An IoT-Oriented Data Storage Framework in Cloud Computing Platform

Lihong Jiang, Li Da Xu, *Senior Member*, *IEEE*, Hongming Cai, Zuhai Jiang, Fenglin Bu, and Boyi Xu

***Abstract*—The Internet of Things (IoT) has provided a promising opportunity to build powerful industrial systems and applications by leveraging the growing ubiquity of Radio Frequency IDentification (RFID) and wireless sensors devices. Benefiting from RFID and sensor network technology, common physical objects can be connected, and are able to be monitored and managed by a single system. Such a network brings a series of challenges for data storage and processing in a cloud platform. IoT data can be generated quite rapidly, the volume of data can be huge and the types of data can be various. In order to address these potential problems, this paper proposes a data storage framework not only enabling efficient storing of massive IoT data, but also integrating both structured and unstructured data. This data storage framework is able to combine and extend multiple databases and Hadoop to store and manage diverse types of data collected by sensors and RFID readers. In addition, some components are developed to extend the Hadoop to realize a distributed file repository, which is able to process massive unstructured files efficiently. A prototype system based on the proposed framework is also developed to illustrate the framework's effectiveness.**

***Index Terms*—Cloud computing, data storage, file repository, Internet of Things (IoT), IoT database, multiple databases.**

I. Introduction

THE INTERNET OF THINGS (IoT) technology has obtained great development over the last few years and is increasingly influencing various industrial development [1]. The IoT refers to uniquely identifiable objects and their virtual representations in an Internet-like structure. The term "Internet of Things" was first used by Kevin Ashton in 1999 and became popular through the Auto-ID Center and related market analysis publications. Radio Frequency IDentification (RFID) tags, sensors, actuators, and mobile phones are often seen as prerequisites for the IoT. In other words, the key technologies of IoT include RFID technology, sensor network and detection technology, internet technology, intelligent computing technology, etc. Based on such technologies, IoT can connect a variety of physical objects, through unique addressing schemes, to an Internet-like structure, which enables the objects to interact and cooperate with each other to reach common goals [2]. However, technical challenges must be tackled before these systems can be widely applied.

In order to properly manage the physical objects involved in the IoT system and the devices used to monitor the objects, collect and transit data, we are facing series of challenges. The ubiquitous sensors, RFID readers, and other devices involved in the IoT systems can generate data rapidly so that the data must be processed with a high throughput. Furthermore, because the volume of the data is very large and can increase rapidly, a data storage solution for the IoT data must not only be able to store massive data efficiently but also support horizontal scaling. Moreover, the IoT data can be collected from many different sources and consisted of various structured and unstructured data; data storage components are expected to have the ability to deal with heterogeneous data resources.

For the challenges mentioned above, a data storage platform with the ability of efficiently storing and managing massive structured and unstructured IoT data is required. Thus, we propose a data storage framework for IoT data. In order to store and manage structured data, a database management model based on combined multiple databases is built. Besides, a file repository is built to implement version management of unstructured data. Furthermore, based on the database management model and the file repository, a RESTful service generating mechanism is proposed to provide HyperText Transfer Protocol (HTTP) interface for those applications accessing the data that stored based on the framework.

Section II reviews related work, Section III gives an overview of the framework, Section IV presents the approach in detail, Section V presents a case study, and Section VI provides a conclusion.

II. Related Work

In the area of IoT data storage and processing, numerous efforts have been made. In the aspect of data disposing in cloud platform, the existing related work can be classified into three types, namely, sensor data integration, data storage, and application supporting technology.

In sensor data integration, a method that supports flexible onboard processing of large volume of sensor data [3] of a vehicle by adopting stream processing techniques is proposed. Researchers [4] provide a method for wireless sensors to reduce the redundant data collection. In [5], the authors create a fast

Manuscript received September 30, 2013; accepted January 28, 2014. Date of publication February 17, 2014; date of current version May 02, 2014. This work was supported in part by the National Natural Science Foundation of China under Grant 71132008, Grant 71171132, and Grant 61373030, and in part by U.S. National Science Foundation under Grant SES-1318470 and Grant 1044845. *(Corresponding author: H. Cai.)* Paper no. TII-14-0097.

L. Jiang, H. Cai, Z. Jiang, and F. Bu are with the School of Software, Shanghai Jiao Tong University, Shanghai 200240, China (e-mail: hmcai@sjtu.edu.cn).

L. D. Xu is with the Institute of Computing Technology, Chinese Academy of Sciences, Beijing, China; with Shanghai Jiao Tong University, Shanghai 200052, China; with the University of Science and Technology of China, Hefei, China; and also with Old Dominion University, Norfolk, VA 23529 USA.

B. Xu is with the College of Economics and Management, Shanghai Jiao Tong University, Shanghai 200052, China (e-mail: byxu@sjtu.edu.cn).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TII.2014.2306384

and robust method using posterior-based approximate joint compatibility test to implement data association. In [6], a five-layer system architecture is proposed to integrate wireless sensor network (WSN) and RFID technologies. In order to support multi-users to do data updating or reading, these databases usually sacrifice some features such as database-wide transaction and consistency to achieve higher availability and scalability [7].

In data storage, many traditional data storage platforms are based on relational database. Although relational databases are still prominent for its data storage, they can hardly provide sufficient performance in big data environment. As complements to relational databases, those tools that can efficiently process massive data in distributed environment, such as Hadoop, and NoSQL database are attracting increasing attention. NoSQL databases provide a series of features that relational databases cannot provide, such as horizontal scalability, memory and distributed index, dynamically modifying data schema, etc. [8]. On the other hand, NoSQL database is lack of completed atomicity, consistency, isolation, durability (ACID) constraint and support for some complicated queries. Hadoop is an open source implementation of Google MapReduce, which supports processing massive data with high performance by MapReduce [9]. Many scholars have conducted researches on storing and processing data with NoSQL database and Hadoop. An architecture that combines Hadoop and parallel database has been developed, which allows uniform accessing and managing [10]. A framework is also proposed, which allows developers to use structured query language (SQL) to operate on both relational database and NoSQL database [11]. A save our systems (SOS) application programming interface (API) is proposed to provide a uniform interface for multiple types of NoSQL databases [12]. However, this API does not support relational databases.

