CIT. Journal of Computing and Information Technology, Vol. 27, No. 4, December 2019, 17–30
doi: 10.20532/cit.2019.1004801

Temporal RDF(S) Data Storage and Query with HBase

Li Yan, Zheqing Zhang and Dan Yang

Nanjing University of Aeronautics and Astronautics, Nanjing, China

Resource Description Framework (RDF) is a metadata model recommended by World Wide Web Consortium (W3C) for describing the Web resources. With the arrival of the era of Big Data, very large amounts of RDF data are continuously being created and need to be stored for management. The traditional centralized RDF storage models cannot meet the need of large-scale RDF data storage. Meanwhile, the importance of temporal information management and processing has been acknowledged by academia and industry. In this paper, we propose a storage model to store temporal RDF based on HBase. The proposed storage model applies the built-in time mechanism of HBase. Our experiments on LUBM dataset with temporal information added show that our storage model can store large temporal RDF data and obtain good query efficiency.

ACM CCS (2012) Classification: Information systems → World Wide Web → Web data description languages → Semantic web description languages → Resource Description Framework (RDF)
Information systems → World Wide Web → Web data description languages → Markup languages → Extensible Markup Language (XML)
Information systems → Data management systems → Database design and models → Data model extensions → Temporal data

Keywords: temporal RDF, HBase, storage, query

1. Introduction

In recent years, Resource Description Framework (RDF), providing a complete grammar system and contributing to computer's automatic processing, has been widely used in various fields for its simplicity, extensibility, openness and ease of exchange [25]. This has resulted in the rapid growth of RDF data and the issue of efficient and scalable management of large-scale RDF data. RDF data management typically involves their storage and queries. Among them, RDF data storage provides the infrastructure for RDF data management [26].

To deal with massive RDF data, much work has been devoted to parallel computing techniques and distributed systems for improving the ability to manage RDF data. Efforts on distributed storage of RDF data mainly concentrate on two aspects [26]. The first aspect focuses on developing distributed RDF storage systems specifically with traditional distributed computing architectures, such as RDFPeers [2], 4store [3], Bigdata [4], and YARS [5]. These systems always have high reliability and expandability, but their data structures are complex, which may cause high communication overhead, and the security is difficult to control. The second aspect focuses on storing RDF data with NoSQL (Not Only SQL) databases, which have flexible data models and excellent performance in reading/writing for massive data. NoSQL databases can be divided into four basic categories: *key-value stores*, *document databases*, *column-oriented databases* and *graph databases* [6, 7, 8, 9]. The storage model proposed in this paper is based on Apache HBase [10], which is a column-oriented distributed database built on Apache Hadoop and is an open-source implementation of Bigtable [11]. Nowadays, many efforts are dedicated to efficient storage and query of massive RDF data based on HBase [12-19].

The real world is dynamic. Time is an essential dimension in describing data change and is hereby an important part of many applications [28, 29]. To represent and deal with temporal

data, in last two decades various temporal database models have been proposed and some temporal database management systems have been developed [30]. More importantly, time information has been introduced into RDF and temporal RDF model has been proposed [1, 32]. In the context of temporal RDF model, few issues such as *construction* [27], *query* [32] and *index* [33] have been investigated. We argue that massive RDF data are stored in NoSQL databases and essentially RDF data are temporally relevant. In order to efficiently manage massive temporal RDF data, it is crucial to store temporal RDF data in NoSQL databases. Unfortunately, the models and approaches proposed for storing classical (non-temporal) RDF data in NoSQL databases cannot be directly applied to storing temporal RDF data due to additional temporal information in RDF data. This is why very different approaches are proposed for dealing with temporal RDF data in [27, 32, 33] instead of directly applying the corresponding approaches for classical RDF data. To the best of our knowledge, there is not any report on temporal RDF storage although more attention has been paid to classical RDF data storage in databases. The present paper tries to fill this gap.

In this paper, we propose to apply HBase to storing temporal RDF data. We analyze the characteristics of HBase database and particularly identify possible problems of the built-in time mechanism in HBase database. On this basis, we propose a storage model for temporal RDF, which considers the learning experience of seniors and supports temporal RDF query. We verify our approach with experiments on LUBM dataset.

The rest of this paper is organized as follows: Section 2 presents a brief overview of related work. The temporal RDF is presented in Section 3. Section 4 proposes the storage model and query strategy of temporal RDF data based on HBase database. The experimental evaluations are presented in Section 5. Section 6 concludes the paper and sketches our future work.

2. Related Work

The prototype of HBase is Bigtable. Apache HBase provides Bigtable-like capabilities on top of Hadoop, which uses HDFS as file storage system and supports MapReduce, an open source computing architecture. In addition, HBase adopts a data structure called HTable, which is similar to traditional relational table. For this reason, the current RDF storage models in HBase almost refer to RDF's storage structure in relational databases. However, HBase is different from relational databases after all. At this point, considering the characteristics of HBase and the corresponding query methods that usually take advantage of MapReduce, some efforts are carried out for proposing new storage models.

In [13], six index tables are introduced to store RDF data, which are *S_PO*, *P_SO*, *O_SP*, *PS_O*, *SO_P*, and *PO_S*, respectively. Here (S, P, O) means a triplet with form of subject, predicate and object. The six index tables are reduced to three index tables in [14], which are very efficient for simple queries because all combinations of RDF triple patterns are covered. In addition to the storage schema, a MapReduce strategy is proposed for SPARQL BGP (Basic Graph Pattern) processing, which applies a greedy method to select join key and eliminated multiple triple patterns. In [15], queries are processed by connecting HBase to Jena, a well-known SPARQL query processor. Jena-HBase is created in [16]. Apart from this, they propose various triple storage schemas and evaluate those schemas in terms of query processing time based on Jena-HBase. H2RDF, a query system for RDF based on HBase and MapReduce, is developed in [17], which uses SP_O, PO_S, and OS_P index tables to reduce data redundancy. In view of queries, H2RDF firstly parses SPARQL queries through Jena, and then uses MapReduce or Centralized query according to the join complexity to ensure the query performance.

A vertical partition like model is designed in [19], which creates two tables (Pso, Pos) for each predicate and occupies less space. To deal with complex query, they propose a path index to reduce the numbers of join operation. In [20], a hybrid storage schema is adopted, which is a combination of simple triple and vertical partition storage models. As a result, triples are compressed by sequential encoding keys and storage space is managed efficiently.

