This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/JSEN.2015.2483499, IEEE Sensors Journal

MongoDB-based Repository Design for IoT-generated RFID/Sensor Big Data

Yong-Shin Kang, Il-Ha Park, Jongtae Rhee and Yong-Han Lee, *Member*, *IEEE*

***Abstract*—IoT-generated data are characterized by its continuous generation, large amount, and unstructured format. Existing relational database technologies are inadequate to handle such IoT-generated data due to the limited processing speed and the significant storage-expansion cost. Thus, big data processing technologies, which are normally based on distributed file systems, distributed database management, and parallel processing technologies, have arisen as a core technology to implement IoT-generated data repositories. In this study, we propose a sensor-integrated RFID data repository-implementation model using MongoDB, the most popular big data-savvy document-oriented database system now. Firstly, we devise a data repository schema that can effectively integrate and store the heterogeneous IoT data sources such as RFID, sensor, and GPS, by extending the event data types in Electronic Product Code Information Services (EPCIS) standard, a *de facto* standard for the information exchange services for RFID-based traceability. Secondly, we propose an effective shard key to maximize query speed and uniform data distribution over data servers. Lastly, through a series of experiments measuring query speed and the level of data distribution, we show that the proposed design strategy, which is based on horizontal data partitioning and a compound shard key, is effective and efficient for the IoT-generated RFID/sensor big data.**

***Index Terms*— Big Data, EPCIS, IoT, MongoDB, RFID, Sensor, Supply Chain**

I. Introduction

In the last couple of decades, radio-frequency identification (RFID) technology has been widely used in logistics, manufacturing, defense, environment, health care, agriculture, retail, aviation, and information technology. Moreover, the use of the Internet of Things (IoT) enabling technologies, including RFID, sensors, global positioning systems (GPSs), and automated actuators has lately been expanded to production management, factory management, quality management, logistic management, utility management, and inventory management in various industries [1]-[3]. In the manufacturing industry, for example, once an RFID tag is attached to individual parts or products, their location information can be collected in real time, thereby enabling flexible production planning and shipment order placement. Furthermore, if quality issues arise, their causes can be analyzed using corresponding sensor data collected from the relevant manufacturing facility to pinpoint the source of the quality problems. In the food industry, safety management of perishable food has been extended from production to disposal [4]. IoT technologies like RFID sensor tags and GPS have already started to be used for freshness management purposes in food supply chains. For example, monitoring food quality using the real-time sensor and traceability data can support various operational logistic decision makings. Even in the logistics and transportation industry, IoT technologies are used, to keep pace with the international awareness on climate change and environmental issues, in order to fulfill the requirements of "green," "low carbon," and "energy saving" logistics. IoT-generated data on fuel consumption, carbon emissions, and engine idling can be collected and analyzed in real time in order to plan logistics that minimizes carbon emissions.

IoT-generated data, such as RFID and sensor data, is not only constantly generated in real time as the supply chain and the manufacturing processes continue, but also provided in a variety of data formats. In addition, if several billion tags and sensors were connected through the Internet, an unprecedented number of transactions and amounts of data would be generated [5], [6]. An automotive manufacturing traceability system, for instance, has to store hundreds of Giga byte data only for handling 30 components of a single vehicle production line as illustrated in Section IV. A single automobile is composed of around 25,000 parts and the average production cycle time is one minute, meaning that a fully-implemented futuristic IoT-based manufacturing environment, where RFID tags are attached to most of the parts and logistics units, will easily overwhelm the traditional database systems. Furthermore, the databases will be quickly flooded with not only RFID-tracking information but also sensor data such as temperature, humidity, vibration, pressure, and images collected every other few seconds. In this sense, the IoT-generated data in a supply chain are definitely "big data", satisfying the sufficient conditions in terms of volume, velocity, and variety of data.

However, we have many limitations on processing large amounts of unstructured data, such as IoT-generated big data, using existing relational database (RDB) technologies [7]-[9].

This work was supported by the Dongguk University Research Fund of 2014 and the Agriculture Research Center (ARC, 710003-03-1-SB110) program of the Ministry for Food, Agriculture, Forestry and Fisheries, Korea

Y.-S. Kang is with the Nano Information Technology Academy, Dongguk University-Seoul, 82-1, Pil-dong 2-ga, Jung-gu, Seoul, 100-272, Korea (e-mail: yskang@dgu.edu).

I.-H. Park, J. Rhee and Y.-H. Lee are with the Department of Industrial and Systems Engineering, Dongguk University-Seoul, 30, Pildong-ro 1-gil, Jung-gu, Seoul, 100-715, Korea (e-mail: ihpark@dgu.edu, jtrhee@dgu.edu, yonghan@dgu.edu)

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/JSEN.2015.2483499, IEEE Sensors Journal

For example, existing technologies use a "scale-up" system expansion scheme, which replaces existing equipment with higher-performance equipment when the performance degrades due to a significant increase in the amount of data. This approach is, however, not viable because this requires repeating equipment investments, especially in handling the fast-growing big data. To overcome this drawback, Not Only SQL (NoSQL) database technologies, such as HBase, MongoDB, Cassandra, and CouchDB, have been developed. NoSQL technologies can store unstructured data because of their easy data schema-modification capability, and require lower server expansion cost than relational databases because of their "scale-out" scheme compared to the "scale-up" scheme of RDB's. Besides, NoSQL database systems can process massive input and output data efficiently by virtue of the distributed storage and processing approach over the multiple data nodes.

In this study, we propose a sensor-integrated RFID data repository-implementation model that can integrate and store a large amount of IoT-generated RFID/sensor data collected from supply chain processes, and can quickly process and query them. To achieve this goal, firstly we design a MongoDB-based RFID/sensor data repository that can integrate and store RFID and sensor data by referencing event types of Electronic Product Code Information Services (EPCIS) [10] in the EPCglobal network, which is a de-facto standard for RFID information exchange. Secondly, we suggest an effective shard key. A shard is a horizontal partition of data in a database, or the database server in which the partition is located. A shard key is a set of fields to which the partitioning is carried out with respect. Therefore cautious selection of a shard key is a critical decision in maximizing query speed and uniform data distribution over data servers. A combination of fields among RFID/sensor event-data fields as a shard key is selected based on intensive experiments as well as suggestions in literature. Lastly, we verify our suggestions based on simulated data. We generate a huge amount of RFID/sensor event data using a simulation model on a virtual automotive-parts supply chain for our experiments, which verify the sharding performance of the proposed RFID/sensor data repository in terms of the amount of data, the number of clients, and the number of distributed servers.

II. Background

A. NoSQL

The term 'NoSQL' collectively refers to the database technologies which do not abide by the strict data model of relational databases. By sacrificing some of the properties (such as ACID transactional properties) of relational database model, NoSQL databases can achieve higher availability and scalability, which are essential requirements for big data processing. Unlike relational databases NoSQL databases do not need to have a fixed schema with pre-defined data structures and constraints to be finalized in an early stage of database design. In addition, "shared nothing" architecture of NoSQL allows horizontal scaling – replicating and partitioning data over many nodes, consequently achieving stable and fast read/write operations of massive data [11], [12].

TABLE I
CLASSIFICATION OF NOSQL DATABASES [12]

Key-value stores	Document databases	Column-oriented databases
Voldemort,	SimpleDB,	BigTable,
Riak,	CouchDB,	HBase,
Redis,	MongoDB,	HyperTable,
Scalaris,	Terrastore	Cassandra
Tokyo Cabinet		

Strauch et al. [11] classified NoSQL databases into three types – key-value stores, document databases, and column-oriented databases depending on their data models. In a key-value store, all the data instances are stored in the form of key-value pair. Document databases are based on the same key

-value structure as key-value stores, but the values are in the form of a more complex data structures called "documents" such as XML documents. Unlike the key-value stores, document databases generally support secondary indexes and multiple-type/nested documents in a database. Column-oriented databases store data tables as sections of columns of data, rather than as rows of data. This type of database is efficient when the majority of the database operations are of OLAP, which requires intensive column-oriented calculations like aggregation. Unlike row-oriented databases, column-oriented databases can easily add and delete columns. Table I shows representative NoSQL database solutions of each type.

