Vol.2019-OS-146 No.13
2019/5/31

特性を考慮した Hadoop ジョブの I/O 性能の向上

中上誠[†1]　Jose A. B. Fortes[†2]　山口実靖[†1]

概要：Hadoop は ビッグデータ処理などで用いられる重要なプラットフォームとなっている．ビッグデータ処理は，大容量ストレージとして HDD を用いることも多い．Hadoop のジョブにはそれぞれ，使用資源(CPU や HDD)やストレージアクセス方法(シーケンシャルやランダム)に特徴があり，これら特徴を考慮してデータをストレージに配置することによりジョブの実行時間の短縮が可能であると予想される．本稿では，ジョブの特徴を考慮したストレージ内ファイル格納場所最適化手法を提案する．そして，性能評価によりその有効性を示す．

キーワード：Hadoop，MapReduce，Filesystem，SWIM

1. はじめに

Hadoop は MapReduce モデルに基づいた著名なビッグデータ処理プラットフォームである[1]．ビックデータ処理には大規模ストレージとしてハードディスクドライブ（HDD)を用いることも多い．HDD は格納位置によりアクセス性能が異なるため，ジョブの特性を考慮し HDD におけるファイルの格納位置を制御することにより，Hadoop の性能を向上させることができると考えられる．多くの場合 Hadoop は，HDD などの大容量記憶装置に格納された大規模データセットの分析に使用され，HDD にシーケンシャルにアクセスする[2]．そして，HDD においてシーケンシャルアクセス速度の高い位置を積極的に活用することによってシーケンシャルアクセス速度を向上させる手法が過去に提案されている[2]．この手法は，I/O バウンドの Hadoop ジョブのパフォーマンスを向上させることができる．

本稿では，ジョブの特性を考慮してファイル格納位置を制御することにより Hadoop の I/O 性能を向上させる手法を提案し，Statistical Workload Injector MapReduce（SWIM）を用いてその性能を評価する．SWIM は Hadoop パフォーマンスの現実的な実験的評価を可能にするために提案されたベンチマークアプリケーションである[3]．SWIM は多くの種類のワークロードをエミュレートすることができる[1]．

本稿ではまず，Map-heavy，Shuffle-heavy，Reduce-heavy の SWIM ジョブに着目し，それらの特性を明らかにするために I/O 使用率と CPU 使用率を調査する．そしてこの調査により，Map-heavy が CPU バウンド，Shuffle-heavy および Reduce-heavy が I/O バウンドであることを示す．次に，これら 3 種類のジョブが順次実行されるケースおけるファイル格納位置の最適化について説明する．そして，通常手法，過去に提案されたファイル格納位置制御手法と，本稿で提案するファイル格納位置制御手法の性能評価を行う．

本稿の構成は以下の通りである．2 章で関連研究を紹介する．3 章で，SWIM ジョブの基礎性能の調査，および HDD におけるファイル格納位置とシーケンシャルアクセス速度の関係について説明する．4 章で，複数種類のジョブが順次実行される状況における Hadoop 性能向上手法を提案する．5 章で，3 つの手法の比較評価を行う．6 章にて考察を述べる．7 章にて本稿をまとめる．

2. 関連研究

2.1 MapReduce

図 1 に示すように，MapReduce ジョブは，Map フェーズ，Shuffle フェーズ，Reduce フェーズの 3 つのフェーズで構成されている．Map フェーズでは，入力ファイルが Input Split と呼ばれる小さなファイルに分割され，Mapper が Input split を受け取り，ユーザーが定義した Map タスクを実行して中間ファイルとなる Key-Value ペアを生成する．Shuffle フェーズでは，mapper が生成した中間ファイルがソートされ，Key ごとにグループ化されて reducer にファイルを転送される．Reduce フェーズでは，reducer が受け取った Key-Value ペアに対してユーザーが定義した Reduce タスクが実行され，出力ファイルが作成される．

2.2 SWIM

SWIM は，Hadoop を実行した際の履歴からそれを模擬した負荷を生成することができ，現実的な性能の調査が可能なベンチマークである．SWIM によって生成された負荷は Submit time seconds，Inter job submit gap seconds，Map input bytes，Shuffle bytes，Reduce output bytes の 5 つの変更可能なパラメータを持っている[4]．SWIM には Facebook システムの負荷を模擬した設定のファイルがあり，本稿ではこの Facebook トレースのパラメータを変えることによって得られるさまざまな Hadoop ジョブを用いて評価を行う．

2.3 ファイル格納位置制御手法

本節にて，既存のファイル格納位置動的制御手法[2]を紹介する．定記録密度方式（ZBR）の HDD では，一般に内側部のゾーンより外側部のゾーンの方がより高いシーケンシャルアクセス速度を持つ．本手法では，HDD の空き領域のうち最もシーケンシャルアクセス速度が高い位置（空き領域のうち最も外周部のゾーン）にファイルが格納されるように制御することによって，シーケンシャル I/O 性能を