Application supporting technology of IoT in cloud environment focuses on data access and data preference isolation. Services provided a promising technique to access data in distributed environment [13]. A privacy-enhanced discovery service, named SHARDIS [14], is also provided for RFID-based product information. Data preference isolation aims to facilitate multitenants secure data sharing for application purposes. Multitenant database based on relational database in [15] is compared with three patterns of data isolation, which are independent databases, sharing table and independent table in a sharing database. The algorithm introduced in [16] is used to realize multitenants isolation based on sharing table, in which a logical table is divided into several chunks, and these chunks are mapping into different physical tenant. In [17], a dynamic data instance allocation by resource calculations with constraints for multitenant in software as a service (SaaS) applications is proposed.

All these researches have provided useful references for data disposing in IoT framework. However, in cloud computing environment, a unified data storage approach covering structural data and unstructured data, data preference isolation, and effective service invocation are needed to be considered in IoT application development.

III. Architecture

To meet the requirements of managing massive IoT data in cloud platform, the data storage framework should deal with various types of data, which are collected from many different devices, such as RFID readers, monitors, thermometers, etc. These data are different in data structures, volume, accessing methods, and some other aspects, as such; they can hardly be stored and accessed efficiently by a single method. Besides, the data volume may increase quite rapidly, so that the framework must be able to process the data with a high throughput.

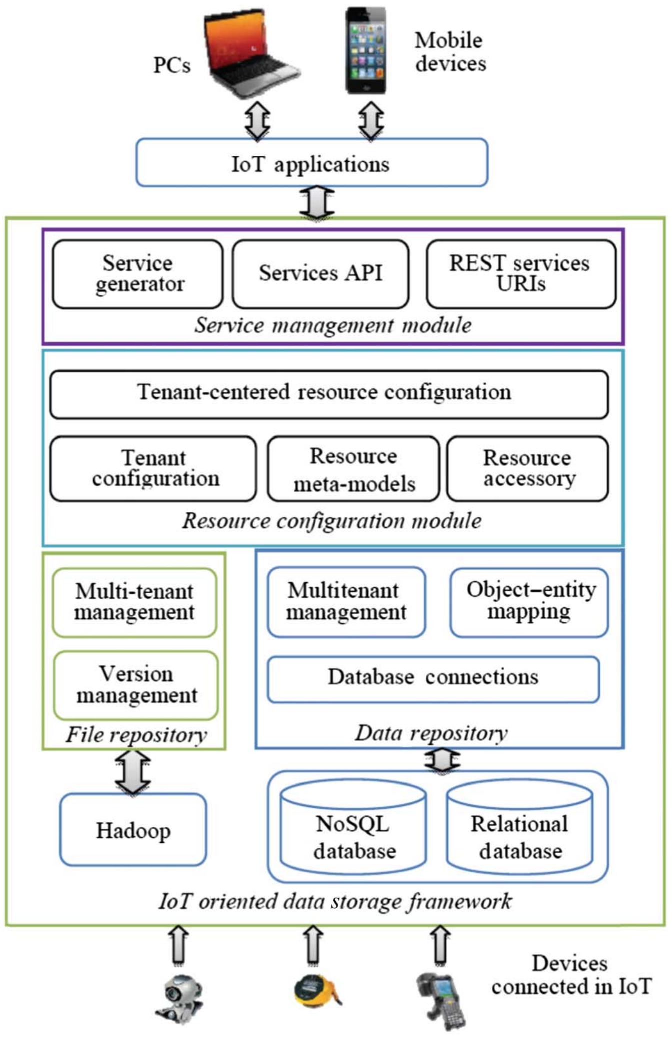


Fig. 1. Architecture of the proposed framework.

The architecture of the proposed framework is shown in Fig. 1. The data storage framework consists of several modules.

1) *File repository*: File repository makes use of hadoop distributed file system (HDFS) to store unstructured files in a distributed environment. We also add a version manager and a multitenant manager to implement the management of the versioned model files and the isolation of tenants' data. A file processor is used to improve the file repository's ability for handling small files.
2) *Database module*: Database module combines multiple databases and uses both NoSQL database and relational database for managing structured data. This module also provides unified API and object–entity mapping for multiple databases to hide their differences in implementing and interfacing so that the development of data access

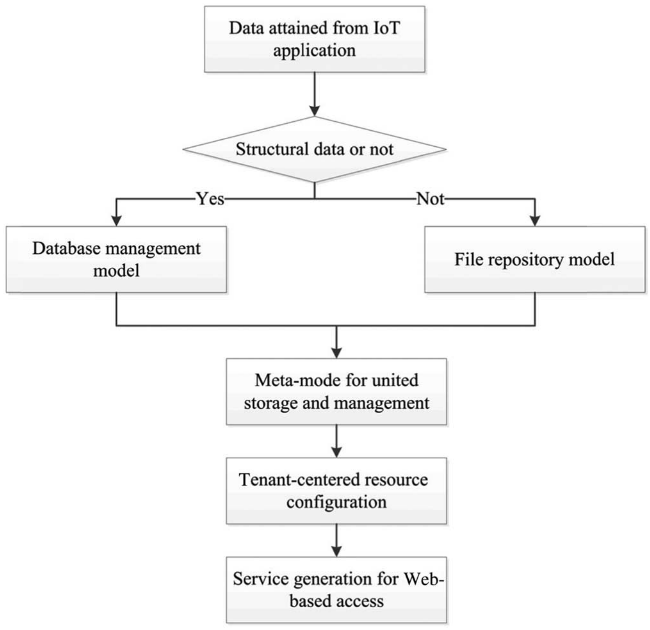

Data attained from IoT application
Structural data or not
Yes
Not
Database management model
File repository model
Meta-mode for united storage and management
Tenant-centered resource configuration
Service generation for Web-based access

Fig. 2. Simplified flowchart of the approach.

modules and the application migration of databases can be simplified.

