2012 IEEE International Conference on Green Computing and Communications, Conference on Internet of Things, and Conference on Cyber, Physical and Social Computing

A Storage Solution for Massive IoT Data Based on NoSQL

Tingli LI[1,2], Yang LIU[1,2], Ye TIAN[1], Shuo SHEN[1], Wei MAO[1]
[1]Computer Network Information Center, Chinese Academy of Sciences
[2]University of Chinese Academy of Sciences
4, South 4th Street, Zhongguancun, Haidian district, Beijing 100190, China
{litingli, liu-yang, tianye, shenshuo, mao}@cnnic.cn

***Abstract*—Storage is an important research direction of the data management of the Internet of Things. Massive and heterogeneous data of the Internet of Things brings the storage huge challenges. Based on the analysis of the IoT data characteristics, this paper proposed a storage management solution called IOTMDB based on NoSQL as current storage solutions are not well support storing massive and heterogeneous IoT data. Besides, the storage strategies about expressing and organizing IoT data in a uniform manner were proposed, some evaluations were carried out. Furthermore, we not only just concerned about descripting the data itself, but also cared for sharing of the data, so a data sharing mechanism based on ontology was proposed. Finally, index was studied and a set of query syntaxes to meet the needs of different kinds of IoT queries based NoSQL was given.**

Keywords-Internet of Things; data storage; massive data; heterogeneous data; NoSQL

I. INTRODUCTION

The Internet of Things (IoT) is the network that connects objects to the Internet through kinds of information perception devices, so that all the ordinary physical objects which can be individually addressed are able to exchange information with each other, and ultimately achieve the goal of intelligent recognition, locating, tracking, monitoring and management.

Data is one of the most valuable aspects of the IoT. In the environment of the IoT, data is from different kinds of sensors and represents billions of objects. By and large, the IoT data shows following features:

1) *Multi-source and Heterogeneity* IoT data is sampled by various of perception devices such as RFID (Radio Frequency Identification) readers, cameras, temperature sensors etc. The data from different devices and standards has different semantics and structures.

2) *Huge scale* The Internet of Things contains a large number of perception devices, these devices' continuously and automatically collecting information leads to a rapid expansion of data scale.

3) *Temporal-spatial correlation* Different from the traditional Internet data, the data from the Internet of Things has the time and space attributes inherently to describe the dynamic state changes of an object's location over time.

4) Interoperability In spite of the fact that most of the IoT applications now are isolated, the IoT will eventually have to achieve data-sharing to facilitate collaborative work between different applications. Take telemedicine service as an example, when a patient is in emergency, in addition to physiological data, the traffic information is also needed to assess the arrival time of the ambulance to decide what kind of first-aid plan to take.

5) Multi-dimensional Now an IoT application usually integrates several sensors to simultaneously monitor a number of indicators, such as temperature, humidity, light, pressure, etc. And thus the sample data is usually multidimensional.

There are three main forms of IoT data storage [1-3]: *local*, *distributed* and *centralized*. The *local* form indicates that the data sampled by a sensor is stored in the local storage unit of it. (The concept sensor in this article refers to all kinds of IoT perception devices such as RFID readers, temperature sensors, GPS sensors etc.) The *distributed* form means that the data is stored in some nodes in a network through distributed technologies; intermediary mechanisms are used to ensure the access to the data. The *centralized* form refers to that the data of a network is collected by a node and then is sent to and stored in a data center. Because of the limited storage capacity and the constrained power of the sensors, local and distributed forms are not suitable for the IoT applications which need huge-scale data and intensive queries. Furthermore, these two methods are not convenient to share data between different applications [4].

Consequently, more and more researches concentrate on the centralized form. To meet the demands of huge and complex IoT applications, our work also focuses on this kind of form. We designed a storage solution named IOTMDB based on NoSQL [5] (Not Only SQL), which has been successfully applied to the Internet to cope with big data, to solve the storage and management problems of massive IoT data.

The reminder of this paper is structured as follows. In sections II we reviewed the related work of current IoT data storage solutions and NoSQL. Section III displayed the architecture design of the IOTMDB. The data sharing mechanism and storage strategies were given in section IV and V respectively. Section VI gave a set of query syntaxes based NoSQL. The conclusions are drawn in section VII.

II. RELATED WORK

A. IoT Data Management

In the aspect of storing/managing massive WSN and RFID data, a lot of work has been done. Gonzalez [19] proposed a model called RFID-Cuboids to store massive RFID data. Yu [20] proposed a one-to-one model and a many-to-many model for

978-0-7695-4865-4/12 $26.00 © 2012 IEEE
DOI 10.1109/GreenCom.2012.18

storing WSN data. For the purpose of managing heterogeneous data from different devices, a scheme of IoT data management based on SOA was proposed in [6]. But the efficiency of the application will be reduced because of the SOA. Besides, the major shortcoming of the above solutions is that they are only for certain data format and they lack systematicness.