ⓒ 2019 Information Processing Society of Japan

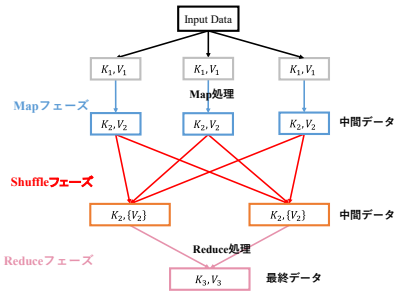

図 1　MapReduce の概要

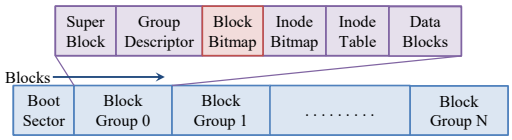

図 2　ext2/3 の概要

表 1　実験機の仕様

CPU	AMD Phenom 2 X4 965 Processor
OS	CentOS 6.10 x86_64 minimal
カーネル	Linux 2.6.32.57
メインメモリ	4GB
HDD	500GB(ext3)
Hadoop Ver.	2.0.0-cdh4.2.1

表 2　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	6.0Gbps
容量	500GB
バッファサイズ	32MB
回転数	7200rpm

向上させる．

この手法は，ext2/3 ファイルシステム[5]を用いて実行されている．これらのファイルシステムでは，図 2 のようにディスクは 4KB を 1 つのブロックとして管理され，複数のブロックを集めてブロックグループを構成する．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，データブロックが用意されている．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否かを管理しており，inode ビットマップは各 inode が使用中であるか否かを管理しており，inode テーブルは各ファイルの inode 情報(ファイルの格納位置，アクセス権限，更新日時など)を管理しており，データブロックにはファイルが格納される．この手法の実装は，ディスクの最外周部に相当する低アドレス部以外のブロックビットマップを使用中へと変更し，内周部にファイルが格納されることを回避する．また，この手法にはディスクの空き領域サイズを監視する機能，空き領域の容量が指定された閾値を下回ったときに空き領域を動的に拡張する機能，空き領域の容量が指定された閾値を上回ったときに空き領域を動的に縮小する機能がある．空き領域の動的拡張機能は，空き領域の監視機能により使用可能領域の容量が指定された閾値を下回ったときに起動され，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．空き領域の動的縮小機能は監視機能により使用可能領域の容量が指定された閾値を上回ったときに起動され，使用可能領域内のブロックをディスクの内周側から順に使用可能領域の容量が閾値を下回るまで使用禁止状態へと変更していく機能である．このファイル格納位置動的制御手法は MapReduce ジョブと並列で動作する．

2.4　ジョブ特性を考慮した格納位置制御

前節の格納位置制御手法は，I/O バウンドでありストレージにシーケンシャルにアクセスするジョブに対して特に効果的に機能する．すなわち，この手法の効果はジョブの特性に依存する．

文献[6][7]にて，MapReduce ジョブを Map-heavy, Shuffle-heavy，Reduce-heavy に分類し，それらのジョブの特性の調査を行っている．また，I/O バウンドである Shuffle-heavy ジョブにファイル格納位置制御手法を適用し，当該手法の有効性が示されている．また，文献[8]において，多種類のジョブが混在する環境におけるジョブの特性を考慮したファイル格納位置制御の考察が行われている．ただし，これら既存研究では各種類のジョブにおけるファイルの格納位置の最適化は行われていない．つまり，本稿における提案と異なり，各種類のジョブのファイルが最外周部以外に配置される可能性を排除していない．

3.　基礎性能評価

3.1　SWIM ジョブの振る舞い

本節にて，Map-heavy，Shuffle-heavy，Reduce-heavy 各ジョブの使用資源(CPU や HDD)の特性について述べる．具体的には，Map-heavy，Shuffle-heavy，Reduce-heavy ジョブの I/O 使用率，CPU 使用率，ディスク使用量の推移を調査し，その結果を示す．調査におけるパラメータは以下のように設定した．*Submit time* と *inter job submit gap* は，すべてのジョブで 1 と設定し，*Input file size* は 4GB とした．*Map input bytes* を3.0×10^{11}，*Shuffle bytes* と *Reduce output bytes* を 1 としたジョブを Map-heavy ジョブとし，*Shuffle bytes* を1.0×10^{12}，その他を 1 としたもの Shuffle-heavy ジョブとする．*Reduce output bytes* を1.0×10^{12}，その他を 1 としたものを Reduce-heavy ジョブとする．計算機および測定用 HDD の仕様をそれぞれ表 1 および表 2 に示す．Map-heavy ジョブ実行時の I/O 使用率と CPU 使用率を図 3 に，Shuffle-heavy ジョブ実行時と Reduce-heavy ジョブ実行時の I/O 使

