Journal of Big Data

RESEARCH Open Access

An empirical study on the evaluation of the RDF storage systems

Bilal Ben Mahria*, Ilham Chaker and Azeddine Zahi

*Correspondence:
bilal.benmahria@usmba.ac.ma
Sidi Mohamed Ben Abdellah University, 2202 Fez, Morocco

Abstract

In this paper, we introduce three new implementations of non-native methods for storing RDF data. These methods named RDFSPO, RDFPC and RDFVP, are based respectively on the statement table, property table and vertical partitioning approaches. As important, we consider the issue of how to select the most relevant strategy for storing the RDF data depending on the dataset characteristics. For this, we investigate the balancing between two performance metrics, including load time and query response time. In this context, we provide an empirical comparative study between on one hand the three proposed methods, and on the other hand the proposed methods versus the existing ones by using various publicly available datasets. Finally, in order to further assess where the statistically significant differences appear between studied methods, we have performed a statistical analysis, based on the non-parametric Friedman test followed by a Nemenyi post-hoc test. The obtained results clearly show that the proposed RDFVP method achieves highly competitive computational performance against other state-of-the-art methods in terms of load time and query response time.

Keywords: RDF data, Non-native methods, Statement table, Property table, Vertical portioning, Friedman test, Nemenyi test, Load time, Query response time

Introduction

We are witnessing a paradigm shift, where the ever-growing of huge amount of data has created unprecedented challenges for traditional data processing systems [5]. In fact, managing and processing this huge and growing volume of data represents a tremendous challenge, considering their size and complexity, load time, and the desirable response time [18]. Consequently, such challenges made the traditional systems need new strategies to efficiently storing and retrieving data at an impressive rate [15]. As a response to this call, the semantic repositories systems (SR) has been proposed, which combine the characteristics of database management systems (DBMS) and the inference engines to support efficient managing of data [18].

A semantic repository is a database management system that allows storing, querying, and managing structured data. Indeed, semantic repository is still not a largely adopted term and it is often referred to semantic graph database, reasoner, ontology server, semantic store, metadata store, RDF database, and RDF triplestore [41]. Compared to the traditional DBMSs such as relational databases, the major benefit of using

the semantic repositories is the usage of semantic data paradigm, called RDF data model [50]. In fact, the RDF data model gives the ability to change the data schema on the fly without interfering with the data, discover new facts and build new data based on semantic rules using the inference capability, and seamlessly integrate data that comes from diverse sources [26].

Generally, semantic repository can be categorized into two categories: native and non-native storage systems [43]. More precisely, the native storage systems can be broadly classified as persistent disk-based [32, 36, 51] and main-memory based systems [6, 39, 52]. Within the non-native systems, the developed methods fall roughly into one of three categories [22]: statement tables [7, 20, 29, 42], property table [4, 30, 53], and vertical partitioning [2, 38, 47]. Even though all these approaches have been successfully applied for storing RDF data, the problem of selecting the most appropriate system for storing and querying the RDF data, has not been sufficiently addressed and it is a subject of much interest currently.

In the context of this paper, we consider the issue of how to choose an appropriate strategy for storing the RDF data. Generally, different RDF datasets and queries often require different storage solutions. Therefore, the choice of the most appropriate and efficient storage strategy needs balancing between loading and query response time performance, considering several factors, related to the data scalability and the reasoning capability [18]. In this respect, several works have been developed to study the performance of RDF stores [8, 9, 11, 19, 28, 35, 43, 45, 55] and all these studies suggested the use of query response time as a performance metric for choosing the appropriate RDF storage systems. However, selecting the most suitable storage system is essentially related not only to the query response time metric but also to the load time metric.

Different from the previous research efforts, a key benefit of our work is that it allows to categorize and empirically compare the non-native and native systems, based on different performance metrics, namely, the query response time and load time. What is more, three implementations for storing RDF data based on the non-native approach, named RDFSPO, RDFPC and RDFVP, have been proposed in this paper. These methods are respectively based on the statement table, property table and vertical partitioning approaches.

Thus, to demonstrate the usefulness and the performance of the proposed implementations, we have conducted several comparative experiments between the proposed methods and the existing ones. In this respect, we have considered the native and non-native store systems. On one hand, the non-native systems used in this study are three proposed implementations, which are the RDFSPO, the RDFPC and the RDFVP as well as the existing method Jena SDB.[1] On the other hand, the representatives of the native systems that have been chosen for comparison are RDF4JM,[2] RDF4JD[2], and TDB.[3] In addition, to ensure different reasoning capabilities and data scalability, these systems differ in their storage mechanisms and query response manner. More precisely, we have evaluated one memory-based system (RDF4JM), two disk-based systems (RDF4JD and

[1] https://jena.apache.org/documentation/sdb/.

[2] https://rdf4j.eclipse.org/.

[3] https://jena.apache.org/documentation/tdb/.

TDB), and four systems with persistent storage (RDFSPO, RDFPC, RDFVP and SDB). The experimental results are evaluated in term of the load time and query response time. Furthermore, to bring a significant reason to the obtained results, we have applied a statistical analysis based on the Friedman Test [16, 17] followed by a Nemenyi post hoc test [16, 17] to further explore which method performs statistically. To the best of our knowledge, no other studies report such statistical analysis for supporting the validity of the obtained results in this field.

To summarize, the major contribution of this paper includes the following aspects: (1) Introduce three proposed non-native implementations for storing the RDF data. (2) We present the most appropriate performance aspects for selecting the relevant RDF storage strategies, and analyze their advantages and drawbacks. (3) We provide an empirical study based on the statistical analysis, including Friedman and Nemenyi post hoc Test.

The outline of this paper is demonstrated as follows. In "Classification Of RDF storage systems" Section, we start by providing a classification of RDF storage systems. In "Our proposed implementations" Section, we detail our proposed implementations for Storing RDF data. In "Experiments setup" Section, we describe the aforementioned experiments. "Experimental results and discussion" Section is devoted to introduce the experiment studies and discussions. Finally, "Conclusion" Section concludes the paper and suggests directions for future works.

Classification of RDF storage systems

Among the plethora of existing systems for storing the RDF data, we can distinguish between solutions that are implementing their own storage backend, denoted as native systems, and those that are using an existing database management system, denoted as non-native systems. A classification of RDF data storage techniques is proposed in Fig. 1.

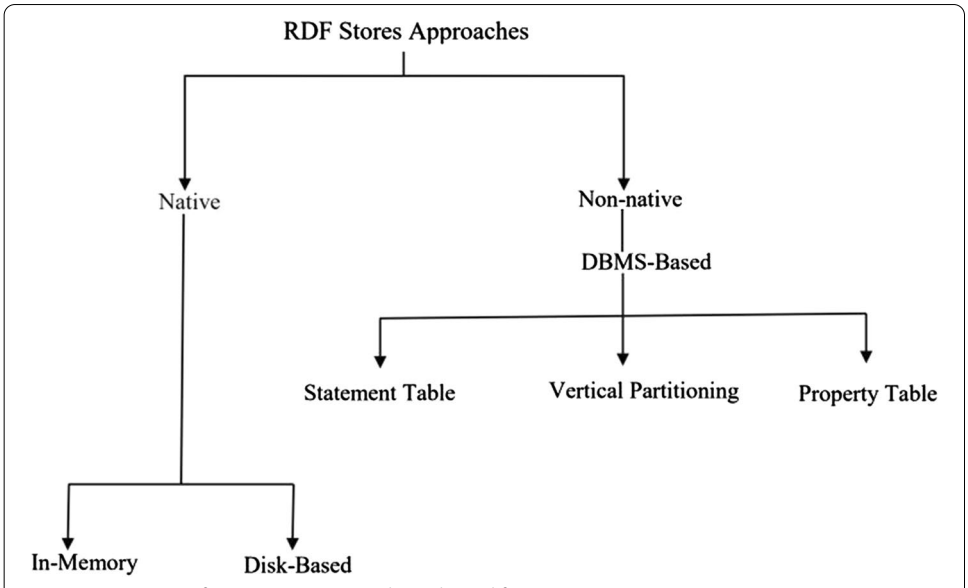

Fig. 1 A taxonomy of RDF storage approaches adopted from [22]

Native systems

Native systems denote systems that avoid the benefit from existing systems for storing the RDF data. They are constructed from scratch based on indexing technique specific to RDF data model [43]. In fact, the native systems provide greater flexibility than traditional databases and reduce the data load time. In addition, the native systems offer many other traditional database functions, such as transaction processing, access control, logging and data recovery. More precisely, these systems can be broadly classified as persistent disk-based and main-memory systems [43].

