Hadoopを利用した超並列計算処理のチューニング指針

清水 宣行　山口 崇　土田 誠　山口 亜希子

Performance Guide for Massively Parallel Processing Using Hadoop

Nobuyuki Shimizu, Takashi Yamaguchi, Makoto Tsuchida and Akiko Yamaguchi

データ量の爆発的な増加と共に，企業でのHadoopの利用が急激に増加している．その理由の1つが超並列計算によるバッチ処理時間の劇的な短縮であるが，システム・リソースやデータ量などの実行条件によりHadoop設定値をチューニングする必要がある．他のミドルウェアと同様に，デフォルト値がパフォーマンス最適値とは限らないが，そのチューニング手法は確立されていないという課題がある．そこで本稿では，スケールアウト，圧縮の適用，Reducerの制御など，Hadoopチューニングの検証結果を考察し，大規模データをより短時間で処理するためのチューニング指針を提示する．

Many companies have started to employ Hadoop to deal with the explosive increase in the volume of data. One reason why Hadoop is being used is to reduce batch job execution time by using massively parallel processing, but it is necessary to tune Hadoop's settings in accordance with system resources and the volume of data. As is the case with other middleware, the default values are not tuned for high performance; however, a Hadoop tuning guide has not yet been established. In this paper, we consider our test results with regard to factors including scaling-out, data compression and the number of reducers used, and suggest how to tune Hadoop's settings in order to process large scale data transactions quickly.

Key Words & Phrases：Hadoop，パフォーマンス・チューニング，超並列計算，ビッグデータ，InfoSphere BigInsights
Hadoop, performance tuning, massively parallel processing, BigData, InfoSphere BigInsights

1．はじめに

全世界で生成されるデータ量が爆発的に増加している．IDC（International Data Corporation）のレポートによると,2011年に世界で作成されるデータ量は,1.8ZB（Zetta Byte=2^{70} byte）に達すると予測されており,この5年間でデータ量が9倍に増えたことになる［1］．実際にNew York Stock Exchangeは毎日1TB以上のデータを，Facebookは毎日20TB以上の圧縮データを，欧州原子核研究機構は毎日40TB，年間で15PBのデータを生成している．

こうした背景から「ビッグデータ」を活用する機運が企業の間に広まっている．これは従来の「コスト削減のためのIT」から「ビッグデータを活用しビジネスに貢献するIT」への転換のチャンスであるとも言える．そのため企業は，ペタバイトやテラバイト・クラスの大規模データを効率的に処理する実行基盤を求めており，秒レベルの処理時間が許容される領域において，とりわけApache Hadoop［2］（以下，Hadoop）が注目されている．Hadoopは，以下のアプローチにより超並列計算（Massively Parallel Processing）を実現し，効率的に大量データを処理している．

1）データをプログラムがある場所に移動するのではなく，データのある場所にプログラムを移動する
2）ディスク／筐体を複数並べて同時に利用する

では,Hadoopを利用することで,どのくらいのデータ量を，どのくらいの時間で処理できるのであろうか．Yahoo社から，1,406台のノード数で1TBのデータを62秒でソート処理したというレポートが公開されている［3］．しかしながら，この62秒という結果に至るまでのパフォーマンス・チューニングの情報は公開されていない．最適なパフォーマンスを得るためには，システム・リソースやデータ量などの実行条件からHadoopの設定値をチューニングする必要がある．しかしながら，これまでに公開されている数多くのHadoopに関するレポート［4］は，技術開発や実証実験が中心であり，チューニング手法は明らかにされていない．

そこで本稿では，Hadoopのパフォーマンス検証を行い，スケールアウトによる処理時間への影響，既存サーバーの能力を向上させるHadoopの設定値とその効果を確認する．その結果の考察をもとに，大規模データをより短時間で処理するためのチューニング指針を提示することを目的とする．

提出日:2011年9月20日　再提出日:2011年12月8日

2. Hadoopとパフォーマンス・チューニングの意義

2.1 Hadoop 概要

Hadoopとは，大量のデータを複数のコンピューターで並列処理するためのプラットフォームである．Googleが学術論文［5］に公開した一部仕様をもとにし，Doug Cutting氏が中心となりオープンソース・プロジェクトとして開発された［6］．

以下，Hadoopの構成と処理概要を示す．

• Hadoopの構成

Hadoopのコアは分散ファイルシステムのHDFS(Hadoop Distributed File System)と並列分散処理のMapReduceから構成される．HDFSはMapReduceで処理するデータを扱う分散ストレージであり，HDFSクラスターに参加した複数のマシンを1つの巨大な仮想ストレージとして管理する．MapReduceは，巨大なデータ集合をHadoopで処理するプログラミングモデルである．システムの全体を管理するMaster Node群と，複数台並んで実際に処理を行うSlave Node群に分かれ，これらNode群が協調して動作することで図1のようなHadoopクラスターが構成される．