ⓒ 2019 Information Processing Society of Japan

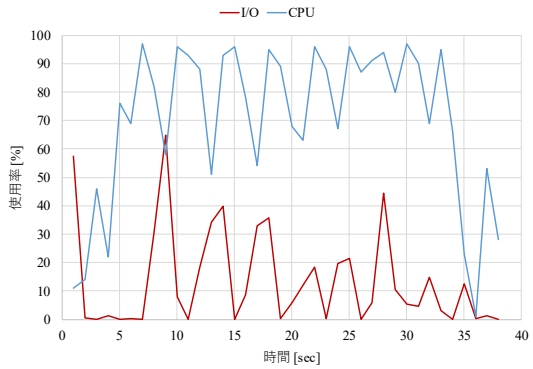

図 3　Map-heavy 実行時の I/O，CPU 使用率

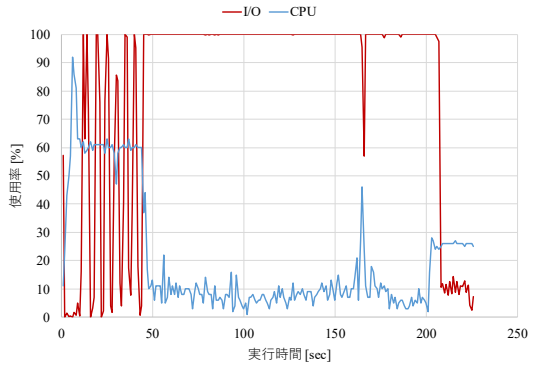

図 4　Shuffle-heavy 実行時の I/O，CPU 使用率

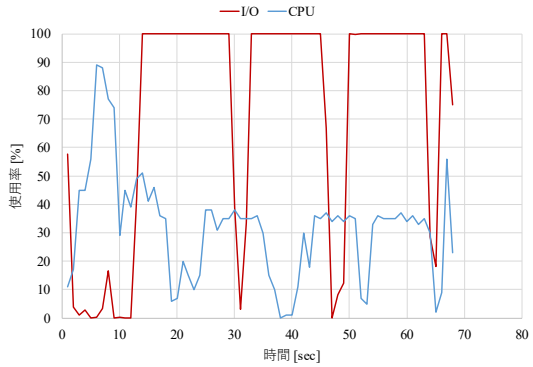

図 5　Reduce-heavy 実行時の I/O，CPU 使用率

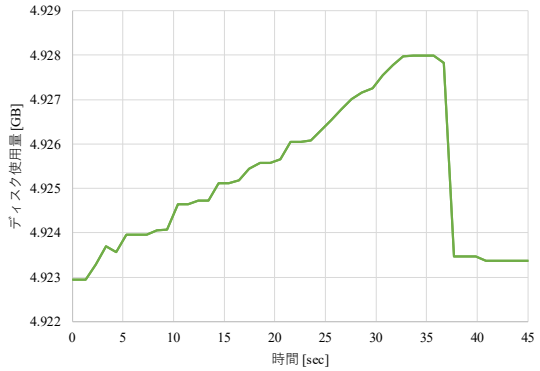

図 6　Map-heavy 実行中のディスク使用量の推移

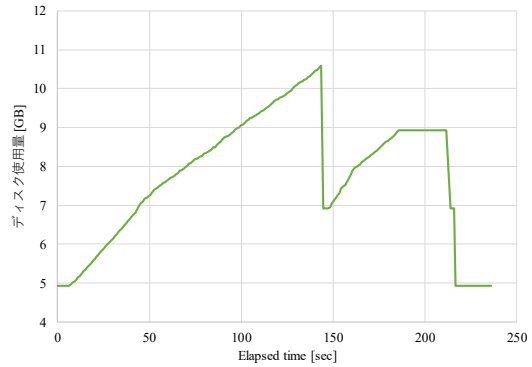

図 7　Shuffle-heavy 実行中のディスク使用量の推移

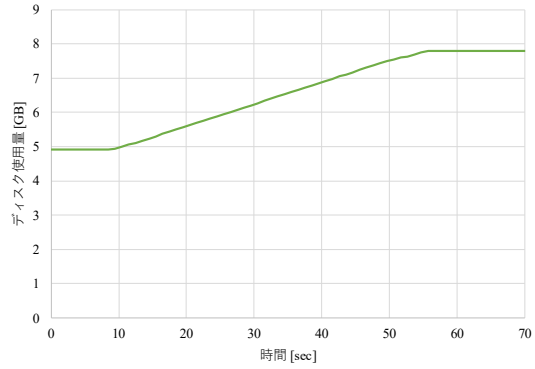

図 8　Reduce-heavy 実行中のディスク使用量の推移

用率と CPU 使用率を図 4，図 5 に示す．Map-heavy ジョブ実行時のディスク使用量の推移，Shuffle-heavy ジョブ実行時のディスク使用量の推移，Reduce-heavy ジョブ実行時のディスク使用量の推移を図 6，7，8 に示す．これらの結果から，Map-heavy ジョブは CPU バウンドであり，実行時に生成されるファイルが一時的ファイルであること，すなわち実行中に削除されることがわかる．Shuffle-heavy ジョブは I/O バウンドであり，実行中に生成されるファイルが一時的ファイルであることがわかる．これに対して，Reduce-heavy ジョブは I/O バウンドであり，実行時に生成されるファイルが恒久的に残ることがわかる．

