社団法人　電子情報通信学会
THE INSTITUTE OF ELECTRONICS,
INFORMATION AND COMMUNICATION ENGINEERS

信学技報
TECHNICAL REPORT OF IEICE.

Cassandraによる局所性を考慮した分散並列処理の提案

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-h@osgl.is.ocha.ac.jp,oguchi@computer.org, ††{atsuko.takefusa,hide-nakada}@aist.go.jp

あらまし　近年，大量のデータを高速に処理することが必要な場面が増え，分散 KVS (Key Value Store) と呼ばれる NoSQL 型データベース管理システムが注目され始めた．分散 KVS に格納された大容量データを効率よく活用するには，データの並列処理が必要となるが，分散 KVS から対象データを取り出した後，再度データを分散させて並列処理を行うと，処理効率が悪くなってしまう．そこで本研究では，大規模データを扱う分散 KVS である Apache Cassandra に着目し，大規模データをより高速に処理するための手法を提案する．Cassandra に保存された値に対して任意の処理を行うには，値を取得し，その後処理を行うのが通常である．しかし，Cassandra の読み出し性能があまり高くない上に，取得する値のデータ量が大きくなると通信量が多くなり処理が遅くなることが予想される．そこで本研究では，Cassandra に保存された値に対し任意の処理を効率よく行えるようにするために，まず，UDF と類似した機能を Cassandra に追加する．この機能を利用し，各データノード上でユーザが指定した処理を行い結果のみをクライアントに返す手法を提案する．これにより通信データ量を抑えることができ，また，異なる複数の値に対して並列に処理を実行可能になり，より高速化できる．本稿では提案手法の実装の第一段階として，1 つの値に対し，任意の処理を行い結果のみを取得する機能を実装し，その特性を評価した．その結果，本提案手法は処理対象の値のサイズが比較的大きい場合には有効であることが示せたと同時に，その一貫性レベルを調整することで処理の高速化が可能であることを確認した．

キーワード　NoSQL, 分散 KVS, Apache Cassandra,UDF

Toward Data Locality-aware distributed parallel processing on Cassandra

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

† Ochanomizu University　Otuka 2-1-1,Bunkyo-Ku,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-h@osgl.is.ocha.ac.jp,oguchi@computer.org, ††{atsuko.takefusa,hide-nakada}@aist.go.jp

Abstract　The explosive growth of data causes Big Data processing by using distributed KVS (Key Value Store), known as a NoSQL database management system. In order to effectively utilize large amount of data, stored in such a Distributed KVS, it is required for parallel data processing. However, it is difficult to achive high performance if data are retrieved from the distributed KVS and then distributed and processed over multiple computers. In this study, we focus on Cassandra, one of distributed KVS implementations. In general, a client node retrieves required data from Cassandra, and then the data are processed. But, read overheads of Cassandra is higher than its write overheads, and the processing time becomes longer with increasing amount of data. In order to achive efficient data processing, we propose data locality-aware distributedparallel processing on Cassandra. We extend Cassandra to allow users to define a function executed on Cassandra data nodes. On the related data node, the requested data are processed by using each user defined function, and then the results return to the requested user. As a result, the amount of transfered data can be reduced. In this paper, We have implemented the proposed method and we evaluateits performance characteristics. The results showed that the proposed method is beneficial especially for large data set and further performance could be achieved by adjusting consistency levels.

Key words　NoSQL, distributed KVS, Apache Cassandra,UDF

1. はじめに

クラウド技術の普及により，映像共有システムやソーシャルネットワークサービス等，大量の利用者が情報を共有しつつ利用できるような大規模アプリケーションが多く開発されている．これに伴い，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量が爆発的に増加している．その結果大量のデータを高速に処理することが必要となり，従来のデータベース管理システムであるRDBMS ではデータの格納や処理の柔軟性に不満が出るようになり，分散環境で一貫性の制限を緩和して処理性能を重視する分散 KVS(Key Value Store) と呼ばれる NoSQL 型データベース管理システムが注目され始めた．分散 KVS では，RDBMS では管理が困難な規模の大容量データを分散環境で管理することができる．一般に，データの複製を生成，管理する機能も提供されており，可用性が高いことが知られている．管理されている大容量データを効率よく活用するためには，対象となるデータに対して複数計算機を利用して並列処理を行う必要がある．しかしながら，分散 KVS からデータを取り出した後に再度データを複数計算機に分散させて並列処理を行うと，データの転送オーバヘッドが非常に大きくなってしまう．