3) *Service module*: Service module is built to generate RESTful service automatically. This module extracts the metadata through configuration, then mapping to the data entities and files stored in the databases and file repository according to the extracted metadata and finally generating corresponding RESTful service.
4) *Resource configuration module*: Resource configuration module supports static and dynamic data management in terms of predefined meta-model. Thus, data resource and related services can be configured based on tenant requirements. Furthermore, data disposing mechanism such as load balanced and isolated preferences can also be carried out.

IV. Approach

In this section, the details of the approach and the implementation of the data storage framework will be discussed in the following aspects: 1) database management model; 2) file repository model; 3) resource configuration; and 4) RESTful service generating.

A. Flowchart of the Approach

Based on data disposing process, the approach is divided into several stages as data acquisition and storage, resource configuration, and data utility. A simple flowchart of the approach is shown in Fig. 2.

B. Database Management Model

The key task of the database management is combining multiple databases and unifying access interfaces. Object–entity mapping and query adapting are the main approaches used in this model. Moreover, the model also integrates multitenant data isolation mechanism.

1) Object–Entity Mapping: Object–entity mapping maps objects in real world to entities in databases so that developers can operate data in databases as well as operate the objects in real world. Since there are multiple databases, mapping for different databases is considered through abstracting the structures of data collected.

The mainstream NoSQL databases can be classified into four categories in terms of data storage model: key-value store, document store, column store, and graph store. Although these storage models are quite different, the record structure that contains properties can be located and mapped to an object. A series of the records with the same property collection can be mapped to a data entity such as a table in a relational database. For example, in key-value store, a key-value pair can be treated as a property of a record and a group of pairs can be treated as the properties of a record and a series of groups form a data collection.

Besides mapping from objects to entities, the relations between entities also need to be maintained. Generally, NoSQL databases have no support for foreign key constraint, so this feature has to be entirely provided by our framework. The key to the problem is the way how the foreign key is stored. The solution provided in our framework to this problem is based on Pedro's design [18] with some modifications. In order to avoid extra join operations, each record's foreign keys are stored as a single property instead of an extra table (for an NoSQL database, the term "table" indicates the structure, which is equivalent to the table in the relational databases, such as "collection" in MongoDB). For one-to-one and many-to-one relations, the foreign key property stores only one value, while for one-to-many and many-to-many relations the foreign key property will be normalized to store a value set. In addition, the operations to deal with a value set must be implemented if the database does not provide, such as "contains" (returns a Boolean value to indicate whether the set contains the given value), "add" (adds a given value to the set; if the value already exists in the set, no action will be taken), "remove" (removes the given value from the set if the value exists in the set), etc.

2) Query Adapting Method: The unified queries created by calling the unified API cannot be directly executed by databases. They must be translated to the queries by a group of adapters so that the databases can accept them. For the relational database, the procedure of adapting means translating unified queries into SQL sentences, which has been implemented by ORM frameworks.

The biggest challenge for implementing the adaptors is that some functions provided by relational databases are not supported by NoSQL databases. Thus, the adaptors cannot directly translate the queries containing operations not supported by the target database. We need to implement the operations outside the databases. The implementation of most functions is straightforward, such that the value restrictions are not supported by NoSQL databases. We implement them by filtering the result record set (RS) before it is returned to the requestor; however, the joining operation is relatively complex to implement.

Join operations are well supported by a relational database management system (RDBMS), while mainstream NoSQL databases do not provide API for them, since the joining operations cannot be efficiently executed by NoSQL databases according to their architecture and design. In addition, the joining operation should be able to join entities stored in different databases so that the databases can be combined more seamlessly. Considering a query with join operation, we define as the following:

Definition 1: Restriction (R): A *restriction* is a condition that restricts the range of the value of a property. It is defined as the form of a tuple as follows:

$$\langle R\rangle ::= \langle e, p, o, v\rangle$$
$$(e \in E, p \in P(e), o \in Op).$$

Here, E represents the set of all entities stored in databases, e is an entity, and P represents a property set. $P(e)$ represents the set of entity e's all properties. Op represents the set of operators that indicate a constraint for a value, such as less than ($<$) and equal to ($=$). v is the value that is used to restrict the value range of the property combining with the operator.

Definition 2: Join Restriction (JR): A JR is used to indicate how the two entities are joined. It connects two entities by restricting the relation of the range of values of a property of each entity. For instance, "book.author = author.name." It is defined as the form of a tuple as follows:

$$\langle \mathrm{JR}\rangle ::= \langle e1, p1, o, e2, p2\rangle$$
$$(e1, e2 \in E, p1 \in P(e1),\ p2 \in p(e2),\ o \in Op).$$

Here, $e1$ and $e2$ represent the two entities to be joined and the entity represented by $e1$ is the main entity, $p1$ is a property of $e1$, $p2$ is a property of $e2$, and $p1$ and $p2$ must be of the same type. The operator o restricts the relationship between $p1$ and $p2$.

Definition 3: Non-Join Query (NJQ): An NJQ is a query that does not contain any join operation. It can be defined as the form of a tuple as follows:

$$\langle \mathrm{NJQ}\rangle ::= \langle e, i, \mathrm{Restriction\_Set}\rangle$$
$$(e \in E, i \in I).$$

Here, e represents the target entity of the query, I represents the query instruction set with elements indicating the base operations of queries, such as finding, saving, updating, and deleting. Generally, a query must be assigned one and only one instruction i, which is a member of set I. Restriction_Set is a set of restrictions used to describe the conditions of the query. Restriction_Set must be finite and can be empty. If a query has an empty Restriction_Set, then the query will operate all records in the target table mapped to the entity e.

