クラウドコンピューティングにおけるデータアクセス方法と評価

M2010MM038　志村　惇

指導教員　青山　幹雄

1. はじめに

近年，クラウドコンピューティング(以下クラウド)が新しい情報システムの基盤として注目されている．コスト削減やコンピュータ資源の拡張性の高さから，基幹システムをクラウドに移行する企業が増加している．

オンプレミスのアプリケーションでは RDB(Relational DataBase)が主流である．しかし，アプリケーションの多様化から，クラウドのアプリケーションにおいても RDB の機能に対する要求が高まっている．

2. 研究の目的

RDB と KVS ではアクセスインタフェースおよびデータ構造に大きな違いがある，この 2 つ違いは RDB と KVS を統一的に扱うための障害となっている．本稿ではクライアントからデータストアに対するデータアクセスの差異を埋めることでアクセスインタフェースを統一し，クラウドのアプリケーションで RDB と KVS を統一的に扱えるようにする．

3. 研究課題

RDB と KVS など NoSQL では扱うデータ構造が異なる．データ構造との違いから，アプリケーションで両データベースを利用する場合，それぞれのデータベースのデータ構造に適したデータを容易に利用できる必要がある．

3.1. データモデルの違い

RDB と KVS では異なるデータモデルを利用している．RDB ではデーブル毎に定義されたスキーマを持ち，定義されたスキーマに基づいてデータの操作を行う．

KVS ではキー項目に対してバリュー項目のみの単純なハッシュテーブルに基づくデータ構造である，またバリュー項目で扱われるデータは，レコード毎に異なる(図 1).

図 1 データ構造の違い

3.2. データアクセス方法の違い

RDB と KVS ではデータベースにアクセスするための方法が異なる(図 2).

RDB ではデータベースのデータ操作をするために SQL を利用し，複雑な条件によるデータ操作が可能である．それに対して KVS では SQL はサポートしておらず，KVS には独自のデータ操作技術が存在する．

また，KVS には標準のデータアクセス方法が無く，それぞれの DBMS 毎に用意されたアクセス方法を利用する必要がある．

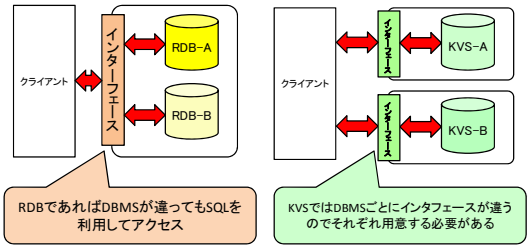

図 2 インタフェースの違い

4. アプローチ

4.1. データアクセス方法の分類

SQL と NoSQL の差異に着目し，表1に示す 3 つのデータアクセス方法を対象にデータストアへのアクセス方法を分類，比較する．各アクセス方法の振る舞いを図 3 に示す．

表 1 データアクセス機能

モデル	SQL	NoSQL	NoSQL
実装	MySQL	MySQL	Cassandra
Connection	getConnection	openIndex	Open
Read	Select	Find	Get
Insert	Insert into	Insert	Set
Update	Update	Update	Set
Delete	Delete	Delete	Del

4.2. MySQL

RDB は通常，SQL を利用してデータの操作を行う．そのためにサーバ側ではデータを整形し SQL のクエリを生成する必要がある．

SQL を利用することでデータストアに対して複雑なクリエの実行，テーブルをまたいだデータ操作が可能である．

4.3. HandlerSocket

MySQL に対して NoSQL と同様のデータアクセスを提供する API である. 扱いたいデータを API に引き数として渡すことで MySQL にアクセスできる.

4.4. Cassandra

NoSQL データベースとして Cassandra がある.

Cassandra ではデータ構造として, キーデータとバリューデータのみの単純なデータ構造でキーデータによってバリューデータを呼び出しデータの操作を行う.

Cassandra では SQL をサポートしておらず, データを操作するには特定の API に対して引き数としてキーのデータを渡しバリューデータを操作する.

