Hadoop Cassandra と Cassandra を用いた並列分散処理機構の性能比較

菱沼　直子† 　竹房あつ子†† 　中田　秀基†† 　小口　正人†

† お茶の水女子大学　〒 112-8610 東京都文京区大塚 2-1-1
†† 産業技術総合研究所　〒 305-8568　茨城県つくば市梅園 1-1-1
E-mail: †naoko@ogl.is.ocha.ac.jp,oguchi@computer.org, ††{atsuko.takefusa,hide-nakada}@aist.go.jp

あらまし　クラウドコンピューティングの発展に伴い，大量に生成されるデータを蓄積し，高速に処理することが求められている．このような処理は従来の RDBMS では難しいことから，大量に生成されるデータの蓄積には分散 KVS が，高速な処理には MapReduce 処理系と分散ファイルシステムが用いられている．しかし，蓄積した大容量データを処理するためには分散 KVS から分散ファイルシステムにデータを転送しなければならず，そのコストが問題となる．Apache Cassandra には，Apache Hadoop 連携機能がある．この連携機能を使用すると転送コストの発生を防ぐことが予想されるが，その性能は明らかでない．我々は，既発表研究において，データを蓄積した分散 KVS 上で直接高速データ処理を行う手法を提案し，分散 KVS の Apache Cassandra を拡張した並列データ処理機構を実装した．本稿では，Hadoop Cassandra と本実装との比較を行い，これらの特性を明らかにする．比較実験より並列データ処理機構では，データサイズによらず高速に処理することができ，本提案手法の有効性が示された．

キーワード　分散並列処理，NoSQL，分散 KVS，Apache Cassandra，通信データ量

Performance Comparison of Hadoop Cassandra and Distributed Parallel Processing on Cassandra

Naoko HISHINUMA†, Atsuko TAKEFUSA††, Hidemoto NAKADA††, and Masato OGUCHI†

† Ochanomizu University　2-1-1 Otsuka, Bunkyouku Tokyo 112-8610 JAPAN
†† National Institute of Advanced Industrial Science and Technology (AIST)
1-1-1 Umezono, Tsukuba, Ibaraki,305-8568, JAPAN
E-mail: †naoko@ogl.is.ocha.ac.jp,oguchi@computer.org, ††{atsuko.takefusa,hide-nakada}@aist.go.jp

1. はじめに

クラウド技術の普及により，個人が生み出す情報が大量にネットワーク上に保存されるようになり，ネットワーク上に存在するデータが爆発的に増加している．このような大容量データは，厳密な一貫性が必ずしも必要ではないため結果一貫性を保証する．Apache Cassandra [1] [2] や Apache HBase [3] などの分散 KVS の枠組みが既に広く使われている．SNS などでサービスを提供するためには，保持しているデータを活用してユーザが必要とする情報を抽出する統計処理やマイニング処理などが不可欠となる．データ保持に使用される分散 KVS はそのような処理を念頭に置いて設計されておらず，MapReduce [4] のような処理系を利用することで，高速に処理することができる．現在ではこれらの処理に Apache Hadoop [5] を用いるのが一般的である．

しかし，Hadoop を用いる際には分散 KVS から処理を行う Hadoop Distributed File System(以下 HDFS) [6] 等の分散ファイルシステムへデータを転送するため，その転送コストが発生してしまう．この問題点に対し，Hadoop のデータストアに利用される HBase には coprocessor [7] という機能が実装されており，カウント，集約などの単純なプロセスをデータ保持しているサーバ上で実行している．しかし，Hadoop はバッチ処理向きのシステムであり，リアルタイムに発生するデータを格納しつつ，処理するには不向きである．大容量データを高速に蓄積可能な Cassandra には，Hadoop 連携機能 (以下 Hadoop Cassandra) が実装されている．この機能を用いれば転送コストを削減できると予想されるが，Hadoop Cassandra も MapReduce 処理を行うため，その性能は明らかではない．

我々は既発表研究 [8] において，リアルタイムに蓄積されたデータに対して高速データ処理を実行する手法を提案した．提案手法では，データアフィニティを考慮してデータを格納している Cassandra のデータノード上で直接データ処理を実行す

る．我々は，上記手法を，Cassandra の標準コマンドを機能拡張し，値を保存している各データノード上で処理をローカル実行し，処理結果のみをリクエストの答えとして返す並列データ処理機構を実装した．