In [18], provenance datasets are serialized as RDF graphs which are stored in HBase [18]. Here, an RDF graph identifier is used as the unique row id, a complete RDF graph is stored as one aggregate value in data column family,

and bitmap indices are compacted to perform expensive query processing operations. Triples are separately stored in HBase tables according to the subject's class in [21], where a table P is created in order to store all <subject, object> with the same predicate. All data is divided by classes so that each HBase table is relatively small. This can lead to a better performance of table query and traversal.

Note that the proposals for RDF data storage mentioned above do not consider temporal information, both in RDF data model and in HBase. Actually, the classical RDF data model and HBase do not explicitly support temporal information modeling. The timestamp mechanism in HBase can be applied to record multiple values, but its implicit temporal interval representation can cause wrong or misleading results during temporal query processing. In [22], the characteristics of column-oriented NoSQL databases are clearly clarified and two alternative table representations are introduced to explicitly address temporal data management and processing. In the context of RDF data model, temporal RDF data model is proposed for temporal information modeling in [1, 32]. Being different from the classical RDF data model, the temporal RDF data model contains temporal information and cannot be directly stored in the classical HBase database by using the existing solutions of RDF data storage in HBase. Although there are a few efforts in temporal RDF data management (*e.g.*, *construction* [27], *query* [32] and *index* [33]), to the best of our knowledge, the present paper is the first effort in storing temporal RDF data with HBase database.

3. Temporal RDF

RDF is a metadata model for building an infrastructure of machine-readable semantics for data on the Web. The RDF specification includes a built-in vocabulary with a normative semantics (RDFS), which deals with inheritance of classes and properties. Temporal RDF based on the point-based temporal domain is proposed in [1].

Definition 1 (Temporal RDF model). A simple temporal interpretation of RDF is a tuple (I, T, M), in which $I = \{I_1, \ldots, I_n\}$ is a set of simple interpretations. In $I_i = (C_i, P_i, R_i, Ext, CExt)$, C_i is a set of classes; P_i is a set of properties; R_i is the set of all resources, which is actually the universe of RDF, containing a distinguished subset L_i called literal values; Ext: $P_i \rightarrow R_i \times R_i$ is used to express the relationship between resources; $CExt$: $C_i \rightarrow 2^{Ri}$ maps each class $c \in C_i$ to a subset of R_i (*i.e.* $C_i = CExt(rdfs: Class)$, which means that each element of C_i is an extension of *rdfs*: *Class*).

T is a set of times. M: $I \rightarrow 2^T$ is a timestamp function that maps an interpretation to a timestamp (a set of times).

The temporal RDF model is defined with only one time-dimension, which can be further extended to multi-dimension. The temporal RDF model used in this paper has two time-dimensions, which are valid time and transaction time, respectively. Here, the time points are encoded as intervals. A time point set {1, 2, 3}, for example, is encoded as [1, 3] and a single time point {1} can be described as [1, 1]. Then, a temporal RDF triple is represented as follows:

$$(Subject, Predicate, Object): t,\ t = [V][T]$$
$$Valid(t) = [V],\ Transaction(t) = [T].$$

Here, (*Subject*, *Predicate*, *Object*) is a standard RDF triple and t is a temporal label composed by $[V]$ and $[T]$, which is applied at the level of triple. $[V]$ is the valid time of a triple, which refers to the time when the data is true in the modeled reality. $[T]$ records the transaction time when the triple is edited. In addition, *Valid*: $1 \rightarrow [V]$ and *Transaction*: $t \rightarrow [T]$ are defined to get valid time and transaction time from time label t, respectively.

A temporal RDF query language named T-SPARQL is proposed in [23]. T-SPARQL temporally extends the standard query language SPARQL, which is characterized by graph patterns. BGP is the basic query mode of graph patterns, consisting of a set of triple patterns. All triple patterns in a BGP must be exactly matched when a query is executed. Generally speaking, there are eight modes in triple patterns: (S, P, O), (?S, P, O), (S, ?P, O), (S, P, ?O), (?S, ?P, O), (?S, P, ?O), (S, ?P, ?O), (?S, ?P, ?O). The triple pattern with temporal information contains a temporal variable and the corresponding temporal constraint is represented by time binary relationships in FILTER keyword. Figure 1 depicts an example of T-SPARQL.