• Hadoopの処理概要

HDFSのName Nodeがデータ配置場所などのメタデータを管理し，Data Nodeが実際にデータを読み書きする．一方，MapReduceのJobTrackerが1つのジョブをタスクと呼ばれる細かな処理単位に分割し，Slave Nodeに処理を割り振った後，TaskTrackerが割り振られたタスクを実行し応答を返すというのが基本的な挙動である．

2.2 パフォーマンス・チューニングの流れ

Hadoopを利用したシステムにおいてもWebシステムなどと同様に，パフォーマンス要件を洗い出し，システムに落とし込み，テストにて確認する．パフォーマンス要件を満たせない場合には，図2のサイクルを繰り返し，パフォーマンス要件にシステムを近づける．Hadoopの場合，一般的にMapReduceジョブの実行時間の充足がパフォーマンス要件となる．以下，Hadoop固有のチューニング・サイクルの項目を説明する．

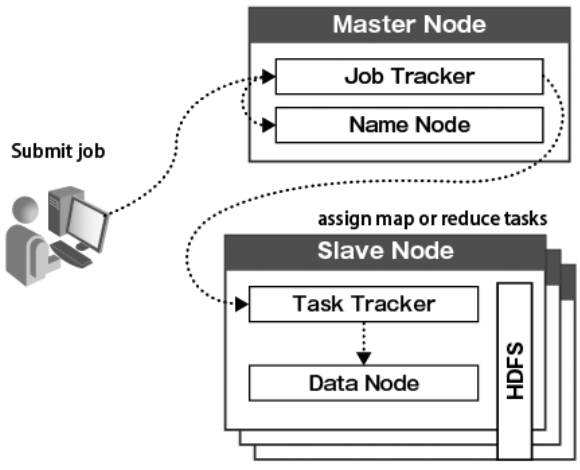

図1. Hadoopクラスター構成

• Hadoop基盤系のチューニング

Hadoop設定パラメーターの変更など，MapReduceの処理内容を変更しないチューニング項目．

• Java Virtual Machine（JVM）のチューニング

ガベージ・コレクション（GC）によるプログラム停止時間を最小にするチューニング項目．

• アプリケーションのチューニング

MapReduceの処理内容を変更することでパフォーマンスを向上するチューニング項目．

2.3 パフォーマンス・チューニングの意義

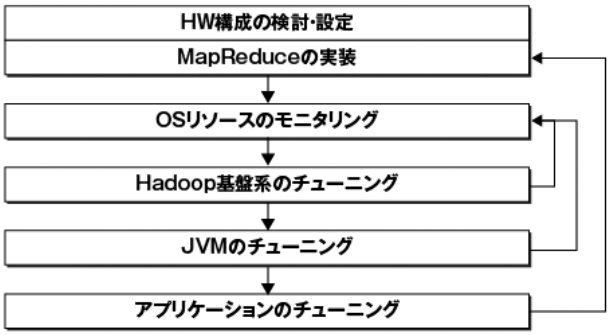

図2. チューニング･サイクル

OSやHadoopのデフォルト値ではパフォーマンスの最適値とならないことが多い．システム･リソースやトポロジー，データ量などの実行条件によりチューニングを行う必要がある．4章にて説明するが，例えば筐体数を1台から9台に増やすことでMapReduceジョブの実行時間を20%に短縮できた．また，データを圧縮（LZO: Lampel-Ziv-Oberhumer）することでMapReduceジョブの実行時間を66%に短縮できた．データ量が大きくなるほど，MapReduceジョブの実行時間が長くなるため，パフォーマンス・チューニングによる効果の影響度が大きくなる．3時間の実行時間を66%にするということは，実行時間を1時間短縮したことになる．

3. パフォーマンス検証概要

3.1 検証のための解析データ

自動車走行中に，Smart Phone 経由で収集した GPS による位置情報を検証のための解析データとして利用する．その解析データは，図３に示すように，数秒間隔で取得された「時刻」「緯度」「経度」「速度」「向き」などの情報で構成されている．1 レコードはバイト・クラスのデータ量であるが，数十万人が 1 日運転しただけでテラバイト・クラスの大規模データとなり，短時間で分析処理するためには Hadoop などの超並列計算の利用が必要となる．また，車の位置情報データの有効活用により，精度の高い渋滞予測や駐車場の混雑状況の提供といったドライバーの利便性を向上することができるため，さまざまな企業で位置情報のビジネス展開が検討・実用化されている［7］.

