社団法人　電子情報通信学会
THE INSTITUTE OF ELECTRONICS,
INFORMATION AND COMMUNICATION ENGINEERS

信学技報
TECHNICAL REPORT OF IEICE.

MapReduceを用いた大規模消費電力ログの体現ビュー実現手法

伊勢　勇輝† 　山本晋太郎† 　柗本　真佑† 　中村　匡秀†

† 神戸大学　〒 657–8531 兵庫県神戸市灘区六甲台町 1–1
E-mail: †{ise,shintaro}@ws.cs.kobe-u.ac.jp, ††{shinsuke,masa-n}@cs.kobe-u.ac.jp

あらまし　スマートシティでは，都市内の住宅や生活インフラから大量のデータを収集して，都市の現状を把握し，状況に応じた付加価値サービスが提供される．蓄積されるスマートシティデータはビッグデータであるため，用途の異なるアプリケーションがそれぞれ自分で 1 次データ（生データ）を検索，加工することは非常に時間がかかる．そこで本論文では，データベースの**体現ビュー (materialized view)** の考え方を導入し，大規模データをあらかじめアプリケーションの望む形に整形して保持する方式を検討する．本稿では特に，HBase 分散 KVS に蓄積されたスマートハウスの消費電力ログから，毎分，毎時，毎日という異なる時間幅の体現ビューを，MapReduce を用いて生成する手法を開発した．評価実験では，体現ビューを用いた提案手法と，1 次データに直接アクセスする従来手法の応答時間を比較する．実験の結果，アプリケーションが同じ条件のデータに何度もアクセスする場合や，条件に合致する 1 次データ数が多い場合に，特に提案法が優れることがわかった．

キーワード　大規模住宅ログ，体現ビュー，高速・効率的データアクセス，MapReduce，KVS，HBase

Implementing Materialized View of Large-Scale Power Consumption Log Using MapReduce

Yuki ISE†, Shintaro YAMAMOTO†, Shinsuke MATSUMOTO†, and Masahide NAKAMURA†

† Kobe University　Rokkoudai-cho 1–1, Nada-ku, Kobe, Hyogo, 657–8531 Japan
E-mail: †{ise,shintaro}@ws.cs.kobe-u.ac.jp, ††{shinsuke,masa-n}@cs.kobe-u.ac.jp

Abstract　The smart city provides various value-added services by collecting large-scale data from houses and infrastructures within a city. However, it takes a long time for individual applications to use and process the large-scale raw data directly. To reduce the response time, we use the concept of *materialized view* of database. For a given requirement of an application, the proposed method constructs a materialized view for caching the application-specific data. In this paper, we especially develop a method that uses MapReduce for large-scale power consumption data stored in HBase KVS. We conduct an experimental evaluation to compare the response time between cases with and without the materialized view. As a result, the proposed method with materialized view is effective especially when applications repeatedly access the same data, or when the application-specific data is derived from a large set of raw data.

Key words　large-scale house log, materialized view, high-speed and efficient data access, MapReduce, KVS, HBase

1. はじめに

スマートシティ [1], [2] とは，ICT 技術を用いて都市全体の高度な効率化を目指す次世代都市計画である．スマートシティでは，都市内の様々なモノやシステムから情報を集め，その情報に基づいて都市の現状を把握し，状況に応じた付加価値サービスが提供される．こうした情報は，都市内に設置されたセンサやシステムのロガー等から収集される．例えば，スマートハウスから取得される電力や家電機器の使用状況などの住宅情報や，環境センサから取得される気温や湿度などの環境データ，道路や鉄道から収集される交通データなどが知られている．

スマートシティから収集される情報は，非常に大規模かつ多種多様な**ビッグデータ**である．我々はスマートシティのビッグデータを一元的に蓄積・管理し，様々なアプリケーションから横断的に利用するためのプラットフォームの開発を目指している．先行研究 [3] [4] では，スマートハウスから得られる住宅ログを扱うプラットフォーム **Scallop4SC (Scalable Logging Platform for Smart City)** を提案している．Scallop4SC では，スマートハウスから取得した大量の住宅ログ（例えば消費電力のログ）を分散キーバリューストア (KVS) [5] に蓄積しておき，API を通し