そこで本研究では分散 KVS の実装のひとつである Apache Cassandra [1] [2](以下 Cassandra) に着目し，大規模データをより高速に処理するための手法を提案する．Cassandra に保存されたデータに対して任意の処理を行う場合では，Cassandra から処理の対象となる値を取得し，その後処理を行うのが通常の流れである．しかし，Cassandra の読み出し処理性能はあまり高くない [3] 上に，取得する値のデータ量が大きくなると通信データ量が多くなり処理が遅くなってしまうことが考えられる．そこで本提案手法では，Cassandra 上の値に対し任意の処理を効率よく行えるようにするために，UDF(User Defind Function) と呼ばれる 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と類似した機能を Cassandra に追加する．この機能により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なる．その後，定義された処理を各データノード上で実行し処理結果のみをクライアントに返す．これにより通信データ量を抑えることができ，また，処理対象の値を複数指定すると，異なる値に対して並列処理が可能になり，より高速に処理が行えると考えられる．

本稿ではその第一段階として，ユーザが指定した 1 つの値に対し任意の処理を行い，その結果のみを取得する機能を実装し，その性能特性を評価した．その結果，処理を行う値のデータサイズが大きくなると，本提案手法では実行時間の増加が見られたが，通常の Cassandra の読み出し手法を利用した際よりも増加の割合が低かった．よって本提案手法は処理対象の値のサイズが比較的大きい場合には有効な手法であることが示せたと同時に，一貫性のレベルを調整することで処理の高速化が可能であることを確認した．

2. Apache Cassandra

2.1 Apache Cassandra の概要

本研究では，KVS 型データベースである Apache Cassandra に着目した．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 [4]．Cassandra の特徴としては，カラム型データ構造を持つリッチデータモデルである点が挙げられる．KVS 型のデータベースは通常 1 つのキーで 1 つのバリューを管理しているが，Cassandra はキースペース，カラムファミリ，キー，カラムの 4 つ，もしくはスーパーカラムを加えた 5 つのキーで 1 つのバリューを管理している．これにより通常の分散 KVS よりも複雑なデータ管理が可能となっている．Cassandra のデータモデルを図 1 に示す．

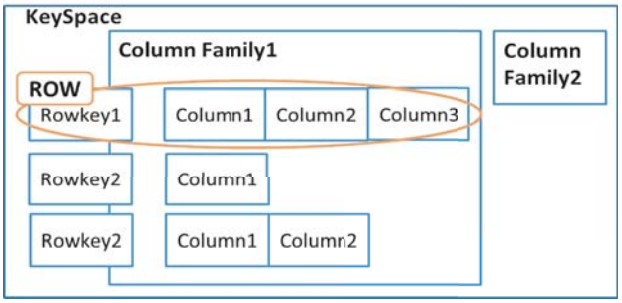

図 1　Cassandra のデータモデル

その他の主な特徴としては，耐障害性の高さ，非中央集中型で単一故障点がない，データの分散保持を考慮した分散特性や一貫性の程度 (ConsistencyLevel) をユーザが自由に設定可能といった事が挙げられる．また Cassadnra は書き込み性能を重視した分散 KVS として開発されているため，書き込みの際にはディスクへのランダムアクセスが発生せず処理が高速になり，読み出しの際にはディスクへのランダムアクセスが発生し処理が低速になるという実装になっている．

2.2 書き込み/読み出し処理の流れ

2.2.1 Cassandra の構成と処理の流れ

Cassandra は，クライアントノード，プロキシノード，データノードからなる．

・クライアントノード　任意のプロキシノードに接続し，リクエストを送信．その後処理された結果を受け取る．

・プロキシノード　キーを元にデータノードのリストを作成し，全データノードにリクエスト内容を記したメッセージを送信．その後，ConsistencyLevel に応じたレスポンスを待ち，クライアントノードに結果を返す．プロキシノードはクライアントが最初に接続したデータノードが担う．