本稿では，実装した並列データ処理機構と Hadoop Cassandra との性能比較を行い，これらの特性を調査する．比較実験より，複数の値に対してワードカウント処理を実行した際の実行時間は並列データ処理機構を用いた方が短く，本提案手法の有効性を示すことが出来た．

本稿は以下のように構成される．まず 2 章で関連研究について述べ，3 章で Cassandra の概要，4 章で既存研究で提案した手法の概要と実装した並列データ処理機構についての説明をする．5 章で本実装と Hadoop Cassandra との比較について述べ，6 章で本論文のまとめと今後の課題を述べる．

2. 関 連 研 究

本研究で用いた Cassandra は，書き込み性能を重視した分散 KVS である．Cassandra を用いた研究として [9] がある．この研究は，既存のクラウドストレージが読み出し性能と書き込み性能のどちらか一方を重視している点に着目し，同一クラウドストレージ内で読み出しと書き込み両方の性能を両立可能な MyCassandra Cluster を提案し，実装した．読み出し最適化と書き込み最適化のストレージエンジンを選択可能で，それぞれ適切な方へリクエストの振り分けを行うことで両者共に性能が出せるように工夫した仕組みを備えている．

ParaLite [10] は，SQLite をベースにした並列 RDBMS である．Collective Query と呼ばれる機能を用いて，SQL クエリの並列実行をサポートしている．また，ParaLite は UDX(User-Defined eXecutables) と呼ばれる，クライアントが発行する SQL クエリの中にシェルコマンドを埋め込むことができる機能を提供する．ユーザが実行したい処理をプラグインとして定義し，リクエストの答えとして処理結果のみを取得する点で本提案手法における User Defined Function(UDF) と類似している．UDF は，SQL 中でデータに適用可能な関数をユーザが独自にプラグインとして定義できる機能である．

Jobcast [11] は，KVS 上でデータ処理を可能にする並列分散処理フレームワークである．Cassandra と同様にリングトポロジを利用し，P2P により隣接ノードの情報を保持している．Jobcast も本実装と同様にユーザが定義した Java プログラムをジョブとして，各データノードに転送し，実行する実装になっている．我々は既存の分散 KVS を機能拡張しているのに対し，Jobcast はシンプルな KVS をベースにしている点が本研究と異なる．

SNS などをはじめとする大規模アプリケーションは，展開するサービスによってストレージに対して様々なニーズがある．現状の分散ストレージでは複数アプリケーションの異なるニーズを 1 つのシステム上で満たすことができない．そこで，様々なアプリケーションのニーズに対応する分散ストレージとして Comet [12] が研究されている．通常の分散 KVS を拡張し，アプリケーションはトラッキングやアクセスログやコンテキストへの適応など，ニーズに合った機能を分散ストレージに対して実行可能になる．UDF を用いて 1 つの分散 KVS 上でユーザが任意の処理を実行可能にする本実装と類似していると考えられる．

UDF 機能を用いて任意の処理を並列化する手法に関連する研究として Hive [13] や SQL/MapReduce [14] などがある．Hive は SQL-Like な宣言的言語 HiveQL をサポートしており，HiveQL のクエリは MapReduce のジョブにコンパイルされ Hadoop で実行される．これにより，ユーザは RDBMS 上のデータを SQL で操作するように Hadoop 上でデータ操作が可能になる．HiveQL は UDF も機能もサポートしているため，ユーザが任意の処理を並列化実行することができる．SQL/MapReduce は SQL の UDF を機能拡張し，MapReduce を用いることで並列型データ処理を実行可能にしている．これらの実装は本研究でユーザが任意の処理を並列実行するため，UDF として JAR ファイルを転送する機能を実装している点，今後の拡張方向として CQL を用いる点で類似している．しかし，Hive や SQL/MapReduce は並列化に MapReduce を用いているが，本実装は処理を複数のデータノード上へ転送することで並列化している点が異なる．

3. Apache Cassandra

Apache Cassandra は，Facebook 社が開発し，Apache プロジェクトとしてオープンソース化された分散データベース管理システムである [2]．Cassandra の主な特徴として，耐障害性，非中央集中型で単一故障点がないこと，一貫性の程度をユーザが自由に設定可能であることが挙げられる．必要とする一貫性のレベルをクライアントがクエリに記述することで，自由に設定することが出来るという点は，RDBMS はもちろん，他の NoSQL でもあまり見られない，Cassandra 固有の特徴と言える．

