2013 IEEE International Conference on Cloud Computing Technology and Science

Materialized View as a Service for Large-Scale House Log in Smart City

Shintaro YAMAMOTO, Shinsuke MATSUMOTO, Sachio SAIKI, Masahide NAKAMURA
Graduate School of System Informatics, Kobe University, JAPAN
1-1 Rokkodai-cho, Nada-ku, Kobe, Hyogo 657-8501, Japan
Email:shintaro@ws.cs.kobe-u.ac.jp, {shinsuke, masa-n}@cs.kobe-u.ac.jp, sachio@carp.kobe-u.ac.jp

***Abstract*—Smart city provides various value-added services by collecting large-scale data from houses and infrastructures within a city. To use such large-scale raw data, individual applications usually take expensive computation effort and large processing time. To reduce the effort and time, we propose *Materialized View as a Service (MVaaS)*. Using the MVaaS, each application can easily and dynamically construct its own *materialized view*, in which the raw data is cached in an appropriate format for the application. Once the view is constructed, the application can quickly access necessary data.**

In this paper, we design a framework of MVaaS specifically for large-scale *house log*, managed in our smart-city data platform Scallop4SC. In the framework, each application first specifies how the raw data should be filtered, grouped and aggregated. For a given data specification, MVaaS dynamically constructs a MapReduce batch program that converts the raw data into a desired view. The batch is then executed on Hadoop, and the resultant view is stored in HBase. We conduct an experimental evaluation to compare the response time between cases with and without the proposed MVaaS.

***Keywords*- large-scale: house log: materialized view: high-speed and efficient data access: MapReduce: KVS: HBase;**

I. INTRODUCTION

Smart city refers to a next-generation city planning, encouraging to improve the efficiency of the city with ICT technologies [1][2]. The smart city provides various value-added services. A significant characteristic of the services is to use various information of houses and infrastructures within the city. The information include energy consumptions of devices, operation log of household appliances and equipment, environmental data such as temperature and humidity, traffic data from roads and railroads. These are gathered from sensors and system loggers deployed in heterogeneous systems. In general, the information gathered from the smart city is *Big Data*, comprising large-scale and heterogeneous data items. Our long term goal is to provide a universal platform, on which applications and services can extensively use the smart city data for various purposes.

In our previous research [3][4], we proposed a logging platform, called *Scallop4SC (Scalable Logging Platform for Smart City)*. Exploiting cloud technologies Hadoop and HBase, Scallop4SC processes and stores the large amount of *house logs* (e.g., power consumption logs) from *smart homes* within a smart city. Through API, Scallop4SC provides the smart city data for various applications, such as energy visualization, detection of wasteful use, and peak shaving.

In general, required data vary from one application to another. Hence, each application has to transform the raw data in Scallop4SC into its appropriate format and granularity. However, due to the size and variety of the raw data, the transformation poses expensive computation and long processing time for the application.

To cope with the problem, we introduced the concept of *materialized view* in Scallop4SC [5]. The materialized view is a database technology that *caches* results of queries in an actual table to improve the response time [6]. Our experiment showed that the materialized view dramatically improved the response time. However, each materialized view was statically created by a proprietary MapReduce program. Thus, application developers had to be familiar with complex knowledge of Hadoop/MapReduce and HBase, for implementing their own materialized views. It was also difficult to reuse the existing views for other applications.

This motivates us to encapsulate complex creation and management of the materialized views in an abstract cloud service. This is what we call *Materialized View as a Service (MVaaS)* in this paper. For a given *recipe* of required data (called *data specification*), MVaaS dynamically creates a materialized view for an individual application. Once the view is constructed, the application can quickly access necessary data through API of the view.

In this paper, we design a framework of MVaaS specifically for Scallop4SC. In the framework, an application developer of Scallop4SC creates a data specification prescribing how the raw data should be filtered, grouped and aggregated. Based on the data specification, MVaaS dynamically generates a MapReduce batch program that converts the raw data into a desired view. The batch is then executed on Hadoop, and the resultant view is stored in HBase. Finally the materialized view is accessed via MVaaS API. Thus, the developer can easily create and use own materialized view.

