


Chapter 13

Towards Massive RDF Storage in NoSQL Databases: A Survey

Zongmin Ma

 <https://orcid.org/0000-0001-7780-6473>

Nanjing University of Aeronautics and Astronautics, China

Li Yan

Nanjing University of Aeronautics and Astronautics, China

ABSTRACT

The resource description framework (RDF) is a model for representing information resources on the web. With the widespread acceptance of RDF as the de-facto standard recommended by W3C (World Wide Web Consortium) for the representation and exchange of information on the web, a huge amount of RDF data is being proliferated and becoming available. So, RDF data management is of increasing importance and has attracted attention in the database community as well as the Semantic Web community. Currently, much work has been devoted to propose different solutions to store large-scale RDF data efficiently. In order to manage massive RDF data, NoSQL (not only SQL) databases have been used for scalable RDF data store. This chapter focuses on using various NoSQL databases to store massive RDF data. An up-to-date overview of the current state of the art in RDF data storage in NoSQL databases is provided. The chapter aims at suggestions for future research.

INTRODUCTION

The Resource Description Framework (RDF) is a framework for representing information resources on the Web, which is proposed by W3C (World Wide Web Consortium) as a recommendation (Manola and Miller, 2004). RDF can represent structured and unstructured data (Duan, Kementsietsidis, Srinivas and Udrea, 2011), and more important, metadata of resources on the Web represented by RDF can be shared and exchanged among application programming without semantic missing. Here metadata mean

DOI: 10.4018/978-1-5225-8446-9.ch013

the data that specify semantic information about data. Currently RDF has been widely accepted and has rapidly gained popularity. And many organizations, companies and enterprises have started using RDF for representing and processing their data. We can find some application examples such as the United States¹, the United Kingdom², New York Times³, BBC⁴, and Best Buy⁵. RDF is finding increasing use in a wide range of Web data-management scenarios.

With the widespread usage of RDF in diverse application domains, a huge amount of RDF data is being proliferated and becoming available. As a result, efficient and scalable management of large-scale RDF data is of increasing importance, and has attracted attentions in the database community as well as the Semantic Web community. Currently, much work is being done in RDF data management. Some RDF data-management systems have started to emerge such as *Sesame* (Broekstra, Kampman and van Harmelen, 2002), *Jena-TDB* (Wilkinson, Sayers, Kuno and Reynolds, 2003), *Virtuoso* (Erling and Mikhailov, 2007 & 2009), *4Store* (Harris, Lamb and Shadbolt, 2009), *BigOWLIM* (Bishop *et al.*, 2011) and *Oracle Spatial and Graph with Oracle Database 12c*⁶. Here BigOWLIM is renamed to OWLIM-SE and further to GraphDB. Also some research prototypes have been developed (e.g., RDF-3X (Neumann and Weikum, 2008 & 2010), SW-Store (Abadi, Marcus, Madden and Hollenbach, 2007 & 2009) and RDFox⁷).

RDF data management mainly involves scalable storage and efficient queries of RDF data, in which RDF data storage provides the infrastructure for RDF data management and efficient querying of RDF data is enabled based on RDF storage. In addition, to serve a given query more effectively, it is necessary to index RDF data. Indexing of RDF data is enabled based on RDF storage also. Currently many efforts have been made to propose different solutions to store large-scale RDF data efficiently. Traditionally relational databases are applied to store RDF data and various storage structures based on relational databases have been developed. Based on the relational perspective, Sakr and Al-Naymat (2009) present an overview of relational techniques for storing and querying RDF data. It should be noted that the relational RDF stores are a kind of centralized RDF stores, which are a single-machine solution with limited scalability. The scalability of RDF data stores is essential for massive RDF data management. NoSQL (for “not only SQL”) databases have recently emerged as a commonly used infrastructure for handling Big Data because of their high scalability and efficiency. Identifying that massive RDF data management merits the use of NoSQL databases, currently NoSQL databases are increasingly used in massive RDF data management (Cudre-Mauroux *et al.*, 2013).

This chapter provides an up-to-date overview of the current state of the art in massive RDF data stores in NoSQL databases. We present the survey from three main perspectives, which are key value stores of RDF data in NoSQL databases, document stores of RDF data in NoSQL databases and RDF data stores in graph databases. Note that, due to the large number of RDF data-management solutions, this chapter does not include all of them. In addition to provide a generic overview of the approaches that have been proposed to store RDF data in NoSQL databases, this chapter presents some suggestions for future research in the area of massive RDF data management with NoSQL databases.

The rest of this chapter is organized as follows. The second section presents preliminaries of RDF data model. It also introduces the main approaches for storing RDF data. The third section introduces NoSQL databases and their database models. The fourth section provides the details of the different techniques in several NoSQL-based RDF data stores. The final section concludes the chapter and provides some suggestions for possible research directions on the subject.

RDF Model and RDF Data Storage

Being a W3C recommendation, RDF (Resource Description Framework) provides a means to represent and exchange semantic metadata. With RDF, metadata about information sources are represented and processed. Furthermore, RDF defines a model for describing relationships among resources in terms of uniquely identified attributes and values.

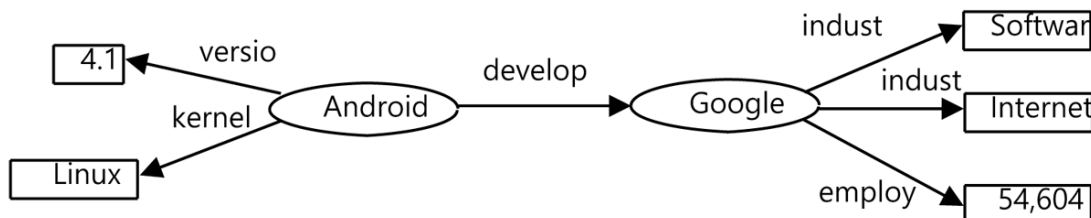
The RDF data model is applied to model resources. A resource is anything which has a universal resource identifier (URI) and is described using a set of RDF statements in the form of *(subject, predicate, object)* triples. Here *subject* is the resource being described, *predicate* is the property being described with respect to the resource, and *object* is the value for the property.

Formally, an RDF triple is defined as (s, p, o) , in which s is called the subject, p the predicate (or property), and o the object. Triple (s, p, o) means that the subject s has the property p whose value is the object o . The abstract syntax of RDF data model is a set of triples. Among these triples, it is possible that an object in one triple (e.g., o_i in (s_i, p_i, o_i)) can be a subject in other triples (e.g., (o_i, p_j, o_j)). Then RDF data model is a directed, labelled graph model for representing Web resources (Huang, Abadi and Ren, 2011). A key concept for RDF is that of URIs (Unique Resource Identifiers), which can be applied in either of the s, p and o positions to uniquely refer to entity, relationship or concept (Kaoudi and Manolescu, 2015). In addition, literals are allowed in the o position. Let I, B , and L denote infinite sets of IRIs, blank nodes, and literals, respectively. Then we have $(s, p, o) \in (I \cup B) \times I \times (I \cup B \cup L)$. Concerning the syntaxes for RDF, we have RDF/XML, N-Triple and Turtle. Among them, N-Triple is the most basic one and it contains one triple per line.

In (Bornea *et al.*, 2013), a sample of DBpedia RDF data is presented and it contains 21 tuples and 13 predicates. Let us look at a fragment of the sample, which contains 6 triples $\{(\text{Google}, \text{industry}, \text{Software}), (\text{Google}, \text{industry}, \text{Internet}), (\text{Google}, \text{employees}, 54,604), (\text{Android}, \text{developer}, \text{Google}), (\text{Android}, \text{version}, 4.1), (\text{Android}, \text{kernel}, \text{Linux})\}$ and 5 predicates $\{\text{industry}, \text{employees}, \text{developer}, \text{version}, \text{kernel}\}$. For a triple, say $(\text{Google}, \text{industry}, \text{Internet})$, its resource is *Google*, this resource has the property *industry*, and the value for the property is *Internet*. With the set of triples, we have an RDF graph shown in Figure 1.

As we know, in the context of common data management, data are stored early in file systems and late in databases (such as relational databases and object-oriented databases). Similarly, RDF data management relies on RDF data storage also. It is especially true for managing large-scale RDF data in the real-world applications. Nowadays many approaches of RDF data storages have been developed. Given the large number of RDF data-management solutions, there is a richness of perspectives and approaches to RDF data storages. But few classifications of RDF data-storage approaches have been reported in the

Figure 1. An RDF graph



literature. Basically, proposals for RDF data storage are classified into two major categories in (Sakr and Al-Naymat, 2009; Bornea *et al.*, 2013), which are *native stores* and *relational stores*, respectively. Here the native stores use customized binary RDF data representation and the relational stores distribute RDF data to appropriate relational databases. Viewed from three basic perspectives of the relational, entity and graph-based perspectives, proposals for RDF data storage are classified into three major categories in (Luo *et al.*, 2012), which are *relational stores*, *entity stores* and *graph-based stores*, respectively.