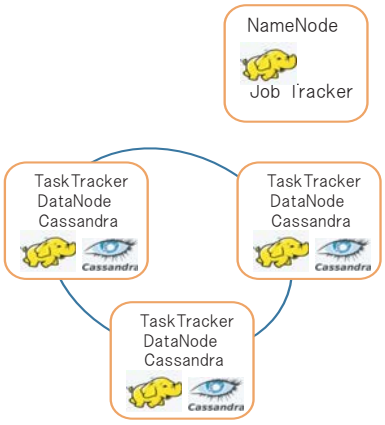

図 1　Hadoop Cassandra の概要

図 1 に Hadoop Cassandra の概要を示す．Hadoop Cassandra は Cassandra version0.6 から標準で実装されている機能であり，Cassandra のデータノードを Hadoop の TaskTracer として動作させるものである．通常の Hadoop はデータの入出力は HDFS に対して行うが，Hadoop Cassandra ではストレージ

として HDFS の代わりに Cassandra を用いており，Cassandra に蓄積されたデータに対して MapReduce を用いて処理する．ストレージに Cassandra を用いることで，耐障害性を高くすることができる．この実装を用いれば転送コストを削減できることが予想されるが，MapReduce のプログラミングモデルの制約があることや，小さなデータの扱いには不向きであるという課題がある．Hadoop Cassandra を用いた場合の処理の例を図 2

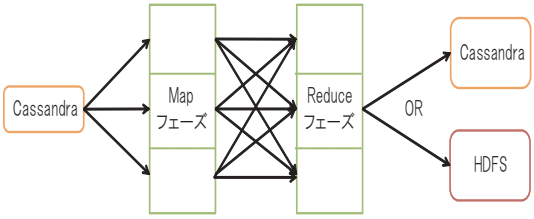

図 2　Hadoop Cassandra の処理過程

に示す．Hadoop Cassandra では値の入力を Cassandra から行い，通常の MapReduce と同様に Map フェーズ Reduce フェーズを実行し，処理結果を Cassandra もしくは HDFS 上に書き出すことが可能である．Cassandra の get range slice コマンド (次章で述べる) を利用して図 3 のように Cassandra のリングを分割し，レンジ 1 のデータは nodeA が担当し，2 は nodeB，3 は nodeC,4 は nodeD が担当する．この時，Cassandra の実装上，レンジ 1 のデータは nodeA が元々保持しているため，データの転送が発生せず各データを保存しているノード上でローカルに Map 処理を実行する仕組みとなっている．

4.　既発表研究

4.1　提案手法の概要

Cassandra に蓄積された大容量データに対して統計処理やマイニング処理等を行う場合は，Cassandra から処理の対象となる値を取得した後，分散ファイルシステムや MapReduce 処理系を用いて処理を行う流れになる．しかし，Cassandra の読み出し処理性能はあまり高くない上に，対象とする値のデータサイズが大きいと Cassandra から分散ファイルシステム等へ転送する際に大きな転送遅延が生じてしまう．そこで我々は [8] において，Cassandra に蓄積された大容量データに対して，データアフィニティを考慮して高速データ処理を可能にするための手法を提案した．

ここでいうデータアフィニティとは，各値が保存されているデータノード上で処理を実行する事を意味している．

提案手法では，UDF(User Defined Function) でユーザが実行したい処理をプラグインとして定義可能にする．定義された処理を，処理対象データを保持している各データノード上でローカルに実行し，処理結果のみをクライアントに返すことで転送コストを削減し，高速データ処理を実現する．

提案手法の概要を図 4 に示す．

(1)　UDF として，任意の処理 X をプラグインとして定義する．

(2)　クライアントからデータを格納している各データノードへリクエストを送信する．

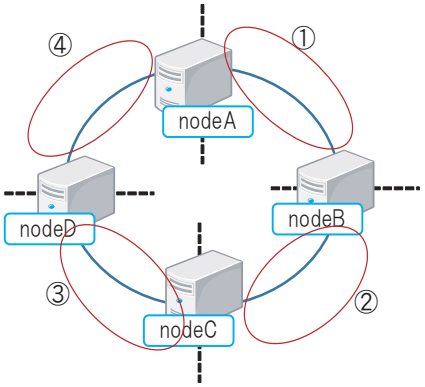

図 3　Hadoop Cassandra でのデータ分割イメージ

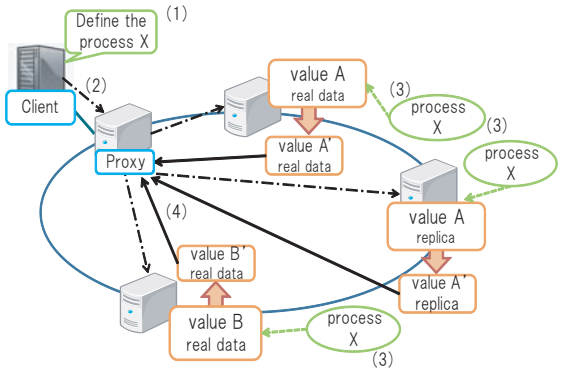

図 4　提案手法の概要

(3)　各データノード上にある，異なる値 A,B に対して定義された処理 X を並列実行し，処理結果を新たな値 A',B' とする．各値のレプリカに対しても同様の処理を行う．

