# Cassandraによるデータアフィニティを考慮した並列分散処理の実装

菱沼 直子[1]　竹房 あつ子[2]　中田 秀基[2]　小口 正人[1]

**概要**：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．この問題の解決に向けて，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．我々は既発表研究において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でいる．本稿では，並列データ処理機構の特性を明らかにするため，データの蓄積を行いつつ，高速データ処理を行った場合の性能と，処理の偏りと性能の相関を調査した．評価より，本実装はWrite中の影響を受けることが確認でき，影響を少なくするためには処理の偏りを小さくし，処理を担当しないノードの発生を防ぐことがより重要であることが分かった．

# Implementation of Parallel Distributed Processing in Consideration of Data Affinity by Cassandra

Naoko Hishinuma[1]　Atsuko Takefusa[2]　Hidemoto Nakada[2]　Masato Oguchi[1]

## 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]などの処理性能を重視する分散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．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[5]．

本研究では上記転送コスト発生を防ぐため，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．本提案手法では，分散KVS上で高速な蓄積と高速な処理を実現するため，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[6]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Cassandra上で高速に処理するための手法を提案する．

我々は，既発表研究において格納された値に対して直接高速データ処理を行うためCassandraの標準コマンドを機能拡張し，Cassandraに保存された複数の異なる値に対し，

[1] お茶の水女子大学
Ochanomizu University, Bunkyo, Tokyo 112-8610, Japan
[2] 産業技術総合研究所
AIST, Tsukuba, Ibaraki 305-8568, Japan

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並列データ処理機構を組み込んでいる．並列データ処理機構が Cassandra を単なるストレージとして最もシンプルな形で用いた場合に比べ，処理の高速化を達成できていることを確認している [7]．しかし，本実装は 1 つの分散 KVS 上で蓄積と高速データ処理の両方を行うため，書き込み処理中に高速データ処理が実行されると性能に影響があることが予想される．また，本実装はどのデータノードで高速データ処理が実行されるかはランダムに決定されるため，処理の偏りが生じてしまい，全体の性能に影響を与えることも予想される．本稿では，書き込み処理と高速データ処理がお高速データ処理に与える影響と，高速データ処理の偏りが性能に与える影響を調査した．評価から，同時に実行されている書き込み処理が高速データ処理の性能に影響を与えることが分かり，影響を少なくするためには処理の偏りを小さくし，処理を担当しないノードの発生を防ぐことが重要であることが分かった．

## 2. 提案手法の設計

### 2.1 提案手法の概要

本研究では，Apache Cassandra に着目する．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．Cassandra は Eventual Consistency（結果整合性）を採用しているため大容量データを高速に蓄積することを得意とする．

Cassandra に高速蓄積された大容量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，HDFS など分散ファイルシステム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い，Cassandra から分散ファイルシステム等へ転送する際に大きなコストが生じてしまう．

我々は上記転送コストの発生を防ぐため，Cassandra 上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ている [7]．本提案手法では，UDF(User Defined Function) に類似した機能を用いることで，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．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．定義された処理をデータを保存している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．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．

提案手法の概要を図 1 に示す．

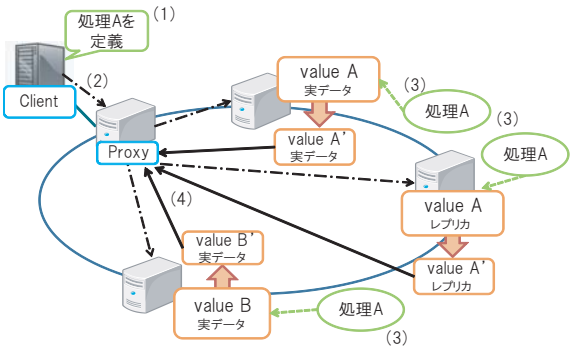


**図 1** 提案手法の概要

(1) UDF と類似した機能を用い，任意の処理 A 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．
(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 A 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 1 では値 B のレプリカの処理は省略している．

