Cassandra によるデータアフィニティを考慮した 並列分散処理の提案と性能評価

理学専攻　情報科学コース　菱沼　直子

1　はじめに

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，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．

そこで，本研究では転送コスト発生を防ぎ，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，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 を用い，データアフィニティを考慮して大容量データを格納している Cassandra のデータノード上で高速に処理する．我々は，上記手法を Cassandra の読み出しコマンドを拡張し並列データ処理機構という形で実装した．評価実験より，並列データ処理機構の特性を明確にする．

2　Apache Cassandra

本研究では，KVS 型データベースである Apache Cassandra[1][2] に着目した．Apache Cassandra は，Facebook 社が開発した分散データベース管理システムである．Cassandra の特徴としては，読み出しコストを多少犠牲にしながら，書き込み性能を向上させた分散 KVS であること，単一故障点 (SPOF) がないこと，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．

3　提案手法の設計

本研究では 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する．本提案手法では，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．

(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．

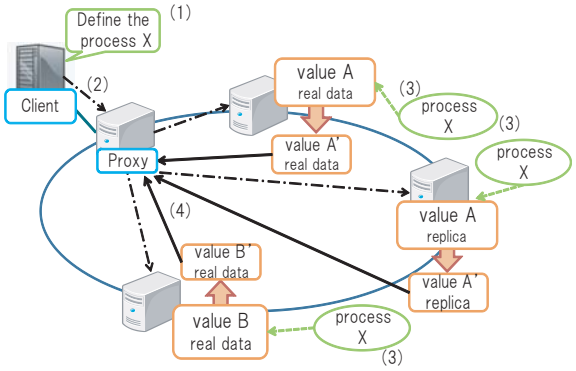

図 1: 提案手法の概要

本研究では，上記手法を 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という形で実装した．並列データ処理機構は，UDF を Java で定義しており，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，処理を各データノード上で実行して，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．本実装では定義した処理の JAR ファイルをデータノード上へ転送することで任意の処理を実行可能にしている．

4　性能評価

実装した並列データ処理機構の性能評価として基本性能，データ処理中にバックグラウンドで書き込み処理が行われている場合の性能を評価する．ノード数 3 台，5 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

4.1　基本性能

前章で説明した実装を用いて，各データノード上でワードカウント (`wc`) を行い処理結果のみを取得する場合 (サーバ側処理) と，Cassandra から対象の値を取得後 `wc` を行う場合 (クライアント側処理) での性能比較を行った．

図 2，3 にデータノード数を変化させた際の `wc` 処理 (1

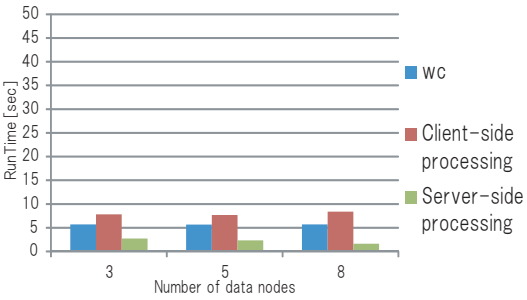

図 2: 処理対象の値を 10 とした場合の各処理の実行時間

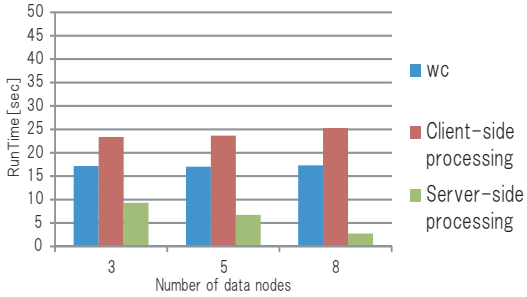

図 3: 処理対象の値を 30 とした場合の各処理の実行時間

台のマシン上で実行)，クライアント側処理，サーバ側処理の実行時間を示す．縦軸が実行時間 (sec)，横軸がデータノード数で，図 2 が処理対象の値を 10，図 3 が処理対象の値を 30 とした際の実行時間を表している．グラフより，データノード数，処理対象の値に関わらず提案手法であるサーバ側処理の実行時間が短くなっていることから，本提案手法の有効性が示せた．この結果は，提案手法が処理結果のみを返していることによる通信データ量の削減が出来ているためである．

