Vol.2011-HPC-129 No.6
2011/3/15

合成ベンチマークによるMapReduceのI/O性能評価手法

小川宏高[†1]　中田秀基[†1]　工藤知宏[†1]

MapReduceは，大規模なデータインテンシブ計算の実装手段として広範に用いられている．その単純なプログラミングモデルゆえ，MapReduceはデータインテンシブ計算のみならず，より一般的なHPC分野の並列分散アプリケーションを実装するためのプログラミングツール，謂わば“High-Performance MapReduce”処理系，としても利用されつつある．こうしたMapReduce処理系の高性能計算への適用を考えたとき，定量的な性能評価基準は必須であるが、MapReduceに関してはそのような基準が存在しない．我々はこうした問題を解決するために，合成ベンチマークによるMapReduceのI/O性能の評価を行うことを提案する．具体的には，MapReduceアプリケーションのデータ入出力をモデル化するとともに，各パラメータを任意に設定して実行できるベンチマークソフトウェアを実現する．本稿ではモデル化ならびに合成ベンチマークの実装の概要を示すとともに，SSDとSAS HDDを用いた小規模なクラスタでの実行結果を示す．

I/O Performance Evaluation on MapReduce by Using Composite Benchmarks

Hirotaka Ogawa ,[†1] Hidemoto Nakada [†1]
and Tomohiro Kudoh [†1]

MapReduce has been very successful in implementing large-scale data-intensive applications. Because of its simple programming model, MapReduce has also begun being utilized as a programming tool for more general distributed and parallel applications, e.g., HPC applications. Applying MapReduce to such HPC applications, quantitive metrics are indispensable for measuring MapReduce runtime performance. However, it seems that there are no feasible metrics in MapReduce. To resolve such situations, we propose conducting I/O performance evaluation by using synthetic benchmarks. To be more concrete, we first identify the I/O parameter set for a single MapReduce workload, and provide a benchmark application which can be customized for any parameter sets. In this paper, we first outline our synthetic benchmark software, and show the benchmark results in our experimental cluster with SSD and SAS HDD.

1. はじめに

MapReduce[1)]は，大規模なデータインテンシブアプリケーションの実装手段として知られ，実際にそのオープンソース実装であるHadoop MapReduce[2)]は広範に用いられている．その単純なプログラミングモデルゆえ，Hadoop MapReduceはデータインテンシブアプリケーションのみならず，より一般的なHPC分野の並列分散アプリケーションを実装するためのプログラミングツール，謂わば“High-Performance MapReduce”処理系，としても利用されつつある．

こうしたMapReduce処理系の高性能計算への適用を考えたとき，定量的な性能評価基準は必須であると言ってよい．ある所定の計算を一定以上の性能で実行するのにどれだけの計算能力，記憶容量，ネットワークが必要なのか．あるいは逆に所定の計算機リソースで所定の計算を行うのにどれだけの性能が期待できるのか．それが見積もれなければ，MapReduce処理系の優劣を議論することもできないし，MapReduceに適した，あるいは特化したHPCシステムを構築することができない．

ところが，一般のHPCアプリケーションに関してはLinpackやNAS Parallel Benchmarksがde factoな性能評価基準となり得ているのに対して，MapReduceに関してはそのような基準が存在しない．例えば，処理系の研究開発に関する多くの文献において，（著者ら自身も含めて）実装者が自ら定めたWordCountやK-meansなどのベンチマーク結果を示しているが，比較可能なメトリックがなければそれらは「テストケース」の実行結果を示しているに過ぎず，より広範に利用されつつあるMapReduce処理系の評価として有益かどうかについても疑問が残る．

我々はこうした問題を解決するために，合成ベンチマークによるMapReduceのI/O性能の評価を行うことを提案する．具体的には，MapReduceアプリケーションのデータ入出力をモデル化するとともに，各パラメータを任意に設定して実行できる合成ベンチマークソフトウェアを実現する．

本稿では，モデル化ならびにベンチマークの実装の概要を示すとともに，Fusion-io ioDrive

†1 産業技術総合研究所 / National Institute of Advanced Industrial Science and Technology (AIST)

Duo と SAS HDD を用いた小規模なクラスタでの実行結果を示す.

