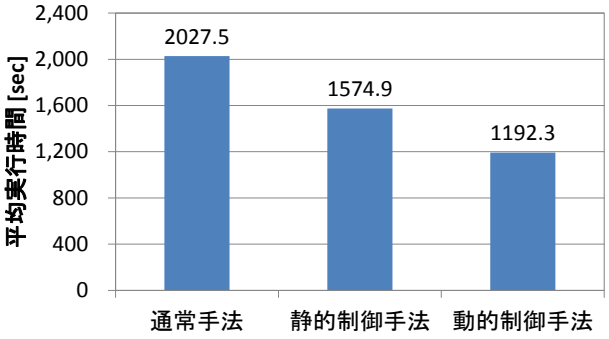


図 12. 平均実行時間（入力データサイズ 16 GB, HDFS ブロックサイズ 64 MB）

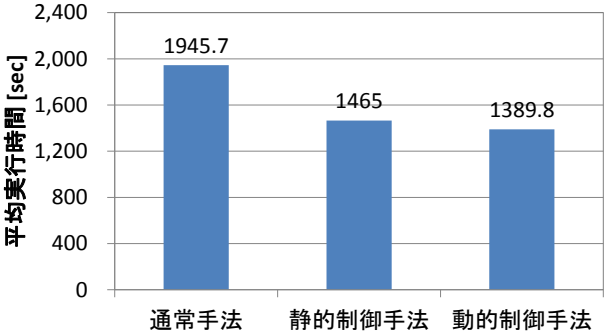


図 13. 平均実行時間（入力データサイズ 16 GB, HDFS ブロックサイズ 128 MB）

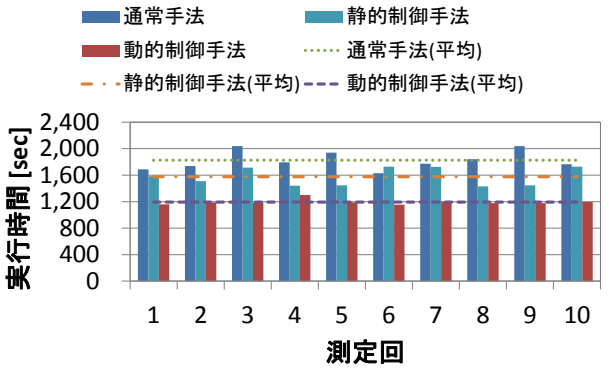


図 14. 実行時間分布（入力データサイズ 16 GB, HDFS ブロックサイズ 64 MB）

以上の結果より，静的制御手法よりも動的制御手法の方が安定した高い性能を得ることが可能であり，ファイル格納位置の動的制御手法の有効性が確認できる．

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[4] [5] ．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[6] [7] 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．

Dremel[8] , Impala[9] , Camdoop[10] や, Li らの研究[11]にて，大規模データ処理における効率的な問い合わせの集約方法が提案されている．しかし，これらの手法は HDD のゾーンやゾーンごとのシーケンシャル I/O 速度の違いには着目しておらず，これらの手法と我々の提案手法はアプローチが異なっている．

I/O スケジューラの研究として，HDD における I/O 速度の向上手法[12] [13] や，フラッシュメモリにおける I/O 速度向上手法[14] が提案されている．これらの研究において，シーケンシャル I/O を中断することが性能の劣化につながることなどが考察されているが，ファイル配置はファイルシステムが行うことを前提としており, シーケンシャル I/O 速度の向上手法については考察されていない．

8. 考察

初めに，動的制御手法の負荷について考察する．動的制御手法では，静的制御手法と比較して定期的な使用可能領域容量監視の負荷と，動的使用可能容量拡張の負荷が増えている．監視プロセスが定期的に調査する情報はファイルシステムの空きブロック数情報であり，ファイルの作成や削除が頻繁に行われている状況においてこの情報は常に OS のページキャッシュに格納されている．よってこの情報の取得は, I/O 負荷量を増やすことはなく I/O バウンドのジョブの性能への影響は極めて小さい．また，監視は 5 秒に 1 回行っており，監視プロセスの CPU 使用率は平均 0.03%であった．よって，I/O バウンドのジョブ実行中においてはこの CPU 消費も性能にほとんど影響を与えない. 動的拡張処理ではブロックビットマップ領域へのアクセス命令を発効することになるが，書き込み命令であるため OS の遅延書き込みが機能し即時にディスクアクセスを発生させることはない．また，多くの場合では当該拡張領域は拡張の後に頻繁にアクセスされるため，ブロックビットマップへの再書き込み（上書き）が多く発生することになり，本拡張がディスクアクセス数を増加させることはないと考えられる．

次に，監視間隔と拡張起動閾値の関係について考察する．1 回監視が行われると，次の監視が実行されるまでの間は使用可能領域の拡張は行われない．この間に使用可能領域を全て使用してしまうと，ディスクに空き領域があるにも可関わらず使用可能ブロックを確保できない状態が発生するが，これは以下の方法により回避することができる．書き込みキャッシュが存在しない環境では,「*ディスクの書き込み速度×監視間隔* 」以上のディスク容量を監視間隔の間に使用することはできず,「*拡張起動閾値＞ディスク書き込み速度×監視間隔*」を満たすよう拡張起動閾値を定めれば，この発生を回避できる．書き込みキャッシュがある状況では，書き込みキャッシュの容量は極めて短い時間で（メモリ書き込みに要する時間で）使用することが可能である．よって監視間隔の間に書き込める量はディスク書き込みとキャッシュ書き込みの合計となり,「*拡張起動閾値＞ディスク書き込み速度×監視間隔*＋書き込みキャッシュサイズ」を満たせばブロックを確保できないことの発生を回避できる．本稿の実験の例では，書き込みキャッシュサイズは最大で実メモリサイズ（2 GB）であり, HDD 速度は約 200 MB/秒（4 章参照）, 監視間隔は 5 秒, 起動閾値が 5 GB であり，これが満たされている．拡張起動閾値はチューニングパラメータであり，アプリケーションのディスク使用動作を事前に把握している状況では拡張起動閾値を削減してさらなる性能向上が可能となる．

