DEIM Forum2013 F2-3

Cassandraによるデータアフィニティを考慮した分散並列処理の提案と実装

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，大量のデータを高速に処理することが必要となっている．そこで，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 を用いると，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．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．本研究では分散 KVS の実装のひとつである Apache Cassandra に着目し，データアフィニティを考慮して大容量データをより高速に処理するための手法を提案する．本稿では，Cassandra 上に分散して保存された異なる値に対し，事前に指定された処理を値が格納されている計算機上で並列に実行できる機能を実装した．性能評価を行い，本提案手法は Cassandra を通常使用した際よりも処理が高速に行えることを示す．
キーワード　分散並列処理，NoSQL，分散 KVS，Apache Cassandra，通信データ量

Toward Data affinity-aware 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，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，極めて多数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関して不満が出るようになった．そこで，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 を用いると，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．分散 KVS に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．そのため，大容量データ処理では従来のように処理を行う場所に値を転送するのではなく，値が保存されている場所に処理を割り当てることが必要だと考えられる [1].

本研究では分散 KVS の実装のひとつである Apache Cassandra [2] [3](以下 Cassandra) に着目し，データアフィニティを考慮して大容量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，Cassandra から処理の対象となる値を取得し，その後

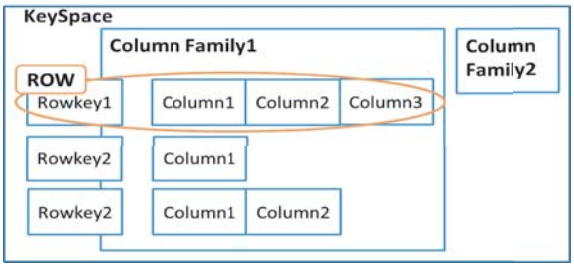

図 1　Cassandra のデータモデル

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 [4] 上に，対象とする値のデータ量が大きいと値を取得するための通信処理が遅くなってしまうことが考えられる．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 Defined Function) と類似した機能を Cassandra に追加する．この機能を用いてユーザが実行したい処理をプラグインとして定義し，定義された処理を各データノード上で実行して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る．また，処理対象の値を複数指定することで異なる値に対して複数のデータノード上で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．よって，本稿では Cassandra に保存された複数の異なる値に対し，事前に指定した処理を各データノード上で実行し，処理結果のみを返す機能を実装する．評価実験から，本提案手法は Cassandra を通常使用した際よりも通信データ量を削減させ，処理が高速に行ることを示す．

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 [5]．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．Cassandra のデータモデルを図 1 に示す．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．

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，リクエストを送信する．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する．その後，一貫性レベル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，追記ログの形式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．図 2 中の CommitLog，Memtable，SSTable はそれぞれ，ディスク上のすべての書き込みをログの形式で記録するファイル，メモリ上に常駐する構造体，ディスク上にある変更不可なファイルである．

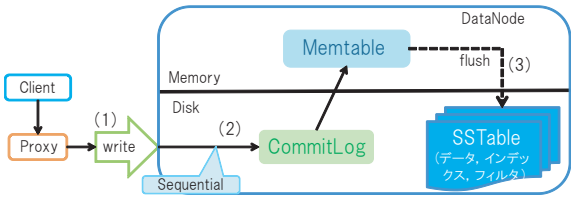

図 2　書き込み処理の流れ

(1)　書き込みリクエストがプロキシノードに送信され，担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 ファイルにロウデータがフラッシュされる．

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マップとして保存され，SSTable にはインデックス，ロウデータ，ブルームフィルタが保存される．一度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．

(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．

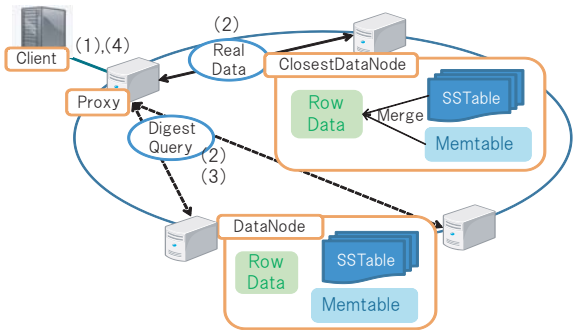

図 3　読み出し処理の流れ

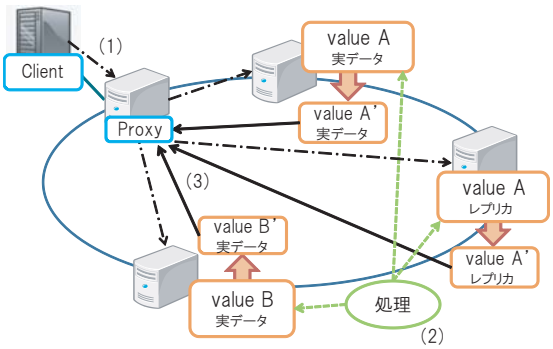

図 4　提案手法の概要

(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)　プロキシノードは一貫性レベル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