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX ub: <http://www.lehigh.edu/zhp2/2004/0401/bench.owl#>
SELECT ?X, ?Y, ?Z
WHERE{
     ?X rdf:type ub:Student .
     ?Y rdf:type ub:Department .
     ?X ub:memberOf ?Y | ?ts .                    ──────► Triple Pattern
     ?Y ub:subOrganizationOf <http://www.University0.edu> .
     ?X ub:takesCourse ?Z | ?tt .
     FILTER ( Valid(?ts) overlaps Valid(?tt)
                  && Valid(?ts) during [2008-01-01, 2009-01-01] ) .
}
```

Figure 1. An example of T-SPARQL.

4. RDF Storage Model

4.1. Characteristics of HBase

HBase organizes data in tables with a name of HTable, which supports row-level transactions. A row is uniquely identified by a *rowkey*. The data in the table is sorted in ascending order by the rowkey. It means that rows with the same prefix are stored in adjacent positions. Intra-row data are grouped by column families that need to be defined in advance. Although each row has the same column families (called column qualifier), the columns in a column family can be different. The cell value in every column can have several versions sorted by the corresponding timestamps in a descending order. Generally, the timestamp is attached by the system when a data is inserted into HBase. Users can indicate the time-to-live (TTL) property to denote how long the data can exist in the database system. In physical storage, data are stored in key-value pairs with the following format.

$$[\textit{Rowkey}, \textit{Column Family}, \textit{Column Qualifier}, \textit{Timestamp}] \rightarrow \textit{Cell value}$$

This storage structure provides physical support for flexible data modeling and can store sparse data without wasting space.

Note that, although the multi-version mechanism provided by HBase can meet some basic requirements of temporal data modeling, there are still many defects in the maintenance of temporal data.

- Fixed time granularity

As we know, time has different calculation units (*e.g.*, year, month, day, hour, minute, second, and millisecond). In real-world applications, the time granularity of data is set according to specific scenario. For example, weather can be recorded by days or hours; stock price can be updated in seconds; running time of a program can be accurate to milliseconds. Obviously, different application scenarios need to use different time granularity. The time mechanism in HBase, however, only uses second as a fixed time granularity. Users cannot choose a proper time granularity according to the given scenario. Although the time granularity in seconds may be applicable in many application scenarios, other time representations in applications are not supported by HBase.

- Single time dimension and immutable TTL

In many temporal data models, temporal information is generally multi-dimensional. The temporal RDF data model in this paper contains two time-dimensions. The built-in time mechanism supported by HBase is one-dimensional and cannot represent multi-dimensional time. In addition, users can use TTL to control data lifetime. But when the TTL is set, it is applied to entire column family. In real applications, it is needed to control the life cycle of data at different levels, without being limited to a column family.

- Implicit time interval expression

Generally, we encode time points as an interval. A time interval has start time and end time.

The attached timestamp in HBase can represent the start time explicitly. But the end time is determined by TTL or the timestamp of the next version implicitly. This can cause a wrong or misleading result during the temporal query processing.

First, suppose that an TTL is set. Then a cell value may have two time-intervals and this can result in a confusion for users. Let us look at an example of an e-book table shown in Table 1. This table contains two column families: CF1 and CF2. Here CF1 has a column named Supplier and there is no TTL that is explicitly set (the default value is ∞). CF2 has a column named Price and the TTL value is 10. It is also shown there is a kind of book b1 in the table, in which the price of b1 at time1 is 34.6. According to the TTL value, the time interval of price 34.6 is [1, 11]. Later, the price of b1 is changed to 48.5 at time 3. Then the price 34.6 has another time interval [1, 2]. At this point, which one should be chosen will be decided by users. However, there is an ambiguity if two users chose different time intervals and compare their data.

Second, users can update the data via Put and Delete commands with a specified timestamp arbitrarily. It means that the implicit time intervals of data have high uncertainty. Let us look at Table 1 again. One user acquires the suppliers of b1 at time 6 and gets s1[1,2], s2[3,4] and s3[5, ∞]. Suppose that the version s2 is deleted at time 7. Then the time interval of s1 is changed from [1, 2] to [1, 4] and this causes inconsistent query results. Hence, users are required to check all data operations before using time information. This is time consuming and laborious.

Table 1. An example of e-book.

Book	CF1:Supplier TTL = ∞	CF2:Price TTL = 10
b1	1: s1	1: 34.6
	3: s2	3: 49.0
	5: s3	5: 43.5

4.2. Storage Model

RDF Schema defines the vocabularies that are used by RDF to describe data, including classes, properties, inheritance relationships between classes (rdfs:subClassOf), inheritance relationships between properties (rdfs:subPropertyOf), domain and range of a property (rdfs:domain, rdfs:range). Note that these classes are the subclass of rdfs:Class. RDF data are the instances of RDFS, which actually record application information. We design a storage model to organize temporal RDF Schema and RDF instances in different tables. We first create two tables, TClass and TProperty, to store the temporal RDF Schema. We present the definitions of these two tables as follows:

Definition 2 (TClass table). *TClass* = (*Ct*, *PC*: pc_1t, ..., *PC*: pc_mt, *SC*: sc_1t, ..., *SC*: sc_nt, *Ins*: i_1t, ..., i_kt), in which

1. $t = [V][T]$.
2. *Ct* is temporal class.
3. $PC = \{pc_1t, ..., pc_mt\}$ is a set of temporal parent classes of *C*. Function *ParClass*: *Ct* → *PC* can be used to get *PC* based on the known *Ct*.
4. $SC = \{sc_1t, ..., sc_nt\}$ is a set of temporal subclasses of *Ct*. Function *SubClass*: *Ct* → *SC* can be used to get subclasses, and $SubClass(pc_it) \supset SubClass(Ct)$.
5. $Ins = \{i_1t, ..., i_kt\}$ is the set of temporal instances of *Ct*.