4.2 書き込み処理中の高速データ処理性能

本実装では 1 つのシステム上で書き込み処理とデータ処理を行っているため，書き込み処理中にデータ処理を実行すると影響が出ることが予想される．そこで本章では，並列データ処理機構がデータ処理中にバックグラウンドで書き込み処理を実行するとどの程度性能に影響が出るのか調査する．

図 4，5 にデータノード数を 3 台，8 台，処理対象の値の数を 50，100，200 とし，書き込み throughput を 0〜10Mbyte/sec 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)，書き込み throughput(byte/sec) を表す．

図 4，5 より，書き込み負荷を与えない場合と 10Mbyte/sec 程度の書き込み負荷を与えた場合の実行時間を比較した結果，実行時間増加の傾向があることから書き込み負荷がデータ処理性能に影響を与えていることが確認できる．ただし，データノード数が少ない場合 (3 台) には 10Mbyte/sec 程度の書き込み負荷を与えるとデータ処理性能の劣化が大きいが，データノード数が多い場合 (8 台) には 10Mbyte/sec 程度の書き込み負荷を加えても効率よく処理することができていた．以上の結果から本実装はデータノード数を増加させることでスケールアウトできていることも示せた．

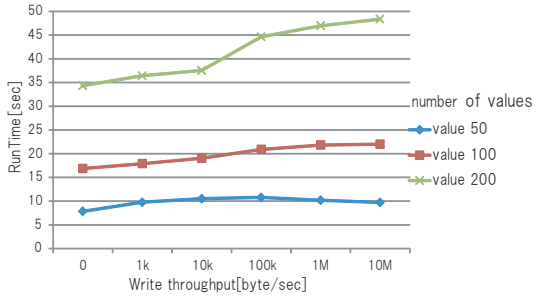

図 4: ノード数 3 の場合の書き込み処理中の高速データ処理実行時間

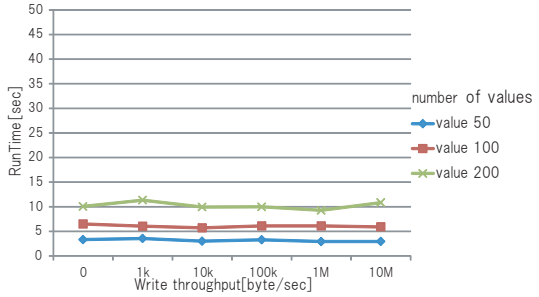

図 5: ノード数 8 の場合の書き込み処理中の高速データ処理実行時間

5 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抑えるため，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．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並列データ処理機構を実装した．

評価結果から，Cassandra を単なるストレージとして利用するクライアント側処理に比べ，本提案手法を用いたサーバ側処理の方が実行時間が短いことが確認でき，本実装の有効性が示せた．また，書き込み処理がデータ処理に影響を与えることが確認でき，本実験環境ではデータノード数が 3 台の場合は 10Mbyte/sec 程度の書き込み負荷を与えるとデータ処理性能の劣化が見られるのに対し，8 台の場合は 10Mbyte/sec の書き込み負荷を与えても効率よく処理を実行できていた．

今後の課題としては，ワードカウント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．実装面の課題としては，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 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] A.Lakshman and P.Malik. ”Cassandra - A Decentralized Structured Storage System,” Operating Systems Review, vol.44, No.2, pp.35-40, April 2010.

[2] Eben Hewitt.,Cassandra: The definitive guide, trans. Shinpei Ohtani and Takashi kobayashi. O’Reilly JAPAN, 2011.