*The Knowledge Engineering Review*, Vol. 31:4, 391–413. © Cambridge University Press, 2016
doi:10.1017/S0269888916000217

# Storing massive Resource Description Framework (RDF) data: a survey

ZONGMIN MA[1,2], MIRIAM A. M. CAPRETZ[3] and LI YAN[1,2]

[1]*College of Computer Science & Technology, Nanjing University of Aeronautics and Astronautics, Nanjing 211106, China; e-mail: zongmin_ma@yahoo.com*;
[2]*Collaborative Innovation Center of Novel Software Technology and Industrialization, Nanjing 210023, China; e-mail: yanli@nuaa.edu.cn*;
[3]*Department of Electrical and Computer Engineering, Western University, London, Canada, ON N6A 5B9; e-mail: mcapretz@uwo.ca*

**Abstract**

The Resource Description Framework (RDF) is a flexible model for representing information about resources on the Web. As a W3C (World Wide Web Consortium) Recommendation, RDF has rapidly gained popularity. With the widespread acceptance of RDF on the Web and in the enter prise, a huge amount of RDF data is being proliferated and becoming available. Efficient and scalable management of RDF data is therefore of increasing importance. RDF data management has attracted attention in the database and Semantic Web communities. Much work has been devoted to proposing different solutions to store RDF data efficiently. This paper focusses on using relational databases and NoSQL (for 'not only SQL (Structured Query Language)') databases to store massive RDF data. A full up-to-date overview of the current state of the art in RDF data storage is provided in the paper.

## 1 Introduction

Recent years have witnessed a tremendous increase in the amount of data available on the Web (Hassanzadeh *et al*., 2012). At the same time, Web 2.0 applications have introduced new forms of data and have radically changed the nature of the modern Web. In these applications, the Web has been transformed from a publish-only environment into a vibrant forum for information exchange (Hassanzadeh *et al*., 2012). The main purpose of the Semantic Web, proposed by W3C founder Tim Berners-Lee in his description of the future of the Web (Berners-Lee *et al*., 2001), is to provide a common framework for data sharing across applications, enterprises, and communities. By giving data semantic meaning (through metadata), this framework enables machines to consume, understand, and reason about the structure and purpose of data.

The core of the Semantic Web is built on the Resource Description Framework (RDF) data model (Manola & Miller, 2004). RDF provides a flexible and concise model for representing metadata of resources on the Web. RDF can represent structured as well as unstructured data and is quickly becoming the *de facto* standard for representation and exchange of information[1] (Duan *et al*., 2011). Nowadays, the RDF data model is finding increasing use in a wide range of Web data-management scenarios. Governments (e.g. from the United States[2] and United Kingdom[3]) and large companies and organizations

[1] http://www.w3.org/RDF
[2] http://www.data.gov/
[3] http://www.data.gov.uk/

https://doi.org/10.1017/S0269888916000217 Published online by Cambridge University Press

(e.g. New York Times[4], BBC[5], and Best Buy[6]) have started using RDF as a business data model and representation format, either for semantic data integration, search-engine optimization, and better product search, or to represent data from information extraction. Furthermore, in the Linked Open Data (LOD) cloud (Bizer *et al*., 2009), Web data from a diverse set of domains like Wikipedia, films, geographic locations, and scientific data are linked to provide one large RDF data cloud. With the increasing amount of RDF data which is becoming available, efficient and scalable management of massive RDF data is of crucial importance. Massive RDF data simply mean the number of triples is very huge (e.g. from million triples to billion triples and up).

As a new data model, the RDF data-representation format largely determines how to store and index RDF data and furthermore influences how to query RDF data. Management of RDF data (especially massive RDF data) typically involves two primary technical challenges: scalable storage and efficient queries. In addition, to serve a given query more effectively, it is necessary to index RDF data. These three issues are actually closely related. Indexing of RDF data is enabled based on RDF storage, and efficient querying of RDF data is supported by the indexing structure. Among these three issues, RDF data storage provides the infrastructure for RDF data management. Currently, with the RDF format gaining widespread acceptance, much work is being done in RDF data management, and a number of research efforts have been undertaken to address these issues. Some RDF data-management systems have started to emerge such as Sesame (Broekstra *et al*., 2002), Jena-TDB (Wilkinson *et al*., 2003), Virtuoso (Erling & Mikhailov, 2007, 2009), 4store (Harris *et al*., 2009), BigOWLIM (Bishop *et al*., 2011), SPARQLcity/SPARQLverse[7], MarkLogic[8], Clark & Parsia/Stardog[9], and Oracle Spatial and Graph with Oracle Database 12c[10]. BigOWLIM was renamed to OWLIM-SE and later on to GraphDB. In addition, some research prototypes have been developed (e.g. RDF-3X (Neumann & Weikum, 2008, 2010), SW-Store (Abadi *et al*., 2007, 2009), and RDFox[11]).

Note that RDF data management has been studied in a variety of contexts. This variety is actually reflected in a richness of perspectives and approaches to storage of RDF data sets, which is typically driven by particular classes of query patterns and inspired by techniques developed in various research communities (Luo *et al*., 2012). As a result, few efforts have been made to review the state of the art in RDF data management. The topic of mapping SQL databases to RDF is reviewed by Sequeda *et al*. (2011). Viewed from three basic perspectives, a survey of the state of the art in RDF storage and indexing is presented by Luo *et al*. (2012). These three basic perspectives are the relational perspective, the entity perspective, and the graph-based perspective. Based on the relational perspective, Sakr and Al-Naymat (2009) gave an overview of relational techniques for storing and querying RDF data.

Focussing on big data processing, Cudre-Mauroux *et al*. (2013) presented an empirical evaluation by comparing several NoSQL (for 'not only SQL') stores for RDF processing when running two standard RDF benchmarks, the Berlin SPARQL Benchmark (BSBM) (Bizer & Schultz, 2009) and the DBpedia SPARQL Benchmark (DBPSB) (Morsey *et al*., 2011) on a cloud infrastructure. Benchmarking has been a core topic of RDF data-management research, and several RDF benchmarks have been developed in addition to the BSBM and the DBPSB benchmarks. SPARQL Performance Benchmark ($SP^2$Bench) (Schmidt *et al*., 2008, 2009) was used to compare experimentally several RDF storage strategies. A benchmark creation methodology was proposed for SPARQL Protocol and RDF Query Language (SPARQL) by Morsey *et al*. (2012), which was applied to the DBpedia knowledge base. Then the

[4] http://data.nytimes.com/
[5] http://www.bbc.co.uk/blogs/bbcinternet/2010/07/bbc_world_cup2010_dynamic_sem.html
[6] http://www.chiefmartec.com/2009/12/best-buy-jump-starts-data-web-marketing.html
[7] http://sparqlcity.com/
[8] http://www.marklogic.com/
[9] http://clarkparsia.com/
[10] http://www.oracle.com/us/products/database/options/spatial/overview/index.html
[11] http://www.cs.ox.ac.uk/isg/tools/RDFox/

https://doi.org/10.1017/S0269888916000217 Published online by Cambridge University Press

generated benchmark was used to compare several popular triple-store implementations, including Virtuoso, Sesame, Jena-TDB, and BigOWLIM. Data sets for RDF data management were classified into two categories by Duan *et al*. (2011): RDF benchmark data sets and real RDF data sets. Among these, the RDF benchmark data sets included the BSBM data set, the SP$^2$Bench data set, the Lehigh University Benchmark (LUBM) data set (Guo *et al*., 2005) and TPC-H[12], and the real RDF data sets included DBpedia, UniProt (Apweiler *et al*., 2004), YAGO (Suchanek *et al*., 2008), the Barton library data set[13], WordNet[14], and the Linked Sensor data set (Patni *et al*., 2010). After comparing data generated with existing RDF benchmarks and data found in widely used real RDF data sets, Duan *et al*. concluded that existing benchmark data have little in common with real data. Real data mean data taken from real data sets, which cover the whole structuredness spectrum, while benchmark data sets are very limited in their structuredness and are mostly relational like. Therefore, any conclusions drawn from existing benchmark tests might not actually translate to expected behaviour in real settings.

This paper focusses on RDF data storage and presents a full up-to-date overview of the current state of the art in RDF data storage. The various approaches are classified according to their storage strategy, including RDF data stores in traditional databases and RDF data stores in NoSQL databases. Unlike other RDF data-store surveys, which do not provide NoSQL RDF storage and benchmarks to be included in their studies (e.g. Sakr & Al-Naymat, 2009; Luo *et al*., 2012), this survey presents a review of several major NoSQL databases and benchmarks for RDF data management. The objective of this paper is twofold. The first is to provide a generic overview of the approaches that have been proposed to store RDF data. The second is to identify and analyze research opportunities in the area of RDF data management. Note that, due to the large number of RDF data-management solutions, it was not feasible to include all of them in this paper. First, this paper only concentrates on RDF data stores, and does not discuss in depth RDF indexing and querying although they are closely relevant to RDF data stores. Second, the paper does not include early RDF native stores, which use customized binary RDF data representation and are built directly on the file system. Finally, the paper does not cover in depth all possible distinctions between RDF data-management systems, especially the RDF data-management systems developed by commercial vendors.

The rest of this paper is organized as follows. Section 2 presents preliminaries of the RDF data model. It also introduces the main approaches for storing RDF data. Section 3 provides details of the different techniques in several alternative relational approaches. Section 4 provides details of different techniques in several NoSQL-based approaches. Section 5 presents several representative benchmarks for RDF data stores. Section 6 concludes the paper and provides some suggestions for possible research directions.

## 2 Resource Description Framework data model and Resource Description Framework data stores

The RDF is a W3C Recommendation that has rapidly gained popularity. RDF provides a means of expressing and exchanging semantic metadata (i.e. data that specify semantic information about data). By representing and processing metadata about information sources, RDF defines a model for describing relationships among resources in terms of uniquely identified attributes and values.

In the RDF data model, the universe is modelled as a set of resources, where a resource is anything that 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.

The abstract syntax of RDF model is a set of triples. Formally, an RDF triple is defined as $(s; p; o) \in (I \cup B) \times I \times (I \cup B \cup L)$, where $I$, $B$, and $L$ are infinite sets of *IRI*s, blank nodes, and RDF literals, respectively. In a triple $(s, p, o)$, $s$ is called the subject, $p$ the predicate (or property), and $o$ the object. The interpretation of a triple statement is that subject $s$ has property $p$ with value $o$. Note that any object in one

[12] The TPC Benchmark H, `http://www.tpc.org/tpch`
[13] The MIT Barton Library data set, `http://simile.mit.edu/rdf-test-data/`
[14] WordNet: a lexical database for English, `http://www.w3.org/2006/03/wn/wn20/`

https://doi.org/10.1017/S0269888916000217 Published online by Cambridge University Press

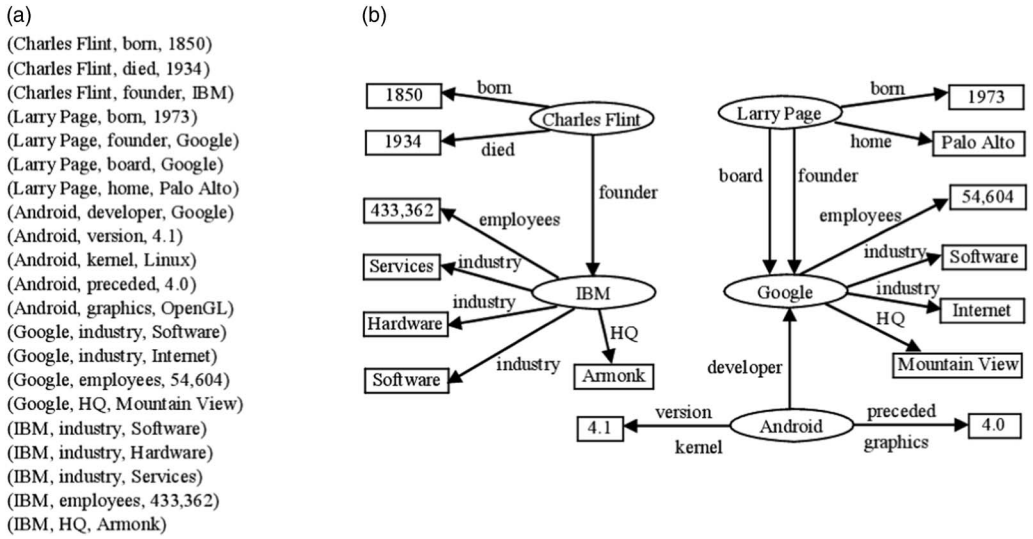



**Figure 1** Sample Resource Description Framework (RDF) triples and graph view. (a) Sample DBpedia data (Bornea *et al*., 2013). (b) Sample RDF graph

triple, say $o_i$ in ($s_i$, $p_i$, $o_i$), can play the role of a subject in another triple, say ($o_i$, $p_j$, $o_j$). Therefore, RDF data is a directed, labelled graph data format for representing Web resources (Huang *et al*., 2011).