・データノード　リクエストメッセージの内容をみて，それに応じた処理を行い，処理の結果をプロキシノードに返す．

2.2.2 書き込みの流れ

書き込みはいつでも可能で，シーケンシャルに行われるため，ランダム I/O が発生せず高速に行える実装となっている．書き込み処理の主な流れを図 2 に示す．

(1)　書き込みリクエストがプロキシノードに送信され，担

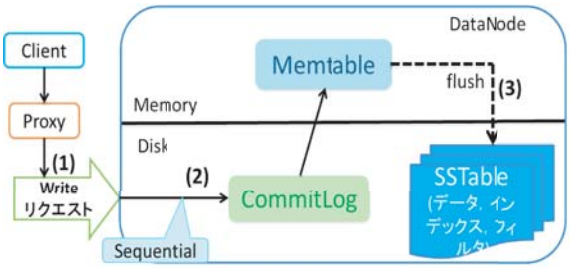

図 2　書き込み処理の流れ

当するデータノードにリクエストが送信される．

(2)　リクエストを受け取ったデータノードはディスク上にある CommitLog にシーケンシャルに書き込みを行い，同時にメモリ上の Memtable に書き込む．

(3)　Memtable 上のデータ量が閾値を超える，もしくは一定時間が経過したらディスク上の SSTable ファイルにフラッシュされる．

書き込みは CommitLog に書き込みが行われた段階で処理完了とみなす．Memtable 上のデータはロウキーとカラムファミリの Map として保存され，SSTable にはインデックス，ロウデータ，ブルームフィルターが保存される．一旦フラッシュが実行されると，SSTable 上のファイルは変更不可能になり，それ以上の書き込みは受け付ず，コンパクション処理で複数の古い SSTable ファイルを一つの新しいファイルにマージする処理のみ実行可能となる．

2.2.3　読み出しの流れ

読み出しはメモリ上のデータに加え，ディスク上のデータも読む必要があり，ランダム I/O が発生する実装となっている．そのため書き込みに比べると処理が低速になる傾向がある．読み出し処理の主な流れを図 3 に示す．

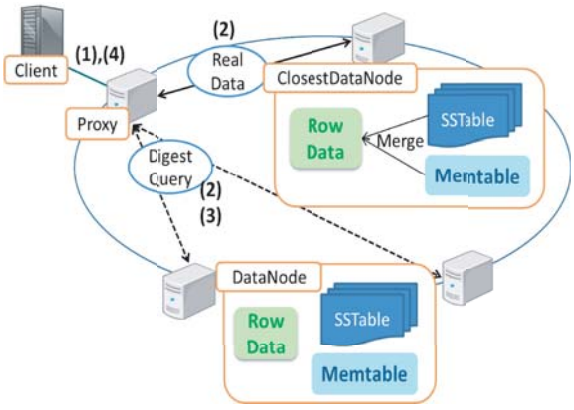

図 3　読み出し処理の流れ

(1)　読み出しリクエストがクライアントから任意のプロキシノードに送信される．

(2)　リクエストを受け取ったプロキシノードはClosest データノードにのみ実データの問い合わせを行い, その他のデータノードに対しては Digest(データのハッシュ値のみを求める) 問い合わせを実行する．

(3)　実データの問い合わせを受け取った Closest データノードは SSTable と Memtable をマージし，問い合わせに一致するロウデータを探索し，プロキシノードに返す．

(4)　プロキシノードは ConsistencyLevel に応じたレスポンスを待ち，その後クライアントへと結果を送信する．

プロキシノードは受け取った実データと Digest を比較し，一致していなかった場合には非同期にバックグラウンドでリードリペア処理を実行する．コンパクションを行うことで，読み込みが必要な SSTable ファイル数を制限し，また使用されていないデータによって埋められているスペースを取り戻すことが可能になっている．

3.　提案手法/設計

本研究の提案手法を説明する．Cassandra に保存されたデータに対して任意の処理を行う場合では，Cassandra から処理の対象となる値を取得し，その後任意の処理を行うのが通常である．しかし，Cassandra の読み出し処理性能はあまり高くなくない上に，取得する値が大きいと通信データ量が多くなり処理が遅くなってしまうことが考えらる．

そこで本提案手法では，Cassandra 上の値に対し任意の処理を効率よく行えるようにするために，まず UDF(User Defind Function) と呼ばれる 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と類似した機能を Cassandra に追加する．この機能により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なる．その後，定義された処理を各データノード上で実行し処理結果のみをクライアントに返す．これにより通信データ量を抑えることができ，また，処理対象の値を複数指定すると異なる値に対して並列処理が可能になり，より高速に処理が行えると考えられる．