Cassandra に保存されたデータに対して任意の処理を行う場合，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，対象とする値のデータ量が大きいと値を取得するための通信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データアフィニティを考慮して効率よく行えるようにするために，まず UDF(User Defined Function) と類似した機能を Cassandra に追加する．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．この機能により Cassandra で管理するデータに対して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て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複数のデータノード上で並列処理が可能になって，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)　クライアントから各データノードへリクエストを送信する.

(2)　各データノード上の，異なる値 A,B に対してプラグインとして定義された処理を各値に対して並列実行し，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．図 4 では値 B のレプリカの表示は省略している.

(3)　処理結果である値 A',B' をリクエストの答えとして返す.

本稿では Cassandra に保存された複数の異なる値に対し，事前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．事前に指定される処理は，引数に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 コマンドを対象データを管理しているデータノード上で実行可能となっている．各値のレプリカに対しても定義した処理を実行し，リクエストの答えをハッシュ値 (Digest) で返す.

4.　評　　価

Cassandra に保存された値に対し処理を行う際に，Cassandra を通常利用した場合と，前章で説明した実装を用いた場合の性能比較を行う.

4.1　実 験 環 境

ノード数 8 台からなるクラスタに，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 1.1.0 を用いた．測定に用いたノードの性能を表 1 に示す.

表 1　マシン性能

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	8GByte
HDD	500GB 7200RPM SAS Disk
RAID Controller	SAS-6IR
Network	1Gbps

4.2　測 定 概 要

Cassandra によるクラスタを構築し，Cassandra の標準コマンド multiget [5] を使用して複数の値を取得してからワードカウントコマンド (`wc`) を実行した場合（以下クライアント側処理）と，今回実装した，複数の値に対して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機能を使用した場合（以下サーバ側処理）の，データノード数，処理対象の値の数の変化に伴う実行時間の差異を比較した．今回の測定

では，1 つの処理対象の値である 20MByte のテキストに対して wc を実行し，処理対象の値の数が 10 個の場合には wc が 10 回実行される．図 5，6 に処理対象の値の数を 2 とした際のそれぞれの処理の流れを示す．値 B のレプリカの表示については省略している．

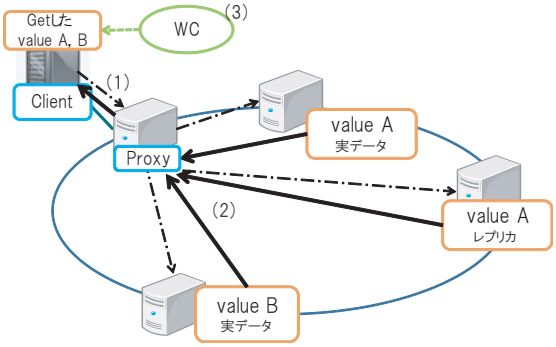

図 5　クライアント側処理の流れ

クライアント側処理の流れ

(1)　読み出しリクエストをクライアントから送信する．

(2)　担当するデータノードはリクエストに対する答えとして値 A，B をプロキシに返し，プロキシはその値をクライアントに返す．

(3)　読み出した値 A，B にをファイルに書き込み，ファイルに対して wc を直列実行する．ここで言う直列実行とは取得した値に対し wc 処理を 1 台のノード上で順次行うことを意味している．