Recently, there is a little work has focused on systematically dealing with challenges from massive IoT data. Ding [7] proposed a massive IoT data oriented framework to support massive IoT data management. However, the core of Ding's solution is RDBMS (Relational Database Management System), though the *join* operation is avoided as all the data is stored in one table, the concurrency is not well supported because of the lock mechanisms adopted by the RDBMS. Therefore, the performance of this solution is poor, especially compared to NoSQL-based solutions [15,16]. Besides, data-sharing is essential between different application fields in the IoT, but the data representation method of Ding's solution or other solutions only focus on describing the data itself without considering how to sharing data. An open database system should satisfy both rather than only describing the data itself in isolation.

B. *NoSQL*

The key function of NoSQL is storing and managing unstructured data in the form of key-value[5,8]. One of the biggest differences between NoSQL and RDBMS is that NoSQL systems separate data storage and data management, while the RDBMS (Relational Data Management System) tries to satisfy both. NoSQL systems are schema-free. The data stored in NoSQL systems doesn't need to have a pre-defined data model nor need to fit well into relational tables. This feature entrusts storing heterogeneous data with greater flexibility. NoSQL mainly has consideration in BASE (Basic availability, Soft state, Eventual consistency) rather than ACID (Atomicity, Consistency, Isolation, Durability). Therefore, it typically does not maintain complete consistency across distributed servers to gain high performance at the expense of high consistency.

Currently there are several NoSQL systems available. Widely used open source systems are MongoDB, CouchDB, HBase, Redis and Cassandra; widely used non-open source systems are Google's BigTable and Amazon's Dynamo.

In [9] a query processing system in WSN based NoSQL was developed, but the work is somewhat rudimentary. It just focused on WSN rather than the Internet of Things, and how to store massive IoT data as well as how to effectively organize and manage the data was not considered in the article.

Based on what we have analyzed above, there are still some shortcomings of current solutions to deal with the problems of storage and management of IoT data. Further work is needed to meet its demands.

III. System Architecture

In the aspect of centralized data processing, parallel database [10, 11] technology has been widely used. They are able to support massive structured data by organizing several relational stores into a cluster. But the efficiency of this kind of database is considerably inferior to NoSQL systems. Besides, because of the distributed lock mechanism the concurrent degree is tremendously low. Furthermore, traditional SQL-based databases are of schema, it's considerably not flexible. Therefore, parallel databases are unable to effectively support storing and managing massive and heterogeneous IoT data.

There is another great choice for centralized IoT data processing—cloud database [12, 13]. The cloud database is one of the key cloud computing technologies. Now most of the cloud databases are NoSQL databases. As has addressed above, NoSQL databases are able to provide rapid access to IoT data. They also offer advantages for representing and storing spatial-temporal relationships of IoT data. Therefore, we chose NoSQL systems to deal with IoT data storage and management issues.

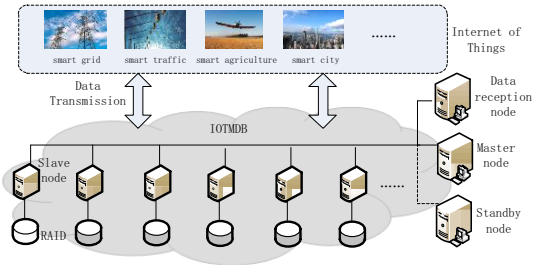

Figure 1. IOTMDB System Architecture

The system architecture of IOTMDB is shown in Fig. 1. In the IOTMDB system, there are four main roles: master node, standby node, data reception node and slave node. The *master node* is the manager of the cluster. Its duty includes accepting connections form clients, maintaining a map between key range and chunks as well as a map between chunks and shards (partitions), monitoring the write request and estimating whether the chunk reaches its max capacity and so on. The *standby node* is used to avoid single point invalidation, users are not aware of it, it's a duplicate of the master node, when the master node fails, it will take over the master node. The *data reception node* is used to receive the data form sensors and does some preprocessing work. The duty of the *slave node* is somewhat simple; it does not concern the management of the entire cluster. All the application data is stored in slave nodes.

We considered this shared-nothing architecture, since it is known to be more scalable than shared-memory or shared-disk models [14]. Sensor data are collected and sent into an IOTMDB system.

IV. Data Sharing Mechanism

Existing storage solutions only concentrate on describing the objects themselves without considering relationships and interoperability between them. This remarkably decreases the

value of the data and the availability of the information systems based on these data. Besides, various data is used to describe one object; the semantic conflict must be solved. We proposed a standard based on ontology to compensate for these defects.

The concept ontology [17] has been widely applied in the field of artificial intelligence, software engineering, and information architecture, etc. By defining shared and common domain knowledge, ontology can help between machines or between machines and people to more accurately communicate and realize semantic exchange rather than just the syntax level interaction. According the theory of ontology, we defined four types of classes in our standard: entity class, field class, activity class and application class.