The persistent disk-based storage is a way to store RDF data permanently on file system by using the most influential indexing techniques, such as B + tree [48], AVL[54] and B − tree[31]. Among the existing solutions we can mention [1, 10, 12, 32, 36, 51]. It is important to notice that, reading from and writing to disks slow the search process to an unacceptable level and induce an important performance bottleneck [15].

To overcome this issue, the in-memory solutions were used. The in-memory based storage allocates a certain amount of the main memory (RAM) to store the whole RDF data. When working on RDF data stored in main memory, some of the most factors that must be covered are the loading and parsing of RDF file [15]. Therefore, the RDF store that uses the in-memory approach must have a memory efficient data representation that leaves enough space for the operation of search algorithms. The following works fall in this category [6, 10, 23, 27, 34, 44, 52, 53].

Non-native systems

The non-native stores refer particularly to systems that use the relational database management systems (RDBMS) or other related systems to store RDF data permanently. Currently, RDBMS is widely considered to be the best performing place for persistent RDF data due to the great effort achieved in developing solutions that make the storage of RDF data efficient, scalable and robust [37]. In order to discuss the RDF stores over RDBMS, the first issue to be covered is how to map RDF data to relational tables. In this respect, there are three storage strategies that are: statement table, property table, and vertical partitioning [15].

The Statement table [35] is the most straightforward way to map RDF data to a relational database. As depicted in Fig. 2, it consists of creating a table with three columns (subject, predicate, object), where each row separately corresponds to an RDF statement. In fact, the way that all the data is combined into a large single table brings the problem of low efficiency of the query, since a simple SELECT query needs a large number of self-joins. Specifically, if the number of RDF statements increase, the query response time will increase with the increment of self-joins times. In order to improve the efficiency of queries, the indexes techniques are then added for each of the column for reducing the cost of self-join query [35]. However, the storage of RDF triples in a single table make the queries very slow to execute and may overtake the size of the main memory as indicated in [36]. Many early RDF stores use statement table approach, such as [7, 20, 27, 29, 40].

Property table (PT) [3, 35] has been proposed later and can be classified into two types: clustered property table (CP) and property-class table (PC). The former contains clustered of properties that tend to describe the same subject (Fig. 3). The latter exploits the "rdf: type" predicate to cluster similar sets of subjects in the same table (Fig. 4).

Subject	Predicate	Object
ID1	type	Book
ID1	title	"A Developer's Guide to the Semantic Web "
ID1	author	"Liyan Yu"
ID1	date	"2014"
ID2	Type	Movie
ID2	Title	"A Beatiful Mind"
ID2	Actor	"Russel Crowe"
ID2	date	"2001"
ID2	Language	"English"
ID3	Type	Book
ID3	title	"Unlimited Memory"
ID3	language	"English"
ID4	Type	Journal
ID4	Title	"Journal On Data Semantics"
ID5	Title	"One Man's Dream"
ID5	Date	"1993"
ID5	Type	Movie
ID6	type	Book
ID6	date	"2015"

Fig. 2 Statement table approach

Subject	Type	Title	Date
ID1	Book	"A Developer's Guide to the Semantic Web "	"2014"
ID2	Movie	"A Beatiful Mind"	"2001"
ID3	Book	"Unlimited Memory"	NULL
ID4	Journal	"Journal On Data Semantics"	NULL
ID5	Movie	"One Man's Dream"	"1993"
ID6	Book	NULL	"2015"

Fig. 3 Clustered property table (CP)

Class: Book

Subject	Title	Author	Date
ID1	"A Developer's Guide to the Semantic Web "	"Liyan Yu"	"2014"
ID3	"Unlimited Memory"	NULL	NULL
ID6	NULL	NULL	"2015"

Class: Movie

Subject	Title	Actor	Date
ID2	"A Beatiful Mind"	"Russel Crowe"	"2001"
ID5	"One Man's Dream"	NULL	"1993"

Fig. 4 Property class approach (PC)

Generally, the main idea is to discover clusters of subjects often appearing with the same set of properties. This approach has been presented in several works like [4, 10, 30, 53]. In fact, the immediate consequence of applying PT is that the complex SPARQL queries can be retrieved without an expensive self-joins. However, as indicated in [3] the PT approach has three major drawbacks. The first one is the problem of generating many null values, which enforces a substantial performance overhead. The second one is that the PT cannot handle the multi-valued properties. The third one refers to the complex queries that prove that the PT is still expensive, because most of these queries need union clauses and joins to collect data from several tables. In this respect, an alternative solution has been proposed, which is vertical partitioning approach.

Vertical partitioning (VP) [3, 35] refers to the vertical division of RDF statement based on predicate. As depicted in Fig. 5, the RDF triples are divided on n tables with two columns, where n is the number of unique predicate in the RDF dataset. In each of these tables, the first column involves the subjects that are defined by the predicate and the second column consist of the object values of those subjects. It is important to mention that the name of predicate can be used as the table name. Compared to the PT approach, the VP approach provides a support for multi-valued attributes, and the null values are simply omitted from the table. In addition, for a given query only the table corresponding to the properties involved in that query requires to be read, and no clustering algorithm is needed to split the RDF triples into two-column. The VP approach is used in several works as [3, 21, 38].

Our proposed implementations

In this work, we proposed an implementation of the three approaches presented in the previous section. These implementations are named RDFSPO, RDFPC, and RDFVP, which are based respectively on the statement table, Property tables and vertical partitioning approaches.

RDF subject-predicate-object method (RDFSPO)

The RDFSPO method is a non-native store that consists of storing the RDF data using the relational database as backend. Concisely, the main algorithm for implementing the RDFSPO method is depicted in Algorithm 1. Generally, the RDFSPO method mapped the RDF data directly onto a three column wide table SPO (subject-predicate-object). It important to mention that our algorithms for mapping RDF data onto the relational database are automatic and generic.

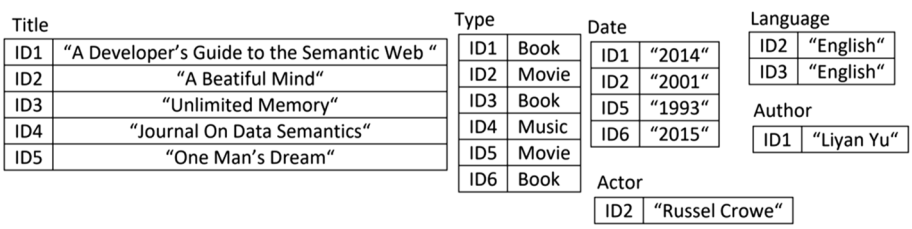

Fig. 5 Vertical partitioning approach (VP)

Algorithm 1: RDF Subject-Predicate-Object Method

```
Input: RDF dataset
Output: RDF dataset stored in RDB.
Var:
       iter: stmtIterator; model: Model; in: InputStream; t: Triple; stmt: Statement;
subject: Resource; predicate: Property; object: RDFNode;
Begin
1:     DBUTILS.createSPOTable();
2:     model← ModelFactory.createDefaultModel();
3:     in←FileManager.get().open(inputFileName);
4:     if(in is NULL) then
5:         ThrowExcpetion();
6:     Else
7:         Model.read(in, " ");
8:     End if
9:     iter←model.displayAllStatment();
10:    while(iter.hasNext())
11:        stmt←iter.nextStatment();
12:        subject←stmt.getSubject();
13:        predicate←stmt.getPredicate();
14:        object←stmt.getObject();
15:        if(object ∈ Resource) then
16:           t.setObjet(Object.toString());
17:        else
18:           t.setObject(object.toString);
19:         end if
20:    end while
21:     DBUTILS.insertIntoSPOTable(t);
22: END
```

RDF property clustering method (RDFPC)

As we aforementioned in the previous section, the statement table approach has several obvious drawbacks. More specifically, the RDFSPO method generates large amount of unnecessary replication of information that is appeared in several rows. In this respect, in order to bypass all the limitations introduced by the RDFSPO, another implementation is proposed. The proposed algorithm for this implementation is given in Algorithm 2.