2. MapReduce の実行モデル

MapReduce は一般に，キーバリューペアのリストデータを処理するためのプログラミングモデルとその分散実装を指す.

MapReduce のプログラミングモデルは，入力キーバリューペアのリストを受け取り，出力キーバリューペアのリストを生成する分散計算モデルである．MapReduce の計算は，Map と Reduce という二つのユーザ定義関数からなる．Map 関数は 1 個の入力キーバリューペアを取り，0 個以上の中間キーバリューペアを生成する．MapReduce のランタイムは中間キーバリューペアを中間キーごとにグルーピングし，Reduce 関数に引き渡す．Reduce 関数は中間キーと，そのキーに関連付けられたバリューのリストを受け取り，0 個以上の結果を出力する．各 Map 関数，Reduce 関数はそれぞれ独立しているため，同期なしに並列に実行することができる.

オープンソース実装の代表例である Hadoop を例に，MapReduce の分散実装の概要を**図 1** に示す.

Hadoop では，まず MapReduce ジョブへの入力をスプリット[⋆1]と呼ばれる固定長の断片に分割し，スプリット内のレコードに対して map 関数を適用する．この処理を行うタスクを Map タスクと呼び，各スプリットに関して並列に実行される．map 関数が出力した中間キーバリューデータは，partition 関数（キーのハッシュ関数など）によって R 個に分割され，各パーティションごとにキーについてソートされる．ソートされたパーティションはさらに combiner と呼ばれる集約関数によってコンパクションされ，最終的には Map タスクが動作するノードのローカルディスクに書き出される.

一方の Reduce 処理では，まず Map タスクを処理したノードに格納されている複数のソート済みパーティションをリモートコピーし，ソート順序を保証しながらマージする.（結果的に得られた）ソート済みの中間キーバリューデータの各キーごとに reduce 関数が呼び出され，その出力は HDFS 上のファイルとして書き出される.

⋆1 スプリットのサイズが小さければ負荷分散が容易になる一方，スプリットの管理とタスク生成のオーバーヘッドが顕著になる．このため，スプリットのサイズは通常バックエンド分散ファイルシステム HDFS のブロックサイズと等しくなるようになっている.

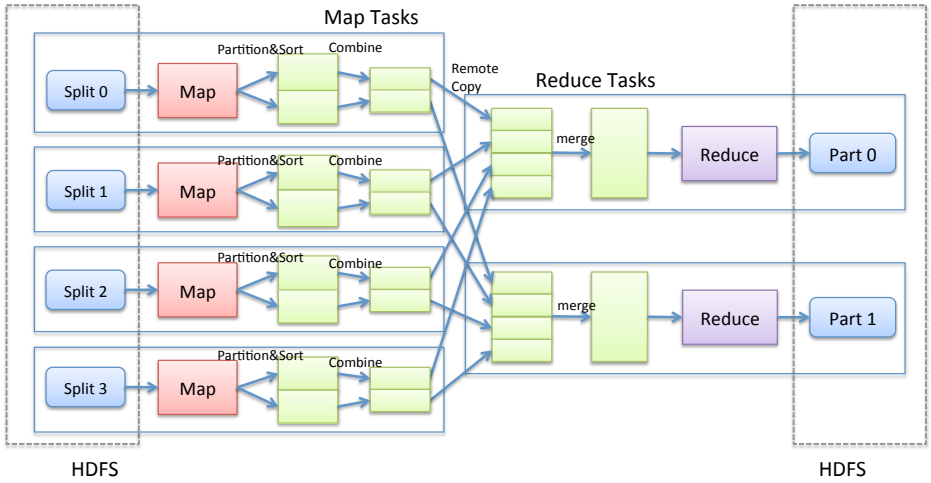

図 1 MapReduce Execution

3. MapReduce 合成ベンチマーク

3.1 MapReduce 入出力ワークロード

ここでは Hadoop MapReduce の実装に即して MapReduce アプリケーションの入出力ワークロードをモデル化する．まず，単純化のために以下のような仮定をする:

