Vol.2011-ARC-195 No.23
Vol.2011-OS-117 No.23
2011/4/14

Cassandraを使ったスケーラビリティのあるCMSの設計

玉城将士[†1]　谷成雄[†2]　河野真治[†3]

本研究では，スケーラビリティのある CMS を開発するために，100 台の PC クラスタを使い Cassandra のスケーラビリティの検証を行った．その結果 Cassandra の特徴やスケールする条件の検証，スケーラビリティの検証環境を構築することが出来た．
今回は，その検証結果を元にスケーラビリティを確保する方法を検討した．その方法に則り CMS を設計し，データ構造として用いる非破壊的木構造の実装を行った．

Design of Scalable CMS using Cassandra

Shoshi TAMAKI,[†1] Yu TANINARI[†2]
and Shinji KONO [†3]

To develop scalable CMS , We built scalability verification environment with 100 PC Clusters to verify scalabilty of Cassandra. As a result , We confirm a scalability verification method , feature and scale condition in Cassandra.
In this time , We considered how to secure scalabilty confroming to the verification. According as the method , We designed CMS and implemented Monotonic Tree-Modification for the CMS's data structure

†1 琉球大学理工学研究科情報工学専攻
Interdisciplinary Infomation Engineering, Graduate School of Engineering and Science, University of the Ryukyus.
†2 琉球大学工学部情報工学科
Infomation Engineering, University of the Ryukyus.
†3 琉球大学工学部情報工学科
Infomation Engineering, University of the Ryukyus.

1. 研究の目的

Cassandra は複数のサーバーで動作を想定した分散データベースである．前回は，実際に分散させることによって高価なサーバーを超えることが出来る性能を出すことが出来るのか，また，どの様に Cassandra 上で動くソフトウェアを開発することによって性能を発揮することが出来るのかを，90 台の PC クラスタ上でベンチマークを取り検証した．結果として，Cassandra の特徴やスケールする条件の検証，スケーラビリティの検証環境を構築することが出来た．

本研究では，前回構築したスケーラビリティの検証環境と Cassandra の検証結果を用いて，多段キャッシュと非破壊的木構造をデータ構造に用いた CMS の設計・開発を行った．

2. Cassandra

Cassandra は, FaceBook が自社のために開発した分散 Key-Value ストアデータベースであり，Dynamo[2] と BigTable[5] を合わせた特徴を持っている．2008 年にオープンソースとして公開され, 2009 年に Apache Incubator のプロジェクトとなった．2010 年には Apache のトップレベルプロジェクトとなり, 現在でも頻繁にバージョンアップが行われている．

2.1 ConsistencyLevel

Cassandra には, ConsistencyLevel が用意されている．これは, 整合性と応答速度どちらを取るか選ぶためのパラメータであり, リクエストごとに設定することが出来る．
また, Read と Write で ConsistencyLevel の意味は異なる．この ConsistencyLevel を適用するノードの台数を ReplicationFactor といい, Cassandra の設定ファイル，またはクライアントより設定することが出来る．

2.2 SEDA

SEDA(Staged Event-Driven Architecture) は, Cassandra で使用されているアーキテクチャである[3][4]．処理を複数のステージに分解しタスクキューとスレッドプールを用意し処理を行う．処理の様子を図 1 に示す．
タスクが各ステージのタスクキューに入ると, スレッドプールにどれかのスレッドがタスクキューの中からタスクを取り出し処理を行う．処理が終わるとそのタスクを次のステージの

© 2011 Information Processing Society of Japan

タスクキューに入れる. 同様にして次のステージのスレッドプールがタスクキューからタスクを受け取り処理を行う.
このアーキテクチャは数多くのスレッドを生成するためマルチコアな PC と多数のタスクがある状況で性能を発揮することができる.
実際，Cassandra には 20 以上のスレッドが動作している．しかし, あまりにもスレッドプールやタスクが多すぎると, コンテキストに切り替えに時間がかかり性能は低下する.