て様々な用途のアプリケーションやサービス（例えば，消費電力の可視化，無駄の振り返り，ピークカット制御）に提供する．

通常，アプリケーションが要求するデータの範囲やその形式は，アプリケーションによってまちまちである．例えば，消費電力の可視化では，宅内の機器ごとの消費電力の時系列データが欲しい．また，ピークカット制御では宅内機器のトータルの消費電力量が必要となる．したがって，各アプリケーションは，蓄積された 1 次データ（生データ）をそのまま利用するのではなく，欲しいデータを検索し，必要な形式に加工・整形してから利用することが一般的である．しかしながら，スマートシティから蓄積される 1 次データは大規模なため，検索や加工に非常に時間がかかる．そのため，アプリケーションの実行時に毎回データ検索や加工を行うことは現実的ではない．

本研究の目的は，蓄積された大規模なスマートシティデータを，様々なアプリケーションに効率的に提供するための手法を提案することである．提案法のキーアイデアとして，データベースの**体現ビュー (materialized view)** [6] の考え方を取り入れる．体現ビューはクエリの結果をあらかじめ実際のデータテーブルにキャッシュし，高速なデータアクセスを可能とする技術である．提案法では，分散 KVS である **HBase** [7] に蓄積された 1 次データに対し，アプリケーションがあらかじめ欲しい形に検索・整形した体現ビューを生成しておく．アプリケーションは，1 次データにアクセスするのではなく，体現ビューにアクセスすることで高速かつ効率的なデータアクセスを実現する．スマートシティデータは基本的に更新がないログデータなので，取得した 1 次データを随時体現ビューに変換しても支障がない．1 次データから体現ビューへの変換は，Hadoop/MapReduce [8] を用いたバッチ処理で行う．必要に応じて計算ノードを増やすことで効率的な体現ビューの構築が可能となる．

本稿では対象を消費電力ログに絞り，提案手法の妥当性を検証する．具体的には，実際のスマートハウス環境において 3 秒に 1 回ずつ記録した 32 種の機器の消費電力ログを，MapReduce バッチで処理し，毎分，毎時，毎日という異なる時間幅の体現ビューを生成する．また評価実験では，まず体現ビューを用いた提案手法と，1 次データに直接アクセスする従来手法の応答時間を比較する．次に，MapReduce バッチに要する時間を評価する．実験の結果，アプリケーションが同じ条件のデータに何度もアクセスする場合や，条件に合致する 1 次データ数が多い場合に，特に提案法が優れることがわかった．

2. 準　　備

2.1 スマートシティにおける付加価値サービス

スマートシティでは，都市内の情報を集めて都市の現状を把握し，状況に応じた付加価値サービス（**スマートシティサービス**と呼ぶ）が提供される．期待されるサービス分野としては，例えば，省エネルギー，交通の最適化，局地的な経済トレンド分析，エンターテインメント，地域医療・健康，災害対策，農業支援など多岐にわたる．

また各サービス分野内でも様々なバリエーションのサービスが考えられる．例えば，省エネルギー分野に焦点を当てると，次のようなスマートシティサービスが考えられる．

（1） **消費電力可視化サービス [9]:** スマートシティ内の住居（スマートハウス）から，消費電力情報を取得し，都市の電力使用状況を見える化するサービス．各住居単位や町単位，機器単位，現在消費電力量，過去の消費電力量の推移など，様々な観点から可視化する．実際に使用した消費電力を目に見える形で可視化することにより，住民の省エネ意識の向上をはかる．

（2） **無駄検出サービス [10]:** スマートハウス内の消費電力やセンサのデータから，無駄な電力使用がないのかを検出し，通知するサービス．また, 過去にどのような無駄があったのかを振り返ることも可能である．家毎など自分が振り返りたい情報について検索をかけて見ることができるサービス．

（3） **ピークカットサービス [11]:** 住居，または，地域の消費電力が一定値を超えた場合に，あらかじめ設定しておいた機器を自動的に停止あるいは運転を順延させて，同時最大消費電力を抑えるサービス．

スマートシティサービスのために利用されるデータは，都市内の大量のモノやシステムから収集される．したがって，その量 (volume)，種類 (variety) は非常に大規模となる．また，状況を把握するために，なるべく最近のデータを反映する必要があり，情報の鮮度・スピード (velocity) も重要になる．これらから，スマートシティサービスのデータは**ビッグデータ**となる．

街や住居からデータを収集して分析し，活用するサービスやアプリケーションは従来からも存在する．しかしそれらのほとんどは，データ容量の制限から，アプリケーションに必要とされるデータのみを必要な粒度で蓄積し，利用する形態がほとんどである．一方，クラウド時代では，データ容量の制限は事実上なくなる．よって，**生ビッグデータ（１次データ）**を資産として蓄積し，様々な用途に横断的に活用，再利用できるようなプラットフォームが求められている．

2.2 先行研究:Scallop4SC

我々は先行研究において，スマートシティ内で取得される多種多様かつ巨大なログデータを蓄積・活用するためのスマートシティ向けデータ処理プラットフォーム **Scallop4SC** の開発を行なっている [3] [4].