Figure 1 shows a sample of DBpedia data from Bornea *et al*. (2013) and presents the corresponding RDF graph. In this sample, there are 21 triples and 13 predicates.

To manage RDF data, RDF data must be stored. Two different levels of RDF data storage can be distinguished: logical storage and physical storage. This paper mainly focusses on logical storage of RDF data. Few classifications of RDF data-storage approaches have been reported in the literature. Proposals for RDF data storage (Sakr & Al-Naymat, 2009; Bornea *et al*., 2013) were classified into two major categories: *native stores* and *relational stores*. Native stores use customized binary RDF data representation and are built directly on the file system. Relational stores distribute RDF data to appropriate relational databases. Viewed from three basic perspectives (i.e. the relational, entity, and graph-based perspectives), proposals for RDF data storage were classified into three major categories in Luo *et al*. (2012): *relational stores*, *entity stores*, and *graph-based stores*. In relational stores, an RDF graph is just a particular type of relational data, and then all RDF triples are stored in a single relational table or the triple predicate values are interpreted as column names in a collection of relation schemas. In entity stores, an RDF graph is treated as a collection of entity descriptions. In graph-based stores, the RDF data model can be seen as essentially a graph-based data model. Note that the label of the edge in an RDF graph is an URI, which may or may not appear also as a node and have description in the graph. This fact does not make RDF less graph-based model. In fact, Cypher-based and Blueprints-based graph databases use *Property Graphs*, which go even further apart from the 'standard graph' structures and require even more meta-modelling.

Currently, to process large-scale RDF data, NoSQL databases are used and a number of RDF data-management and data-analysis problems (e.g. large-scale caching of LOD and entity name servers) merit the use of big-data infrastructure (Cudre-Mauroux *et al*., 2013). Another important category for NoSQL RDF storage is graph databases.

This paper investigates two types of RDF data stores: traditional database stores and NoSQL database stores. Figure 2 illustrates this classification for RDF data stores.

Given the large number of RDF data-management solutions, there is a richness of perspectives and approaches to storage of RDF data sets. Viewed from particular classes of query patterns, for example, RDF data stores can be classified relying on query language primitives supported by the underlying infrastructure. Considering that the major focus of the paper is on database-based RDF data stores, the classification given in Figure 2 is based on database models which are used to store RDF data.

https://doi.org/10.1017/S0269888916000217 Published online by Cambridge University Press

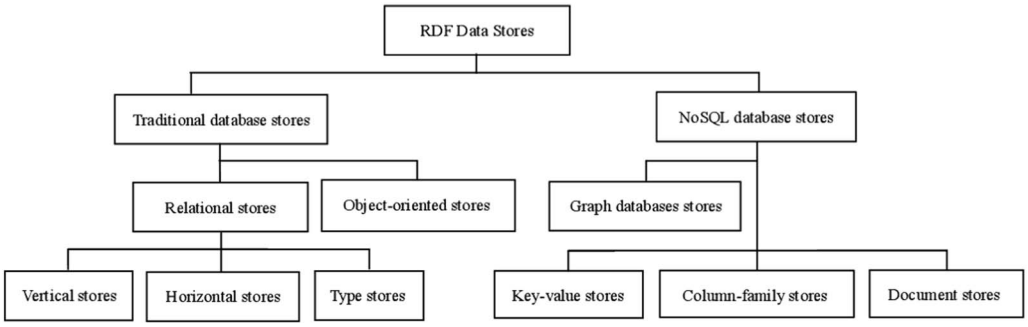



**Figure 2** Classification of Resource Description Framework (RDF) data stores

Traditionally, databases are classified into relational databases and object-oriented databases. In addition, NoSQL databases have only recently emerged as a commonly used infrastructure for handling big data. So, two top categories of RDF data stores in Figure 2 are *traditional database stores* and *NoSQL database stores*, respectively. For the traditional database stores, corresponding to two kinds of traditional database models, two categories of RDF data stores in the traditional database stores are *relational stores* and *object-oriented stores*, which apply relational database and object-oriented database models, respectively. Depending on concrete data models adopted, the NoSQL database stores are categorized into *key-value stores*, *column-family stores*, *document stores*, and *graph databases*. Note that in the relational stores of RDF data, several different relational schemas can be designed, depending on how to distribute RDF triples to an appropriate relational schema. This results in three major categories of RDF relational stores, which are *vertical stores*, *horizontal stores*, and *type stores*. They are formally illustrated in the following section.

Recent research has devoted considerable efforts to the study of managing massive RDF graphs in distributed environments. From this viewpoint, current RDF data stores can be classified into *centralized RDF stores* and *distributed RDF stores* (Papailiou *et al*., 2013). Centralized RDF stores are single-machine solutions with limited scalability. In contrast to centralized RDF stores, distributed RDF stores hash partition triples across multiple machines and parallelize query processing. Typically, NoSQL database stores are distributed RDF stores.

## 3 Resource Description Framework stores in traditional databases

A number of attempts have been made to use traditional databases to store RDF data, and various storage schemes for RDF data have been proposed. Some ideas and techniques developed earlier for object-oriented databases, for example, have already been adapted to the RDF setting. RDF data were stored in an object-oriented database by mapping both triples and resources to objects in Bonstrom *et al*. (2003). An object-oriented database model was proposed for storage of RDF documents (Chao, 2007a, 2007b), but the RDF documents were encoded in XML (eXtensible Markup Language).

Relational database management systems (RDBMSs) are currently the most widely used databases. It has been shown that RDBMSs are very efficient, scalable, and successful in hosting various types of data, including some new types of data such as XML data, temporal/spatial data, media data, and complex objects. RDBMSs derive much of their performance from sophisticated optimizer components that make use of physical properties specific to the relational model, such as sortedness, proper join ordering, and powerful indexing mechanisms.

Basically, three approaches can be identified for storing RDF data in relational databases (Sakr & Al-Naymat, 2009; Luo *et al*., 2012). The first is called *vertical stores* (e.g. Broekstra *et al*., 2002; Harris & Gibbins, 2003; Harris & Shadbolt, 2005; Neumann & Weikum, 2008, 2010; Weiss *et al*., 2008), in which all RDF triples are directly stored in a single relational table over relational schema (*subject*, *predicate*, *object*). As a result, each RDF triple becomes a tuple of the relational database. Vertical stores are also known as *triple stores*. The second approach for storing RDF data is called *horizontal stores* (e.g. Abadi

https://doi.org/10.1017/S0269888916000217 Published online by Cambridge University Press

| Subject | Predicate | Object |
|---|---|---|
| Charles Flint | born | 1850 |
| Charles Flint | died | 1934 |
| Charles Flint | founder | IBM |
| Larry Page | born | 1973 |
| Larry Page | founder | Google |
| Larry Page | board | Google |
| Larry Page | home | Palo Alto |
| Android | developer | Google |
| Android | version | 4.1 |
| Android | kernel | Linux |
| Android | preceded | 4.0 |
| Android | graphics | OpenGL |
| Google | industry | Software |
| Google | industry | Internet |
| Google | employees | 54,604 |
| Google | HQ | Mountain View |
| IBM | industry | Software |
| IBM | industry | Hardware |
| IBM | industry | Services |
| IBM | employees | 433,362 |
| IBM | HQ | Armonk |

**Figure 3** Vertical stores of Resource Description Framework data

*et al*., 2007, 2009), in which a subject–object relation is directly represented for each predicate of RDF triples and a predicate is a column name. Depending on whether a relational table contains only one predicate or all predicates of RDF triples, two kinds of horizontal stores can be further identified. If a relational table contains only one predicate, a set of binary relational tables results, and each relational table corresponds to a predicate. It is apparent that horizontal stores are actually predicate-oriented stores (Bornea *et al*., 2013). The third approach for storing RDF data is called *property stores* (e.g. Matono *et al*., 2005; Sintek & Kiesel, 2006; Levandoski & Mokbel, 2009), in which one relational table is created for each RDF data type and a relational table contains the properties as *n*-ary table columns for the same subject. Actually, property stores are type-oriented stores (Bornea *et al*., 2013). In the following discussion, property stores are referred to as *type stores*.

For the RDF triples in Figure 1(a), Figures 3, 4, and 5 show the relational representation of vertical stores, horizontal stores, and type stores, respectively. The relational table in Figure 3 contains 21 triples as table tuples. In Figure 4(a), one relational table is used, which contains 13 predicates of RDF triples as table columns. In Figure 4(b), 13 relational tables are used for 13 predicates of RDF triples, and each relational table contains one predicate as a table column. In Figure 5, three relational tables are used for three data types of people, companies, and operation systems, respectively, and each relational table contains all predicates of the RDF triples with the same type as the table columns.

Among the three approaches for storing RDF data in relational databases, vertical stores use a fixed relational schema, and new triples can be inserted without considering RDF data types. Therefore, vertical stores can handle dynamic schema of RDF data. However, vertical stores generally involve a number of self-join operations for querying, and therefore efficient querying requires specialized techniques. To overcome the problem of self-joins in vertical stores, horizontal stores using a single relational table are proposed. However, it commonly occurs that in the single relational table containing all predicates as columns, a subject occurs only with certain predicates, which leads to a sparse relational table with many null values. In addition, a subject may have multiple objects for the same predicate (e.g. *IBM* has objects *Software*, *Hardware*, and *Services* for the same predicate *industry*). Such a predicate is called a multi-valued predicate. As a result, the relational table in a horizontal store contains multi-valued attributes. Finally, when new triples are inserted, new predicates result in changes to the relational schema, and dynamic schema of RDF data cannot be handled. To solve the problem of null values as well as that of multi-valued attributes, horizontal stores using a set of relational tables are proposed, where each predicate corresponds to a relational table. However, horizontal stores using a set of relational tables generally involve many join operations for querying. In addition, when new triples are inserted, new predicates result in new relational tables, and dynamic schema of RDF data cannot be handled. A vertical store in (*p*, *s*, *o*) shape (it is shown as (*s*, *p*, *o*) in Figure 3) would equal the sequential concatenation of all tables in a horizontal store which uses a set of relational tables (e.g. Figure 4(b)).

https://doi.org/10.1017/S0269888916000217 Published online by Cambridge University Press

(a)

| Subject | born | died | founder | board | home | developer | version | kernel | preceded | graphics | industry | employees | HQ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Charles Flint | 1850 | 1934 | IBM | | | | | | | | | | |
| Larry Page | 1973 | | Google | Google | Palo Alto | | | | | | | | |
| Android | | | | | | Google | 4.1 | Linux | 4.0 | OpenGL | | | |
| Google | | | | | | | | | | | {Software, Internet} | 54,604 | Mountain View |
| IBM | | | | | | | | | | | {Software, Hardware, Services} | 433,362 | Armonk |