図3. 解析データ（1レコード）

3.2 検証のためのアプリケーション

車両毎に発生する位置情報から，車両毎の最高速度を算出する MapReduce アプリケーションを作成する．Hadoop への入力ファイルとなる解析データは，ファイルサイズが約 10GB，レコード件数が約 8,000 万件の CSV ファイルである．Map 処理では，Key として 5 カラム目の GPS-ID（図３では”10001”）を，Value として 11 カラム目の速度（図３では”32.25”）を抽出し，Reduce 処理にて Key 毎の最高速度を算出する．MapReduce 処理実行後の出力ファイルは，図４に示す CSV ファイルとなり，Key 毎の最高速度が 2 カラム目（図４では Key:”10001”の最高速度は”122.399994”）で算出される.

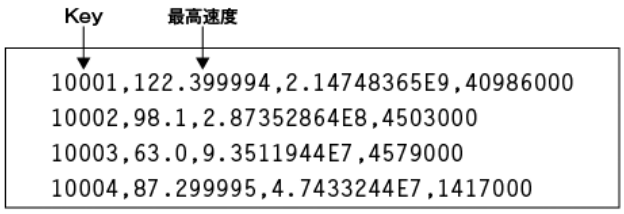

図4. 出力ファイル

3.3 検証内容

図５の検証環境にて MapReduce アプリケーションを実行，ジョブ実行時間のほか，HDFS へのコピー時間，システム・リソースの使用状況を複数回計測しその平均値を算出する.

Master Node が 1 台，Slave Node が 9 台から構成される Hadoop クラスター環境を，IaaS 型パブリック・クラウドである IBM Smart Business Cloud – Enterprise（以降，SBCE）［8］上に構築する．Master Node として SBCE の仮想マシン・インスタンスの Platinum を，Slave Node として Silver を選択する［9］.

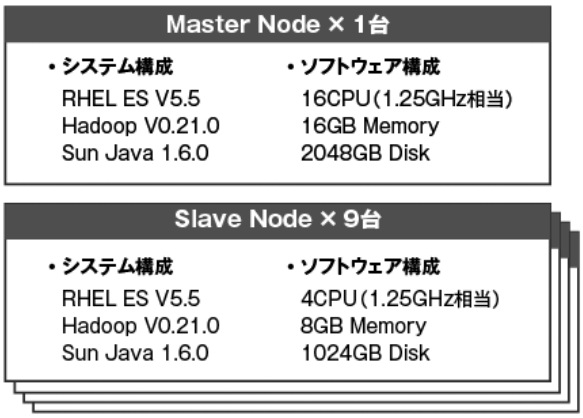

図5. Hadoopクラスター環境 検証構成

当検証では，以下の 6 つの項目をチューニング対象とする．これらは「Hadoop: The Definitive Guide」［10］，および「Hadoop 徹底入門」［11］を参考に，有効度が高いものを取り上げた.

<Hadoop 基盤系チューニング >

- 項目 1: スケールアウト

Hadoop の真骨頂はスケールアウトであり，Slave Node 数を増やすことでジョブ実行時間を短縮できる．Slave Node を 1 台／3 台／6 台／9 台としたケースのパフォーマンスを比較する.

- 項目 2: 圧縮

Hadoop のアプリケーションは，圧縮ファイルを直接処理できる．これは入力データだけではなく，MapReduce 処理の中間データや出力データも圧縮することができる．データ圧縮により，転送時間の短縮とストレージ領域の節約効果が期待できる．圧縮形式として，bzip ／ zlib ／ LZO を利用したケースのパフォーマンスを比較する.

<JVM チューニング >

- 項目 3: JVM ヒープ・サイズ

最小ヒープ・サイズと最大ヒープ・サイズを調整することで，GC 実行時間を短縮する効果が期待できる．最大ヒープ・サイズを 1024MB に固定し，最小ヒープ・サイズを 128MB ／ 512MB ／ 1024MB としたケースのパフォーマンスを比較する．

< アプリケーション・チューニング >

- 項目 4: レプリカ数

HDFS 内のデータは,「ブロック」という特定サイズ（デフォルト :64MB）で区切られた単位で管理される．各ブロックは，複数の Data Node に分散配置され，ブロック単位で複製配置されることで信頼性が確保される．複製配置されたブロックを「レプリカ」と呼ぶ．HDFS への書き込み時にはレプリカに対する書き込みが同時発生するため，レプリカ数を減らすことでジョブ実行時間を短縮する効果が期待できる．レプリカ数を 0 個／ 1 個／ 2 個としたケースのパフォーマンスを比較する．