3.2　シーケンシャルアクセス

測定用 HDD の最初のアドレスから最後のアドレスまで 64MB の読込/書込を行うプログラムを実行し，HDD の各ゾーンにおけるシーケンシャルリード/ライト速度を調査した．最初と最後のアドレスは，それぞれ最外側のゾーンと最内側のゾーンに対応する．図 9 にデバイスファイルに直接読込/書込したときのシーケンシャルリード/ライト速度を，図 10 に ext3 を介して読込/書込した時のシーケンシャルリード/ライト速度を示す．これらの図より，最外周部のゾーンの速度は最内周部のゾーンの速度の約 2 倍となっていることがわかる．

3.3　結合 I/O サイズ

ジョブ実行中の時間的かつ空間的に連続した I/O 要求を 1 つの要求として結合することによって得られる結合 I/O サイズ[10]について調査した．ジョブが I/O バウンドでありかつ結合 I/O サイズが大きい場合は，ファイルを外周部ゾーンに配置することにより I/O 性能を向上することが

ⓒ 2019 Information Processing Society of Japan

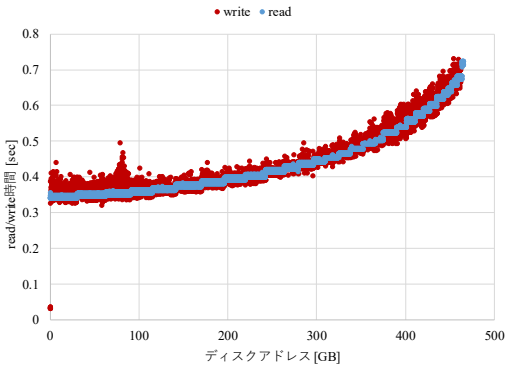

図 9　HDD のゾーン毎のシーケンシャル read/write 速度(デバイスファイル)

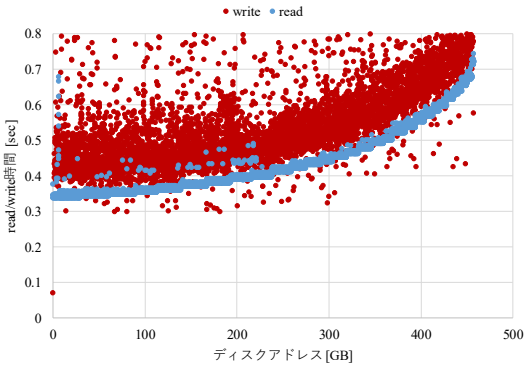

図 10　HDD のゾーン毎のシーケンシャル速度 read/write 速度(ext3)

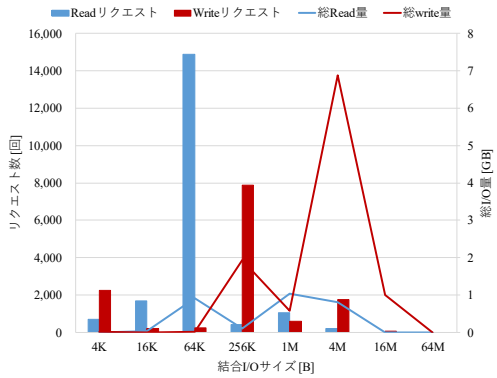

図 11　Shuffle-heavy ジョブの結合 I/O サイズ

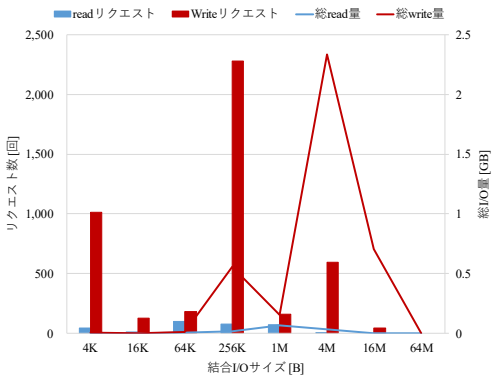

図 12　Reduce-heavy ジョブの結合 I/O サイズ

できると期待できる. Map-heavy ジョブは CPU バウンドであり，ファイル格納位置を最適化することによって性能を向上することはできないと予想される．本稿では，Shuffle-heavy ジョブと Reduce-heavy ジョブの結合 I/O サイズを調査した．図 11 と図 12 に，Shuffle-heavy ジョブと Reduce-heavy ジョブの結合 I/O サイズを示す. 図より, Shuffle-heavy ジョブ，Reduce-heavy ジョブともに結合 I/O サイズの大きい I/O 要求が多く発行されており，HDD が非常に高いシーケンシャル性でアクセスされていることがわかる．したがって，これらのジョブに HDD の外側部ゾーンを積極的に利用させることによってシーケンシャル I/O 速度を向上させることができると予想される．

4. 提案手法

4.1 ファイル配置

この節にて，ジョブの特性を考慮してファイルの格納位置を制御することによって，Hadoop ジョブの I/O 性能を向上させる手法を提案する．本稿では複数種類のジョブが同時ではなく順次実行される例を取り扱う. 提案手法では，次の優先順位に従ってファイルをディスク外周部に配置する．