(b)

borntable

| Subject | born |
|---|---|
| Charles Flint | 1850 |
| Larry Page | 1973 |

died table

| Subject | died |
|---|---|
| Charles Flint | 1934 |

founder table

| Subject | founder |
|---|---|
| Charles Flint | IBM |
| Larry Page | Google |

board table

| Subject | board |
|---|---|
| Larry Page | Google |

home table

| Subject | home |
|---|---|
| Larry Page | Palo Alto |

developer table

| Subject | developer |
|---|---|
| Android | Google |

version table

| Subject | version |
|---|---|
| Android | 4.1 |

kernel table

| Subject | kernel |
|---|---|
| Android | Linux |

preceded table

| Subject | preceded |
|---|---|
| Android | 4.0 |

graphics table

| Subject | graphics |
|---|---|
| Android | OpenGL |

industry table

| Subject | industry |
|---|---|
| Google | Software |
| Google | Internet |
| IBM | Software |
| IBM | Hardware |
| IBM | Services |

employees table

| Subject | employees |
|---|---|
| Google | 54,604 |
| IBM | 433,362 |

HQ table

| Subject | HQ |
|---|---|
| Google | Mountain View |
| IBM | Armonk |

**Figure 4** (a) Horizontal stores of Resource Description Framework (RDF) data using a single relational table. (b) Horizontal stores of RDF data using a set of relational tables

people table

| Subject | born | died | founder | board | home |
|---|---|---|---|---|---|
| Charles Flint | 1850 | 1934 | IBM | | |
| Larry Page | 1973 | | Google | Google | Palo Alto |

operation systems table

| Subject | developer | version | kernel | preceded | graphics |
|---|---|---|---|---|---|
| Android | Google | 4.1 | Linux | 4.0 | OpenGL |

companies table

| Subject | industry | employees | HQ |
|---|---|---|---|
| Google | {Software, Internet} | 54,604 | Mountain View |
| IBM | {Software, Hardware, Services} | 433,362 | Armonk |

**Figure 5** Type stores of Resource Description Framework data

So, viewed from the physical side of RDF stores, two approaches of vertical stores and horizontal stores are much more similar.

The type-store approach is actually a trade-off between the two kinds of horizontal stores. Compared with horizontal stores using a single relational table, type stores contain fewer null values (no null values in horizontal stores using multiple relational tables), and involve fewer join operations than horizontal stores using multiple relational tables (no join operations in horizontal stores using a single relational table). It should be noted that, like horizontal stores using a single relational table, type stores may contain multi-valued attributes and new predicates, resulting in changes to relational schema when new triples are inserted.

In relational databases, it is crucial and well known that in order to sustain workloads at any quality of surface, data need to be partitioned and clustered such that common query predicates have all the corresponding data stored near each other (locality). Horizontal and vertical RDF stores never get any locality with respect to predicates, that is qualifying tuples tend to be found scattered around the database. This is because subject URIs tend to be represented as integer numbers that are essentially random (depend mostly on bulkload order). Type stores are the only kind of design that allow relational techniques like table partitioning and table clustering to be applied. Type stores could be quite useful as they allow the highly typical star queries to be executed with much less joins.

Some major features of relational RDF data stores are summarized in Table 1.

https://doi.org/10.1017/S0269888916000217 Published online by Cambridge University Press

**Table 1** Major features of relational Resource Description Framework data stores

| | Join operations | Multi-valued attributes | Null values | Relational schema | Number of relation(s) |
|---|---|---|---|---|---|
| Vertical stores | *More self-joins* | *No* | *No* | *Fixed* | *Fixed* |
| Horizontal stores | | | | | |
| One table for all predicates | *No* | *Yes* | *Yes and many* | *Dynamic* | *Fixed* |
| One table for each predicate | *More joins* | *No* | *No* | *Dynamic* | *Dynamic* |
| Type stores | *Fewer joins* | *Yes* | Yes and fewer | *Dynamic* | *Dynamic* |

Understanding the major features of relational RDF data stores is very crucial and useful in the RDF store designs. In addition, certain high-level physical issues such as data ordering should be considered in the RDF store designs. For OWLIM and Virtuoso, for example, the former is a row store whereas the latter is a column store, and the former uses a query executor with tuple-at-a-time interpretation whereas the latter uses vectorized execution. As a result, Virtuoso has a performance edge for analytical queries, not so much for transactional queries.

### *3.1 Vertical stores*

Vertical stores (also called triple stores) use a single relational table to store a set of RDF statements, in which the relational schema contains three columns for *subject*, *property*, and *object*. Formally, each triple, say ($s$, $p$, $o$), occurs in the relational table as a row, that is, tuple $<s, p, o>$. Here subject $s$ is placed in column *subject* of this row, predicate $p$ is placed in column *property* of this row, and object $o$ is placed in column *object* of this row. Because vertical stores quickly encounter scalability limitations, several approaches have been proposed to deal with these limitations by using extensive sets of indices or by using selectivity estimation information to optimize the join ordering.

Sesame, a generic architecture for storing and querying RDF and RDF schema, is introduced by Broekstra *et al*. (2002). An important feature of the Sesame architecture is its abstraction from the details of any particular repository used for the actual storage. Therefore, Sesame can be ported to a large variety of different repositories. The implementation of Sesame (Broekstra *et al*., 2002) uses both PostgreSQL and MySQL as database platforms.

An RDF storage scheme called Hexastore RDF is proposed by Weiss *et al*. (2008). This scheme enhances the vertical partitioning idea and takes it to its logical conclusion. RDF data are indexed in six possible ways, one for each possible ordering of the three RDF elements. Each instance of an RDF element is associated with two vectors; each such vector gathers elements of one of the other types, along with lists of the third resource type attached to each vector element. Hence, a sextuple indexing scheme emerges. This format enables quick and scalable general-purpose query processing; it confers significant advantages (up to five orders of magnitude) over previous approaches for RDF data management, at the price of a worst-case fivefold increase in index space. Note that Hexastore focusses on exhaustive indexing of pairs of positions in triples such as SP, SO, …, OP. Being different from Hexastore, RDF-3X focusses on exhaustive indexing off all permutations of triples of positions such as SPO, SOP, …, OPS and TripleT (Wolff *et al*., 2015) focusses on exhaustive indexing of all single positions, S, P, and O.

### *3.2 Horizontal stores*

Under the horizontal representation, RDF data can be stored directly in a single table. This table has one column for each predicate occurring in the RDF graph and one row for each subject. Formally, for a triple ($s$, $p$, $o$), object $o$ is placed in column $p$ of row $s$. Note that for two triples, say ($s_i$, $p_i$, $o_i$) and ($s_j$, $p_j$, $o_j$), one may have $s_i = s_j$ and either $p_i = p_j$ or $p_i \neq p_j$. At this point, if $s_i = s_j$ and $p_i \neq p_j$, $o_i$ and $o_j$ are placed in different columns $p_i$ and $p_j$ of the same row. However, if $s_i = s_j$ and $p_i = p_j$, $o_i$ and $o_j$ are placed in the same column of the same row, and a set of values $\{o_i, o_j\}$ results. Of course, it is possible that $s_i \neq s_j$ and $p_i = p_j$. Then $o_i$ and $o_j$ are placed in the same column of different rows $s_i$ and $s_j$. It is very common that for any two

https://doi.org/10.1017/S0269888916000217 Published online by Cambridge University Press

triples ($s_i$, $p_i$, $o_i$) and ($s_j$, $p_j$, $o_j$) in the context of massive RDF triples, they have different subjects and different predicates, and $o_i$ and $o_j$ are placed in different columns of different rows. As a result, row $s_i$ has a *null value* in the $p_j$ column and row $s_j$ has a *null value* in the $p_i$ column. This will lead to a sparse table with many null values.

To solve these problems of null values and sets of values, efforts have been made to partition a single table vertically into a set of property tables using predicates. Each predicate has a table over the schema (*subject*, *object*), in which a binary relation between a subject and an object with respect to the given predicate is represented. Formally, for two triples, say ($s_i$, $p_i$, $o_i$) and ($s_j$, $p_j$, $o_j$), if $p_i = p_j$, there are two tuples $<s_i, o_i>$ and $<s_j, o_j>$ in the same table. However, if $p_i \neq p_j$, the tuples $<s_i, o_i>$ and $<s_j, o_j>$ occur in two different tables. It is clear that the number of relational tables is the same as the number of predicates in the RDF data sets.

SW-Store was proposed by Abadi *et al*. (2007, 2009) as an RDF data store that vertically partitions RDF data (by predicates) into a set of property tables, maps them onto a column-oriented database, and builds a subject–object index on each property table. Note that the implementation of SW-Store relies on the C-Store column-store database (Stonebraker *et al*., 2005) to store tables as collections of columns rather than as collections of rows. Current relational database systems, for example, Oracle, DB2, SQL Server, and Postgres, are standard row-oriented databases in which entire tuples are stored consecutively. In addition, the results of an independent evaluation of SW-Store are reported by Sidirourgos *et al*. (2008).

Extending the SW-Store approach, an approach called SPOVC is proposed by Mulay and Kumar (2012). The main techniques used in this approach are horizontal partitioning of logical indices and special indices for values and classes. The SPOVC approach uses five indices, namely, *subject*, *predicate*, *object*, *value*, and *class*, on top of column-oriented databases.

### *3.3 Type stores*

To reduce null values in horizontal stores with a single table, the property-table approach (i.e. type stores) has been proposed. The basic idea of this approach is to divide one wide table into multiple smaller tables so that each table contains related predicates as its columns. Formally, for two triples, say ($s_i$, $p_i$, $o_i$) and ($s_j$, $p_j$, $o_j$), suppose that $p_i$ and $p_j$ are related. Then these two triples occur in the same table, with one row for each subject. Furthermore, when $s_i = s_j$ and $p_i \neq p_j$, $o_i$ and $o_j$ are placed in different columns $p_i$ and $p_j$ of the same row; when $s_i = s_j$ and $p_i = p_j$, $o_i$ and $o_j$ are placed in the same column of the same row, and a set of values $\{o_i, o_j\}$ results; when $s_i \neq s_j$ and $p_i = p_j$, $o_i$ and $o_j$ are placed in the same column of different rows $s_i$ and $s_j$. It is not difficult to see that designing a schema for property tables depends on identifying related predicates.

Jena is an open-source toolkit for Semantic Web programmers (McBride, 2002). It implements persistence for RDF graphs using an SQL database through a JDBC connection. Jena has evolved from its first version, Jena 1, to a second version, Jena 2. In the Jena RDF, the grouping of predicates is defined by applications (Wilkinson *et al*., 2003; Wilkinson, 2006). Applications typically have access patterns in which certain subjects or properties are accessed together. In particular, the application programmer must specify which predicates are multivalued. For each such multi-valued predicate $p$, a new relational table is created, with a schema consisting of *subject* and $p$. Jena also supports so-called *property-class* tables, in which for each value of the rdf:type predicate, a new table is created. The remaining predicates that are not in any defined group are stored independently.

Using a dynamic relation model, a system called FlexTable is proposed to store RDF data (Wang *et al*., 2010). In FlexTable, all triples of an instance are coalesced into one tuple, and all tuples are stored in relational schema. To partition all the triples into several tables, first, a schema evolution method is proposed, based on a lattice structure, to evolve schema automatically when new triples are inserted; second, a page layout with an interpreted storage format is proposed to reduce the physical adjustment cost during schema evolution.