RDF data stores can be classified relying on the underlying infrastructure. First, for the native stores, we can identify *main-memory-based RDF native store* and *disk-based RDF native store*. The major difference between these two native stores is that the former works on RDF data stored completely in main memory and the latter works on RDF data stored in disk. The disk-based RDF native store is built directly on the file system. It is not difficult to see that the native stores can only deal with small-scale RDF data. At this point, traditional relational databases are hereby applied to store RDF data. In the relational stores of RDF data, different relational schemas can be designed, depending on how to distribute RDF triples to an appropriate relational schema. This has results in three major categories of RDF relational stores (Sakr and Al-Naymat, 2009; Bornea *et al.*, 2013; Luo *et al.*, 2012), which are *triple stores*, *vertical stores* and *property (type) stores*. In the triple stores, all RDF triples are directly stored in a single relational table over relational schema (*subject, predicate, object*), and each RDF triple becomes a tuple of the relational database. In the vertical stores, a subject-object relation is directly represented for each predicate of RDF triples and a relational table contains only one predicate as a column name. As a result, we have a set of binary relational tables over relational schema (*subject, object*), each relational table corresponds to a different predicate, and this predicate can be the name of the corresponding relational table. In the type stores, one relational table is created for each RDF data type, in which an RDF data type generally corresponds to several predicates, and a relational table contains the properties as *n*-ary table columns for the same subject.

The reason why there are three major approaches for storing RDF data in relational databases is that each approach has its advantages and disadvantages simultaneously. First, the triple stores use a fixed relational schema and can hereby handle dynamic schema of RDF data, but the triple stores involve a number of self-join operations for querying. Second, the vertical stores using a set of relational tables generally involve many table join operations for querying and cannot handle dynamic schema of RDF data because new predicates of new inserted triples result in new relational tables. Finally the type stores involve fewer table join operations than the vertical stores using multiple relational tables and, compared with the triple stores using a single relational table, no self-join operations, but the type stores generally contain null values and multi-valued attributes, and cannot handle dynamic schema of RDF data because new predicates of new inserted triples result in changes to relational schema.

Several representative relational RDF data stores and their major features are summarized in Table 1.

It can be seen from Table 1 that any one of the approaches for relational RDF data stores presented above cannot deal with RDF data stores well. So, some efforts have been made to store RDF data by using two or more of three major store approaches together or revising three major store approaches (Kim, 2006; Sperka and Smrz, 2012; Bornea *et al.*, 2013). Such an approach is called *hybrid stores*. It should be noted that the hybrid stores cannot still satisfy the need of managing large-scale RDF data.

The native RDF stores and the relational RDF stores (including the triple stores, vertical stores, type stores and hybrid stores) are actually categorized as *centralized RDF stores*. Centralized RDF stores are single-machine solutions with limited scalability. To process large-scale RDF data, recent research has devoted considerable effort to the study of managing massive RDF data in distributed environments.

Table 1. Representative relational RDF data stores and their major features

| | Typical store systems | Advantages | Disadvantages |
|------------------------|---|--|--|
| Triple stores | <i>Sesame</i> (Broekstra, Kampman and van Harmelen, 2002); <i>Hexastore</i> (Weiss, Karras and Bernste, 2008) | <i>no null values; no multi-valued attributes; fixed relational schema</i> | <i>too many self-joins</i> |
| Vertical stores | <i>SW-Store</i> (Abadi, Marcus, Madden and Hollenbach, 2007 & 2009) | <i>no null values; no multi-valued attributes</i> | <i>too many table joins; dynamic relational schema</i> |
| Type stores | <i>Jena</i> (McBride, 2002); <i>FlexTable</i> (Wang, Du, Lu and Wang, 2010); <i>RDFBroker</i> (Sintek and Kiesel, 2006) | <i>few table joins</i> | <i>null values; multi-valued attributes; dynamic relational schema</i> |

Distributed RDF stores can hash partition triples across multiple machines and parallelize query processing. NoSQL databases have recently emerged as a commonly used infrastructure for handling Big Data (Pokorny, 2011). Massive RDF data management merits the use of NoSQL databases and currently NoSQL databases are increasingly used in RDF data management (Cudre-Mauroux *et al.*, 2013). Typically, NoSQL database stores of RDF data are distributed RDF stores. Depending on concrete data models adopted, the NoSQL database stores of RDF data are categorized into *key-value stores*, *column-family stores*, *document stores* and *graph databases stores*.

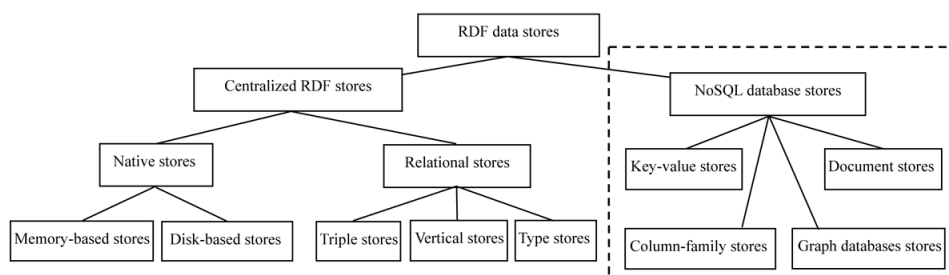
Summarily, we can classify current RDF data stores into the centralized RDF stores and the NoSQL database stores of RDF data (Papailiou *et al.*, 2013). The centralized RDF stores can be further classified into the native RDF stores and the relational RDF stores, in which the native RDF stores contain main-memory-based native RDF store and disk-based native RDF store, and the relational RDF stores include triple stores, vertical stores and type stores. The NoSQL database stores of RDF data are further classified into key-value stores, column-family stores, document stores and graph databases stores. Figure 2 illustrates this classification for RDF data stores.

The native RDF stores and the relational RDF stores have been reviewed in (Sakr and Al-Naymat, 2009). The focus of this chapter is to investigate NoSQL database stores of RDF data. Before that, we first sketch NoSQL databases in the following section.

NoSQL Databases for Big Data

We use the term of Big Data to refer to massive and complex datasets that are made up of a variety of data structures. Big Data can be found in diverse application domains such as web clicks, social media,

Figure 2. Classification of RDF data stores



scientific experiments, datacenter monitoring and so on. Actually, there is not a common definition of Big Data so far (Stuart and Barker, 2013). But Big Data are generally characterized by three basic Vs: *Volume*, *Variety* and *Velocity* (Laney, 2001).

- **Volume:** Volume means that Big Data have big data scale in the range of TB to PB and even more.
- **Variety:** Variety means Big Data have rich data types with many formats such as structured data, unstructured data, semistructured data, text, multimedia, and so on.
- **Velocity:** Velocity means that Big Data must be processed speedily. Also, Velocity means that Big Data are being produced speedily.

Then Big Data is high-volume, high-velocity and high-variety information assets that demand cost-effective, innovative forms of information processing for enhanced insight and decision making. In addition to the three Vs above, a V associated with *Veracity* has been introduced to Big Data in (Snow, 2012) and a V associated with *Value* has been introduced to Big Data in (Gamble and Goble, 2011).

- **Veracity:** Veracity means that inherent imprecision and uncertainty in data should be explicitly considered so that the reliability and predictability of imprecise and uncertain Big Data can be managed.
- **Value:** Value means that Big Data must be worthwhile and has value for business.

The Veracity of data is a basis of Big Data processing because the data with volume and variety may contain errors, noises or imperfection. Actually, the Veracity of data is a precondition and guarantee of Big Data management, which can increase the robustness and accuracy of Big Data processing. Regarding to the Value of data, Value sets a basic criterion in the choice and processing of Big Data, which is especially true in the context of the data with volume, variety and velocity.

Also, there are several other Vs that are applied to describe the properties of Big Data in literature (e.g., *Visualization*, *Visualization* and *Volatility*). Among them, Volatility means that Big Data that we are interested in are temporally valid. It is possible that, at one point, specific data are no longer relevant to the current processing and analysis of Big Data. It should be noted that the several Vs mentioned above only characterize some properties of Big Data partially. So, some other characteristics rather than the Vs presented above are assigned to Big Data.