(4)　処理結果である値 A',B' をリクエストの答えとして返す．

図 4 では値 B のレプリカの処理は省略している．

4.2　並列データ処理機構の概要

前節で説明した手法を実現させるために実装した並列データ処理機構について説明する．Cassandra の標準読み出しコマンド `get` と `multiget slice` を拡張し，読み出し処理のリクエストを送信すると，Cassandra に保存された複数の異なる値に対して，実行時に指定した処理を各データノード上で実行し，処理結果のみをリクエストの答えとして返す機能を実装した．本実装では UDF を Java で定義し，読み出しリクエストと共に定義した処理の JAR ファイルをデータノード上へ転送することでユーザが指定した任意の処理を実行可能にしている．Cassandra の主な標準読み出しコマンドの内容は表 1 に示す．

表 1　Cassandra の主な標準読み出しコマンド

Command	Function
`get`	指定した一つの値を取得
`multiget slice`	RowKey を指定して対応するそれぞれの Row を取得
`get range slices`	RowKey の range を指定し複数の Row を取得

本実装の概要を図 4 を用いて説明する．

(1)　実行したい処理 A の JAR ファイルを作成する．

(2)　読み出しリクエスト共に実行したい処理 X の JAR ファイルを転送する．JAR ファイルを転送しない場合は通常の読み出しが実行される．

(3)　各データノード上にある，異なる値 A,B に対して転送した処理 X を並列実行し，処理結果を新たな値 A',B' とする．

(4)　処理結果である値 A',B' をリクエストの答えとして返す．

各値のレプリカに対しても実行時に指定した処理を実行し，リクエストの答えをハッシュ値 (Digest) で返して整合性が保たれていない場合はバックグラウンドで同期処理を実行する．

```
/* public abstract class UDF {
     public abstract ByteBuffer
       processEach(ByteBuffer val)
               throws  Exception;
   } */
public class UDFImpl extends UDF {
  @Override
  public ByteBuffer
       processEach(ByteBuffer value)
  throws Exception {
    // write User defined
    //           function body here.
    return result;
  }
}
```

図 5　UDF の一例

```
String udf =
"{\"classname\": \"<class name>\", ";
udf += "\"jar\":\"/<path>/my_udf.jar\", ";
udf += "\"option1\":\"<option1 value>\" }";

Map<ByteBuffer, List<ColumnOrSuperColumn>>
results =
  client.multiget_slice_udf(
<row keys>, <column parent>,
<slice predicate>,<ConsistencyLevel>, udf);
```

図 6　リクエストプログラムの例

並列データ処理機構の Java の API は以下のようになっている．図 5 に UDF の定義例を示す．並列データ処理機構では，`org.apache.cassandra.db.SliceByNamesReadCommand` と `org.apache.cassandra.db.SliceFromReadCommand` の `getRow()` メソッドを拡張し，各値に対して任意の処理ができるように拡張した．`getRow()` メソッドは，表 1 のコマンドを呼びだされた際に Cassandra のデータノード側で値を取得するために呼び出されるメソッドである．ユーザは，abstract class である `UDF` を継承したクラス `UDFImpl` を作成し，そのクラスの `processEach()` メソッド内に任意の処理を実装する．また，`UDFImpl.class` を含む JAR ファイルを事前に作成しておく．

図 6 は，UDF を実行する際のプログラム例を示している．文字列 `udf` には，JSON 形式でクラス名，JAR ファイル名を指定しておく．また，ユーザの定義した UDF に必要な任意の実行オプションも同様に JSON 形式で指定することができる．この `udf` を，`multiget_slice` を拡張した `multiget_slice_udf` 関数に渡すことで，コマンドリクエストとともに指定した JAR ファイルが Cassandra のデータノードへ送信され，データノード上で指定した処理を実行することができる．