For each subject $s$ in the RDF graph $G$, a set of predicates satisfying $\{p | (\exists o) \wedge ((s, p, o) \in G)\}$ is obtained, which is called the *signature* of $s$ (Sintek & Kiesel, 2006). The predicates in the set are considered as related predicates. Then, for each signature, a corresponding predicate relational table called the signature table is created. The relational schema contains the subject and the set of predicates. Based mainly on the concepts of

https://doi.org/10.1017/S0269888916000217 Published online by Cambridge University Press

signatures and signature tables which are organized in a lattice-like structure, RDFBroker, an RDF store, was introduced by Sintek and Kiesel (2006). Note that the approach by Sintek and Kiesel (2006) actually creates many small tables. To improve query evaluation performance, various criteria are proposed for merging small tables into larger ones, but this introduces null values when a predicate is absent.

Based on RDF document structure, a storage scheme is proposed by Matono and Kojima (2012), which stores RDF graphs without decomposition. Considering that adjacent triples have a strong relationship and can be described for the same resource, a set of adjacent triples that refer to the same resource is defined as an RDF paragraph. Then the table layout is constructed based on RDF paragraphs (Luo *et al*., 2012).

Levandoski and Mokbel (2009) proposed a data-centric schema-creation approach for storing RDF data in relational databases. With the aim of maximizing the size of each group of predicates and meanwhile minimizing the number of null values that occur in the tables, association rule mining is used to determine automatically the predicates that often occur together. According to the support threshold, which measures the strength of correlation between properties in the RDF data, properties which are grouped together in the same cluster may constitute a single *n*-ary table, and properties which are not grouped in any cluster may be stored in binary tables. Finally, in the partitioning phase, the formed clusters are checked, and the trade-off is made between storing as many RDF properties as possible in clusters while keeping null storage to a minimum based on the null threshold. Actually, the approach by Levandoski and Mokbel (2009) provides a tailored schema for each RDF data set, using a balance between *n*-ary and binary tables.

Each triple in the form (*subject*, *predicate*, *object*) is classified into categories by Matono *et al*. (2005) according to the type of predicate, and then subgraphs are constructed for each category. The graph is decomposed into five subgraphs according to the type of predicate: class inheritance graphs, property inheritance graphs, type graphs, domain-range graphs, and generic graphs. Each subgraph is stored by applicable techniques into distinct relational tables. More precisely, all classes and properties are extracted from RDF schema data, and all resources are also extracted from RDF data. Each extracted item is assigned an identifier and a path expression and stored in the corresponding relational table. Because the proposed scheme retains schema information and path expressions for each resource, the path-based relational RDF database (Matono *et al*., 2005) can process path-based queries efficiently and store RDF instance data without schema information.

### *3.4 Hybrid stores*

In addition to the three basic relational stores of RDF data sets (i.e. vertical, horizontal, and type stores), efforts have also been made to store RDF data by using two or more of the basic store types together or revising the three basic store types. Actually, in Levandoski and Mokbel (2009), for example, type stores are used for properties which are grouped together in the same cluster, and horizontal stores with multiple relational tables are used for properties which are not grouped in any cluster.

A hybrid storage scheme for RDF data management was proposed by Kim (2006), in which some frequently appearing properties in RDF data were identified. Then sets of RDF data with one or more frequently used properties were physically grouped and stored together in a corresponding property table. RDF data with less important properties were grouped together in a common triple table. There was no duplication among tables. To process a query having a specific or distinguishable property, the evaluation was done by accessing only the corresponding property table.

Sperka and Smrz (2012) used two types of relational tables: *class tables* and *property tables*. A class table has the form of *iri* (*member*), where *iri* is the name of a relation and *member* is the only attribute. A class table contains members of the class, and therefore rows of the table are given by loaded triples of the form *any:iri1rdf:type any:iri2*, that is, *iri1* becomes a row in table *iri2*. A property table has the structure *iri* (*subject*, *object*), where *iri* is the name of a relation. A property table emerges when a triple of the form *any:irirdf:typerdf:Property* or *any:iri1any:property any:iri2* is imported (where *any:property* is not an ontology-structuring property or *rdf:type*, that is, an IRI not interpreted by the system). A row (*iri1*; *iri3*) is created in table *iri2* when a triple of the form *iri1iri2iri3* is encountered.

A novel storage and query mechanism for RDF was introduced by Bornea *et al*. (2013). The proposed mechanism works on top of existing relational representations. Identifying that there are challenges,

https://doi.org/10.1017/S0269888916000217 Published online by Cambridge University Press

**Table 2** Major features of typical relational stores for Resource Description Framework (RDF) data

| | Scale | Dynamics in load and update | Query languages | Failover |
|---|---|---|---|---|
| Vertical stores | | | | |
| Sesame | *Small scale* | *Update support* | *RQL* | *No* |
| Hexastore | *Small scale* | *No* | *SPARQL* | *No* |
| Horizontal stores | | | | |
| SW-Store | *Large scale* | *No* | *SPARQL* | *No* |
| Mulay and Kumar (2012) | *Large scale* | *Update support* | *SPARQL* | *No* |
| Type stores | | | | |
| Jena 2 | *Small scale* | *Update support* | *RDQL/SPARQL* | *No* |
| FlexTable | *Small scale* | *Update support* | *SPARQL* | *No* |
| RDFBroker | *Large scale* | *Update support* | *SeRQL* | *No* |
| Matono and Kojima (2012) | *Large scale* | *No* | *SPARQL* | *No* |
| Levandoski and Mokbel (2009) | *Middle scale* | *No* | *Unclear* | *No* |
| Hybrid stores | | | | |
| Matono *et al*. (2005) | *Large scale* | *No* | SQL | *No* |
| Kim (2006) | *Small scale* | *No* | *SPARQL* | *No* |
| Sperka and Smrz (2012) | *N/A* | *No* | *SPARQL* | *No* |
| Bornea *et al*. (2013) | *Large scale* | *Load support* | *SPARQL* | *No* |

RQL = RDF Query Language; SPARQL = SPARQL Protocol and RDF Query Language; RDQL = RDF Data Query Language; SeRQL = Sesame RDF Query Language.

including data sparsity and schema variability, in storing RDF in relational databases, Bornea *et al*. (2013) treated the columns of a relation as flexible storage locations that were not pre-assigned to any predicate, but rather predicates were assigned to them dynamically during insertion. The assignment ensured that a predicate was always assigned to the same column or more generally the same set of columns.

Some typical approaches for relational stores of RDF data and major features are summarized in Table 2. In Table 2, several query languages are used, which are *RQL* (RDF Query Language) (Karvounarakis *et al*., 2002), *RDQL* (RDF Data Query Language)[15], *SeRQL* (Sesame RDF Query Language)[16], and finally the W3C Recommendation SPARQL[17].

## 4 Resource Description Framework stores in not only SQL databases

For massive RDF data, say DBpedia, there are 333 M triples, 150k types, and thousands of predicates (Bornea *et al*., 2013). To tackle the big-data challenge, research has recently moved onward to distributed RDF data-management systems. A first attempt in this direction is 4store[18]. 4store stores RDF data as quads with the form (*model*, *subject*, *predicate*, *object*), where URIs, literals, and blank nodes are all encoded using a cryptographic hash function. Two types of computational nodes are defined in a distributed setting. The first is storage nodes, which store the actual data, and the second is processing nodes, which are responsible for parsing incoming queries and handling all distributed communications with the storage nodes during query processing. 4store partitions the data into non-overlapping segments and distributes the quads based on a hash partitioning of their subject. Data in 4store are organized as property tables. YARS2 (Harth *et al*., 2007) is a native distributed RDF processing system which uses simple hashing as its triple partitioning strategy and builds in main memory a set of six sparse indices on a subset of the combinations of RDF triple components. Clustered TDB (Owens *et al*., 2009) uses hashing on *subject*, *object*, and *predicate* to distribute each triple three times to the server cluster.

[15] `http://www.w3.org/Submission/RDQL/`
[16] `http://www.w3.org/2001/sw/wiki/SeRQL`
[17] `http://www.w3.org/TR/rdf-sparql-query/`
[18] `http://4store.org/`

https://doi.org/10.1017/S0269888916000217 Published online by Cambridge University Press

In distributed RDF data management, RDF data partitioning is an important problem. Basically, based on how the RDF data set is partitioned and how partitions are stored and accessed, two categories of existing distributed RDF systems can be identified (Lee & Liu, 2013). The first category generally partitions an RDF data set across multiple servers using horizontal (random) partitioning. We call it *horizontal partitioning*. Then the partitions are stored using distributed file systems such as Hadoop Distributed File System (HDFS)[19], and queries are processed by parallel access to the clustered servers using a distributed programming model such as Hadoop MapReduce. SHARD directly stores RDF triples in HDFS as a text file and runs one Hadoop job for each clause (triple pattern) of a SPARQL query (Rohloff & Schantz, 2011). RDF triples are stored in HDFS by hashing on predicates, and the system runs one Hadoop job for each join in a SPARQL query (Husain *et al*., 2011). Also using HDFS, a storage framework was proposed by Husain *et al*. (2009) to store RDF data, and then based on the proposed storage schema, Hadoop's MapReduce framework was used to retrieve RDF data. The MapReduce framework has been explored for scalable processing of RDF data. MapReduce was used by Zhang *et al*. (2012a) for efficient join processing over a large RDF graph to minimize query response time as much as possible. With the aim of optimizing RDF graph pattern matching on MapReduce, a Nested TripleGroup Data Model and Algebra was proposed by Ravindra *et al*. (2011) for efficient graph pattern query processing in the cloud, and this work was further extended with a scan sharing technique used to optimize processing of graph patterns with repeated properties (Kim *et al*., 2012).

The second category partitions an RDF data set across multiple nodes using hash partitioning on the subject, object, and predicate of RDF triples or any combination of these. We call it *hash partitioning*. Then the partitions are stored locally in a database such as a key-value store like HBase or an RDF store like RDF-3X and accessed through a local query interface. Unlike the horizontal partitioning described above, the hash partitioning resorts to distributed computing frameworks such as Hadoop MapReduce only to perform the cross-server coordination and data transfer required for distributed query execution such as joins of intermediate query results from two or more partition servers.

In order to improve data scalability for RDF data management, in addition to data partitioning, some additional issues like failover, consistency, and transaction isolation are essential for distributed RDF data storage and access. In particular, corresponding to RDF data partitioning, the issue of accessing RDF triples partitioned among servers must be considered. First, the stored partitions can still be accessed even if some of partition servers are failed. Second, it is necessary to minimize communication among partition servers. For the latter, clustering partitions has been applied for RDF data management. Virtuoso has a commercial Cluster Edition that allows it to scale. The other systems that have this are BigData and 4store. For the former, replication is an effective method for failover. Actually, replication can store partitions locally and reduce communication among partition servers. So, the replication-based cluster architectures are designed for better query scalability and resilience. A semantic hash-partitioning method is introduced by Lee and Liu (2013), which extends simple hash partitioning by combining direction-based triple grouping with direction-based triple replication. Apache Cassandra developed originally by Facebook (Lakshman & Malik, 2010) provides decentralized data storage and failure tolerance based on replication and failover. The RDF graph is partitioned into compact subgraphs and each subgraph is assigned to a host (Huang *et al*., 2011). Using replication at the borders of the partitions, queries which do not exceed a certain diameter can then be processed in parallel over all partitions without further communication between hosts. Note that OWLIM only has replicated clustering. It means that it is not able to parallelize one query over multiple servers, and just to load balance a workload with many queries over multiple identical servers.