Leavitt [13] pointed out that NoSQL databases will not replace relational databases, which are more mature and already widely installed, in the near future. However, for some specific purposes such as handling unstructured massive data even requiring a high level of scalability, NoSQL databases will be a better choice. Today we have a wide variety of NoSQL database products (of different types as mentioned above) on the market, which are built to fit specific purposes. Therefore, NoSQL databases will have their own niches, which are even expanding rapidly according to growing needs for big data processing in various fields. In the near future, NoSQL proponents and vendors will focus on developing better application compatibility and management tools. As a result, the adoption of NoSQL databases will be expedited. As reported in [14], five NoSQL products are already included in the top providers of operational database management systems in a Gartner's Magic Quadrant report, and MongoDB is the most popular one of them.

Veen et al. [7] compared the read/write performance of an SQL database (PostgreSQL) with NoSQL database (Cassandra and MongoDB) for the sensor data-storage purpose, and concluded that Cassandra is a good choice for relatively bigger sensor data, while MongoDB is for smaller sensor data with higher priority to the writing performance. Li et al. [15] proposed a MongoDB-based data storage architecture with a preprocessing mechanism for raw-data classification, and defined a query language working for this architecture. Jiang et al. [16] suggested an IoT-based data repository framework for combining HDFS, MySQL, and MongoDB together.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/JSEN.2015.2483499, IEEE Sensors Journal

B. MongoDB

MongoDB, developed by 10gen, is a document-oriented NoSQL database that offers high performance and scalability. Unlike other NoSQL databases, its data structure is designed independently as a document unit so that a schema definition is not needed. Moreover, MongoDB uses a scale-out scheme, which is flexible against hardware expansion, and supports auto-sharding. Thus, the automatic distribution of data over a number of servers can be conveniently carried out [17]-[20]. Fig. 1 shows the configuration of horizontal data partitioning, called sharding, and replica sets to ensure high availability, safety, and data consistency. They also enable distributed expansion for data processing involving large amounts of data. The functions of the MongoDB components are as follows.

- *mongod*: This is the primary daemon process for the MongoDB system. It handles data requests, manages data access, and performs background management operations.
- *mongos*: This is a routing service for MongoDB shard configurations that processes queries from the application layer and determines the location of these data in the shared cluster.
- *replica sets*: This is a group of mongod processes that maintain the same dataset. If the primary mongod is unavailable, the replica set will elect a new primary mongod.
- *shard*: This stores data. For easy availability and data consistency, each shard is a replica set.
- *config server*: This server stores the cluster's metadata. These data contain a mapping of the cluster's dataset to the shards.

Even distribution of data among shards is controlled by a shard key. A shard key is either a single indexed field or an indexed compound field that exists in every document. MongoDB divides the input data into chunks, logical units of stored data, according to the shard key by using either range-based partitioning or hash-based partitioning. Once the shard key is set up, it cannot be modified. Hence, appropriate shard key selection is a very important decision factor in the MongoDB design.

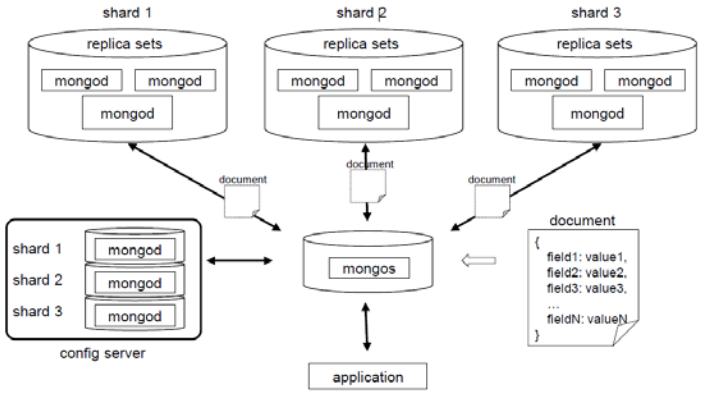

Fig. 1. MongoDB sharding architecture (edited from [18]).

There have been various research on the performance of MongoDB. Nyati et al. [21] compared the insertion/searching performance of MongoDB to MySQL in a single machine, showing that MongoDB outperformed MySQL. Kanade et al. [22] carried out an experimental comparative study between embedding and referencing design patterns, showing that the embedding pattern performs better in terms of query response time. Liu et al. [23] proposed an algorithm to solve irregular distribution of data among distributed storages, and demonstrated that the proposed approach can improve the throughput t and read/write response time of the existing automatic data distribution.

C. EPC Information Services (EPCIS)

EPCIS is an RFID event repository, which is one of the core components of the EPCglobal Architecture Framework [10]. It helps store RFID event information and share the information among supply chain partners. Electronic Product Code (EPC) refers to a coding scheme for unambiguous code for the designation of physical goods [24]. It can assign codes, according to an appropriate coding scheme such as Serialized Global Trade Item Number (SGTIN), a Shipment Container Code (SSCC), and a Global Returnable Asset Identifier (GRAI), to objects depending on the purposes. The EPCIS solutions should provide predefined vendor-independent capture/query interfaces to receive/supply RFID event data, although the vendors can freely implement the services by their own ways. Fig. 2 shows a summary of event types and fields of EPCIS event data. The schema of a MongoDB-based RFID/sensor data repository, extended from these event types, is described in Section III.

Le et al. [25] proposed a Cassandra-based column-family style EPCIS repository, and showed that it outperforms MySQL-based implementation in terms of response time, throughput, and flexibility. Li et al. [26] implemented an EPC discovery service (DS), which is one of the core information services along with EPCIS, using HBase. The proposed DS successfully provides even nested traceability information using a recursive discovery algorithm. Gomes [27] proposed an IoT infrastructure with an EPCIS module, which is based on NoSQL repository and works in a cloud virtual machine environment. Byun and Kim [28] implemented a MongoDB-based EPCIS architecture, and compare the query response time performance with Fosstrack EPCIS (a highly referenced open source EPCIS implementation) and Cassandra-based EPCIS, and demonstrated it performs better. However, they did not address an optimal design of MongoDB schema, and the proposed architecture does not completely conform to the EPCIS standard.

● : mandatory ○ : optional

	Object Event	*Aggregation Event*	*Transaction Event*	*Quantity Event*	Description
eventTime	●	●	●	●	The date and time at which event occurred.
recordTime	○	○	○	○	The date and time of the event were recorded by a repository.
epcList	●		●	●	A list of observed of EPCs naming the physical objects.
childEPCs		●			A list of the EPCs of the contained objects.
parentID		○	○		The identifier of the parent of the association.
epcClass				●	The identifier specifying the object class to which the event pertains
action	●	●	●	●	How an event relates to the lifecycle of the entity being described. {*ADD,OBSERVE,DELETE*}*
quantity				●	The number of objects within the class described by this event.
readPoint	○	○	○	○	The read point at which the event took place.
bizLocation	○	○	○	○	The business location where the objects may be found.
bizStep	○	○	○	○	The business step of which the event was a part.
disposition	○	○	○	○	The business condition of the objects.
bizTransationList	○	○	●	○	A list of business transactions that define the context of the event. {*bizTransactionType*, *bizTransaction*}**
extensions	○	○	○	○	This identifies the addition of new data members.

