Cassandra を用いた並列分散処理機構の開発と Hadoop Cassandra との比較

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 ‡産業技術総合研究所

1. はじめに

クラウド技術の普及により，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の保持には，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分散 KVS が利用されている．これらのデータから，ユーザが必要とする情報を抽出するためには，データの統計処理やマイニング処理が不可欠となる．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，Apache Hadoop 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出来る．しかし，Hadoop を用いる際には分散 KVS から処理を行う HDFS 等の分散ファイルシステムへデータを転送する必要があり，その際に大きなコストが発生してしまう．

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 Apache Cassandra[1] には，Hadoop 連携機能が実装されている．この機能を用いれば，Cassandra 上に保存されている値に対して Hadoop を利用することが可能である．しかし，Hadoop Cassandra ではデータ処理に MapReduce を用いるため，小さなデータを扱う処理やリアルタイム処理は不向きである．そこで我々は既存研究において，転送コスト発生を防ぎつつ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，並列データ処理機構という形で実装した [2]．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，本実装の特性を調査する．比較実験より，小さなデータを扱う場合には並列データ処理機構を用いた方が実行時間が短く，本提案手法の有効性を示すことが出来た．

2. 関連研究

関連研究として Jobcast[3] があげられる．Jobcast 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．本実装と異なる点としては，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あげられる．

3. Hadoop 連携機能 (Hadoop Cassandra)

Hadoop Cassandra は，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 上に蓄積されたデータに対して MapReduce を用いて処理する．この実装を用いれば問題となっていた転送コストを削減できると予想されるが，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．

4. データアフィニティを考慮したデータ処理手法

我々は [2] において，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．提案した手法では，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する．定義された処理を，処理対象データを保存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 1 に示す．

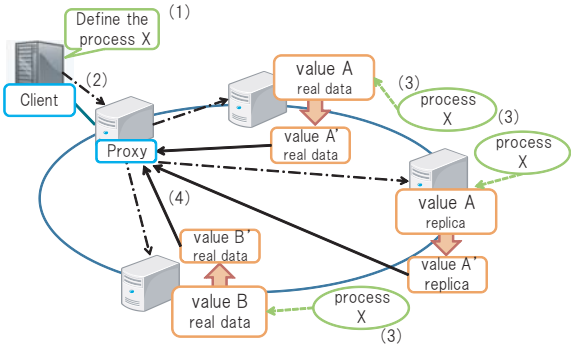

図 1: 提案手法の概要

(1) UDF として，処理 X を定義する．

(2) 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，値 A,B に対して定義された処理 X を並列実行し，処理結果を値 A',B' とする．

(4) 値 A',B' をリクエストの答えとして返す．

既存研究 [2] では上記手法を実現させるために Cassandra の標準読み出しコマンド，`get`，`multiget slice` を拡張し，並列データ処理機構を実装した．並列データ処理機構は，UDF を Java で定義し，読み出しリクエストを送信すると，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し，Cassandra に保存された複数の異なる値に対して処理を各データノード上で実

Comparision with a Distributed Parallel Proccessing Mechanism for Cassandra and Hadoop Cassandra
† Naoko Hishinuma, Masato Oguchi
‡ Atsuko Takefusa,Hidemoto Nakada
Ochanomizu University (†)
National Institute of Advanced Industrial Science and Technology (AIST)(‡)

行し，処理結果のみをリクエストの答えとして返す実装となっている．

5. 比較実験

本研究では，Cassandra に保存された値に対して処理を行う際に，前章で説明した実装を用いた場合と Hadoop Cassandra を用いた場合の性能比較を行う．

ノード数 3 台，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 1.2.0 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	8GByte
HDD	500GB 7200RPM SAS Disk x 2

Cassandra 上に保存されている値 500 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 500 回実行するのにかかる時間を測定する．測定に用いたパラメータ設定を表 2 に示す．

表 2: 測定に用いたパラメータ設定

データノード数	3, 8
処理を行う値の数	500
一つの値の大きさ (byte)	1K,10K,100K,1M

図 2，3 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の変化を示す．縦軸が実行時間 (sec)，横軸が値の大きさ (byte) を表す． 図 2，3 より，値の大きさ，

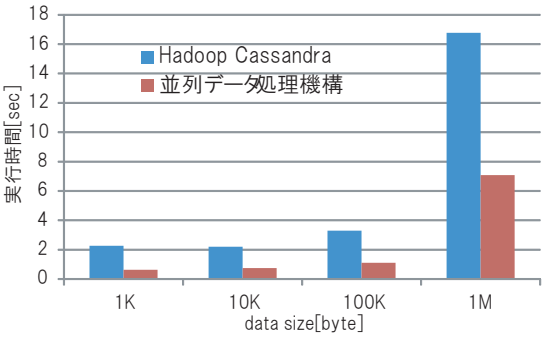

図 2: ノード数が 3 台の場合の実行時間

ノード数に関わらず並列データ処理機構を用いた場合の方が実行時間を抑えることが出来ている．また，並列データ処理機構では台数増加に伴い性能向上の傾向がみられるが，Hadoop Cassandra ではそのような傾向は確認できなかった．Hadoop Cassandra があまり高い性能を出すことが出来なかった原因としては，処理を実行するノードに値を集め，処理を 1 ノードで実行しているため，転送コストが削減できていなかったためと考えられる．以上の結果より，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．

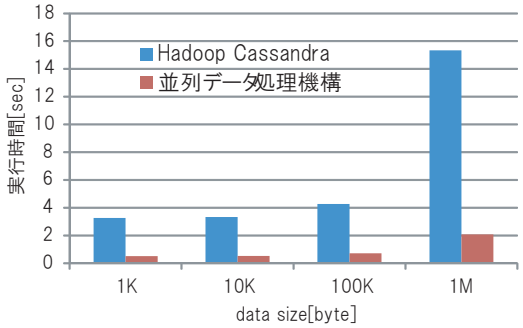

図 3: ノード数が 8 台の場合の実行時間

6. おわりに

我々は既存研究において，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るため，Cassandra に着目して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，Cassandra の標準コマンドを機能拡張し並列データ処理機構という形で実装済みである．これに対し，Hadoop のストレージとして Cassandra を用いた Hadoop Cassandra という実装があるため，本稿では並列データ処理機構と Hadoop Cassandra の性能比較を行い，本実装の特性を調査した．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．

参考文献

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

[2] 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.

[3] 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.