Definition 4: Join Query (JQ): A JQ is a query that contains one or more join operations. It can be defined as the form of a tuple as follows:

$$\langle \mathrm{JQ}\rangle ::= \langle e, i, \mathrm{Restriction\_Set},\ \mathrm{JoinRestriction\_Set}\rangle$$
$$(e \in E, i \in I).$$

Here, i and Restriction_Set have the same meaning with those in Definition 3. JoinRestriction_Set is a set of JRs. If a query needs to join n entities ($n > 2$), the joining operation will be converted into $n - 1$ joining restrictions of which each joins two entities.

Definition 5: Record Set (RS): A RS is a collection of the records. A RS can indicate either the result of a query containing finding instruction or the records stored in a table. An RS can be described as the form of a tuple as follows:

$$\langle \mathrm{RS}\rangle ::= \langle e, \mathrm{Record\_Set}\rangle$$
$$(e \in E).$$

Here, if the record is the data of a table, e represents the entity mapped to the table; if the RS is the result of a query, e represents the entity mapped to the class of the record objects. Record_Set is the data, which may be a list of record objects or the records stored in a table.

The procedure for executing a JQ can be described by a pseudo code segment containing a recursive function (executeJR) as follows:

```
Function executeJR (JR, $\mathrm{JoinRestriction\_Set}$)
result_set := $\emptyset$;
$\mathrm{JoinRestriction\_Set} := \mathrm{JoinRestriction\_Set\text{-}JR}$;
$\mathrm{rs1} := \{r | r \in \mathrm{JR.Restriction\_Set} \wedge r.e == \mathrm{JR.e1}\}$;
$\mathrm{record\_set1} := \mathrm{execute(NJQ(JR.e1, finding, rs1))}$;
If ($\exists \mathrm{jr2}(\mathrm{jr2} \in \mathrm{JoinRestriction\_Set} \wedge \mathrm{jr2}.e1 == \mathrm{JR.e2})$) do
    $\mathrm{record\_set2} := \mathrm{executeJR(jr2, JoinRestriction\_Set)}$;
  else
    $\mathrm{rs2} := \{\mathrm{r|r} \in \mathrm{Restriction\_Set} \wedge \mathrm{r.e} == \mathrm{JR.e2}\}$;
    $\mathrm{record\_set2} := \mathrm{execute(NJQ(JR.e2, finding, rs2))}$;
  End if
  For (record1 $\in$ result_set1) do
    For (resord2 $\in$ result_set2) do
    If (satisfy(JR, record1, record2)) do
      result_set := result_set $\cup$ {record1};
    break;
    else
      continue;
    end if
    end For
  end For
  return result_set;
End function
```

```
Procedure executeJoinQuery (JQ)
  executed := JQ.JoinRestriction_Set;
  rs := null; //the result set
  For (JR $\in$ JQ.JoinRestriction_Set $\wedge$ JR.e1 == JQ.e) do
    If (rs == null)
      rs := executeJR(JR, executed);
    else
      rs := rs $\cap$ executeJR(JR, executed);
    end if
  end For
  return rs;
end Procedure
```

C. File Repository Model

File repository model describes the method of the file repository managing unstructured data files. File repository is based on Hadoop's HDFS. HDFS is also extended and wrapped to implement the features required by the file repository.

The data generated by the devices in the IoT are valuable only if they can be identified. For example, if we store a video file, although we cannot find out where and when the video is generated, then the video file is worthless. Thus, the data storage framework should not only store the data but also their information of the two dimensions: time and space. For the data generated in the IoT, we use a generation timestamp and an electronic product code (EPC) of the devices to identify the data generated by a device in one time.

In the databases, it is easy to store the timestamps and EPCs as two properties in the data collections, whereas in the file repository, the condition is different. The file repository does not provide extra space to store the two properties in a file because storing the properties in the file might break the original file structure and cause other problems. Our method is storing the timestamp and EPC in the corresponding file's name. Every IoT data file is named with a string consists of the EPC and the timestamp. In the file name, the first part is the EPC that is transformed to a 24-digit hexadecimal number and the second part is the timestamp with in the form of a decimal.

D. Resource Configuration for Tenants

Based on the difference of tenants, resources are configured so as to archive effective execution performances. Tenants, services, meta-models, and configuration policy are involved in this module.

1) *Tenant configuration*: Based on tenant, data resources, and related elements are allocated, such as data authority, services, and ranks. Service-level agreements (SLAs) are also set by tenant configuration.
2) *Service interfaces*: When data resources are allocated, related services interfaces could also be configured for distributed users' utility.

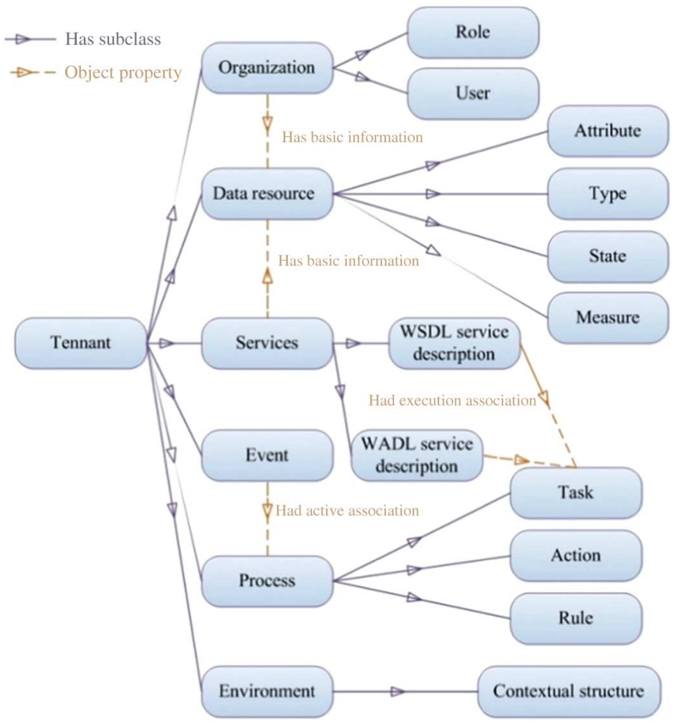


Fig. 3. Metamodel for tenant-centered resources configuration.

