スキーマ定義に基づくSQLライクなKey-Valueストアクライアント

善明　晃由† 　津田　 均†

† 株式会社サイバーエージェント
〒101–0021 東京都千代田区外神田１－１８－１３ 秋葉原ダイビル８階８０２号室
E-mail: †{zenmyo_teruyoshi,tsuda_hitoshi}@cyberagent.co.jp

あらまし　本稿では，Key-Value ストア (KVS) を用いたアプリケーション開発を簡素化するクライアントライブラリを提案する．KVS を用いる場合，アプリケーションのデータや問い合わせを Key-Value の形式に変換する処理が必要となる．Key-Value への変換はアクセスパターンに適したスキーマで行う必要があり，アプリケーション毎に異なる．また，アクセスパターンの追加や変更の際には変換処理の修正が必要となる．このため，KVS を用いたアプリケーション開発では，変換処理の実装や保守が課題となる．提案するクライアントライブラリは事前に定義されたスキーマに基づきアプリケーションデータと Key-Value の間の変換を自動化することで，様々なスキーマの Key-Value に SQL ライクな言語でアクセスすることを可能にする．これにより，KVS に関する知識がない開発者でも KVS を用いたアプリケーション開発が可能となる．本稿では，提案するクライアントライブラリの概要と，当社で開発運用している HBase を用いたデータ集計アプリケーションへの適用例について述べる．

キーワード　Key-Value ストア, SQL

1. はじめに

大規模データ処理を行うシステムにおいて，Key-Value ストア (KVS) [3], [5] の活用が重要になっている．KVS は Key と Value の組み合わせ（Key-Value）を管理するデータストアである．Key に基づくデータの読み書きのためのインタフェースを提供する．Key が複数のフィールドに分割されより構造化されている KVS も存在するが，この場合も同様にデータの操作は Key に基づいて行われる．KVS はデータ構造やインタフェースは簡潔なものしか提供しないが，スケーラビリティや可用性を実現している．

KVS を用いたアプリケーションを開発する際は，アプリケーションのデータと Key-Value の変換処理が必要となる．例えば，複合キーをもつレコードを KVS に管理することを考える．この場合，書き込み時に複合キーを構成する複数の属性を Key に，例えば連結するなどの方法で集約し，読み込み時に集約された属性を分割する必要がある．

アプリケーションデータと Key-Value の変換方法（本稿では，スキーマと呼ぶ）は，アプリケーションの性能にも影響を与える．例えば，複合キーを構成する属性（$a1, a2$ とする）を連結する場合，$a1 - a2$ という順序の場合は $a1$ を固定した問い合わせ処理は効率的に処理できるが，$a2$ のみを固定した問い合わせを処理するには全てのデータを走査する必要がある．また，属性が多いレコードについては，全てのレコードを連結して一つの Key-Value に変換する方法や，属性ごとに別の Key-Value に分割する方法がある．後者は，属性毎の読み書きが多い場合は有効であるが，Key の値が重複するためデータ容量が多くなってしまうという短所がある．このように，KVS を利用する際は，アプリケーションの要件に基づいてスキーマを適切に設計することが重要となる．

適切なスキーマは，アプリケーションのデータ構造や問い合わせパターンによって異なるため，Key-Value との変換処理はアプリケーション毎に実装する必要がある．また，アプリケーションに対する要件の変更に対して，スキーマを変更が必要となることもあるため，変換処理の保守も KVS を利用する際の課題となる．

KVS のデータを効率的に管理するために，Apache Phoenix [1] など SQL ライクな言語で Key-Value を扱うためのツールが開発されている．しかし，これらのツールはスキーマを限定しており，ツールの提供するスキーマに合わないものに適用することが困難である．

本稿では，スキーマ定義に基づいて変換処理を自動化することで，多様なスキーマに対応可能な KVS のクライアントライブラリを提案する．Key-Value との変換処理における課題として，アプリケーションデータや問い合わせ処理と Key-Value や KVS に対する操作が一対一対応しないことがある．提案手法では，スキーマ定義に基づきアプリケーションデータや問い合わせ処理を，Key-Value のフィールドでラベル付けされて木構造の中間形式に変換し，Key-Value や KVS に対する操作を導出する．また，HBase を対象とした提案手法の実装と，当社で開発運用しているデータ解析アプリケーションへの適用例について説明する．