（1）一時的ファイルであり，ジョブが I/O バウンドである
（2）一時的ファイルであり，ジョブが I/O バウンドでない
（3）一時的ファイルでなく，ジョブが I/O バウンドである
（4）一時的ファイルでなく，ジョブが I/O バウンドでない

HDD 外周部に一時的なファイルを格納する場合, 一時的なファイルが消えたあとに再度 HDD 外周部を活用することができる．提案手法では HDD 外周部を何度も活用するために，外周部ゾーンに恒久的ファイルが格納されないように制御する．

提案手法では HDD を 2 つの領域に分け，外周部にあたる領域に一時的ファイルを，内周部にあたる領域に恒久的ファイルを格納するように制御する．本稿では，2 つの領域のうちどちらの領域にファイルを格納させるかのみを制御する手法を提案手法 1 とする．この手法は文献[8]にて提案されたものと同一である．これに対して，2 つの領域のうちどちらの領域にファイルを格納させるかを制御するのに加え，それぞれの領域の中において最外周部から順にファイルを格納させるように制御する手法を提案手法 2 とする．

4.2 提案手法の実装

提案手法の実装は，ext2/3 を用いておこなった．2 章で述べたように，ext2/3 ファイルシステムではディスクは 4KB を 1 つのブロックとして管理され，複数のブロックを集めてブロックグループが構成される．そして，ブロックグループ毎に block bitmap，inode bitmap，inode table，data block が用意されている．

提案手法においては, Map-heavy ジョブと shuffle-heavy

ⓒ 2019 Information Processing Society of Japan

ジョブが実行される際にはディスク最外周部以外のブロックグループの block bit map を全て 1 に書き換え使用中とし，強制的にディスク最外周部にファイルが格納されるようにする．Reduce-heavy ジョブが実行される際には，ディスク最外周部の block bit map を全て 1 に書き換え使用中とし，ディスク最外周部以外の領域にファイルが格納されるようにする．

また，提案手法 2 にはディスクの空き領域を監視する機能，空き領域の容量が指定の閾値を下回ったときに空き領域を動的に拡張する機能，空き領域の容量が指定の閾値を上回ったときに空き領域を動的に縮小する機能を動作させる．この機能は 2.3 節で紹介した既存のファイル格納位置動的制御手法と同一のものである．

5. 性能評価

本章では提案手法の性能を評価する．測定で用いた一連のジョブセットを図 13 に示す．ジョブセットは，Map-heavy グループ，Shuffle-heavy グループ，Reduce-heavy グループ，Map-heavy グループ，Shuffle-heavy グループ，Reduce-heavy グループ….の順に並べられた 27 個のジョブグループで構成されている．1 セットあたりに，9 つの Map-heavy グループ，9 つの Shuffle-heavy グループ，9 つの Reduce-heavy グループがある．各ジョブグループは 20 個のジョブで構成されている．ジョブセットを実行後，HDD は Hadoop ジョブの出力ファイルによってほぼ全て使用されている状態になる．主に Reduce-heavy の出力ファイルによって占有されている状態になる．Shuffle-heavy ジョブのファイルは一時的なものであり，プロセスが I/O バウンドであるため，ディスク最外周部に格納される．Map-heavy ジョブのファイルも一時的なものであるため，ディスク最外周部に格納される．Reduce-heavy ジョブのファイルは HDD に恒久的に残るため，ディスク最外周部には格納しない．本測定においては，ブロックグループ 0~3726 のうち Shuffle-heavy ジョブと Map-heavy ジョブのファイルをブロックグループ 0~199 に，Reduce-heavy ジョブのファイルをブロックグループ 200~3726 に格納されるようにした．

図 14，図 15 に既存手法および提案手法におけるジョブのファイル配置を示す．既存手法では，Map-heavy ジョブおよび Shuffle-heavy ジョブのファイルは Reduce-heavy ジョブの恒久的なファイルより高いアドレス（内周）の位置に格納されるため，ジョブ実行が進むにつれて内周側のゾーンを使うことになる．これに対して提案手法では，図 15 に示すように最外周部のゾーンに恒久的ファイルを格納しないので，Map-heavy ジョブと Shuffle-heavy ジョブは常に最外周部のゾーンを利用することができる．

図 16 にジョブセットを 5 回実行したときの平均実行時間を，図 17 に 1 回目のジョブセットの実行時における 9 個の Map-heavy グループの実行時間を，図 18 に Shuffle-heavy