Algorithm 2: RDF Clustered Property Method

```
Input: RDF dataset
Output: RDF dataset stored in RDB.
Var:
       properties: Hashset<Property>;list: List<RDFNode>;stmt: Statement; subject: Resource;
       predicate,pre: Property; object,obj: RDFNode; s,str: table
Begin
1:     s←node.toString().split("/");
2      last←s.length-1;
3:     if(s[last].contains("#") then
4:        scrpitSQL←createTable(str[1])
5:     else
6:        scriptSQL←createTable(str[last])
7:     end if
8:     stmt←model.displayAllStatment(subject,null,null)
9:     while(stmt.hasNext())
10:       if(s[last].contains("#")) then
11:         str←s[last].split("#");
12:         insertScriptSQL←insertPC(str[1]);
13:       else
14:           insertScriptSQL←insertPC(str[0]);
15:       end if
16:       stmt1←model.displayAllStatment(subject,null,nul);
17:       while(stmt1.hasNext())
18:          properties.add(pre);
19:           list.add(obj);
20:       end while
21:       for(Property pr: properties)
22:          if(pr.getLocalName().toString().equals("type"));
23:              scriptSQL←insertScriptSQL.replace("["," ");
24:              scriptSQL←insertScriptSQL.replace("]"," ");
25:           end if
26:      end For
27:           file←writeToFile(insertScriptSQL)
28:    end while
29:    DBUTILS.insertIntoClusterdTable(file)
30:  End
```

Algorithm 3: RDF Vertical Partitioning Method

```
Input: RDF dataset
Output: RDF dataset stored in RDB.
Var:
      iter: stmtIterator; model: Model;in:InputStream;
      t: Triple;stmt: Statement;subject: Resource;
      predicate: Property;object: RDFNode;listOfProperties: List
Begin
1:    model←ModelFactory.createDefaultModel();
2:    in←FileManager.get().open(inputFileName);
3:    if(in is NULL) then
4:       ThrowException()
5:    else
6:       Model.read(in," ");
7:    end if
8:    listOfProperties←listOfPredicate();
9:    for each pr in listOfProperties
10:      tblName←pr.getLocalName().toString();
11:      DBUtils.createTabe(tbleName);
12:   end for
13:   stmt←DisplayAllStatement(null,predicate,null)
14:   while(stmt.hasNext())
15:      state←stmt.nextStatment();
16:      subject←state.getSubject();
17:      object←state.getObject();
18:      t.setSubject(subject);
19       t.setObject(object);
20       DBUtils.isertIntoTable(tblName,t);
21    end while
22    DBUtils.createNewTabe(tbleName);
23    tstListStatmentByProperty(predicate);
24   end
```

RDF vertical partitioning method (RDFVP)

The RDFVP method is an alternative to the RDFPC implementation, where we can omit all the limitation of the RDFPC method (based on the property table approach) and speed up queries over the RDF dataset. The implementation of the RDFVP method is depicted in Algorithm 3.

Experiments setup

A set of appropriate experiments have been arranged in order to study the performance characteristics of the RDF data management systems, analyzed in the previous sections. For this reason, a set of well-known from the literature datasets are selected. The information detailed of these datasets is summarized in Table 1. As proof of concept, we ran our experimental study against four popular RDF stores including Jena TDB [15], Jena SDB [15], RDF4jM [10], and RDF4jD [10] in addition to three proposed methods that use the MYSQL backend, which are RDFSPO, RDFPC and RDFVP. In this context, we have collected 17 datasets for testing the dimensional performance of these systems. Finally, it is important to note that all the experimental simulations were conducted on a personal computer under Windows 10, with Intel core i7 2.70 GHZ processor and 16 GB RAM.

Table 1 The basic statistics of datasets

Datasets	#Triples	#Subjects	#Predicates	#Objects
allSWGroups[a]	345	56	19	165
Wikimovies[a]	505	157	22	121
NATS[b]	4036	266	33	568
Tissues[c]	4113	1426	7	2663
Locations[c]	5374	949	17	2885
VSRR[d]	6221	500	13	1387
AQMNEHTN[e]	8000	500	16	1456
CDI[f]	13,462	1000	29	1493
DHDS[e]	14,802	1000	30	2477
DASH[e]	16,152	1000	33	4635
Journals[c]	37,581	4473	12	29,358
Diseases[c]	69,487	18,061	8	36,185
Enzyme[c]	84,398	14,633	14	47,391
GeneSymbol[c]	1,19,485	17,077	7	85,345
Protein[b]	1,60,537	14,635	8	1,17,034
swdf[g]	2,42,249	23,308	171	76,529
Geo_coordinates_en[h]	1,569,180	4,96,990	4	1,045,824

[a] https://www.wikidata.org

[b] https://www.cdc.gov/tobacco/data_statistics/surveys/nats/index.htm

[c] https://www.uniprot.org/

[d] https://www.cdc.gov/nchs/nvss/vsrr/drug-overdose-data.htm

[e] https://healthdata.gov/search/type/dataset

[f] https://www.cdc.gov/cdi/index.html

[g] https://lod-cloud.net/dataset/semantic-web-dog-food

[h] https://wiki.dbpedia.org/Datasets

Description of existing RDF storage systems

This subsection is devoted to describe the four existing systems that we have used to demonstrate the efficiency of our proposed methods. These systems are: Jena TDB, Jena SDB, RDF4J main memory-based (RDF4JM), and RDF4J Disk-Based (RDF4JD). Table 2 is an overview of the existing storage systems that we used in this study.

The Jena SDB [15] is the non-native persistent triple store that uses a relational database for the storage and query RDF data. It adopts the statement table approach and basically used for several RDBMS, such as MYSQL, PostgreSQL, Oracle, SQL server and DB2. More precisely, the RDF statements stored are combined in tables with respectively three or four columns. In the former (i.e., table with three columns), an SPO (subject-predicate-object) primary key is created and additional PO and OS indexes are defined. In the latter (i.e., table with four columns), the primary key refers to GSPO where G represents the named graph and five additional indexes are presented: GPO, GOS, SPO, OS, and PO.

Table 2 Characteristics of proposed RDF storage systems

RDF storage system	Approach	Category	Reasoning strategy	Query language
Jena SDB	Statement table	Non-native	Backward chaining	SPARQL
Jena TDB	Disk-based	Native	Backward chaining	SPARQL
RDF4JD	Disk-based	Native	Forward chaining	SPARQL
RDF4JM	Main memory-based	Native	Forward chaining	SPARQL

The Jena TDB [15] is the native persistent triple store that uses the disk-based approach for retrieving and storing RDF data. It holds three composite indexes in the form of B + trees: subject-predicate-object (SPO), Predicate-object-subject (POS), object-subject-predicate (OSP). A dataset backed by TDB is stored in a single directory in the filing systems. This dataset consists of three components. The first one is the node table that stores the of RDF terms. The node table is also called a dictionary. The second one is the triple and quad indexes. Quad indexes are used for named graphs, while triple indexes for the default graph. The third one is the prefixes table that uses a node table and index for mapping prefixes to URIs.