2. Key-Value ストアと利用における課題

本節では，Key-Value ストアの例として Apache HBase(HBase) のデータ構造やインタフェースについて説明し，その利用における課題について議論する．

2.1 HBase の概要

HBase は Bigtable に基づいて開発されたオープンソースの分散 Key-Value ストアである．HBase で管理されるデータは

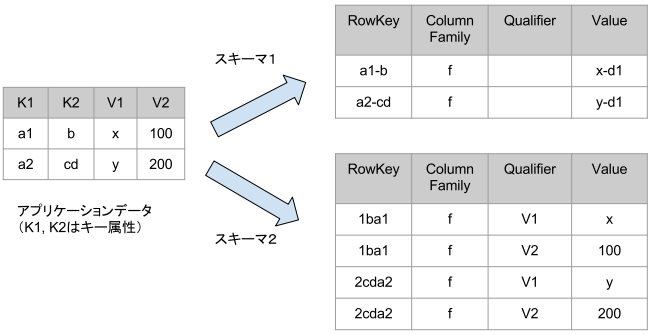

図 1　Key-Value への変換例

Key でソートされ，リージョンと呼ばれる単位に分割される．各リージョンはリージョンサーバで管理され，HBase ではリージョンサーバを追加することでデータ量の増加に対してスケールさせることが可能である．

HBase で管理される Key-Value は，Rowkey，Column Family，Qualifier，Value の 4 つのフィールドを持つ (注1)．このうち，Rowkey，Column Family，Qualifier は Key を構成するフィールドである．Rowkey は Key の先頭に位置するフィールドであり，アトミックに操作可能なデータの単位を定義する．同じ Rowkey をもつ KeyValue の集合は Row と呼ばれ，データのリージョンへの分割も Row を基に行われる．Column Family は同時にアクセスされるデータをまとめるためのフィールドであり，同じ Rowkey を持つデータであっても Column Family 毎に別のファイルで管理される．Qualifier は Key の末尾に位置するフィールドである．

HBase におけるデータの操作は下記のインタフェースを介して行う．(注2)．

- Get，および，Scan

Key を指定してデータを読み取る．Get では一つの Rowkey を指定し，Scan では Rowkey の範囲を指定する．また，対象とする Column Family や Qualifier やフィルタを定義することができる．

- Put，および，Delete

Row や KeyValue の書き込みや削除を行う．

2.2　KVS におけるスキーマ設計

KVS を利用する際には，アプリケーションデータの Key-Value への変換方法の設計が重要となる．本稿ではこの変換方法をスキーマと呼ぶ．例えば，HBase のデータは，Rowkey でソート，分割されるためその設計が重要となる．また，Column Family や Qualifier についても性能やインタフェースの利用法に影響を与えるため注意する必要がある．

図 1 を用いてスキーマの設計について説明する．図 1 では，１種類のアプリケーションにデータに対して以下の２つの スキーマを例示している．

- スキーマ１

キー属性と非キー属性をそれぞれ，セパレータ文字 (-) で連結し，Rowkey と Value に設定する．

- スキーマ２

文字数をプレフィックスとして追加した K2（1b，2cd）と K1 を連結したものを Rowkey に設定し，非キー属性は属性名を Qualifier に設定し，その Value に属性値を設定する．

スキーマ 1 は，K1 が Rowkey の先頭に配置されているため，K1 を指定した問い合わせに強く，先頭部分が固定された正規表現の問い合わせも効率的に処理できる．しかしながら，セパレータ文字列が K1 の値に含まれる場合にはそのまま利用できず，別途エスケープ処理などを行う必要がある．これに対してスキーマ 2 では，文字数をプレフィックスとして追加することで K1 と K2 を区別するため，利用できる文字に制限がない．一方，文字数が Rowkey の先頭に文字数が配置されているため，K2 を完全に指定した問い合わせでなければ効率的に処理できない．