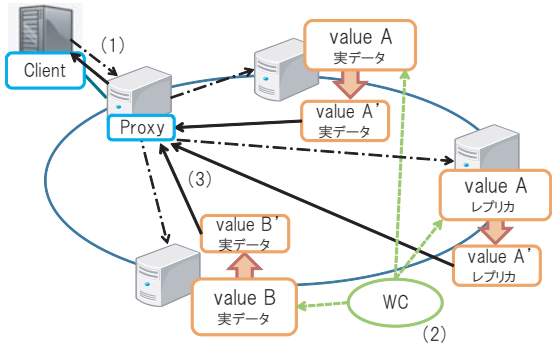

図 6　サーバ側処理の流れ

サーバ側処理の流れ

(1)　読み出しリクエストをクライアントから送信する．

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

(3)　データノードは値 A'，B' をプロキシに返し，プロキシ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) を一つの処理とする．データノード数を 3 台，5 台，8 台と変化させ，Cassandra のレプリケーション数は 3，一貫性レベルを ONE，ALL，value 数を 5〜70 まで変化させ，実行時間を測定した．value 数は処理対象の値の数を表している．

4.3　クライアント側処理，サーバ側処理性能比較

通常手法であるクライアント側処理と，提案手法であるサーバ側処理の性能比較を行った．図 7，8 にデータノード数を 3 台，5 台，8 台とし，一貫性レベルを ONE，value 数を 10，30 と指定した場合の wc の直列実行，クライアント側処理の値取得のみにかかる時間，クライアント側処理全体の処理時間，サーバ側処理の実行時間を示す．図 7 が value 数 10，図 8 が value 数 30 としている．縦軸が実行時間 (sec) で，横軸がデータノード数となっている．二つのグラフより，サーバ側処理の実行時間が value 数に関わらずクライアント側処理の実行時間に対し，5 分の 1 以下に抑えられている．これは，サーバ側処理が wc の処理結果をリクエストのレスポンスとして返すため，通信データ量が削減できたことと，処理を分散させて並列実行しているとにより，wc を直列実行しているクライアント側処理より wc 処理にかかる時間を短縮できたためである．また，クライアント側処理において，データノード数が増加してるのにも関わらず実行時間が一定になっていることがグラフから見られる．これは，クライアント側処理における値取得のみにかかる時間も台数に関わらず一定であることから，Cassandra から取得してきた，データサイズが比較的大きな値をプロキシがクライアントに返す過程がボトルネックになっているためだと考えられる．

図 7　ノード数を変化させた際の各処理の実行時間 (value 数=10)

図 8　ノード数を変化させた際の各処理の実行時間 (value 数=30)

4.4　value 数，データノード数を考慮したサーバ側処理性能

次に value 数，データノード数を考慮した際のサーバ側処理の実行時間を測定した．データノード数を増加させた場合のサーバ側処理の平均値を図 9 に示す．縦軸が実行時間 (sec) で，横軸がデータノード数となっている．図 9 より，value 数に関わらず，データノード数が増加すると実行時間が減少しており，

スケールアウトできているといえる．

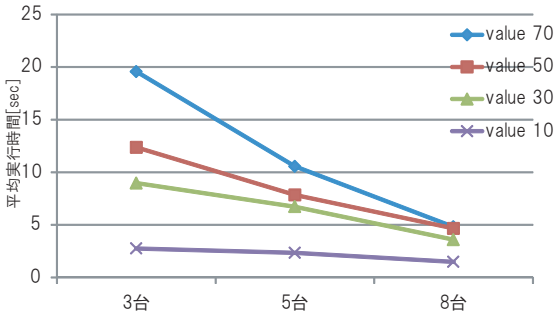

図 9　データノード数増加に伴う平均実行時間の変化

図 10〜12 に，図 9 の詳細を示す．図 10，11，12 はデータノード数をそれぞれ 3 台，5 台，8 台とし，value 数を 10〜70 まで増加させた場合の実行時間を表している．縦軸が実行時間 (sec) で，横軸が試行回数を表す．グラフより，データノード数，value 数に関わらず処理時間にばらつきが生じていることが確認できた．これは処理がどのレプリカ (今回の場合は 3 つ) のデータノードで実行されかはランダムに決定されるため，一部のデータノードに処理が偏る可能性があるためである．データノード数が 8 台の場合にはあまり大きなばらつきが生じていないことがグラフから見て取れる．これは試行回数が 10 回とあまり多くないため，データノード数が 8 台の際には特定のデータノードへの処理の極端な偏りが生じにくいためだと考えられる．以上のことより，台数を増やすことで，データノードあたりのデータ処理量が少なくなり，より高速に処理が行えているといえる．