TClass is a table that records the temporal classes in the temporal RDF Schema and the related instances in the temporal RDF dataset. The structure of TClass is shown in Figure 2. In Figure 2, row key of this table is (Class, t). Each row contains three column families: PC, SC and Ins. PC stores the direct parent classes of the class in row key, where one class occupies one column, SC stores all subclasses, and Ins records all instances of the class in the temporal RDF data. The cell values are set to "1" for the columns. Note that the timestamp (valid time, transaction time) in all column qualifiers must be contained in the time interval of the row key. It means that all records related to a class exist on the premise of the existence of that class.

RowKey: Class, t
Column Family: PC
Column parentClass1,t : 1
Column parentClass2,t : 1
Column Family: SC
Column subClass1,t : 1
Column subClass2,t : 1
Column Family: Ins
Column ins1,t : 1
Column ins2,t : 1

Figure 2. The structure of TClass.

Definition 3 (TProperty table). *TProperty* = (*Pt*, *Domain:* d_1t, ..., *Domain*: d_kt, *Range*: r_1t, ..., *Range*: r_lt, *PP*: pp_1t, ..., *PP*: pp_mt, *SP*: sp_1t, ..., *SP*: sp_nt), in which

1. $t = [V][T]$.
2. *Pt* is temporal property.
3. *Domain* = $\{d_1 1, ..., d_kt\}$ is the domain of *Pt*, that is composed of d_1t, ..., d_kt.
4. *Range* = $\{r_1t, ..., r_lt\}$ is the range of *Pt*, that is composed of r_1t, ..., r_lt.
5. *PP* = $\{pp_1t, ..., pp_mt\}$ is a set of temporal parent properties of *Pt*. Function *ParProp*: $Pt \rightarrow PP$ can be used to obtain *PP*.
6. *SP* = $\{sp_1t, ..., sp_nt\}$ is a set of temporal sub-properties of *Pt*. *SubProp*: $Pt \rightarrow SP$ is the function mapping *Pt* to *SP*, and $SubProp(pp_it) \supset SubProp(Pt)$.

TProperty stores properties information with inheritance relationships in temporal RDF Schema. The structure of this table is shown in Figure 3, which contains four column families named *Domain*, *Range*, *PP* and *SP*. The row key of the table is (*Property*, *t*), and its domain and range are recorded in *Domain* and *Range*, respectively. *PP* is the column family that stores the direct parent property and column family *SP* stores all the child properties with temporal information. Being the same as TClass, all information is stored as column qualifiers with values that are set to be "1", and the time interval is included in the lifetime and validity period of the property to ensure the data accuracy.

RowKey: Property, t
Column Family: Domain
Column domain1,t : 1
Column domain2,t : 1
Column Family: Range
Column range1,t : 1
Column range2,t : 1
Column Family: PP
Column parentPropety1,t : 1
Column parentPropety2,t : 1
Column Family: SP
Column subPropety1,t : 1
Column subPropety2,t : 1

Figure 3. The structure of TProperty.

The structures of TClass and TProperty contain only a few column families and are easy to be understood. Even though each row has different columns, there is not any waste of space in the key-value storage mode. In addition, the data in these tables are stored in groups by column families. This can reduce the IO cost of querying RDFS. Moreover, users can obtain some hierarchy or inference information and this can avoid multiple queries. The time intervals of RDFS are recorded in row keys and columns. This can solve the above-mentioned problems of built-in time in HBase and guarante time constraints between RDFS hierarchies.

In addition to TClass and TProperty to store temporal RDF Schema, we further create three tables SP_OT, OS_PT and PO_ST to store temporal RDF triples. Being different from the tables for temporal RDFS, none of the row keys of the three tables for temporal RDF triples contain any time information. We present the definitions of these three tables as follows.

Definition 4 (SP_OT table). *SP_OT* = (*SP*, *OT*: o_1t, ..., *OT*: o_nt), in which

1. $t = [V][T]$
2. *SP* is the subject and predicate of a temporal triple.
3. *OT* = $\{o_1t, ..., o_nt\}$ is the set composed by corresponding objects of the triples.

Definition 5 (OS_PT table). *OS_PT* = (*OS*, *PT*: p_1t, ..., *PT*: p_nt), in which

1. $t = [V][T]$
2. OS is the object and subject of a temporal triple.
3. $PT = \{p_1t, ..., p_nt\}$ is the set composed by corresponding predicates of the triples.

Definition 6 (PO_ST table). $PO\_ST = (PO, ST: s_1t, ..., ST: s_nt)$, in which

1. $t = [V][T]$.
2. PO is the predicate and object of a temporal triple.
3. $ST = \{s_1t, ..., s_nt\}$ is the set composed by corresponding subjects of the triples.

The structure of SP_OT, OS_PT and PO_ST are shown in Figure 4. They contain only one column family. Row keys of these tables are composed of two elements in a triple and the other element acts as a column in the table with the temporal information of triple. In temporal RDF queries, the variable constraints between triples are still the body of query blocks and the temporal constraints are the secondary data filtering. Therefore, the time stored in columns can be processed by the filters of HBase. Furthermore, triple redundancy in the tables can effectively deal with different triple patterns, which will be explained in the next section. Note that PO_ST and OS_PT do not maintain the triples with a predicate of rdf:type because the class instances can be obtained directly from the table TClass.

RowKey: Subject, Predicate
Column Family: OT
Column object1,t : 1
Column object2,t : 1
(a) SP_OT table

RowKey: Object, Subject
Column Family: PT
Column predicate1,t : 1
Column predicate2,t : 1
(b) OS_PT table

RowKey: Predicate, Object
Column Family: ST
Column subject1,t : 1
Column subject2,t : 1
(c) PO_ST table

Figure 4. The structures of SP_OT, OS_PT, PO_ST.

Now we apply the examples to illustrate our storage method with the above table structures. The temporal RDF Schema in Figure 5 contains 4 classes and 2 properties. Among them, *Student*[3, *now*][3, *UC*] is a subclass of *Person*[1, *now*][1, *UC*] and *GraduateStudent*[4, *now*][4, *UC*] is a subclass of class *Student*. The domain of property *degreeFrom*[2, *now*][2, *UC*] is *Person* and its range is *University*. *masterDegreeFrom*[4, *now*][4, *UC*] is a sub-property of *degreeFrom*.

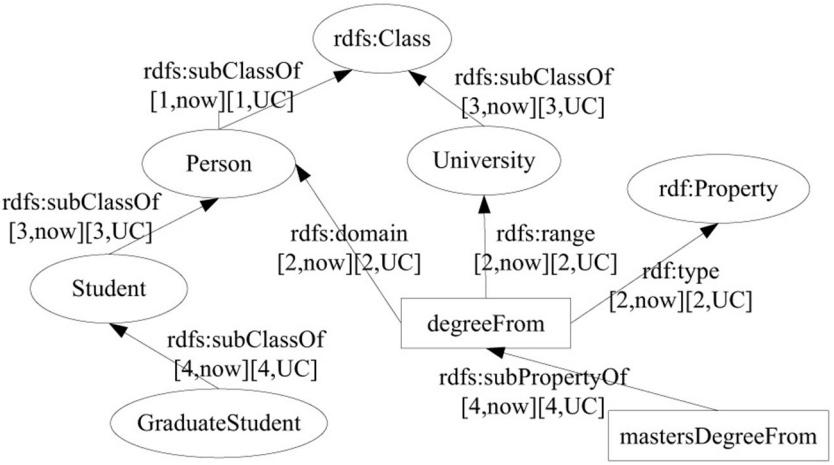

Figure 5. An example of temporal RDFS.

Then, the TClass and TProperty tables are shown in Table 2 and Table 3, respectively.

Figure 6 is an example of temporal RDF graph, which depicts the information of *student*1[5, 12][5, UC] at university3. The student takes course1 during [6,7] and participates in two departments during [5, 8]. The graph contains 8 temporal triples. Among them, 4 temporal triples describe the relationships between instances and classes and the other 4 temporal triples describe the relationships between instances. The former is stored in the SP_OT table, the latter are stored in the tables SP_OT, OS_PT and PO_ST. These four tables are shown in Table 4, Table 5 and Table 6, respectively.

Table 2. An example of TClass table.

RowKey	Column Family "PC:"	Column Family "SC:"	Column Family "Ins:"