NoSQL data-management systems have emerged as a commonly used infrastructure for handling big data outside the RDF space. The various NoSQL data stores were divided into four major categories by Grolinger *et al*. (2013): *key-value stores*, *column-family stores*, *document stores*, and *graph databases*. Key-value stores have a simple data model based on key-value pairs. 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 data set consists of several rows. Each row is addressed by a primary key and is composed

[19] `http://hadoop.apache.org/hdfs`

https://doi.org/10.1017/S0269888916000217 Published online by Cambridge University Press

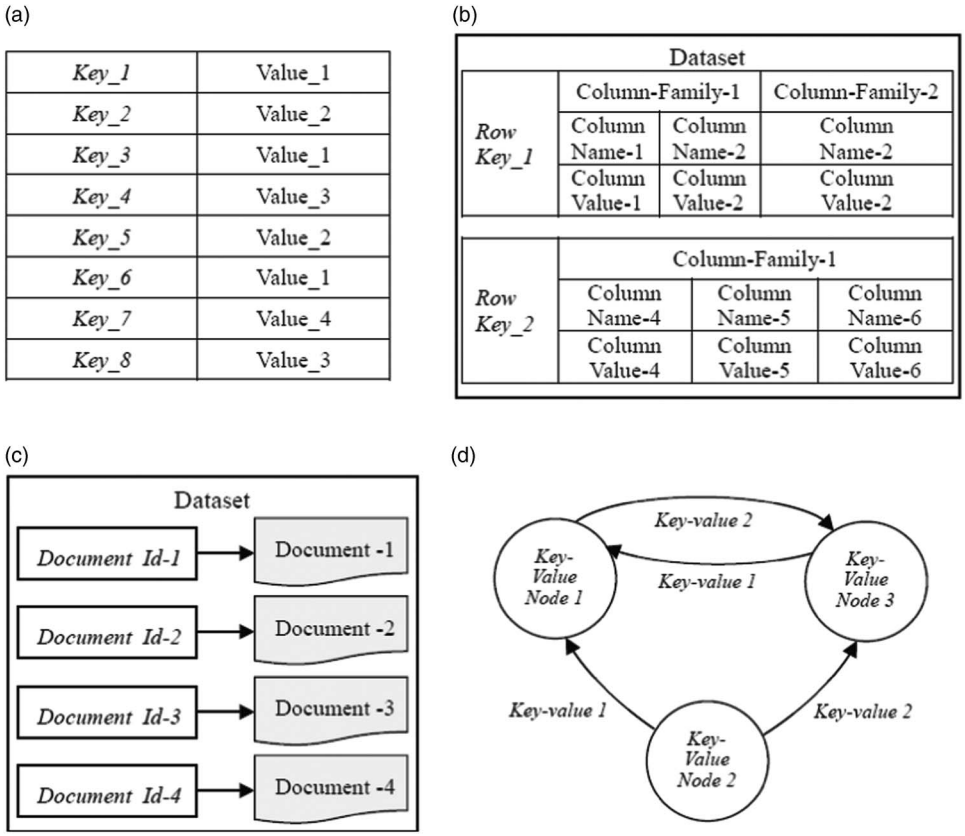



**Figure 6** Different types of NoSQL data model (Grolinger *et al*., 2013). (a) Key-value store. (b) Column-family store. (c) Document store. (d) Graph database

of a set of column families. Note that different rows can have different column families. Representative column-family stores include Apache HBase[20], which directly implements the Google BigTable concepts. According to Grolinger *et al*. (2013), there is one type of column-family store, say Amazon SimpleDB (Stein & Zachrias, 2010) 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. In addition, Cassandra (Lakshman & Malik, 2010) provides the additional functionality of super columns, which are formed by grouping various columns together. 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. Typically, CouchDB[21] and the Couchbase server[22] use the JSON format for data storage, whereas MongoDB[23] stores data in BSON (Binary JSON). Graph databases use graphs as their data model, and a graph is used to represent a set of objects, known as vertices or nodes, and the links (or edges) that interconnect these vertices.

Illustrative representations of these NoSQL models were presented by Grolinger *et al*. in 2013 and are shown in Figure 6.

Actually, massive RDF data-management merits the use of big-data infrastructure because of the scalability and high performance of cloud data management. A number of efforts have been made to develop RDF data-management systems based on NoSQL systems. SimpleDB by Amazon was used as a back end to store RDF data quickly and reliably for massive parallel access (Stein & Zachrias, 2010).

[20] http://hbase.apache.org/
[21] http://couchdb.apache.org/
[22] http://www.couchbase.com/couchbase-server/overview
[23] http://www.mongodb.org/

https://doi.org/10.1017/S0269888916000217 Published online by Cambridge University Press

Cloud-based key-value stores (e.g. BigTable) were used by Gueret *et al.* in 2011, and a robust query engine was developed over these key-value stores. In addition, there is a new RDF store on the block, called SPARQLcity[24], which is a Hadoop-based graph analytical engine for performing rich business analytics on RDF data with SPARQL. SPARQLcity is the first just in time compiled engine in SPARQL query execution. Given the fact that NoSQL systems offer either no support or only high latency support (MapReduce) for effective join processing, however, SPARQL queries with many joins, which is the mainstay, run into big problems on such systems. The normal NoSQL APIs (Application Programming Interfaces) that are centred on individual key lookup (whether one looks up a value, column-family, or document) simply have too high latency if one has to join tens of thousands (or billions) of RDF triples.

Several NoSQL systems for RDF data were investigated by Cudre-Mauroux *et al*. (2013), including *document stores* (e.g. CouchDB[25]), *key-value/column stores* (e.g. Cassandra[26] and HBase[27]), and *query compilation for Hadoop* (e.g. Hive[28]). Major characteristics of these four NoSQL systems are described (Cudre-Mauroux *et al*., 2013). First, Apache HBase is an open-source, horizontally scalable, row-consistent, low-latency, and random-access data store. HBase uses HDFS as a storage back end and Apache ZooKeeper[29] to provide support for coordination tasks and fault tolerance. HBase is a column-oriented distributed NoSQL database system. Its data model is a sparse, multi-dimensional sorted map. Here, *columns* are grouped into *column families*, and timestamps add an additional dimension to each cell. HBase is well integrated with Hadoop, which is a large-scale MapReduce computational framework. The second HBase implementation uses Apache Hive, a SQL-like data-warehousing tool that enables querying using MapReduce. Third, Couchbase[30] is a document-oriented, schema-less distributed NoSQL database system with native support for JSON documents. Couchbase is intended to run mostly in-memory and on as many nodes as needed to hold the whole data set in RAM (random-access memory). It has a built-in object-managed cache to speedup random reads and writes. Updates to documents are first made in the in-memory cache and are only later processed to disk using an eventual consistency paradigm. Finally, Apache Cassandra is a NoSQL database management system originally developed by Facebook (Lakshman & Malik, 2010), which provides decentralized data storage and failure tolerance based on replication and failover.

Among the NoSQL systems available, HBase has been the most widely used. To manage distributed RDF data, HBase and MySQL Cluster were used by Franke *et al.* (2011) to store RDF data. An empirical comparison of these two approaches was then conducted on a cluster of commodity machines. The Hexastore (Weiss *et al*., 2008) schema was applied for HBase to store verbose RDF data (Sun & Jin, 2010). Also based on HBase, two distributed triple stores called $H_2$RDF and $H_2$RDF+ were developed to optimize distributed joins using MapReduce (Papailiou *et al*., 2012, 2013). The main differences between $H_2$RDF and $H_2$RDF+ are found in the join algorithms and the number of maintained indices (three versus six). Combining the Jena framework with the storage provided by HBase, Khadilkar *et al.* (2012) developed several versions of a triple store. A scalable technique was created (Przyjaciel-Zablocki *et al*., 2012) for performing indexed nested loop joins, which combines the power of the MapReduce paradigm with the random-access pattern provided by HBase. Using a combination of MapReduce and HBase, a storage schema called RDFChain was proposed by Choi *et al.* (2013) to support scalable storage and efficient retrieval of a large set of RDF data. Focussing on large data sets from various areas of provenance which record the history of an *in silico* experiment, large collections of provenance graphs were serialized as RDF graphs in an Apache HBase database (Chebotko *et al*., 2013). On this basis, storage, indexing, and query techniques for RDF data in HBase were proposed, which are better suited for provenance data sets than generic RDF graphs.

Figure 7 presents an overview of the architecture typically used by Jena-HBase (Khadilkar *et al*., 2012). In Jena-HBase, the concept of a store is applied to provide data manipulation capabilities on underlying

[24] http://sparqlcity.com/
[25] http://couchdb.apache.org/
[26] http://cassandra.apache.org/
[27] http://hbase.apache.org/
[28] http://hive.apache.org/query
[29] http://zookeeper.apache.org/
[30] http://www.couchbase.com/

https://doi.org/10.1017/S0269888916000217 Published online by Cambridge University Press

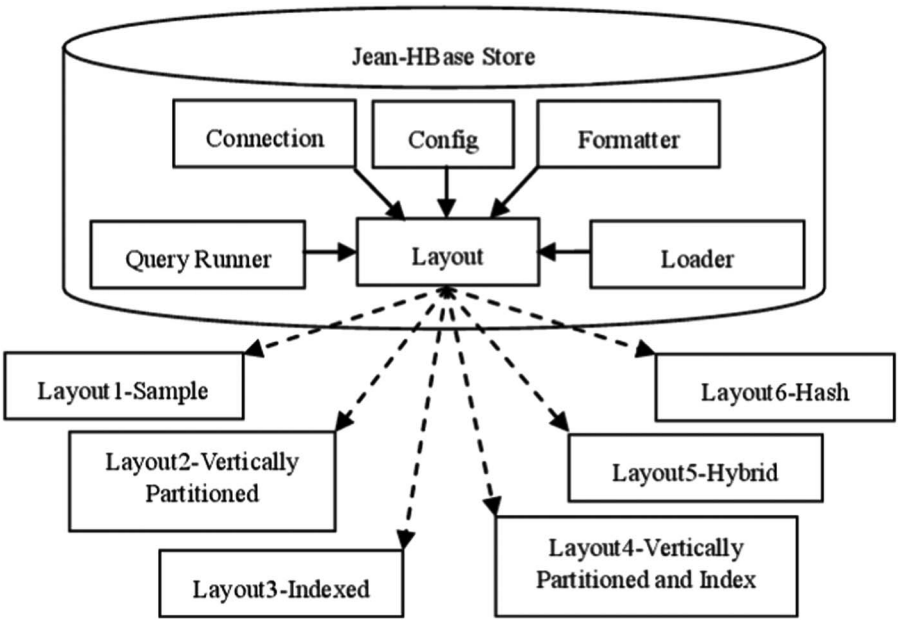



**Figure 7** Architectural overview of Jena-HBase

**Table 3** Storage schemas for Jena-HBase layouts

| Layout type | Storage schema |
|---|---|
| Simple | Three tables each indexed by subjects, predicates, and objects |
| Vertically partitioned (VP) | For every unique predicate, two tables, each indexed by subjects and objects |
| Indexed | Six tables representing the six possible combinations of a triple, namely, SPO, SOP, PSO, POS, OSP, and OPS |
| VP and indexed | VP layout with additional tables for SPO, OSP, and OS |
| Hybrid | Simple + VP layouts |
| Hash | Hybrid layout with hash values for nodes and a separate table containing hash-to-node mappings |

HBase tables. A store represents a single RDF data set and can be composed of several RDF graphs, each with its own storage layout. A layout uses several HBase tables with different schemas to store RDF triples, and each layout provides a trade-off in terms of query performance/storage. Then all operations on an RDF graph are implicitly converted into operations on the underlying layout. The operations on the underlying layout include the following.

a Formatting a layout, that is, deleting all triples while preserving tables (Formatter block in Figure 7).
b Loading–unloading triples into a layout (Loader block in Figure 7).
c Querying a layout for triples that match a given <S, P, O> pattern (Query Runner block in Figure 7).
d Additional operations include maintaining an HBase connection (Connection block in Figure 7) and maintaining configuration information for each RDF graph (Config block in Figure 7).

A summary of the storage schema used by each layout is presented in Table 3 (Khadilkar *et al.*, 2012).