- map 関数の入力キーバリューペア 1 個，および 1 個のキーに対する reduce 関数の出力を分散ファイルシステム上のファイル 1 個で表現するものとする．Hadoop では，例えばファイル名にキー，ファイルコンテントに値を格納することで容易にこの表現が実現できる.
- ファイルは単一のブロックに格納され，つまりブロックに分割されないものとする．また，map 関数への入力時に単一の入力スプリットとして取り扱うものとする.
- バリューのサイズは各フェーズにおいて一定であるものとする.
- ファイルの分散は平均化されており，ストレージノード間での偏りが無視できるものとする.
- Reduce タスクはストレージノードの数だけ実行されるものとする．言い換えると，最大の並列度で Map タスク，Reduce タスクとも実行されるものとする.

これらの前提は，HDFS 以外のファイルシステムとの比較や Hadoop 以外の MapReduce システムとの比較を容易にするためのものである．また，MapReduce アプリケーションのワークロードの一般的な特性を損なうこともない．

以下では，MapReduce アプリケーションのワークロードを規定するパラメータを順次説明する．

- nodes
  ノード数．ストレージノード数，Map タスクが起動されるノード数，Reduce タスクの数に等しい．
- initialKeyCount
  入力キーの個数．ノードあたりの入力キーバリューペアの個数は (initialKeyCount/nodes) に等しい．
- initialValueLength
  入力キーに対応する値のサイズ．
- mapoutKeyCount
  map 関数が生成する中間キーの個数．ノードあたりの中間キーバリューペアの個数は (mapoutKeyCount/nodes)，一回の map 関数の出力する中間キーバリューペアの個数は (mapoutKeyCount/initialKeyCount) にそれぞれ等しい．
- mapoutValueLength
  map 関数が生成する中間キーに対応する値のサイズ．
- mapoutUniqueKeyCount
  map 関数が生成する中間キーのうち，ユニークなものの個数．この値は reduce 関数の入力キー，ならびに出力キーの個数に等しい．また，各 map 関数は，同一のキーを持つ中間キーバリューペアを ((mapoutKeyCount/initialKeyCount)/mapoutUniqueKeyCount) 個生成する．
- reduceoutValueLength
  reduce 関数が生成する値のサイズ．

これらの他に partition 関数，combiner の振る舞いも規定できる．

- partitionMethod
  中間キーを partition する際の方法．各 map 関数が生成する中間キーが平均的に Reduce タスクに分散される *average*，中間キーがローカルに起動される Reduce タスクに分散される *local*，中間キーが隣接ノードに起動される Reduce タスクに分散される *shifted*（必ず中間キーバリューデータの全交換が発生する）などを取り得る．ただし，Hadoop の実装では，局所性を利用した Reduce タスクの割り付けはできないため，local や shifted の実現は不可能である．
- combinerEnabled
  combiner の適用の可否の指定．combiner 自体は reduce 関数と同一の関数が適用されるものとする．ただし，Hadoop の実装では，combiner 関数の適用を強制することは不可能なため，適用を保証するものではない．
- combiningFactor
  combiner によってソート済みのパーティションが集約される比率．上述の通り，Hadoop では制御できない．

3.2 合成ベンチマークの Hadoop 実装

合成ベンチマークは，Hadoop 0.20.2 の Java アプリケーションとして記述してある．すべてのキーバリューペアはキーを long 型で，値を byte 配列型でそれぞれ表現している他は，実装においてとりたてて技術的に困難な点はない．