また，スキーマ 1 では，アプリケーションの１レコードが一つの Key-Value に変換されているため，データの重複がなく空間効率がよい．しかし，複数の属性がまとめて Value に配置されているため，不必要なデータの読み取りが必要となることもあり，常に集約されたデータの分割処理が必要になる．さらに Value に複数の値が固定的に構造化されているためアプリケーションレベルでの属性の追加等に対応しにくいという問題もある．一方，スキーマ 2 では，属性ごとに Key-Value を分けているため，属性を指定した読み書きやアプリケーションレベルでの属性の追加等に柔軟に対応できる．しかし，複数の Key-Value で同じ Rowkey が重複することや， Row あたりのデータサイズが大きくなるとリージョンの分割がしにくくなるという問題がある．

2.2.1　KVS 利用における課題

前述のようにアプリケーションデータを Key-Value に変換する際には，様々な選択肢があり，それらを組み合わせてアプリケーションの要件に合うスキーマを設計する必要がある．また，アプリケーションの要件によっては，複数のスキーマを使う必要がある場合もある．例えば，図 1 の例で，K1 を固定した問い合わせと K2 を指定した問い合わせが同程度ある場合は，K1 と K2 をそれぞれ Rowkey の先頭に設定した二つのスキーマを用意することが有効である．

HBase を利用したアプリケーションを開発する場合は，図 1 のような変換処理を実装する必要がある．一般に変換処理は，アプリケーションの機能やアクセスパターンなどに応じて複数の処理を実装する必要がある．スキーマに変更があった際は，すべての変換処理を修正する必要がある．このため HBase を利用したアプリケーションでは，スキーマを管理し，変換処理を適切に保守していく必要がある．

変換処理の実装に不整合がある場合は，データのアクセスに不整合が発生する．例えば，書き込み処理で行った修正を読み込み処理で対応しなかった場合は，データの読み込み不可や誤ったデータの読み取りなどが発生する．このような状況を回

(注1)：他にタイムスタンプや各フィールドのサイズ等のフィールドもあるが提案手法ではあつかわないため省略する．

(注2)：他にもバッチ処理や，Compare And Swap の操作などがあるが本稿では省略する．

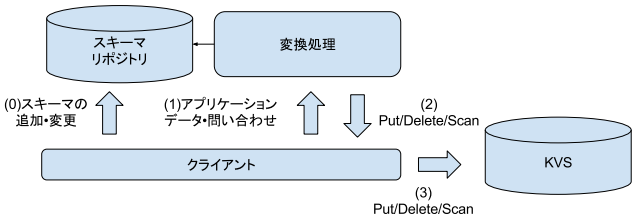

図 2　提案手法の概要

避するためには，スキーマとその変換処理を適切に管理する枠組みが必要である．

3. 提 案 手 法

図 2 に提案手法の概要を示す．提案手法では，一元的に管理されたスキーマ定義に基づきアプリケーションデータや問い合わせを KVS への読み書き操作 (Put，Delete，Scan) に変換する．スキーマが一元的に管理されそれに基づいて変換処理がおこなわれるため，KVS を利用する際の変換処理の実装や保守を効率化することができる．

3.1 スキーマモデル

提案手法では関係モデルに倣って，アプリケーションレベルのレコードをタプル，およびその定義をリレーションと呼ぶことにする．リレーションは一つ以上の属性で定義され，タプルは各属性の値を持つ．スキーマはリレーション毎に一つ以上定義され，KVS のフィールド (HBase での Rowkey，Column Family など) への属性の配置方法を定義する．

例えば，図 1 の例においてリレーションは，

$$R = (K1, K2, V1, V2)$$

となり二つのスキーマはそれぞれ，以下のように表現できる．なお，各フィールドは，リスト ([]) で表現し，リストの要素を順に並べたものを各フィールドの値とする．また，$R.K1$ などは各属性の値，$'f'$ は定数，$length(R2.K2)$ は R2.K2 の値の文字数，$R[qualifier]$ は，同じ Key-Value の Qualifier で指定される属性の値とする．

- スキーマ 1

$$\begin{aligned} Rowkey &= [R.K1, '-', R.K2] \\ ColumnFamily &= ['f'] \\ Qualifier &= [''] \\ Value &= [R.V1, '-', R.V2] \end{aligned}$$