RDF4J (formerly known as Sesame) [10] is an open source Java framework for storing, querying, and reasoning with RDF and RDF Schema. It can be used as a database for RDF and RDF Schema, or as a Java library for applications that need to work with RDF internally. It defined the necessary tools to parse, interpret, query, and store the RDF data, embedded in a separate database or in a remote server. Generally, RDF4J is a native RDF store that defined a set of database implementations including the main memory store (RDFJM) and disk-based store (RDF4JD).

Query implementation details

To test the query response time in the different RDF storage systems, we used 12 queries that we will detail later. There are two different ways to create a query. The first method is designing a query based on certain features and thereby evaluating the how those features work. The second method of designing the query is based more on the real world use cases. For our experimental study, we adopt some queries from the SP^2Bench [46] that are based on the second way combined with other queries that are constructed with the first way. Table 3 summarizes all the tested queries in this experimental study. Generally, these queries also vary according to their characteristics as shown in Table 4.

As depicted in Table 4, the tested queries fall in one of the three following categories: (1) star query: is the most frequently used type, it only consists of subject-subject joins where a join variable is represented by the subject piece of all the triple patterns involved in the query; (2) chain query: comprises subject-object joins where the triple patterns

Table 3 The tested queries implemented in this experimental study

Q1. Return the list of authors and titles of papers that have a rdf: type value proceeding
Q2. Select all proceedings papers with property swrc: pages or swrc: month or swrc: isbn
Q3. Extract all proceedings papers with properties dc: creator, swrc: booktitle, swrc: isPartOf, rdfs: seeAlso, dc: title, swrc: homepage, dc: subject and optionally swrc: abstract, considering their values
Q4. Return all proceedings papers that they are accepted in iswc conference 2008
Q5. Select all distinct pairs of paper author names for authors that have published in the same journal
Q6. Return all full papers and short papers accepted in the rule-ML conference 2011
Q7. Extract all proceedings papers with the machine learning topic
Q8. Extract 10 resources that are somehow related to each other
Q9. Return all the proceedings papers accepted in all conferences described in swdf dataset with the semantic web or machine learning topics
Q10. Return all subjects that stand in any relation to person "Annabel Bourde"
Q11. Compute authors that have published with Annabel Bourde or with an author that has published with Annabel Bourde
Q12. Extract all the name of persons and their papers including the type and the subject

Table 4 Characteristics of tested queries

Characteristic	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Q10	Q11	Q12
Simple query	×	–	–	–	–	–	–	×	–	×	–	–
filters	–	×	–	–	×		–	–	–	–	×	–
More than 6 patterns	–	–	×	–	×	×	–	–	×	–	–	×
Optional operator	–	–	×	–	–	–	–	–	–	–	–	–
Limit modifier	–	–	–	–	–	–	–	×	–	–	–	–
Offset modifier	–	–	–	–	–	–	–	×	–	–	–	–
Union operator	–	–	–	–	–	×	–	–	×	–	×	–
Distinct operator	–	–	–	×	×	–	×	–	×	–	×	×
Order by	–	–	×	–	–	–	–	–	–	–	–	–
And operator	×	×	×	×	×	×	×	×	×	–	×	×
Or operator	–	×	–	–	–	–	–	–	–	–	–	–
Type of query	SQ	SQ	SQ	SQ	TQ	ST	TQ	–	TQ	–	CQ	CQ

SQ start query; *TQ* tree query; *CQ* chain query

are consecutively linked like a chain. (3) Tree query: includes subject-subject joins and subject-object joins [33].

Experimental results and discussion

The main goal of this work is to evaluate the different RDF data storage systems whether native or non-native, to choose the most appropriate strategy for storing the RDF data based on specific characteristics. In this context, the experimental results are discussed as the following. Firstly, the evaluation of our proposed implementations in terms of loading time and query response time. Secondly, the evaluation of the existing systems always in terms of loading time and query response time. Finally, the comparison between the proposed and existing systems. More precisely, the load time metric is repeated 10 times and the average elapsed CPU time is computed, while the query response time is calculated by taking the average of response time of executing each query ten times consecutively. In our study, we used 12 queries that were defined to answer the real life questions. The dataset used in our analysis is the SWDF, which contains 2,42,249 triples.

Evaluation of the proposed RDF data storage systems

In the first part of this subsection, we present an evaluation of the three proposed systems, namely RDFSPO, RDFPC and RDFVP in terms of the loading time and query response time.

Load time metric

Considering the result presented in Fig. 6a, it is clearly seen that the RDFVP performs considerably better than the RDFPC and RDFSPO. On one hand, the fundamental idea behind the VP is partitioning data using fully decomposed storage model [13, 14]. Since the data comes from different tables of the same database is more manageable than a situation in which the same datasets are stored in a single table as in the case of the RDFSPO.

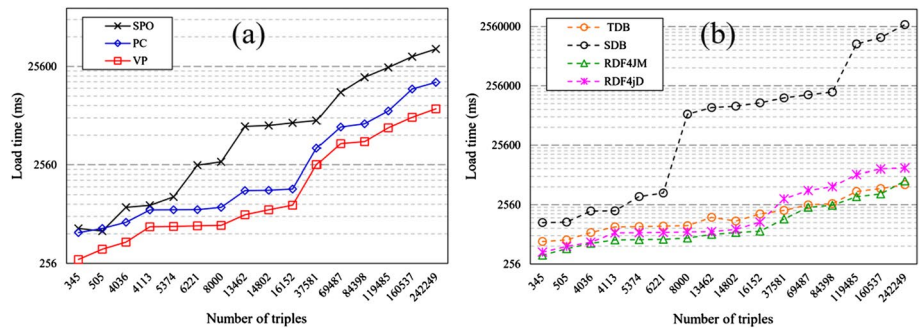

Fig. 6 Load time. **a** The load time of the proposed methods. **b** The existing methods

On the other hand, the reason behind why RDFVP performs better than the RDFPC is associated with the process used by RDFPC to load data. In fact, the RDFPC uses a clustering strategy based on the "rdf: type" property for storing the RDF data in relational database. This means that before loading the data, the RDFPC takes into consideration the RDF data clustering based on the rdf: type property. Consequently, the load time is computed as the sum of the time of clustering the RDF data and loading time.

Query response time

Considering the results presented in Fig. 7, RDFSPO is very slow to answer all queries compared to RDFPC and RDFVP. This is essentially due to the use of a single table for the storage of all triples. Indeed, when the number of triples increases, this single table may exceed main memory size [3]. Additionally, the complex queries with multiple patterns require many self-joins, resulting in poor performance. Another interesting observation that we can make about the RDFSPO is that it uses table scanning search to find the appropriate records, which leads to slowing the query execution time [3].

As depicted in Fig. 7, the RDFPC method performs better than the other tested methods. At the same time, it closely follows the proposed RDFVP method, which gives the best results for all the queries. The immediate consequence of the RDFPC method is that it can avoid the excessive number of self-joins generated by RDFSPO method. More precisely, while RDFPC method improves performance by reducing the number of self-joins and rdf: type predicate, it introduces complexity by storing useless information like null values, and can cause the loss of information with regard to handling the many-to-many relationship. Therefore, we can clearly see that the RDFVP method performs better than RDFPC method. On the whole, RDFVP provides a significant performance improvement by overcoming the limitations encountered in the RDFPC method. Consequently, it could be introduced as the best promising alternative to the existing models for retrieving data from repositories.

Evaluation of the existing RDF data storage systems

In a similar way to the previous section, in this second section, we will compare the load time and query response time for storing the RDF data by using the existing systems, namely, TDB, SDB, RDF4JM and RDF4JD.