図 2 に示したように，3.1 で述べたパラメータを Configuration ファイルとして与えることで動作をカスタマイズすることができる．

4. 評　　価

3 で述べた合成ベンチマークを用いて特徴的な 3 つの MapReduce の I/O ワークロードを再現し，実クラスタ環境での性能を評価した．以下では評価環境について説明した上で，ワークロードごとの実験結果を示す．

4.1 評 価 環 境

評価には，**表 1** に示すように，1 台のマスターノードと 16 台のワーカーノード（ストレージノードと MapReduce 実行ノードを兼ねる）からなる小規模クラスタを用いた．各ノードは 10Gbit Ethernet で接続され，各ワーカーノードは Fusion-io ioDrive Duo 320GB と Fujitsu の 147GB SAS HDD を備えている．

使用している Hadoop のバージョンは，Cloudera Distribution for Hadoop の 0.20.2+320 である．dfs.replication を 1 に設定することで HDFS のレプリカ生成を抑制している．また，各 Map タスク，Reduce タスクが使用できるヒープのサイズは 2GB に設定してある．

4.2 ワークロード 1: read-intensive

このワークロードでは，入力データが多く，相対的に Map タスクの処理がドミナントに

```
<?xml version="1.0"?>
<configuration>
  <property>
    <name>iocomposite.nodes</name>
    <value>4</value>
  </property>
  <property>
    <name>iocomposite.initialKeyCount</name>
    <value>10</value>
  </property>
  <property>
    <name>iocomposite.initialValueLength</name>
    <value>2000</value>
  </property>
  <property>
    <name>iocomposite.mapoutKeyCount</name>
    <value>1000</value>
  </property>
  <property>
    <name>iocomposite.mapoutUniqueKeyCount</name>
    <value>256</value>
  </property>
  <property>
    <name>iocomposite.mapoutValueLength</name>
    <value>16384</value>
  </property>
  ...
</configuration>
```

図 2 Configuration ファイル

表 1 Benchmarking Environment

Number of nodes	17 (*)
CPU	Intel(R) Xeon(R) W5590 3.33GHz
Number of CPU per node	2
Number of Cores per CPU	4
Memory per node	48GB
Operating System	CentOS 5.5 x86_64
Storage (ioDrive)	Fusion-io ioDrive Duo 320GB
Storage (SAS HDD)	Fujitsu MBA3147RC 147GB/15000rpm
Network Interface	Mellanox ConnextX-II 10G Adapter
Network Switch	Cisco Nexus 5010

(*) one node is reserved for the master server.

なるアプリケーションを再現する．具体的には入力データの総量が 16GiB となるように，initialKeyCount と initialValueLength の組み合わせを選んで，合成ベンチマークの実行時間を測定した．以下にパラメータを示す．

- initialKeyCount: 16｜64｜256｜1024｜4096｜16384｜65536
- initialValueLength: 1GiB｜256MiB｜64MiB｜16MiB｜4MiB｜1MiB｜256KiB
- mapoutKeyCount: 256
- mapoutUniqueKeyCount: 16
- mapoutValueLength: 1
- reduceoutValueLength: 1
- combinerEnabled: false

図 3 に結果を示す．ioDrive と SAS HDD の明らかな差異が見られない．一方，initialKeyCount が 4096 以上，initialValueLength が 4MiB 以下で大幅な性能の低下が見られることが分かる．これは入力スプリットの管理と Map タスクの生成に要するオーバーヘッドの増加によるもので，ベンチマークは妥当な結果を示したと言ってよい．

4.3 ワークロード 2: write-intensive

このワークロードでは，出力データが多く，相対的に Reduce タスクの処理がドミナントになるアプリケーションを再現する．具体的には出力データの総量が 16GiB となるように，mapoutUniqueKeyCount と reduceoutValueLength の組み合わせを選んで，合成ベンチマークの実行時間を測定した．以下にパラメータを示す．