Person, [1, now][1, UC]		Student, [3, now][3, UC]: "1"	student1, student2, teacher1,teacher2
		GraduateStudent, [4, now][4, UC]: "1"	graduateStudent1
Student, [3, now][3, UC]	Person, [3, now][3, UC]: "1"	GraduateStudent, [4, now][4, UC]: "1"	student1, student2
GraduateStudent, [4, now][4, UC]	Student, [4, now][4, UC]: "1"		graduateStudent1
University, [3, now][3, UC]			university3

Table 3. An example of TProperty table.

RowKey	Column Family "Domain:"	Column Family "Range:"	Column Family "PP:"	Column Family "SP:"
degreeFrom, [2, now][2, UC]	Person, [2, now][2, UC]: "1"	University, [2, now][2, UC]: "1"		masterDegreeFrom, [4, now][4, UC]: "1"
masterDegreeFrom, [4, now][4, UC]	Person, [4, now][4, UC]: "1"	University, [4, now][4, UC]: "1"	University, [4, now][4, UC]: "1"	

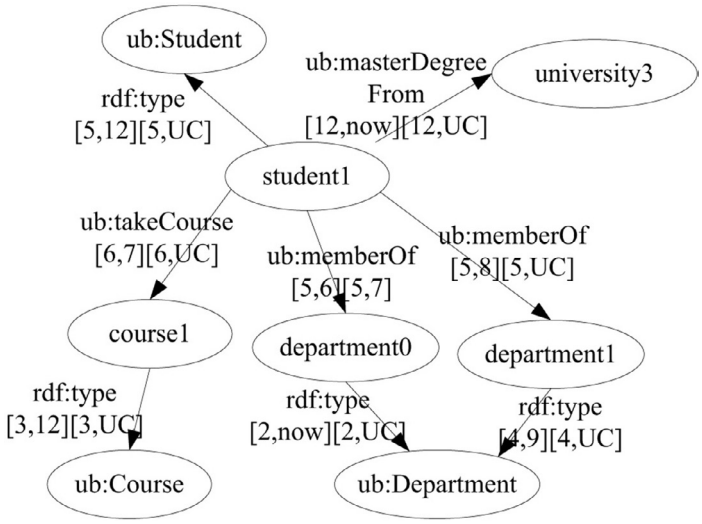

Figure 6. An example of temporal RDF.

Table 4. An example of SP_OT table.

RowKey	Column Family "OT:"
student1, rdf: type	ub: Student, [5, 12][5, UC]: "1"
student1, ub: takeCourse	course1, [6, 7][6, UC]: "1"
student1, ub: memberOf	deparment0, [5, 6][5, 7]: "1"
	deparment1, [5, 8][5, UC]: "1"
student1, ub: masterDegreeFrom	university3, [12, now][12, UC]: "1"
course1, rdf: type	ub: Course, [3,12][3,UC]: "1"
department0, rdf: type	ub: Department, [2, now][2, UC]: "1"
department1, rdf: type	ub: Department, [4, 9][4, UC]: "1"

Table 5. An example of OS_PT table.

RowKey	Column Family "PT:"
course1, student1	ub: takeCourse, [6, 7][6, UC]: "1"
deparment0, student1	ub: memberOf, [5, 6][5, 7]: "1"
deparment1, student1	ub: memberOf, [5, 8][7, UC]: "1"
university3, student1	ub: masterDegreeFrom, [12,now][12, UC]: "1"

Table 6. An example of PO_ST table.

RowKey	Column Family "ST:"
ub: takeCourse, course1	student1, [6, 7][6, UC]: "1"
ub: memberOf, deparment0	student1, [5, 6][5, 7]: "1"
ub: memberOf, deparment1	student1, [5, 8][7, UC]: "1"
ub: masterDegreeFrom, university3	student1, [12, now][12, UC]: "1"

4.3. Query Strategy

In Subsection 4.2 it is shown that, for temporal RDF storage, the storage model introduced in this paper can be divided into two parts. TClass and TProperty are used to record RDF Schema, which covers the domain information involved, including inheritance information and relevant class instances. SP_OT, OS_PT and PO_ST are designed for RDF data, which can satisfy all triple patterns in query block, and simple queries can be hereby responded quickly.

A comparison of diverse temporal triple patterns is summarized in Table 7. It is shown that, if any two elements in the triple are known, the triple pattern can generate a row key based on the known binding values and then select an appropriate table from SP_OT, OS_PT and PO_ST, to perform the Get operation provided by HBase.

Table 7. Diverse triple patterns.

Triple Pattern	Tables
(S, P, ?O) : t	SP_OT
(S,?P, ?O) : t	SP_OT
(S, ?P, O) : t	OS_PT
(?S, P, O) : t	PO_ST
(?S, ?P, O) : t	OS_PT
(?S, P, ?O) : t	PO_ST
(?S, ?P, ?O) : t	SP_OT | OS_PT
(?S, rdf:type, C) : t	TClass

Note that the triple pattern with rdf:type as a known predicate is an exception, which should be queried from the TClass table. The case of only one known element is handled by the Scan operation. The table data in HBase are arranged in alphabetical order and the rows with the same prefix in row key are adjacent. Therefore, matching a triple pattern can be completed quickly by using the known data to set the start and end row keys of Scan.

There are some RDF queries that always contain the triple pattern with a wide hierarchy. Such queries cannot be processed by the Get or Scan operation. Let us look at an example in Figure 7. This query example contains two non-temporal triple patterns to get the Student members of Department0. The first triple can acquire the student instances and the second triple can get the subjects satisfying this triple. However, if the triples are executed directly, the returned result will not match the query seman-

tics because *ub:Student* and *ub:memberOf* have implicit subclasses and sub-properties. Therefore, a preprocessing is required before executing such queries. We need to search the TClass or TProperty table to get the corresponging instances or sub-properties, and then use these sub-properties to extend the triple patterns.

It is required that all triple patterns must meet the temporal constraints given in FILTER keyword. Temporal constraints mean some time interval relationships with one or two time variables. In order to reduce the amount of data transmission and the number of queries, a FILTER parsing is divided into two parts. One part is composed of time interval relationships with a single variable, which is added to the relevant triple patterns. Another part is handled in the triple pattern joins. In this paper, we adopt the greedy multiple join strategy proposed in [13] and use MapReduce to perform join operations. The detailed steps for a query are as follows:

1. Decomposing the query block into non-temporal triple patterns and temporal constraints;
2. Extracting RDFS domain objects from non-temporal triple patterns;
3. Retrieving subclasses or sub-properties from the TClass or TProperty table for domain instances and predicates, and then extending the relevant triple patterns;
4. Adding temporal constraints with a single variable to the triple patterns;
5. Determining the join strategy with the greedy algorithm and selecting a join method for the temporal constraints with two variables;
6. Executing temporal triple patterns with the filters provided by HBase and making joins with the MapReduce programs.

5. Experiments

Our experimental sets include one master server and two slave nodes each. All machines have the same configuration: Intel Core i5-2450M 2.5GHZ and Ubuntu 64-bit. The master server has 4GB main memory and each slave node has 2GB main memory. We use RDF data provided by LUBM, including the univ-bench, Uba generator and a set of test queries. The univ-bench is the benchmark which defines the university domain ontology. The Uba is an OWL text generator which requires some specific parameters to generate file sets of different size. All instance files can be parsed into triples by Jena. Note that the RDF data from LUBM do not contain temporal information. For the purpose of our experiments, we add temporal information by a random time generator. Then we have two temporal RDF datasets D1 and D2 shown in Table 8, which are imported into HBase by BulkLoad.

Table 8. Temporal RDF datasets.

DataSets	Universities	Temporal RDF Triples
D1	5	874899
D2	15	13739264

In order to verify the validity of our storage model, we refer to the LUBM test queries and create the following 7 test queries with temporal constraints, which are executed on D1 and D2.

Query 1:

