Construction of RDF Knowledge Graph with MongoDB

Li Yan
College of Computer Science and Technology
Nanjing University of Aeronautics and Astronautics
Nanjing, China
yanli@nuaa.edu.cn

Hui Hu
College of Computer Science and Technology
Nanjing University of Aeronautics and Astronautics
Nanjing, China

Zongmin Ma
College of Computer Science and Technology
Nanjing University of Aeronautics and Astronautics
Nanjing, China
zongminma@nuaa.edu.cn

***Abstract*—Resource Description Framework (RDF) is widely used in semantic extraction, unified organization, and intelligent processing of large amounts of data because of its machine intelligibility. For example, knowledge graph based on RDF is commonly used in intelligent search, recommendation system, and smart medical treatment. And RDF is used to express the relationship between entities and process the semantics of data. Many efforts have been made to convert various data (such as relational database, XML, and JSON) into RDF. Yet, the effective generation of usable RDF data is still an urgent problem to be solved. With the wide use of NoSQL database, massive data is stored in NoSQL database, but the research on generating RDF from NoSQL database is not emphasized. We put forward a formal definition of MongoDB, and according to this definition, we propose a method of automatically extracting data from MongoDB and building corresponding RDF. Based on this method, we have also implemented a prototype system named M2R to validate method performance. The experimental results show that our approach is feasible and efficient.**

Keywords*—*RDF, automatic construction, OWL, MongoDB

I. Introduction

The Semantic Web is an intelligent network, which enables machines to understand the concept of data and the logical relationship between different data by adding metadata. The purpose of the Semantic Web is to provide a general semantic framework so that data can be shared and reused without being limited by applications, businesses, and communities [3]. The core of the Semantic Web technology stack is Resource Description Framework (RDF). RDF is a model framework proposed by the World Wide Web Consortium (W3C), which is used to describe the content of the Semantic Web in a standardized way. RDF is widely used in semantic extraction, unified organization, and intelligent processing of large amounts of data because of its machine comprehensibility (such as adding semantics to resource description on the network) and the characteristics that RDF format data can be shared, exchanged, and integrated without losing semantics [18]. RDF is widely accepted and used, which leads to the enthusiasm of building RDF and the surge of RDF data [13]. However, it is still an urgent problem to generate effective and available RDF data.

Building RDF from existing massive data is in a good direction. As far as we know, much work has been made to build RDF from different types of data sources, such as relational databases, XML, JSON. The existing research work mainly focuses on generating RDF data onto relational databases (e.g., direct mapping and R2RML) and XML (e.g., [4]). With the widespread use of JSON, a lightweight data exchange format on the Web, a group of people began to study how to convert JSON to RDF (e.g., [14]). With the continuous development and application of network technology, the amount of available data is increasing, and the traditional databases cannot effectively deal with big data [7]. NoSQL (not only SQL) databases are developed by big data management, which are a supplement to the traditional databases. Although NoSQL databases are widely utilized in various areas of the Internet, research on building RDF from NoSQL databases is still insufficient. It is a good idea that extract knowledge to generate RDF from a NoSQL database containing massive data to provide a standard and unified information processing framework.

To better describe our method of constructing RDF with MongoDB, we put forward formal definitions of MongoDB database and RDF based on their data model. This approach can deal with the semantic information contained in the MongoDB database. Based on this method, we implemented a mapping tool named M2R, which can automatically build RDF based on the MongoDB database and is convenient for non-professional users.

The rest of this paper is organized as follows. Section 2 presents the related work. Section 3 provides some preliminaries of the formal definitions of RDF and MongoDB. Section 4 details mapping rules of converting MongoDB to RDF. In Section 5, based on the mapping rules, we present the algorithm to construct RDF from MongoDB. Section 6 summarizes the thesis and points out the prospects of future work.

