ヒュージページを活用する インメモリ分散処理フレームワーク

宮里 勇也[1]　山田 浩史[1]

概要：分散処理システムとは，複数のコンピュータで処理を分散して行うシステムで，大規模なデータを処理することができ，サーバのコスト削減やリソースの拡張性といったメリットが存在する．中でも Apache Spark のようなインメモリ分散処理フレームワークでは，処理を出来る限りメモリ内で行うことで，従来のものよりも高速な処理が可能となる．インメモリ分散処理では膨大なデータをメモリ上で扱う都合上，アドレス変換が大きなボトルネックになってしまう．アドレス変換のボトルネックを解消する機構としてヒュージページがある．Apache Spark にヒュージページを適用する場合，主に THP というヒュージページ管理機構が用いられるが，THP にはメモリ肥大化や，メモリのコンパクションのために CPU が占有されるなど様々な問題が存在している．Apache Spark においても THP によるヒュージページ割り当ての効果は大きいが，THP のデメリットも受けてしまうため闇雲にヒュージページを利用することはできない．本研究ではインメモリ分散処理フレームワーク上でヒュージページを効率的に利用できる手法を提案する．提案方式では，Apache Spark のデータ領域の中でも，ヒュージページの効果が大きい領域にのみヒュージページの割り当てを行い，それ以外では通常のページ割り当てを行うことで無駄なヒュージページ割り当てを避ける．これにより，メモリ肥大化やコンパクションなどのデメリットを抑えつつ，ヒュージページによる性能向上の恩恵を受けることを狙う．実験の結果，マイクロベンチマークのようなキャッシュデータへのアクセス頻度が高いワークロードでは一部のみのヒュージページ割り当てで全体割り当てと同等以上のパフォーマンスが得られた．

キーワード：分散処理システム，ヒュージページ

1. はじめに

Spark に代表されるインメモリ分散処理フレームワーク [1] は，マシンに搭載されている膨大なメモリを活用するアプリケーションを構築するフレームワークである．インメモリ分散処理フレームワークは分散処理フレームワーク [2] の一種で，各マシン内での処理を可能な限りメモリ内で行うことで，巨大なメモリを要するアプリケーションの実現やストレージ I/O を抑えながらの処理速度向上が期待できる．たとえば，巨大なグラフを解析したり，大量のデータを必要とする機械学習などのアプリケーションの実現に利用される．インメモリ分散処理フレームワークにおいて膨大なメモリが必要な理由として，通常の分散処理フレームワークとの処理方式の違いが挙げられる．分散処理フレームワークでは，データの処理を行うたびにストレージへの書き出しを行う．対して，インメモリ分散処理フレームワークでは，データの処理が終わった後，ディスクへの書き出しを行わず，そのままメモリ内に保持し続けることで処理の高速化を実現している．このデータを保持するために膨大なメモリが必要となる．実際に多くのメモリを搭載するマシンの例として，Amazon EC2 が提供する High Memory インスタンス [3] では，最大 24TB ものメモリを搭載しており，この巨大なサイズのメモリをインメモリ分散処理フレームワークなどのアプリケーションが利用することになる．

インメモリ分散処理フレームワークなどの膨大なメモリを利用するアプリケーションでは，アドレス変換が大きなボトルネックとなることが知られている [4,5]．メモリ管理にはページングが広く利用されており，ページという単位で仮想アドレスと物理アドレスの変換を行う．高速化のために一度変換したページは TLB という高速なハードウェアに保存することでアドレス変換を高速に行うことが可能となるが，TLB はサイズがあまり大きくない．そのため，膨大なメモリに対しては TLB にキャッシュしておける量

[1] 東京農工大学
Tokyo University of Agriculture and Technology

が少なくなり，TLB を使ってアドレス変換できる範囲が狭く，多くのアドレス変換は TLB を使わない遅いものとなってしまう．

このような問題点を解決するために，ヒュージページという機構が存在する．ヒュージページでは，ページのサイズを通常よりも大きいサイズで使用することで一度に変換できるアドレスの範囲を大きくすることができる．これによって TLB に保存できるアドレスの範囲が増え，多くのページで高速なアクセスが可能となり，他にもページテーブルサイズの削減やページウォーク時間の短縮といった効果が得られる．