```
SELECT ?X
WHERE {
    ?X rdf:type ub:GraduateStudent .
    ?X ub:takesCourse
    http://www.Department0.University0.edu/GraduateCourse0 | ?t.
    FILTER (Tansaction (?t) before
    [2014-06-30, 2014-06-30])}
```

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX ub: <http://www.lehigh.edu/zhp2/2004/0401/bench.owl#>
SELECT ?X
WHERE{
    ?X rdf:type ub:Student.
    ?X ub:memberOf <http://www.Department0.University0.edu> .
}
```

Figure 7. An example of wide hierarchy query.

Query 2:

```
SELECT ?X, ?Y, ?Z
WHERE{
    ?X rdf:type ub:GraduateStudent .
    ?Y rdf:type ub:University .
    ?Z rdf:type ub:Department .
    ?X ub:memberOf ?Z | t1.
    ?Z ub:subOrganizationOf ?Y | ?t2.
    ?X ub:undergraduateDegreeFrom ?Y | ?t3.
FILTER (Valid(?t) overlaps Valid(?t2) && Valid(?t3) during [2016-04-30, now])}
```

Query 3:

```
SELECT ?X
WHERE {
    ?X rdf:type ub:Publication .
    ?X ub:publicationAuthor http://www.Department0.University0.edu/AssistantProfessor0 | ?t.
    FILTER (Tansaction(?t) during [2013-01-31, 2015-01-31])}
```

Query 4:

```
SELECT ?X, ?Y1, ?Y2, ?Y3
WHERE {
    ?X rdf:type ub:Professor .
    ?X ub:worksFor http://www.Department0.University0.edu | ?t1.
    ?X ub:name ?Y1 .
    ?X ub:emailAddress ?Y2 | ?t2 .
    ?X ub:telephone ?Y3
    FILTER (Valid(?t) during Valid(?t2) && Tansaction(?t1) overlaps [2013-01-31, 2015-01-31])}
```

Query 5:

```
SELECT ?X
WHERE {
    ?X rdf:type ub:Person .
    ?X ub:memberOf http://www.Department0.University0.edu |?t.
    FILTER (Valid(?t) starts [2017-05-20, 2017-05-20])}
```

Query 6:

```
SELECT ?X
WHERE {
    ?X rdf:type ub:Student | ?t
    FILTER (Tansaction(?t) equals [2016-09-08, 2018-04-30])}
```

Query 7:

```
SELECT ?X, ?Y
WHERE {
    ?X rdf:type ub:Student .
    ?Y rdf:type ub:Course .
    ?X ub:takesCourse ?Y | ?t1.
    http://www.Department0.University0.edu/AssociateProfessor0 ub:teacherOf, ?Y | ?t2
FILTER (Valid(?t1) overlaps Valid(?t2) && Tansaction(?t1) ends [2018-03-10, 2018-07-25])}
```

Table 9 presents the number of different operations included in these queries. Note that Get and Scan are not time-consuming operations. A join operation means the running of a MapReduce program, which greatly affects the response time of queries. Figure 8 presents response times of the seven queries over D1 and D2. Furthermore, the increment ratios of response time for the seven queries are calculated by (D2-D1)/D1 and shown in Figure 9.

Table 9. Operations in temporal queries.

Query	Query1	Query2	Query3	Query4	Query5	Query6	Query7
Get	2	4	2	3	3	1	2
Scan	0	3	0	3	0	0	1
Joins	1	**3**	1	1	1	**0**	2

First, it is shown in Figure 8 that, for a given query, its response time over D1 is less than that over D2 because D2 is larger than D1. So, generally speaking, a given query will take more time as the size of the dataset increases. But we can observe from Figure 9 that the query time increases only by a factor of 3 while the amount of RDF data is increased by 10 times. This demonstrates the advantages of the storage model based on HBase.

Second, it is shown in Figure 8 that different queries may have very different response times. In particular, Query 2 has the longest response times over both D1 and D2 and Query 6 has the shortest response times over both D1 and D2, compared to other queries over D1 and D2. The main reason why the response times of Query 2 are significantly higher than other queries is that Query 2 contains three join operations. The main reason why the response times of Query 6 are significantly less than other queries is that Query 6 contains only one triple pattern, not involving the MapReduce calculations. Then this

query can be executed directly via Java APIs provided by HBase. The situation occurring in Query 2 or Query 6 will further be aggravated while the query is issued over very differently sized datasets. This is why, for Query 2 and Query 6, their response times over D1 are significantly less than their response times over D2.

Now let us look at other queries. It is shown in Figure 8 that, for Query 1, Query 3 and Query 5, their response times over D1 and D2 are relatively stable because of their high selectivity. Among these three queries, Query 1 does not contain a hierarchical reasoning of classes, but its implicit hierarchical information can be directly obtained from TClass and TProperty. So, Query 1 does not cause an excessive time consumption.

Query 4 is similar to Query 1 in terms of the query structure, but it involves a hierarchical reasoning and the triple with sub-properties of ub:worksFor must be extended. At this point, for Query 4 over D2, more data are involved in join operations and its response time (72.612 sec.) is a little longer than the response time of Q1 over D2 (62.823 sec.) (*i.e.*, 15.58% longer). As for Query 7, it is actually an extension of Query 6 and has a higher data selectivity. However, Query 7 needs more MapReduce calculations for join operations. So, the response time of Query 7 is much longer than the response time of Query 6.

In addition, temporal constraints with a single variable are executed in the Get or Scan operation by the filters in advance. This can reduce the IO cost. As for the multivariable temporal constraints, they are handled during the join processing and there are no redundant join operations. This is one of the reasons why most response times of the queries do not increase when data volumes increase.