- initialKeyCount: 16
- initialValueLength: 1

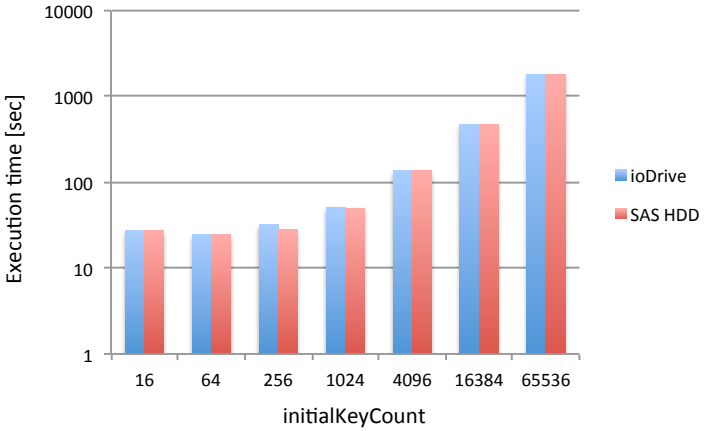

図 3 Read-intensive MapReduce workload

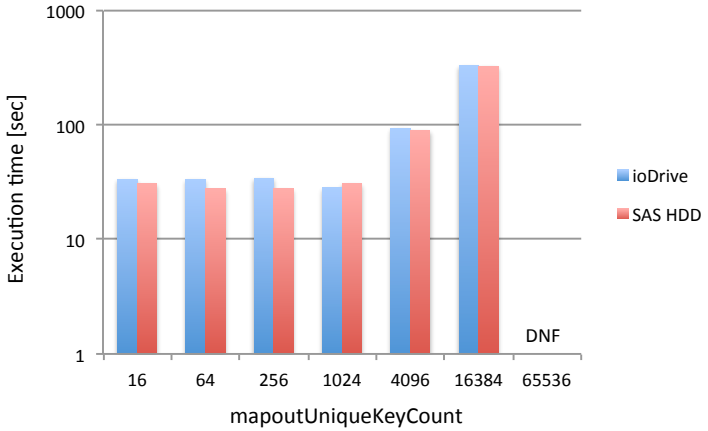

図 4 Write-intensive MapReduce workload

- mapoutKeyCount: 4194304
- mapoutUniqueKeyCount: 16 | 64 | 256 | 1024 | 4096 | 16384 | 65536
- mapoutValueLength: 1
- reduceoutValueLength: 1GiB | 256MiB | 64MiB | 16MiB | 4MiB | 1MiB | 256KiB
- combinerEnabled: false

図 4 に結果を示す．read-intensive と同様，ioDrive と SAS HDD の明らかな差異が見られない。一方で，mapoutUniqueKeyCount が 4096 以上，initialValueLength が 4MiB 以下で性能の低下が見られることが分かる．これは Reduce タスクの生成と出力データの管理に要するオーバーヘッドの増加によるもので，ベンチマークは妥当な結果を示したと言ってよい．

また，mapoutUniqueKeyCount を 64K 以上に設定すると，Reduce タスクが同時にオープンできるファイル数を超えるためにベンチマークは完走できなかった．

4.4 ワークロード 3: shuffle-intensive

このワークロードでは，入出力データそのものは小さいが，シャッフルに要する処理がドミナントになるアプリケーションを再現する．具体的には Map タスクの出力キーバリューペアの総量が 16GiB となるように，mapoutKeyCount と mapoutValueLength の組み合わせを選んで，合成ベンチマークの実行時間を測定した．以下にパラメータを示す．

- initialKeyCount: 16
- initialValueLength: 1
- mapoutKeyCount: 16 | 64 | 256 | 1024 | 4096 | 16384 | 65536 | 262144
- mapoutUniqueKeyCount: 16
- mapoutValueLength: 1GiB | 256MiB | 64MiB | 16MiB | 4MiB | 1MiB | 256KiB | 64KiB
- reduceoutValueLength: 1
- combinerEnabled: false | true