Another important category of NoSQL RDF storage is the concept of graph databases (Angles & Gutierrez, 2008). Focussing on the structure of RDF data, these data are viewed as a classical graph in which subjects and objects form the nodes and triples specify directed and labelled edges. Note that here RDF graphs may contain cycles and have labelled edges. Angles and Gutierrez (2005) surveyed graph database models and query languages and they further propose that RQL should incorporate graph database query language primitives.

https://doi.org/10.1017/S0269888916000217 Published online by Cambridge University Press

**Table 4** Major features of typical not only SQL stores for Resource Description Framework (RDF) data

| | Distributed/ decentralized | Data partitioning | Replication and failover | Storage back end | Running | Join query |
|---|---|---|---|---|---|---|
| Column-family stores | | | | | | |
| HBase | *Distributed* | *Hash* | *Fault tolerance* | *HDFS* | *Disk* | *Hadoop* |
| Hive | *Distributed* | *Hash* | *Partially* | *Apache Hive* | *Disk* | *MapReduce* |
| Cassandra | *Decentralized* | *Hash* | *Failure tolerance* | *Apache Cassandra* | *Disk* | *Hadoop* |
| Document stores | | | | | | |
| Couchbase | *Distributed* | *Hash* | *Failure tolerance* | *JSON* | Mostly in-memory | *MapReduce* |
| Graph databases | | | | | | |
| Trial | *N/A* | *N/A* | *N/A* | *Classical graph* | *N/A* | *Triple algebra* |
| Trinity.RDF | *Distributed* | *Hash* | *No* | *Native graph* | *Memory based* | *SPARQL* |
| Neo4j | *Centralized* | *None* | *No* | *Property graph* | *Disk* | *Cypher* |
| Dydra | *Distributed* | *Hash* | *Failure tolerance* | *Property graph* | *Disk* | *SPARQL* |

HDFS = Hadoop Distributed File System; JSON = JavaScript Object Notation; SPARQL = SPARQL Protocol and RDF Query Language.

Identifying that the standard graph database model (essentially labelled graphs) is different from the triples-based RDF model, a triple-based model called *Trial* was introduced by Libkin *et al*. (2013), which combines the usual idea of triple stores used in many RDF implementations with that of graphs with data. Actually, one of the major problems encountered in modelling RDF data as classical graphs is that an edge or a labelled edge cannot represent the ternary relation given by an RDF triple. It is natural to use hypergraphs for this purpose with three-node connecting edges instead of classical two-node edges. Hypergraphs are represented naturally by bipartite graphs (Hayes & Gutierrez, 2004), in which the concept of an RDF bipartite graph is introduced as an intermediate model for RDF data.

With a focus on distributed and Web-scale RDF data management, a memory-based graph engine called *Trinity.RDF* is introduced (Zeng *et al*., 2013). Trinity.RDF models RDF data in its native graph form, in which entities (i.e. subjects and objects of RDF triples) are represented as graph nodes and relationships (i.e. predicates of RDF triples) are represented as graph edges. Each RDF entity is represented as a graph node with a unique *id* and stored as a key-value pair in the Trinity memory cloud. Formally, a key-value pair (*node-id*, <*in-adjacency-list*, *out-adjacency-list*>) consists of the node-id as the key and the node research directions to explore in managing voluminous adjacency list as the value. The adjacency list is divided into two lists: one for neighbours with incoming edges and the other for neighbours with outgoing edges. Each element in the adjacency list is a (predicate, node-id) pair, which records the *id* of the neighbour and the predicate on the edge.

Nowadays, representative graph database products for RDF data mainly include *Neo4j*[31] and *Dydra*[32]. *Dydra* is a cloud-based graph database. With Dydra, RDF data is natively stored as a property graph, directly representing the relationships in the underlying RDF data, and can be accessed and updated via an industry-standard query language specifically designed for graph processing.

Some typical approaches for NoSQL stores of RDF data and major features are summarized in Table 4.

## 5 Benchmarks for Resource Description Framework data stores

RDF data-management systems are numerous, and it is necessary to test these systems under data and workloads with various characteristics. A number of RDF benchmarks have been developed to test the performance of RDF stores. The focus of RDF benchmarks is mainly on RDF store performance in terms of scalability (i.e.

[31] http://www.neo4j.org/
[32] http://www.dydra.com

https://doi.org/10.1017/S0269888916000217 Published online by Cambridge University Press

the number of triples in the tested RDF data). Basically, RDF benchmarks can be categorized into general-purpose RDF benchmarks and special-purpose RDF benchmarks, which are called foundational RDF benchmarks and extended RDF benchmarks, respectively, in the following.

### 5.1 Foundational Resource Description Framework benchmarks and their usages

The most representative and widely used RDF benchmarks are the LUBM (Guo *et al*., 2005), the BSBM (Bizer & Schultz, 2009), the SP$^2$Bench (Schmidt *et al*., 2009) and the DBPSB (Morsey *et al*., 2011). LUBM, BSBM, and SP$^2$Bench create a synthetic data set based on a use case scenario and define a set of queries covering a spectrum of query characteristics. DBPSB takes a different approach and proposes a benchmark creation methodology based on real-world data and query logs.

- The LUBM is one of the first RDF benchmarks which considers a university domain. LUBM uses an artificial data generator to generate synthetic data for universities, their departments, their professors, employees, courses, and publications. In addition, the LUBM benchmark provides 14 test queries for the purpose of testing the performance of RDF stores.
- The BSBM is based on an e-commerce use case in which a set of products is provided by a set of vendors and consumers post reviews regarding these products. The BSBM benchmark comes with 12 queries and two query mixes (sequences of the 12 queries) for the purpose of testing RDF store performance.
- The SP$^2$Bench benchmark uses the Digital Bibliography & Library Project (DBLP) as its domain and generates a synthetic data set mimicking the original DBLP data, including information about publications and authors. The SP$^2$Bench benchmark is accompanied by 12 queries.
- The DBPSB extracts structured information from Wikipedia. The data set consists of ~150 million triples (22 GB) (Duan *et al*., 2011). The entities stored in the triples come from a wide range of data types, including Person, Film, (Music) Album, Place, Organization, etc. Then a wide range of queries can be evaluated over the variety of entities stored in the data set, but there is no defined set of representative queries.

Concerning the RDF benchmarks described above, Morsey *et al*. (2012) made some comments about their main characteristics. LUBM relies solely on plain queries without SPARQL features such as FILTER or REGEX, and its querying strategy (10 repeats of the same query) accommodates caching. The BSBM data and queries are artificial, and the data schema is very homogeneous and resembles a relational database. SP$^2$Bench relies on synthetic data and a small (25 M triples) synthetic data set for querying.

Aiming to develop industry-strength benchmarks for graph and RDF data-management systems, the Linked Data Benchmark Council[33] recently develops two benchmarks, which are the Social Network Benchmark (SNB)[34] and the Semantic Publishing Benchmark (SPB)[35] (Angles *et al*., 2014). The SNB aims at testing graph data-management technologies for three scenarios: interactive (transaction query workload), business intelligence (analytical query workload), and graph analytics (graph analysis algorithms, such as PageRank). A wide variety of systems could potentially execute one or more workloads of the SNB. Currently, the Interactive Workload is in draft release stage, the other two workloads are still under development. The SPB is based on the BBC news website, and models a mixed query and update workload with a limited amount of semantic inferencing. SPB performance is measured by producing a workload of CRUD (Create, Read, Update, Delete) operations which are executed simultaneously.

Based on the comparison among the benchmarks of LUBM, SP$^2$Bench, BSBM V2.0, BSBM V3.0, and DBPSB2 in Morsey *et al*. (2012), SNB and SPB are included and a comprehensive comparison among these benchmarks is shown in Table 5.

Various RDF benchmarks have been used to evaluate existing RDF store systems. SP$^2$Bench was used by Schmidt *et al*. (2008) to compare experimentally four RDF storage strategies which all relied on a physical

[33] http://ldbc.eu/
[34] http://ldbcouncil.org/benchmarks/snb/
[35] http://ldbcouncil.org/benchmarks/spb/

https://doi.org/10.1017/S0269888916000217 Published online by Cambridge University Press

**Table 5** Comparison of six Resource Description Framework benchmarks

| | Test data | Test queries | Distributed queries | Use case | Classes | Properties |
|---|---|---|---|---|---|---|
| LUBM | Synthetic | Synthetic | 14 | Universities | 43 | 32 |
| $SP^2$Bench | Synthetic | Synthetic | 12 | DBLP | 8 | 22 |
| BSBM | Synthetic | Synthetic | 12 | E-commerce | 8 | 51 |
| DBPSB | Real | Real | 20 | DBpedia | 239 (base) + 300 K (*YAGO*) | 1200 |
| SNB | Synthetic | Synthetic | 14 | Social network | 19 | 29 (attributes) 20 (relations) |
| SPB | Synthetic | Synthetic | 25 | BBC | *Unclear* | *Unclear* |

LUBM = Lehigh University Benchmark; DBLP = Digital Bibliography & Library Project; BSBM = Berlin SPARQL Benchmark; DBPSB = DBpedia SPARQL Benchmark; SNB = Social Network Benchmark; SPB = Semantic Publishing Benchmark.

**Table 6** Resource Description Framework (RDF) stores tested using different RDF benchmarks

| | Traditional database stores | Cloud-based stores | Other type stores |
|---|---|---|---|
| LUBM | Sesame, Oracle Spatial and Graph with Oracle Database 12c | Virtuoso, HBase, BigData | AllegroGraph, BigOWLIM, YARS2 |
| $SP^2$Bench | Sesame, Jena-SDB | Virtuoso, HBase, Stardog | N/A |
| BSBM | C-Store, Sesame, Jena-SDB, Jena-TDB | Virtuoso, HBase, CouchDB, Hive, Cassandra, BigData, Stardog | 4store, BigOWLIM |
| DBPSB | Sesame, Jena-SDB, Jena-TDB | Virtuoso, HBase, CouchDB, Hive, Cassandra | 4store, BigOWLIM |
| SNB | N/A | Virtuoso | N/A |
| SPB | GraphDB | Virtuoso | N/A |

LUBM = Lehigh University Benchmark; BSBM = Berlin SPARQL Benchmark; DBPSB = DBpedia SPARQL Benchmark; SNB = Social Network Benchmark; SPB = Semantic Publishing Benchmark.

relational database back end. The first system considered was the *Sesameengine*. The remaining scenarios were the triple table store, the vertically partitioned store (Abadi *et al*., 2007), and a purely relational DBLP model. A benchmark creation methodology was proposed by Morsey *et al*. (2012) for SPARQL, which was applied to the DBpedia knowledge base. Then the generated benchmark was used to compare several popular triple-store implementations, including Virtuoso, Sesame, Jena-TDB, and BigOWLIM.

Using four representative NoSQL stores (CouchDB, Hive, HBase, and Cassandra) as well as a native RDF store (4store), Cudre-Mauroux *et al*. (2013) presented an empirical evaluation comparing these five stores for RDF processing when running standard RDF benchmarks (BSBM and DBPSB) on a cloud infrastructure. Each of the evaluated systems exhibited its own strengths and weaknesses.

Table 6 presents a comparison of RDF stores tested using different RDF benchmarks.

### *5.2 Extended Resource Description Framework benchmarks*

In addition to general-purpose foundational RDF benchmarks, several benchmarks have also been developed specifically for RDF data management. Minack *et al*. (2009), for example, extended the LUBM benchmark with synthetic scalable full-text data and corresponding queries for full-text-related query performance evaluation. RDF stores were benchmarked for basic full-text queries (classic IR (information retrieval) queries) as well as for hybrid queries (structured and full-text queries).