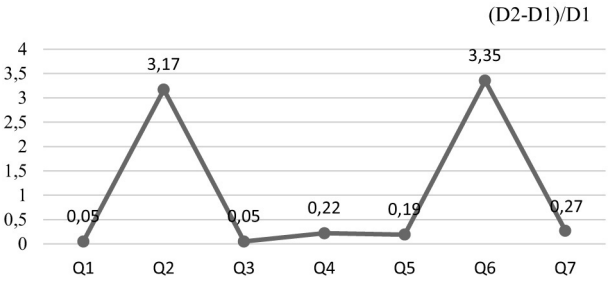

Figure 9. Response increment ratio.

6. Conclusion

To store temporal RDF data, in this paper we investigate the structural characteristics of HBase and present the problems in its built-in time mechanism. On this basis, we propose a HBase storage model with 5 tables for temporal RDF data storage, which can preserve data semantics of temporal RDF and solve temporal representation in the built-in time mechanism. Based on the storage model, we put forward the query

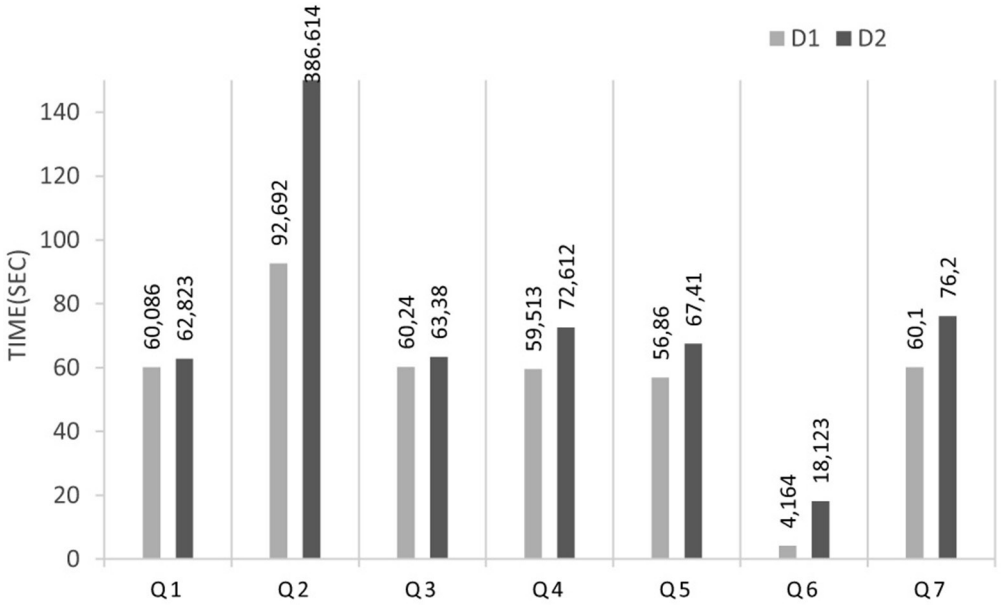

Figure 8. Response times.

strategy. We classify temporal constraints according to the number of variables in the binary relationship. Constraints with a single variable are executed by filters provided by HBase, and this can reduce the amount of data involved in join operations and reduce the IO cost. Other constraints are dealt with during the join operations and this does not increase the number of join operations. All triple patterns are covered by the SP_OT, OS_PT and PO_ST tables. With the proposed query approach, the queries with wide hierarchies can be executed effectively for TClass and TProperty tables. Furthermore, we used a cluster with three nodes and the LUBM test queries to verify the validity of our storage and query strategy on two datasets.

Note that there is no benchmark of temporal RDF. So, we do not test our approach with very large datasets, say TB datasets. In the near future, we will enlarge our datasets and increase the cluster nodes to optimize our storage model. In addition, the response time of queries based on the MapReduce framework is not quick enough. In the next phase, we will improve the processing algorithm of join operation and adjust the configuration parameters of MapReduce to obtain better query performance.

Acknowledgment

The work was supported by the Basic Research Program of Jiangsu Province (BK20191274) and in part by the National Natural Science Foundation of China (61772269 and 61370075).

References

[1] C. Gutierrez *et al.*, "Introducing Time into RDF", *IEEE Transactions on Knowledge & Data Engineering*, vol. 19, pp. 207–218, 2007. http://dx.doi.org/10.1109/TKDE.2007.34

[2] M. Cai and M. Frank, "RDFPeers: A Scalable Distributed RDF Repository Based on a Structured Peer-to-Peer Network", in *Proc. of the International Conference on World Wide Web*, 2004, pp. 650–657. http://dx.doi.org/10.1145/988672.988760

[3] S. Harris *et al.*, "4store: The Design and Implementation of a Clustered RDF Store", in *Proc. of the International Conference on Scalable Semantic Web Knowledge Base Systems*, 2009, pp. 94–109.

[4] SYSTAP. Bigdata RDF database. http://www.Systap.com

[5] A. Harth *et al.*, "YARS2: A Federated Repository for Querying Graph Structured Data from the Web", in *Proc. of the International Semantic Web Conference*, 2007, pp. 211–224. http://dx.doi.org/10.1007/978-3-540-76298-0_16

[6] V. Gudivada *et al.*, "NoSQL Systems for Big Data Management", in *Proc. of the IEEE World Congress on Services*, 2014, pp. 190–197. http://dx.doi.org/10.1109/SERVICES.2014.42

[7] J. Pokorný, "New Database Architectures: Steps Towards Big Data Processing", in *Proc. of the IADIS European Conference on Data Mining*, 2013, pp. 3–10.

[8] A. Ribeiro *et al.*, "Data Modeling and Data Analytics: A Survey from a Big Data Perspective", *Journal of Software Engineering & Applications*, 2015, vol. 8, no. 12, pp. 617–634. http://dx.doi.org/10.4236/jsea.2015.812058

[9] H. Hu *et al.*, "Toward Scalable Systems for Big Data Analytics: A Technology Tutorial", *IEEE Access*, vol. 2, no. 1, pp. 652–687, 2017. http://dx.doi.org/10.1109/ACCESS.2014.2332453

[10] Apache HBase. https://hbase.apache.org/

[11] F. Chang *et al.*, "Bigtable: A Distributed Storage System for Structured Data", *ACM Transactions on Computer Systems*, vol. 26, no. 2, pp. 1–26, 2008. http://dx.doi.org/10.1145/1365815.1365816

[12] H. Choi *et al.*, "SPIDER: A System for Scalable, Parallel/Distributed Evaluation of Large-Scale RDF Data", in *Proc. of the ACM Conference on Information and Knowledge Management*, pp. 2087–2088. http://dx.doi.org/10.1145/1645953.1646315

[13] D. J. Kim *et al.*, "Scalable RDF Store Based on HBase and MapReduce", in *Proc. of the International Conference on Advanced Computer Theory and Engineering*, 2010, pp. 633–636.