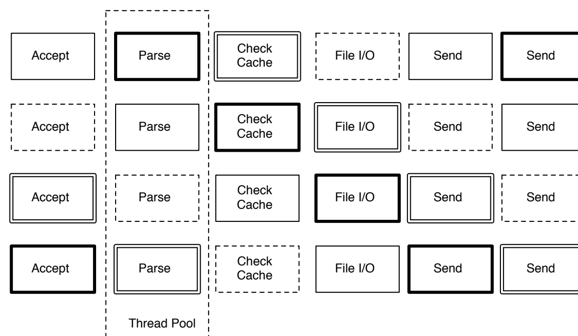

図 1　SEDA

2.3　PC クラスタを用いた Cassandra の検証結果

前回の研究[1)] で，PC クラスタを用いた Cassandra の検証で行った MySQL と Cassandra の比較から得られた特徴について簡単にまとめる.
クラスタ管理ツールの Torque を使用し, 使用するノード数を指定してクラスタにジョブを投げて PHP スクリプトを実行させる．この PHP スクリプトは対象のサーバーにリクエストを 10000 回送信するスクリプトである．実験の概要図を図 2 に示す.

この実験では，徐々に負荷をかけるクラスタの台数を増加させ，並列度を上げていく．クラスタすべてが処理を完了するまでの平均をグラフにプロットし，比較した.

2.3.1　2Core を搭載したコア数の少ないサーバーを用いた検証

Read は Cassandra/MySQL ともに，似たような性能低下の推移をしていたが Cassandra の方が遅い．しかし, Write は Cassandra の方が断然速く動作している．この実験では, Cassandra の動作を基準に考えたため書き込みのコマンドに REPLACE を使用した. REPLACE は置き換えるようなコマンドである.
そのため, INSERT に比べて多少遅くなる. SEDA は複数のスレッドで動作しているためコア数が少ないサーバーでは性能が出にくいことがわかる.

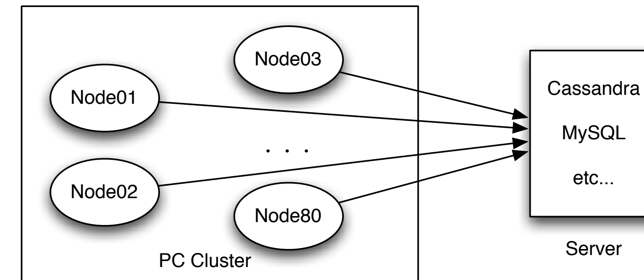

図 2　PC クラスタを用いた Cassandra の検証環境

2.3.2　4Core8Threads を搭載したコア数の多いサーバーを用いた検証

Read/Write 共に MySQL の性能を超えることに成功した. Read においてはコア数が少ない場合に超えることが出来なかったが, 並列度が 70 度付近で MySQL を上回る性能がでていた. Cassandra の平均時間は並列度が増加しても, MySQL よりは平均時間の上昇は少ない．これは, SEDA の特徴である多くのタスクを並列に実行すると性能を発揮することを確認することが出来た.
また, SEDA はマルチスレッド前提であるため, コア数が少ないサーバーでは性能が出ず, コア数の多いサーバーで性能が発揮できるということが分かる.

2.3.3　クラスタ化した Cassandra を用いた検証

Read/Write ともに, 今回の場合は分散を行わなかったほうが性能を引き出せてることが分かった．これは, 実験に使用したデータが Read/Write 共に 1 つだけで, 結局は同じノードにリクエストが転送されている．そのため, リクエストは 1 台のノードに集中する．よって, 性能が出ないのではないかと考えられる. Cassandra をただ増やすだけでは性能は得ることが出来ず, データも分散させて実験を行わなければならない.

© 2011 Information Processing Society of Japan

Vol.2011-ARC-195 No.23
Vol.2011-OS-117 No.23
2011/4/14

2.3.4 まとめ

以上の実験より，Cassandra はコア数の多いサーバーかつ READ/WRITE を並列に行い，なおかつ使用するデータ構造も工夫が必要であるということが分かった．

3. スケーラビリティのある CMS の設計

Cassandra の検証で得られた結果に基づき，スケーラビリティのある CMS の設計を行う．スケーラビリティがあるということは以下のような特徴がある．

(1) 大きな負荷がかかっても性能が低下しない

(2) ノードの台数を増やすだけで性能を維持することが出来る

これらの特徴を実現する為に，２つの方法が挙げられる．

(1) データのコピーを複数用意する方法

　データのコピーを複数用意することにより，データのアクセスが集中することを防ぐ．

(2) データの更新通知を行わずポーリングを行う方法

　複数コピーされたデータを同期するためには更新を通知する必要があるが，実際には全ての更新結果をコピー先が把握する必要はなく，コピー先が必要になったときのみ同期を行えば良い．