II. Related Work

The first type of RDF construction method mainly generates RDF based on relational databases. Up to the present moment, there has been a lot of research work about building RDF based on relational databases because of the widespread applications of RDB in many areas. Here, we list several representative research jobs. In [16], formal definitions of relational databases are given. And then, based on the mapping relationship between the relational database and the RDF data model and formal definitions of RDB, the mapping rules for the relational database to the RDF data model are defined to generate RDF data. This method studies monotonicity, information preservation, query preservation, and semantic preservation and proves that this method must be information preservation and query preservation. Information preservation means the conversion process of RDB to RDF would not lose information and exist some way to reconstruct the RDF to the original RDB. Query preservation means that a query about a relational database can be converted into an equivalent RDF query above RDF dataset converted by the RDB with the same semantics. According to the mapping rules of relational tables to RDF and schema information storing in system database information_schema, RDF files are generated with MySQL in [5]. The World Wide Web Consortium (W3C) proposed two standards, direct mapping, and R2RML, to extract semantic information from relational databases and generate RDF. Most methods of building RDF based on RDB follow either direct mapping or R2RML standards.

The second type of RDF construction method generates RDF with XML. Founded on XQuery and SPARQL, XSPARQL is proposed in [4], which supports querying XML and RDF data using the same framework and converting these two kinds of data to each other. Using the declarative and semantic of RDF SPARQL and the expressive power of XML XQuery, a method proposed in [10] transforms XML data to RDF based on keyword or graph query. In [8], the authors propose an RDF template language based on simple XPath expressions and a conversion method from XML onto RDF based on the template language. By analysing the tree structure of XML, paper [12] divides XML elements into three sub-models and then proposes related mapping rules to map the XML to RDF. Paper [6] proposes a set of mappings to convert XML Schema into Shape Expressions (ShEx), which is an RDF validation language, and develops a prototype system to obtain ShEx from XML Schema.

JSON is a more lightweight data exchange format than XML in the web application, easy to parse and efficient to transfer. With the widespread use of JSON on the Web, some work about converting JSON to RDF has been carried out. In [14], the authors define a formal mapping language based on the JSON-Pointer syntax to transform JSON documents into RDF. With this mapping language, they implement a mapping tool from JSON to RDF called J2RM. For helping developers use the JSON output of SPARQL queries, the authors in [11] transform the SPARQL JSON output to JSON-LD, a lightweight syntax to serialize Linked Data in JSON and launched by the W3C JSON-LD Working Group.

Although a great deal of work has been done to transform data onto various formats into RDF, little research concentrates on NoSQL. Compared with traditional relational databases, NoSQL databases generally have no fixed schema, which causes difficulty in constructing RDF with NoSQL. Based on the correspondence with Key-Value, [1] and [17] define the mapping rules to transform MongoDB documents to RDF/OWL. However, neither of these methods can handle documents with complex structures, such as multi-layer nested documents and arrays. [9] use the Formal Concept Analysis (FCA) method to construct a concept lattice from MongoDB, defines the transformation rules from concept lattice to ontology, and builds the ontology based on MongoDB data. Unfortunately, this method requires users to have professional background knowledge of formal concept analysis to build concept lattices from MongoDB. As far as we know, there is no tool to support the construction of concept lattice with MongoDB.

In this paper, based on MongoDB and RDF, we propose formal definitions of MongoDB and RDF. Relying on the definitions, we propose a construction method of transforming MongoDB data to RDF, guaranteeing semantic preservation in the transformation process. Unlike [1], [17] and [9], our method supports to process data with complex structure. At the same time, we provide an automated conversion tool for non-professional users. Finally, we prove the effectiveness of our method.

III. Preliminaries

(1) RDF data model

RDF data model consists of a group of RDF statements, which are represented by the triple of subject, predicate, and object. The subject is the described resource, the predicate represents the relationship between the subject and the object, and the object is the attribute value of the predicate, which can be a resource or a literal. A resource is anything with a Universal Resource Identifier (URI), and the object is a literal or resource, depending on whether the corresponding predicate represents a relationship between resources or an attribute of a resource. RDF is a domain-independent universal description language, which does not define any domain semantics. To overcome the defect that RDF does not define domain semantics, people use the RDF Schema to define domain semantics. The RDF Schema defines a set of modeling primitives with fixed semantics, including rdfs:Class, rdfs:Literal, rdfs:subClassOf, rdfs:subPropertyOf, rdfs:domain, rdfs:range, and others. Based on RDF and RDF Schema, OWL adds modeling language to enhance expression ability [2].