* *ObjectEvent* - *ADD*: commissioned; *DELETE*: decommissioned; *OBSERVE*: simple observation
AggregationEvent - *ADD*: physically aggregated; *DELETE*: physically disaggregated; *OBSERVE*: simple observation
TransactionEvent - *ADD*: associated to business transactions; *DELETE*: disassociated from business transactions; *OBSERVE*: simple observation

** *bizTransactionType*: An identifier that indicates what kind of business transaction (i.e. purchase order, invoice number)
bizTransaction : An identifier that denotes business transaction ID

Fig. 2. Four event types of EPCIS (edited from [29]).

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/JSEN.2015.2483499, IEEE Sensors Journal

III. MONGODB-BASED RFID/SENSOR DATA REPOSITORY

In this section, we describe the design of our RFID/sensor data repository in consideration of the MongoDB design pattern as well as the process for selecting an optimal shard key. With reference to the database schema of a well-known open-source EPCIS, we propose a MongoDB schema, which combines RFID and sensor data. Furthermore, we select an optimal compound shard key according to the theoretical guidelines suggested in previous literature.

A. Relational Database Schema for EPCIS Event Types

Fosstrak [30], which is an open-source RFID software platform project, provides an EPCIS implementation based on MySQL, which is one of the popular relational database systems. Fig. 3 shows the `ObjectEvent` schema of the Fosstrak EPCIS. Since EPCIS event fields may have multiple values, such as `epcList`, `childEPCs`, `bizTransactionList`, and `extensions`, a field table with multiple values and event tables are normalized with one-to-many or many-to-many relationships to overcome addition/deletion/update anomalies. The `AggregationEvent`, `TransactionEvent`, and `QuantityEvent` are also designed in the same manner.

According to the EPCIS standard, every event has an extension point, to which additional data members can be attached. Therefore, we can easily store additional information from the various IoT data sources, such as sensors, GPS's, and other intelligent devices.

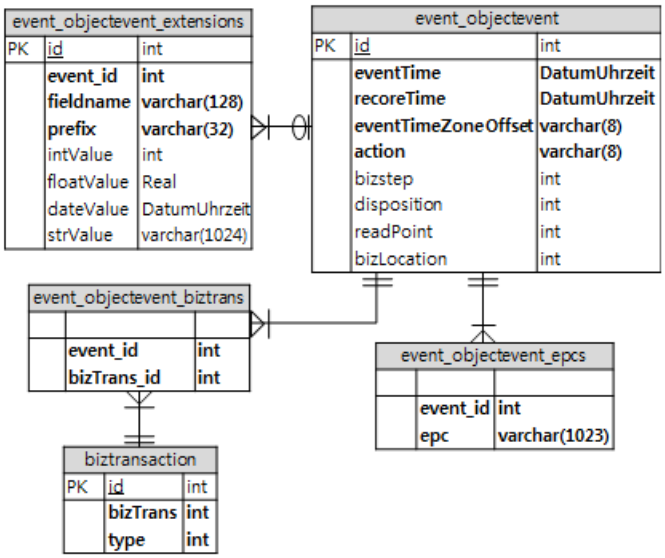

Fig. 3. ObjectEvent schema of Fosstrak EPCIS (revised from [30])

B. MongoDB for EPCIS Implementation

As explained in Section III.A, EPCIS events are represented by structured data types, which have even one-to-many or many-to-many relationships among them. That means we cannot use simple key-value stores for the purposed of RFID/sensor data repository. On the other hand, RFID/sensor data repository are used mainly for event tracking or search queries rather than any type of column-wise aggregation operations as in OLAP, meaning that column-oriented databases are not preferable to document databases either. In these senses, the document database is right choice for representing RFID/sensor data repository. Furthermore, the queries between EPCIS and accessing applications are all in the form of XML messages so that data transformation processes are unnecessary. Among document databases, MongoDB is the most popular NoSQL database according to [31].

C. Design of MongoDB-based RFID/sensor data repository

The design patterns of the MongoDB data model fall into two categories: *embedding* and *referencing*, as shown in Fig. 4. Embedding is a scheme in which other documents related to a given document are embedded as a sub-document, whereas referencing is a scheme where related documents are separated into other collections (such as tables in RDBs). In the referencing scheme, a specific field in the referencing collection is to be set as a link to the referenced collection, as a foreign key does in RDBs [19], [20].

As an EPCIS event consists of logically-related entities (such as `bizTransactionList` and extension) which have one-to-many or many-to-many relationships, we need to decide whether to include the entities in an event collection (i.e., embedding) or to separate them physically into other collections (i.e., referencing). The decision is basically a matter of which is more important between data integrity (plus storage saving) and query performance. In this study, we propose an embedding scheme-based RFID/sensor data repository because RFID/sensor data are basically raw data that, once stored, is rarely modified. This means that the data have no possibility of modification/deletion anomalies and therefore, no need to perform normalization. Moreover, as `bizTransactionList` and extension entities are optional members of each event, which is a *weak* relationship, there is no reason to separate them physically into additional collections. In addition, the embedding scheme, which does not carry out normalization, is appropriate for a large RFID/sensor data repository, which requires faster read and write performance. Fig. 5 shows the embedding scheme-based design of the proposed RFID/sensor data repository. The `eventType` field was added to distinguish four different event types. The entities who has multiple values like `epcList` and `childEPCs` are defined as array-type members, while `bizTransactionList` and `extensions` are embedded as subdocument-type members.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/JSEN.2015.2483499, IEEE Sensors Journal

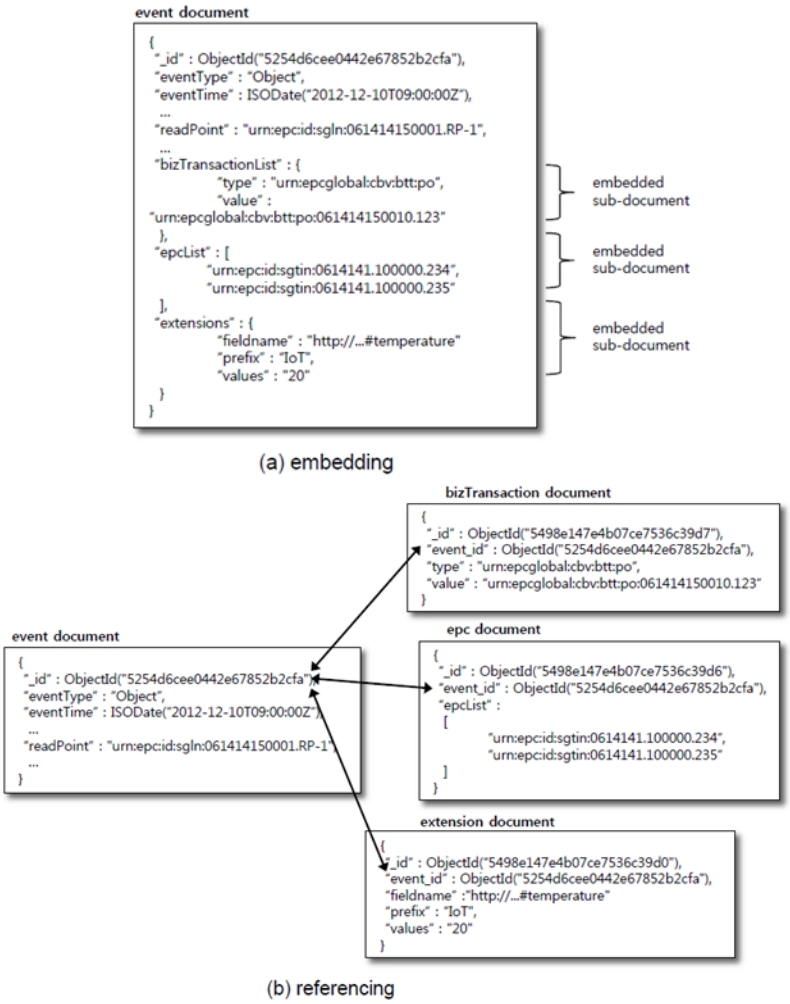

Fig. 4. Embedding vs. referencing

eventType	String <Object, Aggregation, Transaction, Quantity>	
eventTime	Timestamp	
recordTime	Timestamp	
epcList	ArrayOfString	
childEPCs	ArrayOfString	
parentID	String	
epcClass	String	
action	String <ADD, OBSERVE, DELETE>	
quantity	NumberLong	
readPoint	String	
bizLocation	String	
bizStep	String	
disposition	String	
bizTransactionList	Sub-document	
	Type	String
	value	String
extensions	Sub-document	
	fieldName	String
	prefix	String
	values	ArrayOfString

Fig. 5. MongoDB-based RFID/sensor data repository schema.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/JSEN.2015.2483499, IEEE Sensors Journal

As explained in Section III.A, every EPCIS event can add extension fields, called "extensions." This member has mandatory fields - `fieldName`, `prefix`, and `values`. As shown in Fig. 6, sensor data and GPS data can be added to an RFID event with the corresponding sensor type, prefix of name space, and sensor value.