Generally, the extended benchmarks are proposed for the RDF data management in the context of special applications. To deal with linked stream data (LSD), for example, the RDF data model has been extended to represent stream data generated from sensors and social network applications. In this context,

https://doi.org/10.1017/S0269888916000217 Published online by Cambridge University Press

Le-Phuoc *et al*. (2012) have proposed a customizable evaluation framework and a corresponding methodology for realistic data generation, system testing, and result analysis. Based on the proposed LSBench platform, extensive experiments were conducted to compare various state-of-the-art LSD engines, taking into account the underlying principles of stream processing. Also for RDF stream processing, Zhang *et al*. (2012b) introduced a streaming RDF/SPARQL benchmark called SRBench for the purpose of assessing the abilities of streaming RDF/SPARQL processing engines in applying Semantic Web technologies to streaming data. Three real-world data sets from the LOD cloud (Bizer & Schultz, 2009) were chosen and used in the SRBench benchmark. SRBench was extended in Dell'Aglio *et al*. (2013) and a benchmark called CSRBench was then developed. CSRBench is applied to address query result correctness verification using an automatic method.

The RDF data model has also been extended to represent spatial data generated from geospatial applications. For geospatial RDF stores, a benchmark for querying geospatial data encoded in RDF was proposed by Kolas (2008), in which the LUBM benchmark was extended to include spatial entities and to test the functionality of spatially enabled RDF stores. Correspondingly, LUBM queries were extended to cover four primary types of spatial queries: *spatial location queries*, *spatial range queries*, *spatial join queries*, and *nearest-neighbour queries*. Actually, geospatial constraints have been considered in some benchmarks, for example, the benchmarks developed in the LOD2 project[36]. More recently, a benchmark for geospatial RDF stores, called Geographica, was developed by Garbis *et al*. (2013). Geographica uses both real-world and synthetic data to test the functionality and performance of various prominent geospatial RDF stores. The real-world workload uses publicly available linked geospatial data covering a wide range of geometric types (e.g. points, lines, polygons).

## 6 Conclusions and future work

This paper has provided an up-to-date overview of the current state of the art in RDF data storage. The focus of the paper is on storing massive RDF data sets. From the survey, it is clear that RDF data management is a very active research area and that many research efforts are ongoing. This paper presents the survey from three main perspectives: RDF data stores in traditional databases, RDF data stores in NoSQL databases, and benchmarks for RDF data stores. Note that due to the paper size limit, the paper concentrates only on RDF data stores and does not discuss indexing and querying RDF data.

The RDF model offers new research directions to explore in managing voluminous RDF data (Hassanzadeh *et al*., 2012). As a fairly new research area, it provides a number of research challenges and many interesting research opportunities for both the data-management community and the Semantic Web community. RDF data storage provides the infrastructure for RDF data management. On this basis, efficient RDF data queries are very potential. It is especially true for querying massive RDF data. Efficient querying of RDF data is supported by query optimization, which is even more important than query execution. But query optimization is not easy in SPARQL because there is no schema and thereby no correlations between attributes in star patterns. Characteristic sets are applied in RDF query optimization in Neumann and Moerkotte (2011). Also, techniques like index sampling during query optimization (to test the true cardinality of query predicates) as well as heuristics should be very useful for SPARQL query optimization. In addition to query optimization of RDF data, three major directions for future research are emphasized as follows.

It is clear from the survey that in order to tackle the big-data challenge, research has moved onward to distributed RDF data-management systems. As a result, RDF data management in NoSQL databases has only recently been gaining momentum because of the scalability and high performance of cloud data management. The actual subject of storing RDF data is RDF data indexing. Most NoSQL databases applied in RDF data management are developed for big data outside the RDF space, and their indexing techniques, for given RDF structure and contents, may not be sufficient to allow different types of RDF usage scenarios. So, a major research direction is the study and development of richer structural indexing

[36] http://www.openlinksw.com/dataspace/doc/oerling/weblog/Orri%20Erling%27s%20Blog/1808

https://doi.org/10.1017/S0269888916000217 Published online by Cambridge University Press

techniques and related query-processing strategies, following the success of such approaches for big data outside the RDF space in the cloud environment.

RDF provides a means of expressing and exchanging semantic metadata. Reasoning is essential for semantic data processing. Adding reasoning to RDF data management makes it possible to infer new facts and to exploit the semantics of properties and the information asserted in the knowledge base. So, reasoning is an important feature of any RDF store in many usage scenarios. It is shown in the paper that RDF data are mainly stored in the traditional databases or NoSQL databases. However, these two kinds of databases are all very weak in reasoning. So, another major open issue is the incorporation of RDFS (RDF Schema) and OWL (Web Ontology Language) into RDF data management for reasoning. Currently, relatively little work has been done on the impact of reasoning on RDF data management (Theoharis *et al*., 2005; Salvadores *et al*., 2010, 2011; Bishop *et al*., 2012; Sperka & Smrz, 2012). Stardog[37] as well as Oracle RDF Store[38] have support for the most advanced OWL profiles, but their scalability is questionable. The work of Urbani *et al*. (2009) only demonstrates scalable reasoning in concept. OWLIM supports some partial levels of reasoning, but merely by materializing this at import. Virtuoso has minimal reasoning support, but it is done at runtime. This makes it more flexible and easy to load, but can make queries significantly slower. Actually, inference can be seen as an important specific type of indexing and finally affects performance and completeness of query results.

The final major issue is benchmark studies of performance and data quality. More large-scale real RDF data sets should be introduced and applied in addition to existing RDF benchmarks and data found in widely used real RDF data sets. This is especially true for RDF data management in the context of the diversity of RDF applications (e.g. computing biology (Anguita *et al*., 2013) and geological information systems (Garbis *et al*., 2013)).

## Acknowledgements

This work was supported in part by the National Natural Science Foundation of China (61572118 and 61370075).

## References

Abadi, D. J., Marcus, A., Madden, S. & Hollenbach, K. 2007. Scalable semantic web data management using vertical partitioning. In *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. *VLDB Journal* **18**(2), 385–406.

Angles, R., Boncz, P. A., Larriba-Pey, J.-L., Fundulaki, I., Neumann, T., Erling, O., Neubauer, P., Martinez-Bazan, N., Kotsev, V. & Toma, I. 2014. The Linked Data Benchmark Council: a graph and RDF industry benchmarking effort. *SIGMOD Record* **43**(1), 27–31.

Angles, R. & Gutierrez, C. 2005. Querying RDF data from a graph database perspective. In *Proceedings of the Second European Semantic Web Conference*, 346–360.

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* **III**, 220–227.

Apweiler, R., Bairoch, A., Wu, C. H., Barker, W. C., Boeckmann, B., Ferro, S., Gasteiger, E., Huang, H., Lopez, R., Magrane, M., Martin, M. J., Natale, D. A., O'Donovan, C., Redaschi, N. & Yeh, L. S. 2004. UniProt: the universal protein knowledge base. *Nucleic Acids Research* **32**, D115–D119.

Berners-Lee, T., Hendler, J. & Lassila, O. 2001. The semantic web. *Scientific American* **284**(5), 34–43.

Bishop, B., Kiryakov, A., Ognyanoff, D., Peikov, I., Tashev, Z. & Velkov, R. 2011. OWLIM: a family of scalable semantic repositories. *Semantic Web* **2**(1), 1–10.

Bishop, B., Kiryakov, A., Tashev, Z., Damova, M. & Simov, K. I. 2012. OWLIM reasoning over FactForge. In *Proceedings of the 1st International Workshop on OWL Reasoner Evaluation*.

Bizer, C., Heath, T. & Berners-Lee, T. 2009. Linked data—the story so far. *International Journal of Semantic Web and Information Systems* **5**(3), 1–22.

[37] http://www.stardog.com/

[38] http://www.oracle.com/technetwork/database/options/spatialandgraph/overview/rdfsemantic-graph-1902016.html

https://doi.org/10.1017/S0269888916000217 Published online by Cambridge University Press

Bizer, C. & Schultz, A. 2009. The Berlin SPARQL benchmark. *International Journal on Semantic Web and Information Systems* **5**(2), 1–24.

Bonstrom, V., Hinze, A. & Schweppe, H. 2003. Storing RDF as a graph. In *Proceedings of the First Conference on Latin American Web Congress*, 27–36.

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. In *Proceedings of the 2013 ACM International Conference on Management of Data*, 121–132.

Broekstra, J., Kampman, A. & van Harmelen, F. 2002. Sesame: a generic architecture for storing and querying RDF and RDF schema. In *Proceedings of the 2002 International Semantic Web Conference*, 54–68.

Chang, F., Dean, J., Ghemawat, S., Hsieh, W. C., Wallach, D. A., Burrows, M., Chandra, T., Fikes, A. & Gruber, R. E. 2008. BigTable: a distributed storage system for structured data. *ACM Transactions on Computer Systems* **26**(2), 4:1–4:26.

Chao, C.-M. 2007a. An object-oriented approach for storing and retrieving RDF/RDFS documents. *Tamkang Journal of Science and Engineering* **10**(3), 275–286.

Chao, C.-M. 2007b. An object-oriented approach to storage and retrieval of RDF/XML documents. In *Proceedings of the 19th International Conference on Software Engineering & Knowledge Engineering*, 586–591.

Chebotko, A., Abraham, J., Brazier, P., Piazza, A., Kashlev, A. & Lu, S. 2013. Storing, indexing and querying large provenance data sets as RDF graphs in Apache HBase. In *Proceedings of IEEE Ninth World Congress on Services*, 1–8.

Choi, P., Jung, J. & Lee, K.-H. 2013. RDFChain: chain centric storage for scalable join processing of RDF graphs using MapReduce and HBase. In *Proceeding of the 2013 International Semantic Web Conference*, 249–252.

Cudre-Mauroux, P., Enchev, I., Fundatureanu, S., Groth, P., Haque, A., Harth, A., Keppmann, F. L., Miranker, D. P., Sequeda, J. F. & Wylot, M. 2013. NoSQL databases for RDF: an empirical evaluation. In *Proceedings of the 12th International Semantic Web Conference*, 310–325.

DeCandia, G., Hastorun, D., Jampani, M., Kakulapati, G., Lakshman, A., Pilchin, A., Sivasubramanian, S., Vosshall, P. & Vogels, W. 2007. Dynamo: Amazon's highly available key-value store. In *Proceedings of the 21st ACM Symposium on Operating Systems Principles*, 205–220.

Dell'Aglio, D., Calbimonte, J.-P., Balduini, M., Corcho, O. & Valle, E. D. 2013. On correctness in RDF stream processor benchmarking. In *Proceedings of the 12th International Semantic Web Conference*, 326–342.

Duan, S., Kementsietsidis, A., Srinivas, K. & Udrea, O. 2011. Apples and oranges: a comparison of RDF benchmarks and real RDF datasets. In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of Data*, 145–156.

Erling, O. & Mikhailov, I. 2007. RDF support in the Virtuoso DBMS. In *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*, De Virgilio, R., Giunchiglia, F. & Tanca, L. (eds). Springer-Verlag, 501–519.

Franke, C., Morin, S., Chebotko, A., Abraham, J. & Brazier, P. 2011. Distributed semantic web data management in HBase and MySQL Cluster. In *Proceedings of the 2011 IEEE International Conference on Cloud Computing*, 105–112.

Garbis, G., Kyzirakos, K. & Koubarakis, M. 2013. Geographica: a benchmark for geospatial RDF stores. In *Proceedings of the 12th International Semantic Web Conference*, 343–359.

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.

Gueret, C., Kotoulas, S. & Groth, P. 2011. TripleCloud: an infrastructure for exploratory querying over web-scale RDF data. In *Proceedings of the 2011 IEEE/WIC/ACM International Joint Conference on Web Intelligence and Intelligent Agent Technology—Workshops*, 245–248.

Guo, Y., Pan, Z. & Heflin, J. 2005. LUBM: a benchmark for OWL knowledge base systems. *Journal of Web Semantics* **3**(2–3), 158–182.