We conduct a performance evaluation using large-scale power consumption data, recorded in a real smart home environment for a year. It is shown that the proposed MVaaS is especially effective in cases where the applications repeatedly access the same data, or the view is derived from a large set of raw data.

978-0-7695-5095-4/13 $31.00 © 2013 IEEE
DOI 10.1109/CloudCom.2013.154

II. Preliminaries

A. Smart City and Services

The principle of the smart city is to gather data of the city first, and then to provide appropriate services based on the data. Thus, a variety of data are collected from sensors, devices, cars and people across the city. A smart city provides various value-added services, named *smart city services*, according to the situation of a city. Promising service fields include energy saving [7], traffic optimization [8], and agricultural support [9].

The size and variety of gathered data become huge in general. Velocity (i.e., freshness) of the data is also important to reflect real-time or latest situations and contexts. Thus, the data for the smart city services is truly *big data*.

The limitation of the storage is relaxed significantly by cloud computing technologies. It is now possible to store various kinds of data as they are, and to reuse the raw data for various purposes. We are interested in constructing a data platform to manage the big data for smart city services.

B. Scallop4SC (Scalable Logging Platform for Smart City)

We have been developing a data platform, called *Scallop4SC*, for smart city services [3][4]. Scallop4SC is specifically designed to manage data from *houses*. The data from houses are essential for various smart city services, since a house is a primary construct of a city. In near future, technolgies of smart homes and smart devices will enable to gather various types of house data.

Scallop4SC basically manages two types of house data: *house log* and *house configuration*. The house log is history of values of any dynamic data measured within a smart home. The house configuration is static meta-data explaining a house.

Figure 1 shows the architecture of Scallop4SC. For each house in a smart city, a logger measures various data and records the data as house log. The house log is periodically sent to Scallop4SC via a network. Due to the number of houses and the variety of data, the house log generally forms big data. Thus, Scallop4SC stores the house log using *HBase* NoSQL-DB, deployed on top of *Hadoop* distributed processing. On the other hand, the house configuration is static but structural data. Hence, it is stored in *MySQL* RDB to allow complex queries over houses, devices and people.

Scallop4SC API (shown in the middle in Figure 1) provides a basic access method to the stored data. Since Scallop4SC is an application-neutral platform, the API just allows basic queries (see [4]) to retrieve the *raw data*. Application-specific data interpretation and conversion are left for individual applications.

C. Introducing Materialized View in Scallop4SC

In general, individual applications use the smart city data in different ways. If an application-specific data is derived from much of raw data, the application suffers from

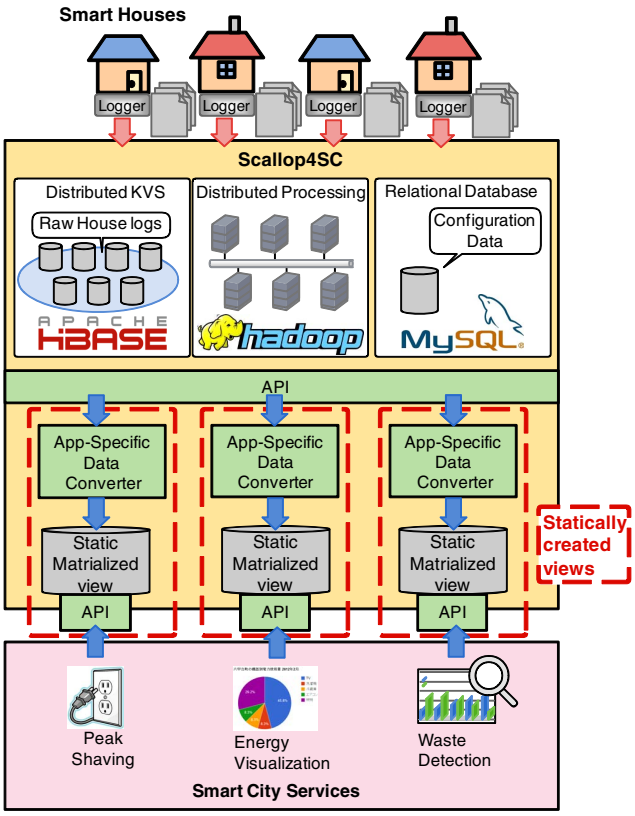

Figure 1. Scallop4SC with static MVs

expensive data processing and long processing time. This is because the application-specific data conversion is left to each application. If the application repeatedly requires the same data, the application has to repeat the same calculation to the large-scale data, which is quite inefficient.