- 項目 5: Reducer 数

Map 処理が作成した中間ファイルを集計し，最終的なデータを作成するのが Reduce 処理である．Reduce は，結果データを集約するため，デフォルトで 1 個の Reducer プロセスで処理される．Reducer 数を増やし並列実行させることで，Reduce 時間を短縮する効果が期待できる．Reducer 数を 1 個／ 9 個／ 18 個としたケースのパフォーマンスを比較する．

- 項目 6: Combiner

Combiner は，Map の出力結果を同一ノードで連続して Reduce 処理するプログラムである． Map 処理 → Shuffle 処理（ソート・マージ）→ Reduce 処理のうち，Shuffle にかかる負荷を軽減し，データ転送量を減らす効果が期待できるが，アプリケーションの改変を必要とするケースがある．Combiner を使用したケースと，Combiner を使用しないケースのパフォーマンスを比較する．

4. パフォーマンス検証結果と考察

4.1 スケールアウト

図 6 に示すように，Slave Node 数を 1 台から 3 台に増やすことでそのジョブ実行時間が 16 分 37 秒から 7 分 18 秒（44%）に，1 台から 9 台に増やすことで 3 分 17 秒（20%）に短縮された．Slave Node 数の増加に伴いジョブ実行時間が短縮されたが，1/3，1/9 のようにリニアな短縮までは至らなかった．これは，Hadoop が複数台のサーバーで超並列計算するがゆえに，別ノードへのデータ転送などに時間を要することが阻害要因であると推察される．まずは，Hadoop にてスケールアウトさせたとしても，この程度の摩耗が生じうることを認識しなければならない．ただし，理論上は，式1にてジョブ実行時間を算出することができ，データ転送などの阻害要因がなければノード数 = 総タスク数となるまでスケールアウトの効果を享受できる［4］．

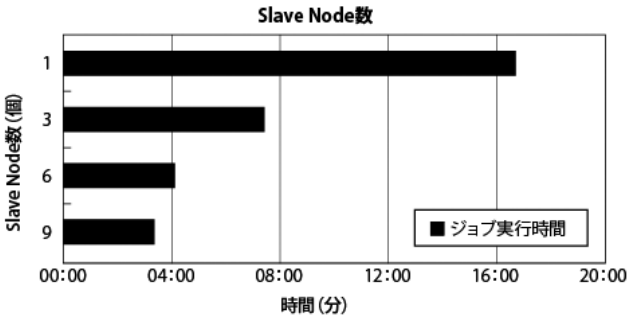

図6. スケールアウト検証結果

1 タスク当たりの処理時間 ×（総タスク数 ÷ 並列処理多重度）+ データ転送などの負荷（式 1）

また，Reduce 処理は，各 Slave Node の Map 処理が終了すると実行可能になるが，ある Slave Node の Map 処理に時間を要すると，Reduce 処理側で待ち時間が生じる．そのため，同じ Hadoop クラスター内においては，各 Slave Node のシステム・キャパシティーやネットワーク転送速度を同じにするなど，Slave Node の Map 処理時間を均等にする配慮が必要になる．

4.2 圧縮

Hadoop は，圧縮率や分割可否の異なる 4 種類の圧縮形式（gzip ／ bzip ／ zlib ／ LZO）を利用できる．

bzip は，4 つの中で圧縮率が一番高く，圧縮後にファイル分割できる圧縮形式である．Slave Node 数を 9 個と

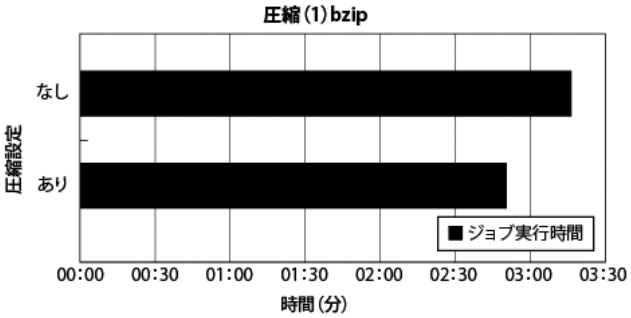

図7. 圧縮（bzip）検証結果

しても，9 ノードで MapReduce 処理を実行できる．bzip 圧縮なし／ bzip 圧縮ありの検証結果（図 7）を確認すると，ジョブ実行時間（= 黒色グラフ）が 3 分 17 秒から 2 分 51 秒（86%）に短縮した．劇的に短縮したとは言えないが，入力ファイルを bzip2 コマンドで圧縮するだけで設定が完了するため，手軽に使えるというメリットがある．反面，圧縮時間がかかるのが課題であり，10GB の解析データを bzip2 コマンドで圧縮するのに 1 時間程度を要した．gzip コマンドでは，10 分程度で圧縮処理が完了した．

