A RELATIONAL DATABASE AND KEY-VALUE STORE COMBINED MECHANISM FOR MASSIVE HETEROGENEOUS SENSOR DATA MANAGEMENT

Zhiming Ding [1], Qi Yang [2] and Limin Guo[1]
[1]*Institute of Software, Chinese Academy of Sciences, South Fourth Str. 4, Zhongguancun, Beijing 100190, P.R.China*
[2]*National Center of ITS Engineering & Technology, Xitucheng Road 8, Beijing 100088, P.R.China*

Keywords: Sensor Data Management, Spatial-Temporal Data, Massive Data Processing, Cloud Data Management.

Abstract: Massive sensor data management is an important issue in large-scale sensor based systems such as the Internet/web of Things. However, existing relational database and cloud data management techniques are inadequate in handling large-scale sensor sampling data. On the one hand, relational databases can not efficiently process frequent data updates caused by sensor samplings. On the other hand, current cloud data management mechanisms are largely key-value stores so that they can not support complicated spatial-temporal computation involved in sensor data query. To solve the above problems, we propose a *Relational Data-Base and Key-Value store combined Cloud Data management* ("*RDB-KV CloudDB*") framework, in this paper. The experimental results show that the RDB-KV CloudDB can provide satisfactory query processing and sensor data updating performances in large scale sensor-based systems.

1 INTRODUCTION

Massive sensor data management is an important issue in large-scale sensor based systems such as the Internet/web of Things. However, existing relational database and cloud data management techniques are inadequate in handling large-scale centralized sensor sampling data. On the one hand, relational databases (Güting, Almeida, and Ding, 2006) and middle-ware systems (Gurgen, Roncancio, Labbé, et al., 2008) can not efficiently process frequent data updates caused by sensor samplings. On the other hand, current cloud data management mechanisms (Abadi, 2009) are largely key-value stores so that they are not suited for sensor data management. Detailed reasons are as follows:

(1) Sensor data are highly heterogeneous since there could be various kinds of sensors even in a same sensor-based system. Different kinds of sensors can have different semantics and data formats, and it is important to keep the semantics of the data for querying and for interoperation.

(2) Spatial-temporal attribute is intrinsic for sensor data. Every sampling value corresponds to a sampling location and a sampling time, which are crucial information for query processing in sensor-based systems. In a lot of cases, data are queried through spatial-temporal constrains and other complicated conditions, not through keyword searches on identifiers of sensors.

(3) More information about a monitored object is contained in the sequence of its sampling values than in individual sampling values. Queries about the state of the monitored object can not be answered simply through keyword matches, since data are sampled discretely and the chances for the querying time to coincide with the sampling time are very low. To answer queries correctly and efficiently, we need to organize the sampling data of the same object into a sequence and answer the queries through interpolation.

(4) The sensor sampling data to be managed are dynamically changing stream data. The data are always changing because of insertions of new sampling values and deletions of obsolete sampling values.

To support massive heterogeneous sensor data management, we propose a *Relational DataBase and Key-Value store combined Cloud Data management* ("*RDB-KV CloudDB*") framework, in this paper. The overall architecture of the RDB-KV CloudDB is depicted in Figure 1.

Ding Z., Yang Q. and Guo L..
A RELATIONAL DATABASE AND KEY-VALUE STORE COMBINED MECHANISM FOR MASSIVE HETEROGENEOUS SENSOR DATA MANAGEMENT.
DOI: 10.5220/0003800301510154
In *Proceedings of the 1st International Conference on Sensor Networks* (SENSORNETS-2012), pages 151-154
ISBN: 978-989-8565-01-3

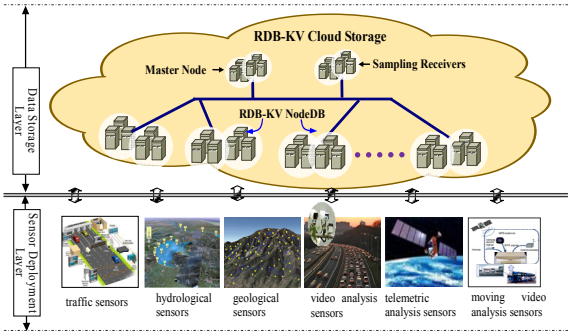

Figure 1: Architecture of the RDB-KV CloudDB.