Definition 1 (RDF data model with OWL vocabulary) An RDF data model is defined as a tuple: RWM=(RB,PA,RI,RL,TP) where:

(a) $RB=RC \cup RD \cup RP$, RB is a finite set of basic properties of RDF, RDF Schema, and OWL. RC is a finite set of class resources, RD is a finite set of data types, and RP is a finite set of attribute resources.

(b) $RP=RDP \cup ROP$, RDP is a finite set of datatype properties, and ROP is a finite set of object properties.

(c) $PAxiom=Dxiom \cup Rxiom$, PAxiom represents a finite set of all property axioms.

(d) $Dxiom_p=\{c|c$ is the domain of property p, $p \in RP$ and $c \in RC\}$, $Dxiom_p$ means that the domain of property p is class c.

(e) $Rxiom_p=Rxiom_{dp} \cup Rxiom_{op}$, Rxiom_dp={x|x is the range of DatatypeProperty dp, $dp \in RDP$, $x \in RD\}$, $Rxiom_{op}=\{y|y$ is the range of ObjectProperty op, $op \in ROP$, $y \in RC\}$, $Rxiom_p \subseteq Rxiom$, $Rxiom_{dp}$ means that the range of DatatypeProperty p is x. $Rxiom_{op}$ means that the range of ObjectProperty p is y.

(f) RI is a finite set of all RDF individual (URI or IRI).

(i) RL is a finite set of literal.

(j) TP is a finite set of triple (S, P, O).

(2) MongoDB data model

As a document-oriented database which is a type of NoSQL, MongoDB is an open-source database and provides many features such as high performance, high availability, and automatic extension. This database consists of the collection composed of documents, which are equivalent to tables in relational databases. Each document formed of a set of key-value pairs is similar to the record of the table. Regard BSON as document storage and data interchange format, MongoDB supports dynamic schema design and allows documents in a collection to have different fields and structures. Next, we present a formal representation of MongoDB data model.

Definition 2 (MongoDB Model) A MongoDB model is defined as a tuple: M=(C,K,T,DF,RF,D,V) where:

(a) C is a finite set of all collections. K represents a finite set of all fields. T represents all data types supported by MongoDB, T(k) indicates the type of field k.

(b) $K = K_B \cup K_E \cup K_D \cup K_M \cup K_A$. K_B is a finite set of basic type field, K_E is a finite set of embedding document field, K_D is a finite set of DBRef referencing field, K_M is a finite set of manual referencing field, and K_A is a finite set of array type field. K_M is user define fields, which need user to complete referencing collection and referencing field. $K_M(k) = (\text{referencing collection}, \text{referencing field})$ denotes referencing collections of manual referencing field k.

(c) DF is a finite set of field domain in the database, df(k) indicates the domain of field k.

(d) D is a finite set of all document in the databases, $c \in C, D(c) \subset D$ denote all documents in collection c. $\alpha(c) \subset K$ denote all fields in collection c. $d \in D(c), \alpha(d)$ denote all fields in document d in collection c.

(e) V is a finite set of fields value, $k \in K, V(k)$ denotes value of field k, $T(v)$ is same as $T(k)$.

IV. Mapping Rules from MongoDB to RDF Model

According to the formal definitions of the RDF data model and the MongoDB model given in the previous section, in this section, we provide the rules for mapping MongoDB to RDF with some OWL vocabulary from two aspects: schema mapping and instance mapping. Assume that φ denotes mapping process.

Rule 1: $\forall\, df \in DF \rightarrow \varphi(df) \in RC$.

All field domains in MongoDB are transformed into RDF Class. Field domain in DF is usually Collection name or embedding document field, but sometimes maybe array field the value of which includes many documents usually used to represent the one-to-many or many-to-many relationship. For example, in Table 1, sample data has two collections and two embedding document fields, particularly, listings, reviews, address, and location that are converted into RDF Class.

Rule 2: $\forall\, k \in K_B \rightarrow \varphi(k) \in RDP \text{ AND } Dxiom_{\varphi(k)} = \varphi(DF(k)) \text{ AND } Rxiom_{\varphi(k)} = \varphi(T(k))$.