gzip と zlib は，圧縮後にファイルを分割することができないため，HDFS 上への分散配置ができない．Slave Node 数を 9 個としても，1 ノードでしか MapReduce 処理が実行できない．つまり，単一ブロックのファイルにしか有効でなく，使用ケースが限定される．参考までに Slave Node 数が 1 台の環境で zlib にて Map 処理の入力ファイルを圧縮したところ，ジョブ実行時間が 15 分 48 秒から 10 分 2 秒（64%）に短縮された．

LZO は，bzip と同様に，圧縮後のファイル分割が可能である圧縮形式である．図 8 に示すように，圧縮しない場合の 3 分 1 秒に対して Map 処理の入力ファイルを圧縮すると 2 分 18 秒（76%）に，Map 処理の出力ファイルを圧縮すると 2 分 8 秒（71%）に，Reduce 処理の出力ファイルを圧縮すると 2 分（66%）にジョブ実行時間が短縮された．後述する Combiner と並び，最も効果が出たパフォーマンス・チューニング項目であった．ただし，LZO のライブラリーが GPL（GNU General Public License）ライセンス［12］であるため Apache のディストリビューションに含まれておらず，LZO コーデックの入手／設定／ビルドを別途実施する必要がある．また，解析データのサイズを，10GB から 1GB に変更し Map 処理の入力ファイルを圧縮したところ，ジョブ実行時間はほとんど変わらなかった．この結果より，入力ファイル・サイズがギガバイト・クラスでないと圧縮設定の効果が少ないと推察される．

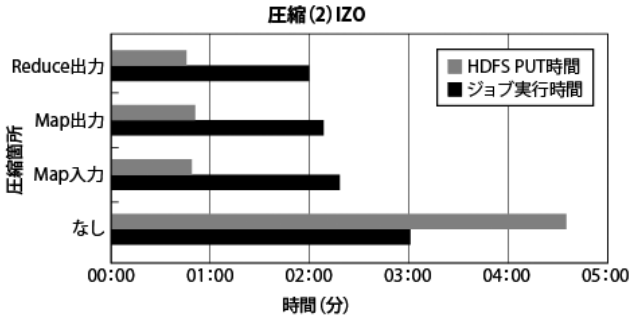

図8. 圧縮（LZO）検証結果

もう一点圧縮の効果として，どの圧縮形式を選択した場合でも，入力ファイルが圧縮されることで HDFS への PUT 時間が大幅に短縮された．圧縮形式が LZO の場合，図 8 に示すように，10GB ファイルの HDFS への PUT 時間（= 灰色グラフ）は，圧縮しない場合の 4 分 35 秒に対して 50 秒程度（18%）に短縮できている．ただし，圧縮処理自体に CPU リソースを使用するため，データ圧縮時の CPU 使用率に余裕があることが前提条件である．さらに，圧縮処理がマルチコアに対応していない点にも配慮した上で圧縮処理を行うノードを選定すべきである．1 ノードに複数の CPU を積載したとしても，圧縮時には 1 つの CPU しか使用できない．

4.3 JVM ヒープ・サイズ

最大ヒープ・サイズを 1024MB で固定し，最小ヒープ・サイズを 128MB ／ 512MB ／ 1024MB にて調整した結果が図 9 である．GC の実行時間（= 灰色グラフ）にばらつきがあったが，ジョブの実行時間（= 黒色グラフ）はほとんど変わらなかった．Web アプリケーションのようなオンライン処理と比較すると Hadoop によるバッチ処理はメモリー空間の使用方法が単純であるため，適切なヒープ・サイズである限り，ヒープの細かいチューニングはパフォーマンスへの影響は少ないと推察される．実際に，ジョブ実行中にヒープの拡大／縮小が繰り返されることはなかった．また，Heap Dump を取得し，オブジェクトの使用状況を確認したところ，ほとんどが Hash Table の Array，DataInputBuffer，DataOutputBuffer の 3 つで占められていた．