Fig. 7 shows a document stored in the proposed MongoDB-based RFID/sensor data repository in XML format. This document is the basic unit of data, in which an EPCIS receives and supplies RFID/sensor events through capture and query interfaces respectively.

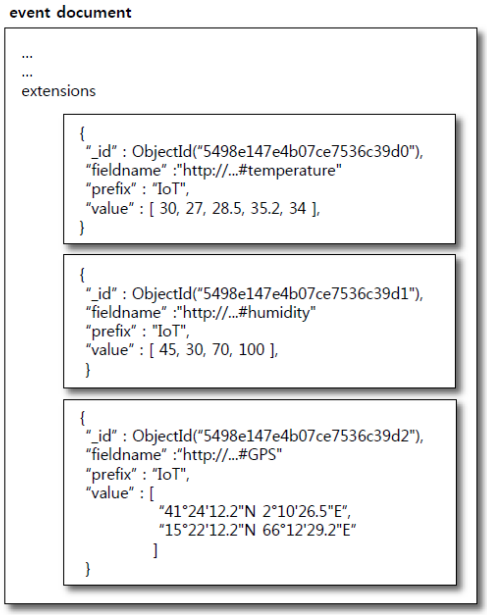

Fig. 6. Example of `extensions` containing sensor data.

D. Selection of Shard Key

A shard key should be determined to divide input data into chunks effectively, and to distribute data through the shard evenly. An ideal shard key is a compound key of two fields, in which the first should have a moderately coarse-grained cardinality and the other should be a more fine-grained field with higher cardinality [17], [19]. Once selected as a shard key, the field is to be indexed too. Thus, the choice of a field as a shard key should also involve considering whether it is frequently used in queries as criteria.

Fig. 7. Example of XML format for RFID/sensor event.

Two fields - `eventType` and `readPoint` - are the only ones with finite cardinality among MongoDB-based RFID/sensor event member fields. The `readPoint` field is appropriate as the first part of the compound shard key, in the sense that each `readPoint`, which represents a point where an object is recognized, is identified by a unique ID. Once a logistic process in business is defined, all the objects will pass through almost the same (finite) set of `readPoints`. Note that `readPoint` is optional in theEPCIS event schema. In order to use `readPoint` as a shard key, implementers should define `readPoint` as a Non-Null field having a finite set of values in developing RFID/sensor systems.

On the other hand, the `eventType` field neither has appropriately grained cardinality (it has only four as shown in Fig. 2) nor is frequently used in queries compared to `readPoint`. Actually, the most commonly used criteria in queries are of what (`epc`), when (`eventTime`), and where (`readPoint`). As the second part of the compound shard key, a high-cardinality field is desirable. Two candidates are `eventTime`, which records event time, and `epcList`, which represents the ID of an object. The `eventTime` field is characterized by an increasing value whenever an event occurs, while the `epcList` field assigns a new ID whenever a new object appears. Thus, both of them can be viewed as high-cardinality and fine-grained key. However, due to the MongoDB characteristics that an array-type field cannot be a shard key, `epcList` cannot be considered as a shard key. Hence, `eventTime` should be selected as the second part of the compound shard key.

It is true that even if `eventTime`, which is a high-cardinality and fine-grained key, is chosen as a single shard key, data can be evenly split into shards. However, in this case, RFID/sensor data currently being generated will be headed to a specific server in a row, causing congestions. Another problem can occur when `readPoint`, which has a moderate level of cardinality, is adopted as a single shard key, such that it cannot guarantee uniform data distribution and chunk split upon continuous data accumulation. Such a situation can occur when the cardinality of reappoint is too low compared to the number of events generated. As shown in Fig. 8, if the number of possible values of `readPoint` is five, only five chunks can be formed, and no more splits occur even if a large amount of event data is accumulated as time goes.

Consequently, in this study, we adopt the compound shard key {`readPoint`, `eventTime`} as the optimal shard key. As shown in Fig. 9, a chunk is created per `readPoint`. Once the number of objects in a single chunk increases, a chunk can be split further according to `eventTime`, hence ensuring uniform

distribution of data. Moreover, as both the `readPoint` and `eventTime` fields are core keys for object and environment tracking, they help improve the query performance in addition to the benefits of uniform distribution of data. In Section IV, we verify the appropriateness of our shard key selection and the scalability of a MongoDB-based RFID/sensor data repository through experiments.

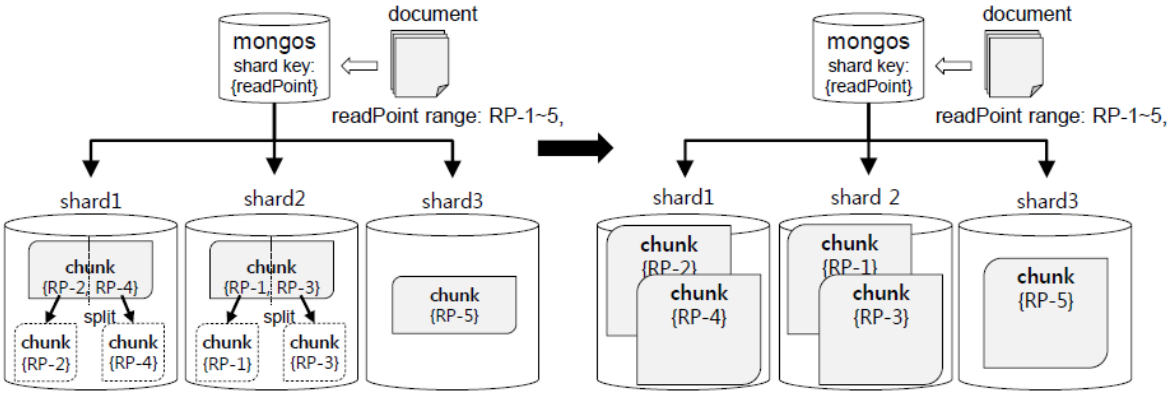

Fig. 8. Chunks that cannot be split.