All basic type fields are converted into RDF DatatypeProperty that domain is RDF Class mapped by field domain, and the range is limited to the field type. For instance, in the sample data, field "name" and "type" are basic type fields converted into RDF DatatypeProperty. The type of the two fields is string type. But the field "name" domain is collection name "listing", field "type" domain is "location". Therefore, the first DatatypeProperty domain is RDF Class corresponding to collection name "listing", and the other is RDF Class corresponding field "location".

Most datatypes in MongoDB can be transformed into RDF datatypes with XML Schema Definition (XSD). But the others (such as ObjectId) cannot be mapped in XSD. Therefore, we defined customized identifiers to represent these datatypes into RDF. Table I gives the mapping rules from MongoDB datatypes to RDF datatypes.

TABLE I. Mapping MongoDB datatypes to RDF datatypes

MongoDB	RDF
String	xsd:string
Binary data	xsd:hexBinary
ObjectId	mongodb:ObjectId
Boolean	xsd:boolean
Date	xsd:date
Null	mongodb:null
Regular Expression	mongodb:re
JavaScript	mongodb:js
32-bit integer	xsd:int
Timestamp	xsd:datetime
64-bit integer	xsd:long
Decimal128	xsd:decimal
Min key	mongodb:minkey
Max key	mongodb:maxkey
Object	Class
Array	Seq

Rule 3: $\forall\, k \in K_E \rightarrow \varphi(k) \in ROP \text{ AND } Dxiom_{\varphi(k)} = \varphi(DF(k)) \text{ AND } Rxiom_{\varphi(k)} = \varphi(k)$.

All embedding fields are extracted into RDF object properties domain of which is the RDF Class corresponding to field domain and range is class corresponding to the embedding field. For example, embedding fields "address" and "location" are transformed into RDF object properties.

Rule 4: $\forall\, k \in K_D \rightarrow \varphi(k) \in ROP \text{ AND } Dxiom_{\varphi(k)} = \varphi(DF(k)) \text{ AND } Rxiom_{\varphi(k)} = \varphi(V(k).\$ref,\ V(k).\$db)$.

A DBRef field is extracted into RDF object property domain of which is the class corresponding to the field domain, and range is class corresponding to the field value ($ref and $db in the DBRef). For example, DBRef field "review" is converted into RDF object property that domain is class corresponding to the field domain (collection "listing") and range is also class corresponding to field value (referenced collection "reviews").

Rule 5: $\forall$ k∈ K_M→ ϕ (k)∈ ROP AND Dxiom_φ(k) =φ(DF(k)) AND Rxiom_φ(k) =φ(K_M (k)).

All manual referencing fields are converted into RDF object properties that domain is class corresponding to the field domain and range is class corresponding to user-defined information for the manual referencing field. In Table 1, a user defines the referenced collection "reviews" and referenced field "_id" of the manual referencing field "review_id" by providing extra information. The manual referencing field is converted into the RDF object property, domain of which is a class corresponding to the field domain, and the range is corresponding to the referenced collection "reviews".

Rule 6: $\forall\, k \in K_A \rightarrow \varphi(k) \in ROP \text{ AND } Dxiom_{\varphi(k)} = \varphi(DF(k)) \text{AND } Rxiom_{\varphi(k)} = \text{ rdf: Seq}$.

For each array field, create a object property and map field value to Seq, an RDF container.

Rule 7: $\forall\, c \in C, d \in D(c) \rightarrow \varphi(V(d.\_id)) \in RI$.

For each document of a collection, create an RDF individual from document identifier (is always field "_id").

Rule 8: $\forall c \in C, d \in D(c), k_i \in \alpha(d) \cap K_B, k_j \notin \alpha(d) \cap K_B \rightarrow \varphi(V(k_i)) \in RL \text{ AND } \varphi\left(V(k_j)\right) \in RI$.

With this rule, each field value of the document is converted into an RDF individual or a literal. When the field belongs to a basic type field, it will be transformed into a literal. Otherwise, the field value will be mapped into an RDF individual. For example, when the field belongs to the array field, create a RDF container Seq and add each value in array into the container.

On the basis of formal mapping rules above, it is not difficult to develop the mapping algorithm from MongoDB into RDF.

V. System Implementation and Experimental Analysis

A. System architecture

To validate our method of mapping MongoDB data to an RDF model, we developed a prototype system named M2R, which supports extracting the hierarchical structure of MongoDB database data, mapping it to an RDF concept layer, and transforming document data into RDF instance.