Scallop4SC のアーキテクチャを図 1 に示す．各スマートハウスからロガーを通して収集されたログ情報はネットワークを通じて，Scallop4SC の管理する HBase 分散 KVS データベース上に蓄積され，Hadoop 分散処理基盤により処理が施される．スマートシティ全体の構成情報は，MySQL 関係データベース (RDB) 上で管理される．各種スマートシティサービスは Scallop4SC の提供する API を通じてログ情報や構成情報にアクセスし，住民へのサービスとして提供される．

現在の Scallop4SC プロトタイプでは，われわれの研究グループの 32 種類の機器から，3 秒に 1 回ずつ消費電力データを取得している．1 分に 640 行 (= 20 件 x 32 機器)，1 時間に 38,400 行，1 日で 921,600 行のエントリが追加される．また，Scallop4SC では消費電力以外にも，センサから取得される環境データや，機器を操作した操作ログなど様々な種類のデータも蓄積されていくので，そのデータの種類・量は更に膨大なものになる．

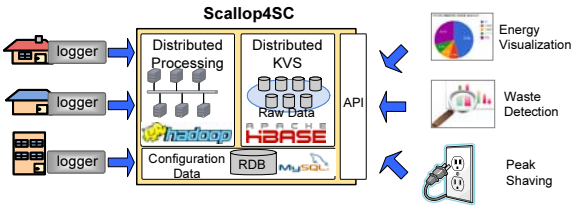

図 1　従来の Scallop4SC アーキテクチャ

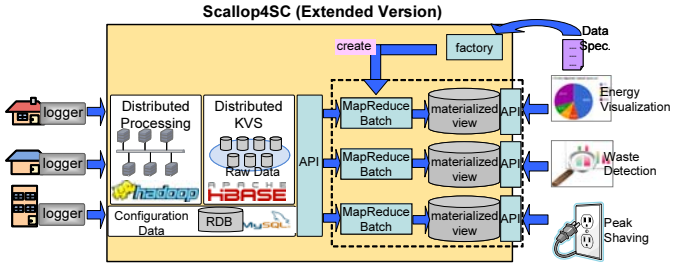

図 2　拡張された Scallop4SC アーキテクチャ

2.3　現状の課題

Scallop4SC に蓄積された大規模な生データ (1 次データ) は，多種多様なアプリからアクセスされることになる．しかし，アプリケーションが要求するデータの種類や範囲，形式はアプリケーションごとに異なる．現バージョン Scallop4SC の API は低レベルなデータ取得しか実装されておらず，現状アプリケーションが必要に応じて，自ら必要な形に加工・整形している．

しかしながら，アプリケーションが必要とするデータが大量の 1 次データから計算される場合，非常に時間がかかる．また，同条件のデータを繰り返し要求する場合には，大規模データに対して同じ計算を繰り返すため，非効率である．

様々なアプリケーションが，必要なデータに高速にアクセスし，欲しいデータを効率的に取得できる仕組みが必要である．

3.　提 案 手 法

3.1　アーキテクチャ

上記課題を解決するキーアイデアとして，本稿ではデータベースの**体現ビュー**の考え方を導入する．アプリケーションが要求するデータの範囲や形式は，スマートシティの生データをある条件に基づいて眺めた見方，すなわちビュー (view) と捉えることができる．通常のビューはデータを閲覧するときに初めてクエリが実行され，1 次データから動的に生成される．これに対し，体現ビューはクエリの結果をあらかじめ実際のデータテーブルにキャッシュしておき，こちらを閲覧することで高速なデータアクセスを可能とする．