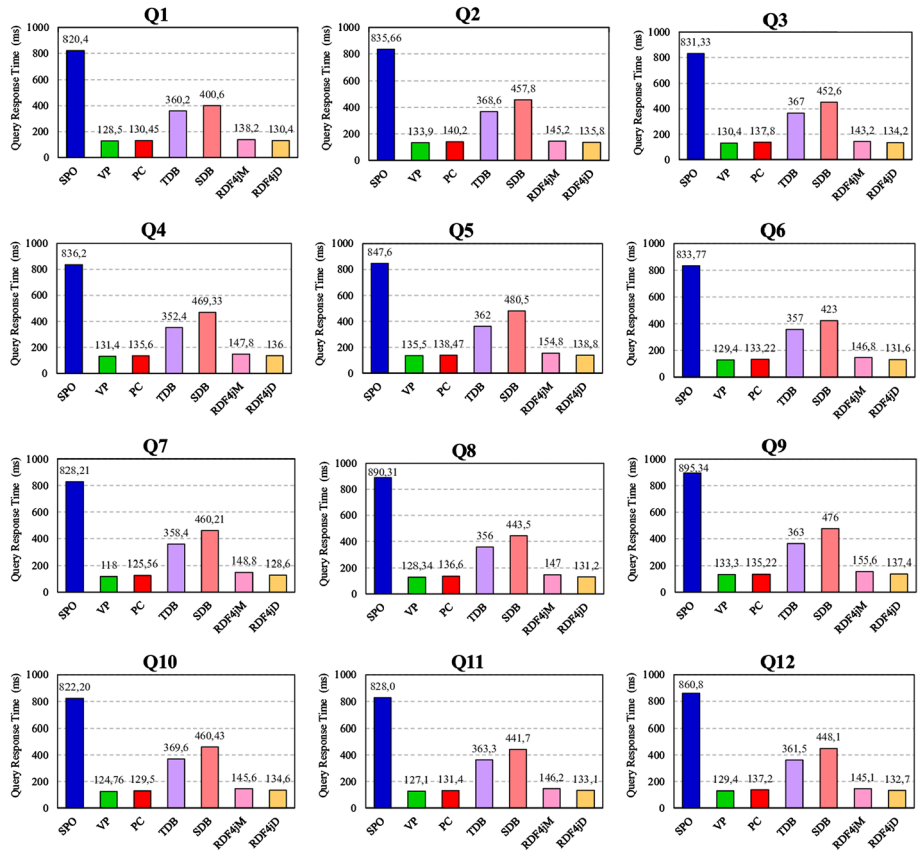

Fig. 7 Query response time comparison between the proposed methods and existing ones

Load time metric

By analyzing the results of Fig. 6b, it turns out that SDB showed a poor load performance while RDF4JM is fast for small datasets when compared to the TDB and RDF4JD. We have tested both RDF4JM and RDF4JD on larger scale datasets. We observe that they did not scale good in data loading, as can be seen from Fig. 6b. For instance, for loading 2,42,249 triples, RDFJM, RDF4JD, and TDB took respectively 64,231 ms, 1,05,764 ms, and 5,551,87 ms. This lead to an interesting remark that can be made about the performance of these repositories in terms of scalability: the disk-based (RDF4JD) and memory-based (RDF4JM) scale good with small datasets, whereas the persistent system (TDB) can scale with both small and large datasets.

In fact, the obtained performances of these systems can be justified by the following reason: the applicability of the different strategies for reasoning. More precisely, the RDF4JD and RDF4JM systems use the forward chaining strategy. This means that the loading of data gets slower because the repository is extending the inferred closure after each transaction. In other words, during the loading, the inferred data should be stored.

On the other hand, the TDB uses the backward chaining strategy, and therefore its loading of the data is quite faster when compared to those of other repositories using forward chaining, since less time is required for the computation and maintenance of the inferred data [18]. Finally, we can remark that even though the SDB system also uses

the backward chaining strategy, it exhibits higher computation time on the contrary to the TDB method. Indeed, this increase in loading time can be related to the fact that the SDB involves additional steps, namely, loading data in the java virtual machine and then on the database for storing data [15].

Query response time

Concerning the query response time of the existing methods (Fig. 7), we can observe that RDF4JD and RDF4JM give better performance when compared with the SDB and TDB. This is due to the inference strategy used for retrieving the data. More specifically, the RDF4JM and RDF4JD use the forward chaining strategy, which means that no deduction and satisfiability checking are required when we query the repository [18]. Whilst, the TDB and SDB methods use the backward chaining. In this case, the query response time is slower because extensive query rewriting (expansion and reformulation) has to be performed [18].

In fact, SDB is not able to compete with any of the other existing systems and shows a poor performance.

Since, it uses the relational database backend for storing the RDF data. This indicates that we must adopt the SPARQL-to-SQL rewriting [49]. Hence, the query response time computed takes into consideration also the rewriting time.

Comparison of the proposed methods versus the existing ones

In order to compare the proposed RDF storage systems against the existing ones, we consider only the two best methods of each group: RDFPC and RDFVP represent the group of the proposed systems and TDB and RDF4JM for the existing ones.

Load time metric

By analyzing the results of the Fig. 8, it is noticeable that the RDFVP, is the best choice when the storage concerns small datasets. Whereas, for a higher number of triples, it is very obvious that the TDB performs better than all the tested systems.

To further justify the efficiency and effectiveness of the proposed methods against the existing ones, we have performed another detailed experimental analysis using the DBPedia (geo_coordinate) dataset that contains 1,569,180 triples. In this respect, Fig. 9 shows the load time in ms for an increasing number of triples starting from 10 to 100% with the interval of 10% of the original number of the triples. From the result we can confirm that the RDFVP gives a good performance load time on small number of triples, whereas as the number of triples increases the TDB proves its robustness in terms of scalability and can be useful for loading a large dataset.

Query response time

Now by comparing the existing systems, the obtained results by RDFVP are more stable than those produced by the other existing ones. In addition, the RDFVP method exhibits the best query response time, which ensures the usefulness of the proposed implementation. The reason behind these results is related to two important aspects. The first one is that RDFVP does not support the reasoning strategy, which means that the systems that adopt the reasoning applicability may lead to an increase in the response time. The

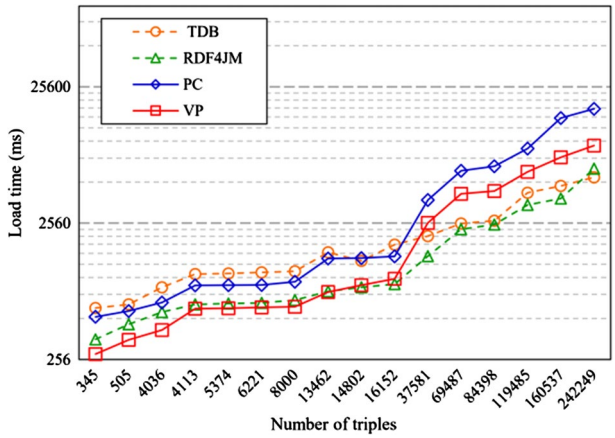

Fig. 8 Load time of the proposed methods against the existing ones

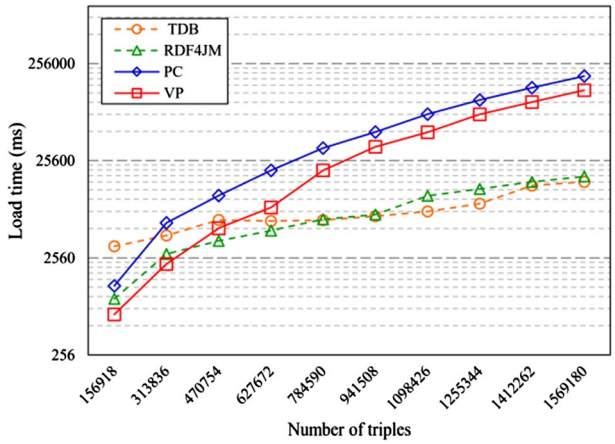

Fig. 9 Comparison of the proposed methods against the existing ones in terms of scalability

second one is about the query language that RDFVP uses, which is SQL. In fact, it is certainly not always true that SQL is fast than the SPARQL because it depends on the system that is used to perform the query as well as the characteristics of the dataset [18]. Basically, the RDFVP is capable of providing a useful implementation for exploratory analysis and other semantic web applications.

