DEIM Forum 2012 D6-4

並列データインテンシブ処理基盤のI/O性能評価に関する実験的考察

山田　浩之† 　合田　和生† 　喜連川　優†

† 東京大学生産技術研究所　〒153–8505 東京都目黒区駒場 4-6-1
E-mail: †{hiroyuki,kgoda,kitsure}@tkl.iis.u-tokyo.ac.jp

あらまし　企業が保持する情報は今後急速に増加することが予想され, データ解析による大規模データからの価値創出に対する期待は高い. 近年, 大規模データ解析においてはオープンソースの並列データインテンシブ処理基盤が広く利用され, 多様な計算機環境への導入が進みつつある. 本稿では, オープンソースの並列データインテンシブ処理基盤のI/O性能評価を実施し, 結果を考察する.

キーワード　性能評価, 大規模データ解析, 並列データインテンシブ処理基盤

An Experimental Study on I/O Performance in Parallel Data Intensive Platform

Hiroyuki YAMADA†, Kazuo GODA†, and Masaru KITSUREGAWA†

† Institute of Industrial Science, The University of Tokyo
4-6-1 Komaba, Meguro-ku, Tokyo, 153–8505 Japan
E-mail: †{hiroyuki,kgoda,kitsure}@tkl.iis.u-tokyo.ac.jp

Abstract　The digital information is rapidly increasing and analysis for large-scale data is becoming more important. Recently, parallel data intensive platform is widely deployed for the large-scale data analysis. In this paper, we examined the I/O performance of an open source parallel data intensive platform.

Key words　Experimental Study, Large-scale Analytical Processing, Parallel Data Intensive Platform

1. はじめに

企業が保持する情報は大規模化しつつあり, データ解析による大規模データからの価値創出への期待は高い. IDCの報告[1]によると, 2010年段階では1EB（エクサバイト）を超えるデジタル情報が既に生成され, 2015年には8EBにも届く勢いであることが予測されている. データ量の増加に伴い, 解析対象のデータも大規模になりつつあり, eBayでは, 40PBものデータウェアハウスの運用が報告されている[2]. また, 米国でのスマートグリッドプロジェクトにおいては, スマートメータ等のセンサー機器から年間1EBもの解析対象データの生成が予測されている[3]. このように, 今後はこれまでの想像をはるかに超える大規模なデータの解析を行う必要があり, より高速・高効率なデータ解析処理基盤が求められている.

企業における大規模データ解析は, 大規模ストレージに接続されたハイエンドサーバ等の集約ステムにより従来から行われてきた. 他方, 近年では多数のコモディティサーバ, OS, DBMSを一体化したデータウェアハウス・アプライアンスの登場により, 並列処理による解析が行われ始めている. データウェアハウス・アプライアンスは, 多数のコモディティサーバを利用した並列データベースアーキテクチャを採用し, 構造化データ処理に対して優位であることを特徴としている. データ解析技術としては1990年代までの並列データベース技術を応用したものに過ぎないと考えられるが, データ解析システムを安価にパッケージ化することで, 大規模データ解析が一般企業へと普及する大きなきっかけとなった.

並列処理による大規模データ解析が普及しているなか, 並列データインテンシブ処理基盤のHadoop[4]により, この流れは加速しつつある. HadoopはGoogle社のMapReduce[5]のオープンソース実装であり, 非構造化データ処理の柔軟な扱いを特徴としている. また, YahooやFacebookでは数千ノードでの使用実績があるなど, スケーラビリティに対しても優位性が高い. さらに, Cloudera等のディストリビューションにより, その導入や管理コストも大きく低下していることから, ウェブ企業でのデータ解析におけるデファクトスタンダードとなりつつある. 各社データウェアハウスベンダにおいても, 独自のデータ解析処理基盤とHadoopとの連携が進められていることもあり, エンタープライズ分野においてもその活用が進んでいる.