1) Entity class: Used to describe the information of the entities such as people, cars, microwave ovens, etc. For instance, we regard a shed as an entity, and we define the basic properties of this kind of entity as follows: <Shed: temperature, light, humidity>. Subclasses can extend properties based on it.

2) *Field class:* Used to describe the application fields in IoT such as traffic, logistics and agriculture. The same entities in different fields may play different roles. So, the concerned information about entities in different fields is various. Take people for example, the basic properties are as follows: <People: name, height, weight>. In the medical field, we focus on a person's biometric information, then this entity will be extended as follows: <People: name, height, weight, heartbeat, pulse, blood_pressure>; in the field of logistics, if the person is a messenger, we focus on his communication, so the entity will be extended like this: <People: name, height, weight, mobile, fix_phone, fax>.

3) *Activity class:* Used to describe certain activities such as *buy*, *order*, *fill* (light) and *capture* (a photo).

4) *Application class:* Used to complete specific applications through using *activities* as well as specializing and instantiating *entities* in certain *fields*. An application may consist of several entities and activities.

These classes provide us common understanding about entities in different fields. Take a smart stock management system of a supermarket for example. The system will count the stock of goods according to the sampling data stored in the database every day. Once it finds some kind of goods, say mouse (an *entity* of digital product *field*), are not adequate it will first search (an *activity*) which group of suppliers (*entities* in *field* e-commerce) could provide this kind of mouse, then compare (an *activity*) the prices, and finally order (an *activity*) mouse to a selected digital product supplier. All the processes can be completed by machines automatically. Thanks to the shared information, common understandings and precise definitions, the application is able to complete tasks correctly. For instance, in the above example, the system correctly orders mouse for computers rather than pet mouse.

We use the XML schemas as the data exchange media for the reason that the XML language can be comprehended by both human beings and machines and it has been widely used in different domains to exchange information.

In addition to the standard proposed above, to enable data searching, locating and sharing, we are developing a public service platform called RNS[21] based on DNS for the management of the Internet of Things data. Its architecture is shown in Fig.2. It mainly consists of three parts: register system, resolution system and search system. Users or IOTMDB systems are able to use it to achieve the goal of data locating and exchange. Each time an object is added in to IOTMDB, it will be registered in RNS with the information including its *tid* (things id, such as EPC), *sid* (standard id, to indicate what standard the tid belongs to) and some description information such as filed information. Users are able to use the search system and resolution system to search and locate data.

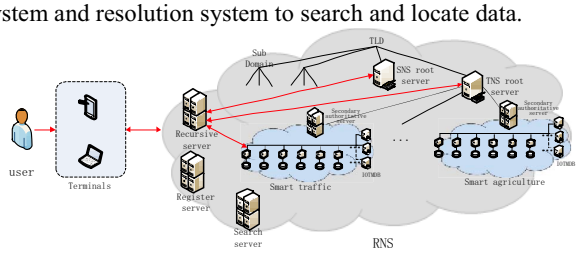

Figure 2. Architecture of RNS

V. STORAGE STRATEGIES

A. Data Preprocessing

Data preprocessing is mainly completed in the data reception node. There are many kinds of applications in the Internet of Things, and consequently the preprocessing demands are different. However, in addition to some particular needs, there are still a lot of common needs. Therefore, we designed a preprocessing mechanism to satisfy both particular needs and common needs.

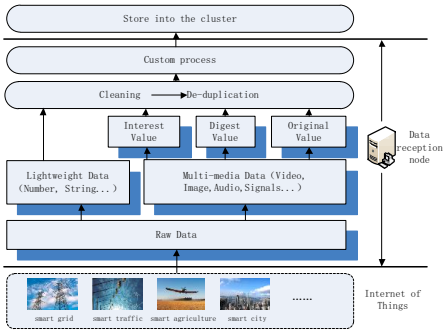

Figure 3. IOTMDB preprocessing

As Fig. 3 shows, the raw data is divided into two categories: light-weight data like numerical data and characters; and multi-media data like videos, images, audios and signals. For these two kinds of data, the processes of preprocessing are different. For further discussions, let's first give some explanations to following terms:

Interest value: the value generated by certain algorithms from the multi-media data. It's the information the application cares for most. For instance, the people flow density at the railway station, the average level of noise near a ward room and the number of traffic violations an hour.

Digest value: also generated by a certain algorithm such as MD5, it's mainly used to briefly describe the multi-media data. Different from the interest value, this value lacks semantic. However, it's useful equally. The benefits and usage of digest value will be discussed in detail below.