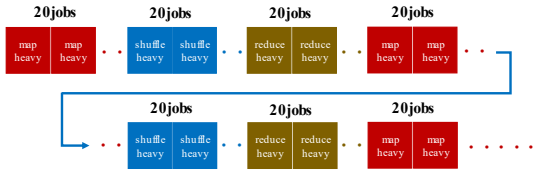

図 13　測定に用いるジョブセット

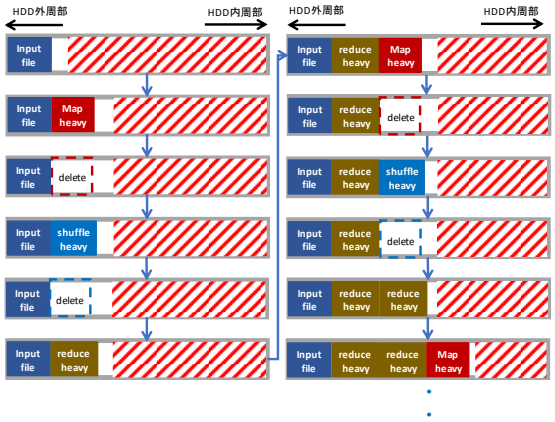

図 14　既存手法によるファイル配置

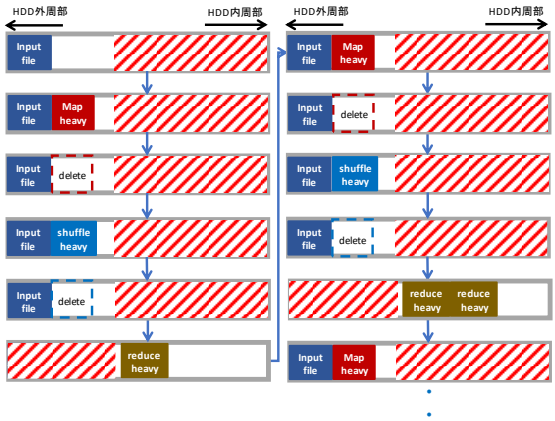

図 15　提案手法によるファイル配置

グループの実行時間を，図 19 に Reduce-heavy グループの実行時間を示す．

既存手法を用いることにより，通常手法より実行時間を 5.6%短くできることがわかる．同様に，提案手法 1 を用いることにより通常手法より 15%短く，提案手法 2 を用いることにより通常手法より 16.7%短くできることが分かる．図 17 から，Map-heavy ジョブに対しては既存手法と提案手法ともにファイル格納位置の制御による実行時間の短縮がほとんどできていないことがわかる．これは，先に示したように Map-heavy ジョブが CPU-intensive であり，I/O 性能向上による Hadoop 性能の向上の程度が小さいことが原因である．図 18 から，Shuffle-heavy ジョブに対しては既存手法，提案手法ともに実行時間を短縮することができているが，既存手法と比べて提案手法はより大きく実行時間を短くすることができていることが分かる．既存手法ではジョブの実行が進むにつれて，Shuffle-heavy グループの実行時