近年のHadoopによる大規模データ解析においては, I/O帯域を確保するために, 各ノードに多数のディスクドライブを搭

載する流れが進んでいる [6], [7]. しかし, 現状の Hadoop は必ずしも I/O 処理能力が高くない [8]. 本稿では, TPC-H データセットを用いた並列データインテンシブ処理基盤 Hadoop の I/O 性能評価を行うことにより, Hadoop の I/O 処理能力を検証するとともに, その実行アーキテクチャについて考察する.

2. 実験環境

実験に用いたハードウェア構成を以下に述べる. Hadoop 用サーバとして IBM System x3850 M2 を 1 台と, データ用ストレージとして AMS500 を用いた. 当該サーバは, Intel Xeon X7350 を 4 ソケット, メモリを 48GB(DDR2 ECC SDRAM RDIMM), ローカル HDD として SAS ディスク (10Krpm) を 4 台搭載している. ローカル HDD は OS と Hadoop の一部の管理データの保存用に用いた. また, ストレージは 32 台の FC ディスク (10Krpm) を使用した. 2 台の FC ディスクを LU(RAID0) とし, 16LU を構築した. サーバ・ストレージ間は 4 本の 4Gbps ファイバチャネルにより接続されている. ハードウェア構成の概要を図 1 に示す.

次に, 実験に用いたソフトウェア構成を以下に述べる. OS は Linux(2.6.18), Hadoop はバージョン 0.20.2 を使用した. また, 比較として商用の MapReduce 処理系(注1)を使用した. HDFS のブロックサイズは 128MB, I/O のバッファサイズ (io.file.buffer.size) はデフォルト値の 4KB とした. また, 特に言及がない限り, Mapper 数は 16 とした(注2).

実験においてはソフトウェアのレベルで複数の LU を扱う必要があるが, 今回は以下の 2 つのストライピング方法を用いた.

- LVM のストライピングを使用する方法
- Hadoop 自体の分散 I/O 機構を利用する方法

1 つ目は, Hadoop は 1 つの論理ディスクドライブを扱い, LVM が複数の LU を使い分ける方法となる. 2 つ目は, LU ごとに異なるディレクトリにマウントし, HDFS のデータ保存用ディレクトリ (dfs.data.dir) と MapReduce 処理の中間結果用のディレクトリ (mapred.local.dir) に複数ディレクトリを指定し, Hadoop 自体が複数の LU を使い分ける方法となる. 商用の MapReduce 処理系においては 1 つ目の方法のみを用いた.

実験データは TPC-H を用いた. スケールファクタを 100 とし, LINEITEM 表 (約 75GB) を使用した. 実験クエリは, 以下の 3 つの問い合わせを実施した.

- Job1: 1Map ジョブ (map 処理は空)
- Job2: 1Map ジョブ (map 処理は選択処理)

Job1 は, LINEITEM 表を入力とし, map 処理は何も行わない. つまり, Hadoop のフレームワークを介して LINEITEM 表を全スキャンするのみの処理となる. Job2 は, LINEITEM 表を入力とし, map 処理は shipdate カラムに対する選択処理 (選択率 1%) を行う. 選択結果の HDFS への書き込みは行わない.

(注1)：商用の MapReduce 処理系は, Hadoop の分散ファイルシステムを C++により書き換えたものとなっている. MapReduce 処理層は, Hadoop と同様の Java プログラムを使用している.

(注2)：CPU コア数と同数に設定するのが推奨されているため.