Harris, S. & Gibbins, N. 2003. 3store: efficient bulk RDF storage. In *Proceedings of the First International Workshop on Practical and Scalable Semantic Systems*.

Harris, S., Lamb, N. & Shadbolt, N. 2009. 4store: the design and implementation of a clustered RDF store. In *Proceedings of the 5th International Workshop on Scalable Semantic Web Knowledge Base Systems*, 94–109.

Harris, S. & Shadbolt, N. 2005. SPARQL query processing with conventional relational database systems. In *Proceedings of the International Workshop on Scalable Semantic Web Knowledge Base Systems*, 235–244.

Harth, A., Umbrich, J., Hogan, A. & Decker, S. 2007. YARS2: a federated repository for querying graph structured data from the web. In *Proceedings of the 6th International Semantic Web Conference*, 211–224.

Hassanzadeh, O., Kementsietsidis, A. & Velegrakis, Y. 2012. Data management issues on the semantic web. In *Proceedings of the 2012 IEEE International Conference on Data Engineering*, 1204–1206.

Hayes, J. & Gutierrez, C. 2004. Bipartite graphs as intermediate model for RDF. In *Proceedings of the 2004 International Semantic Web Conference*, 47–61.

https://doi.org/10.1017/S0269888916000217 Published online by Cambridge University Press

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

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

Husain, M. F., Doshi, P., Khan, L. & Thuraisingham, B. 2009. Storage and retrieval of large RDF graph using Hadoop and MapReduce. In *Proceedings of the First International Conference on Cloud Computing*, 680–686.

Karvounarakis, G., Alexaki, S., Christophides, V., Plexousakis, D. & Scholl, M. 2002. RQL: a declarative query language for RDF. In *Proceedings of the 11th International Conference on World Wide Web*, 592–603.

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

Kim, H. S., Ravindra, P. & Anyanwu, K. 2012. Scan-sharing for optimizing RDF graph pattern matching on MapReduce. In *Proceedings of the 2012 IEEE Fifth International Conference on Cloud Computing*, 139–146.

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

Kolas, D. 2008. A benchmark for spatial semantic web systems. In *Proceedings of the 2008 International Workshop on Scalable Semantic Web Knowledge Base Systems*.

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

Lee, K. & Liu, L. 2013. Scaling queries over big RDF graphs with semantic hash partitioning. *Proceedings of the VLDB Endowment* **6**(14), 1894–1905.

Le-Phuoc, D., Dao-Tran, M., Pham, M.-D., Boncz, P., Eiter, T. & Fink, M. 2012. Linked stream data processing engines: facts and figures. In *Proceedings of the 11th International Semantic Web Conference*, 300–312.

Levandoski, J. J. & Mokbel, M. F. 2009. RDF data-centric storage. In *Proceedings of the 2009 IEEE International Conference on Web Services*, 911–918.

Libkin, L., Reutter, J. L. & Vrgoc, D. 2013. Trial for RDF: adapting graph query languages for RDF data. In *Proceedings of the 32nd ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, 201–212.

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*, De Virgilio, R., Guerra, F. & Velegrakis, Y. (eds). Springer-Verlag, 31–60.

Manola, F. & Miller, E. 2004. RDF primer, W3C Recommendation. `http://www.w3.org/TR/2004/REC-rdf-primer-20040210/`.

Matono, A., Amagasa, T., Yoshikawa, M. & Uemura, S. 2005. A path-based relational RDF database. In *Proceedings of the 16th Australasian Database Conference*, 95–103.

Matono, A. & Kojima, I. 2012. Paragraph tables: a storage scheme based on RDF document structure. In *Proceedings of the 23rd International Conference on Database and Expert Systems Applications*, 231–247.

McBride, B. 2002. Jena: a semantic web toolkit. *IEEE Internet Computing* **6**(6), 55–59.

Minack, E., Siberski, W. & Nejdl, W. 2009. Benchmarking fulltext search performance of RDF stores. In *Proceedings of the 6th European Semantic Web Conference*, 81–95.

Morsey, M., Lehmann, J., Auer, S. & Ngomo, A. C. N. 2011. DBpedia SPARQL benchmark-performance assessment with real queries on real data. In *Proceedings of the 10th International Semantic Web Conference*, 454–469.

Morsey, M., Lehmann, J., Auer, S. & Ngomo, A. C. N. 2012. Usage-centric benchmarking of RDF triple stores. In *Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence*, 2134–2140.

Mulay, K. & Kumar, P. S. 2012. SPOVC: a scalable RDF store using horizontal partitioning and column oriented DBMS. In *Proceedings of the 4th International Workshop on Semantic Web Information Management*.

Neumann, T. & Moerkotte, G. 2011. Characteristic sets: accurate cardinality estimation for RDF queries with multiple joins. In *Proceedings of the 27th International Conference on Data Engineering*, 984–994.

Neumann, T. & Weikum, G. 2008. RDF-3X: a RISC-style engine for RDF. *Proceedings of the VLDB Endowment* **1**(1), 647–659.

Neumann, T. & Weikum, G. 2010. The RDF-3X engine for scalable management of RDF data. *The VLDB Journal* **19**(1), 91–113.

Owens, A., Seaborne, A., Gibbins, N. & Schraefel, M. 2009. Clustered TDB: a clustered triple store for Jena. In *Proceedings of the 13th International Conference on World Wide Web*.

Papailiou, N., Konstantinou, I., Tsoumakos, D., Karras, P. & Koziris, N. 2013. $H_2$RDF+: high-performance distributed joins over large-scale RDF graphs. In *Proceedings of the 2013 IEEE International Conference on Big Data*, 255–263.

Papailiou, N., Konstantinou, I., Tsoumakos, D. & Koziris, N. 2012. $H_2$RDF: adaptive query processing on RDF data in the cloud. In *Proceedings of the 21st World Wide Web Conference*, 397–400.

Patni, H., Henson, C. & Sheth, A. 2010. Linked sensor data. In *Proceedings of the 2010 International Symposium on Collaborative Technologies and Systems*, 362–370.

https://doi.org/10.1017/S0269888916000217 Published online by Cambridge University Press

Przyjaciel-Zablocki, M., Schatzle, A., Hornung, T., Dorner, C. & Lausen, G. 2012. Cascading map-side joins over HBase for scalable join processing. In *CoRR 2012*.

Ravindra, P., Kim, H. S. & Anyanwu, K. 2011. An intermediate algebra for optimizing RDF graph pattern matching on MapReduce. In *Proceedings of the 8th Extended Semantic Web Conference*, 46–61.

Rohloff, K. & Schantz, R. E. 2011. Clause-iteration with MapReduce to scalably query datagraphs in the SHARD graph-store. In *Proceedings of the Fourth International Workshop on Data-Intensive Distributed Computing*, 35–44.

Sakr, S. & Al-Naymat, G. 2009. Relational processing of RDF queries: a survey. *SIGMOD Record* **38**(4), 23–28.

Salvadores, M., Correndo, G., Harris, S., Gibbins, N. & Shadbolt, N. 2011. The design and implementation of minimal RDFS backward reasoning in 4store. In *Proceedings of the 8th Extended Semantic Web Conference*, 139–153.

Salvadores, M., Correndo, G., Omitola, T., Gibbins, N., Harris, S. & Shadbolt, N. 2010. 4s-reasoner: RDFS backward chained reasoning support in 4store. In *Proceedings of the 2010 IEEE/WIC/ACM International Conference on Web Intelligence and International Conference on Intelligent Agent Technology—Workshops*, 261–264.

Schmidt, M., Hornung, T., Kuchlin, N., Lausen, G. & Pinkel, C. 2008. An experimental comparison of RDF data management approaches in a SPARQL Benchmark scenario. In *Proceedings of the 7th International Semantic Web Conference*, 82–97.

Schmidt, M., Hornung, T., Lausen, G. & Pinkel, C. 2009. $SP^2$Bench: a SPARQL Performance Benchmark. In *Proceedings of the 25th International Conference on Data Engineering*, 222–233.

Sequeda, J. F., Tirmizi, S. H., Corcho, O. & Miranker, D. P. 2011. Survey of directly mapping SQL databases to the semantic web. *Knowledge Engineering Review* **26**(4), 445–486.

Sidirourgos, L., Goncalves, R., Kersten, M. L., Nes, N. & Manegold, S. 2008. Column-store support for RDF data management: not all swans are white. *Proceedings of the VLDB Endowment* **1**(2), 1553–1563.

Sintek, M. & Kiesel, M. 2006. RDFBroker: a signature-based high-performance RDF store. In *Proceedings of the 3rd European Semantic Web Conference*, 363–377.

Sperka, S. & Smrz, P. 2012. Towards adaptive and semantic database model for RDF data stores. In *Proceedings of the Sixth International Conference on Complex, Intelligent, and Software Intensive Systems*, 810–815.

Stein, R. & Zachrias, V. 2010. RDF on cloud number nine. In *Proceedings of the 4th Workshop on New Forms of Reasoning for the Semantic Web: Scalable & Dynamic*, 11–23.

Stonebraker, M., Abadi, D. J., Batkin, A., Chen, X., Cherniack, M., Ferreira, M., Lau, E., Lin, A., Madden, S., O'Neil, E., Rasin, A., Tran, N. & Zdonik, S. 2005. C-Store: a column-oriented DBMS. In *Proceedings of the 31st International Conference on Very Large Data Bases*, 553–564.

Suchanek, F. M., Kasneci, G. & Weikum, G. 2008. YAGO: a large ontology from Wikipedia and WordNet. *Journal of Web Semantics* **6**(3), 203–217.

Sun, J. L. & Jin, Q. 2010. Scalable RDF store based on HBase and MapReduce. In *Proceedings of the 3rd International Conference Advanced Computer Theory and Engineering*, V1-633–V1-636.

Theoharis, Y., Christophides, V. & Karvounarakis, G. 2005. Benchmarking database representations of RDF/S stores. In *Proceedings of the 4th International Semantic Web Conference*, 685–701.

Urbani, J., Kotoulas, S., Oren, E. & Harmelen, F. 2009. Scalable distributed reasoning using MapReduce. In *Proceedings of the 8th International Semantic Web Conference*, 634–649.

Wang, Y., Du, X. Y., Lu, J. H. & Wang, X. F. 2010. FlexTable: using a dynamic relation model to store RDF data. In *Proceedings of the 15th International Conference on Database Systems for Advanced Applications*, 580–594.

Weiss, C., Karras, P. & Bernstein, A. 2008. Hexastore: sextuple indexing for semantic web data management. *Proceedings of the VLDB Endowment* **1**(1), 1008–1019.

Wilkinson, K. 2006. Jena property table implementation. Technical report HPL-2006-140, HP Labs.

Wilkinson, K., Sayers, C., Kuno, H. A. & Reynolds, D. 2003. Efficient RDF storage and retrieval in Jena2. In *Semantic Web and Databases Workshop*, 131–150.

Wolff, B. G. J., Fletcher, G. H. L. & Lu, J. J. 2015. An extensible framework for query optimization on TripleT-based RDF stores. In *Proceedings of the Workshops of the EDBT/ICDT 2015 Joint Conference*, 190–196.

Zeng, K., Yang, J. C., Wang, H. X., Shao, B. & Wang, Z. Y. 2013. A distributed graph engine for web scale RDF data. *Proceedings of the VLDB Endowment* **6**(4), 265–276.

Zhang, X. F., Chen, L. & Wang, M. 2012a. Towards efficient join processing over large RDF graph using MapReduce. In *Proceedings of the 24th International Conference on Scientific and Statistical Database Management*, 250–259.

Zhang, Y., Pham, M. D., Corcho, O. & Calbimonte, J. P. 2012b. SRBench: a streaming RDF/SPARQL benchmark. In *Proceedings of the 11th International Semantic Web Conference*, 641–657.

https://doi.org/10.1017/S0269888916000217 Published online by Cambridge University Press