データアフィニティを考慮したCassandraによる並列分散処理の実装と評価

菱沼直子[†] 竹房あつ子[††]
中田秀基[††] 小口正人[†]

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HDFSなどの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．我々はこの問題に着目し，問題解決に向けて，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，実装する．本稿では，大容量データを扱う分散KVSであるApache Cassandraを拡張し，データアフィニティを考慮した並列データ処理機構を組み込んだ．評価結果から，本提案手法はCassnadraを通常使用した際よりも処理が高速に行えることが示された．

Implementation and Evaluation of Data Affinity-based Parallel Distributed Processing on Cassandra

Naoko Hishinuma,[†] Atsuko Takefusa,[††] Hidemoto Nakada[††]
and Masato Oguchi [†]

The spread of cloud computing enhances the necessity of accumulation of large amounts of data and high-speed data processing. Because it is difficult for a traditional RDBMS to process such Big Data, distributed KVS and distributed file systems are used for data accumulation and high-speed data processing, respectively. However, processing of the accumulated data causes large overheads of transmission from distributed KVS to the distributed file system. In order to address this issue, we propose a method that performs high-speed data processing directly on a distributed KVS. In this paper, we extend the Apache Cassandra database, a distributed KVS to handle large amounts of data, to enable data affinity-based parallel processing. The experimental results showed that the proposed method performs efficient processing compared with a conventional manner.

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，大量に生成されるデータの蓄積とその高速な処理が求められるようになり，従来のデータベース管理システムであるRDBMSではデータの蓄積や処理の柔軟性に関して不十分となってきた．そこで，大量に生成されるデータの蓄積にはApache Cassandra[1)2)]やApache HBase[3)]などの分散KVS(Key Value Store)が用いられ，高速な処理にはHadoop Distributed File System(以下HDFS)[4)]等が用いられている．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，その際に発生するコストが問題となる．よって，本研究では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，実装する．本稿では，大容量データを高速に蓄積可能な分散KVS型データベースCassandra[5)]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．Cas-

† お茶の水女子大学
Ochanomizu University
†† 産業技術総合研究所
National Institute of Advanced Industrial Science and Technology (AIST)

ⓒ 2013 Information Processing Society of Japan

sandra を機能拡張し，格納されたデータを効率よく活用するために，Cassandra に保存された複数の異なる値に対し，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れ，処理が高速に行えることが示された．

本稿は以下のように構成される．まず第 2 章で Cassandra と提案手法の概要について述べ，第 3 章で提案手法の実験概要と評価結果を示す．第 4 章で関連研究について述べ，第 5 章で本論文のまとめと今後の課題を述べる．

2. 提案手法の設計

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[2]．Cassandra 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点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．

Cassandra に保存されたデータに対して任意の処理を行う場合は，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である．しかし，Cassandra の読み出し処理性能はあまり高くない[6] 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処理できるようにするために，まず UDF(User Defined Function) に類似した機能を追加する．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．処理対象の値を複数指定した場合は，異なる値に対して並列処理が可能になり，より高速な処理が期待できる．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る．

本提案手法の概要を図 1 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) 各データノード上にある，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 1 では値 B のレプリカの処理は省略している．

(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．

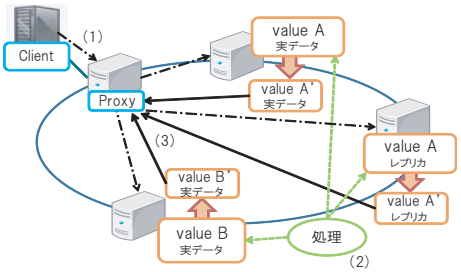

図 1 提案手法の概要

本稿では Cassandra の標準コマンド multiget[2] を拡張し，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) で返し，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．

3. 評　　価

本研究では，Cassandra に保存されたデータに対して処理を行う際に，Cassandra を通常使用した場合と，前章で説明した実装を用いた場合の性能比較を行う．

3.1 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 1.1.0 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