3) *Resource configuration*: The module provides policy for tenant-centered resource configuration. Based on execution and management requirements, resources are divided into different parts for load balance and different business purpose such as design area, testing area, and execution area especially in a cloud environment.
4) Meta-models are the core for resource configuration module, which contained relations of data, tenant, service, and events. In fact, meta-models are built based on multiviews business modeling, which could form a semantic base for further data disposing purpose.

Fig. 3 shows the meta-model for data resources allocation. The data resources are divided into different distributed areas for the purposes such as load balance or preference isolation.

Based on these unified view, multitenant configuration and management for structural data and files can be implemented. In fact, tenant is configured in this module, but multitenant management is realized in the different file repository and database modules.

1) Multitenant Configuration for Structured Data: In the proposed framework, multitenant management is related to the method of isolating the private data of every tenant and sharing the public data. Every tenant's private data should be invisible to other tenants. Besides, all of the operations of a tenant should be limited to the tenant's private data and should not affect other tenants' private data.

In order to maintain the balance of performance and the data isolation, we take a separated data collection strategy to implement the data isolation. This strategy separately stores different tenants' data of a same entity in different collections. It provides physical isolation for private data and has an acceptable cost. This strategy also supports a flexible configuration for data

TABLE I
RESOURCE METADATA IN WADL FILE

WADL element	Child element	Example
Complex type	Element	<xs:complexType name="Order"> <xs:element name="id" type="xs:string"/> </xs:complexType>
Resource	Param	<resource path="/{name}/"> <param name="name" style="template" type="xs:string"/> </resource>
Grammars	Include	<grammars> <include href="…/location.xsd"/> </grammars>

sharing by allowing controlling whether a data entity is private or shared. If an entity is set private, the system creates a data collection for every tenant to store private data, which is invisible to other tenants. A tenant's operation on private data will be relocated to its own private data area so that other tenants' data will not be affected.

2) Multitenant Configuration for Unstructured Files: The file repository adopts a data isolation and sharing mechanism allowing controlling the file privacy with file granularity. The basic idea of this mechanism is assigning a workspace for every tenant. The workspace is a directory in the HDFS named by its owner's tenant ID. There is also a sharing space for storing sharing files. For a tenant, only its private workspace and the sharing space are visible. We wrapped the access interface of HDFS, so that all of the operations on the files are limited in the tenants' private workspace and sharing space.

E. RESTful Service Generating

RESTful service has gained widespread acceptance across the Web. It is lightweight and stateless. A RESTful service is exposed as a resource with a unique URI and can be manipulated by HTTP request. The uniform interface enables the resources to be accessed by clients on different platforms [19]. Complex data disposing process can be realized through resource-oriented service composition [20].

The procedure for generating RESTful service can be divided into several stages: 1) configuring resource metadata; 2) mapping resource; and 3) generating service interface.

1) Configuring Resource Metadata: The metadata of resources can be configured by parsing web application description language (WADL) files. All of the metadata corresponding to the resources is contained in the WADL files. The metadata is expressed in XML form, as shown in Table I. We can collect the information of the data contained in the corresponding resource, such as the types of the data and the URI of the resource.

2) Mapping Resource: The metadata of information resources describe the interfaces exposed for the services' consumer, whereas they are not necessarily a precise description of the data entities or files stored in the data storage framework, thus we have to map the resource to the real data by providing the resource-data map in the configuration files.

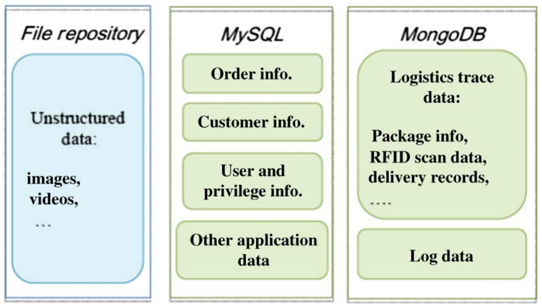


Fig. 4. Data distribution over the data storage framework.

3) Generating Service Interface: Since the resources have been mapped to the real data, we need to expose them to the Web so that they can be accessed by the consumers of the resources. According to the features of RESTful services, the services accept HTTP requests to manipulate the resources. The methods the service should expose are also described in the WADL files. Generally, four types of HTTP requests are mapped to the four types of operations: "GET" maps retrieving, "POST" maps creating, "PUT" map updating, and "DELETE" maps deleting. So we build the service methods according to the map and translate the HTTP requests to the operations on the resources.

V. CASE STUDY AND DISCUSSION

A. Case Study

In this section, we use a logistics delivery scenario as an illustrative example to show how the framework works. In the logistics scenario, a large amount of logistics orders are traced by IoT-based technologies such as RFID readers, sensors, and cameras. The data generated by the devices are first collected and preprocessed by some terminals and then sent to a logistics management application, which is built based on the data storage framework. In order to improve the performance of data storing and accessing, all types of data are stored in different places (MySQLdatabase, MongoDB, and file repository), as shown in Fig. 4.

Then, we take stored logistics data and accessed logistics data as examples to show how the data storage framework supports storing and managing data in runtime.

1) Storing Logistics Data: When the logistics system operates, each package being delivered is traced by a series of devices and generates data frequently. Fig. 5 shows the process of the data generated by the devices being stored in the data storage framework. The data generated by devices are preprocessed in some terminals before they are sent to the logistics management system so that they can be well received. The logistics management system then stores the tracing data in the data storage framework. The data are first divided into structured data, which are stored in the databases, and unstructured data, which are stored in the file repository. Furthermore, the structured data are separately stored in different databases according to the metadata contained in configuration.