Discussion

To give the statistical significance of the differences in load time and query response time of the seven tested methods, a nonparametric Friedman test is performed. The Friedman test [24, 25] is a statistical test that uses the rank of each method on each sample. The Friedman statistic χ_F^2 is given by

$$\chi_F^2 = \frac{12N}{k(k+1)}\left(\sum_{j=1}^{k} R_j^2 - \frac{k(k+1)^2}{4}\right),$$

$$F_F = \frac{(N-1)\chi_F^2}{N(k-1)-\chi_F^2},$$

where $k = 7$ is the number of tested methods, N is the number of samples, which is represented by data sets in the case of load time metric ($N = 17$) and queries in case of query response time ($N = 12$). R_j is the average rank of the method j among all the testing samples. F_F follows a Fisher distribution with $k-1$ and $(k-1)(N-1)$ degrees of freedom.

To conduct the Freidman test, Tables 5, 6 show respectively the load time and query response time metrics using seven methods. In addition, we have reported on each table and for each method the corresponding rank, which is displayed between brackets $(\cdot)$.

The null hypotheses of the Friedman consider that the seven methods are equivalent in terms of computational performance, including load time and query response time. We consider a confidence level $\alpha = 0.05$. According to the Friedman test, the test results p value for load time and query response time are 7.62×10^{-16} and 2.91×10^{-13}, respectively. Therefore, it is evident to reject the null hypotheses, and the seven tested methods are different in terms of computational performance for both load time and query response time.

Then, we use the post-hoc test, namely the Nemenyi test [16, 17], to further analyze the relative performance among the test systems. The performance of two methods is significantly different if the corresponding average ranks differ by at least the critical difference (CD):

Table 5 Comparison of load time (ms) metric of the proposed and existing methods on 17 data sets

Data set	RDFSPO	RDFVP	RDFPC	TDB	SDB	RDF4JM	RDF4JD
All SW groups	581.1 (5)	280.63 (1)	525.8 (4)	611 (6)	1275.14 (7)	359.9 (2)	404.7 (3)
Wikimovies	545.7 (4)	357.42 (1)	580.6 (5)	650 (6)	1300.34 (7)	463.7 (2)	505.5 (3)
NATS	950 (6)	420.42 (1)	670.6 (4)	866.75 (5)	1999.45 (7)	569.7 (2)	593 (3)
Tissues	998.1 (5)	604.31 (1)	895.3 (4)	1084 (6)	2010.23 (7)	648.8 (2)	854.7 (3)
Locations	1217.5 (6)	608.16 (1)	898.8 (4)	1093.12 (5)	3500.56 (7)	659.7 (2)	862.5 (3)
VSRR	2535.5 (6)	616.21 (1)	900.3 (4)	1114.5 (5)	4024.45 (7)	664.6 (2)	870.5 (3)
AQMNEHTN	2744.4 (6)	623.14 (1)	950.2 (4)	1133.5 (5)	85,000.56 (7)	697.5 (2)	886.9 (3)
CDI	6280.8 (6)	800.35 (1)	1410.8 (4)	1562.37(5)	109,958 (7)	807.9 (2)	899 (3)
DHDS	6438.3 (6)	896.42 (2)	1420.4 (5)	1354.37 (4)	115,987.7 (7)	866.8 (1)	984.8 (3)
DASH	6841 (6)	1001.63 (2)	1460.8 (4)	1781.87 (5)	132,024.4 (7)	914.4 (1)	1300.5 (3)
Journals	7214.4 (6)	2576.49 (3)	3790.3 (5)	2055.5 (2)	160,546.5 (7)	1467.5 (1)	3215.5 (4)
Diseases	13,954.7 (6)	4200.14 (3)	6200.5 (5)	2545.5 (2)	180,045.5 (7)	2311.2 (1)	4444.5 (4)
Enzyme	19,885.8 (6)	4410.35 (3)	6700.5 (5)	2668.625 (2)	200,560.7 (7)	2500.8 (1)	5158 (4)
Gene symbol	24,986.3 (6)	6090.42 (3)	9025.6 (5)	4273.375 (2)	1,300,022 (7)	3480.4 (1)	8202.8 (4)
protein	32,154.7 (6)	7788.2 (3)	15,125.8 (5)	4802.75 (2)	1,662,363 (7)	3878.8 (1)	10,162.8 (4)
swdf	38,508.8 (6)	9488.78 (3)	17,644.3 (5)	5551.875 (1)	2,756,740 (7)	6423.1 (2)	10,576.4 (4)
geo_coordinates_en	300,167.7 (6)	119,178.7 (3)	190,222.2 (5)	15,625.25 (1)	26,891,431 (7)	17,549 (2)	123,199.8 (4)
Average rank	(5.76)	(1.94)	(4.53)	(3.76)	(7.00)	(1.59)	(3.41)

The ranks in the parentheses $(\cdot)$ are used in the computation of the Friedman test

Table 6 Comparison of query response time (ms) metric of the proposed and existing methods on 12 queries

Queries	RDFSPO	RDFVP	RDFPC	TDB	SDB	RDF4JM	RDF4JD
Q1	820.4 (7)	128.5 (1)	130.45 (3)	360.2 (5)	400.6 (6)	138.2 (4)	130.4 (2)
Q2	835.66 (7)	133.9 (1)	140.2 (3)	368.6 (5)	457.8 (6)	145.2 (4)	135.8 (2)
Q3	831.33 (7)	130.4 (1)	137.8 (3)	367 (5)	452.6 (6)	143.2 (4)	134.2 (2)
Q4	836.2 (7)	131.4 (1)	135.6 (2)	352.4 (5)	469.33 (6)	147.8 (4)	136 (3)
Q5	847.6 (7)	135.5 (1)	138.47 (2)	362 (5)	480.5 (6)	154.8 (4)	138.8 (3)
Q6	833.77 (7)	129.4 (1)	133.22 (3)	357 (5)	423 (6)	146.8 (4)	131.6 (2)
Q7	828.21 (7)	118 (1)	125.56 (2)	358.4 (5)	460.21 (6)	148.8 (4)	128.6 (3)
Q8	890.31 (7)	128.34 (1)	136.6 (3)	356 (5)	443.5 (6)	147 (4)	131.2 (2)
Q9	895.34 (7)	133.3 (1)	135.22 (2)	363 (5)	476 (6)	155.6 (4)	137.4 (3)
Q10	822.2 (7)	124.76 (1)	129.5 (2)	369.6 (5)	460.43 (6)	145.6 (4)	134.6 (3)
Q11	860,8 (7)	129,4 (1)	137,2 (3)	361,5 (5)	448,1 (6)	145,1 (4)	132,7 (2)
Q12	828,0 (7)	127,1 (1)	131,4 (2)	363,3 (5)	441,7 (6)	146,2 (4)	133,1 (3)
Average rank	(7.00)	(1.00)	(2.50)	(5.00)	(6.00)	(4.00)	(2.50)

The ranks in the parentheses (·) are used in the computation of the Friedman test

$$CD_\alpha = q_\alpha \sqrt{\frac{k(k+1)}{6N}}.$$

For the Nemeyi test, we have the critical tabulated value is $q_\alpha = 2.949$ at a significance level $\alpha = 0.05$. Hence, the CDs for the load time and query response time are respectively equal to 2.1850 (k = 7, N = 17) and 2.6007 (k = 7, N = 12).