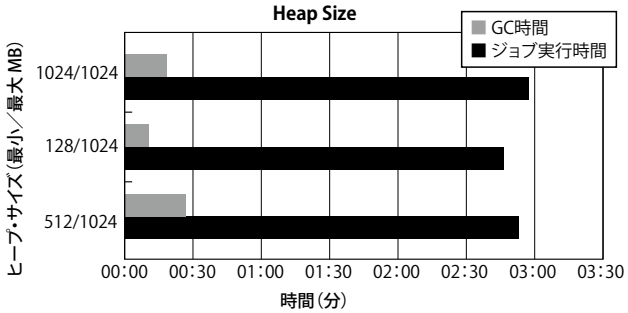

図9. JVMヒープ・サイズ検証結果

4.4 レプリカ数

レプリカ数は HDFS の構成ファイルである hdfs-site.xml 内の dfs.replication パラメーターで指定する．この数値はマスターも含むレプリカ数であり，2 はレプリカ 1 つ，1 はレプリカ無しである（デフォルト :3）．この値を 1 個／

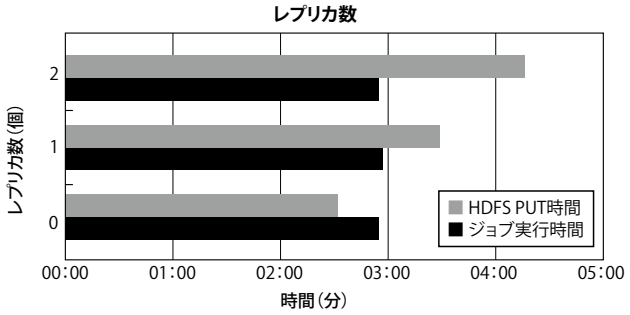

図10. レプリカ数検証結果

2 個／ 3 個と変更したジョブ実行時間の結果が図 10 である.

ジョブの実行時間（= 黒色グラフ）にほとんど差が出ていないのは，今回のジョブが入力は大量であるがユニーク・キーが少なく，HDFS への書き込みを行う Reducer の出力が極めて小さいためである．なお，Mapper 出力はローカル・ファイルであるため HDFS に書き込まない.

一方，レプリカ無しの 2 分 33 秒に対して 1 個のときが 3 分 29 秒（137%），2 個のときが 4 分 16 秒（167%）と劣化方面に有意な差が出ているのは，HDFS への PUT 時間（= 灰色グラフ）である．データの保全のためにはレプリカが必須であるが，レプリカ数の決定の際には Reducer の出力量や，HDFS への PUT 時間などへの配慮が必要となる.

4.5 Reducer 数

Reducer 数の変化によるジョブ実行時間への影響が図 11 である．デフォルトの 1 個に対して，Slave Node 数と同数の 9 個にすることで，2 分 55 秒から 2 分 27 秒（84% 程度）のジョブ実行時間に短縮された．前述の通り Reducer の出力は小さいが，shuffle 処理でソートされ，キーごとのブロックとなったレコードは元の行数と同じであるため，Reducer 自体の処理の負荷は大きい．Reducer

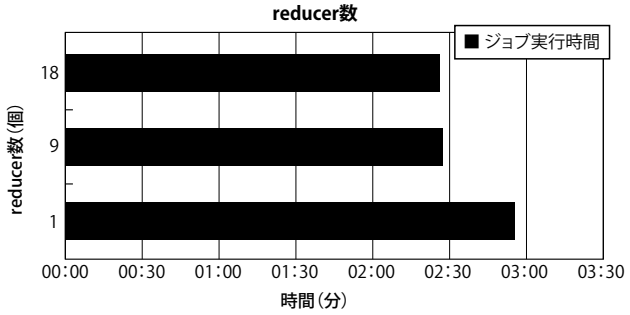

図11. Reducer数検証結果

が複数の場合，並行処理によりパフォーマンスが向上していると言える.

しかし，Reducer 数を 18 個にした場合には有意の差が見られない．これは，今回の最終的な出力行数，つまり入力レコードに含まれるユニークなキーの数が 7 個しか存在しないためである．shuffle を介することで 1 つのキーは必ずどれか 1 つの Reducer で処理されるため，データを処理できる Reducer は最大で 7 個しか存在しえない．つまり，Reducer 数を 9 個にした場合 2 個，Reducer 数を 18 個にした場合 11 個の Reducer はプロセスが起動するだけで処理すべきデータを受領できず，処理改善に貢献しない．むしろプロセスを起動する負荷の分,パフォーマンスが劣化する可能性もある.

また，今回のように Reducer を増やす場合，出力ファイルは part-nnnnn（part-00000，part-00001，part-00002）という形式で複数のファイルに分散される．Reducer が 1 つの場合と同じ出力を得るためには，ユーザー側の事後処理として union を行う必要がある．Reducer を増やす場合は，プロセスが稼働しうるノードのキャパシティーと，ユニーク・キー数，結果ファイルに対する配慮が必要である.