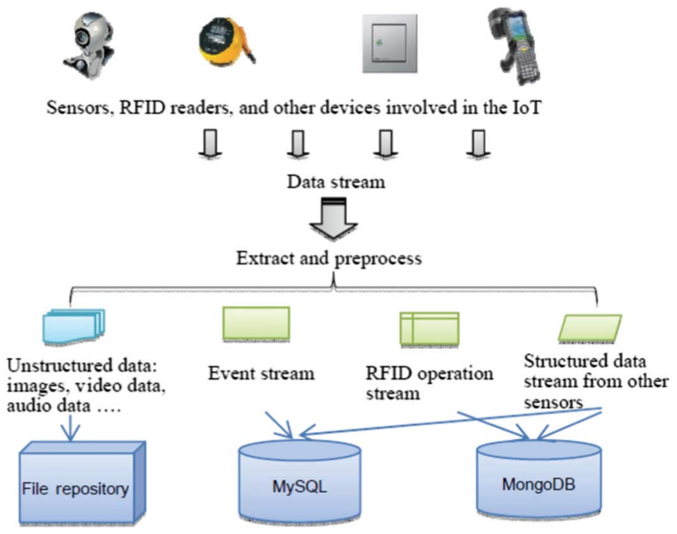

Sensors, RFID readers, and other devices involved in the IoT
Data stream
Extract and preprocess
Unstructured data: images, video data, audio data ....
Event stream
RFID operation stream
Structured data stream from other sensors
File repository
MySQL
MongoDB

Fig. 5. Process of storing logistics data.

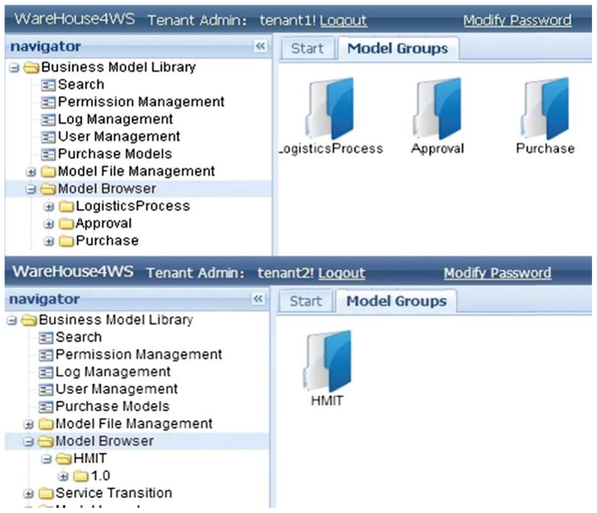
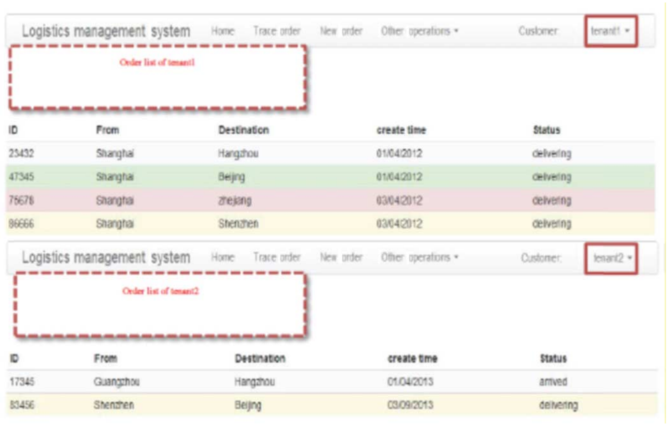


Fig. 6. Information storage and configuration based on tenants.

2) Data Resource Configuration and Storage: When related data from distributed data sources are stored in the platform, we can configure and manage such information for different tenants, as shown in Fig. 6. The IoT data files stored in the file repository are organized in two dimensions. Every file can be precisely located through providing the EPC of the device, which generates the data and the timestamp of the generation. If one EPC is provided, the file repository will list all the files containing the data generated by the device corresponding to the EPC.

3) Accessing Logistics Data: For the purpose of realizing data performance isolation, we store data in independent tables or files but allow sharing a database when disposing structured data. Unstructured information is stored in files with private space. Thus, all the data and files are isolated for different tenants. A tenant can only access own private data and sharing data. As shown in Fig. 7, when we login as the account "tenant1" and



ID	From	Destination	create time	Status
23432	Shanghai	Hangzhou	01/04/2012	delivering
47345	Shanghai	Beijing	01/04/2012	delivering
75678	Shanghai	zhejiang	03/04/2012	delivering
86666	Shanghai	Shenzhen	03/04/2012	delivering

ID	From	Destination	create time	Status
17345	Guangzhou	Hangzhou	01/04/2013	arrived
83456	Shenzhen	Beijing	03/09/2013	delivering

Fig. 7. Order lists of different tenants.

TABLE II
COMPARISON

Feature	SOS interface	Curé's solution	Our solution
Interface style	Object-oriented API	SQL	Object-oriented API
Databases supported	NoSQL databases	NoSQL databases and relational databases	NoSQL databases and relational databases
Configurability	Not mentioned	Support but not flexible	Support flexible configuration
Support for join between different databases	Not support	Not support	Support
Integrated features	No	No	RESTful service, multitenant and version management

browse the order list, we see three orders. When logged in as "tenant2," we see another order list.

B. Discussion

Compared with other methods for integrating multiple databases for storing and accessing massive data [11], [12], our solution has several advantages, as shown in Table II.

From the above comparison, our solution provides a more comprehensive data support framework that combines relational database and NoSQL database, as well as a flexible configurable data disposing platform for IoT application in cloud environment.

The data storage framework is designed to solve the problems brought by massive structured and unstructured IoT data. The framework combines different types of databases and provides unified accessing interface so that different kinds of data can be stored in different databases, according to the nature of the data and operated by the same interface. Besides, we improved the Hadoop platform to realize a distributed file repository, which supports version management of unstructured data files and multitenant data isolation. Moreover, in order to support the model driven development, the solution provides the function of generating RESTful service, which facilitate the transformation from models to services.