図 2 に，提案手法を取り入れた新しい Scallop4SC のアーキテクチャ図を示す．提案手法では，アプリケーション開発者が，あらかじめ，どのようなデータが必要なのかを**データ仕様** (Data Spec.) にまとめて Scallop4SC に与える．次に，Scallop4SC の factory コンポーネントが与えられたデータ仕様を解釈し，要求される体現ビューを構築するための **MapReduce バッチプログラム**を生成する．Scallop4SC は，生成されたバッチプログラムを実行し，1 次データからアプリケーションが望む形の体現ビューとアクセス API を生成する．アプリケーションは，体現ビューのアクセス API を介して，高速に必要なデータにアクセスすることができる．

なお，体現ビューの生成は，Hadoop 分散処理システム上での定期バッチ処理で行われる．体現ビューの生成元となるスマートシティデータは，更新がないログデータがほとんどなので，取得した 1 次データを随時体現ビューに変換しても支障がない．

本研究の最終的な目標は，図 2 のアーキテクチャ全体の実装であるが，このうち本論文では，破線で囲った部分を主に議論する．具体的には，体現ビューの実現方針を決定することと，静的に作成した MapReduce バッチプログラムを実際の住宅ログに対して実行し，アプローチの妥当性を予備評価することを目的としている．本論文では特に，われわれが構築しているスマートハウス環境から得られた消費電力ログを用いる．

3.2　Scallop4SC に蓄積された 1 次データ

体現ビューの実現方針を考察する前に，Scallop4SC に蓄積されるスマートシティの 1 次データを説明する．Scallop4SC は，様々な種類の大規模ログを収容するために，厳密なデータスキーマを用意せずに，各ログデータをシンプルなキーとバリューで表現する．これらを HBase 分散 KVS に格納する．

表 1 に，我々の研究室で取得した消費電力ログの格納例を示す．各行キーは，ログがとられた日付 (date) と時刻 (time)，ログの種類 (type)，ログがとられた住居・場所 (home)，ログを取得した機器（device）の連接で表現される．行キーの構成は，バッチ処理を日時で指定することが多いため，dateTtime が先頭に現れ，分類の粒度が粗い順に，type, home, device と続いている．

各行に対して，列ファミリーは，各データを説明する属性値を示すメタデータ (info) と，データ値 (data) を保持している．例えば，表 1 の 1 行目のエントリは，2013 年 1 月 18 日 22 時 00 分 10 秒の cs27 研究室のエナジータイプ，機器 ID が pow001 の機器の瞬間消費電力値を示している．メタデータ info:は，それぞれのデータ値を独立した列で格納しており，アプリケーションから参照可能になっている．また，データの単位（unit）や，位置 (location) などの付加的な情報も格納されている．

3.3　消費電力データのための体現ビューの実現

ここで表 1 の 1 次データに対して,「機器毎の消費電力量を，毎日の単位で取得したい」というデータ仕様を持つアプリケーションを考える．原理的には，アプリケーションがメタデータ date と device を基に 1 次データを検索し，data の総和を計算すればよい．しかし，機器数が多い場合，あるいは，データの取得間隔が細かい場合には膨大なデータがマッチするため，計算のオーバーヘッドが大きくなる．また，同じ機器，日付のリクエストに対して，何度も同じ計算をすることは非効率である．

そこで，このアプリケーション用の体現ビューを実現することを考える．方針として，Scallop4SC の 1 次データをバッチ処理により，機器ごとの日次総消費電力を計算し，機器. 日付（例えば，pow001.2013-01-18）を行キー，総消費電力を値とする HBase テーブルを作成すれば，このアプリケーション用の体現ビューを実現できる．同様に，別のアプリケーションが「機器ごとの消費電力量を，毎時の単位で取得したい」というデータ

表 1　1 次データの形式

Row Key	**Column Families**							
	info:							**data:**
(dateTtime.type.home.device)	date	time	device	home	location	type	unit	
2013-01-18T22:00:10.Energy.cs27.pow001	2013-01-18	22:00:10	pow001	cs27	s101	Energy	W	140.3
2013-01-18T22:00:10.Energy.cs27.pow002	2013-01-18	22:00:10	pow002	cs27	s101	Energy	W	0
2013-01-19T12:30:00.Energy.cs27.pow001	2013-01-19	12:30:00	pow001	cs27	s101	Energy	W	120.7