本手法の実現には，ファイルシステムのブロック確保位置の制御が必要となる．本稿では ext2/3 ファイルシステムを用いた実装を紹介したが，これは両ファイルシステムではブロック使用管理ビットマップの書き換えによりこれが実現できるからである．ext4 を用いて実装する場合には，これまでの ext2/3 と同等のブロックの使用管理方法の他に ext4 で導入されたエクステントに対応した実装をする必要があり，これにより実現が可能なる．他のファイルシステムでも，ファイル格納位置確保の制御が実現できれば提案手法は実現可能となる．また，ファイルシステムに依らずに，ディスクの外周側から内周側にかけて複数のパーティションに分け，必要に応じて LVM を用いて論理ボリュームを結合していくことで本提案手法と類似の手法が行えるかもしれない．上記方法ではパーティションによりファイル格納領域を複雑に区切るため, Hadoop が持つシーケンシャルアクセスの特性と定記録密度方式の HDD の特性から得られた本稿の考察を裏切る形になると考えられる．具体的には，パーティションサイズを超える大きさのファイルの格納場所の連続性が損なわれることが挙げられる．従って，巨大なファイル格納領域を保持したまま，ディスクの外周部から内周部まで無駄なく使用できるように，ファイルシステムの制御手法で実装を行った．

最後に，本手法適用時における利用可能な記憶容量について考察する．本稿で紹介した静的制御手法および提案した動的制御手法では，使用禁止に設定した空き領域を再び使用可能領域へと変更できるため，利用可能な記憶容量が

減少することはない．容量に余裕がある状態において高速領域を優先的に使用する方法であると言える．

9. おわりに

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，ファイル格納位置の動的制御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, その性能ボトルネックが単一 Reduce 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）の動的制御による単一 Reduce 処理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や既存のファイル格納位置の静的制御手法よりも，提案手法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．また，RAID0 使用時においても同様の提案手法が適用可能であると考えられるので，その環境における評価も行う．

謝辞 本研究は JSPS 科研費 25280022，26730040，15H02696 の助成を受けたものである．

本研究は，JST，CREST の支援を受けたものである．

参考文献

[1] Apache Hadoop, available from <http://wiki.apache.org/hadoop> (accessed 2015-07-01).

[2] 藤島永太, 山口実靖, “ファイル格納位置制御による Hadoop MapReduce ジョブの性能の向上”, FIT2015 RC-003 (2015).

[3] Jeffrey Dean, Sanjay Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters," OSDI'04: Sixth Symposium on Operating System Design and Implementation, (2004).

[4] 小沢健史, 鬼塚真, 福本佳史, 盛合敏, "集約処理を用いた MapReduce 最適化手法の提案と実装", 情報処理学会論文誌: コンピュータシステム, Vol. 6, No.3, pp.71-81, (Sep. 2013).

[5] 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).

[6] Masaya Yamada, Saneyasu Yamaguchi, "Filesystem Layout Reorganization in Virtualized Environment," The 9th IEEE International Conference on Autonomic and Trusted Computing (IEEE ATC 2012), ATC4-2, (2012).

[7] 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 (2015).

[8] Sergey Melnik, Andrey Gubarev, Jing Jing Long, Geoffrey Romer, Shiva Shivakumar, Matt Tolton, Theo Vassilakis, "Dremel: interactive analysis of web-scale datasets," Proc. VLDB Endow., Vol. 3 Issue 1-2, pp. 330-339, (2010).

[9] Cloudeare Impala, available from <http://www.cloudera.com/content/cloudera/en/products-and-services/cdh/impala.html> .

[10] Paolo Costa, Austin Donnelly, Antony Rowstron, Greg O'Shea., "Camdoop: exploiting in-network aggregation for big data applications," In Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation (NSDI'12), pp.3-3, (2012).

[11] Boduo Li, Edward Mazur, Yanlei Diao, Andrew McGregor, Prashant Shenoy, "A platform for scalable one-pass analytics using MapReduce," In Proceedings of the 2011 ACM SIGMOD International Conference on Management of data (SIGMOD '11), pp.985-996, (2011).

[12] Jens Axboe, "Linux block io - present and future," In Proceedings of the Ottawa Linux Symposium, pp.51-61, Ottawa Linux Symposium, (2004).

[13] Sitaram Iyer, Peter Druschel, "Anticipatory scheduling: A disk scheduling framework to overcome deceptive idleness in synchronous I/O," In Proceedings of the eighteenth ACM symposium on Operating systems principles (SOSP '01), (2001).

[14] Yuta Nakamura, Shun Nomura, Kyosuke Nagata, Saneyasu Yamaguchi, "I/O Scheduling in Android Devices with Flash Storage," 8th International Conference on Ubiquitous Information Management and Communication ACM IMCOM (ICUIMC), (2014).