To visually depict the relative performance of the proposed methods (RDFSPO, RDFPC and RDFVP) compared to the other existing ones, the CD diagrams on the different evaluation metrics are shown in Fig. 10a, b, where the average rank of each comparing method is signed along the axis (higher ranks to the right). In each subfigure, a thick line is used to connect the methods whose average ranks are smaller than the critical difference. Otherwise, all systems that are not connected with each other are recognized to have apparently different performances.

The statistical test shown in Fig. 10a reveals that the load time with RDF4JM is statistically better than those with RDFPC, RDFSPO and RDFSDB (as RDFPC − RDF4JM = 4.52 − 1.58 > 2.18, RDFSPO − RDF4JM = 5.76 − 1.58 > 2.18 and SDB − RDF4JM = 7.00 − 1.58 > 2.18). Moreover, it is worth noting that there is no consistent evidence to indicate statistical load time differences between the proposed RDFVP and RDF4JM (as RDFVP − RDF4JM = 1.94 − 1.58 < 2.18), the same conclusion holds for VP with RDF4JD and with TDB.

The results are shown in Fig. 10b indicates that the query response time of our proposed RDFVP method is statistically better than those with RDF4JM, SDB, TDB and RDFSPO (as RDF4JM − RDFVP = 4.00 − 1.00 > 2.600, SDB − RDFVP = 6.00 − 1.00 > 2.600, TDB-RDFVP = 5.00–1.00 > 2.600 and RDFSPO-RDFVP = 7.00–1.00 > 2.600). However, there is no consistent evidence to indicate statistical query response time differences between RDFVP, RDFPC and RDF4JD (as RDFPC −RDFVP = 2.50 − 1.00 < 2.600 and RDF4JD-RDFVP = 2.50–1.00 < 2.600).

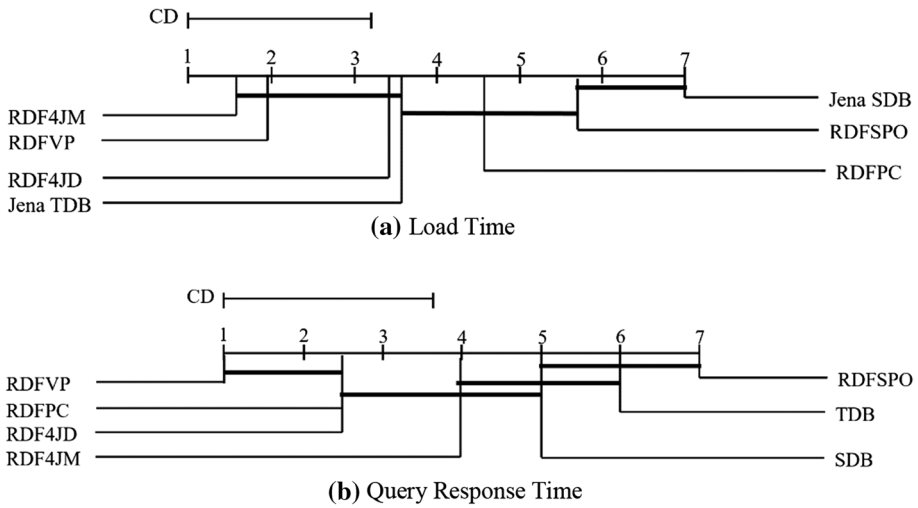

Fig. 10 Computational performance comparison of the proposed RDFVP, RDFSPO and RDFPC against the existing methods with Nemenyi Test. **a** Load time, **b** query response time

To summarize, the proposed RDFVP method achieve highly competitive computational performance against other state-of-the-art methods in terms of load time and query response time.

Conclusion

In this paper, we proposed three new implementations of non-native methods for storing the RDF data. These implementations are respectively based on statement table, property table and vertical partitioning approaches. What is more, we consider the issue of how to select a convenient and efficient storage solutions based on the dataset characteristics. In this respect, two important performance metrics are provided, which include load time and query response time for evaluating the RDF storage systems. In order to show efficiency and the robustness of the proposed and existing RDF storage systems, the experimental studies have been divided into three sections. The first one consisted of evaluating the proposed RDF storage systems. The second section has been devoted to evaluating the existing RDF storage systems. In the third section, we provide a comparison between the proposed and existing RDF storage systems. In addition, to bring a significant reason to the obtained results, we have applied a statistical analysis based on the Friedman Test followed by a Nemenyi post hoc test to further explore which system perform statistically. In future works, we aim to adopt a machine learning algorithms to predict the most appropriate system for a specific dataset. More specifically, we plan to bypass the traditional approaches for estimating the load time and the query response time that are based on statistics about the underlying RDF dataset. In this context, we will show how to model these two metrics as feature vectors to accurately use them as input for a machine learning algorithms.

Abbreviations

SRS: Semantic repository system; DBMS: Data base management system; RDF: Resource description framework; RDFSPO: RDF subject-predicate-object; RDFSPO: RDF subject-predicate-object; RDFPC: RDF property clustering; RDFVP: RDF vertical partitioning; RDF4JM: RDF4J main memory-based; RDF4JD: RDF4J disk-based; PT: Property table; PC: Property class; CP: Clustered property; VP: Vertical partitioning; CD: Critical difference.

Acknowledgements
Not applicable.

Authors' contributions
BBM, IC and AZ conceived of the presented idea. BBM developed the theory, performed the algorithms in addition to writing the manuscript with support from IC and AZ. IC and AZ verified the analytical methods and they helped supervise the project. Finally, all authors discussed the results and contributed to the final manuscript. All authors read and approved the final manuscript.

Funding
Not applicable.

Availability of data and materials
The data that support the findings of this study are available from the corresponding author (Bilal Ben Mahria).

Declarations

Ethics approval and consent to participate
Not applicable.

Consent for publication
Not applicable.

Competing interests
Not applicable.

Received: 5 February 2021 Accepted: 17 June 2021
Published online: 10 July 2021

References

1. Aasman J. Allegro graph: RDF triple database. Oakland: Franz Incorporated; 2006. p. 17.
2. Abadi DJ, Marcus A, Madden SR, Hollenbach K. SW-Store: a vertically partitioned DBMS for semantic web data management. VLDB J. 2009;18:385–406.
3. Abadi DJ, Marcus A, Madden SR, Hollenbach K. Scalable semantic web data management using vertical partitioning. In: Proceedings of the 33rd International Conference Very Large Data Bases. VLDB Endowment; 2007. p. 411–22.
4. Alexaki S, Christophides V, Karvounarakis G et al The ICS-FORTH RDFSuite: managing voluminous RDF description bases. In: SemWeb. 2001.
5. Aluç G, Hartig O, Özsu MT, Daudjee K. Diversified stress testing of RDF data management systems. In: International Semantic Web Conference. Berlin: Springer; 2014. p. 197–212.
6. Atre M, Srinivasan J, Hendler JA. BitMat: a main memory RDF triple store. Tetherless world constellation. Troy: Rensselar Plytehcnic Institute; 2009.
7. Beckett D. The design and implementation of the Redland RDF application framework. Comput Netw. 2002;39:577–88.
8. Bizer C, Schultz A. The berlin sparql benchmark. Int J Semant Web Inf Syst. 2009;5:1–24.
9. Bornea MA, Dolby J, Kementsietsidis A et al. Building an efficient RDF store over a relational database. In: Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data. New York: ACM; 2013. p. 121–32.
10. Broekstra J, Kampman A, Van Harmelen F. Sesame: a generic architecture for storing and querying rdf and rdf schema. In: International Semantic Web Conference. Berlin: Springer; 2002. p. 54–68.
11. Butt AS, Khan S. Scalability and performance evaluation of semantic web databases. Arab J Sci Eng. 2014;39:1805–23.
12. Chen JX, Reformat MZ. Learning categories from linked open data. In: International Conference on Information Processing and Management of Uncertainty in Knowledge-Based Systems. Berlin: Springer; 2014. p. 396–405.
13. Copeland GP, Khoshafian SN. A decomposition storage model. In: Acm Sigmod Record. New York: ACM; 1985. p. 268–79.
14. Corwin J, Silberschatz A, Miller PL, Marenco L. Dynamic tables: an architecture for managing evolving, heterogeneous biomedical data in relational database management systems. J Am Med Inform Assoc. 2007;14:86–93.
15. Curé O, Blin G. RDF database systems: triples storage and SPARQL query processing. Burlington: Morgan Kaufmann; 2014.
16. Demšar J. Statistical comparisons of classifiers over multiple data sets. J Mach Learn Res. 2006;7:1–30.
17. Derrac J, García S, Molina D, Herrera F. A practical tutorial on the use of nonparametric statistical tests as a methodology for comparing evolutionary and swarm intelligence algorithms. Swarm Evol Comput. 2011;1:3–18.
18. Domingue J, Fensel D, Hendler JA. Handbook of semantic web technologies. Berlin: Springer; 2011.
19. Duan S, Kementsietsidis A, Srinivas K, Udrea O. 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. New York: ACM; 2011. p. 145–56.
20. Erling O, Mikhailov I. RDF support in the virtuoso DBMS. In: Networked knowledge-networked media. Berlin: Springer; 2009. pp 7–24.
21. Faye D, Curé O, Blin G, Thiam C. RDF triples management in roStore. In: IC 2011, 22èmes Journées francophones d'Ingénierie des Connaissances. 2012; p. 755–70.