To cope with this, we introduced *materialized view* in Scallop4SC, as shown in the lower part of Figure 1 [5]. The application-specific data can be considered as a *view*, which looks up the raw data based on a certain query. The materialized view is constructed as a table, which *caches* results of the query in advance.

Note, however, that the raw data in Scallop4SC is very large, and that we cannot use SQL for HBase to construct the view. Therefore, in [5] we developed a Hadoop/MapReduce program for each application, which efficiently converts the raw data into application-specific data.

A major limitation of the previous research is that the MapReduce program was statically developed and deployed. This means that each application developer has to implement a proprietary data converter by himself. The implementation requires development effort as well as extensive knowledge of HBase and Hadoop/MapReduce. It is also difficult to reuse the existing materialized views for other applications. These are obstacle for rapid creation of new applications.

III. MaterializedView as a Service for Large-Scale House Log

A. Materialized View as a Service (MVaaS)

To overcome the limitation, we propose a new concept of *Materialized View as a Service (MVaaS)*. MVaaS encapsulates the complex creation and management of the materialized views within an abstract cloud service. Although MVaaS can be a general concept for any data platform with big data, this paper concentrates the design and implementation of MVaaS for *house log* in Scallop4SC.

Figure 2 shows the new architecture of Scallop4SC with MVaaS. A developer of a smart city application first gives an order in terms of *data specification*, describing what data should be presented in which representation. MVaaS of Scallop4SC then dynamically creates a materialized view appropriate for the application, from large-scale house log of Scallop4SC. Thus, the application developer can easily create and use own materialized view without knowledge of underlying cloud technologies.

In the following subsections, we explain how MVaaS converts the raw data of house log into application-specific materialized view.

B. House Log Stored in Scallop4SC

First of all, we briefly review the data schema of the house log in Scallop4SC (originally proposed in [3]).

Table I shows an example of house logs obtained in our laboratory. To achieve both *scalability* for data size and *flexibility* for variety of data type, Scallop4SC stores the house log in the HBase key value store. Every house log is stored simply as a pair of key (**Row Key**) and value (**Data**). To store a variety of data, the data column does not have rigorous schema. Instead, each data is explained by a metadata (**Info**), comprising *standard properties* for house log.

The properties include *date* and *time* (when the log is recorded), *device* (from what the log is acquired), *house* (where in a smart city the log is obtained), *unit* (what unit should be used), *location* (where in the house the log is obtained) and *type* (for what the log is). Details of device, house and location are defined in an external database of house configuration in MySQL (see Figure 2). A row key is constructed as a concatenation of date, time, type, home and device. We assume that these logs are used as *raw data* by various smart city services and applications.

C. Idea of Converting Raw Data into Materialized View

The primary task of MVaaS is to convert the raw data in Table I into an application-specific view, according to a given *data specification* (defined later).

For example, let us consider a statistical application that uses *daily total energy of each device*. If the house logs were stored in RDB, we would specify the following SQL:

```
1 CREATE VIEW dailyConsumptionPerDevice AS
2 SELECT date,device,SUM(Data) FROM houselog
3 WHERE type=Energy AND unit=W
4 GROUP BY date,device;
```