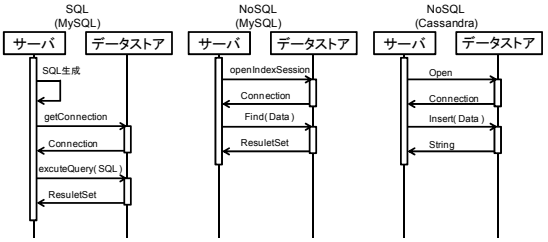

図 3 データアクセスの振る舞い

4.5. **データストアの隠蔽**

クライアントからインタフェースを統一にすることでデータストアを隠蔽しクライアントから実装を意識させない.

クライアント側はデータストアへのアクセスロジックを持たず, データストアが提供するサービスを呼び出す. また, データアクセス機能を Web サービスとして公開し, クライアントはサービスを利用することでデータアクセスができる.

5. データアクセス方法

5.1. データアクセスサービスのアーキテクチャ

SQL と NoSQL に対する, データアクセスを Web サービスとして提供するシステムのアーキテクチャを図 4 に示す.

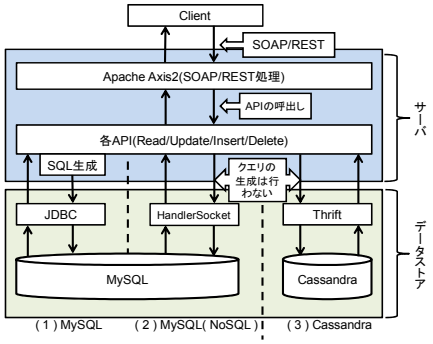

図 4 システムのアーキテクチャ

5.2. クライアントと Web サーバ間

クライアントとサーバ間はアクセスするデータストアに関わらず, 統一的な手法をとる. Web サービス標準のメッセージングプロトコルである SOAP/REST を用いる. クライアントのメッセージにはサーバで行うデータアクセスに必要なデータと利用するサービスの呼び出しと呼び出したサービスに基づいたデータを記載しサーバに対して送信する. 送信メッセージは ApacheAxis2 により処理される.

サーバからデータストアへのアクセス後はクエリの実行結果のレスポンスとして XML 形式で送信する.

5.3. Web サーバとデータストア間

5.3.1. SQL(MySQL)

SQL を用いた MySQL へのアクセスはサーバ側がクライアントからのメッセージに基づいて SQL を生成し JDBC を介して MySQL へのアクセス行う. データの操作は生成された SQL のシンタックスに依存する.

5.3.2. NoSQL(HandlerSocket)

MySQL に対して NoSQL でデータアクセスを行うには HandlerSocket を用いる. クライアントのメッセージに基づいて API を呼び出し CRUD の実行を行うがサーバ側で明示的にクエリの生成は行わず表 1 に示したデータアクセス機能を利用してデータの操作を行う.

5.3.3. NoSQL(Cassandra)

Cassandra に対するデータアクセスはクライアントのメッセージに基づいて呼び出された API に対して引き数としてクライアントからのデータを渡す. Cassandra は SQL をサポートしないのでサーバは Thrift インタフェースを介して CRUD の実行を行う.

6. プロトタイプ

図 4 に示したデータアクセスサービスのアーキテクチャをプロトタイプとして実装を行う.

クライアントから SOAP によるメッセージングを行いデータストアの機能を呼び出し, サーバ側でメッセージを処理し, 3 つのデータストアに対するアクセスの性能を測定する.

6.1. 実装環境

HandlerSocket は Linux 上で実行するため, Ubuntu を OS として環境を実装した.

クライアントの実装は MySQL と Cassandra のインタフェースとして一般的な JDBC および Thrift を親和性が高く, 実装のコストが低い Java を利用する(図 5).

モデル		SQL	NoSQL	NoSQL
クライアント	App	Java 1.6		
	OS	Ubuntu 10.04.3		
サーバ	アクセス	JDBC 5.1	Handler Socket 1.1	Thift 0.8
	DB	MySQL 5.1		Cassandra 0.6
	OS	Ubuntu 10.04.3		

図 5 プロトタイプの実装環境

6.2. 実装方法

Java で構成されたサーバ側のプログラムから各データストアに対して表 2 に示すデータの登録を 100 回行い，サーバプログラムがデータストアへコネクションのオープンを開始した時間からレスポンスを受け取るまでの時間を測定した．100 件の実験結果の平均と標準偏差を表 3 に示す．また，平均と標準偏差の算出では，異常なデータを 29 件除外した．