VI. Conclusion and Future Work

IoT technology has been rapidly developed in the last few years and is increasingly impacting various industrial sectors [21]–[25]. The IoT-oriented data storage framework in the cloud platform is expected to provide IoT data storage, access, and management service [26]–[28]. In the aspect of data storing and accessing, it faces a series of challenges: large volume of data, different data types, rapid generating data, complicated requirements of data management, etc. In this paper, we propose a framework as a feasible solution to the challenges. For the structured data, we create a database management model that combines and extends multiple databases and provides unified accessing API for simplified development and maintenance. For unstructured data, the framework wraps and extends HDFS based on the file repository model to implement version management and multitenant data isolation. Moreover, in order to support remote and cross-platform data access, the data framework integrates RESTful service generating mechanism to provide platform-independent HTTP interface. The IoT-oriented data storage framework in the cloud platform is expected to be applied a variety of applications [29]–[39].

In the future, we will explore the possibility for further optimizing the performance of our framework, and integrating more practical features for IoT data management. In the current version of data storage framework, only limited types of adapters for databases are implemented. We will append more adaptors for other NoSQL databases in future research.

References

[1] L. Atzori, A. Iera, and G. Morabito, “The Internet of Things: A survey,” *Comput. Netw.*, vol. 54, no. 15, pp. 2787–2805, 2010.

[2] L. Tan and N. Wang, “Future internet: The Internet of Things,” in *Proc. 3rd Int. Conf. Adv. Comput. Theory Eng.* (ICACTE), Aug. 2010, vol. 5, pp. v5-376–v5-380.

[3] H. Schweppe, A. Zimmermann, and D. Grill, “Flexible on-board stream processing for automotive sensor data,” *IEEE Trans. Ind. Informat.*, vol. 6, no. 1, pp. 81–92, Feb. 2010.

[4] S. Li, L. Xu, and X. Wang, “Compressed sensing signal and data acquisition in wireless sensor networks and Internet of Things,” *IEEE Trans. Ind. Informat.*, vol. 9, no. 4, pp. 2177–2186, Nov. 2013.

[5] Y. Li, S. Li, Q. Song, and H. Liu, “Fast and robust data association using posterior based approximate joint compatibility test,” *IEEE Trans. Ind. Informat.*, vol. 10, no. 1, pp. 331–339, Feb. 2014.

[6] L. Wang, L. Xu, Z. Bi, and Y. Xu, “Data filtering for RFID and WSN integration,” *IEEE Trans. Ind. Informat.*, vol. 10, no. 1, pp. 408–418, Feb. 2014.

[7] R. Cattell, “Scalable SQL and NoSQL data stores,” *ACM SIGMOD Record*, vol. 39, no. 4, pp. 12–27, 2010.

[8] J. Guo, L. Xu, G. Xiao, and Z. Gong, “Improving multilingual semantic interoperation in cross-organizational enterprise systems through concept disambiguation,” *IEEE Trans. Ind. Informat.*, vol. 8, no. 3, pp. 647–658, Aug. 2012.

[9] K. Shvachko, H. Kuang, S. Radia, and R. Chansler, “The Hadoop distributed file system,” in *Proc. IEEE 26th Symp. Mass Storage Syst. Technol.* (MSST), 2010, pp. 1–10.

[10] Y. Xu, P. Kostamaa, and L. Gao, “Integrating Hadoop and parallel DBMs,” in *Proc. ACM SIGMOD Int. Conf. Manage. Data,* 2010, pp. 969–974.

[11] O. Curé, R. Hecht, C. Duc, and M. Lamolle, “Data integration over NoSQL stores using access path based mappings,” *Lect. Notes Comput. Sci.*, vol. 6860, pp. 481–495, 2011.

[12] P. Atzeni, F. Bugiotti, and L. Rossi, “SOS (Save Our Systems): A uniform programming interface for non-relational systems,” in *Proc. 15th Int. Conf. Extending Database Technol.,* 2012, pp. 582–585.

[13] R. Kyusakov, J. Eliasson, J. Delsing, J. van Deventer, and J. Gustafsson, “Integration of wireless sensor and actuator nodes with IT infrastructure using service-oriented architecture,” *IEEE Trans. Ind. Informat.*, vol. 9, no. 1, pp. 43–51, Feb. 2013.

[14] B. Fabian, T. Ermakova, and C. Müller, “SHARDIS: A privacy-enhanced discovery service for RFID-Based product information,” *IEEE Trans. Ind. Informat.*, vol. 8, no. 3, pp. 707–716, Aug. 2012.

[15] D. Jacobs and S. Aulbach, “Ruminations on multi-tenant databases,” *BTW Proc.*, vol. 103, pp. 514–521, 2007.

[16] S. Aulbach, T. Grust, and D. Jacobs, “Multi-tenant databases for software as a service: Schema-mappingtechniques,” in *Proc. 2005 ACM SIGMOD Int. Conf. Manage. Data,* 2008, pp. 1195–1206.

[17] T. Kwok and A. Mohindra, “Resource calculations with constraints, and placement oftenants and instances for multi-tenant SaaS applications,” *Lect. Notes Comput. Sci.*, vol. 5364, pp. 633–648, 2008.

[18] F. Cruz, P. Gomes, R. Oliveira, and J. Pereira, “Assessing NoSQL Databases for Telecom Applications,” in *IEEE Conf. Commerce Enterprise Comput.,* 2011, pp. 267–270.

[19] A. Marinos, E. Wilde, and J. Lu, “HTTP database connector (HDBC): RESTful access to relational databases,” in *Proc. 19th Int. Conf. World Wide Web,* 2010, pp. 1157–1158.

[20] C. Xie, H. Cai, and L. Jiang, “Ontology combined structural and operational semantics for resource-oriented service composition,” *J. Univers. Comput. Sci.*, vol. 19, no. 13, pp. 1963–1985, 2013.

[21] L. Xu, “Enterprise systems: State-of-the-art and future trends,” *IEEE Trans. Ind. Informat.*, vol. 7, no. 4, pp. 630–640, Nov. 2011.

[22] L. Li, “Technology designed to combat fakes in the global supply chain,” *Bus. Horiz.*, vol. 56, no. 2, pp. 167–177, 2013.