After receiving the data, the processing procedure is as follows:

1) Extracting specific information

According to the type of the raw data (lightweight or multi-media), the processing approach will be determined. For the multi-media data, it's essential to extract specific information in advance. The specific information includes the interest value and the digest value. These two kinds of values are also lightweight, so they are beneficial to querying and storing. The *interest value* is the most significant information of the multi-media data, for instance, we usually only pay attention to some key information lies in the multi-media data rather the multi-media data itself, such as the plate number of a car, which violates regulation, from a snap photo. The *digest value* is a uniform way to briefly describe the multi-media data and is beneficial to de-duplicating and retrieving. For example, if two images are the same, the digest values will be the same. Therefore, when retrieving an image we just need to compare the digest values. In sum, these two types of values dig out the value of multimedia data, reduce the amount of storage in a record and increase the speed of queries.

2) Cleaning

Data from the real world is normally incomplete, of noise or even invalid. Data cleaning tries to ensure the accuracy of the data by filling missing values and smoothing the noise. [18] stated several data cleaning methods. Under the circumstance of huge amount of data is sent to reception node frequently, the algorithm should be simple and effective. For missing values, if the application allows the occurrence of them, we use a global constant such as "unknown" or "-∞" to fill the values; Otherwise, we use the average value of last N values to fill. For noise, we can use the method named binning, which take the values of the surrounding neighborhoods to locally smooth the noise. These two kinds of methods are simple, easy to realize and effective.

It should be noted that in the environment of the IoT, a sudden change of the sample value is normal and therefore we should carefully design the algorithm to avoid regard the sudden change value as noise. So, we only smooth the invalid values, if the value is in normal range we do nothing.

3) De-duplicating

One of the causes of massive IoT data is the high sampling frequency. This results in huge amount of repeat or remarkably similar values. As Fig. 4 shows, a simple method to deal with repeated values is to set a threshold to determine whether a value should be accepted. If the difference between the current value and the last accepted value is not larger than the threshold, it should be abandoned; and otherwise, it should be accepted.

However, the method above is only suitable for one-dimensional data. Multidimensional IoT data makes the issue considerably complex. For example, in one record if one dimension is judged to be accepted, but other dimensions are judged not to be accepted, then the problem that whether this record can be accepted comes. It's hard to decide. So, we use a simple method to cope with the de-duplication of multidimensional data. That is if the change of any dimension of a record is larger than the threshold, the whole record should be accepted. And we use another method to reduce repetitive data--block-level de-duplication.

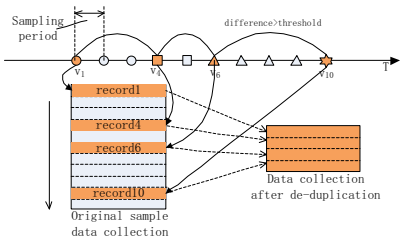

Figure 4. Process of de-duplicating

4) Custom processing.

The data goes through the former processes will be transmitted to custom processing module. This module enables users to process the data according to their own specific needs such as normalizing numerical elements to some value or reducing the dimensions of a record to decrease the scale of the data. It mainly needs users to implement by programming; otherwise, it will do nothing to the data. For instance, we could use AOP (Aspect-Oriented Programming) to implement this module and benefited by this kind of programming we are able to add or delete our functions to the module dynamically without rebuilding the original database code and restarting the running program.

B. Data expression

Based on the data sharing standard, this section will discuss what information should be stored and how to express the data.

The smallest unit of IOTMDB is *SampleElement*, its definition is shown as follows:

$$SampleElement = < key: value > \quad (1),$$

SampleElement is a ordered key-value pair, the $key \in String$ equals to the name of the *value* and the $value \in String \cup Number$ is the actual sample value. Some examples of

SampleElement are <temperature: 50>, <nPeople: 121> or <audioText: “Hello world”>.

The basic unit of the data stored in IOTMDB is called *SampleRecord* which is a set of SampleElement. As the Fig. 5 illustrates, a SampleRecord consists of two parts of information: *static information* and *dynamic information*. Static information is the invariable information about the object such as object ID, the field the object belongs to, etc. Dynamic information is sampling values which describe the state change of the object such as the sampling time, location, speed, or ozone concentration, etc. The SampleRecords belong to the same application are organized in a *collection* and a collection may contains data of different objects of an application. The collection can be regarded as a table in a RDBMS system, and a SampleRecord can accordingly be seen as a row in the table. Although they look much like each other, they are intrinsically different.