インメモリ分散処理フレームワークでは膨大なメモリを利用することから，ヒュージページの恩恵を期待できる．しかし，インメモリ分散処理フレームワークにヒュージページを適用する研究は存在しおらず，効率的なヒュージページの使用法はわかっていない．また，インメモリ分散処理フレームワークでヒュージページを適用するのはアプリケーション単位でしかできないという問題点もある．インメモリ分散処理フレームワークでヒュージページを使うには THP という動的なヒュージページ割り当てを行う機構を利用する．実行時に Java オプションで JVM ごとにヒュージページの使用の有無を指定することで扱うことができる [6]．しかし，THP にはメモリ肥大化やメモリコンパクションのための CPU 占有などのデメリットが存在し，インメモリ分散処理フレームワークの特性を考慮せずアプリケーション全体に闇雲に割り当てることが正しいとは限らない．

本研究では，インメモリ分散処理フレームワークのヒュージページの効果の定量的調査を行い，インメモリ分散処理フレームワーク上でヒュージページを効率的に利用できる手法を提案する．Linux の Transparent Hugepage (THP) および Spark XXXX を用いた調査においては，ヒュージページの割当部分を工夫することでメモリ使用量の増加を抑えながらインメモリ分散処理フレームワークの実行時間を最大で XXXX% 短縮させた．一方でアプリケーションによっては，繰り返し利用されるような一部のメモリに対してのみヒュージページ割り当てを行った場合でも実行時間の削減率には大きな差が出ないこともわかった．本調査の結果を受け，ヒュージページの割り当てが効果的な部分のみにヒュージページを適用し，それ以外では通常のページ割り当てを行うことで無駄なヒュージページ割り当てを避ける機構を提案する．提案機構は言語ランタイム上で稼働し，メモリ肥大化やコンパクションなどのデメリットを抑えつつ，ヒュージページによる性能向上の恩恵を受けることを狙う．具体的には，インメモリ分散処理フレームワークには長期間保存され繰り返し利用されるデータや一時的に使用される中間データなどの特徴があり中間データではヒュージページの効果が薄く，繰り返し利用されるデータに対してヒュージページを割り当てる．

本論文の貢献は以下の通りである．

- Real-world のインメモリ分散処理フレームワークにおいて，ヒュージページの効果を定量的に示した．
- ヒュージページを有効的に活用するインメモリ分散処理フレームワークを提案した．長期間生存して再利用されやすいデータに対してヒュージページ割り当てを行う．
- 提案を OpenJDK X.X.X と Spark X.X.X 上に実装した．JVM 側ではヒュージページでデータを生成する命令を作成し Spark 側でヒュージページに適したオブジェクトをこの命令を用いて作成する．
- マイクロベンチマークを作成して実験を行った．実験の結果，一部のメモリに対してヒュージページ割り当てを行うことができ，全体割り当てよりも実行時間を削減した．

2. 背景

2.1 アドレス変換

2.1.1 ページング

メモリの管理単位をページと呼び，アドレス変換の際に利用する，仮想アドレスと物理アドレスの対応表をページテーブルと呼ぶ．通常は 4096 バイトを 1 ページとして 1 つのマッピングでその領域をアドレス変換することができる．1 つのページテーブルを利用してアドレス変換を行う場合，利用しない無駄なアドレスのマッピングまで作成されるという問題点がある．通常，プロセスが利用する領域はアドレスの一部だけであり，仮想アドレス全体のマッピングを作成すると，ページテーブルのために無駄なメモリを使用することになる．

これを解決するため，多くの場合は多段ページテーブルという機構が採用されている [7]．多段ページテーブルではアドレス変換に利用する上位ビットをいくつかに区切り，小さいページテーブルを複数作成する．上位のページテーブルは次のページテーブルのポインタを指し，利用していないアドレスは NULL ポインタになっている．これにより，利用しない領域に無駄なメモリを使用することを防いでいる．このページテーブルを順番にたどってアドレス変換を行うことをページウォークと呼ぶ．

2.1.2 TLB

TLB(Translation Lookside Buffer) はメインメモリよりも高速なキャッシュであり，アドレス変換の結果を TLB に保存することで，アドレス変換の高速化が可能となる．ページウォークでは，複数のページテーブルへのアクセスが発生し，時間がかかってしまう．そこで，一度変換したアドレスのマッピングは TLB に保存しておく．プログラムの局所性により，一度アクセスされた領域は近いうちに再度アクセスされる可能性が高いため，TLB に保存した

マッピングは何度も参照される可能性が高い．TLB はメインメモリよりも高速に読み書きができるハードウェアで，かつページウォークとは違いアクセスも一度で済むため，TLB に保存されている結果を再利用することでアドレス変換を高速に行うことができる．対して TLB を使用できない TLB ミスが発生した場合は複数のページテーブルにアクセスして順番に辿っていく必要があるためアドレス変換が非常に遅くなってしまう可能性がある [8].