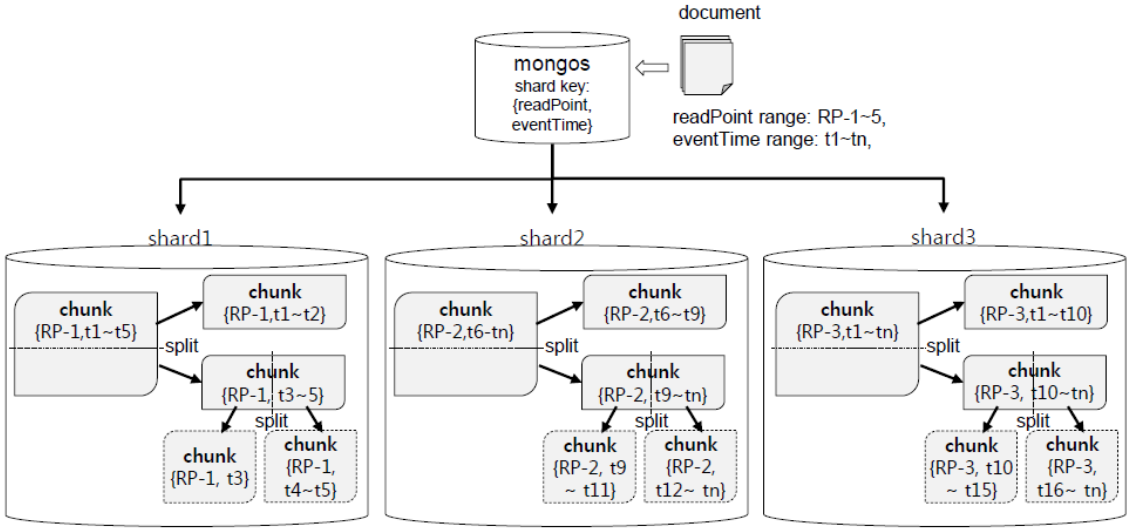

Fig. 9. Well-split chunks by shard key as `readPoint` and `eventTime`.

IV. Experimental Evaluation

In order to validate our proposed design, we carried out a series of experiments using sample data set, which was generated based on a realistic supply chain model as summarized in Table II. The first experiment compares data distribution levels and query performance of different shard key choices in order to validate our choice of shard key. The second experiment compares query performance of MongoDB-based repository with MySQL-based repository on a single machine in order to check if our choice of database (i.e., NoSQL) outperforms a representative relational database (like MySQL). The last experiment checks whether an increase in the number of MongoDB shards improves query performance.

TABLE II
Experiments

No	Test Name	Metrics	Comparison Targets
1	Shard key test	distribution level	different shard-key choices
		response time	two compound shard-key choices
2	MongoDB vs. MySQL performance test	response time	MongoDB-based repository vs. MySQL-based repository
3	Sharding performance test	response time	MongoDB-based repositories with different no. of shards

A. Supply Chain Scenario

To generate a large volume of RFID/sensor data, we assume an automotive supply chain comprising three module manufacturers along with nine part suppliers. Three module manufacturers produce chassis modules, cockpit modules, and front-end modules respectively. In general, chassis module, cockpit module, and front-end module consist of about 100, 130, and 30 components (or parts) respectively. We assumed only a small portion of important parts is traced using RFID/sensor tags. They are suspension, power steering, and brake for the chassis module; instrument panels, cowl crossbars, and ventilation systems for the cockpit module; and headlamps, radiator assemblies, and front bumper beams for the cockpit module. We also assume that the detailed internal logistic processes of each part supplier are divided into two types - molding type and assembly type, as shown in Fig. 10, and simulation parameters such as production rate, processing time, and transportation time are identical for convenience. In addition, the temperature data in all the processes and the GPS data during the transportation stage are assumed to be produced in 10-second intervals. According to the defined process model and parameters, around 100 million events were generated over two simulation months, which require about 200 GB of MongoDB storage.

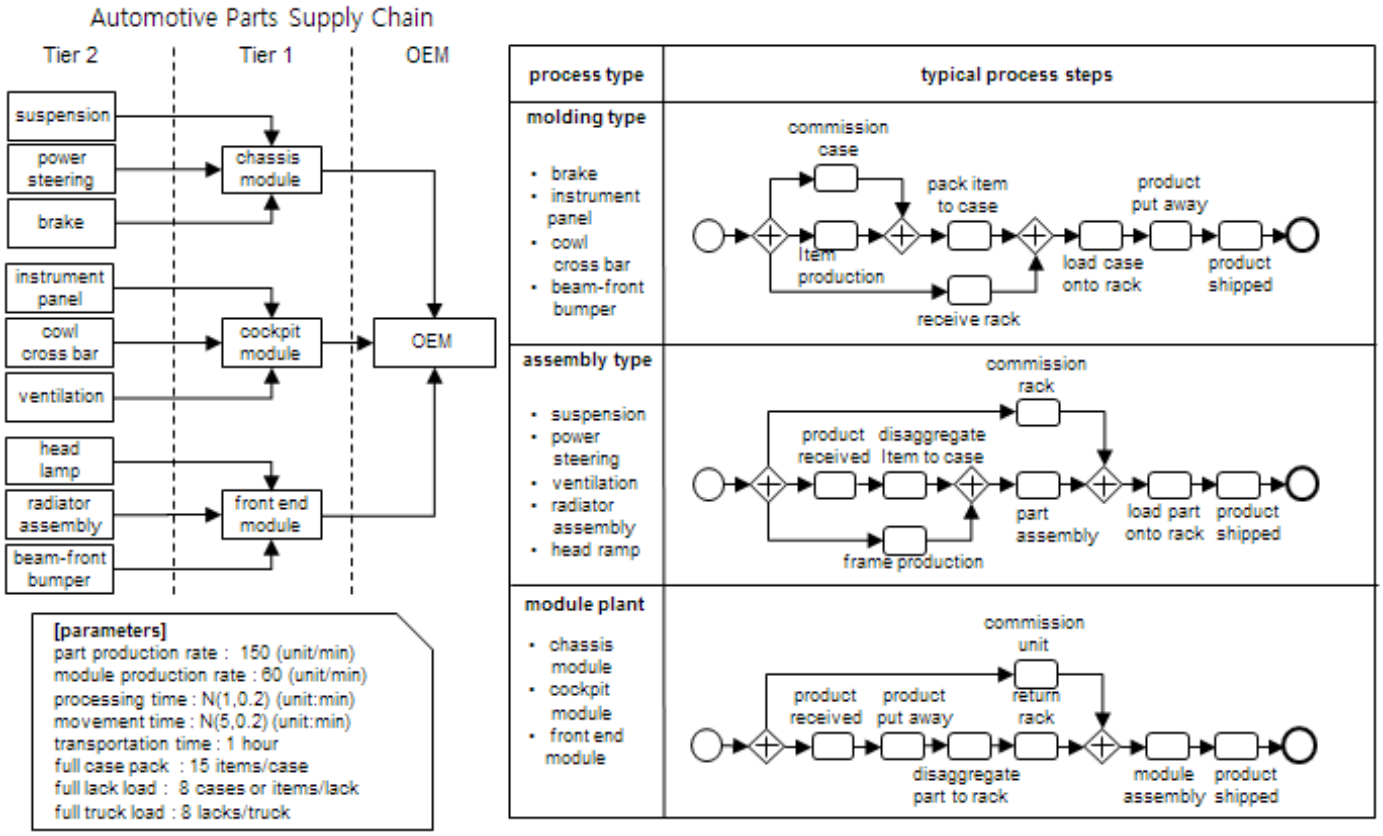

Fig. 10. Supply chain process model.

B. Test Environment

To generate the simulation data, we set up a cluster of three server machines with same specifications, as shown in Fig. 11. Three servers are configured as shards for storing data, and one server is used as a mongos and config server. Client programs for storing and querying data are situated in the same network with mongos, and the tests were conducted in a wired network connection.

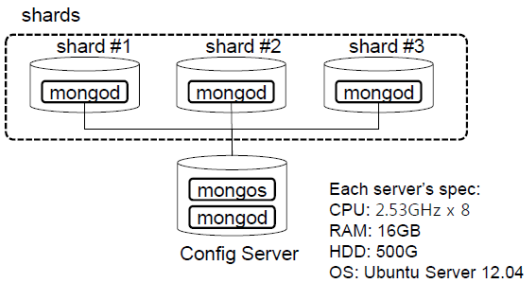

Fig. 11. Configurations for simulation test.