The RDB-KV Cloud Storage adopts a two-layered architecture. The actual sensor sampling data are stored at the leaf nodes, while the master node stores global indices and global catalogues of the whole system. The cloud can not only support keyword searches, but also support spatial-temporal queries, sampling sequence based interpolation queries, and other complicated queries.

Each leaf node of the RDB-KV Cloud Storage is a database (denoted as "RDB-KV NodeDB" in Figure 1) which can manage heterogeneous sensor data in a uniformed manner through a set of data types and operators.

The root nodes of the RDB-KV Cloud Storage include a coordinator (denoted as "Master Node" in Figure 1, which is a RDB-KV NodeDB augmented with global indices and catalogues) for global query processing, and a group of servers for receiving and distributing new sensor sampling values (denoted as "Sampling Receivers" in Figure 1).

The system has global distributed indices built on RDB-KV NodeDBs and the master node so that both SQL queries and keyword searches can be supported efficiently.

2 RDB-KV NODE DATABASE

In this section, we describe how each leaf node of the RDB-KV Cloud Storage works to manage heterogeneous sensor data. In an RDB-KV NodeDB, multiple sampling values of a same object form a "sampling sequence". Through related data types, operators, and indices, the sampling sequence data can be managed and queried at the database kernel efficiently.

In describing the data types and operators of the RDB-KV NodeDB, we use the notation introduced in (Güting, Almeida, and Ding, 2006) and assume that the standard data types, spatial data types, and their related operators have already been designed and implemented.

2.1 Data Types for Expressing Sensor Data in Databases

In this subsection, we first define the data types for expressing individual sensor sampling values, and then define the sampling sequence data type, which is the general format for organizing and managing the sampling data. Through these data types, the database can express and present heterogeneous sensor sampling values in a uniformed manner.

Definition 1 (Sampling Value) A sampling value, denoted as *SamplingValue*, is defined as:

$$SamplingValue = (t, (x, y), npos, schema, value)$$

where t is the time instant when the value is sampled; (x, y) and $npos$ are the Euclidean position and the network position where the value is sampled; *schema* and *value* are the format and the actual value of the sampling respectively.

Through the *SamplingValue* data type, we can express heterogeneous sensor sampling data in a uniformed manner.

A sampling value can have multiple components. For instance, a GPS sampling value can have 2 components: longitude and latitude. Components of a sampling value can be expressed through the Sampling Value Component data type.

Definition 2 (Sampling Value Component) A sampling value component, denoted as *SamplingComponent*, can be defined as follows:

$$SamplingComponent = (cSchema, cValue)$$

where *cSchema* and *cValue* are the schema and the value of the component respectively.

Definition 3 (Sampling Sequence) For a certain monitored object, its sampling sequence is composed of all sampling values of the object for a certain time period, ordered by sampling time. Since the sampling values of the same object share the same schema, we can define the sampling sequence data type as follows:

$$SamplingSequence=(schema, (t_i, ((x_i, y_i), npos_i, value_i, flag_i))_{i=1}^{n})$$

where *schema* describes the format of the sampling values; t_i, (x_i, y_i), $npos_i$, $value_i$ are the time, the Euclidean position, the network position, and the actual value of the ith sampling respectively; and $flag_i$ indicates whether the ith sampling value is a "breaking point" in the sequence.

For static objects whose positions do not move, their sampling sequence can be further simplified as

follows ($flag_i$ is still needed to indicate the situation when the object is temporarily suspended):

SamplingSequence = (*schema*, (*x*, *y*), *npos*, (t_i, $value_i, flag_i)_{i=1}^{n}$)

The above two formats for sampling sequences are a little bit different and the database can differentiate them automatically.