メモリを多く利用するアプリケーションでは，TLB ミス率が増加し，アドレス変換のオーバーヘッドが大きくなる．その理由として，TLB は高速である代わりに，メインメモリよりも圧倒的に容量が少ないことが挙げられる．単純な例として，TLB が 5MB の領域をカバーしている時，使用するメモリサイズが 10MB の場合は TLB ミス率は 50% となるが，使用するメモリサイズが 100MB の場合は TLB ミス率 95% となる．実際には参照の局所性などからこのような極端な値にはならないものの，使用するメモリサイズと TLB サイズの差が広がるほど TLB ミス率は増加することがわかる．

2.2 ヒュージページ

ヒュージページとは通常よりも大きなサイズのページを利用し，一度に多くの領域をアドレス変換することで，効率的なアドレス変換を行う手法である．ヒュージページを使うと TLB に保存できるページ数は同じでも，1 ページのサイズが増えるためより多くのアドレスの範囲を TLB に保存でき TLB ミス率を減らすことができる．また，1 ページのサイズが増えることでページテーブルの削減やページウォークの短縮といった効果も得られる．

2.2.1 Hugetlbapges

Hugetlbpage [9] は Linux におけるヒュージページを扱う方法の 1 つである．Hugetlbpage ではカーネルブート時にヒュージページ専用の領域を確保しておき，プログラム内でメモリを確保する際にヒュージページの利用を明示的に指定することで，ヒュージページを使うことができる．Hugetlbpage は現在あまり利用されておらず，その理由として，実際に利用するメモリの量は事前に分からないというものがある．事前に確保した領域に比べて，必要とするメモリ量が多い場合はヒュージページを割り当てることはできなくなり，少ない場合でもヒュージページ専用で確保した領域はヒュージページにしか使えないため，利用していない無駄なメモリが生まれるといった問題点がある．また，ユーザが明示的にヒュージページの使用を指定しなければならないため，開発者への負担や既存のプログラムでは利用できないといった問題点も存在する．

2.2.2 THP

THP(Transparent Huge page) [10] は Linux におけるヒュージページを扱う方法の 1 つで，Linux Kernel が自動的にヒュージページの割り当てを行うことで透過的にヒュージページを使うことができる．THP では必要な分だけ Linux Kernel が自動でヒュージページ割り当てを行い，利用が簡単なため，Linux ではヒュージページを割り当てる際には主にこの方法が用いられる．しかし，THP では積極的にヒュージページを割り当てようとするため，無駄なヒュージページ割り当ての発生や，ヒュージページ昇格スレッドによって CPU を使いすぎたりと，問題点も多く存在している．

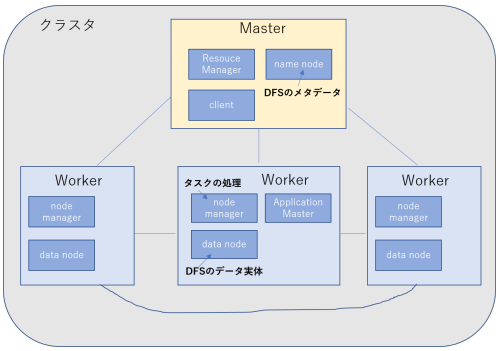

図 1 分散処理フレームワーク概要図

2.3 分散処理フレームワーク

分散処理フレームワークとは，複数のコンピュータをネットワークを繋いで処理を分散し，並列処理を行うことで巨大なデータを効率的に処理するシステムである．代表的な分散処理フレームワークとして，Apache Hadoop [11] や Apache Hive [12], Apache Tez [13] などがある．分散処理フレームワークにおけるネットワークで繋がれた複数台のコンピュータをクラスタと呼び，クラスタは 1 台のマスターノードと，複数台のワーカーノードで構成される．マスターノードでは主にクラスタ全体の管理を行い，実際にデータ保存したり処理を行ったりするのがワーカーノードとなる．