- スキーマ 2

$$\begin{aligned} Rowkey &= [length(R.K2), R.K2, R.K1] \\ ColumnFamily &= ['f'] \\ Qualifier &= ['V1'] or ['V2'] \\ Value &= [R[qualifier]] \end{aligned}$$

3.2 スキーマ定義に基づく変換処理

アプリケーションデータを Key-Value に変換においては，図 1 のスキーマ 2 のように一つのタプルが複数の Key-Value に変換されうることを考慮し，木構造の中間形式を導入する．

木構造への変換は，スキーマの各フィールドの要素を順に適用していき，各要素に対応するノードは先行する要素に対応するノードを子として追加していく．

図 3 は，図 1 のスキーマ 2 を例に，変換の流れを示したものである．まず，スキーマ定義の Rowkey の最初の要素が，$length(R.K2)$ なので属性 $K2$ の値の文字数 (1) を根に設定する．続く要素は $R.K2$ なので，属性 $K2$ の値 (b) をその子ノードとして追加する．これを Rowkey，Column Family，Qualifier，Value の順に行う．Qualifier の要素は，$['V1']$ または $['V2']$ の 2 通りがあるため 2 つのノードを追加し，Value については親ノードの属性に対応する値を設定する．

木構造の中間形式においては，根から葉への各パスを Key-Value とみなすことができる．例えば，図 3 では，2 つのパスによりこのスキーマでは対象のアプリケーションデータは 2 つの Key-Value に変換されることを示している．

問い合わせの変換も同様に木構造を介して行う．なお，現在，提案手法では，現在，各属性値を定数と比較する条件 (`K1 ='a1'` や `V2 > 100`) のみ対象としており，複数の属性を比較する条件 (`K1 = K2`) などは非対応である．

問い合わせ変換の例を図 4 に示す。問い合わせの変換は，属性毎に条件をまとめたものを木のノードに設定する．図 4 の問い合わせでは，まず K2 の値が固定されてるため (`K1= 'b'`)，スキーマ定義にあわせて，根から，(`=1`) と (`='b'`) をノードとして設定する．スキーマにおける次の要素 (K1) に対しては 2 つの条件が指定されておりその両方 (`>='a0'` or `< 'a2'`) をノードに設定する．このようにして構築された木構造を根から参照することで，Key-Value のフィールドにおいて固定されている範囲を決定することができ，例えば図 4 では，Rowkey が $1ba0$ から $1ba2$ の範囲をスキャンすればよいことが分かる．

複雑な条件式については，変換を単純にするために条件式を選言標準形 (Disjunctive normal form, DNF) に変形する．これにより，各連言毎に変換に行い，最終的な結果は各連言の結果をマージすることで構成する．