Being similar with common data (not Big Data) management, Big Data management needs database systems. As we know, the relational databases are very powerful and have been widely applied for structured, semi-structured and even unstructured data. However, the relational databases are unable to manage Big Data. The reason is that the relational databases must meet ACID according to relational databases theory but Big Data management need CAP Theorem. ACID means the type of transaction processing done by relational database management system (RDBMS) as follows.

- **(A)tomcity:** If a part of operation fails, whole operation fails
- **(C)onsistency:** Information is always consistent, avoid read/write errors at all costs
- **(I)solation:** Multiple transactions at the same time do not impact each other
- **(D)urability:** Information has to be stored into database, not queued in memory

CAP Theorem is described by consistency, availability and partition as follows.

- **(C)onsistency:** Whenever data is written, everyone who reads the DB will see the latest version.
- **(A)vailability:** We can always expect each operation terminates in an intended response.
- **(P)artition Tolerance:** Database can still be read from/written to when parts of it are offline; afterwards, when offline nodes are back online, they are updated accordingly.

It is clear that the relational databases are not solutions in Big Data management. A new type of databases called NoSQL is hereby proposed, which means “not only SQL” or “no SQL at all”.

NoSQL database systems have emerged as a commonly used infrastructure for handling Big Data. Comparing to traditional relational databases, NoSQL solutions provide simpler scalability and improved performance (Hecht and Jablonski, 2011; Pokorny, 2011; Moniruzzaman and Hossain, 2013; Gudivada, Rao and Raghavan, 2014) and generally have some characteristics as follows (Tauro, Aravindh and Shreeharsha, 2012).

- Distributed processing
- High availability
- High scalability and reliability
- Scheme-less
- Replication support
- Handle structured and unstructured data
- Data access via API
- Less strict adherence to consistency
- Improvements in performance

It should be noted that NoSQL databases are very diverse and there are more than one hundred NoSQL databases⁸. According to their data model, the various NoSQL databases are classified into four major categories as follows (Hecht and Jablonski, 2011; Grolinger *et al.*, 2013; Bach and Werner, 2014).

- Key-value NoSQL databases (e.g., DynamoDB, Riak, Redis)
- Column-oriented NoSQL databases (e.g., HBase, Cassandra, Hypertable)
- Document-oriented NoSQL databases (e.g., MongoDB, Couchbase Sever, Elasticsearch)
- Graph-oriented NoSQL databases (e.g., Neo4j, InfoGrid, HyperGraphDB)

Key-value stores have a simple data model based on key-value pairs, which contain a set of couples (key, value). A value is addressed by a single key. Here a value may be a string, a pointer (where the value is stored) or even a collection of couples (name, value) (e.g., in Redis⁹). Note that values are isolated and independent from others, in which the relationship is handled by the application logic.

Most column-family stores are derived from Google BigTable (Chang *et al.*, 2008), in which the data are stored in a column-oriented way. In BigTable, the dataset consists of several rows. Each row is addressed by a primary key and is composed of a set of column families. Note that different rows can have different column families. Representative column-family stores include Apache HBase¹⁰, which directly implements the Google BigTable concepts. In addition, Cassandra (Lakshman and Malik, 2010) provides the additional functionality of super-columns, in which a column contains nested (sub)columns and super-columns are formed by grouping various columns together. According to (Grolinger *et al.*, 2013), there is one type of column-family store, say Amazon SimpleDB and DynamoDB (DeCandia

et al., 2007), in which only a set of column name-value pairs is contained in each row, without having column families. So SimpleDB and DynamoDB are generally categorized as key-value stores as well.

Document stores provide another derivative of the key-value store data model that uses keys to locate documents inside the data store. Most document stores represent documents using *JSON* (JavaScript Object Notation) or some format derived from it (e.g., *BSON* (Binary JSON)). JSON is a binary and typed data model which supports the data types list, map, date, Boolean as well as numbers of different precisions. Typically, CouchDB¹¹ and the Couchbase server¹² use the JSON format for data storage, whereas MongoDB¹³ stores data in *BSON*.

Graph databases, a special category of NoSQL databases, use graphs as their data model. A graph is used to represent a set of objects, known as vertices or nodes, and the links (or edges) that interconnect these vertices. Representative graph databases include GraphDB (Güting, 1994) and Neo4j¹⁴. Neo4j is an open-source, highly scalable, robust native graph database that stores data in a graph. Note that graph databases actually are developed originally for managing data with complex structures and relationships such as data with recursive structure and data with network structure rather than Big Data management. Generally, graph databases run in single server (e.g., GraphDB and Neo4j). Recently few efforts have been made to develop distributed graph databases (Nicoara, Kamali, Daudjee and Chen, 2015). Typically, Titan¹⁵ is a scalable graph database optimized for storing and querying graphs containing hundreds of billions of vertices and edges distributed across a multi-machine cluster.

Illustrative representations of four kinds of NoSQL models are shown in Figure 3, which are presented by Grolinger *et al.* (2013).

Following the four major categories of NoSQL models, some NoSQL databases have been developed and applied. Table 2 lists several representative NoSQL databases.

In the NoSQL data models presented above, column-family store and document store can be regarded as a kind of extended key value stores, in which document store is regarded that the key values are set to be the documents, and column-family store is regarded that a key value is a combination of ID of row, column number and timestamp. So sometimes column-family stores of NoSQL databases such as HBase and Cassandra are generally called key value stores of NoSQL databases. The true key value store model is too simple for many application domains such RDF data management.

NoSQL databases are designed especially for storing and processing datasets of Big Data. In the context of RDF datasets, NoSQL databases are applied for massive RDF data. In the following, we provide a survey on how massive RDF data management merits the use of NoSQL databases.

MASSIVE RDF DATA STORAGE IN NOSQL DATABASES

Being a commonly used infrastructure, NoSQL databases have been extensively applied for dealing with common Big Data. Note that, although NoSQL databases are not designed especially for RDF data management, massive RDF data management merits the use of NoSQL databases for Big Data infrastructure because of the scalability and high performance of NoSQL databases for Big Data management.

Depending on the concrete database models of NoSQL, we can identify four major kinds of RDF data stores in NoSQL databases, which are *storing RDF data in column-family stores of NoSQL databases*, *storing RDF data in document stores of NoSQL databases*, and *storing RDF data in graph databases*.

Figure 3. Different types of NoSQL data model

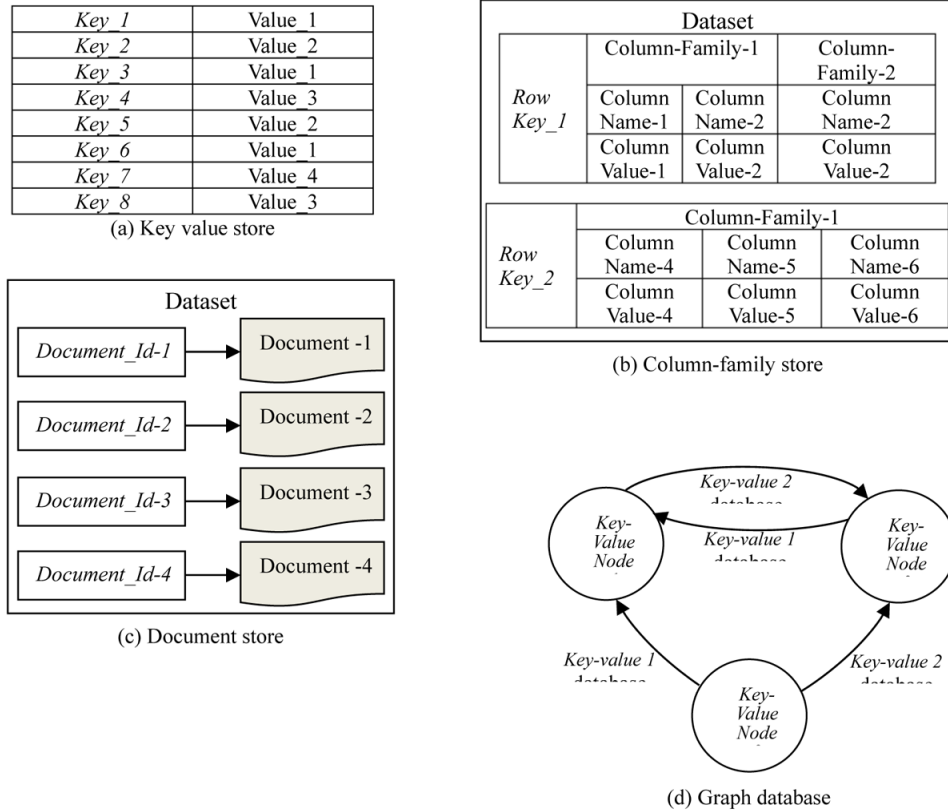


Table 2. Representative NoSQL databases

| | Representative NoSQL Databases | Remarks |
|---------------------|--|--|
| Key value store | Redis; SimpleDB; DynamoDB | In SimpleDB and DynamoDB, a set of column name-value pairs is contained in each row. |
| Column-family store | BigTable; HBase; Cassandra; Hypertable ¹⁶ | Cassandra provides the additional functionality of super-columns. |
| Document store | CouchDB; Couchbase server; MongoDB | CouchDB and Couchbase server use the JSON format for data storage. MongoDB stores data in BSON (Binary JSON) |
| Graph database | GraphDB; Neo4j; Titan | GraphDB and Neo4j run in single server. Titan is a distributed graph databases |