仕様があった場合，機器. 日付 T 時間 (pow001.2013-01-18T15) を行キーとして同様に HBase テーブルを作成すればよい．

上記を一般化して，提案手法では，体現ビューを次のような HBase テーブルで実現する．

行キー: 体現ビューの各データを性質づける．1 次データのメタデータの値を，アプリケーションの利用しやすい形式，順番で連接したものを用いる．1 次データを集約するある観点（条件）を示すことから，この行キーを**集約キー**と呼ぶことにする．

値: 集約キーに対応する値．集約の条件にしたがって 1 次データを計算した値が格納される．

体現ビューの実現に HBase を用いる理由は，体現ビューの各データをキーの指定だけで値を即座に返すという，キャッシュの性質が求められるためである．また，体現ビューのデータの種類はアプリケーションによって異なるため，厳格なスキーマを定義せずに利用したいという要求も反映されている．

ここで例として，機器ごとの消費電力量を，毎分，毎時，毎日の単位で取得したいという要求を持つアプリケーションを考える．この場合，毎分，毎時，毎日，それぞれのデータに対して，以下のような体現ビューを設計することができる．以下では，日付における年月日をそれぞれ YYYY, MM, DD，時刻の時，分をそれぞれ hh, mm と表す．

(a) MinutelyView:　機器ごとの毎分の消費電力量を示す体現ビューなので，以下のような HBase で実現する (表 2(a)).

- 集約キー: [機器 ID.YYYY-MM-DDThh:mm]
- 値:その分でその機器が消費した消費電力量

(b) HourlyView: 機器ごとの毎時の消費電力量を示す体現ビューなので，以下のような HBase で実現する (表 2(b)).

- 集約キー: [機器 ID.YYYY-MM-DDThh]
- 値:その時でその機器が消費した消費電力量

(c) DailyView: 機器ごとの毎日の消費電力量を示す体現ビューなので，以下のような HBase で実現する (表 2(c)).

- 集約キー: [機器 ID.YYYY-MM-DD]
- 値:その日でその機器が消費した消費電力量.

3.4　MapReduce を用いた体現ビューの構築法

1 次データから体現ビューを構築する手段として，提案手法では MapReduce バッチを用いる．MapReduce は，巨大なデータセットを持つ並列可能な問題に対し，複数の計算ノードを用いて並列処理させるためのフレームワークである．入力される各データを指定するキーとバリューの組に変換する map 処理と，同じキーを持つ複数のバリューを集約する reduce 処理から構成される．具体的には，以下の処理を行う MapReduce バッチプログラムを実装する．

（1） **Create フェーズ:** 体現ビューのための HBase テーブル $view$ を新規作成する．

表 2　機器ごとの消費電力量を格納する体現ビュー（毎分，毎時，毎日）

(a) MinutelyView

Aggregation Key	**Column Families**
(deviceID.YYYY-MM-DDThh:mm)	Minutely Consumption
pow001.2013-01-18T15:30	687.1
pow002.2013-01-18T15:30	0
pow001.2013-01-19T16:00	560.6

(b) HourlyView

Aggregation Key	**Column Families**
(deviceID.YYYY-MM-DDThh)	Hourly Consumption
pow001.2013-01-18T15	41363.7
pow002.2013-01-18T15	0
pow001.2013-01-19T16	38393.8

(c) DailyView

Aggregation Key	**Column Families**
(deviceID.YYYY-MM-DD)	Daily Consumption
pow001.2013-01-18	588072.6
pow002.2013-01-18	0
pow001.2013-01-19	633055.6

（2） **Scan フェーズ:** データ仕様に基づいて，取得すべきデータ範囲を決定し，1 次データを絞り込む.

（3） **Map フェーズ:** 絞り込まれた 1 次データの各レコード d に対して，d のメタデータを利用して，集約キー k を生成する．また，d から必要な値 v を抽出する．得られたキーバリュー (k, v) を出力する．

（4） **Reduce フェーズ:** 同じキー k を持つキーバリュー $(k, v_1), (k, v_2), ..., (k, v_n)$ について，データ仕様に基づいた演算 $\otimes$ を行い，値 $val = v_1 \otimes v_2 \otimes ... \otimes v_n$ を得る．得られたキーバリュー (k, val) を出力する．

（5） **Put フェーズ:** (k, val) を $view$ に登録する．

ここで例として，3.3 節で述べた DailyView を作成してみる．まず Scan フェーズでは，2013 年 1 月 18 日の消費電力量を登録する場合，まず 1 次データからキーの先頭が「2013-01-18」で一致するものを検索する．次に Map フェーズでは，得られたデータそれぞれに対して，メタデータ (info) から日付，機器 ID を取得し「deviceID.YYYY-MM-DD」の形で集約キーを生成する．また，そのキーに対する値として消費電力値 (data) から値を取得し設定する．Reduce フェーズでは，各集約キーに対して，足し算 + を適用し，その機器のその日の合計消費電力量を算出する．最後に Put フェーズでは，集約キーとそれに対応する合計消費電力量の組を DailyView に書き込む.