また，複数のスキーマをもつリレーションへの問い合わせは，スキーマの前方が固定されているほど良いと判定されるスコア

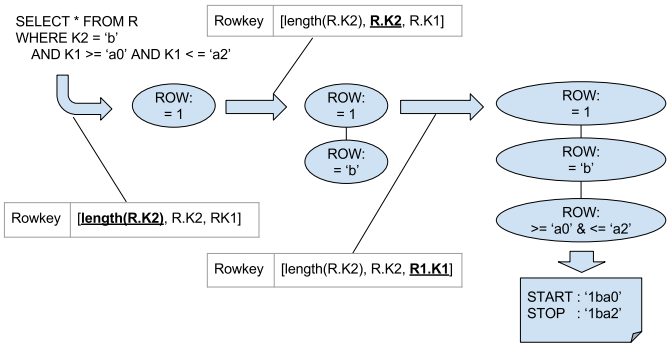

図 4　問い合わせの変換

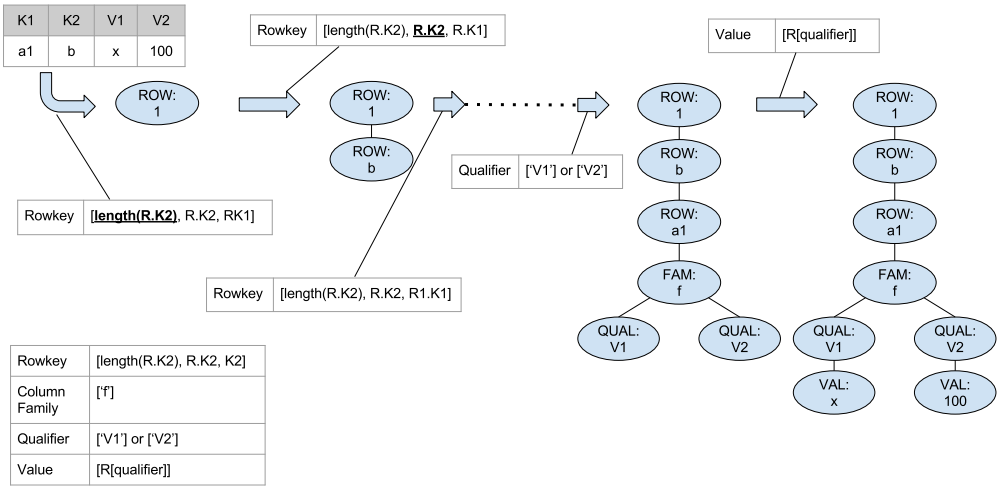

図 3　木構造への変換

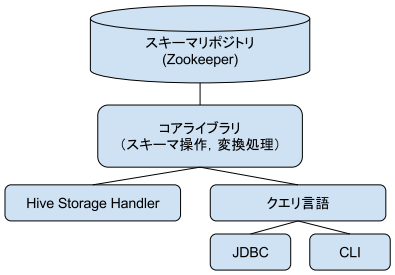

図 5　HBase 向けの実装

付けを行い，スコアに基づいて利用するスキーマを選択する．

4. HBase を対象とした実装

本節では，提案手法を HBase を対象とした実装について説明する．

図 5 に HBase 向けの実装の概要を示す．この実装では，以下の 3 種類のクライアントを提供する．

- Hive Storage Handler
- JDBC ドライバ
- コマンドラインインタフェース (CLI)

Hive Storage Handler は Hive 経由で HBase のデータへアクセスする機能を提供する．また，JDBC ドライバと CLI は，SQL ライクな独自のクエリ言語で HBase へのアクセスを可能とする．スキーマリポジトリとしては，Zookeeper を用いスキーマの追加や変更が複数のクライアントで同期されるようにしている．

スキーマ定義の例を図 6 に示す．図 6 では DEFINE RELATION 文でリレーションのもつ属性などの定義を行い，DEFINE SCHEMA 文を用いてスキーマを定義している．スキーマでは HBase の Key-Value の各フィールドごとに配置する属性を :で連結している．なお，複数のテーブルにデータを分割できるようにテーブル名もフィールドとして扱っている．各属性を分割するためのプレフィックスやサフィックスは，`suffix` や

```
DEFINE RELATION r {
  k1 STRING key,
  k2 STRING key,
  v1 STRING,
  v2 INT
};

DEFINE PRIMARY SCHEMA s1 FOR r {
  TABLE 's1',
  ROW suffix('-'){k1}:k2,
  FAMILY 'f',
  QUALIFIER '',
  VALUE suffix('-'){v1}:v2
};

DEFINE SCHEMA s2 FOR r {
  TABLE 's2',
  ROW size{k2}:k1,
  FAMILY 'f',
  QUALIFIER attr_name['k1','k2'],
  VALUE attr_value['k1', 'k2']
};
```

図 6　スキーマ定義の例

`size` 等の修飾子を用いて定義する．例えば，図 6 において，`row suffix('-'){k1}:k2` は，`k1` の値の末尾に-を追加して `k2` の値と連結したものを rowkey とすることを意味する．また，`attr_value['k1','k2']` は先行する `attr_name['k1','k2']` に対応する属性値を示す．ここで `attr_value['k1','k2']`, `attr_name['k1','k2']` は `k1`, `k2` 以外の属性名を表すため、実際の処理は `v1`, `v2` が対象となる．