### 2.2 並列データ処理機構の概要

既発表研究で実装した機能について説明する．Cassandra の標準コマンド `multiget`[2] を拡張し，`multiget` 処理の読み出し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，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．`multiget` は，指定した複数の key の指定したカラムの値を取得するコマンドである．本実装の概要を図 2 に示す．

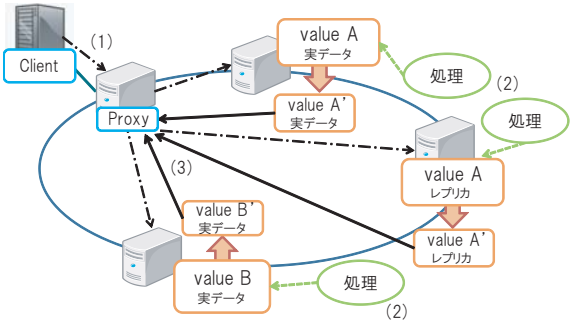


**図 2** 並列データ処理機構の概要

(1) クライアントから各データノードへ読み出しリクエス

ト送信する．

(2) 各データノード上にある，異なる値 A,B に対して事前に指定した処理を並列実行し，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実装になっているが，図 2 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．

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，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事前に指定し処理を実行し，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．

## 3. 書き込み処理中の高速データ処理性能評価

本稿ではまず，書き込み処理が高速データ処理の性能へ与える影響を調査する．

### 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 |

### 3.2 測定概要

並列データ処理機構が書き込み中に高速データ処理を実行するとどの程度の性能に影響が出るのか調査する．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 の場合には `wc` が 10 回実行される．Write なしでは，高速データ処理を行う際に書き込み処理がない状態を表し，Write ありでは高速データ処理と同時に書き込み処理も実行されている状態を表す．図 3 に値の数を 2 とした際の処理の流れを示す．

**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．
この際 Write ありでは書き込み処理が同時に実行されている．

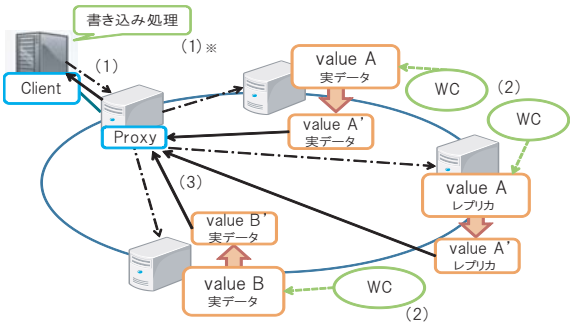


**図 3** 処理の流れ

(2) 担当するデータノードはリクエストされた値 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．

(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，書き込み処理の一貫性レベルを ALL とし，高速データ処理の読み込み時の一貫性レベルを ONE，value 数を 10〜70 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．一貫性レベル ONE は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が 1 つであることを表している．

### 3.3 書き込み処理が実行時間に与える影響

書き込み処理が実行時間に与える影響を調査した．図 4,5 に Write なしの場合，Write あり場合の value 数，クラスタ参加ノード数を変化させた際の実行時間の変化を示す．縦軸が実行時間 (sec)，横軸がクラスタ参加ノード数を表す．実行時間は前節で説明した処理を 10 回繰り返した際の平均実行時間となっている．

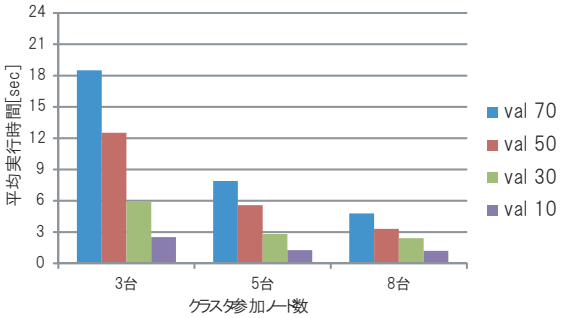


**図 4** Write なし平均実行時間

図 4,5 から，書き込み処理がある場合の方が実行時間が遅くなる傾向があることから，書き込み処理が高速データ処理に影響を与えてしまう事が確認できた．実際にどの程度の影響が生じているかを詳細にするため，実行時間の差と遅延率に着目する．遅延率は以下の式で算出している．