本提案手法の概要を図 4 に示す．

(1)　クライアントからプロキシノードを経由し，各データノードへリクエストが送信．

(2)　各データノード上にある，異なる値 (valueA,B) に対してユーザが指定した同一の処理を行い，その結果を新たな値 (valueA',B') として保存．

(3)　処理を行った結果を格納した値 (valueA',B') をリクエストの答えとして返す．

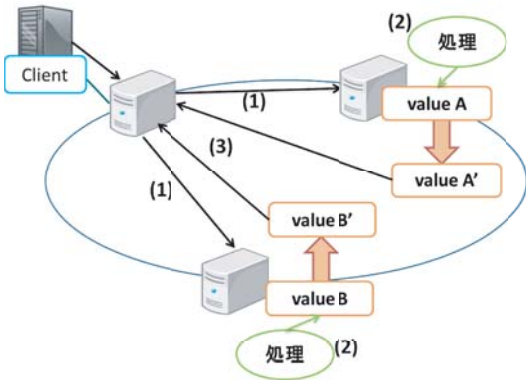

図 4　提案手法の概要

本稿ではその第一段階として，ユーザが指定した一つの値に対して，その値が保存されている各データノード上でユーザが指定した処理を実行し，処理結果をカラムの新たな値としてクライアントに返す機能を実装した．具体的な実装内容としては，

Cassandra の標準参照コマンド get をベースに，ある値に対し，その値を保持しているデータノード上で指定した処理を行いその結果のみを参照する新たなコマンドを作成した．今回の実装で追加したコマンドでは，引数をパス名を指定することにより，各実行時に指定された任意の Linux コマンドを実行することが可能である．追加コマンドを実行した場合，実行時に指定された処理が値を保存してる各データノード上で実行され，処理結果のみをリクエストのレスポンスとして返す．

4. 評　　価

本研究では，前章で説明した実装を用いて，各データノード上で処理を行い処理結果のみを取得する場合と，Cassandra から処理対象の値を取得してきてから処理を行う場合での性能比較を行う．

4.1 実 験 環 境

ノード数 8 台からなるクラスタに，自作コマンドを実装した Cassandra をインストールした．今回の開発では，Cassandra バージョン 1.1.0 を用いた．測定に用いたノードの性能を表 1 に示す．

表 1　machine spec

OS	Linux 2.6.32-5-amd64 Debian GNU/Linux 6.0.4
CPU	Quad-Core Intel(R) Xeon(R) CPU @ 2.66GHz Quad-Core Intel(R) Xeon(R) CPU @ 3.10GHz
Memory	8GByte

4.2 測 定 概 要

Cassandra によるクラスタを構築し，get コマンドを使用し値を取得してからワードカウントコマンド (wc) を実行した際 (以下クライアント側処理) と，各データノード上で wc を実行してその結果のみをクライアントに返す自作のコマンド使用した際 (以下サーバ側処理) のデータ量の増加に伴う実行時間の変化を比較した．それぞれの処理の流れを図 5,6 に示す．