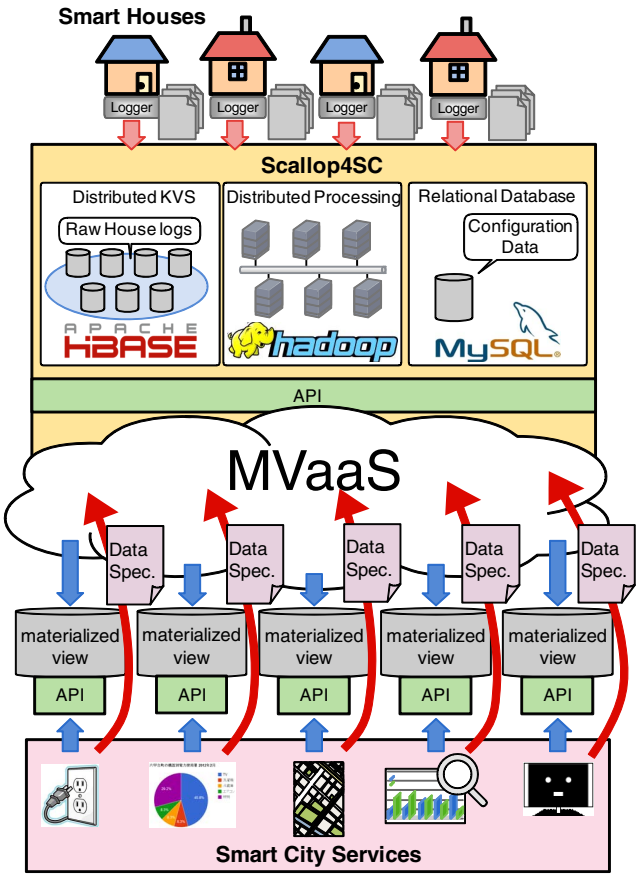

Figure 2. Extended Scallop4SC

As we can see in the above SQL, a typical view creation consists of three steps:

- **Step 1 (Filter):** Filter necessary data records out of all raw data (corresponding to WHERE clause).
- **Step 2 (Group):** For the data records, group multiple data records based on one or multiple properties (corresponding to GROUP BY clause).
- **Step 3 (Aggregate):** For each group, aggregate data values using an aggregate function (e.g., SUM(Data)).

However, our problem is not that simple, because the table of house log is huge and stored in a NoSQL database.

Our key idea is to implement a framework that executes the above three steps with a MapReduce program. Figure 3 shows a scenario of creating a materialized view of `dailyConsumptionPerDevice` from raw data of house log. First, we filter the raw data to extract relevant rows with Energy type and W unit (Step 1). The rows are then grouped according to the same date and the same device (Step 2). For each group, the energy values are aggregated by SUM function to obtain the daily energy consumption

Table I
RAW DATA: HOUSE LOG OF SCALLOP4SC

Row Key	Column Families							
	Data:	Info:						
(dateTtime.type.home.device)		date:	time:	device:	house:	unit:	location:	type:
2013-05-28T12:34:56.Energy.cs27.tv01	600	2013-05-28	12:34:56	tv01	cs27	W	living room	Energy
2013-05-28T12:34:56.Device.cs27.tv01	[power:off]	2013-05-28	12:34:56	tv01	cs27	status	living room	Device
2013-05-28T12:34:56.Environment.cs27.temp3	24.0	2013-05-28	12:34:56	temp3	cs27	Celsius	kitchen	Environment
2013-05-28T12:34:56.Environment.cs27.pcount3	3	2013-05-28	12:34:56	pcount3	cs27	people	living room	Environment
2013-05-28T12:35:00.Device.cs27.tv01	on()	2013-05-28	12:35:00	tv01	cs27	operation	living room	Device
:	:	:	:	:	:	:	:	:

of each device (Step 3). As will be explained later, in our MapReduce program, a map process, a shuffle process and a reduce process will execute Step 1, 2 and 3, respectively. Finally, the resultant view is stored in a new HBase table, which makes the view *materialized*.

D. Describing Data Specification

To generalize the scenario in the previous section for other applications, we here define the *data specification*. Intuitively, the data specification is considered as an order from an application, specifying how the resultant view should be generated. An SQL equivalent to a template of the proposed data specification is expressed as follows:

```
CREATE VIEW $view_name AS
SELECT $prop1, $prop2,...,$prop_n,
       $aggregate_function($expression)
FROM houselog
WHERE $filtering_condition
GROUP BY $prop1, $prop2,...,$prop_n;
```

In the above, `$` represents a placeholder, which can be replaced by a concrete name or expression in accordance with individual context.