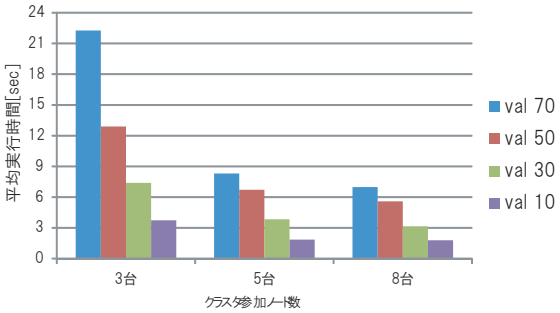


**図 5**　Write あり平均実行時間

$$\text{遅延率} = \frac{(\text{Write なし実行時間}) - (\text{Write あり実行時間})}{(\text{Write なし実行時間})}$$

図 6 は，Write がある場合，ない場合の実行時間の差を表す．縦軸が平均実行時間の差を表し，横軸がクラスタ参加ノード数を表す．図 7 は，上記計算式で算出した，実行時間の遅延率を示す．縦軸が遅延率 (%)，横軸がクラスタ参加ノード数を表す．

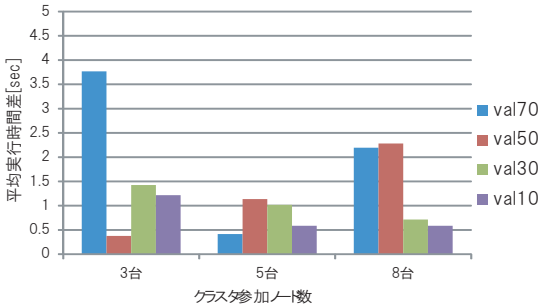


**図 6**　Write によてって生じた実行時間の差

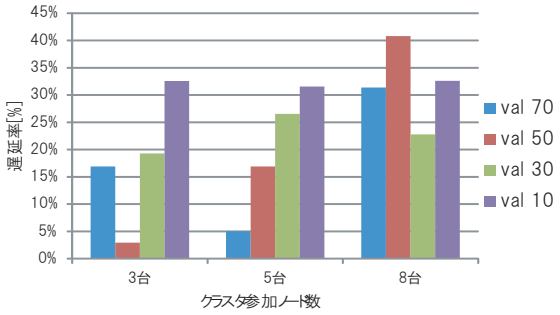


**図 7**　実行時間の遅延率

図 6 と図 7 から，クラスタ参加ノード数が多くなると遅延率が 25 %以上になるケースが多くなり，遅延率の上昇が確認できるが，ノード数 8 台のときの実行時間差は大きくても 2.5 秒程度に抑えられていることが分かる．これは，クラスタ参加ノード数が多くなると，全体の実行時間が短くなるため，わずかな実行時間の差が大きな遅延率として現れている．しかし，Cassandra の使用される状況を考えると，処理のレスポンスの速さが重要となってくるため，書き込み処理と高速データ処理を両立する場合には，この遅延率を極力小さくする工夫が必要となってくる．

## 4.　処理の偏りに関する性能評価

次に，実行時間と処理の偏りの相関を調査した．前章で説明した実験を 10 回繰り返し行った際の，実行時間と処理件数の変化について測定し，同時に各データノード毎の処理全体に占める担当する処理の割合を求めた．

図 8 に value 数を 150，クラスタ参加ノード数を 8 台とした際の実行時間と処理件数の変化を示す．左縦軸が実行時間 (sec)，右縦軸が処理件数を表し，横軸が試行回数となっている．図 9 には，1 回の試行の処理全体に占める，各データノードが担当した処理の割合を示す．行が試行回数を表し，列がデータノードを表している．例えば，1 行目の 3 列目は 1 回目の試行でデータノード 3 が担当する処理の割合が処理全体の 3.3 %であることを意味している．