With the above data types, we can create tables for storing sensor sampling data. For instance:

Create Table IoTData
(ObjectID: String, ObjectType: String,
Owner: String; DepoyedTime: Instant,
Samplings: SamplingSequence)

2.2 Sensor Querying Operators

The data types allow us to express sensor sampling data in databases. To query the sensor data, we need to define a set of operators based on these data types. The most important operator based on the *SamplingSequence* data type is the **atInstant** operator which computes the state of the monitored object at a given time instant. The signature of the operator is as follows:

atInstant:
SamplingSequence × *Instant* → *SamplingValue*

The operator has two input arguments which are of *SamplingSequence* and *Instant* data types respectively, and outputs a value of the *SamplingValue* data type. During the computation, interpolation may be needed so that the database can answer queries about the states of monitored objects at any time during the monitored time periods.

To facilitate the appending of new sampling values to sampling sequences, we define the **samplingAppend** operator with the following signature:

samplingAppend:
SamplingSequence×*SamplingValue*×*Bool*→ *SamplingSequence*

The operators based on the *SamplingValue* data type mainly include 3 data projection operators, **getInstant**, **getPosition**, **getNetPosition,** and a data extraction operator, **getComponent**. Their signatures are as follows:

getInstant: *SamplingValue* → *Instant*

getPosition:
SamplingValue → *Point*
SamplingSequence → *Point* (for static objects)

getNetPosition: *SamplingValue* → *String*

getComponent:
SamplingValue×*integer* → *SamplingComponent*

Besides, we upgrade all the relevant standard operators (such as +, −, ×, /, <, =, >) and spatial operators (such as **inside**, **intersect**, **touches**, **distance**, **direction**, **overlap**) through "lifting" (Güting, Almeida, and Ding, 2006) so that the *SamplingComponent* data type can interoperate with other standard and spatial data types through these operators. For instance, the "=" operator can be "lifted" as follows (assume that "BASE" and "SPATIAL" are the sets of standard data types and spatial data types respectively):

=: $\alpha \times \beta \rightarrow$ *Bool*

where α, $\beta \in$ {*samplingComponent*} ∪ BASE ∪ SPATIAL.

In the RDB-KV NodeDB, queries are submitted in the SQL format no matter whether they are normal SQL queries or keyword searches. For keyword searches, we define the following operator:

keySearch: *string* → set(*tuple*)

With the above operators, we can make various kinds of queries in the SQL format. Let's see some examples.

3 RDB-KV CLOUD DATA MANAGEMENT FRAMEWORK

In a sensor-based system, there could be huge numbers of sensors monitoring the states of various kinds of objects. To manage the sensor sampling data efficiently, we need large numbers of RDB-KV NodeDBs to work together and to form an RDB-KV Cloud Storage system. The RDB-KV Cloud Storage assumes a two-layered structure, with the leaf nodes storing real sensor data and the master node storing global indices and catalogues for global query processing. The architecture of the RDB-KV Cloud Storage is shown in Figure 2.

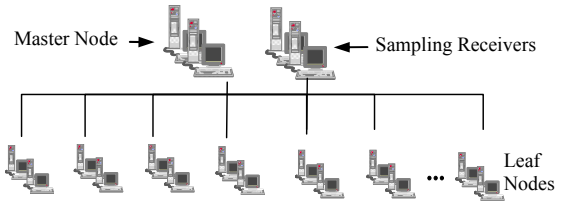

Figure 2: Architecture of the RDB-KV Cloud Storage.

In the RDB-KV Cloud Storage, both the master node and the leaf nodes are RDB-KV NodeDBs, with the master node having some additional modules for global indexing and query processing.