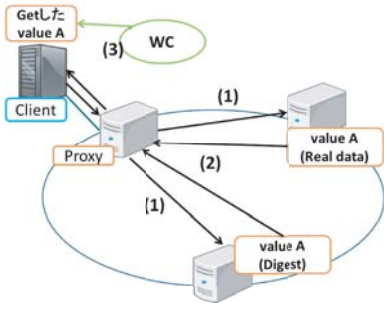

図 5　クライアント側処理の流れ

クライアント側処理の流れ

(1)　読み出しリクエストがクライアントから送信され，プロキシノードが担当するデータノードに転送．

(2)　担当するデータノードはリクエストに対するレスポンスとして値 A をプロキシノードに返し，プロキシノードはその値をクライアントに返す．

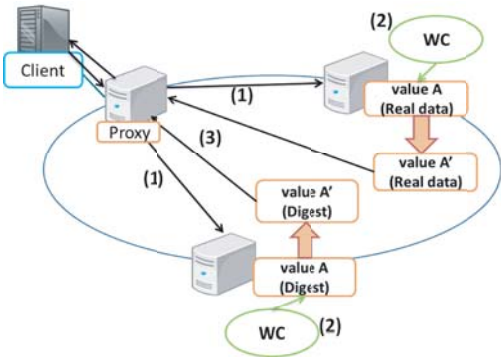

図 6　サーバ側処理の流れ

(3)　読みだした値 A に対して wc を実行する．

サーバ側処理の流れ

(1)　読み出しリクエストがクライアントから送信され，プロキシノードが担当するデータノードに転送．

(2)　担当するデータノードはリクエストに適する値 A に対して wc を実行し，処理結果を新たな値 A' とする．

(3)　データノードは値 A' をプロキシノードに返し，プロキシノードはその値をクライアントに返す．

クライアント側処理，サーバ側処理どちらも (1)〜(3) を一つの処理とし，Cassandra のレプリケーション数は 3 で固定 (デフォルトは 1[オリジナルデータのみ])，一貫性レベルを ONE, ALL(デフォルトは ONE)，データサイズを 10〜100MB と変化させて 1 つの処理が完了するまでの実行時間の測定を行った．

4.3 測定結果/評価

図 7 に一貫性レベルを ONE に指定し，クライアント側処理とサーバ側処理をデータ量を増加させて実行した際の実行時間の変化を示し，図 8 に一貫性レベルを ALL に指定し，クライアント側処理とサーバ側処理をデータ量を増加させて実行した際の実行時間の変化を示す．図 9 にはサーバ側処理を一貫性レベルとデータ量を増加させて実行した際の実行時間の変化を示す．図 7〜9 は全て縦軸が実行時間 (sec) で，横軸がデータ量 (MB) となっている．

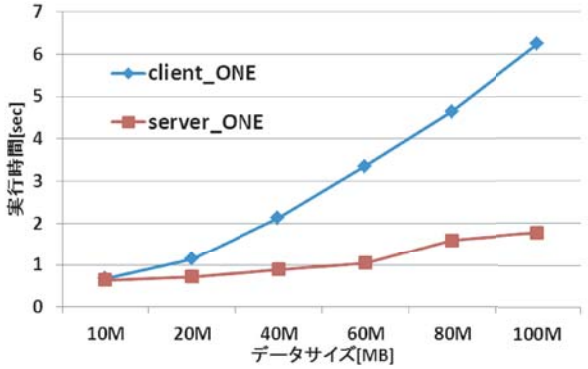

図 7　一貫性レベルを ONE にした際の実行時間の変化

図 7，8 より，一貫性レベルが ONE の場合も ALL の場合もクライアント側処理，サーバ側処理ともに実行時間が増加してる傾向が見られるが，クライアント側処理の方が実行時間の増加の割合が大きいという結果になったことがわかる．つまり，一貫性のレベルにかかわらず処理を行いたい値のサイズが大き

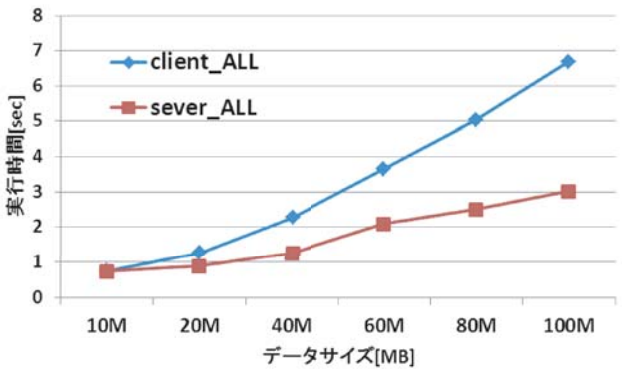

図 8　一貫性レベルを ALL にした際の実行時間の変化

くなると，Cassandra の通常の読み出し時間が長くなること確認できた．よって，処理を行いたい値のデータ量が大きい場面では，通信を必要とするデータのサイズを小さくする本提案手法は有効だということが示せた．また，データサイズがより大きい場合や，ネットワークに遅延が生じている環境においては，本提案手法がより有効になると考えられる．