6.3. 計測データ

Java で作成したクライアントのアプリケーションから，表 2 に示すデータを利用してデータストアに対してデータアクセスを行う．

表 2　登録データ

変数	id	name
データ型	String	String
サイズ	1byte	15byte
データ内容	1 桁の数字	アルファベット 15 文字

6.4. 実験結果

実装結果で得られたデータのグラフを図 6 に示す．計測の試行回数はそれぞれ 100 回であるが，取得データ中の平均から 10 倍のデータは異常なデータとして除外し，それ以外の 71 件のデータを本稿では利用する(図 7)．

(1)　SQL の結果

SQL の結果は計測した方法の中で一貫してデータが安定せず，結果のデータは上下に不安定なデータとなった．

(2)　NoSQL(HandlerSocket)

NoSQL(HandlerSocket)の計測結果は終始，安定した計測結果になっており，SQL を利用する場合と差が生じた．

(3)　NoSQL(Cassandra)

NoSQL(Cassandra)の結果は全体をとおして，安定した結果となった．SQL のように変動することはなく，一定の水準であった．

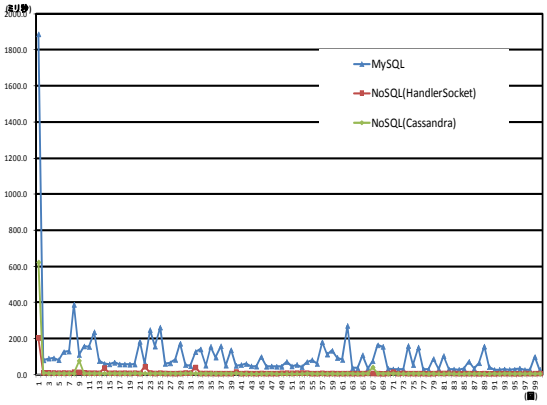

図 6 データ登録実行時間(全測定値)

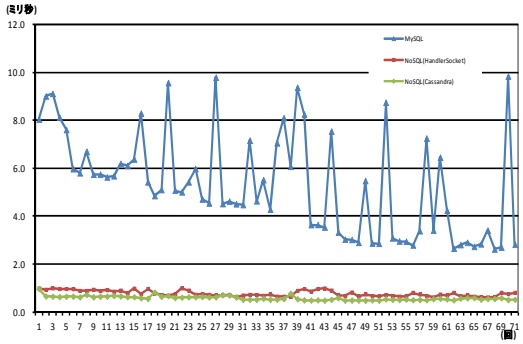

図 7　データ登録実行時間(異常値を除外)

6.5. 計測結果の統計

計測結果の平均と標準偏差を表 3 に示す．

SQL と NoSQL ではデータアクセスの速度に大きな差が発生した．平均としては SQL が NoSQL の 7 倍の時間を要した．

また，SQL はデータのばらつきが大きく，アクセス時間が不安定であった．

表 3　計測結果の統計

モデル	SQL	NoSQL	NoSQL
DB	MySQL	MySQL	Cassandra
平均	5.3	0.8	0.6
標準偏差	2.1	0.1	0.1

7. 評価

7.1. データアクセス方法の差異

MySQL に対して HandlerSocket を利用し，データアクセスする場合は SQL の生成を必要とせず，API を呼び出し引き数にデータを渡すため，Cassandra のデータアクセスと大きな差はない．

HandlerSocket を利用する場合，データストア自体は MySQL を利用するが RDB の特徴である以下の機能が利用できない．

(1)　複雑な条件でのデータ操作
(2)　トランザクション
(3)　テーブルを越えたデータ操作(JOIN)

7.2. データアクセス性能

HandlerSocket を利用して MySQL にアクセスする手法は，Cassandra にアクセスする場合と比較しても，性能は同等であった．

MySQL の場合ではアクセス時間が一定せず全体を通して不安定なデータとなった，一方で HandlerSocket を経由する場合で SQL を利用する場合より高速で，なおかつ 0.8 ミリ秒の前後で推移し全体的に安定した性能を発揮した．

Cassandra に対してデータアクセスする場合は 3 つの方

法中では最も高い性能を見せた，HandlerSocket と同様に全体的に安定したデータとなった．

7.3. HandlerSocket の SQL との併用

HandlerSocket は NoSQL で MySQL に対してデータアクセスするためのインタフェースだが，HandlerSocket を実装していても既存の SQL を利用したデータアクセスは可能である(図 8)．