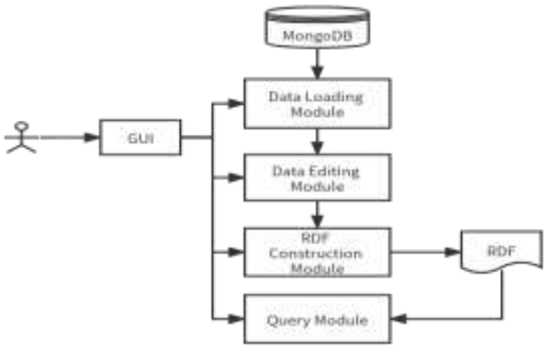

Fig. 1. The system architecture of M2R

As shown in the Fig. 1, the system mainly consists of four modules: database access module, database editing module, RDF construction module and query module.

(a) Database access module: The database access module obtains database connections with information including database address, database port, username, authentication database, and password. With the database connection, the system can access data in MongoDB.

(b) Database editing module: The database editing module executes the MongoDB statements entered by the user to finish the basic operations called CRUD of MongoDB.

(c) RDF construction module: This module constructs RDF with the MongoDB database selected by a user. Firstly, this module parses the documents in the database to extract the hierarchical structure information of the database. Then, the module uses the mapping rules and algorithms defined above to convert the hierarchical structure to the RDF concepts, turn the data to the RDF instance according to the key-value data of the documents. Because without database schema in MongoDB, this module converts each document and gradually refine the built RDF concept until all documents are parsed and mapped.

(d) Query module: This module processes MongoDB query statement input by users. It converts the statement into the corresponding SPARQL query statement and executes the MongoDB query statement and SPARQL query statement. At last, the verification module returns the results of two queries.

The M2R is developed with OpenJDK 17.0.1 platform and particularly its graphical user interface (GUI) is exploited by using JavaFX. The M2R is implemented and run with a PC (Intel i5-5200U (4) @ 2.700GHz, RAM 8 GB, and ArchLinux system).

The screen snapshot of M2R running the MongoDB database sample_analystics as an example is shown in Fig. 2. Fig. 2 shows that the GUI of M2R contains three display areas. In the left area, the TextArea displays the contents of the MongoDB database. And the TextArea (in the upper right) below the label "RDF Schema" shows the corresponding RDF Concept such as Classes and Properties in the form of RDF Turtle. The TextArea (in the lower right) below the label "RDF Individual" exhibits the corresponding RDF individuals.

Fig. 2. The screen snapshot of M2R

B. Experimental results

To verify the feasibility of converting MongoDB data into RDF with OWL2 vocabulary, we collected 8 sample databases provided by MongoDB officials covering finance, geographic, and weather information. These sample databases are shown in Table II. Compared with the traditional relational database, one of the characteristics of a document database is that it can store data nested in one collection to represent a reference relationship. Yet, relational databases need to associate multiple tables with foreign keys to achieve the same effect. Therefore, the measurement of the database considers the maximum nesting depth of documents. These eight sample databases contain different nested document depths, such as the sample_geospatial with zero nested documents, which records geographic information with simple geographic latitude and longitude information.

TABLE II. Sample database information

sample database	database size (MB)	max nested depth
sample_airbnb	90	2
sample_analytics	15.79	2
sample_geospatial	3.48	0
sample_mflix	49.88	2
sample_restaurants	13.36	1
sample_supplies	4.13	1
sample_training	113.88	2
sample_weatherdata	16.15	2