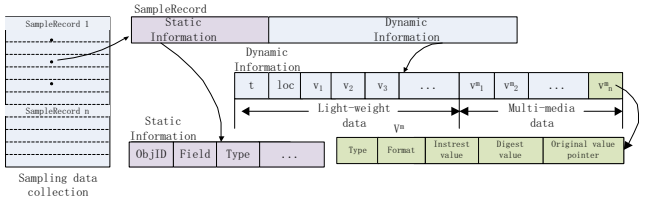

Figure 5. Data expression in IOTMDB

The definition of *SampleRecord* is shown as follows:

$$SampleRecord = ((objID, field, type\ \ldots), (t, loc, (v_i)_{i=1}^{n}, (v_j^m)_{j=1}^{x}) \quad (2),$$

where *objID* $\in$*String* is the ID of the object; *field*$\in$*String* is field of the object belongs to, such as agriculture and logistics, for effectively data sharing *field* is obligatory; $t \in$*Instant* is the time when the data was sampled; *loc*$\in$*Point* is the location where the object was when it was sampled, in most cases it equals to the location of the sensor; *Point* is a 2d value like (x,y), and we allow users to define their custom space; $v \in String \cup Number$ is the value of the lightweight data; $v^m \in SampleMedia$ is the values of the multimedia data. In order to make better use of the multi-media data, we have extracted specific information from it in preprocessing and consequently the definition of *SampleMedia* is as follows:

$$SampleMedia = (type, format, v_{interest}, v_{\text{digest}}, p_{original}) \quad (3).$$

Where both *type* and *format* $\in$*String*; $v_{interest} \in String \cup Number$ is interest information, it can be a single value or a set of values like a array; $v_{\text{digest}} \in String$ is the digest information; $p_{original}$ is a pointer to the original multi-media data.

Through the SampleRecord we are able to express the heterogeneous data in IOTMDB in a uniform way; here are two examples in Table I.

Example 1 is the data about a shed, the number 50, 0.56, and 23.1 are the sampling values of light, humidity, temperature respectively; location (40.1, 20.5) is a GPS value. Example 2 is of speeding monitoring; (123, 1024) is a custom point of a district rather a GPS value; 210 is the speed of the car and BJ0A435, which is the interest value extracted from the photo, is the plate number of the car; *b38767a34dd2d764c0d597986010e5b1* is the digest value; p1 is the pointer to the original data of the image.

TABLE I. EXAMPLES OF SAMPLERECORDS

NO.	SampleRecord
1	((objID:”a01”, field:”agriculture”, type:”shed”), (t:t1, loc: (40.1,20.5), (light:50, humidity:0.56, light:23.1))
2	((objID:”b123”, field:”traffic”, type:”car”), (t:t2, loc:(123,1024), speed:210, capturePhoto:(type:”image”, format:”jpeg”, plateNumber:”BJ0A435”, digestValue:” *b38767a34dd2d764c0d597986010e5b1”,* originalValue:*”*p_1”)))

C. *Data distribution*

1) *Sharding*

In IOTMDB，as a result of the distributed architecture, sharding is inevitable. As shown in Fig. 6, sharding refers to the process that distributing the data to different machines. Sharding enables the cluster to store considerably more data and handle greater load without powerful machines. An IOTMDB system consists of several nodes and thus the cluster will be divided into several logical shards. A machine can maintain either one or several shards according to the needs.

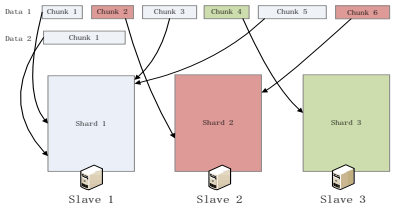

Figure 6. Data Sharding

Sharding provides us more space to store IoT data and help us to improve the performance of the cluster. But does it always make sense to save time and obtain a higher performance? Generally, under the circumstance of sharding, the time T_q of gaining result of a query is shown as the following formula:

$$T_q = 2 * T_t + T_p + \frac{T_s}{N_p} + T_a \quad (1)$$

T_t is the transmission time between the client and the server; T_p is the time of parsing the query command and sending to slave nodes in parallel; T_s is the execution time of the query command in a single node; T_a is the time of aggregating results from different nodes, it’s proportionate to the number of the nodes; The N_p is the number of the nodes to perform tasks in parallel. In general, compared to T_s and T_a, T_t and T_p can be ignored as they are noticeably less than T_s and T_a. So T_q approximately equals to the sum of T_s/N_p and T_a. Because of the parallel

execution, we will save the time of the amount of $\frac{N_p-1}{N_p} * T_s$. Consequently, if T_a is larger than $\frac{N_p-1}{N_p} * T_s$ we will cost more time to obtain the result than that of non-parallel execution.

Therefore, we can draw the conclusion that not all data is suitable for sharding and we need to carefully design the data logical distribution to ensure high performance. In order to verify this idea, we carried following performance test: under the circumstance of sharding and no-sharding we tested update time at different scale of data (1w (1w=10,000), 5w, 10w, 100w, 200w, 500w records). The testing environment parameters are illustrated as follows: the cluster is of 3 nodes; each node has 1 CPU and 1G memory; NoSQL database is MongoDB-2.0.4; each node is a shard.