4.6 Combiner

Combiner は，Reducer 処理の Mapper ノードでの先行実施であると言い換えられる．当検証においては，図 12 に示すように，Combiner を追加することで Combiner なしと比較して，3 分から 2 分（67%）のジョブ実行時

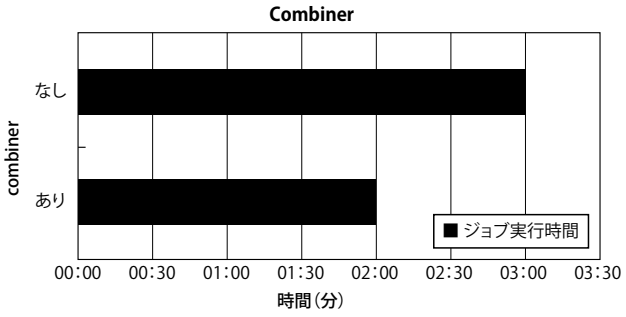

図12.Combiner検証結果

表1. データ型の変更

	改変前	改変後
Mapper 入力	<LongWritable, Text>	<LongWritable, Text>
Mapper 出力	<Text, FloatWritable>	<Text, FloatWritable>
Combiner 入力	<Text, FloatWritable>	<Text, FloatWritable>
Combiner 出力	<Text, Text>	<Text, FloatWritable>
Reducer 入力	<Text, FloatWritable>	<Text, FloatWritable>
Reducer 出力	<Text, Text>	<Text, Text>

間に短縮された. LZO の圧縮形式と並び，最も効果が出たチューニング項目である.

効果の理由は，Mapper の出力を Combiner で「事前 Reduce」することにより Shuffle 処理での Mapper → Reducer プロセスへのファイル転送量が削減され，ソート／マージ処理のサイズが縮小されたためである. 反面，Mapper ／ Reducer に加えて Combiner というアプリケーションの追加開発負荷への考慮が必要となる. Combiner は，Reducer と同じ処理を行うアプリケーションであるため，Mapper の出力と Reducer の出力にて同じデータ型を使う必要がある. 今回，Combiner を使用するに当たり，**表 1** に示すように，同じデータ型が使われていなかったため Reducer を改変する必要があった. Combiner は後からパラメーター変更で容易に対応できるチューニング項目ではないため，アプリケーション設計，コーディング時点での配慮が必要である. また，解析データのサイズを 10GB から 1GB に変更したところ，ジョブ実行時間はほとんど変わらなかった. 圧縮ケースと同様に，入力ファイル・サイズがギガバイト・クラスでないと combiner 設定の効果が少ないと推察される.

5. まとめ

今回のパフォーマンス検証により，Hadoop を利用した超並列計算において，以下のチューニング指針が得られた.

1）Slave Node 数の増加に伴い，ジョブ実行時間を短縮できるが，リニアに短縮できるわけではない.
2）各 Slave Node で Map 処理時間の隔たりがないかを確認する. 同じ Hadoop クラスター内において，各 Slave Node のシステム・キャパシティーやネットワーク転送速度を同じにするなどの事前配慮が必要である.
3）Hadoop クラスター内において，データ転送量を減らすチューニングを実施する. 特に，「圧縮の適用」と「Combiner の追加」による効果が大きい.

✓データ量がギガバイト・クラス以上の場合，「圧縮の適用」と「Combiner の追加」を検討する.
✓圧縮形式として，ファイル分割が可能な圧縮形式である bzip か LZO を選択する. さらに，よりジョブ実行時間を短縮したい場合には，LZO を選択する.

4）ジョブ実行時間以外に，圧縮時間や HDFS へのデータ PUT 時間など，トータルの処理時間を考慮する.

実際に次のチューニングを実施することで，ジョブ実行時間が大きく短縮され，その効果が明らかであることが証明された.

✓スケールアウト
1 台から 3 台に増加 :
16 分 37 秒→ 7 分 18 秒（44%）
1 台から 9 台に増加 :
16 分 37 秒→ 3 分 17 秒（20%）
✓圧縮（LZO）の適用
圧縮あり :3 分 1 秒→ 2 分（66%）
✓Combiner の追加
Combiner あり :3 分→ 2 分（67%）

今回の検証ですべてのチューニング項目を網羅できたわけではない. 例えば，投機的実行，ブロック・サイズの変更，SequenceFile の使用，Pig や Hive，HBase などのアプリケーションでのチューニングなど，未検証項目が残っている. とは言え，本稿で提示したチューニング項目は，Hadoop チューニングの第一歩として多くのシステムに適用可能である. また，弊社が提供する IaaS 型パブリック・クラウドである SBCE 上で Hadoop を実装する際には，当検証の内容／結果を適用することができる. さらに，検証した解析データのサイズは 10GB であるが，提示した4つのチューニング指針はデータ量に依存しない. テラバイト／ペタバイト・クラスの大規模データであっても，この指針を適用することが可能である.