In the proposed MVaaS, the following three items should be specified in the data specification, to create an application-specific materialized view: (1) a filtering condition, (2) grouping properties, (3) an aggregation function.

Filtering Condition: A filtering condition is a condition that extracts relevant rows from all house logs. Equivalently, it corresponds to `$filtering_condition` in the above SQL. In our framework, an (atomic) filtering condition $filter$ is defined by $filter = [prop\ cmp\ val]$, where $prop$ represents a property name (i.e., column qualifier) of the house log table (see Table I), cmp represents a comparison operator (`==, !=, >=, <=, > or <`), and val represents a value over the property. If $filter_1$ and $filter_2$ are both filtering conditions, then $[filter_1$ `&&` $filter_2]$ (logical AND), $[filter_1$ `||` $filter_2]$ (logical OR), `!` $[filter_i]$ (logical NOT) are also filtering conditions.

Grouping Properties: Grouping properties define groups of the filtered data. If multiple rows take the same values with respect to the properties, these rows are grouped into the same group. The grouping properties are equivalent to `$prop1,$prop2,...,$prop_n` in the above SQL.

In the data specification, grouping properties $group$ are defined by $group = [p_1, p_2, ..., p_n]$, where each p_i is one of the followings: `Data, device, house, unit, location, type, TIME, CUSTOM`. Properties from `Data` to `type` are those in Table I. `TIME` specifies temporal granularity of the grouping, which is defined by one from `year, month, date, hour, minute, second`. `CUSTOM` represents a user-defined grouping criteria, such as `AM` gathering all logs taken in the morning.

In the proposed method, concatenation of values of the grouping properties is used as every row key of the materialized view. Therefore, it is important to determine the order of p_i's, by considering priority among the properties.

Aggregate Function: An aggregation function defines how to aggregate the grouped house logs, which corresponds to `$aggregate_function($expression)` in the previous SQL.

In our framework, an aggregation function is defined by $aggregation = aggr(expression)$, where $aggr$ is one of the following functions: `SUM` (total sum), `MAX` (maximum value), `MIN` (minimum value), `AVG` (average), `COUNT` (count of items), `CONCAT` (concatenation of items), `ID` (identity), `CUSTOM` (user-defined function). Also, $expression$ is any expression defined with properties of the houselog table (see Table I).

E. Generating MapReduce Converter

Once a data specification is given, MVaaS dynamically generates a MapReduce program that converts the raw data into the target view. A pseudo code of the MapRedeuce program is as follows:

```
class Mapper
  method map(rowkey, column)
    if(filter(column)==true)
      for (p in group)
        key = concat(key, column.p.getValue())
      value = eval(expression(column))
      EMIT(key, value)

class Reducer
  method reduce(key, [v1, v2, ..., vn])
    result = aggr(v1, v2,...,vn)
    EMIT(key, result)
```

Mapper: For each row of house logs, Mapper class takes a row key and column families. Mapper first check if data in the column satisfies filtering condition. If it does, it generates a key by concatenating the value of each grouping property. The value is obtained by evaluating the expression according