We conducted a series of experiments with databases in Table II and compared them with the work of [1] and [17]. Since [1] and [17] do not provide system source code, we can only reproduce their research work as much as possible according to their papers. We take "sample_training" as example and Table III show the final experiment results. In these tables, the first column represents the evaluating indicator of constructed RDF by these methods, the second column notes the result of our method, and the third to fifth columns show the metric result of related work [1] and [17]. The cell value "Construct time" means time-consuming specified in milliseconds of converting MongoDB to RDF dataset. The cell value "Class" means the number of Class that may be RDF Class, RDFS Class or OWL Class in the converted RDF dataset by these methods. The cell value "DatatypeProperty" means the number of DatatypeProperty in the converted RDF dataset. The cell value "ObjectProperty" means the number of ObjectProperty in the converted RDF dataset. The cell value "Property domain" means the number of Domain axiom in the converted RDF dataset. The cell value "Property range" means the number of Range axiom in the converted RDF dataset. The cell value "triple number" means the size of the converted RDF dataset. The result of above tables shows that our is slightly inferior to [1] and [17] in conversion speed, but it far exceeds them in the integrity of the built RDF. particularly, our method can construct a better RDF model with more Class, DatatypeProperty, ObejctProperty, domain axiom and range axiom triples.

TABLE III. COMPARING RESULT WITH RELATED WORK ABOUT SAMPLE_TRAINING DATABASE

Evaluating indicator	Our Method (M2R)	[1]	[17]
Construct time (ms)	66837	53181	46785
Class	38	15	7
DatatypeProperty	172	65	82
ObjectProperty	37	8	0
Property domain	209	73	0
Property range	209	73	0
Triple	6535861	2410669	2229094
Construct time (ms)	66837	53181	46785

To check the feasibility of our method, we construct some SPARQL queries on the converted RDF. In Table IV, we select a set of MongoDB query statements, construct the corresponding SPARQL statement as shown in Table V to query the RDF dataset created from MongoDB, and verify the correctness and integrity of the mapped RDF data. If the SPARQL query results match the MongoDB query results, then we would hold the conversion is valid and correct.

TABLE IV. QUERIES OF MONGODB WITH SAMPLE DATABASE SAMPLE ANALYTICS

Query 1: Return the name and birthdate of the custom called "serranobrian" db.customers.find({username: "serranobrian"},{name: 1, birthdate: 1})
Query 2: Extract one customer information with properties username, name, address db.customers.findOne({}, {username: 1, name: 1, address: 1})
Query 3: Return purchased product information of the account with id 170945 db.accounts.find({account id: 170945},{products: 1})

Query 4: Return the purchased product information of each account of the customer called "serranobrian"
result = db.customers.findOne({username: "serranobrian"},{accounts: 1})
db.accounts.find({account id: {$in: result.accounts}},{account id: 1,products: 1})

Query 5: Return transcation information with field "date" and "code" in collection transcations with account id 209363 db.transcations.find({account id:209363}, {"transaction.date":1,"transaction.code":1})
Query 6: Return transcation information with field "date" and "code" in collection transcations with account id 209363 in the form of DBRef db.transactions.find({account_id:DBRef("accounts",209363)},{"transaction.date":1, "transaction.code":1})

TABLE V. SPARQL REPRESENTATION OF QUERIES Q1-Q6

Query 1: SELECT ?name ?birthdate WHERE { ?customer rdf:type ns:customer . ?customer ns:has-username "serranobrian"^^xsd:string . ?customer ns:has-name ?name . ?customer ns:has-birthdate ?birthdate .}
Query 2: SELECT ?username ?name ?address WHERE { ?customer rdf:type ns:customer . ?customer ns:has-username ?username . ?customer ns:has-name ?name . ?customer ns:has-address ?address . } LIMIT 1
Query 3: SELECT ?product WHERE { ?account rdf:type ns:accounts . ?account ns:has-accountId "170945"^^xsd:int . ?account ns:ref-products/(rdf:rest*/rdf:first)* ?products . }
Query 4: SELECT ?accountId ?product WHERE { ?customer rdf:type ns:customers . ?customer ns:has-username "serranobrian"^^xsd:string . ?customer ns:ref-accounts/(rdf:rest*/rdf:first)* ?account . ?account rdf:type ns:accounts . ?account ns:has-account id ?accountId . ?account ns:ref-products/(rdf:rest*/rdf:first)* ?products . }
Query 5: SELECT ?date ?code WHERE { ?transactions rdf:type ns:transactions . ?transactions ns:has-accountId "209366"^^xsd:int . ?transactions ns:ref-transaction/(rdf:rest*/rdf:first)* ?transaction . ?transaction rdf:type ns:transaction . ?transaction ns:has-date ?date . ?transaction ns:has-transaction code ?code . }
Query 6: SELECT ?date ?code WHERE { ?account rdf:type ns:accounts . ?account ns:has-accountId 209363^^xsd:int . ?transactions rdf:type ns:transactions . ?transactions ns:ref-account id ?account . ?transactions ns:ref-transaction/(rdf:rest*/rdf:first)* ?transaction . ?transaction rdf:type ns:transaction . ?transaction ns:has-date ?date . ?transaction ns:has-transaction code ?code . }