図 5 に結果を示す．read-intensive，write-intensive と同様，ioDrive と SAS HDD の明らかな差異は見られない．combiner の有無で見ると，mapKeyCount が 1024 以上では combiner によって 30%ほど性能が向上していることが分かる．また，mapKeyCount が小さい領域では，combine するのに十分な数の中間キーバリューデータを Map タスクが保持していないため，むしろ combiner がオーバーヘッドになっていることが分かる．

5. 関 連 研 究

（株）エヌ・ティ・ティ・データによる報告[3] では，Hadoop パッケージに付属している PiEstimator（モンテカルロ法による π 計算），TeraSort（整数ソート）を用いて，それぞれ CPU，I/O の処理性能を同定するとともに，Hadoop の各種パラメータをチューニング

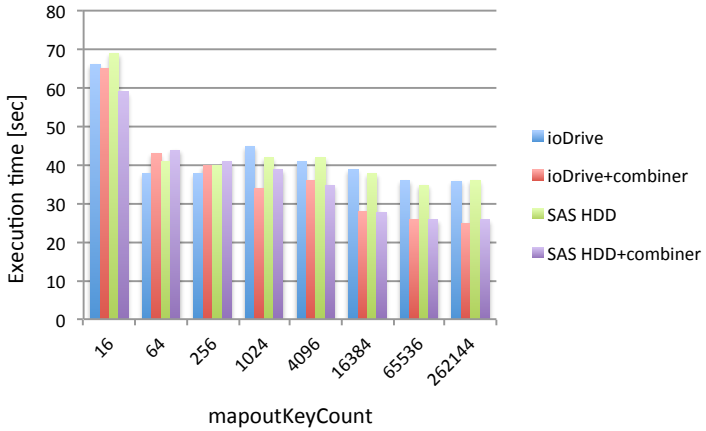

図 5 Shuffle-intensive MapReduce workload

する手法について述べられている．また，TeraSort の小規模データによる Map，Reduce 処理時間に関するプロファイル情報から，大規模データでの TeraSort の実行時間を推定する方法について述べられている．彼らの方法は有意義だが，特定のアプリケーションに特化しており，既知でない I/O パターンを持つアプリケーションの性能を予測したりする目的には用いることができない．

6. ま と め

MapReduce 処理系の優劣を比較可能にすること，また MapReduce に適した（もしくは特化した）HPC システムの構築を支援することを目的として，合成ベンチマークの設計・実装を行い，小規模なクラスタ上で Hadoop での性能評価を行った．我々の合成ベンチマークでは，MapReduce アプリケーションで想定される概ね任意の I/O ワークロードを再現することができる．

他の処理系との比較が第一の目的であるので，今後の課題はもちろん，我々の研究開発している SSS 処理系[4] 上にポーティングし，Hadoop との詳細な性能比較を行うことである．

謝　　辞

本研究の一部は，独立行政法人新エネルギー・産業技術総合開発機構（NEDO）の委託業務「グリーンネットワーク・システム技術研究開発プロジェクト（グリーン IT プロジェクト）」の成果を活用している．

参 考 文 献

1) Dean, J. and Ghemawat, S.: MapReduce: Simplified Data Processing on Large Clusters, *Communications of the ACM*, Vol.51, No.1, pp.107–113 (2008).
2) Apache Hadoop Project: Hadoop, `http://hadoop.apache.org/`.
3) 株式会社エヌ・ティ・ティ・データ：平成 21 年度産学連携ソフトウェア工学実践事業 (高信頼クラウド実現用ソフトウェア開発 (分散制御処理技術等に係るデーターセンターの高信頼化に向けた実証事業)) 事業成果報告書，`http://www.meti.go.jp/policy/mono_info_service/joho/downloadfiles/2010software_research/clou_dist_software.pdf` (2010).
4) Ogawa, H., Nakada, H., Takano, R. and Kudoh, T.: SSS: An Implementation of Key-value Store based MapReduce Framework, *Proceedings of 2nd IEEE International Conference on Cloud Computing Technology and Science (1st International Workshop on Theory and Practice of MapReduce)*, pp.754–761 (2010).