[23] S. Fang, L. Xu, H. Pei, and Y. Liu, “An integrated approach to snowmelt flood forecasting in water resource management,” *IEEE Trans. Ind. Informat.*, vol. 10, no. 1, pp. 548–558, Feb. 2014.

[24] L. Xu, “Introduction: Systems science in industrial sectors,” *Syst. Res. Behav. Sci.*, vol. 30, no. 3, pp. 211–213, 2013.

[25] L. Xu, “Information architecture for supply chain quality management,” *Int. J. Prod. Res.*, vol. 49, no. 1, pp. 183–198, 2011.

[26] S. Li, L. Xu, X. Wang, and J. Wang, “Integration of hybrid wireless networks in cloud services oriented enterprise information systems,” *Enterp. Inf. Syst.*, vol. 6, no. 2, pp. 165–187, 2012.

[27] Q. Li, Z. Wang, W. Li, J. Li, C. Wang, and R. Du, “Applications integration in a hybrid cloud computing environment: Modeling and platform,” *Enterp. Inf. Syst.*, vol. 7, no. 3, pp. 237–271, 2013.

[28] L. Ren, L. Zhang, F. Tao, X. Zhang, Y. Luo, and Y. Zhang, “A methodology towards virtualization-based high performance simulation platform supporting multidisciplinary design of complex products,” *Enterp. Inf. Syst.*, vol. 6, no. 3, pp. 267–290, 2012.

[29] B. M. Wilamowski and O. Kaynak, “Oil well diagnosis by sensing terminal characteristics of the induction motor,” *IEEE Trans. Ind. Electron.*, vol. 47, no. 5, pp. 1100–1107, Oct. 2000.

[30] M. Kataev, L. Bulysheva, A. Emelyanenko, and V. Emelyanenko, “Enterprise systems in Russia: 1992–2012,” *Enterp. Inf. Syst.*, vol. 7, no. 2, pp. 169–186, 2013.

[31] X. Chen, A. Guan, X. Qiu, H. Huang, J. Liu, and H. Duan, “Data configuration in railway signaling engineering-an application of enterprise systems techniques,” *Enterp. Inf. Syst.*, vol. 7, no. 3, pp. 354–374, 2013.

[32] X. Wang and X. Xu, “DIMP: An interoperable solution for software integration and product data exchange,” *Enterp. Inf. Syst.*, vol. 6, no. 3, pp. 291–314, 2012.

[33] X. Chen and Y. Fang, “Enterprise systems in financial sector-an application in precious metal trading forecasting,” *Enterp. Inf. Syst.*, vol. 7, no. 4, pp. 558–568, 2013.

[34] C. Le, X. Gu, K. Pan, F. Dai, and G. Qi, “Public and expert collaborative evaluation model and algorithm for enterprise knowledge,” *Enterp. Inf. Syst.*, vol. 7, no. 3, pp. 375–393, 2013.

[35] N. Li, W. Yi, Z. Bi, H. Kong, and G. Gong, “An optimization method for complex product design,” *Enterp. Inf. Syst.*, vol. 7, no. 4, pp. 470–489, 2013.

[36] L. Li and J. Liu, “An efficient and flexible web services-based multidisciplinary design optimization framework for complex engineering systems,” *Enterp. Inf. Syst.*, vol. 6, no. 3, pp. 345–371, 2012.

[37] W. Tan, W. Xu, F. Yang, L. Xu, and C. Jiang, “A framework for service enterprise workflow simulation with multi-agents cooperation,” *Enterp. Inf. Syst.*, vol. 7, no. 4, pp. 523–542, 2013.

[38] H. Yu, T. Xie, S. Paszczynski, and B. Wilamowski, “Advantages of radial basis function networks for dynamic system design,” *IEEE Trans. Ind. Electron.*, vol. 58, no. 12, pp. 5438–5450, 2011.

[39] F. Wang, B. Ge, L. Zhang, Y. Chen, Y. Xin, and X. Li, “A system framework of security management in enterprise systems,” *Syst. Res. Behav. Sci.*, vol. 30, no. 3, pp. 287–299, 2013.

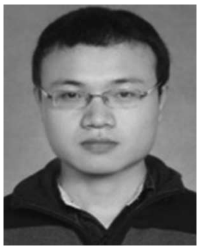
Lihong Jiang received the B.S. and M.S. degrees in computer science, and the Ph.D. degree in management science from Tianjin University, Tianjin, China, in 1989, 1992, and 1996, respectively.

Currently, she is an Associate Professor with the Software School, Shanghai Jiao Tong University, Shanghai, China. Her research interests include enterprise information system and business data analysis.

Li Da Xu (M'86–SM'11), photograph and biography not available at the time of publication.

Hongming Cai received the B.S., M.S., and Ph.D. degrees from Northwestern Polytechnical University, Shaanxi, China, in 1996, 1999, and 2002, respectively.

Currently, he is an Associate Professor with the School of Software, Shanghai Jiao Tong University, Shanghai, China. His research interests include business process management and information system integration.

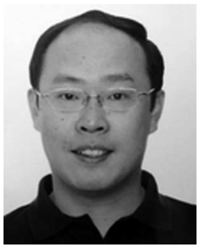


Zuhai Jiang received the B.S. degree from Central South University, Changsha, China, in 2011.

Currently, he is a Postgraduate Student at the Information System Technology Lab, School of Software, Shanghai Jiao Tong University, Shanghai, China. His research interests include data storage, data processing, and web services system.

Fenglin Bu received the B.S. and M.S. degrees from the School of Material Engineering, Shanghai Jiao Tong University, Shanghai, China, in 1982 and 1995, respectively.

He is currently an Associate Professor at the School of Software, Shanghai Jiao Tong University. His research interests include software engineering and business process management.



Boyi Xu received the B.S. degree in industrial automation and the Ph.D. degree in management science from Tianjin University, Tianjin, China, in 1987 and 1996, respectively.

Currently, he is an Associate Professor at the College of Economics and Management, Shanghai Jiao Tong University, Shanghai, China. His research interests include enterprise information systems, electronic commerce, and business intelligence.