Tables VI show the results of query Q4 in Table IV. The SPARQL queries in these tables include all of the information consistent with MongoDB queries. In Query 4, for example, the MongoDB query gets all accounts information stored in an Array of the customer "serranobrian", and then query purchased products by each account. The MongoDB query returns two accounts that the one purchased three products and the other one purchased two products.

TABLE VI. MONGODB AND SPARQL RESULTS OF QUERY 4

{"_id":ObjectId("5ca4bbc7a2dd94ee58162456"), "account_id": 170945, "products": ["Derivatives", "Commodity", "InvestmentStock"] }
{"_id":ObjectId("5ca4bbc7a2dd94ee58162457"), "account_id": 951849, "products": ["Brokerage", "InvestmentStock"] }	

accountId	product
"170945"^^xsd:int	"Derivatives"
"170945"^^xsd:int	"Commodity"
"170945"^^xsd:int	"InvestmentStock"
"951849"^^xsd:int	"Brokerage"
"951849"^^xsd:int	"InvestmentStock"

We illustrate briefly that our construction method is valid, correct, and better than related work for constructing RDF with MongoDB, especially when the database has a complex structure. And we also demonstrate that we can use the converted RDF data to finish the same work as in MongoDB.

Quality assessment of RDF data is crucial because many applications such as intelligent search, recommendation systems, and smart medical treatment may not take full advantage of low-quality RDF dataset [15]. To evaluate the quality of converted RDF, we first manually build an ontology with sample database sample_mflix followed by Ontology Development 101 that is a guide to build an ontology. The reason for choosing the database is that this database is designed for MongoDB rather than simple migration from other kinds of databases such as relational databases which data model is different from MongoDB. Then we choose three metrics, precision, recall, and F-measure adapted from information retrieval.

$$\text{precision} = \frac{|R \cap T|}{|T|}, \text{recall} = \frac{|R \cap T|}{|R|}$$

$$\text{F} - \text{measure} = 2 * \frac{precision * recall}{precision + recall}$$

In metrics precision and recall, the R denotes the manually built ontology, and T denotes converted RDF data with our method. |R| denotes the triple number of R, |T| denotes the triple number of T, and |R ∩ T| denotes same triple number between R and T.

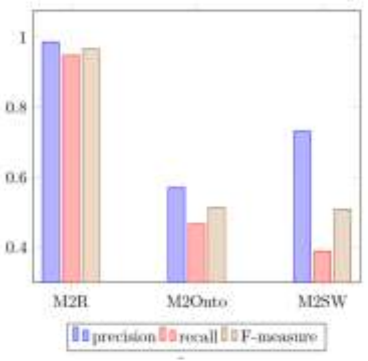

Fig. 3. Comparison of three metrics between our method and related work on sample_mflix

Fig. 3 presents the metrics result between our method and other related work. Our approach performance is better than related work. For example, the F-measure of our method is 96.69% that is higher than 51.43% of [1] denoted by M2Onto and 50.85% of [17] denoted by M2SW. The main reason is that our method constructs RDF Classes and Properties with relationships modelled by embedding and referencing and generates RDF containers to represent Array data of MongoDB. For example, our approach can construct RDF ObjectProperty by receiving user input information about the manual reference relationship.

VI. Conclusions and Future Work

With the rapid development of Web-based applications, data is increasing so dramatically that many people prefer NoSQL databases rather than relational databases to solve the data management problems brought by big data. MongoDB occupies most of the application market for NoSQL databases. Therefore, converting MongoDB data to RDF can effectively use the data and solve insufficient available RDF data. This paper focuses on the characteristics of MongoDB and RDF, proposes formal definitions of the MongoDB data model and RDF model, and then design an algorithm to map MongoDB hierarchy and data to RDF. With the proposed transformation method, we implemented a prototype system named M2R and carried out experiments to verify the feasibility of the work. Experimental results show the feasibility of constructing RDF with MongoDB. In our future work, we will try to optimize the implementation of the mapping algorithm to improve the efficiency of the mapping process and enhance the speed of transformation. We will also consider extending the method to support more document database transformations, not just MongoDB.