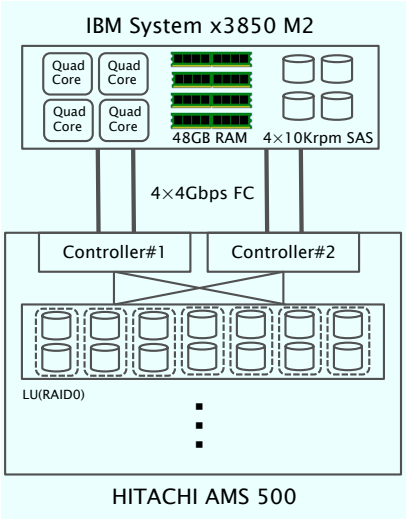

図 1　実験に使用したハードウェア構成

3. 実験結果

実験は, 1LU(2D), 2LU(4D), 4LU(8D), 8LU(16D), 16LU(32D) の 5 つのディスク構成に対して実施した. 図 2 には, Job1, Job2 に対する, 各ディスク構成における読み取りの I/O 転送レートを, 図 3 には, それぞれの CPU 使用率を示している. 図 2 おいては, 比較として, LVM による論理ディスクドライブを Raw デバイスとして扱った場合の, 単一ストリームによる読み取り I/O 転送レートを示している. Raw による I/O 転送レートがディスクドライブ 16 台以上で飽和しているのは, ストレージ側のコントローラの性能によるものである. 全体として, Hadoop および商用 MapReduce 処理系の実行における I/O 転送レートは, ディスクドライブ本来の性能よりも大幅に下回った結果であることがわかる. また, ディスク台数の増加に対して性能向上率は低下の傾向にあり, 特に Job2 においては, 16 台以降で性能が飽和していることがわかる. これは, ディスク台数の増加により I/O 時間が短縮し, 各ジョブにおける CPU 処理が占める時間の割合が増加していることが原因であると考えられる. つまり, Job2 においては 32 ドライブ使用時で CPU 使用率は約 90%(注3) となっており, 性能が CPU により律速されつつあることが, I/O 帯域が飽和している原因であると考えられる. Job1 においても, CPU 使用率は 50%程度であり, I/O 帯域も 400MB/s に満たないことから, いずれにジョブにおいても, I/O 帯域および CPU リソースには空きがあり, これらを完全に活用できていないという問題が確認できる.

当該問題は他の要因により性能が律速されることが原因であることが考えられる. 図 4 に, Mapper 数を 8, 16 とした場合の Job2 において発行された I/O をトレースを示す. Mapper 数が 8 の場合 (図 4 左) に対して, Mapper 数が 16 の場合 (図 4 右) は, ディスクドライブのシーク距離が増加していることがわかる. これは, 各 Mapper は独立にシーケンシャル I/O をバッ

(注3)：16CPU コアを最大限に使用する場合を 100%とする.

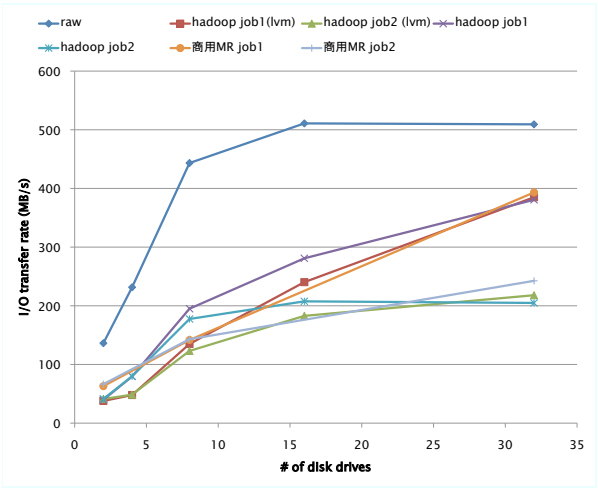

図 2　ディスクドライブ台数に対する Job1, Job2 の I/O 転送レート

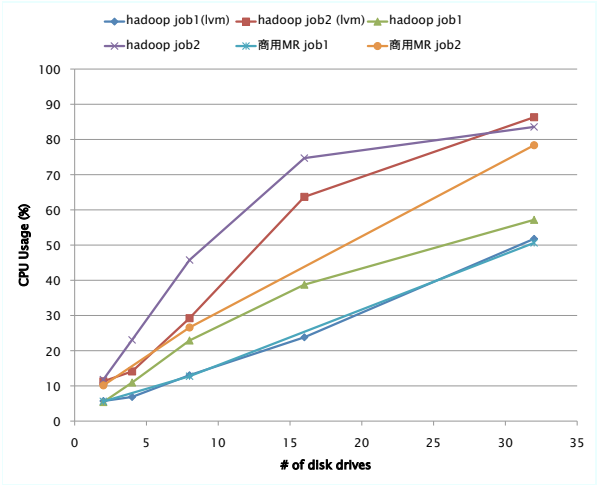

図 3　ディスクドライブ台数に対する Job1, Job2 の CPU 使用率 (user+sys)