この２つの方法に則り設計を行った．

3.1 提案するシステムのアーキテクチャ

提案するシステムのアーキテクチャを図 3 に示す．

Cassandra をバックエンドにその上に CMS の API を提供するサーバーを構築し，Web サーバーが API サーバーを呼び出す形をとる．こうすることで，API サーバー自体のスケーラビリティを bot などを用いて計測することが出来るためである．また，クライアントを Web と限定せず，デスクトップクライアントなど多様なクライアントに対応することが出来る．

提案するシステムでは各ステージ (Cassandra/API サーバー/Web サーバー) で方法 1 に則りキャッシュを用意し，アクセスが集中するのを防ぐ．また，方法 2 に則り，各ステージのキャッシュは必要なとき自分より上のステージよりポーリングし更新の有無を確認する．この方法を用いることでスケーラビリティのある CMS が構築できるのではないかと考えられる．

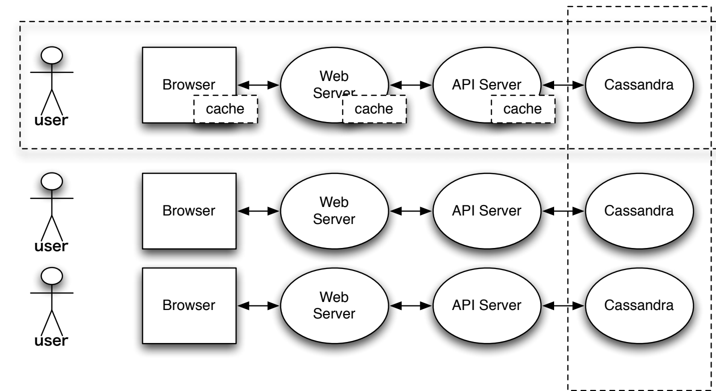

図 3 提案するシステムの概要

3.2 スレッドセーフな木構造の開発

CMS のデータ構造としは，木構造を採用することが出来る．しかし，スケーラビリティのあるシステムで使用するデータ構造はスケールする必要があるため，スケールする木構造を開発する必要がある．そこで，スレッドセーフな木構造である非破壊的木構造について説明する．

3.2.1 破壊的木構造

一般的に使用されている木構造はメモリ上の木構造を書き換えて編集する破壊的木構造である．この木構造は編集する際に木にロックを掛ける必要があり，編集時には木を走査しようとしているスレッドは書き換えの終了を待つ必要，閲覧者がいる場合は木の操作を終了するのを待つ必要があり，スケールしないと考えられる．

ⓒ 2011 Information Processing Society of Japan

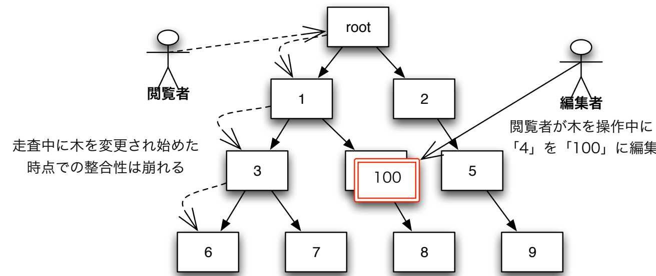

図 4　破壊的木構造

3.2.2　非破壊的木構造

今回設計した CMS ではデータ構造として非破壊的木構造を用いる．非破壊的木構造とは，編集時にルートノードから編集する対象となるノードまでのパスをコピーし，変更のないノードは古い木構造と共有する．そしてコピーしたルートノードを編集された木とする．こうすることで，編集前の木構造は破壊されず，木構造を走査していながら編集することが可能になる．