Storing RDF Data in Column-Family Stores of NoSQL Databases

Among the NoSQL database systems available, HBase is typically a kind of *column-family stores of NoSQL databases* and has been the most widely applied NoSQL database. Apache HBase uses HDFS (Hadoop Distributed File System) as its storage back-end (Hua, Wu, Li and Ren, 2014), and supports MapReduce computing framework (Dean and Ghemawat, 2008). Being similar to the relational databases,

HBase uses data table named *HTable*, in which each row is uniquely identified by a row key. HBase generally creates indexing on the row keys.

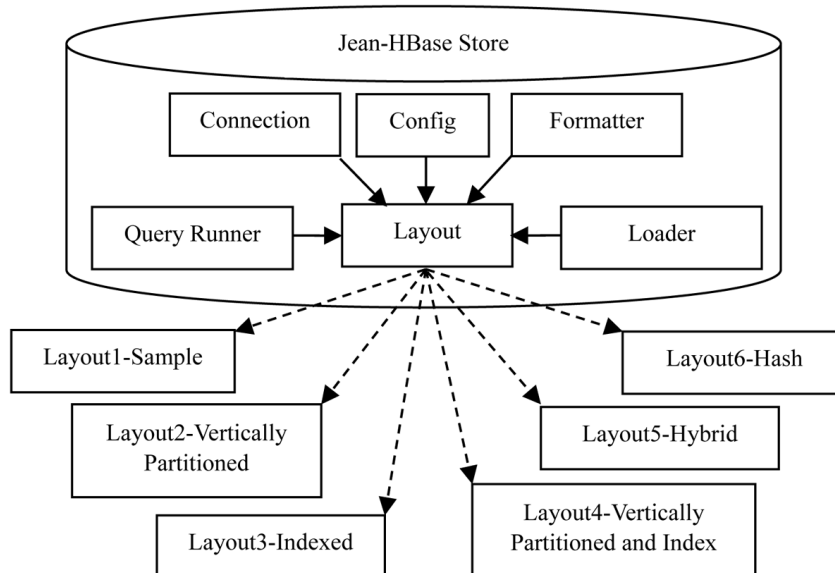
We know that RDF triples can be stored in relational databases (Ma, Capretz and Yan, 2016). Similarly, RDF triples can be stored in HTable of HBase also. Here we identify several major storage structures for HTable-based RDF store as follows.

- First, based on the idea of Hexastore schema developed in (Weiss, Karras and Bernstein, 2008), scalable RDF store based on HBase is proposed in (Sun and Jin, 2010). They store RDF triples into six HBase tables (S_PO, P_SO, O_SP, PS_O, SO_P and PO_S), which covers all combinations of RDF triple patterns. And they index RDF triples with HBase provided index structure on row key.
- Second, based on HBase, two distributed triple stores H_2RDF and H_2RDF+ are developed in (Papailiou, Konstantinou, Tsoumakos and Koziris, 2012) and (Papailiou *et al.*, 2013), respectively. The main difference between H_2RDF and H_2RDF+ is the number of maintained indices (three versus six).

In addition, to manage distributed RDF data, HBase is sometimes applied by combining others together. HBase and MySQL Cluster, for example, are used in (Franke *et al.*, 2011). Combining the Jena framework with the storage provided by HBase, Khadilkar *et al.* (2012) developed several versions of a triple store. Figure 4 presents an overview of the architecture typically used by Jena-HBase (Khadilkar *et al.*, 2012).

Here six types layout are design, which are *simple layout*, *vertically partitioned (VP) layout*, *indexed layout*, *VP and indexed layout*, *hybrid layout*, and *hash layout*. In the simple layout, three HTable tables are created and each is indexed by subjects, predicates and objects (i.e., row key), and other two components are combined together as a column name. In the vertically partitioned (VP) layout, every unique

Figure 4. Architectural overview of Jena-HBase



predicate creates two HTable tables and each is indexed by subjects and objects. In the indexed layout, six HTable tables are created for representing the six possible combinations of a triple, namely, SPO, SOP, PSO, POS, OSP and OPS. In the VP and indexed layout, the vertically partitioned (VP) layout and the indexed layout are combined together. As a result, the VP and indexed layout contains the all HTable tables in the vertically partitioned layout as well as three additional HTable tables for SPO, OSP and OS. In the hybrid layout, the simple layout and the vertically partitioned layout are combined together. So, the hybrid layout contains the three HTable tables in the simple layout as well as the all HTable tables in the vertically partitioned layout. The hash layout is a kind of hybrid layout with hash values for nodes and a separate table containing hash-to-node mappings. Khadilkar *et al.* (2012) compare with experiments these six layouts on features of data loading speed and querying efficiency and finally reveal that the hybrid layout has better performances. But the hybrid layout is complicated in design and implementation.

Storing RDF Data in Document Stores of NoSQL Databases

Document stores of NoSQL databases look like relational databases, in which each row is associated with a document instead of a tuple. It is clear that such a structure is suitable for storing semi-structure data. As mentioned above, documents in the document stores of NoSQL Databases are represented by using *JSON* (JavaScript Object Notation) or *BSON* (Binary JSON). Being a kind of data exchange format, JSON adopts text format and can be easily edited by persons and processed by machines simultaneously. As a result, it is convenient that JSON can be shared and exchanged among different systems.

In order to store massive RDF data in document stores of NoSQL databases, RDF data should be represented with JSON format. As we know, RDF triples contain three components, which are subjects, predicates and objects and meanwhile the structure of JSON only contains two components, which are keys and values. It is clear that they are not consistent. So it is necessary to establish a mapping to transform RDF to JSON. Basically, we can identify several basic mappings from RDF to JSON as follows.

1. Triple-centered mapping
2. Subject-centered mapping
3. JSON-LD approach

In the triple-centered mapping, all triples in an RDF graph can be stored in a rooted document object. The value of this document object is an array, and each element of the array corresponds to a triple of RDF. In the subject-centered mapping, each subject of RDF data is treated as the “*Key*” of an JSON object, and the corresponding “*Value*” is some embedded objects (Alexander, 2008). The “*Key*” of these embedded objects is the predicate of the present subject. JSON-LD¹⁷, developed by W3C as a recommendation, is a data serialization and messaging format for Linked Data. It is primarily intended to be a way to use Linked Data in Web-based programming environments, to build interoperable Web services, and to store Linked Data in JSON-based storage engines. With JSON-LD, Linked Data which are represented by RDF can be easily stored in JSON-based document stores of NoSQL databases such as CouchDB. An JSON-LD document is an instance of RDF data model. JSON-LD can serialize a generic RDF data set after RDF data model is extended. Also a reverse serialization can be made with JSON-LD. API specifications in JSON-LD provide supports to obverse and reverse serializations between RDF and JSON-LD.

A simple example of JSON-LD from <http://json-ld.org/> is shown in Figure 5. This fragment contains information about a person: his name is John Lennon, he was born on 1940-10-09, and his spouse is described in the website of http://dbpedia.org/resource/Cynthia_Lennon. For terms like “name”, “born” and “spouse”, their semantics are defined with IRIs (International Resource Identifiers). In addition, @context is used to define the short-hand names that are used throughout a JSON-LD document, and @id is used to uniquely identify things that are being described in the document with IRIs or blank node identifiers.