2011 年 11 月 15 日に，Apache Hadoop の IBM 版ディストリビューションである InfoSphere BigInsights Enterprise Edition V1.3［13］が発表された. InfoSphere BigInsights は Hadoop をベースにしているため，ここでも本稿のチューニング指針を適用できる. 今後 Hadoop および InfoSphere BigInsights の実装を行う際には，本稿のチューニング指針を参考にされたい.

謝辞

本稿執筆にあたって，ISE. Web ソリューション開発部の万仲 龍樹氏には，大変有益なご助言をいただきました. あらためて深謝いたします.

参考文献

［1］ IDC: IDC iView "Extracting Value from Chaos" http://idcdocserv.com/1142
［2］ Apache Hadoop http://hadoop.apache.org/
［3］ Hadoop Sorts a Petabyte in 16.25 Hours and a Terabyte in 62 Seconds http://developer.yahoo.com/blogs/hadoop/posts/2009/05/hadoop_sorts_a_petabyte_in_162/
［4］ 平成21年度産学連携ソフトウェア工学実践事業 事業成果報告書 http://www.meti.go.jp/policy/mono_info_service/joho/downloadfiles/2010software_research/clou_dist_software.pdf
［5］ MapReduce: Simplified Data Processing on Large Clusters http://research.google.com/archive/mapreduce.html
［6］ Hadoop: a brief history http://research.yahoo.com/files/cutting.pdf
［7］ フローティングカーデータを核に自車の安全･安心から社会貢献へ http://wirelesswire.jp/special/201108/01/article/1.html
［8］ IBM developerWorks: Cloud computing http://www.ibm.com/developerworks/jp/cloud/devtest.html
［9］ IBMクラウド上の仮想サーバー環境 http://www.ibm.com/services/jp/igs/cloud-development/configurations.html
［10］Tom White: "Hadoop: The Definitive Guide" O'Reilly（2010）.
［11］太田一樹,下垣徹,山下真一,猿田浩輔,藤井達朗: "Hadoop 徹底入門" 翔泳社（2011）
［12］GNU General Public License - Wikipedia http://ja.wikipedia.org/wiki/GNU_General_Public_License
［13］InfoSphere BigInsights http://www.ibm.com/software/data/infosphere/biginsights/

日本アイ・ビー・エム
システムズ・エンジニアリング株式会社
Web プラットフォーム開発
IT スペシャリスト

清水 宣行 Nobuyuki Shimizu

［プロフィール］
2000年,日本IBM入社.WebSphere製品の技術サポートチームの一員として,WebシステムのPaaS基盤,SaaS連携に関するデザインおよび構築を担当.近年は,InfoSphere BigInsights,InfoSphere Streamsなど,ビッグデータ関連の技術支援も行っている.
snobu@jp.ibm.com

日本アイ・ビー・エム
システムズ・エンジニアリング株式会社
Web プラットフォーム開発
IT スペシャリスト

山口 崇 Takashi Yamaguchi

［プロフィール］
1994 年，日本 IBM 入社．IT スペシャリストとして WebSphere Application Server を中心に，ハイ・バリュー製品の技術支援を担当．近年は,InfoSphere BigInsights,InfoSphere Streams など,ビッグデータ関連の技術支援も行っている.
tyamag@jp.ibm.com

日本アイ・ビー・エム
システムズ・エンジニアリング株式会社
Web プラットフォーム開発
IT スペシャリスト

土田 誠 Makoto Tsuchida

［プロフィール］
1998 年，日本 IBM 入社．WebSphere 製品を中心としてさまざまな Web システムの構築・運用・技術支援を担当．2011 年から，InfoSphere BigInsights などビッグデータ関連の技術支援を行っている.
e28748@jp.ibm.com

日本アイ・ビー・エム
システムズ・エンジニアリング株式会社
ソフトウェア開発ソリューション
IT スペシャリスト

山口 亜希子 Akiko Yamaguchi

［プロフィール］
2010 年，日本 IBM システム・エンジニアリング（株）入社．Rational Team Concert の製品技術支援を担当．2011 年から，InfoSphere BigInsights，InfoSphere Streams など，ビッグデータ関連の技術支援も行っている.
AHA03999@jp.ibm.com