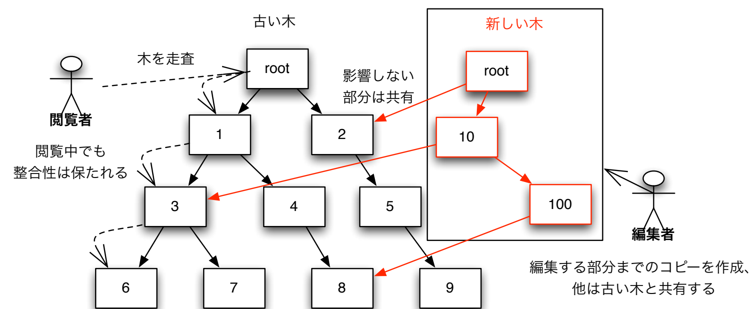

図 5　非破壊的木構造

4. 実　　装

本研究では，実装に Java を使用した．以下に実装した非破壊的木構造について説明する．

4.1　オンメモリな非破壊的木構造の実装

分散リポジトリの考え方を参考にし，非破壊的木構造を実装する上で以下のようなクラスを考えた．

- Node : データを保持するクラス
- NodeID : Node を識別する ID，UUID+Version 番号で構成される．
- Forest : すべての Node を含んだマップ
- Tree : ある Node をルートとする破壊的な木
- TreeEditor : 木構造を非破壊的に編集するエディタ

それぞれのクラスの役割に付いて説明する．

4.1.1　Node/NodeID

Node はデータを保持するクラスである．Node のインスタンスはユニークな NodeID を持ち UUID と Version 番号で構成されている．NodeID.UUID が一致する Node 同士は木構造上同じ位置に存在する Node であり，木の特定の Node を編集する際には木を捜査し同一の UUID を持つ Node を検索するために使用される．

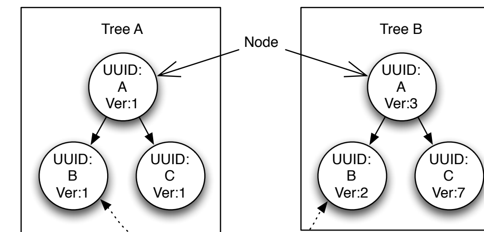

図 6　Node と NodeID の関係

© 2011 Information Processing Society of Japan

4.1.2 Forest

Forest は全 Node のマップで管理しており，NodeID を key として Node を返す．また，NodeID の UUID を key として同じ UUID を持つ Node の中で最新の Node を返す．CMS のコンテンツ全体の Tree と Node の作成，削除，取得はすべてこのクラスが管理している．あるデータ構造を用いて非破壊的木構造を実装する場合，この Forest が中心となる．よって，他のクラスはある程度使い回すことができ，基本的に Forest の実装だけを行えば良い．

4.1.3 Tree

ある Node をルートとする木構造，Forest には全体を表す Tree を持っているが，それとは別に任意の Node を Tree とすることが出来る．こうすることで全体の木を編集しなくても，部分的に木を編集することが出来るようになる．

4.1.4 TreeEditor

TreeEditor は Tree をメンバーとして保持し，木構造を非破壊的に編集する．メソッドに commit/check/update を持ち，1 回または複数回の編集が完了すると自身のルートを Tree に反映する．
もし，メンバーの Tree がすでに他の TreeEditor により編集されていた場合 commit は失敗し merge 処理を行う．

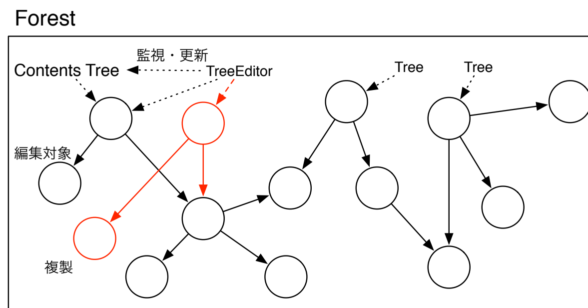

図 7 実装した非破壊的木構造の概要

4.2 Cassandra を使った非破壊的木構造の実装

非破壊的木構造を実装するため，Cassandra に以下のような KeySpace を定義した．

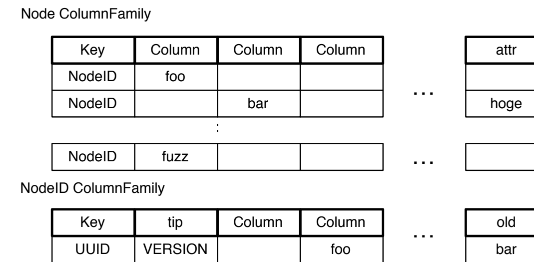

図 8 KeySpace の定義

Node ColumnFamily は NodeID を key として Column は Node の保持するデータを格納する．NodeID ColumnFamily は Node の UUID を key として Node の最新版の Version を格納する．Partitioner には key の分散を考慮して RandomPartitioner を使用している．