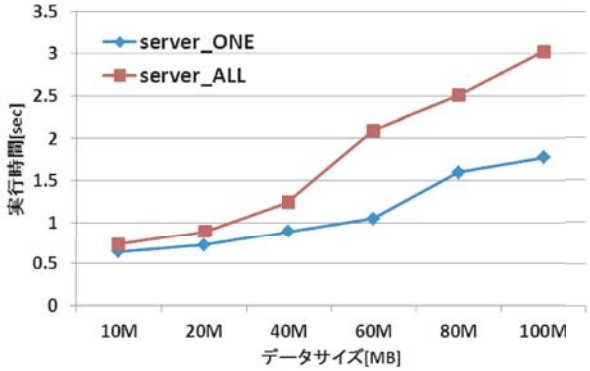

図 9　一貫性レベルを変化させた際のサーバ側処理の実行時間

図 9 のサーバ側処理における一貫性レベルによる比較では，サーバ側処理も一貫性レベルの影響は受けていることがわかる．これは wc を行った結果を一貫性レベルで指定した分だけレスポンスを待ってから返答しているためであり，通常の Cassandra の振る舞い [5] と同様の傾向も見られた．Cassandra では，W(書き込みレプリカ数) + R(読み出しレプリカ数) > N(レプリケーション数) を満たすと強一貫性となることが知られており [4]，書き込み処理を ALL，読み出し処理を ONE として利用することで処理の高速化が可能となる．

5. 関 連 研 究

本研究に関連している研究として ParaLite [6], [7] があげられる．ParaLite は，SQLite をベースにする並列 RDBMS である．これらは，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おり，複数のクライアントにクエリをジョブ単位で分散させることで並列実行を可能にしている．各ジョブをそれぞれのデータノードで実行し，結果を取得し，それらの結果を一か所に集約する．本研究の提案手法も ParaLite と同様に，複数のデータノードで処理を並列に実行することが目的であるが，本提案手法はリクエストを分割するのではなく，同一のリクエストを複数のデータノードに送信し，異なる値に対して処理を実行する仕組みになっている．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によってデータをデータベースから取得し，データに対して任意の処理を実行し，その結果のみを得ることが可能になる．これは，本研究の提案手法で用いた関数をユーザが独自にプラグインとして定義しデータをデータベースから取得し，データに対して任意の処理を実行し，その結果のみを得る手法と類似している．

6. おわりに

現在注目されている分散 KVS の実装の 1 つである Apache Cassandra に着目し，データの局所性を考慮した分散処理の提案をした．本稿ではその第一段階として，ユーザが指定した一つの値に対して，その値が保存されている各データノード上でユーザ指定した処理を実行し，処理結果をカラムの新たな値としてクライアントに返すサーバ側処理を行う自作コマンドを実装した．

サーバ側処理の特性を評価したところ，一貫性のレベルにかかわらずデータサイズが大きくなるとサーバ側処理よりもクライアント処理の方が実行時間増加の割合が大きく，データサイズの影響を受けやすいことが確認できた．また，サーバ側処理は一貫性レベルが高くなるにつれて，処理に時間がかかる傾向があることが確認できた．よって本提案手法は処理対象の値のサイズが比較的大きい場合には有効であることが示せたと同時に，一貫性のレベルを調整することで処理の高速化が可能であることを確認した．データサイズがより大きい場合や，ネットワークに遅延が生じている状況では本提案手法がより有効になると考えられる．

今後の課題としては，分散している異なる値に並列に処理ができるように拡張することと，より大きなデータサイズの値を対象とした評価を行うことがあげられる．

文　献

[1] Avinash Lakshman, Prashant Malik, "Cassandra - A Decentralized Structured Storage System," The 3rd ACM SIGOPS International Workshop on Large Scale Distributed Systems and Middleware, October 2009.
[2] Apache Cassandra:http://cassandra.apache.org/
[3] 菱沼直子，竹房あつ子，中田秀基，小口正人，"Cassandra による KVS データ処理におけるデータ容量と処理性能に関する考察" DEIM Forum 2012，C2-5，2012 年 3 月．
[4] Eben Hewitt(著), 大谷晋平, 小林隆 (訳):Cassandra, オライリー・ジャパン,2011
[5] 菱沼直子，竹房あつ子，中田秀基，小口正人，"様々なシステム構成における Cassandra の処理能力に関する考察" マルチメディア，分散，協調とモバイル (DICOOMO2012) シンポジウム，2D-1，2012 年 7 月．
[6] Ting Chen，Kenjiro Taura，"Data-Intensive Text Processing Workflows with a Parallel Database System" IPSJ SIG Technical Report,Vol.2012-HPC-135NO.23，Augst 2012.
[7] 中谷翔，Ting Chen，田浦健次朗，"ワークフローアプリケーション基盤としての並列 DB の性能評価" 情報処理学会研究報告，Vol.2012-HPC-135NO.24，2012 年 8 月．