ファサイズ単位(注4)で発行しているため, ディスクドライブに対しては完全なシーケンシャルアクセスにはならず, Mapper 数 (同時 I/O ストリーム数) の増加により, シーケンシャリティは低下するためだと考えられる. つまり, 前述の性能律速は, 複数ストリーム I/O によるディスクシークの増加から起因される問題であると考えられる.

一般的に, 前述のような複数ストリーム I/O による性能律速の問題に対しては, I/O バッファサイズの増加, または, I/O のスケジューリングにより解決が図られる. 以下に, それぞれの解決策を Hadoop において検証した結果を示す.

図 5 に I/O バッファサイズを変化させた時の I/O 転送レートを示す. I/O バッファサイズの増加に伴い, I/O 転送レートが低下の傾向にあることがわかる. また, 32MB の時は, 大きな性能低下が確認された. これは, 各 Mapper の逐次的な I/O 処理モデル(注5) が起因していると考えられる. つまり, 小さい

(注4)：デフォルトは 4KB.

(注5)：Hadoop の各 Mapper は I/O 処理と CPU 処理を同時には実行せず, 逐次的に実行することで処理を進めるデータ処理アーキテクチャとなっている.

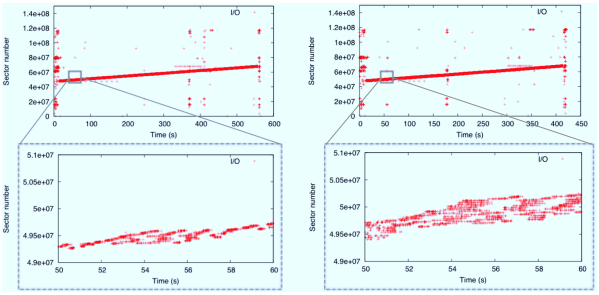

図 4　Hadoop ジョブの I/O トレース (左: 8Mapper, 右, 16Mapper)

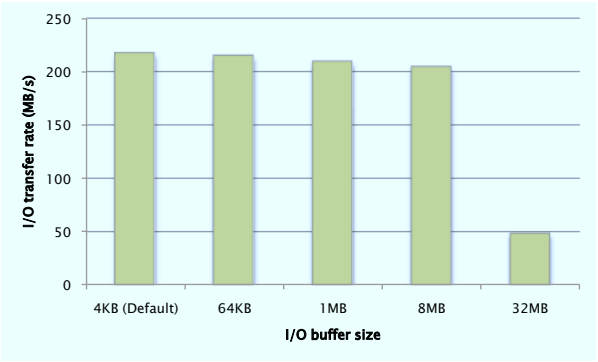

図 5　I/O バッファサイズにおける I/O 転送レート (Job2)

単位で I/O を発行した場合は, I/O 発行の間隔が短いことから OS の先読み機構が効率的に機能し, ディスクドライブに対するシーケンシャリティの増加に加えて, I/O 処理が CPU 処理にオーバーラップして実行される可能性が高いが, 大きい単位で I/O を発行した場合は, ディスクドライブに対するシーケンシャリティは増加するものの, I/O 発行の間隔が長いため OS の先読み機構が効率的に機能せず, I/O 処理と CPU 処理が逐次的に処理されるため, 小さい単位で I/O を発行した場合と比べて性能が低下してしまうと考えられる. このことから, 現状の Hadoop の I/O 処理モデルでは, I/O バッファサイズの増加のアプローチは当該問題に対する解決策とはならないと考えられる.