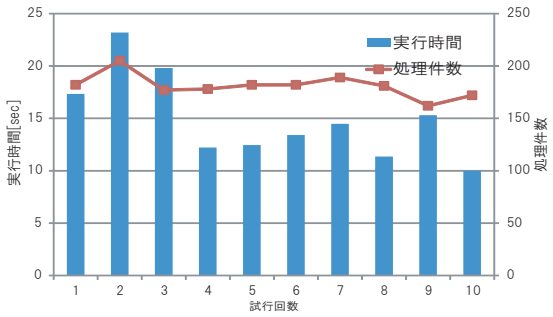


**図 8**　実行時間と処理件数の変化

| MIN | MAX |
|---|---|

| try/ node | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| 1 | 14.8% | 17.6% | 3.3% | 1.1% | 3.3% | 4.4% | 23.6% | 31.9% |
| 2 | 14.1% | 2.9% | 2.4% | 3.4% | 45.4% | 18.5% | 6.8% | 6.3% |
| 3 | 40.1% | 1.1% | 18.6% | 3.4% | 4.0% | 2.8% | 26.0% | 4.0% |
| 4 | 9.0% | 2.2% | 2.2% | 3.4% | 2.8% | 21.3% | 25.3% | 33.7% |
| 5 | 11.5% | 1.6% | 1.6% | 3.3% | 3.8% | 22.0% | 24.2% | 31.9% |
| 6 | 13.7% | 0.5% | 0.5% | 3.3% | 4.4% | 18.7% | 25.8% | 33.0% |
| 7 | 39.2% | 1.1% | 3.2% | 3.2% | 28.0% | 16.9% | 3.2% | 5.3% |
| 8 | 15.5% | 10.5% | 4.4% | 3.3% | 3.9% | 8.8% | 23.2% | 30.4% |
| 9 | 45.1% | 0.0% | 0.0% | 3.7% | 1.9% | 20.4% | 27.2% | 1.9% |
| 10 | 10.5% | 1.2% | 1.2% | 4.7% | 2.3% | 19.8% | 25.0% | 35.5% |

**図 9**　処理全体における各データノードが担当する処理の割合

図 8 で試行 2，3，9 回目に着目すると，実行時間が他の試行回よりも長くなっていることが分かる．この際の処理の偏りについて図 9 で確認したところ，どの試行も処理が特定のデータノードへ 40 %以上集中していた．試行 9 回目については，処理件数が少ないのに実行時間が長くなっているが，これは図 9 から，処理を 1 件も担当していないデータノードが 2 ノードほど存在しているためだと考えられる．また，実行時間が比較的短い試行 4，5，8，10 回目の処理の偏りは，処理が一番集中しているデータノードでも 35 %程度に抑えられていることが確認できる．今回の実験は連続して 10 回処理を繰り返しているため，試行 1 回目はキャッシュが効かないため実行時間が長くなり，試行 10 回目はキャッシュの効き実行時間が短縮されたと考

えられる．以上のことより，実行時間の短縮のためには処理の偏りを小さくすることに加え，処理を担当しないデータノードの発生を防ぐことが重要であることが分かった．

## 5. 関連研究

本研究に関連している研究として ParaLite[8][9] 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 coprocessor[10] 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，Cassandra は Eventual Consistency（結果整合性）を採用しているため，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場面では有効であるなど，使用される状況に違いがあると考えられる．

## 6. まとめと今後の課題

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．既発表研究において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並列データ処理機構を実装済みである．本稿では実装した並列データ処理機構の特性を明らかにするため，書き込み処理，処理の偏りが高速データ処理に与える影響を調査した．

評価したところ，本実装では書き込み処理が高速データ処理に影響を与えることが確認できた．また，処理の偏りと実行時間の関係を調査したところ，書き込み処理が高速データ処理に与える影響を小さくするためには，処理の偏りを小さくし，処理を担当しないノードの発生を防ぐことが重要であることが分かった．

今後の課題としては，処理を担当しないノードの発生を防ぐために，処理の分散機構を組み入れ，性能を向上させることに取り組んでいきたいと考えている．また，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．さらに，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．

### 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] 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.

[6] 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.

[7] 菱沼直子, 竹房あつ子, 中田秀基, 小口正人, ” 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” DEIM Forum 2013，F2-3，2013 年 3 月

[8] 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.

[9] 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.

[10] HBase/coprocessor: https://blogs.apache.org/hbase/