ⓒ 2013 Information Processing Society of Japan

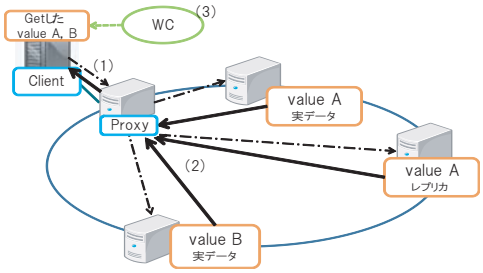

図 2　クライアント側処理の流れ

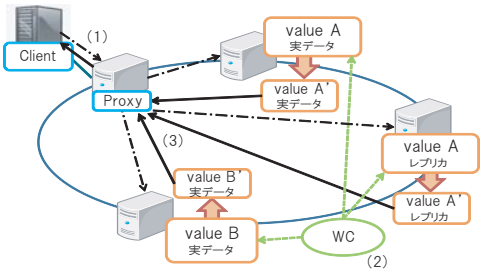

図 3　サーバ側処理の流れ

3.2　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 を使用し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，クラスタ参加ノード数，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，値が 10 の場合には wc が 10 回実行される．図 2，3 に値の数を 2 とした際のそれぞれの処理の流れを示す．

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 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．

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適する値 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．ノード数が 3 台，5 台，8 台のクラスタを用い，Cassandra のレプリカ数は 3(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 または ALL，value 数を 5〜30 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．一貫性レベル ONE，ALL はそれぞれ，処理完了とみなすのに必要なレプリカからのレスポンス数を表している．

3.3　クライアント側処理，サーバ側処理性能比較

図 4，5 にノード数が 3 台，5 台，8 台のクラスタを用いてレプリカ数を 3 として一貫性レベルを ONE，value 数を 10，30 と指定した場合の wc（クライアント側処理における wc にかかった時間），クライアント側処理，サーバ側処理の実行時間を示す．図 4 が value 数 10，図 5 が value 数 30 としている．縦軸が実行時間 (sec) で，横軸がクラスタ参加ノード数となっている．

図 4，5 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とが分かる．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ことにより，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，図 4，5 ともにクライアント側処理に関して，台数変化に伴う実行時間の変化が見られないのは，Cassandra から取得してきた値をファイルに一度書き出している作業がオーバヘッドになっているためだと考えられる．

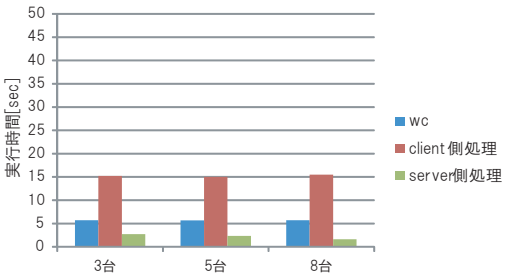

図 4　一貫性レベル ONE，value 数 10 の場合の実行時間

ⓒ 2013 Information Processing Society of Japan

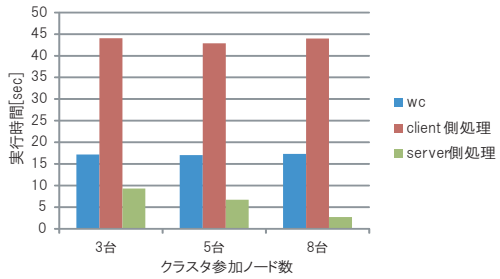

図 5　一貫性レベル ONE，value 数 30 の場合の実行時間

次に図 4，5 のサーバ側処理の結果に着目し，ノード数を変化させた場合のサーバ側処理の実行時間の平均値を図 6 に示す．各データの振れ幅は 5 回実行した際の，最大，最少実行時間を表し，縦軸が実行時間 (sec) で，横軸がクラスタ参加ノード数となっている．図 6 より，value 数に関わらず，ノード数が増加すると実行時間が減少していることが確認できる．最大，最少実行時間に幅があるが，これは処理がどのレプリカ（今回は 3 つ）のデータノードで実行されるかはランダムに決定されるため，一部のノードに処理が偏る可能性があるためである．