次に, Linux の既存の 4 つの I/O スケジューラを変化させた時の Job2 における I/O 転送レートを図 6 に示す. この結果からわかるように, I/O スケジューラによる I/O 転送レートの違いはほとんど見られない. つまり, 現状の Linux I/O スケジューラでは当該問題をうまく解決できていないと考えられる.

3.1　Mapper の I/O 処理内容に基づいたスケジューリングの効果検証

当該問題に対して, Mapper の I/O 処理内容に基づいたスケジューリングの有効性を検証する. Mapper の I/O 処理内容に基づいたスケジューリングとは, 各 Mapper は Split 単位に I/O 処理を行うという事前知識に基づき(注6), I/O 先読みおよび I/O の並び換えを行うことを指す. つまり, 各 Mapper が Split の先頭を読み始めた時点で, その位置から Split 分のデータを読

(注6)：Hadoop は, 通常 HDFS のブロックを Spilt とし, Split ごとに Mapper を割り当てて処理を進める.

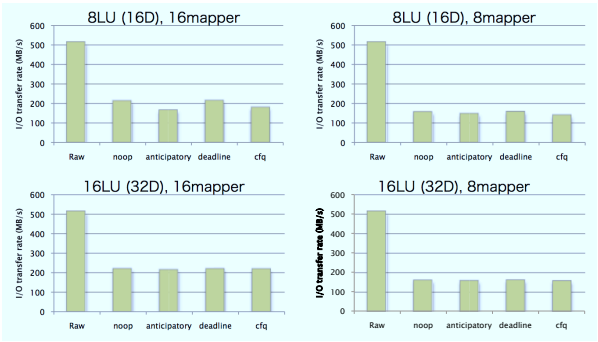

図 6　Linux の I/O スケジューラにおける Job2 の I/O 転送レート

み込むことがわかるため, Split サイズ分のデータの先読みをスケジュールすることが可能となる．これにより, 各 Mapper が map 処理 (CPU 処理) をしている間に, 後の map 処理で必要なデータの I/O 処理をバックグラウンドで実行することができ, I/O 帯域の利用効率の向上が期待できる．

当該スケジューリング機構の検証結果を以下に示す．Hadoop ジョブの I/O トレースに基づいた再生プログラムに対して, 前述のスケジューリング機構を組み込み, 検証を実施した．図 7 に各ジョブにおける I/O 転送レートを示す．Job1(図 7 上) では, 当該スケジューリング機構によりほぼディスクドライブの帯域に近い性能が出ていることがわかる．また, Job2(図 7 下) においても同様に, 当該スケジューリング機構により I/O 転送レートは大幅に向上していることがわかる．ディスクドライブの帯域にまで達していない原因は, Job2 は選択処理も含み Job1 より CPU 処理が多く, この時点で性能が CPU により完全に律速してしまったためである．

4. 関 連 研 究

MapReduce 処理系の性能評価に関しては, これまでも幾つかの議論がなされてきた．Pavlo ら [9] は Hadoop と従来の並列データベースとの性能比較により, MapReduce 処理系の全スキャン指向のアーキテクチャや解析時のパース処理などの欠点を指摘した．これに対して, Google の MapReduce の開発者である Dean ら [10] は, これらの指摘は MapReduce 処理モデルの本質的な問題ではないとしている．Dean らは, MapReduce はデータ管理層非依存の処理フレームワークであり, Google 内においても, BigTable に対する検索結果を MapReduce の入力とする方法や, 日付毎にログファイルを管理し, 解析に必要なファイルのみを読み取る方法などが実際に行われていると述べている．また, アドホックな問い合わせ以外では, データフォーマットとして Google 独自のバイナリフォーマット (ProtocolBuffer) を用いていることも述べている．Jiang ら [11] らは Hadoop に対して, I/O 方式, 索引, パース処理, ソート処理における 4 つの問題点を指摘し, それらの改善策を提案している．それらを改善した Hadoop は文献 [9] で使用された並列データベースに近い性能を引き出せることを実験により示している．