22. Faye DC, Cure O, Blin G. A survey of RDF storage approaches. Rev Afr Rech Inform Math Appl. 2012;15:11–35.
23. Fletcher GH, Beck PW. Scalable indexing of RDF graphs for efficient join processing. In: Proceedings of the 18th ACM Conference on Information and Knowledge Management. New York: ACM; 2009. p. 1513–16.
24. Friedman M. The use of ranks to avoid the assumption of normality implicit in the analysis of variance. J Am Stat Assoc. 1937;32:675–701.
25. Friedman M. A comparison of alternative tests of significance for the problem of m rankings. Ann Math Stat. 1940;11:86–92.
26. Gayo JEL, Prud'Hommeaux E, Boneva I, Kontokostas D. Validating RDF data. Synth Lect Semant Web Theory Technol. 2017;7:1–328.
27. Guha RV. Rdfdb: an rdf database. Disponível. 2000. http://rdfdb.sourceforge.net/. Accessed 15 Nov 2003.
28. Guo Y, Pan Z, Heflin J. LUBM: a benchmark for OWL knowledge base systems. Web Semant Sci Serv Agents World Wide Web. 2005;3:158–82.
29. Harris S, Gibbins N. 3store: Efficient bulk RDF storage. In: 1st International Workshop on Practical and Scalable Semantic Systems (PSSS'03), Sanibel Island, Florida; 2003. p. 1–15.
30. Harris S, Lamb N, Shadbolt N. 4store: the design and implementation of a clustered RDF store. In: 5th International Workshop on Scalable Semantic Web Knowledge Base Systems (SSWS2009). 2009; p. 94–109.
31. Harth A, Decker S. Optimized index structures for querying rdf from the web. In: Third Latin American Web Congress (LA-WEB'2005). New Jersey: IEEE; 2005. p. 10.
32. Harth A, Umbrich J, Hogan A, Decker S. Yars2: a federated repository for querying graph structured data from the web. In: The semantic web. Berlin: Springer; 2007. p. 211–224.
33. Hassan M, Bansal SK. RDF data storage techniques for efficient SPARQL query processing using distributed computation engines. In: 2018 IEEE International Conference on Information Reuse and Integration (IRI). New Jersey: IEEE; 2018. p. 323–30.
34. Janik M, Kochut K. BRAHMS: a workbench RDF store and high performance memory system for semantic association discovery. In: International Semantic Web Conference. Berlin: Springer; 2005. p. 431–45.
35. Karvinen P, Díaz-Rodríguez N, Grönroos S, Lilius J. RDF stores for enhanced living environments: an overview. In: Enhanced living Environments. Berlin: Springer; 2019. p. 19–52.
36. Kolas D, Emmons I, Dean M. Efficient linked-list rdf indexing in parliament. SSWS. 2009;9:17–32.
37. Ma Z, Capretz MA, Yan L. Storing massive resource description framework (RDF) data: a survey. Knowl Eng Rev. 2016;31:391–413.
38. MahmoudiNasab H, Sakr S. AdaptRDF: adaptive storage management for RDF databases. Int J Web Inform Syst. 2012;8:234–50.
39. McBride B. Jena: a semantic web toolkit. IEEE Internet Comput. 2002;6:55–9.
40. McGlothlin J, Khan L. RDFJoin: a scalable data model for persistence and efficient querying of RDF datasets. Database. 2009.
41. Modoni GE, Sacco M, Terkaj W. A survey of RDF store solutions. In: 2014 International Conference on Engineering, Technology and Innovation (ICE). New Jersey: IEEE; 2014. p. 1–7.
42. Murray C, Alexander N, Das S, et al. Oracle spatial. Resource description framework (RDF) 10g Release 2 (10.2). Oracle com 186. 2005.
43. Pan Z, Zhu T, Liu H, Ning H. A survey of RDF management technologies and benchmark datasets. J Ambient Intell Humaniz Comput. 2018;9:1693–704.
44. Reggiori A, van Gulik D-W, Bjelogrlic Z. Indexing and retrieving semantic web resources: the RDFStore model. In: SWAD-Europe workshop on semantic web storage and retrieval. Citeseer; 2003. p. 13–4.
45. Schmidt M, Görlitz O, Haase P, et al. Fedbench: a benchmark suite for federated semantic data query processing. In: International Semantic Web Conference. Berlin: Springer; 2011. p. 585–600.
46. Schmidt M, Hornung T, Lausen G, Pinkel C. SP^ 2Bench: a SPARQL performance benchmark. In: 2009 IEEE 25th International Conference on Data Engineering. New Jersey: IEEE; 2009. p. 222–33.
47. Sidirourgos L, Goncalves R, Kersten M, et al. Column-store support for RDF data management: not all swans are white. Proc VLDB Endow. 2008;1:1553–63.
48. Singh G, Upadhyay D, Atre M. Efficient RDF dictionaries with B+ trees. In: Proceedings of the ACM India Joint International Conference on Data Science and Management of Data. New York: ACM; 2018. p. 128–36.
49. Soussi N, Bahaj M. Semantics preserving SQL-to-SPARQL query translation for nested right and left outer join. J Appl Res Technol. 2017;15:504–12.
50. Thakker D, Osman T, Gohil S, Lakin P. A pragmatic approach to semantic repositories benchmarking. In: Extended Semantic Web Conference. Berlin: Springer; 2010. p. 379–93.
51. Tran T, Ladwig G, Rudolph S. Istore: efficient rdf data management using structure indexes for general graph structured data. Institute AIFB, Karlsruhe Institute of Technology. 2009.
52. Weiss C, Karras P, Bernstein A. Hexastore: sextuple indexing for semantic web data management. Proc VLDB Endow. 2008;1:1008–19.
53. Wilkinson K, Sayers C, Kuno H, Reynolds D. Efficient RDF storage and retrieval in Jena2. In: Proceedings of the First International Conference on Semantic Web and Databases. Citeseer; 2003. p. 120–39.
54. Wood D, Gearon P, Adams T. Kowari: a platform for semantic web storage and analysis. In: XTech 2005 Conference. p. 5–402.
55. Zhang Y, Duc PM, Corcho O, Calbimonte J-P. SRBench: a streaming RDF/SPARQL benchmark. In: International Semantic Web Conference. Berlin: Springer; 2012. p. 641–57.

Publisher's Note