Scallop4SC はファイルシステムに Hadoop を採用しているため，Hadoop の複数の計算ノードを用いて，MapReduce 処理を並列分散させることができる．したがって，巨大な 1 次データ

に対しても，必要に応じて計算ノードを増やすことで効率的な体現ビューの構築が可能となる．

3.5 体現ビューへのアクセス API

生成された体現ビューをアプリケーションに提供するために，アクセス API を定義する．体現ビューの設計方針より，この API は以下のような単純なプログラムで実現できる．

（1） アプリケーションから与えられたパラメータに基づいて，集約キーを生成する．

（2） 体現ビューに集約キーを渡して，値を取得する．

（3） 取得した値をアプリケーションに返す．

例として，機器 ID「pow005」の 2013 年 1 月 18 日 10 時台の消費電力量データをアプリケーションが要求する場合を考える．アプリケーションは，API にパラメータとして（pow005，2013-01-18T10) を渡す．API は，集約キー「pow005.2013-01-18T10」を生成する．API は，生成した集約キーで HourlyView を検索し，消費電力量を取得し，アプリケーションに返す．

4. 評 価 実 験

4.1 実 験 概 要

体現ビューを用いた提案手法の有用性を確かめるために，実際のスマートハウスに蓄積された消費電力ログを用いた評価実験を行う．実験の目的は，以下の事柄を評価することである．

評価項目 E1: 体現ビューの利用によってアプリケーションがどれくらい高速にデータにアクセスできるようになるか？

評価項目 E2: 体現ビューの生成にかかる時間は？

E1 を評価するために，アプリケーションが 1 次データに直接アクセスする場合と，体現ビューにアクセスする場合の両者の応答時間を計測し，比較する．また，E2 を評価するために 1 次データから体現ビューを生成するための 1 回のバッチ処理にかかる時間を計測する．

4.2 実 験 環 境

実験で利用する 1 次データは，我々の研究グループで開発しているスマートハウス環境 CS27-HNS において，32 種類の機器から 3 秒毎に記録した消費電力ログである．実験ではこのデータのうち 2 日間分を取り出している．

この 1 次データから，機器毎の毎分，毎時，毎日の消費電力量を集計した 3 つの体現ビューを生成する．ここで，これらの体現ビューは，3.3 節で説明したものと同じである．

MapReduce バッチは，Java を用いた静的な MapReduce プログラムとして実装した．使用したライブラリは，hadoop-core-1.0.3.jar, hbase-0.94.2.jar である．実行環境は，研究室の Linux クラスタ (Pentium4, 3.0GHz, 2GB × 8 台) の Hadoop ファイルシステムにインストールされた HBase を用いた．

4.3 実 験 方 法

評価項目 E1, E2 にそれぞれ対応した 2 種類の実験を行った．

■実験 1: 応答時間の比較

実験において機器毎の毎分，毎時，毎日の消費電力を求める下記の API を考える．

```
double getMinutelyConsumption(device, date, time);
double getHourlyConsumption(device, date, time);
double getDailyConsumption(device, date);
```

上記の API を，直接 1 次データにアクセスする従来方式と，体現ビューを用いる提案方式の 2 通りで実装した．実験では，上記の各 API に対して，パラメータをランダムに生成して渡し，その実行時間を計測する．これを 100 回繰り返し，平均時間を求めた．従来方式で実装した API からデータを取得する場合 (direct access) と，提案方式で実装した API から取得する場合 (view access) の 2 通りで計測し，両者の結果を比較する．

表 3 実験 1: 応答時間の比較結果（単位:sec）

API	getMinutely()	getHourly()	getDaily()
direct access	0.320	16.943	408.277
view access	0.008	0.002	0.001

表 4 実験 2: 体現ビュー生成時間（単位:sec）

	MinutelyView	HourlyView	DailyView
CPU Time	130.123	220.682	370.183
# of Records Processed	640	38,400	921,600