Hadoop のエンタープライズでの利用が進むにつれて, MapReduce 処理系を再実装する流れも出始めている．MapR

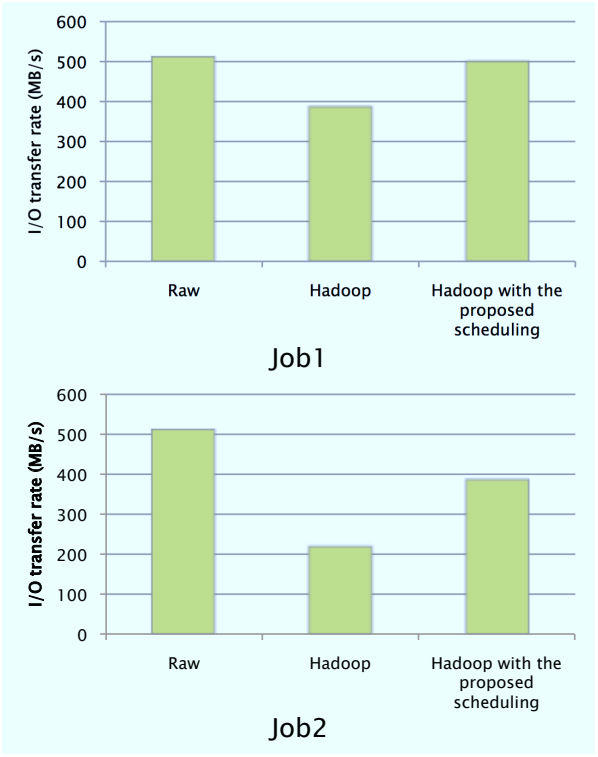

図 7　Mapper の I/O 処理内容に基づいたスケジューリング機構の効果

は, Hadoop の I/O 処理性能や NameNode の冗長性の低さの問題点を指摘し, Hadoop と互換な分散ファイルシステムや分散 NameNode を実装している．MapR のホワイトペーパー [8] によると, 各ノードに 12 台にディスクドライブを搭載した 10 ノードのクラスタ環境における実験では, 従来の Hadoop に対して約 3 倍のシーケンシャル I/O 性能が出ていることを示している．

並列データインテンシブ処理基盤における研究では, 並列データベースで培われた技術の応用による高速化技法や, 柔軟な処理モデルやプログラミングモデルに関する取り組みが多く見られる．Dittrich ら [12] は, 全データのスキャンを基本とした Hadoop に対して, クラスタ化索引の適用によるスキャン I/O 量の削減方法について議論している．また, Abouzeid ら [13] は, 各ノードのデータ管理層に単一の DMBS を配置し, Hadoop を DBMS 間のコミュニケーション機構として利用する方法を提案している．さらに積極的なパーティショニング方法 [14] の適用による, 結合処理におけるディスク I/O 並びにネットワーク I/O の削減方法について議論している．一方, MapReduce 処理モデルの低い柔軟性を指摘し, より柔軟な処理モデルやプログラミングモデルに関する研究として, Borkar ら [15] は, 予めシステムが提供するデータ処理オペレータ (ノード) とコネクタ (エッジ) による有向非巡回グラフ (DAG) によりジョブを表現する Hyracks というシステムを提案している．同様に Isard ら [16] も, 任意の DAG により柔軟なデータ処理プログラムを記述できる Dryad というシステムを提案している．また, Hive [17] や Pig [18] では, SQL に類似した高レベルな言語によ

り Hadoop ジョブを記述することで, MapReduce のプログラム開発の難易度を下げる取り組みが進められている.

5. おわりに

近年の並列データインテンシブ処理基盤による大規模データ解析においては, I/O 帯域を確保するために, 各ノードに多数のディスクドライブを搭載する流れが進んでいる. 本稿では, そのような背景を鑑み, TPC-H データセットを用いた Hadoop の I/O 性能を検証した. 実験の結果, 現状の Hadoop のアーキテクチャでは I/O 帯域および CPU リソースに空きがあるにも拘わらず性能が飽和してしまうという問題があることを確認した. また, 当該問題に対し, I/O 先読みを利用したスケジューリング機構の検証を行い, その有効性を明らかにした.