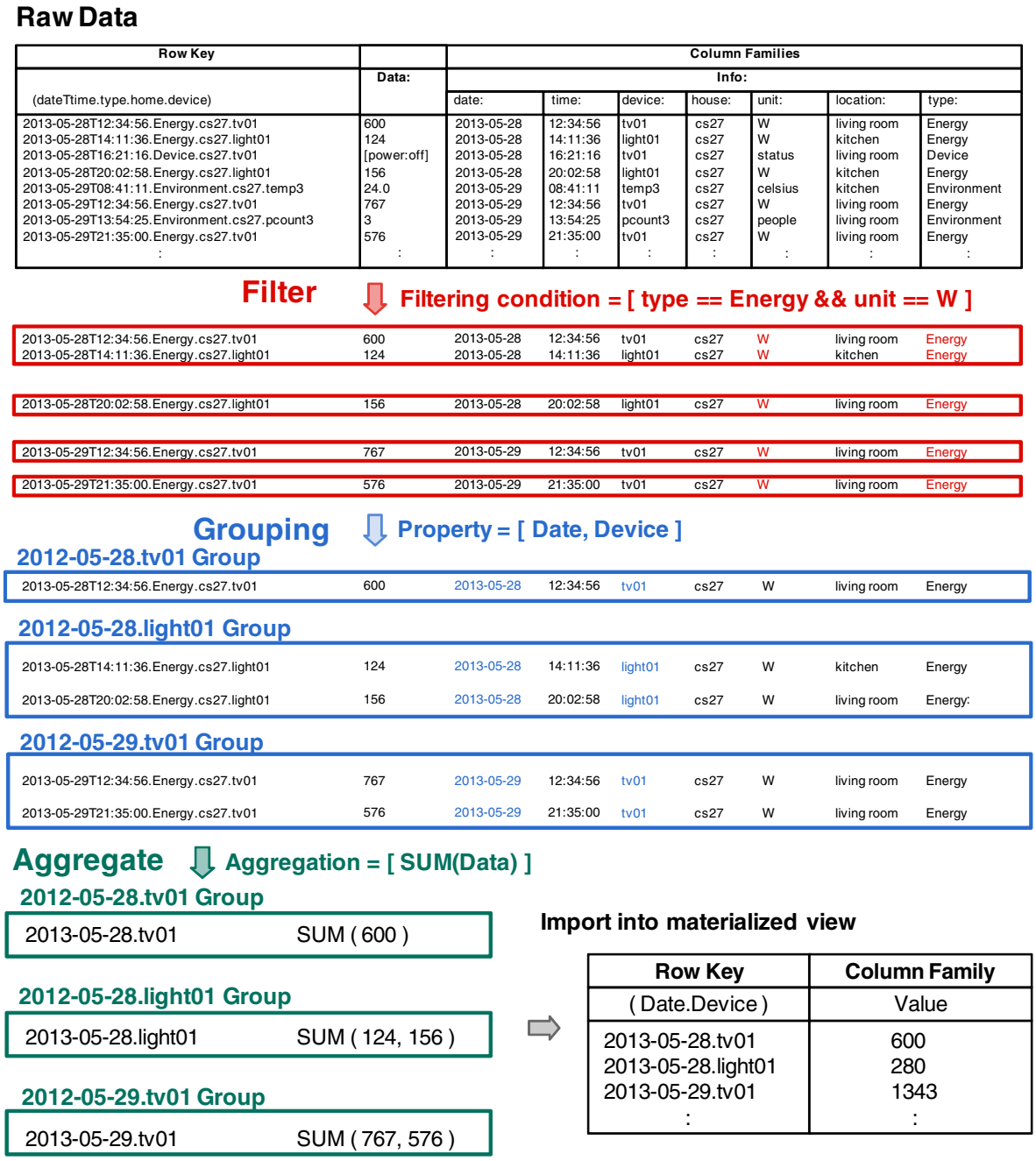

Figure 3. Flowchart of converting raw data into materialized view

to values of column. The pair of key and value is emitted to Reducer.

Reducer: After the map process, data values $v_1, v_2, ..., v_n$ with the same key are gathered and passed to Reducer. Reducer just aggregate values $v_1, v_2, ..., v_n$ using a designated aggregated function. The key and the result is emitted as a row of the materialized view.

The MapReduce program is then compiled and executed on Hadoop in Scallop4SC. The resultant key-values are stored in a new HBase table as a materialized view. The creation of the view generally takes time, depending on the data specification and the size of raw data chosen. However, once it is created, the application can access the data quickly.