■実験 2: 体現ビューの生成時間の計測

1 次データから，3 種類の体現ビュー MinutelyView, HourlyView，DailyView を生成する MapReduce バッチの実行時間を計測する．ここで，MinutelyView は，消費電力ログから 1 分間の総消費電力の計算に要する時間を計測する．これを 2 日分繰り返し，その平均時間を計測した．同様に，HourlyView は 1 時間，DailyView は 1 日のデータの処理時間を計測し，それぞれ平均時間を算出した．

4.4 実 験 結 果

表 3 に実験 1 の結果を示す．行 direct access が 1 次データから直接データを取得した場合，行 view access が体現ビューからデータを取得した場合を示す．

API の getMinutely() は毎分の消費電力量を取得するのにかかる平均時間，getHourly() は毎時の消費電力量を取得するのにかかる平均時間，getDaily() は毎日の消費電力量を取得するのにかかる平均時間を示す．

表から，view access は direct access に比べて大幅に応答時間を短縮できていることが分かる．また，分，時間，日と取得するデータ件数が多くなるほど，direct access は長い応答時間を要するのに対し，view access はほぼ一定の応答時間になっている．

次に，表 4 に実験 2 の結果を示す．MinutelyView, HourlyView, DailyView それぞれの体現ビューの生成に要する MapReduce プログラム実行時間と処理された 1 次データのレコード数を示している．表から，体現ビューの生成時間は，処理するデータのレコード数に応じて大きくなることがわかる．

5. 考　　察

5.1 データアクセスに関するスケーラビリティ

表 3 の実験 1 の結果から，アプリケーションがデータにアクセスする際のスケーラビリティについて考察する．従来法 direct access 方式の API では，本実験での消費電力の計算において，getMinutely() では 640 件，getHourly() では 38,400 件，getDaily() では 921,600 件のレコードを集計している．応答時

間の測定結果では，1 次データのレコード数にほぼ比例して，応答時間が長くなっている．よって，従来法は必要とする 1 次データのレコード数に対して，スケーラビリティに乏しい．

一方，提案法 view access 方式の API では，どの API も数ミリ秒程度の非常に短い応答時間である．事前に必要な形式に整形済みのため，データアクセスに関してはスケーラビリティに優れる．getMinutely() に一番時間がかかっているのは，体現ビューの行数に関連していると思われる．集約の単位が細かい毎分の体現ビューが，集約キーのバリエーションが最も多くなる．その結果，わずかではあるがキーバリューの検索・取得に時間がかかってしまうのが原因だと考えられる．

5.2 体現ビュー生成にかかるバッチ処理のコスト

次に表 4 の実験 2 の結果から，体現ビューの生成コストについて考察する．バッチ処理に要する時間は，体現ビュー作成時に処理する 1 次データのレコード数に応じて大きくなる．処理時間がレコード数に厳密に比例しないのは，MapReduce バッチ処理時のオーバーヘッドのためと推測される．1 次レコード数が少ない MinutelyView 作成時おいても，オーバーヘッドの割合が大きくなっている．実験の結果では，1 分あたりの 1 次データを処理するのに，1 分以上かかっている．したがって，1 分毎のバッチ処理を行うことは運用上好ましくない．

1 次データのレコード数は，アプリケーションから与えられるデータ仕様に依存する．したがって，大量のデータ集計を行うデータ仕様に対しては，スケーラビリティの確保をはからなければならない．幸い，MapReduce 処理は並列分散が容易で，計算ノードを増加することで，大規模なデータに対して容易にスケールアウトが可能となる．1 度にバッチ処理を行うデータの範囲や，バッチ処理を実行するタイミングをうまく制御することで，より効率的な体現ビューの生成が可能になると考えられる．これらの運用の効率化に関する事柄は将来課題としたい．

5.3 直接アクセスと体現ビューの使い分け

提案法を用いるためには，事前に体現ビューを生成する前処理が必要となる．したがって，一度に必要な 1 次データのレコード数があまり多くない場合や，頻繁にアクセスしないデータについては，直接アクセスを採用することも考えられる．ここでは，実験 1 と実験 2 を合わせた結果，つまり，ビュー生成にかかる時間と体現ビューからのデータ取得時間をあわせた時間を評価し，トータルでどちらの効率が良いかを考察する．