5.　Hadoop Cassandra との比較

Hadoop Cassandra と前章で述べた並列データ処理機構の性能比較を行う．

5.1　実験環境

ノード数が最大 8 台からなるクラスタに，提案手法による機能拡張を行った Cassandra をインストールした．今回の開発では，Cassandra バージョン 1.2.0 を用いた．測定に用いたノードの性能を表 2 に示す．

表 2　マシンスペック

OS	Linux 2.6.32-5-amd64
	Debian GNU/Linux 6.0.4
CPU	Intel(R) Xeon(R) CPU @ 2.66GHz x4
	Intel(R) Xeon(R) CPU @ 3.10GHz x4
Memory	8GB
HDD	500GB 7200RPM SAS Disk x 2
RAID Controller	SAS-6IR (RAID 0)
Network	1Gbps

5.2　実験概要・評価

Hadoop Cassandra と並列データ処理機構の比較を行う前に，Hadoop Cassandra の基本性能を調査した．測定に用いたパラメータ設定を表 3 に示す．図 7 にデータノード数を 3 台，5 台，8 台と変化させた場合の実行時間の変化を示す．グラフより，データノード数が 3 台，5 台の場合よりも 8 台の方が実行時間が短いことが分かる．この結果より，Hadooop Cassandra も並列データ処理機構同様に台数増加に伴って性能向上するスケールアウトする性質があるといえる．

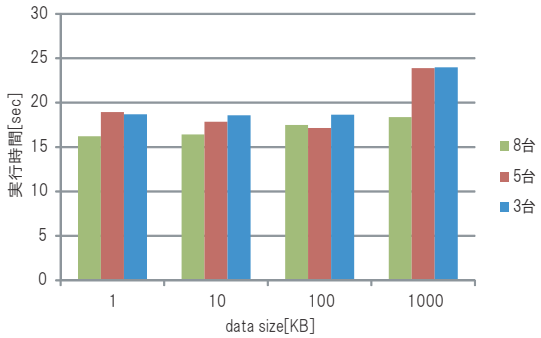

図 7　ノード数・データサイズを変化させた場合の実行時間

次に，Cassandra 上に保存されている値 (テキストデー

タ)1000 個に対して，実装した並列データ処理機構と Hadoop Cassandra を用いて `word count` を 1000 回実行するのにかかる時間を測定する．一つのテキストデータサイズを 1KB〜1MB まで変化させた場合の実行時間の変化を調査する．

表 3　測定に用いたパラメータ設定

データノード数	3,5,8
処理を行う値の数	1000
一つの値の大きさ	1KB,10KB,100KB,1MB

図 8，9 にデータノード数を 3 台，8 台とし，一つの値の大きさを 1Kbyte〜1Mbyte まで変化させた場合のそれぞれの実行時間を示す．縦軸が実行時間 (sec)，横軸は値の大きさ (KB) を表す．図 8，9 より，値の大きさ，ノード数に関わらず並列データ

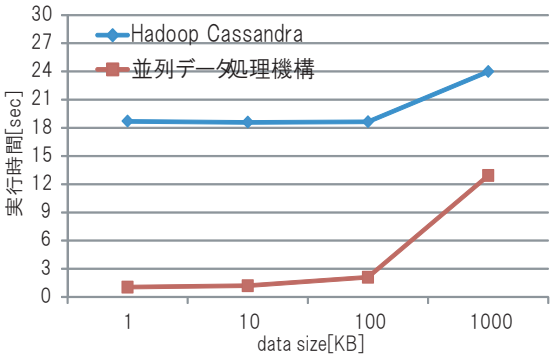

図 8　ノード数が 3 台の場合の実行時間

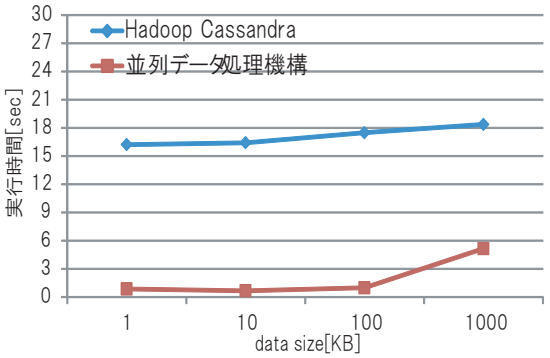

図 9　ノード数が 8 台の場合の実行時間

処理機構を用いた場合の方が高速に処理していることが分かる．これは，今回の実験ではデータサイズが比較的小さい値で行ったため，Hadoop Cassandra を用いた場合，MapReduce を行う際の入出力にかかる時間がオーバーヘッドとなったためと考えられる．データサイズが 1MB の時は Hadoop Cassandra と並列データ処理機構との実行時間の差は 12 秒程度になっていた．