アプリケーションは MySQL に対して SQL と NoSQL を利用したデータアクセスが同時に可能となる，これによって高速処理を行いたいデータアクセスは NoSQL，複雑なクエリによるデータアクセスは SQL を用いるといったデータや状況に応じたインタフェースの使い分けが可能となる．

HandlerSocket を利用したインタフェースの使い分けはアプリケーションの効率的なデータアクセスを実現し高性能なアプリケーションの開発を可能にする．

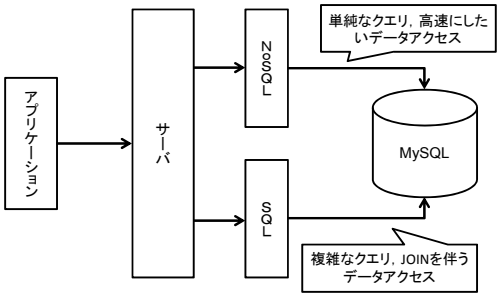

図 8 MySQL へのアクセス

7.4. HandlerSocket の Cassandra との併用

HandlerSocket を KVS と同時に利用する場合，扱うデータの差別化が難しいが，扱うデータの特性を考慮し，それぞれの住み分けはできる．扱うデータは表 4 のように分類できる．

表 4 扱うデータの分類

インタフェース	SQL	NoSQL	NoSQL
DB	MySQL	MySQL	Cassandra
データ	更新系	更新系	参照系
クエリ	複雑	単純	単純
速度	低速	高速	高速

7.5. クライアントとデータストアの関係

本稿における，データストアアクセスのアーキテクチャを利用することで密結合であったアプリケーションとデータストアの関係を疎結合としクライアントのアプリケーションはデータアクセス機能が定義された Web サービスを呼び出すことでデータストアへのアクセスが可能となる．

クライアントとデータストア間が疎結合となることでシステムはデータストアを中心にアプリケーションを動的に変更することが容易になる．アプリケーションが変わる場合，それまで利用してきた格納データに対して，データアクセス API を利用するだけでデータアクセスを可能にする．

8. 今後の課題

8.1. 動的アクセス

本稿では，サーバからデータストアへの動的データアクセスのプロセスが考慮されていない．データストアの実装を隠蔽し，統一的に扱うためにクライアントはメッセージ内容にデータアクセス先を明示しないのでサーバ側で動的にアクセス先を選択する必要がある．そのためにサーバ側でデータアクセスを選択する方法を考慮する必要がある．

8.2. スキーマの整合

本稿では，クライアントのアプリケーションとデータストアのスキーマの整合がとれているという前提でプロトタイプの実装を行った．クライアントとデータストアで扱うデータのスキーマ変更があった場合にスキーマの整合をとる方法を考慮する必要がある．

9. まとめ

本稿ではクラウドコンピューティング環境におけるデータアクセスの方法として MySQL に対して NoSQL によるデータアクセスの方法と従来の SQL を利用してデータアクセス方法，さらに KVS である Cassandra に対するデータアクセスの 3 点を比較し，その性能とそれぞれに適してデータの提示を行った．

HandlerSocket は MySQL への単純なデータアクセスを Cassandra 並の高速なアクセスを可能にし，アプリケーションのデータアクセスの効率化を向上させるが，その代わりに RDB の特徴である，トランザクションの処理，複雑なクエリの実行，JOIN を利用したテーブルの結合などの機能は利用できなくなる．

クラウドコンピューティングでは RDB と KVS の統一的利用が求められる．HandlerSocket を利用して RDB に NoSQL でデータアクセスを利用することでインタフェースを共用することが可能となる．

参考文献

[1] Apache Software Foundation, Cassandra, http://cassandra.apache.org/.

[2] Apache Software Foundation, Apache Axis2, http://axis.apache.org/axis2/java/core.

[3] E. Hewitt, et al., Cassandra: The Definitive Guide, O'Reilly, 2011 [大谷 晋平 他, (訳), オライリージャパン, Cassandra. 2011].

[4] 佐々木 達也, NoSQL データベースファーストガイド, 秀和システム, 2011.