Storing RDF Data in Graph Database

Compared to relational database model which uses flat relational table, graph databases adopt graph model with vertices, edges and property, and are hereby very suitable to handle the data with network structure (Angles and Gutierrez, 2008). Many techniques and algorithms proposed in the context of Graph Theory (GT) can be applied in graph databases. Neo4j is a popular graph database, in which the data model it uses to express its data is a graph, storing nodes and relationships that connect them, supporting user defined properties on both constructs. Basically, Neo4j can traverse the vertices and edges of graph at the same speed and the traversing speed is not influenced by the amount of data constituting the graph. So Neo4j can support scalable store of big graph well.

RDF data model can be regarded as a kind of graph model. It is advocated in (Angles and Gutiérrez, 2005) to store RDF data in graph databases. So, it is a natural way to store massive RDF data in Neo4j. But the standard graph-database model is different from the triple-based RDF model. In order to manage massive RDF data with Neo4j, two basic approaches can be adopted. First, Neo4j can be extended and provided with an interface of processing RDF data and at this point, RDF data are actually stored with native or relational stores. Second, massive RDF data can be directly stored in Neo4j. Following the second approach, DBpedia data are stored in Neo4j in the project of Dbpedia4neo¹⁸, and SPARQL (Simple Protocol and RDF Query Language) querying and some graph algorithms are then investigated.

As we know, SPARQL is a vendor-independent standard query language for the RDF triple data model (Prud'hommeaux and Seaborne, 2008), which is developed by W3C (World Wide Web Consortium). It should be noted that, although it is shown that graph databases can model RDF data in a natural way very well, some primary and useful querying operations for graph databases are not supported by current

Figure 5. A simple example of JSON-LD

```
{
  "@context": "http://json-ld.org/contexts/person.jsonld",
  "@id": "http://dbpedia.org/resource/John_Lennon",
  "name": "John Lennon",
  "born": "1940-10-09",
  "spouse": "http://dbpedia.org/resource/Cynthia_Lennon"
```

SPARQL query language. In this context, in order to manipulate RDF data, SPARQL query language should be extended to incorporate graph database query language primitives (Angles and Gutiérrez, 2005).

Storage of Special RDF Data in NoSQL Databases

With the wide application of RDF in concrete domains, some additional semantics specialized in the domains need to be accommodated into RDF data model. And this has resulted that special massive RDF data are emerging. Here we only illustrate several special types of massive RDF data and investigate their store in NoSQL databases.

Things in the real world may change as time goes on. Here we concentrate on temporal data modeling. As we know, big data have the well-known “3V” properties, which are *volume*, *velocity* and *variety*. Among them, velocity actually implies that big data are inherently temporal data. Traditionally, temporal data are managed in the relational databases. With the popularity of NoSQL for scalability and storage of big data, to manage big data that change over time, many research efforts have been invested in temporal big data management in NoSQL databases (Monger, Mata-Toledo and Gupta, 2012; Cuzzocrea, 2015). For the purpose of storing temporal big data, two major categories of NoSQL databases, which are column-oriented NoSQL databases (Hu and Dessoach, 2014 & 2015) and graph-oriented NoSQL databases (Castellort and Laurent, 2013), respectively. In addition, in (Brahmia, Brahmia, Grandi and Bouaziz, 2016), a framework called temporal JSON Schema (τ JSchema) is proposed to define a temporal JSON schema. On the basis, temporal JSON-based (document-oriented) NoSQL databases can be managed.

Temporal data have been dealt with in the context of NoSQL databases. But traditional RDF data model is unable to deal with temporal data. Data from real-world applications may be temporal. A temporal RDF model is proposed in (Gutiérrez, Hurtado and Vaisman, 2007; Hurtado and Vaisman, 2006), which contains temporal RDF triples. A temporal triple is shaped as $(s, p, o):[ts, te]$, where (s, p, o) is an RDF triple and $[ts, te]$ is a time interval. Here, $(s, p, o):[t_1, t_2]$ denotes such a fact that triple (s, p, o) is valid at time interval $[ts, te]$. In (Pugliese, Udreă and Subrahmanian, 2008), a concept of a normalized *t*RDF database is proposed, in which a temporal RDF database consists of a set of temporally annotated RDF triples with a form of $(subject, property:annotation, object)$. Then, we have three forms of temporal RDF triples: $(s, p:\{T\}, o)$, $(s, p:<n:T>, o)$ and $(s, p:[n:T], o)$, where $p:\{T\}$ means a *t*RDF triple is valid in a time points $\{T\}$, $p:<n:T>$ means a *t*RDF triple is valid at least at n distinct time points in T and $p:[n:T]$ means a *t*RDF triple is valid at most at n distinct time points in T . It is believed that massive RDF data management and massive temporal data management can work together and create a new issue of massive temporal RDF data management. How to deal with large scale temporal RDF triples on a cloud environment is receiving a recent attention (Ahn *et al.*, 2017).

In addition to temporal characteristics, things in the real world generally hold spatial characteristics, which may further change as time goes on. It means that we do need to manage spatial data and spatio-temporal data also. Some efforts have devoted to dealing with large spatio-temporal data with NoSQL databases (e.g., (Tear, 2014), (Orrù *et al.*, 2017) and (Detti *et al.*, 2017)). In the context of RDF data, spatial RDF data has been proposed and investigated in (Liagouris, Mamoulis, Bouros and Terrovitis, 2014; Shi, Wu and Mamoulis, 2016); both spatial and temporal (i.e., spatiotemporal) information are modeled with RDF in (Hoffart, Suchanek, Berberich and Weikum, 2013; Kyzirakos *et al.*, 2013; Wang, Zou and Zhao, 2014; Smeros and Koubarakis, 2016). Spatiotemporal data naturally have the properties of Big Data. To manage large spatiotemporal data in RDF, there is an increasing interest in storing large spatiotemporal RDF data with NoSQL databases.

The temporal/spatial/spatiotemporal RDF data can be viewed as a kind of annotated RDF (Udrea, Recupero and Subrahmanian, 2010). The annotations connected to RDF triples can be interpreted as various different meanings in the real world such as uncertainty, trust, provenance and so on. One of possible semantics for the annotations is uncertainty, which extensively exists in knowledge representation and reasoning. In this direction, two major categories of uncertain RDF data models can be identified, which are probabilistic RDF for objective uncertainty (e.g., (Lian and Chen, 2011), (Lian, Chen and Huang, 2015) and (Fang and Zhang, 2016)) and fuzzy RDF for subjective uncertainty (e.g., (Rubiera, Polo, Berrueta and Ghali, 2012) and (Ma, Li and Yan, 2018)), respectively. Here, let us look at the fuzzy RDF as the example of uncertain RDF.

Fuzzy RDF actually means fuzzy triples. Two levels of fuzziness can be identified in fuzzy triples, which are triple-level fuzziness and element-level fuzziness, respectively. The triple-level fuzzy RDF model is formally a 4-tuple (s, p, o, μ) , where μ is a numeric value between 0 and 1. As we know, a triple (s, p, o) contains three components (elements). So, the element-level fuzziness of triples can be further identified as the fuzziness in whole three elements and the fuzziness in part of three elements. In (Ma, Li and Yan, 2018), the fuzzy RDF triple is formally a fuzzy triple $(\mu_s/s, \mu_p/p, \mu_o/o)$, where s is a fuzzy subject and $\mu_s \in [0, 1]$ denotes the degree of subject to the universe of an RDF dataset, p is a fuzzy predicate and $\mu_p \in [0, 1]$ expresses the fuzzy degree to the property or relationship being described, o is a fuzzy object and $\mu_o \in [0, 1]$ represents the fuzzy degree of the property value. In (Ma and Yan, 2018), the fuzzy RDF triple is formally a fuzzy triple $(s, p, (o, \mu))$, which means that subject s has property p and p further has value o with membership degree μ . More importantly, the issue of formally mapping fuzzy RDF data to relational databases is investigated in (Ma and Yan, 2018). It is believed that storing large fuzzy RDF data in NoSQL databases is essential for massive fuzzy RDF data management, but there are no report so far on this topic.

SUMMARY AND FUTURE WORK

With the increasing amount of RDF data which is becoming available, efficient and scalable management of massive RDF data is of increasing important. NoSQL databases are designed for storing and processing Big Data datasets, and massive RDF data management merits the use of Big Data infrastructure because of the scalability and high performance of cloud data management. In this chapter, we provide an up-to-date overview of the current state of the art in massive RDF data stores in NoSQL databases. RDF data management is a very active area of research, and there are a lot of research efforts in this area. The chapter presents the survey from three main perspectives, which are key value stores of RDF data in NoSQL databases, document stores of RDF data in NoSQL databases and RDF data stores in graph databases. Note that this chapter only concentrates on massive RDF data stores in NoSQL databases, and does not discuss issues of indexing and querying massive RDF data in NoSQL databases.