以上の結果より，並列データ処理機構はデータサイズによらず高速に処理することができ，本提案手法の有効性が示された．

6. おわりに

我々は既発表研究において，大容量データを高速処理する際に発生するコストを抑えつつリアルタイム処理を可能にするため，分散 KVS の実装の 1 つである Apache Cassandra を拡張してデータアフィニティを考慮した並列データ処理機構を実装した．この実装は Cassandra に保存された複数の異なる値に対して，値を保存している各データノード上で事前に指定した処理をローカル実行し，処理結果のみをリクエストの答えとして返す仕組みとなっている．これに対し Hadoop のストレージとして Cassandra を用いた Hadoop Cassandra という機能があり，本稿では並列データ処理機構と Hadoop Cassandra の性能比較を行い，これらの特性を調査した．比較結果より，並列データ処理機構は値の大きさ，データノード数に関わらず，高い性能を出すことが出来た．

今後の課題としては，word count 以外の処理での比較や，より大規模な環境での比較を行うことがあげられる．実装の課題としては，sum や average などの集約演算処理を実行可能にするため，集約演算処理のフェーズを追加することがあげられる．本実装でこのような機能を追加する場合には CQL(Cassandra Query Language) と呼ばれる SQL ライクな言語を利用して集約演算する手法が考えられる．

文　献

[1] A.Lakshman and P.Malik. ”Cassandra - A Decentralized Structured Storage System,” ACM SIGOPS Operating Systems Review, vol.44, No2, PP.35-40,April 2010.

[2] Eben Hewitt.,Cassandra: The definitive guide, trans. Shinpei Ohtani and Takashi kobayashi. O’Reilly JAPAN, 2011.

[3] The Apache software Foundation. Apache HBase. http://hbase.apache.org/hbase/.

[4] J.Dean and S.Ghemawat. ”MapReduce: Simplified Data Processing on Large Clusters” In Proceedings of the 6th conference on Symposium on Opearting Systems Design & Implementation - Volume 6 (OSDI’04), Vol. 6. USENIX Association, Berkeley, CA, USA, 10-10.

[5] Tom White.,Hadoop: The definitive guide, trans. Ryuji Tamagawa. O’Reilly JAPAN, 2010.

[6] Dhruba Borthakur. ”HDFS Architecture,” 2008 The Apache Software Foundation.

[7] The Apache software Foundation. Apache HBase/coprocessor. https://blogs.apache.org/hbase/entry /coprocessor_introduction.

[8] N.Hishinuma, A.Takefusa, H.Nakada, M.Oguchi. ”Implementation of Data Affinity-based Distributed Parallel Processing on a Distributed Key Value Store” In Proc. ACM IMCOM2014, Siem Reap, Cambodia, January 2014.

[9] S.Nakamura, K.Shudo. ”A Cloud Storage Supporting Bath Read Heavy and Write Heavy Workloads” Proceedings of the 5th Annual International Systems and Storage Conference. ACM, 2012. p. 5.

[10] Ting Chen, Kenjiro Taura. ”ParaLite: Supporting Collective Queries in Database System to Parallelize User-Defined Executable” In Proceedings of the 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid12), Ottawa, 2012:474-481

[11] NAKAGAWA Ikuo, NAGAMI Kenichi. ”Jobcast-Parallel and Distributed Processing Framework: Data Processing on a Cloud Style KVS Database” In Proc. IEEE/IPSJ 12th International Symposium on Applications and the Internet

(SAINT2012),pp.123-128 IEEE, 2012.
[12] R.Geambasu, A.A.Levy, T.Kohno, A.Krishnamurthy and H.M.Levy. "Comet: An active distributed key-value store" In Proc. OSDI. 2010, October. p.323-336.
[13] A.Thusoo, J.S.Sarma, N.Jain, Z.Shao, P.Chakka, S.Anthony, H.Liu, P.Wyckoff and R.Murthy. "Hive: a warehousing solution over a map-reduce framework" In Proc. VLDB Endow. 2, 2 (August 2009), 1626-1629.
[14] E.Friedman, P.Pawlowski, and J.Cieslewicz. "SQL/MapReduce: a practical approach to self-describing, polymorphic, and parallelizable user-defined functions" In Proc. VLDB Endow. 2, 2 (August 2009), 1402-1413.