Acknowledgment

The work was supported in part by the National Natural Science Foundation of China (62176121 and 61772269).

References

[1] Abbes H, Gargouri F. Big data integration: A mongodb database and modular ontologies based approach. Proceedings of the 20th International Conference KES-2016, York, UK, 5-7 September 2016, Procedia Computer Science, vol 96. Elsevier, pp 446-455.

[2] Antoniou G, van Harmelen F. A semantic web primer. MIT Press, 2004.

[3] Berners-Lee T, Hendler J, Lassila O. The semantic web. Scientific American 284(5):34-43.

[4] Bischof S, Decker S, Krennwallner T, et al. Mapping between RDF and XML with XSPARQL. J Data Semant 1(3):147-185.

[5] Bytyçi E, Ahmedi L, Gashi G. RDF mapper: Easy conversion of relational databases to RDF. Proceedings of the 14th International Conference on Web Information Systems and Technologies, WEBIST 2018, Seville, Spain, September 18-20, 2018. SciTePress, pp 161-165.

[6] García-González H, Gayo JEL. Xmlschema2shex: Converting XML validation to RDF validation. Semantic Web 11(2):235-253.

[7] Gupta R, Gupta H, Mohania MK. Cloud computing and big data analytics: What is new from databases perspective?, First International Conference on Big Data Analytics, New Delhi, India, December 24-26, 2012.

[8] Huang J, Lange C, Auer S. Streaming transformation of XML to RDF using xpath-based mappings. Proceedings of the 11th International Conference on Semantic Systems, Vienna, Austria, September 15-17, 2015.

[9] Jabbari S, Stoffel K. Ontology extraction from MongoDB using formal concept analysis. In: 2017 2nd International Conference on Knowledge Engineering and Applications. IEEE, London, pp 178-182, 2017.

[10] Kharrat M, Jedidi A, Gargouri F. A semantic approach for transforming XML data to RDF triples. 14th IEEE/ACIS International Conference on Computer and Information Science, Las Vegas, NV, USA, June 28 - July 1, 2015. IEEE Computer Society, pp 285-290.

[11] Lisena P, Troncy R. Transforming the JSON output of SPARQL queries for linked data clients. Companion of the Web Conference 2018 on The Web Conference 2018, Lyon , France, April 23-27, 2018. ACM, pp 775-780.

[12] Liu Y, Hong Z. Mapping XML to RDF: An algorithm based on element classification and aggregation. Journal of Physics: Conference Series 1848(1):012,012.

[13] Ma Z, Capretz MAM, Yan L. Storing massive resource description framework (RDF) data: a survey. Knowl Eng Rev, 31(4): 391-413.

[14] Méndez SJR, Haller A, Omran PG, et al (2020) J2RM: an ontology-based json-to-rdf mapping tool. Proceedings of the ISWC 2020 Demos and Industry Tracks: From Novel Ideas to Industrial Practice co-located with 19th International Semantic Web Conference, Globally online, November 1-6, 2020 (UTC), CEUR Workshop Proceedings, vol 2721. CEUR-WS.org, pp 368-373.

[15] Sejdiu G, Rula A, Lehmann J, et al. A Scalable Framework for Quality Assessment of RDF Datasets. ISWC 2019. Springer International Publishing, Cham, Lecture Notes in Computer Science, pp 261-276.

[16] Sequeda JF, Arenas M, Miranker DP. On directly mapping relational databases to RDF and OWL. Proceedings of the 21st World Wide Web Conference 2012, Lyon, France, April 16-20, 2012. ACM, pp 649-658.

[17] Soussi N, Bahaj M. Exploiting nosql document oriented data using semantic web tools. Advanced Intelligent Systems for Sustainable Development (AI2SD'2018). Springer, Cham, pp 110-117.

[18] Yan L, Sheng J, Tu Y, et al. Automatic construction of RDF with web tables. Expert Syst Appl 182:115, 200.