RDF data management typically involves the scalable storage and efficient queries of RDF data. In addition, to better serve a given query, it is needed to index RDF data, which is especially true for massive RDF data. RDF data stores in NoSQL databases provide an infrastructure for massive RDF data management. Currently, some efforts are concentrating on massive RDF data querying based on cloud computing (e.g., Garcia and Wang, 2013; Husain *et al.*, 2010 & 2011; Kim, Ravindra and Anyanwu, 2013, Li *et al.*, 2013). It should be noted that RDF data management based on NoSQL databases has only recently been gaining momentum, and the research in this direction is in its infancy. There are many

research challenges and many interesting research opportunities for both the data management community and the Semantic Web community. Here we emphasize several major directions for future research.

1. First, following the success of NoSQL approaches for Big Data outside the RDF space in cloud environment, a major direction for research is the study and development of richer structural indexing techniques and related query processing strategies for RDF data in NoSQL databases.
2. Second, in addition to SELECT type of querying in SPARQL (Simple Protocol and RDF Query Language), a standard RDF querying language, how NoSQL databases can well support other types SPARQL querying such as CONSTRUCT, ASK and DESCRIBE should be investigated.
3. Finally, for massive RDF data management in the context of the diversity in RDF application domains (e.g., computing biology (Anguita *et al.*, 2013) and geological information systems (Garbis *et al.*, 2013)), novel RDF data store structures and querying strategies need to be developed on the basis of NoSQL databases.

ACKNOWLEDGMENT

The work was supported in part by the *National Natural Science Foundation of China* (61772269 and 61370075).

REFERENCES

- Abadi, D. J., Marcus, A., Madden, S., & Hollenbach, K. 2007, Scalable semantic Web data management using vertical partitioning, *Proceedings of the 33th International Conference on Very Large Data Bases*, 411-422.
- Abadi, D. J., Marcus, A., Madden, S., & Hollenbach, K. (2009). SW-Store: A vertically partitioned DBMS for Semantic Web data management. *The VLDB Journal*, 18(2), 385–406. doi:10.1007/00778-008-0125-y
- Ahn, J. H., Eom, J.-H., Nam, S., Zong, N., Im, D.-H., & Kim, H.-G. (2017). xStore: Federated temporal query processing for large scale RDF triples on a cloud environment. *Neurocomputing*, 256, 5–12. doi:10.1016/j.neucom.2016.03.116
- Alexander, K. (2008). *RDF in JSON: A specification for serialising RDF in JSON*. Retrieved from <http://www.semanticscripting.org/SFSW2008>
- Angles, R., & Gutiérrez, C. (2005). Querying RDF data from a graph database perspective. *Proceedings of the 2005 European Semantic Web Conference*, 346-360. 10.1007/11431053_24
- Angles, R., & Gutierrez, C. (2008). Survey of graph database models. *ACM Computing Surveys*, 40, 1:1-1:39.
- Anguita, A., Martin, L., Garcia-Remesal, M., & Maojo, V. (2013). RDFBuilder A tool to automatically build RDF-based interfaces for MAGE-OM microarray data sources. *Computer Methods and Programs in Biomedicine*, 3, 220-227.

- Bach, M., & Werner, A. 2014, Standardization of NoSQL database languages: Beyond databases, architectures, and structure. In *Proceedings 10th International Conference*. Ustron, Poland: Springer.
- Bishop, B., Kiryakov, A., Ognyanoff, D., Peikov, I., Tashev, Z., & Velkov, R. (2011). Owlrim: A family of scalable semantic repositories. *Semantic Web*, 2(1), 1–10.
- Bornea, M. A., Dolby, J., Kementsietsidis, A., Srinivas, K., Dantressangle, P., Udrea, O., & Bhattacharjee, B. (2013). Building an efficient RDF store over a relational database. *Proceedings of the 2013 ACM International Conference on Management of Data*, 121-132. 10.1145/2463676.2463718
- Brahmia, S., Brahmia, Z., Grandi, F., & Bouaziz, R. (2016). τ JSchema: A Framework for Managing Temporal JSON-Based NoSQL Databases. *Proceedings of the 27th International Conference on Database and Expert Systems Applications*, 167-181. 10.1007/978-3-319-44406-2_13
- Broekstra, J., Kampman, A., & van Harmelen, F. (2002). Sesame: A generic architecture for storing and querying RDF and RDF schema. *Proceedings of the 2002 International Semantic Web Conference*, 54-68. 10.1007/3-540-48005-6_7
- Castelltort, A., & Laurent, A. (2013). Representing history in graph-oriented NoSQL databases: A versioning system. *Proceedings of the Eighth International Conference on Digital Information Management*, 228-234. 10.1109/ICDIM.2013.6694022
- Chang, F., Dean, J., Ghemawat, S., Hsieh, W. C., Wallach, D. A., Burrows, M., ... Gruber, R. E. (2008). BigTable: A distributed storage system for structured data. *ACM Transactions on Computer Systems*, 26(2), 4:1-4:26.
- Cudre-Mauroux, P., Enchev, I., Fundatureanu, S., Groth, P., Haque, A., Harth, A., ... Wylot, M. (2013). NoSQL databases for RDF: An empirical evaluation. *Proceedings of the 12th International Semantic Web Conference*, 310-325. 10.1007/978-3-642-41338-4_20
- Cuzzocrea, A. (2015). Temporal aspects of big data management--state-of-the-art analysis and future research directions. *Proceedings of the 22nd International Symposium on Temporal Representation and Reasoning*, 180-185 10.1109/TIME.2015.31
- Dean, J., & Ghemawat, S. (2008). MapReduce: Simplified data processing on large clusters. *Communications of the ACM*, 51(1), 107–113. doi:10.1145/1327452.1327492
- DeCandia, G., Hastorun, D., Jampani, M., Kakulapati, G., Lakshman, A., Pilchin, A., ... Vogels, W. (2007). Dynamo: Amazon's highly available key-value store. *Proceedings of the 21st ACM Symposium on Operating Systems Principles*, 205-220.
- Detti, A. (2017). Application of Information Centric Networking to NoSQL databases: The spatio-temporal use case. *Proceedings of the 2017 IEEE International Symposium on Local and Metropolitan Area Networks*, 1-6. 10.1109/LANMAN.2017.7972131
- Duan, S., Kementsietsidis, A., Srinivas, K., & Udrea, O. (2011). Apples and oranges: a comparison of RDF benchmarks and real RDF datasets. *Proceedings of the 2011 ACM SIGMOD International Conference on Management of Data*, 145-156. 10.1145/1989323.1989340

- Erling, O., & Mikhailov, I. (2007). RDF support in the virtuoso DBMS. *Proceedings of the 1st Conference on Social Semantic Web*, 59-68.
- Erling, O., & Mikhailov, I. (2009). *Virtuoso: RDF support in a native RDBMS*. In *Semantic Web Information Management* (pp. 501–519). Springer-Verlag.
- Fang, H., & Zhang, X. W. (2016). pSPARQL: A Querying Language for Probabilistic RDF (Extended Abstract). *Proceedings of the ISWC 2016 Posters & Demonstrations Track*.
- Franke, C., Morin, S., Chebotko, A., Abraham, J., & Brazier, P. (2011). Distributed semantic web data management in HBase and MySQL Cluster. *Proceedings of the 2011 IEEE International Conference on Cloud Computing*, 105-112. 10.1109/CLOUD.2011.19
- Gamble, M., & Goble, C. (2011). Quality, Trust and Utility of Scientific Data on the Web: Toward a Joint model. *Proceedings of the 2011 International Conference on Web Science*, 15:1-15:8. 10.1145/2527031.2527048
- Garbis, G., Kyzirakos, K., & Koubarakis, M. (2013). Geographica: A benchmark for geospatial RDF stores. *Proceedings of the 12th International Semantic Web Conference*, 343-359.
- Garcia, T., & Wang, T. (2013). Analysis of Big Data technologies and method - Query large Web public RDF datasets on Amazon cloud using Hadoop and Open Source Parsers. *Proceedings of the 2013 IEEE International Conference on Semantic Computing*, 244-251. 10.1109/ICSC.2013.49
- Grolinger, K., Higashino, W. A., Tiwari, A., & Capretz, M. A. M. (2013). Data management in cloud environments: NoSQL and NewSQL data stores. *Journal of Cloud Computing: Advances, Systems and Applications*, 2(22).
- Gudivada, V. N., Rao, D., & Raghavan, V. V. (2014). NoSQL Systems for Big Data Management, *2014 IEEE World Congress on Services*, 190-197. 10.1109/SERVICES.2014.42
- Gutierrez, C., Hurtado, C. A., & Vaisman, A. (2007). Introducing time into RDF. *IEEE Transactions on Knowledge and Data Engineering*, 19(2), 207–218. doi:10.1109/TKDE.2007.34
- Güting, R. H. (1994). GraphDB: Modeling and querying graphs in databases. *Proceedings of 20th International Conference on Very Large Data Bases*, 297-308.
- Harris, S., Lamb, N., & Shadbolt, N. (2009). 4store: The design and implementation of a clustered RDF store. *Proceedings of the 5th International Workshop on Scalable Semantic Web Knowledge Base Systems*, 94-109.
- Hecht, R., & Jablonski, S. (2011). NoSQL evaluation: A use case oriented survey. In *Proceedings of the 2011 International Conference on Cloud and Service Computing*. Hong Kong, China: IEEE. 10.1109/CSC.2011.6138544
- Hoffart, J., Suchanek, F. M., Berberich, K., & Weikum, G. (2013). YAGO2: A spatially and temporally enhanced knowledge base from Wikipedia. *Artificial Intelligence*, 194, 28–61. doi:10.1016/j.artint.2012.06.001