The results are shown in Fig. 7 and Table II. When the data scale is small (1w-10w records), the update time of no-sharding is considerably shorter than that of sharding. But as the data scale increases, the difference decreases. The results of the tests bear out our conclusion above. Therefore, we can conclude that if some IoT applications' data scales are somewhat not big, they'd be better stored in one shard. Otherwise, they should be partitioned to get more space and maintain performance of the cluster.

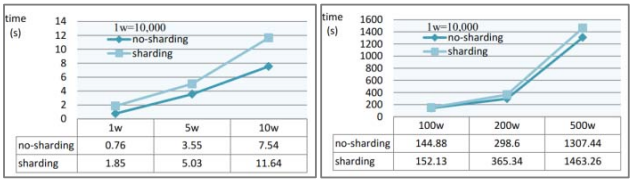

Figure 7. Update time of small-scale/large-scale data

TABLE II. RATIO OF SHARDING AND NO_SHARDING UPDATE TIME

no_sharding / sharding	1w	5w	10w	100w	200w	500w
	0.41	0.71	0.65	0.95	0.82	0.89

2) Storing multi-media data separately from lightweight data

The multimedia data acquires the following characteristics: 1) require far larger space to be stored than the lightweight data; 2) require considerably high bandwidth to be transmitted; 3) the knowledge is not unintuitive to machines, for example, we human beings are able to know there are how many persons in a photo, but for machines they only regard the photo as a set of binary values.

For the first two characteristics, it means that the multi-media data will cost large amount of resources and the access to it will cause a high I/O load. Consequently, if we store multi-media data with the lightweight data in the same machines the performance of the entire cluster will be remarkably decreased.

In order to improve the performance, in IOTMDB the multi-media data should be stored separately from lightweight data. The original value of multi-media data should be stored in another cluster. Absolutely, the users are not aware of there exists another cluster. As has stated above, if we want to get the original multi-media data, we use the pointer in a SampleMedia. There are two main advantages to follow this method: 1) more resources are left to deal with client requests rather than transmitting big multi-media data files; 2) we are able to pertinently do some optimization such as compression to lightweight data and multi-media data respectively.

VI. QUERY SYNTAXES BASED ON NOSQL & INDEX

Queries play a major role in a database abstraction. But the existing NoSQL systems are not fully support our requirements. So, we need to define a set of query syntaxes to meet the IOTMDB's demands.

A. Syntaxes

There are mainly three types of query according to the demands of IoT applications [4]: 1) *historical query*, to get the data of one time node or a time sequence; 2) *tracking query*, for example, to get the delivery path of goods of a supply chain; 3) *preference query*, for example, to get the location of nearest hospital when a man is in danger. In order to cope with characteristics of IoT data, we need to design specific queries. We use the signatures proposed in [7].

For *historical query*, the query syntaxes are shown as follows:

getByTimeStamp:
$Collection \times TimeStamp \rightarrow \{SampleRecord_i\}_{i=1}^{n}$

getByTimeRange:
$Collection \times TimeRange \rightarrow \{SampleRecord_i\}_{i=1}^{n}$

The input arguments are a collection and a timestamp or a time range, the result is a set of SampleRecord. If there are no SampleRecords in certain period, the result will be null. Through this kind of query we are able to get historical state of objects.

For *tracking query*, the query syntax is shown as follows:

getTrackingPath:
$Collection \times objectID \rightarrow sort(\{Point_i\}_{i=1}^{n})$

The input arguments are a collection and an ID of an object and the output is a sorted set of Point. Because the location of an object is associated to the time, the set will be sorted according to the order of time by default. And the following operators help us to project out the path information we focus on:

getRecordSet:
$Collection \times key \rightarrow \{SampleRecord_i\}_{i=1}^{n}$

getElement:
$SampleRecord \times key \rightarrow SampleElement$

The operator **getRecordSet** helps to get a set of SampleRecord according to a *key*, in operator **getTrackingPath** the *key* equals to *objectID*. And operator **getElement** helps to

get on element from a SampleRecord, the location information will be obtained if the *key* equals to *loc*.

For *preference query,* we need to know the nearest objects of an object at a moment or the events happened at the same time. The syntax is shown as follows:

getObjsNearPoint:
$Collection \times Point \rightarrow sort(\{SampleRecord_i\}_{i=1}^{n})$

getObjDistance:
$Point \times Point \rightarrow Distance$

This operator aims to get nearest objects of a Point. The result is a sorted set of SampleRecord. And the set is sorted according to the distance of an object to the Point. The shorter the distance is, the higher the object is ranked. The number of the set can be custom. If a user doesn't give the limitation of the number, it will be set to a default value. Operator **getObjDistance** helps to compute the distance between two points. We could design some encoding schemes and indexes to improve the computing performance.