IV. Experimemtal Evaluation

A. Overview of Experiment

We conduct an experiment to evaluate performance of MVaaS. The experiment is performed on our Scallop4SC prototype, which is a Hadoop cluster comprising 9 nodes (Intel(R) Corei7-3770, 32GB, CentOS-x64). Libraries used for Hadoop/MapReduce are `hadoop-core-1.0.4.jar` and `hbase-0.94.7.jar`. The raw data used in experiment is *power consumption logs* gathered in our smart home environment CS27-HNS [10]. The consumption logs have been taken every 3 seconds from 32 devices, for over two years. For one day, the logs comprises 921,600 data rows (= 20 items x 60 minutes x 24 hours x 32 devices).

From the raw data, we create three kinds of materialized views with MVaaS: *DailyView*, *HourlyView* and *MinutelyView*. They store the total power consumption of each device for every day, hour and minute, respectively.

In the experiment, we first measure the time taken for MVaaS to create each materialized view. We then measure the response time that an application obtains a data item from a materialized view through MVaaS API. For comparison, we also measure the execution time of the conventional method without MVaaS, where the application itself directly obtains raw data and calculates the total consumption.

B. Result of Experiment

Table II shows the result. We can see that MVaaS took about 7 minutes to create a DailyView for converting 921,600 rows by MapReduce. Once the view is created, the application can access any data in the view very quickly (2 msec). On the other hand, the conventional method without MVaaS took 5 minutes for *each* calculation of the daily consumption. Thus, we can see that the creation of a view is efficient especially when the required raw data is large and the application frequently accesses the aggregated data.

On the other hand, when the size of raw data to be aggregated is small, the overhead of MapReduce becomes dominant in the total execution time. As seen in the result of MinutelyView, the response time without MVaaS is just 0.2 second, which might be acceptable for some applications. In such cases, using the conventional method is one possible choice to avoid overhead of creating a materialized view.

Table II
Result of Experiment

Materialized View	Daily	Hourly	Minutely
# of rows aggregated	921,600	38,400	640
Time for MV creation (sec.)	412.7	52.2	34.7
Resp. time with MVaaS API (sec.)	0.002	0.002	0.002
Direct calculation w/o MVaaS (sec.)	328.022	13.682	0.260

V. Conclusion

In this paper, we proposed a materialized as a service (MVaaS) for large-scale house log in a smart city data platform Scallop4SC. Using MVaaS, a developer of a smart city application can easily construct an application-specific view, by which the application can access the necessary data efficiently. For a given data specification, MVaaS dynamically generates a MapReduce program that converts the raw data into the view. The program is then executed on Hadoop of Scallop4SC, and published as a service with API. We have evaluated the proposed method through case studies and performance evaluation.

Acknowledgments

This research was partially supported by the Japan Ministry of Education, Science, Sports, and Culture [Grant-in-Aid for Scientific Research (C) (No.24500079), Scientific Research (B) (No.23300009)].

References

[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 *In International Conference on Cloud Computing Technology and Science (CloudCom2012)*, December 2012, pp. 141–148.

[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 *In International Workshop on Cloud Computing for Internet of Things (IoT-Cloud2012)*, December 2012, pp. 815–820.

[5] Y. Ise, S. Yamamoto, S. Matsumoto, S. Saiki, and M. Nakamura, in *International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD 2013)*, 2012 (to appear).

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

[7] S. Massoud Amin and B. Wollenberg, "Toward a smart grid: power delivery for the 21st century," *Power and Energy Magazine, IEEE*, vol. 3, no. 5, pp. 34–41, 2005.

[8] E. Brockfeld, R. Barlovic, A. Schadschneider, and M. Schreckenberg, "Optimizing traffic lights in a cellular automaton model for city traffic," *Phys. Rev. E*, vol. 64, p. 056132, 2001.

[9] Y. Cho, J. Moon, and H. Yoe, "A context-aware service model based on workflows for u-agriculture," in *International Conference on Computational Science and Its Applications (ICCSA2010)*, vol. 6018, 2010, pp. 258–268.

[10] M. Nakamura, A. Tanaka, H. Igaki, H. Tamada, and K. Matsumoto., "Constructing home network systems and integrated services using legacy home appliances and web services," *International Journal of Web Services Research*, vol. 5, no. 1, pp. 82–98, 2008.