Hu, Y., & Dessloch, S. (2014). Defining Temporal Operators for Column Oriented NoSQL Databases. *Proceedings of the 18th East European Conference on Advances in Databases and Information Systems*, 39-55. 10.1007/978-3-319-10933-6_4

Hu, Y., & Dessloch, S. (2015). Temporal Data Management and Processing with Column Oriented NoSQL Databases. *Journal of Database Management*, 26(3), 41–70. doi:10.4018/JDM.2015070103

Hua, X. Y., Wu, H., Li, Z., & Ren, S. P. (2014). Enhancing throughput of the Hadoop Distributed File System for interaction-intensive tasks. *Journal of Parallel and Distributed Computing*, 74(8), 2770–2779. doi:10.1016/j.jpdc.2014.03.010

Huang, J., Abadi, D. J., & Ren, K. (2011). Scalable SPARQL querying of large RDF graphs. *Proceedings of the VLDB Endowment International Conference on Very Large Data Bases*, 4(11), 1123–1134.

Hurtado, C., & Vaisman, A. (2006). Reasoning with temporal constraints in RDF. *Proceedings of the 2006 International Conference on Principles and Practice of Semantic Web Reasoning*, 164-178. 10.1007/11853107_12

Husain, M. F., Khan, L., Kantarcioglu, M., & Thuraisingham, B. M. (2010). Data intensive query processing for large RDF graphs using cloud computing tools. *Proceedings of the 2010 IEEE International Conference on Cloud Computing*, 1-10. 10.1109/CLOUD.2010.36

Husain, M. F., McGlothlin, J. P., Masud, M. M., Khan, L. R., & Thuraisingham, B. M. (2011). Heuristics-based query processing for large RDF graphs using cloud computing. *IEEE Transactions on Knowledge and Data Engineering*, 23(9), 1312–1327. doi:10.1109/TKDE.2011.103

Kaoudi, Z., & Manolescu, I. (2015). RDF in the clouds: A survey. *The VLDB Journal*, 24(1), 67–91. doi:10.1007/00778-014-0364-z

Khadilkar, V., Kantarcioglu, M., Thuraisingham, B. M., & Castagna, P. (2012). Jena-HBase: A distributed, scalable and efficient RDF triple store. *Proceedings of the 2012 International Semantic Web Conference*.

Kim, H. S., Ravindra, P., & Anyanwu, K. (2013). Optimizing RDF(S) queries on cloud platforms, *Proceedings of the 2013 International World Wide Web Conference*, 261-264. 10.1145/2487788.2487917

Kim, S. W. (2006). Hybrid storage scheme for RDF data management in Semantic Web. *Journal of Digital Information Management*, 4(1), 32–36.

Kyzirakos, K. (2013). The Spatiotemporal RDF Store Strabon. *Proceedings of the 13th International Symposium on Advances in Spatial and Temporal Databases*, 496-500. 10.1007/978-3-642-40235-7_35

Lakshman, A., & Malik, P. (2010). Cassandra: A decentralized structured storage system. *ACM SIGOPS Operating System Review*, 44(2), 35–40. doi:10.1145/1773912.1773922

Laney, D. (2001). *3D data management: Controlling data volume, velocity and variety*. Meta Group, Gartner. Retrieved from <http://blogs.gartner.com/doug-laney/files/2012/01/ad949-3D-Data-Management-Controlling-Data-Volume-Velocity-and-Variety.pdf>

Li, R., Yang, D., Hu, H. B., Xie, J., & Fu, L. (2013). Scalable RDF graph querying using cloud computing. *Journal of Web Engineering*, 12(1 & 2), 159–180.

- Liagouris, J., Mamoulis, N., Bouros, P., & Terrovitis, M. (2014). An Effective Encoding Scheme for Spatial RDF Data. *Proceedings of the VLDB Endowment International Conference on Very Large Data Bases*, 7(12), 1271–1282. doi:10.14778/2732977.2733000
- Lian, X., & Chen, L. (2011). Efficient query answering in probabilistic RDF graphs. *Proceedings of the ACM SIGMOD International Conference on Management of Data*, 157–168. 10.1145/1989323.1989341
- Lian, X., Chen, L., & Huang, Z. (2015). Keyword Search Over Probabilistic RDF Graphs. *IEEE Transactions on Knowledge and Data Engineering*, 27(5), 1246–1260. doi:10.1109/TKDE.2014.2365791
- Luo, Y., Picalausa, F., Fletcher, G. H. L., Hidders, J., & Vansummeren, S. (2012). *Storing and indexing massive RDF datasets*. In *Semantic Search over the Web* (pp. 31–60). Springer-Verlag Berlin Heidelberg. doi:10.1007/978-3-642-25008-8_2
- Ma, Z., Capretz, M. A. M., & Yan, L. (2016). Storing massive Resource Description Framework (RDF) data: A survey. *The Knowledge Engineering Review*, 31(4), 391–413. doi:10.1017/S0269888916000217
- Ma, Z., Li, G., & Yan, L. (2018). Fuzzy data modeling and algebraic operations in RDF. *Fuzzy Sets and Systems*, 351, 41–63. doi:10.1016/j.fss.2017.11.013
- Ma, Z., & Yan, L. (2018). Modeling fuzzy data with RDF and fuzzy relational database models. *International Journal of Intelligent Systems*, 33(7), 1534–1554. doi:10.1002/int.21996
- Manola, F., & Miller, E. (2004). *RDF Primer*. W3C Recommendation. Retrieved from <http://www.w3.org/TR/2004/REC-rdf-primer-20040210/>
- McBride, B. (2002). Jena: A Semantic Web toolkit. *IEEE Internet Computing*, 6(6), 55–59. doi:10.1109/MIC.2002.1067737
- Monger, M. D., Mata-Toledo, R. A., & Gupta, P. (2012). Temporal Data Management in NoSQL Databases. *Romanian Economic Business Review*, 6(2), 237–243.
- Moniruzzaman, A. B. M., & Hossain, S. A. (2013). NoSQL Database: New Era of Databases for Big data Analytics - Classification, Characteristics and Comparison. *International Journal of Database Theory and Application*, 6(4), 1–14.
- Neumann, T., & Weikum, G. (2008). RDF-3X: A RISC-style engine for RDF. *Proceedings of the VLDB Endowment International Conference on Very Large Data Bases*, 1(1), 647–659. doi:10.14778/1453856.1453927
- Neumann, T., & Weikum, G. (2010). The RDF-3X engine for scalable management of RDF data. *The VLDB Journal*, 19(1), 91–113. doi:10.1007/00778-009-0165-y
- Nicoara, D., Kamali, S., Daudjee, K., & Chen, L. (2015). Hermes: Dynamic partitioning for distributed social network graph databases. *Proceedings of the 18th International Conference on Extending Database Technology*, 25–36.
- Orrù, M. (2017). Demonstration of OpenGeoBase: The ICN NoSQL spatio-temporal database. *Proceedings of the 2017 IEEE International Symposium on Local and Metropolitan Area Networks*, 1–2. 10.1109/LANMAN.2017.7972184