図 1 に分散処理フレームワークの概要図を示す．マスターノード内にあるネームノードがクラスタに保存されているファイルのメタデータを管理し，実際のデータはワーカーノード内のデータノードに分散して保存されている．クライアントプログラムが起動されると，マスターノード内のリソースマネージャーが各ワーカーノードにリソースを割り振り，各ノードマネージャーで処理が実行される．処理の進捗はアプリケーションごとに存在するアプリケーションマスターが担当し，全ての処理が終了すると，クライアントプログラムへ結果が返却される．このように，処理を複数のコンピュータに分散することでアプリケーションの実行時間の削減が可能となる．また，コンピュータを追加することでそのままリソースを増やせるスケーラビリティやデータを複製することによる耐障害性の確保といったメリットも存在する．

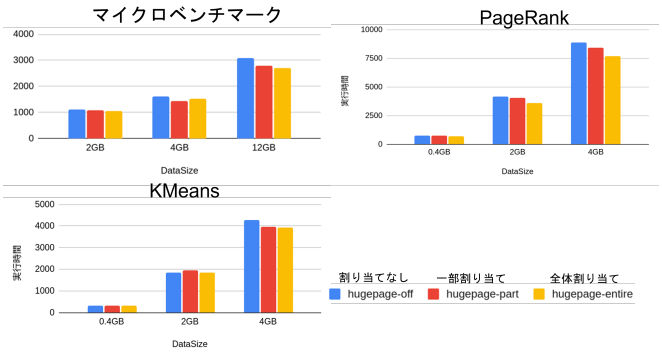

図 2 予備実験結果

2.4 Apache Spark

Apache Spark はインメモリ分散処理フレームワークの 1 つで，分散処理フレームワークのなかでも処理を出来る限りメモリ内で行うことでネットワークや I/O による遅延を防ぐことができる．Hadoop などでは処理のたびにディスクアクセスが発生するため，ボトルネックが存在するという問題点がある．Spark では処理後の一時データの出力先をディスクではなくメインメモリに書き出すことで，余計なディスクアクセスを減らして Hadoop の問題点を解決している．また，ApacheSpark では RDD [14] と DAG [15] という仕組みが用いられており，処理のログを保存しておくことで，高い耐障害性も維持している．これらのことから，分散処理フレームの中でも近年の主流となってきている．

2.5 予備実験

Apache Spark へのヒュージページの効果を調査するために予備実験を行った．予備実験ではヒュージページの割り当てをなし，全体への割り当て，StorageMemory のみへの割り当てと 3 種類の状態で実験を行った．実験結果を図 2 に示す．実験の結果ヒュージページは Apache Spark で効果的に働き実行時間を短縮させたことが分かった．また，マイクロベンチマークや kmeans では一部のメモリ割り当てでも全体割り当てに近い性能が得られることが分かった．しかし，この一部割り当てを行うのは Spark から JNI [16] を通して madvise システムコール [17] を呼び出すことによって，StorageMemory のデータを後からヒュージページに昇格していくという手法であり，ヒュージページになるのが遅く実行時間が短いと効果をあまり受けられないことや，メモリのスキャンにオーバーヘッドがかかるという問題点が存在していた．

3. 提案

本研究ではヒュージページを有効的に活用するインメモリ分散処理フレームワークを提案する．アプローチとしてはヒュージページの効果を受けやすいようなデータはヒュージページで生成して，それ以外は通常のページで割り当てすることで効率の良いヒュージページ割り当てを行う．提案方式によってヒュージページの恩恵も受けつつもデメリットを最小限に抑えることができる．

提案を実現する際にデザインチャレンジとなるのが以下のものである．

- **ヒュージページの割り当てはアプリケーション単位でしか選択できない**
  Apache Spark でヒュージページを使う際には実行時に Java オプションで UseTransParentHugepages を指定して実行する．そうすると THP によって JVM 全体がヒュージページに割当たることになる．この選択は JVM ごとにしか選べないためデータ単位で割り当てることが出来ず Spark の特性に適したデータごとの割り当てを行うことができない．そこでデータ単位でヒュージページ割り当てを行う手法が必要となってくる．
- **ヒュージページの効果を受けやすい適したデータをどう選ぶか**
  データの一部をヒュージページに割り当てることが出来たとして，どのデータに割り当てすれば効率的にヒュージページが使えるかを考える必要がある．