4.2.1 CassandraForest

CassandraForest は初めに Cassandra から最新版のノードの情報をすべて取得しメモリ上にキャッシュする．
オンメモリ上の実装と同様に，作成，削除，取得を管理する．そのため，内部には CassandraClient を保持している．Cassandra の検証より並列に負荷をかけることで，性能を発揮するという結果が得られた．よって，Java のスレッドプールフレームワーク (java.concurrent パッケージ) を用いてコネクションプールを用意し，Cassandra への操作を並列に行う．

Cassandra へのリクエストに対する返り値はすべて Future でやり取りされ，リクエストの結果は必要となったときにのみ同期され展開される．この様に実装することにより高い並列度が期待できる．

4.2.2 CassandraTreeEditor

TreeEditor では，編集する際に Node をそれぞれコピーするため，Cassandra へのリク

© 2011 Information Processing Society of Japan

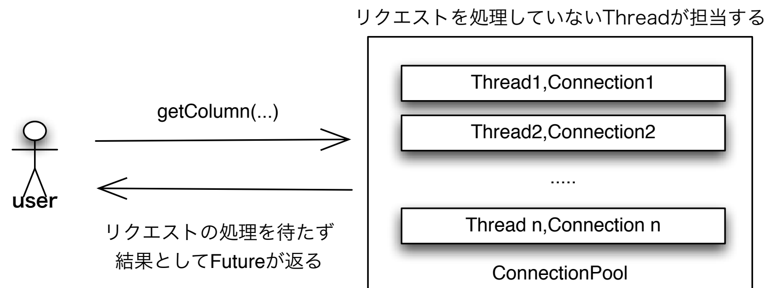

図 9 ConnectionPool

エストが複数発生する．しかし，実際にはコピーが終わったあとにまとめてコピーを行えば良い．Cassandra には `batch_mutate` という機能があり，複数の ColumnFamily と複数の Column に 1 度で変更を加えることができる．
CassandraTreeEditor では，NodeID ColumnFamily と Node ColumnFamily への操作をまとめてリクエストするため，Cassandra へ発生するリクエストの回数は非破壊的に編集した場合でも 1 回である．

5. まとめと今後の課題

本研究では，スケーラビリティのある CMS を開発するため，前回行った Cassandra の検証と検証環境を構築を元に設計を行った．スケーラビリティのある CMS の特徴として，「負荷がかかっても性能が低下しない・ノードの台数を増やすだけで性能を維持できる」と考え，実現方法として「データの複製を用意する・更新通知にはポーリングを利用する」を提案した．
この 2 つの方法を実現したものとして非破壊的木構造に分散リポジトリの要素を組み合わせたデータ構造を開発出来た．
今後は，開発したデータ構造の性能評価と改良を行う．

6. 謝　　辞

この研究は Symphony 社との共同研究「分散化されたテキスト管理システムに関する研究」によって行われました．Symphony 社の社員を始め，指導教官である河野真治先生には様々な助言や協力をして頂きました．ありがとうございました．

参　考　文　献

1) Shoshi TAMAKI,Shinji KONO:Cassandra を用いた CMS の PC クラスタを用いたスケーラビリティの検証, ソフトウェア科学会 (2010)
2) Giuseppe DeCandia, Deniz Hastorun, Madan Jampani, Gunavardhan Kakulapati , Avinash Lakshman, Alex Pilchin, Swaminathan Sivasubramanian, Peter Vosshall , Werner Vogels: Dynamo: Amazon's Highly Avaliable Key-value Store , SOSP (2007)
3) Matt Welsh: The Staged Event-Driven Architecture for Highly-Concurrent Server Applications
4) Matt Welsh, David Culler, Eric Brewer: SEDA : An Architecture for Well-Conditioned , Scalable Internet Services , SOSP (2001)
5) Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C. Hsieh, Deborah A. Wallach Mike Burrows, Tushar Chandra, Andrew Fikes, Robert E. Gruber: Bigtable : A Distributed Storege System for Structured Data

© 2011 Information Processing Society of Japan