上記のように定義したスキーマを用いて，以下のように HBase にアクセスすることができる．各ステートメントの下には対応する HBase への操作が記載されている．

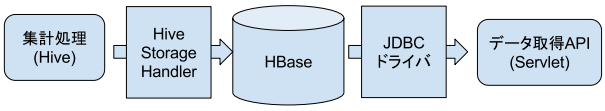

図 7　配信集計システム

```
INSERT INTO r VALUES (’a1’, ’b’, ’x’, 100);
# put "s1", "a1-b", "f:", "x-\x00\x00\x00d"
# put "s2", "\0x1ba1", "f:V1", "x"
# put "s2", "\0x1ba1", "f:V2", "\x00\x00\x00d"
SELECT * FROM r WHERE k1=’a1’ AND k2=’b’;
# scan ’s1’, {STARTROW=>’a1-b’, STOPROW=>’a1-c’}
```

5. データ集計アプリケーションへの適用

本節では，提案手法の広告配信の集計システムへの適用について述べる．対象とする集計システムは，広告配信の効果を測定するために，広告主や配信枠（ヘッダーやフッターなど）様々な軸でインプレッション数やクリック数などの集計を行う．

配信集計システムの概要を図 7 に示す．このシステムは，Hadoop を用いて構築されたデータ解析基盤 [13] 上で，Hive を用いて集計をおこなった結果を Hive Storage Handler を介して HBase に書き込む．書き込まれた集計結果は，JDBC ドライバを用いて読み取られ配信担当者へのレポート作成等に利用される．集計結果取得 API には，プリペアードステートメントの形式で JDBC ドライバ経由で送信するクエリが設定可能である．本集計システムでは，2016 年 2 月時点で一日当たり約 10 億件のログレコードをもとに約 11 万行の集計結果を HBase に書き込んでいる．

図 7 のシステムでは，HBase への書き込みと読み込みでクライアントが異なるが，提案手法によりスキーマが一元管理されるため，スキーマを定義することで，Hive のテーブルとクエリや JDBC ドライバに設定するクエリを記述するだけでシステムを実現している．

集計システムの稼働開始直後，データ取得 API サーバには集計対象日と配信担当者が設定したグループ（アドグループ）ごとに集計結果を確認することを想定し，集計対象日とアドグループを Rowkey の先頭に配置したスキーマを用いていた．しかしながら，運用の中でアドグループを指定せず配信枠毎に集計結果を参照する要件が高まりレイテンシの悪化（1 アクセスあたり 1 分程度）が問題となった．これは Rowkey の先頭に配置したアドグループがクエリで指定されない場合，集計対象日の集計結果を全てスキャンする必要があるためである．

この要件の変化に対する対応も提案手法により容易に対応可能であった．上記の変化に対しては，新しいスキーマ（集計対象日と配信枠の識別子を Rowkey の先頭に配置したもの）を追加し，集計結果 API に対するクエリを変更するのみでレイテンシを 1 秒以下に改善できた．

6. 今後の課題

今後の課題として，複数スキーマを利用する際の一貫性の確保がある．現在，複数スキーマでデータを書き込む際の一貫性の保証はできておらず，なんらかのエラーがあった際は，一部のスキーマでデータのロストが発生する可能性がある．Key-Value ストアに対して一貫性のある更新を行う手法としては，Percolator [9] などで採用されているタイムスタンプを用いる方法などが提案されている．また，結果整合性が保証されていれば厳密な一貫性は必要ない場合も多く [11]，今後，要件に応じて拡張を検討していく必要がある．

複数のリレーションの結合などの複雑なクエリへの対応も今後の課題の一つである．提案手法はスキーマを基に原始的な操作やレコードの変換を行うものであり，結合処理方式などは対象としていないため，MapReduce [4] や Spark [12] などの並列処理フレームワークとの連携も含めて検討する必要があると考える．

また，今後の拡張として異なるデータストアの統合への応用も検討している．様々なデータソースの統合は Business Intelligence(BI) などの分野で重要となっている [6], [7], [10] ．提案手法におけるスキーマは，例えば，HBase のテーブルまで含めて Key-Value とみなす，などデータストアが提供するデータ構造を柔軟にあつかうことが可能であり，様々なデータストアを連携させる際にも有効と考える．