ⓒ 2019 Information Processing Society of Japan

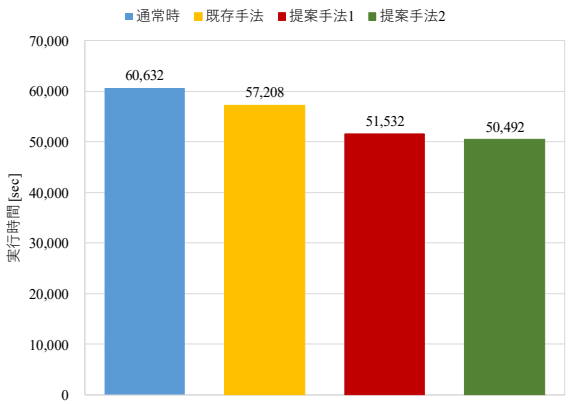

図 16　ジョブセットの実行時間

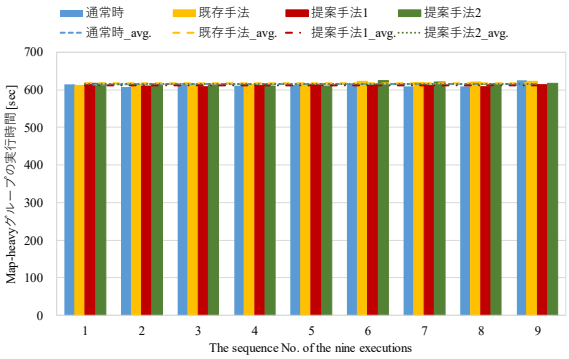

図 17　Map-heavy グループそれぞれの実行時間

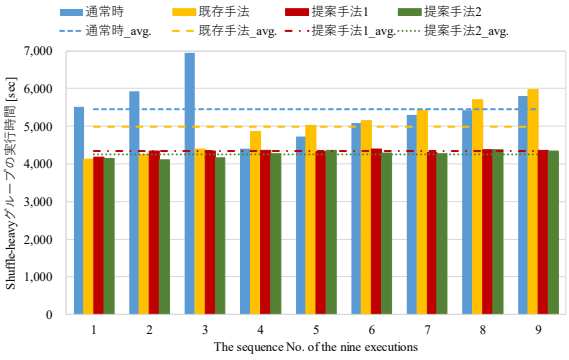

図 18　Shuffle-heavy グループそれぞれの実行時間

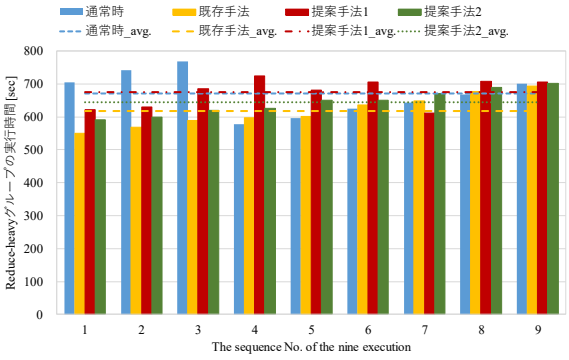

図 19　Reduce-heavy グループそれぞれの実行時間

間が徐々に大きくなっていることが分かり，Shuffle-heavy ジョブの実行ファイルは徐々にシーケンシャル I/O 性能の低いディスク内周部へと格納される様になっている．対して提案手法では，シーケンシャル I/O 性能の高いディスク最外周部を常に利用することができるため実行時間が小さくなっている．また，徐々に増加する現象も発生していない．また，提案手法 1 よりも提案手法 2 の方が実行時間を短くすることができている．提案手法 2 では，Shuffle-heavy ジョブはブロックグループ 0~199 のうちの最外周部を常に利用することができるが，提案手法 1 ではブロックグループ 0~199 のうちの高アドレス部を利用することを排除できていないことが原因である．図 19 から，Reduce-heavy ジョブに対しては既存手法が最も実行時間を短縮していることがわかる．提案手法より既存手法の方が実行時間が短い理由は，既存手法では Reduce-heavy ジョブはディスクの最外周部（ブロックグループ 0~199）を利用できるのに対し，提案手法ではブロックグループ 0~199 は利用することができないためである．

6. 考察

我々が提案した手法は，実行される Hadoop ジョブの特性，すなわち Map-heavy，Shuffle-heavy，Reduce-heavy といったジョブがどのような特性をもっているかが既知であることを前提とする．同じジョブが異なるデータに対して繰り返し実行されるようなアプリケーションであれば，この前提が満たされ我々が提案した手法を利用することができる．例えば，検索エンジンシステムは頻繁にインデックスを更新するが，これは Shuffle-heavy の特性と似ている．同様に，ショッピングサイトでも同一のオンライントランザクション処理（OLTP）ジョブが毎日実行される．オンライン分析処理（OLAP）アプリケーションの場合も，繰り返すジョブの特性は非常に似ている．また，我々が提案した手法が必要とするジョブの特性は容易に得ることができる．I/O 使用率と CPU 使用率はそれぞれ `iostat` コマンドと `vmstat` コマンドによって求めており，ディスク使用量の推移は `df` コマンドを繰り返すことによって求めている．したがって，我々が提案した手法は多くの場合に適用可能であると考えられる．

ディスク外周部に一時的なファイルのみを格納させる方法として，ディスク外周部のパーティションを作成しそのパーティションに一時的ファイルを格納させる方法もある．しかし，この実装方法では一時的ファイルのサイズが既知でかつ一時的ファイルのサイズが変わらない場合にのみ有効となる．一方本稿で提案した方法は，ビットマップを変更することで簡単に外周部の領域サイズを変更することができるため，より多くの状況に適用できる．

Hadoop は，HDFS とローカルファイルシステムの両方にアクセスする．HDFS 上のファイルは主に Hadoop による

ⓒ 2019 Information Processing Society of Japan

入力データの読み取りと出力データの書き込みのためにアクセスされ，ローカルファイルシステム上のファイルは主に中間データの格納に使用される．提案手法はローカルファイルシステムを制御することで実現している．HDFS はローカルファイルシステム上に構築されるため，提案手法は HDFS とローカルファイルシステムの両方のファイルに有効である．

本稿では, SWIM によって生成された代表的なジョブの I/O 性能を向上させる手法を提案した．この手法は SWIM ジョブの特性（CPU 使用率，I/O 使用率，ディスク使用量の推移，ファイルの揮発性）に基づいている．他のアプリケーションの実行を監視することによって，同様の手法を適用することができ，我々の手法のアプリケーションへの適用を一般化することができると考えている．

また，本測定では，Shuffle-heavy ジョブのファイルサイズを事前に調査し，そのファイルサイズに合わせて一時的ファイルを格納する領域をできるだけ小さくしたが，ジョブごとにファイルサイズに変動があり，ジョブのファイルサイズに合わせて領域を小さくすることができない場合は，提案手法 1 と提案手法 2 の性能により大きな差が表れると考えられる．

7. おわりに

本稿では，SWIM ジョブを Map-heavy ジョブ，Shuffle-heavy ジョブ，Reduce-heavy ジョブに分類し，それらのジョブの I/O 使用率，CPU 使用率，実行ファイルサイズといった特性を調査した．そして，対象となるジョブの特徴を考慮して I/O 性能を向上させる方法を提案した．既存のファイル格納位置制御手法は通常時の実行時間より 5.6%の短縮をできていたのに対し，我々の提案した手法では提案手法 1 にて 15.0%，提案手法 2 にて 16.7%の短縮をすることができた．