We have defined a special structure to present multi-media data—SampleMedia. Like SampleRecord, it's also a set of SampleElement. We regard it as an embedded set in a set. To get the information in SampleMedia, we define the following operator:

getMultiMedia:
$SampleRecord \times key \rightarrow SampleMedia$

The SampleMedia is also obtained by a key, after get the SampleMedia; we are able to use **getElement** to get a SampleElement.

In addition, there are some routine queries like update, append and delete. For the limitation of the space, the details of these operations will not be dealt with here.

IOTMDB is a NoSQL based solution. Consequently, the query language is totally different from SQL. Through the some NoSQL systems also support SQL-like language; to avoid confusion, we will use the pseudo-code to display some query examples. Another reason we use the pseudo-code is that, unlike the SQL is the uniform language of RDBMS, different NoSQL systems have different query languages; the pseudo-code helps to provide a common understanding.

Example 1: *query all the plate numbers of the cars which speed is greater than 200 km/h from 2012-07-25 to 2012-07-28. The main process is shown as follows:*

```
for each SampleRecord in
set(getByTimeRange(Collection,"2012-07-25","2012-07-28")
AND getElement(speed, SampleRecord).value > 200):
    return getElement(plateNumber,
        getMultiMedia(key, SampleRecord)).value
```

Example 2: *query the delivery path of a package whose id is ID123*

```
for each SampleRecord in
set(sort(getRecordSet(Collection," ID123"), t))
    return getElement(SampleRecord,loc).value
```

Example 3: *query the nearest 10 mobile-phones' location from the GPS position (112.02, 111.01)*

```
for each SampleRecord in Collection:
    Point = getElement(loc, SampleRecord).value
    sort(set(getObjDistance(Point,(112.02,111.01))))
return set.limit(10)
```

B. Index

In the Internet of Things, one of the key characteristics of the sample data is temporal-spatial correlation. A great design of the index will help temporal-spatial queries more efficient. In IOTMDB, the space is not limited in GPS space, custom space is also allowed. Users are able to define space of any size, and also can expand the space according to their actual needs. Consequently, we designed a two-tier B+ tree index.

As shown in left part of Fig. 8, at the first tier the space is divided into different medium-sized space blocks. To enable the space to infinitely expand, the identification number of space blocks adopts spiral coding strategy. As the space expands, the identification number increases. This tier of index records the mapping relation of the space block ID and the points.

At the second tier, the space is divided into smaller units, in the example of right part of Fig. 8 a space block is divided 4 times and results in a 16×16 grid. Absolutely, we are able to divide more times according to our practical demands, such as 10 times and results in a 1024×1024 grid. This tier of index records the mapping relation of the inner block code and the points in one block.

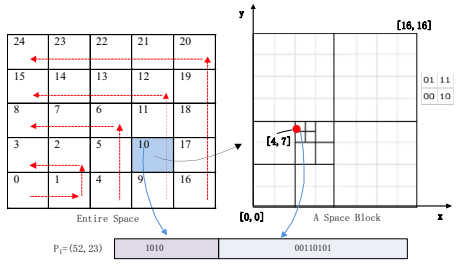

Figure 8. Two-tier index and coding method

Through the method above, a point like (x, y) will be converted to a binary hash value and used to index. As illustrated in Fig. 8, this value consists of two parts. The prefix is the binary value of the space block ID and the suffix is the inner block code. Take point (52, 23) for example. Its space block ID is 10, so the prefix of the hash value is 1010. Adjacent points will be divided into the same block, so the prefix of the hash value will be the same.

The detail of the calculation method of inner block code is given as follows:

1) For the first time, divide the block into 4 sub-blocks and gives each a binary value like follows, so in the first division, the current inner block code is 00:

01	11
00	10

2) Do as the first step N_D-1 times (N_D is the number of division times we have set, in this example N_D equals to 4), and we get the code 11, 01, 01 for the 2nd, 3rd and 4th time respectively.

3) Finally, the complete inner code is 00110101. The header part of the inner block code of adjacent points in a block will be the same or considerably similar.

Therefore, the hash value of the point (52, 23) is 101000110101.

The advantages of this two-tier index are: 1) under the circumstance of massive data, it doesn't need to load the entire index into the memory, we can dynamically load the index of the blocks we need; 2) beneficial to expand the space definitely and flexibly; 3) the two-stage coding method of the hash value makes distance of two points more straightforward and the retrieval more efficient.

VII. CONCLUSIONS

The Internet of Things storage is facing enormous challenges brought by massive IoT data; this paper proposed a storage management solution based NoSQL to solve this problem. NoSQL systems provide us high availability and performance. Based on this we have done some further work to store IoT data according to its characteristics. The main contributions of this paper are as follows:

1) An Internet of Things storage management architecture name IOTMDB based on NoSQL is proposed to meet the needs of IoT data storage. The IOTMDB not only concerns about how to reasonably and effectively store massive IoT data, but also cares for data sharing and collaboration. Combined with a public service platform for the Internet of Things named RNS and the data abstraction based on ontology, we are able to easily search and locate data and finally realizing data sharing between different IoT applications.