当該スケジューリング機構は, 現状に応じた解決策ではあるが, アドホックな機構に過ぎず, 豊富な I/O 帯域および CPU リソースを最大限に活用するにはデータ処理アーキテクチャの抜本的な改変が必要となると考えられる. 今後は, 当該課題の解決に向けて取り組んでいきたい.

文　献

[1] Digital Universe, http://www.emc.com/leadership/programs/digital-universe.htm , 2011.

[2] Oliver Ratzesberger “Agile Enterprise Analytics”, Proc. of USENIX FAST, 2010.

[3] Andres Carvallo, National Smart Grid Forum, 2010.

[4] Hadoop, http://hadoop.apache.org/ .

[5] Jeffrey Dean, Sanjay Ghemawat, “MapReduce: Simplified Data Processing on Large Clusters” Proc. of OSDI, pp. 137–150, 2004.

[6] Alex Loddengaard, “Cloudera’s Support Team Shares Some Basic Hardware Recommendations”, http://www.cloudera.com/blog/2010/03/clouderas-support-team-shares -some-basic-hardware-recommendations/, 2010.

[7] “Greenplum Data Computing Appliance”, http://www.greenplum.com/products/greenplum-hd .

[8] “The MapR Distribution for Apache Hadoop”, http://www.mapr.com/resources/maprforapachehadoopwp .

[9] Andrew Pavlo, Erik Paulson, Alexander Rasin, Daniel Abadi, David DeWitt, Samuel Madden, Michael Stonebraker, “A comparison of approaches to large-scale data analysis”, Proc. of SIGMOD, pp. 165–178, 2009.

[10] Jeffrey Dean, Sanjay Ghemawat, “MapReduce: a flexible data processing tool”, Communications of the ACM, 72–77, 2010.

[11] Dawei Jiang, Beng Chin Ooi Lei Shi, Sai Wu, “The performance of MapReduce: an in-depth study”, Proc. of VLDB, 472–483, 2010.

[12] Jens Dittrich, Jorge-Arnulfo Quiane-Ruiz, Alekh Jindal, Yagiz Kargin, Vinay Setty, and Jrg Schad “Hadoop++: Making a Yellow Elephant Run Like a Cheetah (Without It Even Noticing)”, Proc. of VLDB, pp. 515–529, 2010.

[13] Abouzeid, Azza and Bajda-Pawlikowski, Kamil and Abadi, Daniel and Silberschatz, Avi and Rasin, Alexander, “HadoopDB: an architectural hybrid of MapReduce and DBMS technologies for analytical workloads”, Proc. of VLDB, pp. 922–933, 2009.

[14] Bajda-Pawlikowski, Kamil and Abadi, Daniel J. and Silberschatz, Avi and Paulson, Erik, “Efficient processing of data warehousing queries in a split execution environment”, Proc. of SIGMOD, pp. 1165–1176, 2011.

[15] Borkar, Vinayak and Carey, Michael and Grover, Raman and Onose, Nicola and Vernica, Rares, “Hyracks: A flexible and extensible foundation for data-intensive computing”, Proc. of ICDE, pp. 1151–1162, 2011.

[16] Isard, Michael and Budiu, Mihai and Yu, Yuan and Birrell, Andrew and Fetterly, Dennis, “Dryad: distributed data-parallel programs from sequential building blocks”, Proc. of EuroSys, pp. 59–72, 2007.

[17] Ashish Thusoo, Joydeep Sen Sarma, Namit Jain, Zheng Shao, Prasad Chakka, Suresh Anthony, Hao Liu, Pete Wyckoff, and Raghotham Murthy, “Hive: a warehousing solution over a map-reduce framework” Proc. of VLDB, pp. 1626-1629, 2009.

[18] Christopher Olston, Benjamin Reed, Utkarsh Srivastava, Ravi Kumar, and Andrew Tomkins, “Pig latin: a not-so-foreign language for data processing” Proc. of SIGMOD, pp. 1099-1110, 2008.