いま提案手法，従来手法の 1 回あたりのデータアクセス時間をそれぞれ v，d とする．また，提案手法のバッチ処理にかかる時間を b とする．アプリケーションが n 回データにアクセスするとき，トータルの実行時間はそれぞれ次のようになる．

$$\text{従来手法} = n * d \quad \text{(時間)}$$

$$\text{提案手法} = b + n * v \quad \text{(時間)}$$

b, v, d の値は実験から得られたものを利用し，提案手法の時間が従来手法より短くなる n を計算すると，次のようになった：[getDaily(): $n \geqq 1$]，[getHourly(): $n \geqq 14$]，[getMinutely(): $n \geqq 418$]．集約する 1 次データのレコード数が多ければ多いほど，また，アプリケーションによるデータアクセスの頻度が高かれば高いほど，提案法がより顕著な効果を示すことができる．逆に，極端に頻度が低い場合や，集約レコード数が少ない場合には，従来法を選択することも考えられる．

6. おわりに

本稿では，スマートシティにおいて蓄積された大規模な 1 次データを，様々なアプリケーションが効率的に活用するための手法を提案した．提案手法では，アプリケーションが要求するデータ仕様に基づいて，そのデータをキャッシュする体現ビューを生成することで，効率的なデータアクセスを可能にする．提案する体現ビューを HBase 上に構築し，Hadoop/MapReduce によって生成する方法を示した．またデータのアクセス時間，体現ビューの生成時間を評価し，提案法の有効性を評価した．その結果，大規模な 1 次データを集約する場合や，頻繁にデータにアクセスする場合には，提案手法が非常に効果的であることがわかった．今後の課題は，データ仕様から自動的にバッチ処理を生成する仕組み，バッチ処理の運用効率化について，さらなる検討を行いたい．

謝辞 この研究の一部は，科学技術研究費（基盤研究 C 24500079, 基盤研究 B 23300009），および，関西エネルギー・リサイクル科学研究振興財団の助成を受けて行われている．本研究の構想段階において有用な議論をいただきました同志社大・波多野賢治先生，奈良先端大・宮崎純先生に感謝いたします．

文　献

[1] R.G. Hollands, “Will the real smart city please stand up?,” City: analysis of urban trends, culture, theory, policy, action, vol.12, no.3, pp.303–320, 2008.

[2] A. Mahizhnan, “Smart cities: The singapore case,” Cities, vol.16, pp.13–18, 1999.

[3] S. Yamamoto, S. Matumoto, and M. Nakamura, “Using cloud technologies for large-scale house data in smart city,” In International Conference on Cloud Computing Technology and Science (CloudCom2012), pp.141–148, Dec. 2012.

[4] K. Takahashi, S. Yamamoto, A. Okushi, S. Matsumoto, and M. Nakamura, “Design and implementation of service api for large-scale house log in smart city cloud,” In International Workshop on Cloud Computing for Internet of Things (IoTCloud2012), pp.815–820, Dec. 2012.

[5] Seeger M., “Key-value stores: A pracitical overview,” Computer Science and Media. Ultra-Large-Sites, vol.SS09, pp.1–21, 2009.

[6] I.S. Mumick, “The rejuvenation of materialized views,” International Conference on Information Systems and Management of Data (CISMOD 95), vol.1006, pp.258–264, 1995.

[7] A. Khetrapal and V. Ganesh, “Hbase and hypertable for large scale distributed storage systems,” 2006.

[8] D. Jeffrey and G. Sanjay, “Mapreduce: Simplified data processing on large clusters,” Commun. ACM, vol.51, no.1, pp.107–113, Jan. 2008. http://doi.acm.org/10.1145/1327452.1327492

[9] City of Yokohama, “Yokohama smart city project,” `http://www.city.yokohama.lg.jp/ondan/english/`.

[10] 北岡賢人，瀬戸英晴，まつ本真佑，中村匡秀，“ホームネットワークシステムにおける機器状態ログからのエネルギー浪費行動の検出,” 電子情報通信学会技術研究報告，第 110 巻，pp.37–42, 2011.

[11] Y.-X. Lai, J.J.P.C. Rodrigues, Y.-M. Huang, Hong-GangWang, and C.-F. Lai, “An intercommunication home energy management system with appliance recognition in home network,” Mobile Networks and Applications, vol.17 Issue 1, pp.132–142, 2012.