- Papailiou, N., Konstantinou, I., Tsoumakos, D., Karras, P., & Koziris, N. (2013). H2RDF+: High-performance distributed joins over large-scale RDF graphs. *Proceedings of the 2013 IEEE International Conference on Big Data*, 255-263. 10.1109/BigData.2013.6691582
- Papailiou, N., Konstantinou, I., Tsoumakos, D., & Koziris, N. (2012). H2RDF: Adaptive query processing on RDF data in the cloud. *Proceedings of the 21st World Wide Web Conference*, 397-400. 10.1145/2187980.2188058
- Pokorny, J. (2011). NoSQL databases: A step to database scalability in web environment. *Proceedings of the 2011 International Conference on Information Integration and Web-based Applications and Services*, 278-283. 10.1145/2095536.2095583
- Pokorny, J. (2011). NoSQL Databases: A step to database scalability in Web environment. *International Journal of Web Information Systems*, 9(1), 69-82. doi:10.1108/17440081311316398
- Prud'hommeaux, E., & Seaborne, A. (2008). *SPARQL Query Language for RDF*. W3C Recommendation. Retrieved from <http://www.w3.org/TR/2008/REC-rdf-sparql-query-20080115/>
- Pugliese, A., Udreă, O., & Subrahmanian, V. S. (2008). Scaling RDF with time. *Proceedings of the 2008 International Conference on World Wide Web*, 605-614.
- Rubiera, E., Polo, L., Berrueta, D., & Ghali, A. E. (2012). TELIX: An RDF-Based Model for Linguistic Annotation. *Proceedings of the 9th Extended Semantic Web Conference*, 195-209. 10.1007/978-3-642-30284-8_20
- Sakr, S., & Al-Naymat, G. (2009). Relational processing of RDF queries: A survey. *SIGMOD Record*, 38(4), 23-28. doi:10.1145/1815948.1815953
- Shi, J. M., Wu, D. M., & Mamoulis, N. (2016). Top-k Relevant Semantic Place Retrieval on Spatial RDF Data. *Proceedings of the 2016 International Conference on Management of Data*, 1977-1990. 10.1145/2882903.2882941
- Sintek, M., & Kiesel, M. (2006). RDFBroker: A signature-based high-performance RDF store. *Proceedings of the 3rd European Semantic Web Conference*, 363-377.
- Smeros, P., & Koubarakis, M. (2016). Discovering Spatial and Temporal Links among RDF Data, *Proceedings of the 2016 Workshop on Linked Data on the Web*.
- Snow, D. (2012). *Dwaine Snow's Thoughts on Databases and Data Management*. Retrieved from <http://dsnowondb2.blogspot.cz/2012/07/adding-4th-v-to-big-data-veracity.html>
- Sperka, S., & Smrz, P. (2012). Towards adaptive and semantic database model for RDF data stores. *Proceedings of the Sixth International Conference on Complex, Intelligent, and Software Intensive Systems*, 810-815.
- Stuart, J., & Barker, A. (2013). *Undefined By Data: A Survey of Big Data Definitions*. CoRR, abs/1309.5821
- Sun, J. L., & Jin, Q. (2010). Scalable RDF store based on HBase and MapReduce. *Proceedings of the 3rd International Conference Advanced Computer Theory and Engineering*, V1-633-V1-636.

- Tauro, C., Aravindh, S., & Shreeharsha, A. B. (2012). Comparative Study of the New Generation, Agile, Scalable, High Performance NOSQL Databases. *International Journal of Computer Applications*, 48(20).
- Tear, A. (2014). SQL or NoSQL? Contrasting Approaches to the Storage, Manipulation and Analysis of Spatio-temporal Online Social Network Data. *Proceedings of the 14th International Conference on Computational Science and Its Applications*, 221-236.
- Udrea, O., Recupero, D. R., & Subrahmanian, V. S. (2010). Annotated RDF. *ACM Transactions on Computational Logic*, 11(2), 10:1-10:41.
- Wang, D., Zou, L., & Zhao, D. Y. (2014). gst-Store: An Engine for Large RDF Graph Integrating Spatiotemporal Information. *Proceedings of the 17th International Conference on Extending Database Technology*, 652-655.
- Wang, Y., Du, X. Y., Lu, J. H., & Wang, X. F. (2010). FlexTable: Using a dynamic relation model to store RDF data. *Proceedings of the 15th International Conference on Database Systems for Advanced Applications*, 580-594. 10.1007/978-3-642-12026-8_44
- Weiss, C., Karras, P., & Bernstein, A. (2008). Hexastore: Sextuple indexing for semantic web data management. *Proceedings of the VLDB Endowment International Conference on Very Large Data Bases*, 1(1), 1008–1019. doi:10.14778/1453856.1453965
- Wilkinson, K., Sayers, C., Kuno, H. A., & Reynolds, D. (2003). Efficient RDF storage and retrieval in Jena2. *Semantic Web and Databases Workshop*, 131-150.

ADDITIONAL READING

- Aluc, G., Özsu, M. T., & Daudjee, K. (2014). Workload matters: Why RDF databases need a new design. *Proceedings of the VLDB Endowment International Conference on Very Large Data Bases*, 7(10), 837–840. doi:10.14778/2732951.2732957
- Cardoso, J. (2007). The Semantic Web vision: Where are We? *IEEE Intelligent Systems*, 22(5), 84–88. doi:10.1109/MIS.2007.4338499
- Cattell, R. (2011). Scalable SQL and NoSQL data stores. *SIGMOD Record*, 39(4), 12–27. doi:10.1145/1978915.1978919
- Kelly, J. (2012, August). Accumulo, 2012, Why the world needs another NoSQL database. *Big Data*.
- Schindler, J. (2012). I/O characteristics of NoSQL databases. *Proceedings of the VLDB Endowment International Conference on Very Large Data Bases*, 5(12), 2020–2021. doi:10.14778/2367502.2367565
- Shadbolt, N., Berners-Lee, T., & Hall, W. (2006). The Semantic Web revisited. *IEEE Intelligent Systems*, 21(3), 96–101. doi:10.1109/MIS.2006.62
- Stonebraker, M. (2010). SQL databases v. NoSQL databases. *Communications of the ACM*, 53(4), 10–11. doi:10.1145/1721654.1721659

KEY TERMS AND DEFINITIONS

ACID: ACID means four properties, which are (A)tomcity, (C)onsistency, (I)solation and (D)urability. ACID is the type of transaction processing done by relational database management system (RDBMS).

Big Data: Big data is a broad term for data sets so large or complex that traditional data processing applications are inadequate. There is not a common definition of big data, and big data are generally characterized by some properties such as volume, velocity, variety, and so on.

CAP: CAP theorem means three properties, which are consistency, availability, and partition tolerance.

JSON: JSON (JavaScript Object Notation) is a binary and typed data model which is applied to represent data like list, map, date, Boolean as well as different precision numbers.

NoSQL Databases: NoSQL means “not only SQL” or “no SQL at all.” Being a new type of non-relational databases, NoSQL databases are developed for efficient and scalable management of big data.

RDF: Resource description framework (RDF) is a W3C (World Wide Web Consortium) recommendation which provides a generic mechanism for representing information about resources on the web.

SPARQL: SPARQL (Simple Protocol and RDF Query Language) is an RDF query language which is a W3C recommendation. SPARQL contains capabilities for querying required and optional graph patterns along with their conjunctions and disjunctions.

ENDNOTES

- ¹ <http://www.data.gov/>
- ² <http://www.data.gov.uk/>
- ³ <http://data.nytimes.com/>
- ⁴ http://www.bbc.co.uk/blogs/bbcinternet/2010/07/bbc_world_cup_2010_dynamic_sem.html
- ⁵ <http://www.chiefmartec.com/2009/12/best-buy-jump-starts-data-web-marketing.html>
- ⁶ <http://www.oracle.com/us/products/database/options/spatial/overview/index.html>
- ⁷ <http://www.cs.ox.ac.uk/isg/tools/RDFox/>
- ⁸ <http://nosql-databases.org>
- ⁹ <http://redis.io/>
- ¹⁰ <http://hbase.apache.org/>
- ¹¹ <http://couchdb.apache.org/>
- ¹² <http://www.couchbase.com/couchbase-server/overview>
- ¹³ <http://www.mongodb.org/>
- ¹⁴ <http://neo4j.com/>
- ¹⁵ <http://thinkaurelius.github.io/titan/>
- ¹⁶ <http://hypertable.org>
- ¹⁷ <https://github.com/claudiomartella/dbpedia4neo>
- ¹⁸ <http://www.w3.org/TR/json-ld/>