C. Shard Key Test

This test aims to validate whether the compound shard key `{eventType, eventTime}` proposed in Section III.D guarantees an even distribution of data and better query performance. In addition, a comparison between a single shard key and a compound shard key in terms of evenness of data distribution was made. Table III summarizes the candidates for shard key to compare.

1) Distribution Level Test

According to the process defined in Section IV.A, 10 million events were stored in the RFID/sensor data repository configured as shown in Fig. 11. Table IV, which summarizes the results, indicates that chunk split did not occur sufficiently in the case of single shard keys `{readPoint}` and `{eventType}`, as predicted in Section III.D. The reason is that a few chunks were created as the cardinality of each shard key, and the chunks were stacked with the event data without splitting. On the other hand, in the cases of the single shard key `{eventTime}`, compound shard key `{readPoint, eventTime}`, and compound shard key `{eventType, eventTime}`, the chunks were all sufficiently well split. However, as mentioned in Section III.D, the single shard key `{eventTime}` has a disadvantage in that data input is concentrated in a shard, even though even distribution can be achieved.

2) Query Performance Test

In order to evaluate the query response time of the compound shard key `{readPoint, eventTime}` suggested in Section III.D, we compared the response time of the two databases, which were generated via two candidate compound shard keys shown in Table IV. The queries used in this test are fundamental queries, which are frequently used for object history tracking in a supply chain (see Table V). Other possible queries such as 'quantity shipped' and 'dwell time' queries can be derived from these fundamental queries, meaning that the performance of them are deeply dependent on that of fundamental queries. Hence, we restrict the experiments to the fundamental queries. Furthermore, the fields used in the query (i.e., `parentID`, `epcList`, `readPoint`, and `eventType`) were set as index keys.

TABLE III
CANDIDATES FOR SHARD KEY

No.	Types	Shard Key
1	single	`{readPoint}`
2	single	`{eventType}`
3	single	`{eventTime}`
4	compound	`{readPoint, eventTime}`
5	compound	`{eventType, eventTime}`

TABLE IV
RESULT OF SHARD KEY TEST

	the number of objects			the number of chunks		
Shard Key	Shard #1	shard #2	shard #3	shard #1	shard #2	shard #3
`{readPoint}`	8,717,756	623,206	566,256	3	2	2
`{eventType}`	6	9,836,806	70,406	1	1	1
`{eventTime}`	2,518,689	4,830,775	2,586,389	61	71	61
`{readPoint, eventTime}`	2,777,360	4,376,540	2,763,953	156	158	155
`{eventType, eventTime}`	3,454,937	3,892,420	2,559,867	115	122	116

TABLE V
TEST QUERIES

Query	Description	Query parameters	Query Statement
Q1	Query for event that occurred in a specific period and a specific place	`readPoint`, `eventTime`	`db.collection.find({ readPoint: "readPoint x", eventTime:{ $gte:ISODate("yyyy-MM-dd'T'HH:mm:ss'Z'"), $lt:ISODate("yyyy-MM-dd'T'HH:mm:ss'Z'") });`
Q2	Query for event related to a target product during a specific period	`eventTime`, `epcList`	`db.collection.find({ epcList: "epc x", eventTime:{ $gte:ISODate("yyyy-MM-dd'T'HH:mm:ss'Z'"), $lt:ISODate("yyyy-MM-dd'T'HH:mm:ss'Z'") });`
Q3	Query for object ID contained in a specific case at specific time	`eventType`, `eventTime`, `parentID`	`db.collection.find({ parentID: "epc x", eventType: "AggreationEvent", eventTime:{ $gte:ISODate("yyyy-MM-dd'T'HH:mm:ss'Z'"), $lt:ISODate("yyyy-MM-dd'T'HH:mm:ss'Z'") });`

Using the simulation model, we generated 100 million objects and compared the response time of two compound shard key cases. Every time the number of stored objects reaches a multiple of 20 million, 10 clients access mongos simultaneously and query Q1, Q2, and Q3 30 times each; then, the average response times were recorded. Fig. 12 shows the test results. For Q1, as the number of data increases, the response time of the database with the compound shard key {`readPoint, eventTime`} got remarkably faster than the database with the other compound shard key {`eventType, eventTime`} while they show similar performance for Q2 and Q3.

These two shard key tests confirm that the compound shard key {`readPoint, eventTime`} proposed in this paper guarantees evenly distributed data and is advantageous in query performance.

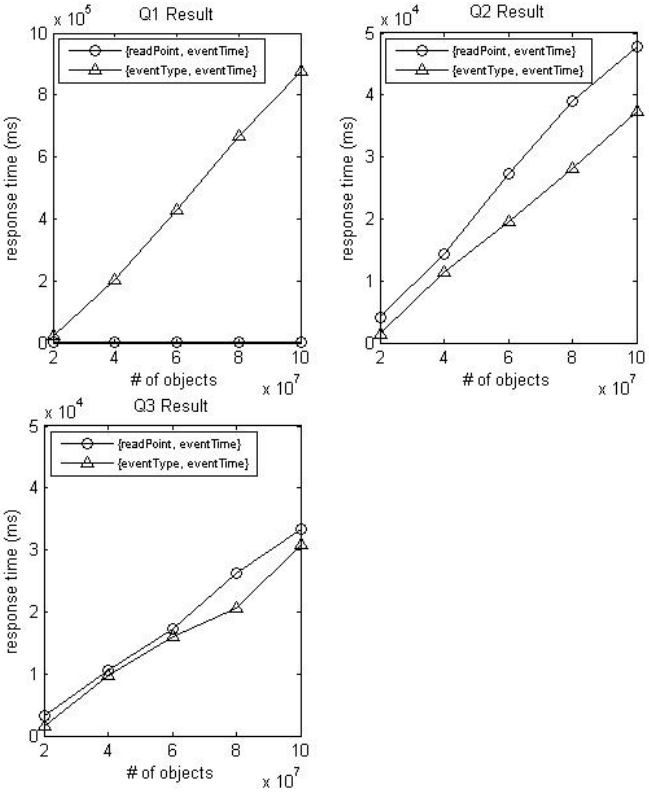

Fig. 12. Result of query performance test (between two compound shard keys).

D. MongoDB vs. MySQL Performance Test

This test simply compares the suggested MongoDB-based RFID/sensor data repository with MySQL-based RFID/Sensor data repository (as a representative relational database). MySQL-based repository complies with Fosstrak EPCIS data schema descripted in Section III.A. For this test, we configured a Mongo-DB based repository which has one shard, and a MySQL-based repository on a single machine. In common with the query performance test in Section IV.C, we used three queries (Q1, Q2 and Q3), and measured the response time in the same way. For a fair comparison, the fields used in the query (i.e., `readPoint`, `eventTime`, `epcList`, and `parentID`) were all indexed in both MongoDB and MySQL.

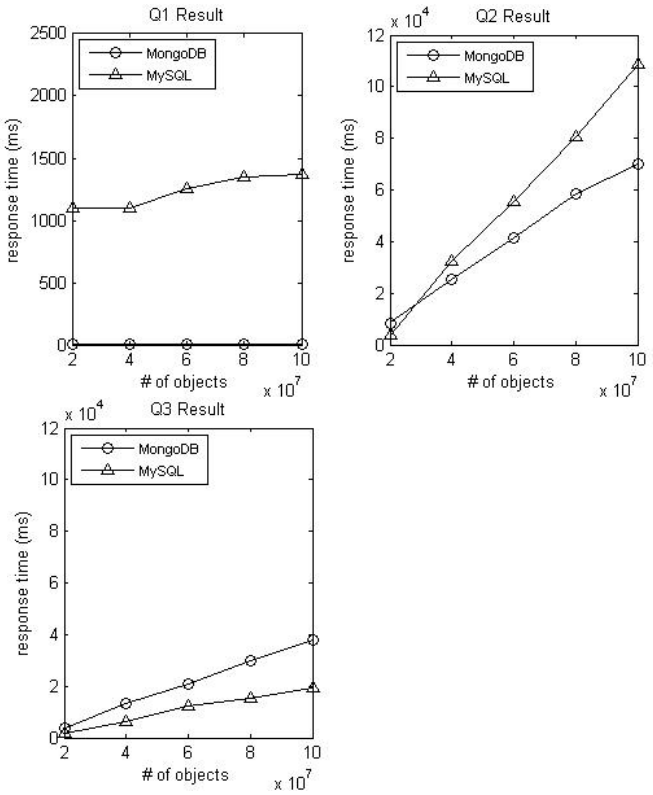

Fig. 13. Result of MongoDB vs. MySQL performance test.