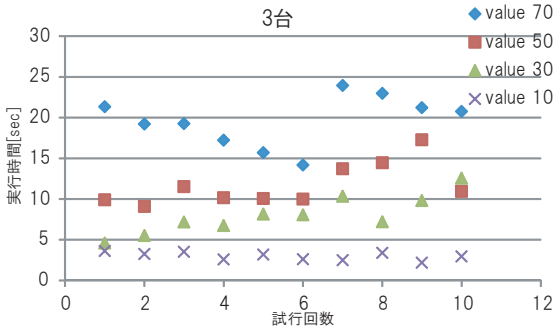

図 10　3 台の場合の実行時間

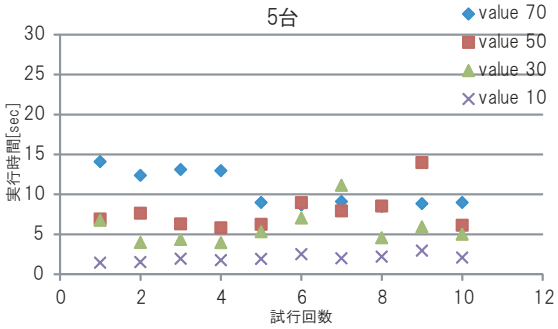

図 11　5 台の場合の実行時間

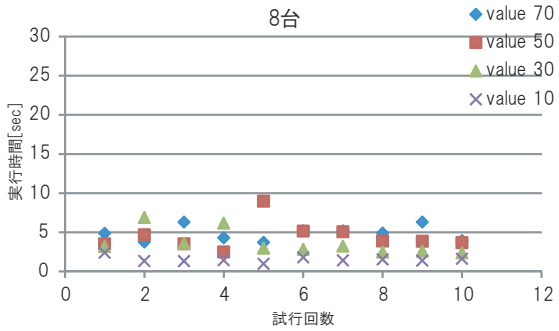

図 12　8 台の場合の実行時間

4.5　一貫性を考慮した処理性能

最後に一貫性レベル ALL を指定した際の実行時間の変化を測定した．図 13 にデータノードを 3 台もしくは 8 台，一貫性レベルを ALL とし，value 数を 30 と指定した場合の各処理の実行時間の変化を示す．縦軸が実行時間 (sec) で，データノード数となっている．一貫性レベル ALL を指定した場合も，クライアント側処理よりもサーバ側処理の方が高速である．また，サーバ側処理ではデータノード数が多い場合が高速になっており，一貫性レベル ONE を指定した時と同様の結果が得られた．よって本提案手法では，高い一貫性を保ちたい状況でも高速に処理することができることを確認した．

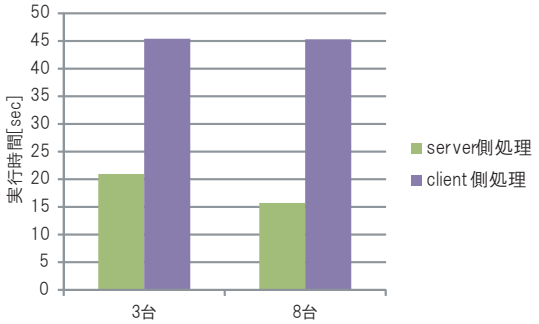

図 13　データノード数を変化させた際の各処理の実行時間

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として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，結果を取得して，それらの結果を一箇所に集約する．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て，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て，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

分散 KVS の実装の 1 つである Apache Cassandra に着目し，データアフィニティを考慮した並列分散処理を提案した．本稿ではユーザが指定した複数の異なる値に対し，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サーバ側処理機能を実装した．

実装した機能の処理時間を評価したところ，クライアント側処理に対して提案手法であるサーバ側処理では，value 数，一貫性レベル，データノード数に関わらず実行時間が大幅に削減されていた．また，サーバ側処理ではスケールアウトができていることが示せたことに加え，データノード数の増加に伴い特定ノードへの処理の集中を抑えることができており，処理を並列分散実行することで処理性能の向上につながることを示した．

今後の課題としては，Cassandra 上に保存されている値が依存関係にあった際の処理を追加すること，UDF と類似した機能を追加すること，Cassandra の Hadoop 連携機能との性能比較を行うことがあげられる．

文　献

[1] Jim Gray, David T. Liu, Maria Nieto-santisteban, Alexander S. Szalay, David DeWitt, and Gerd Heber. ”Scientific Data Management in the Coming Decade” Microsoft Technical Report, MSR-TR-2005-10.

[2] 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.

[3] Apache Cassandra:http://cassandra.apache.org/

[4] 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.

[5] Eben Hewitt(著), 大谷晋平, 小林隆 (訳):Cassandra, オライリー・ジャパン,2011

[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.