2) The IoT data storage strategies are proposed including a preprocessing mechanism to meet both common and specific needs, a unified data expression form and data distribution strategies. These strategies are beneficial to improve the performance of the cluster and store data effectively.

3) A set of query syntaxes based on NoSQL are proposed. The syntaxes are able to meet the different kinds of IoT queries.

In the future, a database model oriented to IOTMDB based on NoSQL will be implemented, and some detailed techniques including data/access balance and compression techniques will be studied. Besides, the work of how to process and analyze the massive IoT data to maximize the value of it will also be concerned.

REFERENCES

[1] Tsiftes N, Dunkels A, "A database in every sensor," Proceedings of the 9th ACM Conference on Embedded Networked Sensor Systems. Seattle, USA. 2011:316-329.

[2] Guo Q, Liu Y, "A Data Storage Method Suitable for WSN," Journal of Shenyang Ligonge University, 2011.10:15-19.

[3] Madden S, Franklin M J, Hellerstein J M, Hong W, "TinyDB: An acquisitional query processing system for sensor networks," ACM Transactions on Database System, 2005, 30(1):122-173.

[4] He Z L, He Y H, "Preliminary Study on Data Management Technologies of Internet of Things," 2011 International Conference on Intelligence Science and Information Engineering. 2011.63:137-140.

[5] R. Cattell, "Scalable SQL and NoSQL Data Stores," Newsletter ACM SIGMOD Record. 2010(Vol. 39, No. 4):12-27.

[6] Fan T R, Chen Y Z, "A Scheme of Data Management In The Internet of Things," Proceedings of IC-NIDC2010. 2010:110-114.

[7] Ding Z M, Gao X, "A Database Cluster System Framework for Managing Massive Sensor Sampling Data in the Internet of Things," Chinese Journal of Computers. 2012(Vol.35, No.6):1175-1191.

[8] R. Hecht, S. Jablonski, "NoSQL Evaluation: A Use Case Oriented Survey," International Conference on Cloud and Service Computing. 2011:336-341.

[9] T.A.M.C. Thantriwatte, C.L. Keppetiyagama, "NoSQL Query Processing System for Wireless Ad-hoc and Sensor Networks," The International Conference on Advances in ICT for Emerging Regions. 2011 : 078-082.

[10] Waas F M, "Beyond conventional data warehousing: Massively parallel data processing with Greenplum Database," Proceedings of the 2nd International Workshop on Business Intelligence for the Real-Time Enterprise (BIRTE 2008). Auckland, New Zealand, 2008:89-96.

[11] Wu H P, Yu H L, Zheng W M, Zhou D M, "A Massive Data Storage and Management Strategy for Online Computer-Assisted Audit System," Chinese Journal of Computers. 2006(Vol.29, No.4):618-624.

[12] Yuriyama M, Kushida T, "Sensor-Cloud infrastructure: Physical sensor management with virtualized sensors on cloud computing," Proceedings of the 13th International Conference on Network-Based Information Systems (NBiS). Takayama, Japan, 2010:1-8.

[13] Lin Z Y, Lai Y X, Lin C, Xie Y, Zou Q, "Research on Cloud Databases," Journal of Software,2012,23(5):1148− 1166.

[14] Kang K D, Basaran C, "Adaptive Data Replication for Load Sharing in a Sensor Data Center," 2009 29th IEEE International Conference on Distributed Computing Systems Workshops. 2009.12:20-25.

[15] Cruz F, Gomes P, "Assessing NoSQL Databases for Telecom Applications," 2011 IEEE Conference on Commerce and Enterprise Computing. 2011:267-270.

[16] MongoDB and MySQL performance testing and results analysis [Online]http://www.ixwebhosting.mobi/2011/09/ 19/2476.html

[17] Yang S M, Xu Y, He Q Y, "Ontology Based Service Discovery Method for Internet of Things," 2011 IEEE International Conferences on Internet of Things, and Cyber, Physical and Social Computing(CPSCom) 2011.104:43-47.

[18] Han J W, Kamber M, "Data mining, concepts and techniques (second edition)," Morgan Kaufmann Press.

[19] Gonzalez H, Han J W, Li Xiao lei, et al, "Warehousing andanalyzing massive RFID data sets," Proc of the 22nd International Conference on Data Engineering. 2006:83-92.

[20] Yu Z C, Xiao B, Zhou S G, "Achieving optimal data storage position in wireless sensor network," Computer the Communications, 2010.33 (1): 92-102.

[21] Tian Y, Liu Y, Yan Z W, Wu S L, Li H Y, "RNS：A Public Resource Name Service Platform for the Internet of Things," unpublished.