The test result is shown in Fig. 13. For Q1 and Q2, MongoDB outperforms MySQL. For Q1 in particular, MongoDB demonstrates a very stable and low response time constantly. On the other hand, for Q3, MySQL is about twice faster than MongoDB. Because Q1 and Q2 are more frequently used queries than Q3 in real practice, MongoDB-based RFID/Sensor data repository will be a better choice even if only the query performance is taken into account. Moreover, if a MongoDB-based repository utilizes multiple shards (unlike this experiment), it will demonstrate higher performance as shown in the following section.

E. Sharding Performance Test

The objective of this test is to confirm whether an increase in the number of shards improves query performance. To this end, this test compared the response time per shard composition (one shard, two shards, and three shards) according to an increase in (1) the size of data volume (we call it volume test) and (2) the number of clients (we call it throughput test). The shard key was set to {readPoint, eventTime}, and the fields used in the query were also all indexed.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/JSEN.2015.2483499, IEEE Sensors Journal

1) Volume Test

We generated 100 million objects using the simulation model, and recorded the average response time among each cluster which configured with one, two, and three shards. Fig. 14 shows the test results. For Q1 and Q3, the number of shards had practically no effect on the response time. Further, there was practically no effect on the search speed owing to the fact that `readPoint` and `parentID` were indexed and the value they can take is significantly smaller than `epcList`. On the other hand, with respect to Q2, the case of three shards shows a response that is around 214 ms faster in the early stages and around 7,059 ms faster in the later stages, as compared to the case of two shards. In addition, the case of three shards shows a response time that is around 5,600 ms faster in the early stages and around 21,000 ms faster in the later stages, as compared to the case of one shard. Therefore, an increase in the number of shards is confirmed to improve query speed.

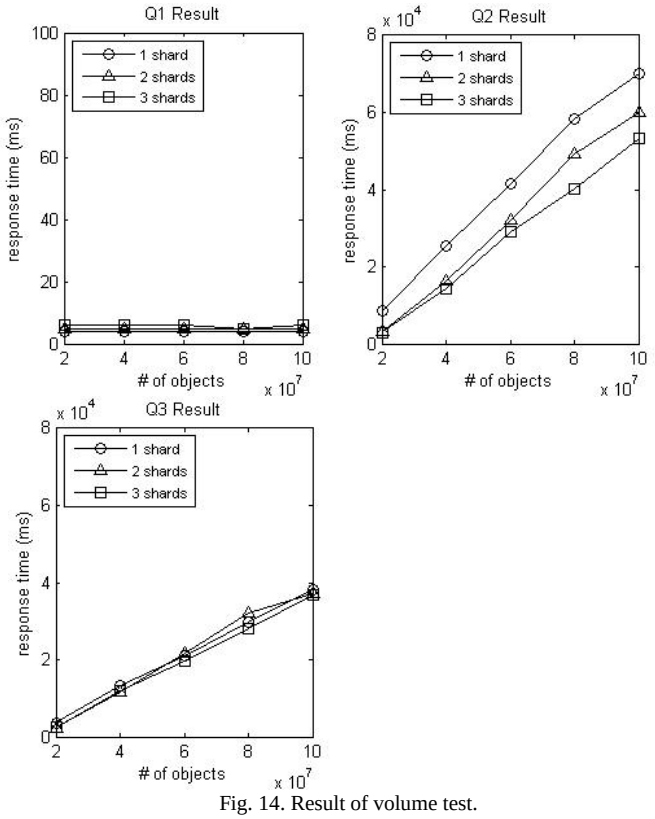

Fig. 14. Result of volume test.

2) Throughput Test

This test compares the response time by shard configuration according to the increase in the number of clients. In this test, the average response times for Q1, Q2, and Q3 were recorded by fixing the data storage volume at 100 million events and changing the number of simultaneously querying clients to 1, 5, 10, 15, and 20. Fig. 15, which summarizes the test results, shows that as the number of querying clients increases, clusters with more shards show better performance. Particularly, in the case of Q2, when the number of simultaneously querying clients is 20, the case of three shards shows a response time that is 20,000 ms faster, as compared to that of two shards, and 45,000 ms faster, as compared to that of one shard.

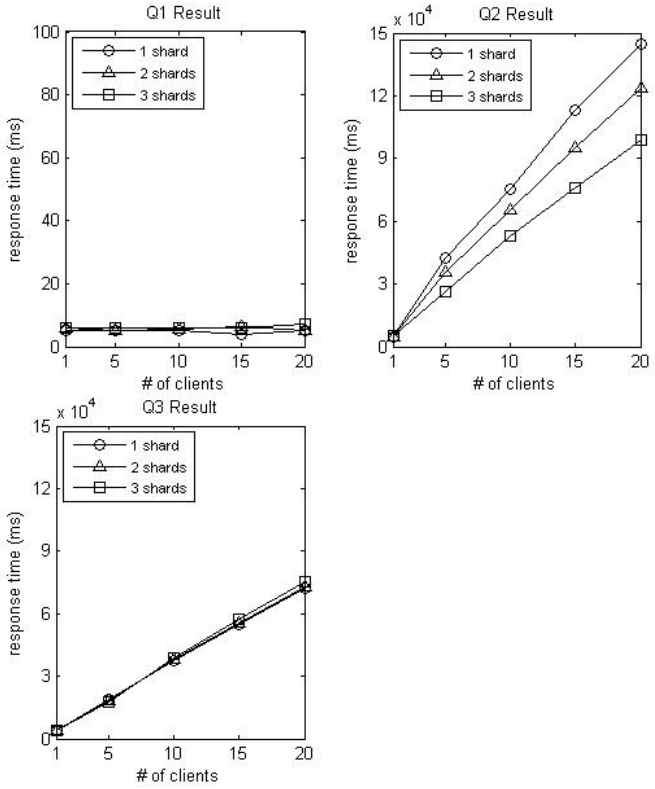

Fig. 15. Result of throughput test.

From the above two tests, it is confirmed that a large improvement in performance can be realized with an increase in the number of shards when a field having various values, i.e., a field having a large search space, is queried to search for documents stored in the database and when the number of simultaneously querying clients is large.

V. CONCLUSION

RFID and sensor technologies are undoubtedly the core technologies of future IoT. Many researchers [32-35] have studied various research issues on integrating RFID and sensor technologies. At present, efforts are being made to integrate these two technologies on the same IoT platform in different fields. Unlike conventional studies that provide IoT platforms at the architecture level only, this study proposed an implementation model of an RFID/sensor data repository on the basis of MongoDB. Furthermore, based on logistic process simulation of automotive parts, the proposed RFID/sensor data repository was empirically validated in terms of even distribution of data and query speed.

Our study is based on a typical manufacturing supply chain scenario. Although it is quite representative, there are still exceptional situations, in which the proposed design would not work effectively. For example, if a specific type of event other than `ObjectEvent` such as `AggregationEvent`, which holds boxing and de-boxing information, is intensively queried, we may have to reconsider our choice of the shard key. In addition, we consider the query response time as the only performance metric. However, there are some possibilities that

growing data might lower the writing speed, due to the nature of the embedding model.