今後は，完全分散モードにて同様の手法の評価を行う予定である．

謝辞 本研究は JST，CREST JPMJCR1503 の支援を受けたものである．

本研究は JSPS 科研費 15H02696，17K00109，18K11277 の助成を受けたものである．

本研究は，NSF ACI 1550126 and supplement DCL NSF 17-077 の支援を受けたものである．

参考文献

[1] G. Jeffrey Dean and Sanjay Ghemawat. 2008. MapReduce: simplified data processing on large clusters. Commun. ACM 51, 1 (January 2008), 107-113. DOI: https://doi.org/10.1145/1327452.1327492

[2] Joichiro Kon，Kenji Nakashima and Saneyasu Yamaguchi，“A Deletion Aware Usable Space Control for SD2，” 2017 Fifth International Symposium on Computing and Networking (CANDAR)，Aomori，2017，DOI: 10.1109/CANDAR.2017.31

[3] GitHub - SWIMProjectUCB/SWIM: Statistical Workload Injector for MapReduce - Project at UC Berkeley AMP Lab, https://github.com/SWIMProjectUCB/SWIM

[4] Yanpei Chen，Archana Ganapathi，Rean Griffith，Randy Katz，” The Case for Evaluating MapReduce Performance Using Workload Suites ”，2011 IEEE 19th Annual International Symposium on Modelling，Analysis，and Simulation of Computer and Telecommunication Systems，July. 2011 ，10 pages，DOI: 10.1109/MASCOTS.2011.1

[5] R.Card and T.Ts’o and S.Tweedle，“Design and Implementation of the Second Extended Filesystem，” First Dutch International Symposium on Linux，1994

[6] Makoto Nakagami，Joichiro Kon，Gil Jae Lee，Jose A.B. Fortes，Saneyasu Yamaguchi，“File Placing Location Optimization on Hadoop SWIM，” 2018 Sixth International Symposium on Computing and Networking Workshops (CANDARW)，Takayama，2018，DOI: 10.1109/CANDARW.2018.00100

[7] 近丈一郎，中上誠，Jose A.B. Fortes，山口実靖，“ファイル格納位置制御による大規模 I/O 性能の向上に関する一考察，” DEIM Forum 2019 J4-3

[8] Makoto Nakagami，Jose A.B. Fortes，Saneyasu Yamaguchi，“Job-aware Optimization of File Placement in Hadoop，” BDCAA 2019 The 1st IEEE International Workshop on Big Data Computation，analysis，and Applications，COMPSAC 2019，July 2019.

[9] Faraz Ahmad，Seyong Lee，Mithuna Thottethodi and T. N. Vijaykumar，” MapReduce with Communication Overlap (MaRCO)”，Journal of Parallel and Distributed Computing，May 2013，Pages 608-620 ，DOI: https://doi.org/10.1016/j.jpdc.2012.12.012

[10] Eita FUJISHIMA Kenji NAKASHIMA Saneyasu YAMAGUCHI，"Hadoop I/O Performance Improvement by File Layout Optimization"，IEICE TRANSACTIONS on Information and Systems，Vol.E101-D No.2 pp.415-427，doi: 10.1587/transinf.2017EDP711

[11]. S.Yamaguchi，M. Oguchi and M. Kitsuregawa，"Trace system of iSCSI storage access，" The 2005 Symposium on Applications and the Internet，Trento，Italy，2005，pp. 392-398. doi: 10.1109/SAINT.2005.65

[12] Saneyasu Yamaguchi，Masato Oguchi，Masaru Kitsuregawa，"iSCSI analysis system and performance improvement of sequential access in a long-latency environment，" Electronics and Communications in Japan (Part III: Fundamental Electronic Science)，Volume 89，Issue 4，Pages 55-69， Wiley Subscription Services，Inc.，A Wiley Company，April 2006. DOI: 10.1002/ecjc.20238

[13] Yuta Nakamura，Kyosuke Nagata，Shun Nomura，and Saneyasu Yamaguchi. 2014. I/O scheduling in Android devices with flash storage. In Proceedings of the 8th International Conference on Ubiquitous Information Management and Communication (ICUIMC '14). ACM，New Y ork，NY，USA，Article 83，7 pages. DOI: https://doi.org/10.1145/2557977.255802

[14] Eita Fujishima and Saneyasu Yamaguchi. 2016. Dynamic File Placing Control for Improving the I/O Performance in the Reduce Phase of Hadoop. In Proceedings of the 10th International Conference on Ubiquitous Information Management and Communication (IMCOM '16). ACM， New York， NY， USA，Article 48，7 pages. DOI: http://dx.doi.org/10.1145/2857546.2857595

[15] Eita Fujishima and Saneyasu Yamaguchi， "Improving the I/O Performance in the Reduce Phase of Hadoop，" 2015 Third International Symposium on Computing and Networking

ⓒ 2019 Information Processing Society of Japan

(CANDAR), Sapporo, 2015, pp. 82-88. doi:
10.1109/CANDAR.2015.24

ⓒ 2019 Information Processing Society of Japan