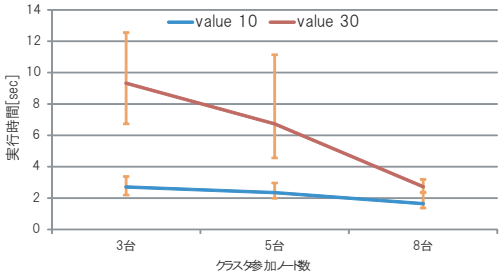

図 6　ノード数を変化させた際のサーバ側処理の実行時間

3.4　一貫性を考慮した処理性能

図 7 にノード数を 3 台，8 台，一貫性レベルを ALL とし，value 数を 5〜30 と増加させ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，横軸が value の数となっている．一貫性レベル ALL を指定した際は，処理を完了とみなすために必要なレスポンスの数が多くなるため全体的に処理が低速となっているが[6]，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った．また，サーバ側処理では参加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．

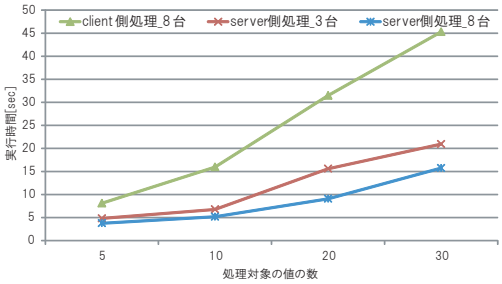

図 7　一貫性レベル ALL で value 数，ノード数を変化させた際の実行時間

4. 関 連 研 究

本研究に関連している研究として ParaLite[7)8)] があげられる．ParaLite は，SQLite をベースにする並列 RDBMS である．これは，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，結果を取得して，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を提供する．これにより，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．この機能は，本提案手法における，UDF と類似した機能を用い，ユーザが実行したい処理をプラグインとして定義する点と，リクエストの答えとして処理結果のみを取得する点が類似している．

また，本提案手法に類似した機能として HBase co-processor[9)] があげられる．これは，Google Bigtable のコプロセッサを基にしており，蓄積された大容量データに対しカウント，集約などの単純なプロセスをサーバ上で実行することで処理性能を向上させている点が本研究と類似している．しかし，HBase は CAP の定理における CP 特性があり，Cassandra は AP 特性があることから，SNS アプリケーションなどのように一貫性を犠牲にしても書き込み処理性能を向上させたい

ⓒ 2013 Information Processing Society of Japan

場面では Cassandra が有効であるなど，使用される状況に違いがあると考えられる．

5. おわりに

大容量データを高速処理する際に発生するコストを無くすため，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手法を提案した．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，従来手法の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，クラスタ参加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，サーバ側処理ではデータノード数の増加に伴い性能が向上していることより，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，本稿では各データノード上で実行した処理結果に対する集約処理が不可能だったため, 集約処理を可能にする機能を追加することがあげられる．また，実行する処理を事前に指定するのではなく，実行時に指定可能とするために UDF と類似した機能を追加することがあげられる．さらに，今回は Cassandra を最もシンプルに用いた手法を比較対象としたため，本研究との類似性の高い Cassandra の Hadoop 連携機能との性能比較を行い，本提案手法の特性をより明確にしていきたいと考えている．

参考文献

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) Eben Hewitt(著), 大谷晋平, 小林隆 (訳):Cassandra, オライリー・ジャパン,2011
3) HBase:http://hbase.apache.org/
4) Dhruba Borthakur, "HDFS Architecture," 2008 The Apache Software Foundation.
5) Brian F. Cooper, Adam Silberstein, Erwin Tam, Raghu Ramakrishnan, Russell Sears, "Benchmarking Cloud Serving Systems with YCSB" ACM Symposium on Cloud Computing, pp143-154, June 2010.
6) 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．
7) 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.
8) 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．
9) HBase/coprocessor:
https://blogs.apache.org/hbase/

ⓒ 2013 Information Processing Society of Japan