4. 設計

4.1 データ単位でのヒュージページ割り当て

前章で述べた通り，Apache Spark においてヒュージページを使う際にはアプリケーション単位でしか使用の有無を選択できない．これは JVM のオプションとして JVM 全体にヒュージページを使うか使わないかしか選べないからである．

予備実験ではデータ単位での割り当てを行うための実装を Spark において行った．手法としては StorageMemory にあるデータを定期的にスキャンし，1 ページに一定以上のサイズが含まれている場合はヒュージページに昇格するようにマークをつけていくという方式となっている．これによって一部のヒュージページ割り当てには成功したが，一度割り当てたデータを後から昇格するため昇格に時間がかかりヒュージページの効果が現れるのが遅いことやメモリをスキャンするためにスレッドを一時停止するためオーバーヘッドが大きいといった問題点があった．

これらを解決するために今回の提案方式では JVM に改良を加え，データごとにヒュージページを使うかどうか選べるようにする．この改良された JVM を使い Spark でデータを生成する際にそのデータの特性に合わせてヒュージページを使うかどうか選ぶことで効率的なヒュージページ割り当てを行う．JVM 側でデータの生成の際に直接ヒュージページとして生成することで昇格を待つ必要がなく，すぐに恩恵が受けられる．また，メモリをスキャンする必要

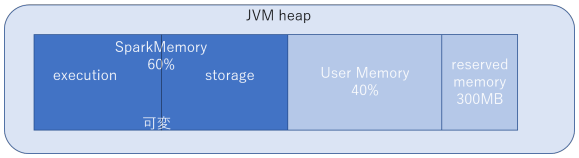

図 3 Spark メモリ管理

がなくなったためオーバーヘッドを気にする必要もなくなる．

4.2 ヒュージページ割り当てを行うデータ

Spark では図 3 に示すような形でメモリの管理が行われている [18]．Reserved Memory は Spark 内部オブジェクト用に確保された領域であり固定の大きさとなっている．User Memory はユーザデータ構造や内部メタデータを保存する領域である．Execution Memory は Spark のタスクを実行するためのメモリであり，主に短命なオブジェクトのための領域である．Storage Memory はキャッシュした RDD などが保存される領域で主に長命なオブジェクトを保存するための領域である．Storage Memory と Execution Memory のサイズは明確に決まっておらず，状況によって領域が移動する．

Spark アプリケーションではファイルからデータをロードし，RDD として Storage Memory に保存，RDD へ繰り替えし処理を行う，という構成が多いため，Storage Memory には頻繁にアクセスされる大きなデータが保存される．そのため，ヒュージページの影響を大きく受ける領域は Storage Memory ではないかと考えられる．よって提案方式ではこの StorageMemory に保存されるデータに対してヒュージページ割り当てを行う．

5. 実装

本研究では，Apache Spark3.2 と OpenJDK11 [19] を実装対象とし，OpenJDK にデータ単位でのヒュージページ割り当てを行うための機能を追加し，Spark にその機能を用いて繰り返し利用されるデータに対してヒュージページを割り当てるように変更を行った．