提案手法では，一つのアプリケーションレコードを複数の Key-Value に対応させるスキーマについては対応しているが，複数のアプリケーションレコードを一つの Key-Value に対応させるスキーマには対応していない．このようなスキーマは，Time Series Database (TSDB) [2], [8] などで有効である．このようなスキーマについては，リストなどのデータ構造を用いて，集約されたレコードを一つのレコードとみなすようなリレーションを定義し，挿入を更新（リストへの挿入）とみなすことで対応することを検討している．

7. ま と め

本稿では，スキーマ定義にもとづいて，アプリケーションデータを Key-Value に変換する手法を提案した．また，Apache HBase を対象とした実装を行い，当社で開発運用しているデータ集計アプリケーションへの適用を行った．提案手法を用いることで，HBase のデータ構造を操作するコードを記述することなくデータ集計アプリケーションを開発し，アクセスパターンの変更にも容易に対応できることを確認した．今後は，一貫性の確保や複雑なクエリへの対応を行い，データの有効活用につなげていく予定である．

文　献

[1] Apache Phoenix. `http://phoenix.apache.org/`.
[2] OpenTSDB. `http://opentsdb.net/`.
[3] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. E. Gruber. Bigtable: A distributed storage system for structured data. In *Proceedings of the 7th USENIX Symposium on Operating Systems Design and Implementation - Volume 7*, OSDI ’06, pages 15–15, Berkeley, CA, USA, 2006. USENIX Association.

[4] J. Dean and S. Ghemawat. Mapreduce: Simplified data processing on large clusters. In *Proceedings of the 6th Conference on Symposium on Opearting Systems Design & Implementation - Volume 6*, OSDI'04, pages 10–10. USENIX Association, 2004.

[5] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels. Dynamo: Amazon's highly available key-value store. In *Proceedings of Twenty-first ACM SIGOPS Symposium on Operating Systems Principles*, SOSP '07, pages 205–220, New York, NY, USA, 2007. ACM.

[6] H. Elmeleegy, Y. Li, Y. Qi, P. Wilmot, M. Wu, S. Kolay, A. Dasdan, and S. Chen. Overview of turn data management platform for digital advertising. *Proc. VLDB Endow.*, 6(11):1138–1149, Aug. 2013.

[7] K. Morton, R. Bunker, J. Mackinlay, R. Morton, and C. Stolte. Dynamic workload driven data integration in tableau. In *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*, SIGMOD '12, pages 807–816. ACM, 2012.

[8] T. Pelkonen, S. Franklin, J. Teller, P. Cavallaro, Q. Huang, J. Meza, and K. Veeraraghavan. Gorilla: A fast, scalable, in-memory time series database. *Proc. VLDB Endow.*, 8(12):1816–1827, Aug. 2015.

[9] D. Peng and F. Dabek. Large-scale incremental processing using distributed transactions and notifications. In *Proceedings of the 9th USENIX Conference on Operating Systems Design and Implementation*, OSDI'10, pages 1–15, Berkeley, CA, USA, 2010. USENIX Association.

[10] A. Petermann, M. Junghanns, R. Müller, and E. Rahm. Graph-based data integration and business intelligence with biiig. *Proc. VLDB Endow.*, 7(13):1577–1580, Aug. 2014.

[11] C. Xie, C. Su, M. Kapritsos, Y. Wang, N. Yaghmazadeh, L. Alvisi, and P. Mahajan. Salt: Combining acid and base in a distributed database. In *Proceedings of the 11th USENIX Conference on Operating Systems Design and Implementation*, OSDI'14, pages 495–509, Berkeley, CA, USA, 2014. USENIX Association.

[12] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica. Spark: Cluster computing with working sets. In *Proceedings of the 2Nd USENIX Conference on Hot Topics in Cloud Computing*, HotCloud'10, pages 10–10. USENIX Association, 2010.

[13] 善明晃由. Ameba におけるログ解析基盤の変遷. In **第** *6* **回** *Web* **とデータベースに関するフォーラム技術報告セッション** *(WebDB Forum 2013)*, 2013.