3.1 Data Distribution Strategies

In the RDB-KV CloudDB, all new sampling values are sent to the Sampling Receivers (see Figure 2) first, and then are distributed among leaf nodes according to their geographical attributes. In the system, each leaf node *site* corresponds to a certain area (called the "service area" of the node, denoted as $\alpha(site)$), and data are distributed among the leaf nodes according to their geographical attributes. The master node and the sampling receivers keep the *Service Area Partition Table (SAP-Table).*

For an arbitrary object *obj*, if it is a static object, then it corresponds to only one tuple which is saved at the leaf node whose service area covers the location of *obj*. If *obj* is a moving object, then it corresponds to multiple replicated tuples − The basic information (including attributes other than "Samplings", that is, ObjectID, ObjectType, Owner, DeployedTime, see Subsection 2.1 for the schema of the IoTData table) are replicated among all leaf nodes through whose service areas *obj* has travelled, while its "Samplings" attribute is partitioned and kept in a distributed manner among these nodes. Figure 3 shows how the "Samplings" attribute of *obj* is distributed

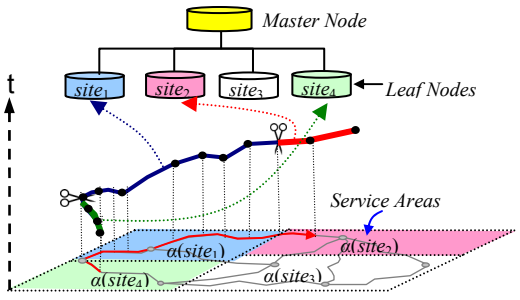

Figure 3: Distribution of the "Samplings" value.

3.2 Global and Local Indices

In sensor-based systems, queries about individual sensors can be roughly divided into three categories: keyword queries, spatial-temporal queries, and value-based queries.

The first kind of queries can be supported efficiently through the *Global Keyword B-Tree Index* which indices the keywords extracted from the database records. The index is a distributed one which involve all the leaf nodes of the RDB-KV cloud to organize the (*keyword*, *siteID*) pairs.

For the second kind of queries, we need the *Sensor-Sampling-Sequence Spatial-Temporal Tree* (S4T-Tree) to index the spatial-temporal attribute of the sampling sequence data. S4T-Tree actually consists of two trees, a spatial R-Tree which indices the locations of static objects and a *Grid-Sketched Spatial-Temporal R-Tree* (GSSTR-Tree) which indices the time-dependent locations (or "trajectories") of moving objects whose locations change over time.

The third kind of queries is very important in real-world applications. For instance, "query all the sensors whose temperatures are above 40 °*C* at time *t*". To support this kind of queries, we need to build a *Grid-Sketched Value-Temporal R-Tree* for every kind of sensors. The records of the index are shown in Figure 4. From the figure we can see that the *Grid-Sketched Value-Temporal R-Tree* can greatly reduce the number of records and the updating frequency in indexing sampling sequences.

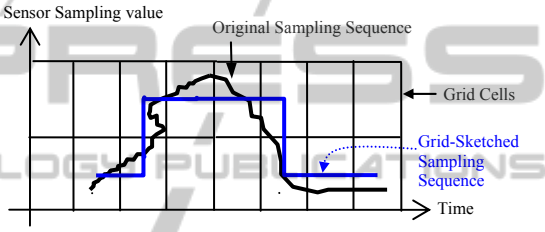

Figure 4: Grid-Sketched Value-Temporal R-Tree.

4 CONCLUSIONS

In this paper, the RDB-KV CloudDB mechanism is proposed which can support spatial-temporal queries, value-based queries and keyword queries efficiently.

ACKNOWLEDGEMENTS

The work is supported by NSFC under grant numbers 91124001 and 60970030.

REFERENCES

Güting R. H., Almeida V. T., Ding Z., 2006. Modeling and Querying Moving Objects in Networks, *The VLDB Journal*, 15(2), pp.165-190.

Gurgen L., Roncancio C., Labbé C., Bottaro A., Olive V., 2008. SStreaMWare: a service oriented middleare for heterogeneous senser data management. In *ICPS'08, The 2008 International Conference on Pervasive Services*, Sorrento, Italy, 6-10 July 2008. New York: ACM.

Abadi D. J., 2009. Data Management in the Cloud: Limitations and Opportunities. *IEEE Data Engineering Bulletin*. 32(1), pp 3-12.