5.1 JVM の改良

Java ではオブジェクトを生成する際に new 命令を使って生成を行う．この new 命令に加えてオブジェクトを生成する際にヒュージページで生成する hp_new 命令を実装した．また，配列を生成する newarray 命令などもあるため，それぞれの生成命令にはヒュージページで生成する命令を実装している．実装のためにまず，JVM に HugepageRegion を追加した．Openjdk11 の標準 GC である G1GC [20] ではメモリが Region という単位で分割され管理されており，HugepageRegion はこの Region をヒュージページとしたものである．hp_new 命令が初めて呼び出されると通常の Region を保存するフリーリストの中から使用したことのない Region を選び HugepageRegion 専用のフリーリストに移動する．一度別の用途に使われた後にフリーリストに返された Region だとヒュージページを割り当てる際に直接ヒュージページ割り当てが出来ず，昇格によってヒュージページになるため優先して使用したことのない Region が選択される．選択された Region は madvise システムコールによって MADV_HUGEPAGE アドバイスが追加される．これによりこの Region にメモリを割り当てる際にはヒュージページとして割り当てられるようになる．以降は hp_new 命令が呼び出された時は HugepageRegion から割り当て，存在しない場合は同じようにフリーリストからもらってくるという手順となっている．また，HugepageRegion を返す場合はヒュージページ専用のフリーリストに返すことでヒュージページと通常のページの分離を行っている．これにより通常は new 命令による通常の割り当て，ヒュージページを使いたい時は hp_new 命令を使うことでデータ単位でヒュージページの使用の有無を選べるようになる．また，追加の工夫として HugepageRegion は最初から Old 領域にするようにしている．HugepageRegion に保存されるデータは長期間利用されるのが前提のため，最初から Old 領域にすることで Young 領域で GC を何度もされて Old 領域へ移動といった手間を省くことができる．

5.2 Spark の改良

Spark 側では StorageMemory のデータを生成する際に hp_new 命令を使う用にする．ここで hp_new 命令を使うケースとして二つ存在している．まず，StorageMemory に保存したいオブジェクトを自身のコード内で生成している場合である．このような場合はユーザが hp_new 命令をコード内で記述してヒュージページとしての生成を行うことになる．これにより hp_new 命令を使用したオブジェクトがヒュージページに割当たるようになる．2 つ目のケースがファイルなどに保存しておいたデータをシリアライズして StorageMemory に RDD として保存する場合である．このケースでは Spark 内でデータをキャッシュする際に利用するバッファーの生成に hp_new を使うようになっている．このバッファーがヒュージページで確保されることでデータがヒュージページとして割当たることになる．この場合ではユーザがコードの変更をすることなく StorageMemory のデータをヒュージページとして割り当てることができる．

6. 実験

提案方式によるヒュージページ割り当てとヒュージページなし，標準のヒュージページ割り当て（全体割り当て），予備実験での手法でのヒュージページ割り当てと 4 つのヒュージページ割り当ての方法の 4 つのヒュージページ

表 1　マシン性能

	マスターノード	ワーカーノード
CPU	Intel Xeon E5-2430 v2	Intel Xeon E-2124
コア数	4	4
RAM	32	64
kernel	5.4.0	5.4.0

表 2　Spark の設定

エグゼキュータ数	4
コア数	1
メモリ	20GB

割り当てポリシーで実験を行った．実験では実行時間やヒュージページ割り当てサイズ，TLB ミス率などを計測した．TLB ミスの計測には perf [21] コマンドを用いている．実験には作成した単純なマイクロベンチマークを利用した．マイクロベンチマークは StorageMemory にデータをキャッシュしそのデータに対してアクセスを繰り返す StorageMemory への負荷が大きい反復ワークロードとなっている．

6.1　目的

本実験の目的は提案方式が正しく機能し，一部への割り当てだけでも全体の割り当てと同等にヒュージページの恩恵を受けられることを確認する．また，今回の提案方式では予備実験で使用した手法のデメリットが解消されていることも確認する．

6.2　実験環境

実験で使用したマシンの性能を表 1 に，Spark の設定を表 2 に示す．実験ではワーカーノードを 2 つ用意し，各ワーカーノード内で 2 つのエグゼキュータを起動することで，合計 4 つのエグゼキュータを動かしている．また，リソースマネージャーとして Hadoop YARN [22] を利用する。

6.3　実行時間

実行時間を図 4 に示す．2GB のようにデータサイズが小さい時は実行時間に対してメモリアクセスの時間が少ないためどの条件でも大きな差はない．4GB, 12GB だと差が表れてきており，今回の提案方式 (hugepage-part) ＞全体割り当て (hugepage-entire) ＞予備実験での手法 (hugepage-part-old) ＞割り当てなし (hugepage-off) の順番に早くなっていることが分かる．今回の提案方式が一番早くなっており，これはヒュージページの恩恵は全体割り当てと同様に受けているのに加えてメモリコンパクションなどのヒュージページのデメリットは抑えられたことが原因だと考えられる．また，最初から Old 領域に割り当てることによる GC の節約も原因の一つだと考えられる．予備実験での手法では実行時間の削減ができてはいるが，他の割り当てありに劣り，ヒュージページを使うのが遅いことやメモリスキャンオーバーヘッドが出ているとわかる．今回の方式では全体割り当て以上に削減できておりその問題点を改善としていることがわかる．