Therefore, we need to investigate the optimal design of MongDB based EPCIS implementation, which can work for other non-trivial query requirements in the future. In addition, it is necessary to make a further comparative study amongst document-based NoSQL database alternatives (not only MongoDB) from the viewpoint of RFID/Sensor big data processing. Our ultimate goal of this research trail is to propose a performance model of the MongoDB-based RFID/sensor data repository, which can support the shard-extension planning.

REFERENCES

[1] E. Ilie-Zudor, Z. Kemeny, F. Blommestein, L. Monostori, and A. Meulen, "A survey of applications and requirements of unique identification systems and RFID technique," *Comput Ind*, vol. 62, pp. 227-252, 2011.

[2] L. D. Xu, W. He, and S. Li, "Internet of things in industries: a survey," *IEEE T Ind Inform*, vol. 10, no. 4, pp. 2233-2243, 2014.

[3] J. Mitsugi, T. Inaba, B. Pátkai, L. Theodorou, J. Sung, T. S. López, D. Kim, D. McFarlane, H. Hada, Y. Kawakita, K. Osaka, and O. Nakamura, Architecture Development for Sensor Integration in the EPCglobal Network, Auto-ID Labs White Paper, WP-SWNET-018, 2007.

[4] Y.-S. Kang, H. Jin, O. Ryou, and Y.-H. Lee, "A simulation approach for optimal design of RFID sensor tag-based cold chain systems," *J. Food Eng*, vol. 113, pp. 1-10, 2012.

[5] L. Jiang, L. D. Xu, H. Cai, Z. Jiang, F. Bu, and B. Xu, "An IoT-oriented data storage framework in cloud computing platform," *IEEE T Ind Inform*, vol. 10, no. 2, pp. 1443-1451, May 2014.

[6] D. E. O'Leary, "'Big Data', The 'Internet of things' and the 'internet of signs'," *Intell. Syst. Account. Finance Manag.*, vol. 20, pp. 53-65, 2013.

[7] J. S. Veen, B. Waaij, and R. J. Meijer, "Sensor data storage performance: SQL or NoSQL, Physical or Virtual," in *Proc. 5th IEEE Cloud*, pp. 431-438, 2012.

[8] A. Castiglione, M. Gribaudo, M. Lacono, and F. Palmieri, "Exploiting mean field analysis to model performances of big data architectures," *Future Gener Comput Syst.*, vol. 37, pp. 203-211, 2014.

[9] G. Noorts, J. Engel, J. Taylor, D. Roberson, R. Bacchus, T. Taher, and K. Zdunek, "An RF spectrum observatory database based on a hybrid storage system," in *Proc. IEEE Dyspan*, pp. 114-120, October 2012.

[10] EPCglobal Inc., EPC Information Services (EPCIS) version 1.0, EPCglobal ratified specification, Available: http://www.gs1.org/gsmp/kc/epcglobal/epcis.

[11] C. Strauch, U. L. S. Sites, and W. Kriha, "NoSQL databases," Lecture Notes, Stuttgart Media University, 2011.

[12] R. Cattell, "Scalable SQL and NoSQL data stores," *Sigmod Rec.*, vol. 39, no. 4, pp. 12-27, 2011.

[13] N. Leavitt, "Will NoSQL databases live up to their promise?" *Computer*, vol. 43, no. 2, pp. 12-14, 2010.

[14] D. Feinberg, M. Andrian, and N. Heudecker, "2013 Gartner Magic quadrant for operational database management systems," Gartner Research Note, Oct. 21, 2013.

[15] T. Li, Y. Liu, Y. Tian, S. Shen, and W. Mao, "A storage solution for massive Iot data based on NoSQL,". In *Proc. IEEE GreenCom*, pp. 50-57, November 2012.

[16] L. Jiang, L. Da Xu, H. Cai, Z. Jiang, F. Bu, and B. Xu, "An IoT-oriented data storage framework in cloud computing platform," *IEEE T Ind Inform*, vol. 10, no. 2, pp. 1443-1451, 2014.

[17] K. Banker, *MongoDB in Action*, Manning Publications Co, 2011.

[18] T. Sasaki, *NoSQL core guide for big data era*, RoadBook, 2011.

[19] R. Copeland, *MongoDB Applied Design Patterns*, O'Reilly Media, Inc., 2013.

[20] K. Chodorow, *MongoDB: the definitive guide*, O'Reilly Media, Inc., 2013.

[21] S. S. Nyati, S. Pawar, and R. Ingle, "Performance evaluation of unstructured NoSQL data over distributed framework," In *Proc. IEEE ICACCI*, pp. 1623-1627, August 2013.

[22] A. Kanade, A. Gopal, and S. Kanade, "A study of normalization and embedding in MongoDB," In *Proc. IEEE IACC*, pp. 416-421, Febuary, 2014.

[23] Y. Liu, Y. Wang, and Y. Jin, "Research on the improvement of MongoDB Auto-Sharding in cloud environment," In *Proc. 7th IEEE ICCSE*, pp. 851-854, July 2012.

[24] F. Thiesse, C. Floerkemeier, M. Harrison, F. Michahelles, and C. Roduner, "Technology, standards, and 4eal-world deployments of the EPC network," *IEEE Internet Comput.*, vol. 13, no. 2, pp. 36-43, 2009.

[25] T. D. Le, S. H. Kim, M.H. Nguyen, D. Kim, S. Y. Shin, K. E. Lee, and R. da Rosa Righi, "EPC information services with No-SQL datastore for the Internet of Things," In *Proc. IEEE RFID*, pp. 47-54, April 2014.

[26] M. Li, Z. Zhu, and G. Chen, "A scalable and high-efficiency discovery service in IoT using a new storage schema," In *Proc. IEEE COMPASC*, pp. 754-759, 2013.

[27] M. M. Gomes, R. D. R. Righi, and C. A. da Costa, "Future directions for providing better IoT infrastructure," In *Proc. UbiComp*, pp. 51-54, September 2014.

[28] J. Byun, and D. Kim, "Oliot EPCIS: New EPC information service and challenges towards the Internet of Things," In *Proc. IEEE RFID*, pp. 70-77, April 2015.

[29] Y.-S. Kang and Y.-H. Lee, "Development of generic RFID traceability services," *Comput Ind*, vol. 64, no. 5, pp. 609-623, 2013.

[30] Fosstrak EPCIS Architecture Guide, Available: https://code.google.com/p/fosstrak/wiki/EpcisArchitectureGuide.

[31] A. B. M. Moniruzzaman and S. A. Hossain, "NoSQL database: new era of database for big data analytics – classification, characteristics and comparison," *Ijdta*, vol. 6, no. 4, pp. 1-14, 2013.

[32] L. Zhang and Z. Wang, "Integration of RFID into Wireless Sensor Networks: Architectures, Opportunities and Challenging Problems," In *Proc. the 5th International Conference on Grid and Cooperative Computing Workshops* (*GCCW '06*), pp. 463-469, 2006.

[33] J. Mitsugi, T. Inaba, B. Pátkai, L. Theodorou, J. Sung, T.S. López, D. Kim, D. McFarlane, H. Hada, Y. Kawakita, K. Osaka and O. Nakamura, "Architecture Development for Sensor Integration in the EPCglobal Network," Auto-ID Labs, White Paper WP-SWNET-018, 2007.

[34] H. Liu, M.B.a.A. Nayak and I. Stojmenovic, "Taxonomy and Challenges of the Integration of RFID and Wireless Sensor Networks," *IEEE Network*, vol. 22, no. 6, pp. 26-35, 2008.

[35] A. Al-Fagih, F. Al-Turjman, W. Alsalih and H. Hassanein, "A Priced Public Sensing Framework for Heterogeneous IoT Architectures," *IEEE Trans. Emerg. Topics Comput.* - Special Issue on Cyber-Physical Systems, vol. 1, no. 1, pp. 133-147, 2013.