[14] D. J. Kim, "Research and Design of RDF Storage System based on HBase", Hangzhou: Zhejiang University, 2011 (In Chinese).

[15] Jena. https://jena.apache.org/

[16] V. Khadilkar *et al.*, "Jena-HBase: A Distributed, Scalable and Efficient RDF Triple Store", in *Proc. of the ISWC 2012 Posters & Demonstrations Track*, 2012.

[17] N. Papailiou *et al.*, "H2RDF: Adaptive Query Processing on RDF Data in the Cloud", in *Proc. of the 2012 International Conference on World Wide Web*, 2012, pp. 397–400. http://dx.doi.org/10.1145/2187980.2188058

[18] A. Chebotko *et al.*, "Storing, Indexing and Querying Large Provenance Data Sets as RDF Graphs in Apache HBase", in *Proc. of the IEEE Ninth World Congress on Services*, 2013, pp. 1–8. http://dx.doi.org/10.1109/SERVICES.2013.32

[19] K. Li *et al.*, "A Distributed RDF Storage and Query Model Based on HBase", in *Proc. of the 2015 International Conference on Web-Age Information Management*, 2015, pp. 3–15. http://dx.doi.org/10.1007/978-3-319-23531-8_1

[20] J. H. Um *et al.*, "Distributed RDF Store for Efficient Searching Billions of Triples Based on Hadoop", *Journal of Supercomputing*, vol. 72 , no. 5, pp. 1825–1840, 2016. http://dx.doi.org/10.1007/s11227-016-1670-6

[21] X. Luo and B. Wu, "Predicate-Oriented Query of RDF Data Based on a Distributed Storage Model", in *Proc. of the 2017 IEEE International Conference on Data Science in Cyberspace*, 2017, pp. 37–43. http://dx.doi.org/10.1109/DSC.2016.43

[22] Y. Hu and S. Dessloch, "Temporal Data Management and Processing with Column Oriented NoSQL Databases", *Journal of Database Management*, vol. 26, no. 3, pp. 41–70, 2015. http://dx.doi.org/10.4018/JDM.2015070103

[23] F. Grandi, "T-SPARQL: A TSQL2-Like Temporal Query Language for RDF", in *Proc. of the 2010 East-European Conference on Advances in Databases and Information Systems*, 2010, pp. 21–30.

[24] A. Dignös *et al.*, "Temporal Alignment", in *Proc. of the 2012 ACM SIGMOD International Conference on Management of Data*, 2012, pp. 433–444. http://dx.doi.org/10.1145/2213836.2213886

[25] R. Z. Ma *et al.*, "SPARQL Queries on RDF with Fuzzy Constraints and Preferences", *Journal of Intelligent and Fuzzy Systems*, vol. 30, no. 1, pp. 183–195, 2016. http://dx.doi.org/10.3233/IFS-151745

[26] Z. M. Ma *et al.*, "Storing Massive Resource Description Framework (RDF) Data: A Survey", *Knowledge Engineering Review*, vol. 31, no. 4, pp. 391–413, 2016. http://dx.doi.org/10.1017/S0269888916000217

[27] D. Yang and L. Yan, "Transforming XML to RDF(S) with Temporal Information", *Journal of Computing and Information Technology*, vol. 26 no. 2, pp. 115–129, 2018. http://dx.doi.org/10.20532/cit.2018.1004005

[28] R. Z. Ma *et al.*, "Coronal Mass Ejection Data Clustering and Visualization of Decision Trees", *The Astrophysical Journal Supplement Series*, vol. 236, no. 1, p. 4, 2018. http://dx.doi.org/10.3847/1538-4365/aab76f

[29] R. Z. Ma *et al.*, "Solar Flare Prediction Using Multivariate Time Series DecisionTrees", in *Proc. of the 2017 IEEE International Conference on Big Data*, 2017, pp. 2569–2578. http://dx.doi.org/10.1109/BigData.2017.8258216

[30] K. Kulkarni and J. E. Michels, "Temporal Features in SQL: 2011", *SIGMOD Record*, vol. 41, no. 3, pp. 34–43, 2012. http://dx.doi.org/10.1145/2380776.2380786

[31] L. Yan *et al.*, "Indexing Temporal RDF Graph", *Computing*, vol. 101, no. 10, pp. 1457–1488, 2019. http://dx.doi.org/10.1007/s00607-019-00703-w

[32] J. Tappolet and A. Bernstein, "Applied Temporal RDF: Efficient Temporal Querying of RDF Data with SPARQL", in *Proc. of the 6th European Semantic Web Conference*, 2009, pp. 308–322. http://dx.doi.org/10.1007/978-3-642-02121-3_25

[33] A. Pugliese *et al.*, "Scaling RDF with Time", in *Proc. of the 2008 International Conference on World Wide Web*, 2008, pp. 605–614. http://dx.doi.org/10.1145/1367497.1367579

Received: May 2019
Revised: December 2019
Accepted: December 2019

Contact addresses:
Li Yan*
Nanjing University of Aeronautics and Astronautics
Nanjing
China
e-mail: yanli@nuaa.edu.cn
*Corresponding author

Zheqing Zhang
Nanjing University of Aeronautics and Astronautics
Nanjing
China
e-mail: zheqingzhang@163.com

Dan Yang
Nanjing University of Aeronautics and Astronautics
Nanjing
China
e-mail: nuaacst@163.com

Li Yan is a full professor in the College of Computer Science and Technology at the Nanjing University of Aeronautics and Astronautics, China. Her current research interests include Big Data knowledge engineering, temporal data management, and computational intelligence.

Zheqing Zhang received his master degree from the Department of Computer Science at the Guangdong University of Technology, China. He is now a PhD candidate in the College of Computer Science and Technology at the Nanjing University of Aeronautics and Astronautics, China. His research interests include knowledge graph and RDF data management.

Dan Yang received her master degree from the College of Computer Science and Technology at the Nanjing University of Aeronautics and Astronautics, China. Her research interests include RDF data management and knowledge graph.