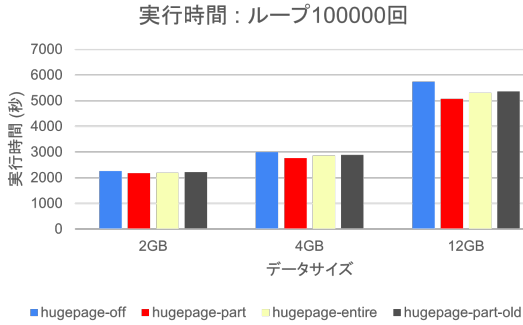

図 4　実験結果：実行時間

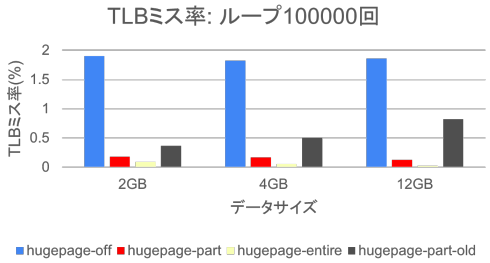

図 5　実験結果：TLB ミス率

6.4　TLB ミス率

TLB ミス率を図 5 に示す全体割り当て，今回の提案方式が両方とも大きく TLB ミス率を削減しており，この TLB ミスの削減で実行時間が削減されたと分かる．予備実験の手法は他よりも削減率が良くないことからヒュージページを使い始めるのが遅く恩恵を十分に受けられていないことが分かる．対して今回の提案方式は全体割り当てに近く大幅な TLB ミス率の削減ができているので，ヒュージページを最初から十分に使えているとわかる．

6.5　ヒュージページ割り当て数の比較

データサイズ 12GB でのヒュージページ割り当て数の遷移を図 6 に示す．今回の提案方式では即座に 3GB の割り当てを行っている．サイズとして 4 つの Executor が 12GB をそれぞれ保存するため，12/4 = 3GB のデータであり，StorageMemory のデータをすぐに全部割り当てることができている．予備実験の方式では割り当てに長い時間がかかっており，ヒュージページの恩恵を十全に受けられるまで時間がかかっているのがわかる．全体割り当てについては StorageMemroy 以外のデータにも割り当てを行っており，ヒュージページを使うサイズが大きくなっている．この時の無駄な割り当てによるオーバーヘッドが実行時間に

出てきていると考えられる．

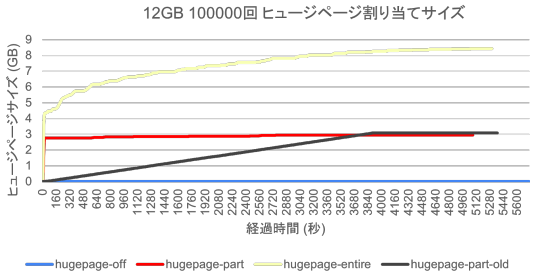

図 6 実験結果：ヒュージページ割り当てサイズ

7. 関連研究

HotTub [23] は JVM のウォームアップ時間を減らすことで分散処理フレームワークのパフォーマンスを向上させる機構である．HotTub では JVM のウォームアップを減らすために JVM の再利用を行う．一度使用した JVM は通常そのまま終了されるところを一旦プールしておき，他のアプリケーションによって JVM を起動する時にプールしておいた JVM を再利用することでウォームアップにかかる時間を短縮する．結果として HotTub はウォームアップ時間を短縮し，HDFS [24] の読み書きや SparkSQL [25] などのワークロードでの実行時間の短縮に成功している．

Yak [26] は分散処理システムに適した GC で世代別 GC と領域ベースの GC を組み合わせたものとなる．Yak では Spark のオブジェクト特性に注目し，オブジェクトが長寿でライフサイクルが同じであるということを利用して，オブジェクト適正に合わせた GC を行うことで GC 時間を短縮させている

Riffle [27] は MapReduce のシャッフルフェーズにオーバーヘッドが大きいことに注目し，シャッフルの際の細かい中間データをマージして一つにすることでシャッフル時間を短縮させることに成功している．

Ingens [28] は THP の問題点を解決し効率的なヒュージページ管理を行う機構である．Ingens ではヒュージページの使用率とアクセス頻度を監視し，その情報をもとに適切なタイミングでヒュージページ割り当てを行うことで断片化の解消やメモリの節約といったパフォーマンスの向上を行っている．

Illuminator [29] は THP のメモリ汚染による断片化を減らし，効率的なヒュージページ管理を行う機構である．Illuminator では Linux に存在するユーザ用の移動可能な movable とカーネル用の移動不可能な unmvable なメモリ領域に加えて movable と unmovable が混在したハイブリッド領域を明示的に管理し，ハイブリッド領域ではヒュージページ割り当てを避けることで無駄にページ移動することを防いでいる．結果として Illuminator はページ移動にかかる時間を短縮させ，パフォーマンスを向上させている．

これらの研究はヒュージページと分散処理フレームワークそれぞれに個別に対処したものであり，インメモリ分散処理フレームワークを対象としたヒュージページ割り当てに関する研究は行われていない．本研究では分散処理フレームワーク固有の特性を考慮して効率的にヒュージページを利用する手法を提案した．

8. おわりに

本研究ではヒュージページを活用するインメモリ分散処理フレームワークを提案した．ヒュージページはインメモリ分散処理フレームワークのような膨大なメモリを利用するアプリケーションと相性がよくパフォーマンスの向上が期待できる．しかし，現在インメモリ分散処理フレームワークでヒュージページを使うにはアプリケーション単位でしか選択できず，データ単位で割り当てることができない．THP にはメモリ肥大化やメモリコンパクションのための CPU 占有などのデメリットが存在するため闇雲にアプリケーション全体に割り当てるのが正しいとは限らず，Spark のデータ特性に合わせた割り当てが必要となってくる．提案方式ではデータ単位でのヒュージページ割り当てを実装し，長期間生存して再利用されやすいデータに対してのみヒュージページ割り当てを行うことで，ヒュージページの恩恵を受けつつ，デメリットを抑えることを狙う．提案方式を Apache Spark と OpenJDK に実装しマイクロベンチマークで実験を行った．実験の結果，実装したシステムが正常に動作し，一部のメモリにのみヒュージページを割り当てることに成功した．そして，提案方式が全体割り当てよりも実行時間を削減できたことを確認した．今後として，マイクロベンチマーク以外の Pagerank [30] や kmeans といった Real-time なワークロードの実験を行う．また，提案方式ではユーザが自身のコード内で生成したオブジェクトをヒュージページにするためにはコードをユーザが変更する必要があった．既存のコードの再利用などのために，StrageMmeory にキャッシュされるオブジェクトについては全てを自動でヒュージページにするように改良を行う必要があると考えられる．

参考文献

[1] others Vol. 10, No. 10-10, p. 95 (2010).
[2] SanjayGhemawat Vol. 51, No. 1, pp. 107–113 (2008).
[3] Amazon EC2 High Memory Instance.
[4] Michael MSwift Vol. 41, No. 3, pp. 237–248 (2013).
[5] Michael MSwiftpp. 178–189 (2014).
[6] TransParent Huge Pages.
[7] Linux Page Table Management.
[8] Intel 64 and IA-32 Architectures Software Developers Manual.
[9] Hugetlbpage Support.
[10] Transparent Hugepage Support.

[11] Apache Hadoop.
[12] Apache Hive.
[13] Apache Tez.
[14] IonStoicapp. 15–28 (2012).
[15] An Introduction to Directed Acyclic Graphs.
[16] Java Native Interface.
[17] MAN Page of MADVISE.
[18] Spark Memory Management.
[19] OpenJDK.
[20] TonyPrintezispp. 37–48 (2004).
[21] Perf.
[22] otherspp. 1–16 (2013).
[23] DingYuanpp. 383–400 (2016).
[24] RobertChanslerpp. 1–10 (2010).
[25] otherspp. 1383–1394 (2015).
[26] OnurMutlupp. 349–365 (2016).
[27] Michael JFreedmanpp. 1–15 (2018).
[28] EmmettWitchelpp. 705–721 (2016).
[29] KGopinathpp. 679–692 (2018).
[30] TerryWinogradStanford InfoLab (1999).