Advances in Science, Technology and Engineering Systems Journal Vol. 6, No. 2, 221-233 (2021)

www.astesj.com

ASTESJ

ISSN: 2415-6698

Categorization of RDF Data Management Systems

Khadija Alaoui*, Mohamed Bahaj

MIET Lab, Faculty of sciences and Techniques, Hassan I University, Settat, 26422, Morocco

ARTICLE INFO

Article history:
Received: 28 December, 2020
Accepted: 23 February, 2021
Online: 10 March, 2021

Keywords:
Triplestore
RDF
OWL
SPARQL
Semantic web
Big Data
Cloud
NoSQL
IoT

ABSTRACT

The wide acceptance of the semantic web language RDF for ontologies creation in various application fields has led to the emergence of numerous RDF data processing solutions, the so-called triplestores, for the storage of RDF data and its querying using the RDF query language SPARQL. Such solutions are however developed under various perspectives and on the basis of various architectures. It is therefore a necessity for users to be able to distinguish between these systems to decide about the appropriate triplestore for an efficient processing of their RDF data depending on their objectives, the characteristics of their data and the technologies at hand. To this end, we give an extended categorization of RDF data stores according to their main characteristics. Furthermore, we review relevant existing triplestores within their respective established categories. The categorization is established according to the motivations behind the adoption of one or the other triplestore for handling the main tasks of data storage and SPARQL querying. Furthermore, the categorization considers various aspects that specifically deal with RDF data modeling, organization of RDF data, the processing of SPARQL queries, scalability, as well as aspects related to the diverse related data processing technologies.

1. Introduction

The "Resource Description Framework" (RDF) has been worldwide used during the last two decades for creating semantic ontologies in various application areas, and it is standardized by the "World Wide Web Consortium" (W3C) as the language of the semantic web (https:// www.w3.org/ TR /rdf11-primer/). RDF represents data in the form of (S, P, O) triples to express the semantic information that an entity or a resource S is in a relationship through the relation or predicate P with an object O that is either a resource or a literal value. This modeling art lets then represent data as RDF directed labeled graphs where in each graph, resources and literal values are representing nodes of the graph and a node n1 is connected to a node n2 with an arc labeled by a predicate P if (n1, P, n2) is an RDF triple. To query the RDF triples, W3C also launched the standard language SPARQL ("Simple Protocol and RDF Query Language" - https:// www.w3.org/ TR /sparql11-overview/). For interlinking purposes and for ontologies identification, entities are also endowed with URIs (Unique Resource Identifier). This mechanism has the advantage of assigning resources to groups, also called ontologies, and allowing interlinking resources of one group to resources of other groups yielding heterogeneous RDF data graphs.

It is exactly this simple semantic format offered by RDF to model data within ontologies that led to the transformation of the classical web to change it from a web of static pages to an intelligent web of interlinked data. The RDF format makes it indeed possible for machines to intelligently navigate inside the interlinked data since it enables formulating semantics about such data. Furthermore, the schema languages RDFS ("RDF Schema" - https:// www.w3.org/ TR /rdf-schema/) and OWL ("Web Ontology Language" - https:// www.w3.org/ TR /owl2-syntax/), which are also W3C standards, do offer various semantic constructs to model the schemas of RDF data and allow intelligent navigation through such data using inference and reasoning techniques. RDF also offers various advantages for semantic modeling of enterprise data through its flexible schema definition and also offers a better alternative to the classical entity-relationship modeling approach [1], [2]. All these factors have led to the appearance of an important number of management systems for handling the storage and the querying of RDF data.

The abundance and variety of RDF data processing systems, also called triplestores, was also encouraged in a natural way by the emergence of various technologies such.as NoSQL (Not only SQL - (Structured Query Language)), P2P (Peer to peer) and Big Data ones and was also imposed by the multiple varieties of RDF applications. A multitude of RDF triplestores have indeed been

*Corresponding Author: Khadija Alaoui, alaoui_khadija@outlook.com

developed, each with its own features that distinguish it from other triplestores. So, for a specific use case or application involving the use of RDF for data modeling, an appropriate RDF storage and processing system must however be well chosen from the multitude of existing RDF triplestores dependently of multiple factors.

In this sense, this work presents an extensive extension of the preliminary categorization of triplestores we gave in our conference paper [3]. The extension consists of a detailed categorization of RDF management systems with a review of relevant triplestores within their associated categories. Beyond the respect of RDF modeling constructs and implementation of elements of its query language SPARQL, RDF data management systems are filtered in accordance to the strategies used either for query processing or data storage. The strategies are enforced on one hand by the system architecture used if it is centralized or distributed, if it is a P2P, a cloud or a big data one. On the other hand, such strategies also depend on the adopted storage and querying methods, if they are relying on other existing data processing frameworks or if they are designed from scratch independently of any such frameworks. Furthermore, each category is presented according to the strategy used to handle RDF data storage and processing taking into consideration the structures used for its storage, indexing schemes and SPARQL implementation. For the organization of data storage, partitioning and indexing schemes are of particular interest since they affect the speed of query execution. This detailed categorization dependently of the data processing architectures and of the used systems characteristics and the targeted deployment machines represents therefore our main contribution in this work. The categorization with respect to such elements is of great importance for data management since they affect in a direct way the performance as well as the scalability of the triplestores at hand. To illustrate the given categorization we also review major relevant existing triplestores within their respective established categories.

The following sections of the paper are structured as follows. Section 2 presents the semantic web standards RDF, SPARQL, RDFS and OWL. Section 3 gives a summary of our categorization approach. Sections 4 to 9 present the main categories with their respective sub-categories. Section 10 summarizes the categories with a discussion on related works. Section 11 concludes this work.

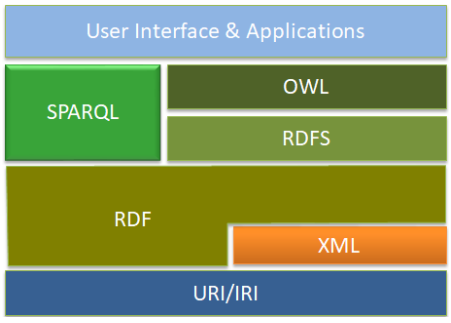

Figure 1: Web semantic Architecture

2. Standards of the Semantic Web

In the following subsections we give a brief presentation of the web semantic standards. The focus is mainly on the main issues related to these standards that are in a direct connection with the tasks of triplestores with respect to RDF data storage and query processing. Figure 1 shows the elements of the sematic web architecture.

2.1. RDF Data Model

The representation of data with RDF is based on modeling all information as a set of sentences of the form 'Subject Predicate Object' yielding triples (S:=Subject,P:=Predicate,O:=Object). Each triple (S,P,O) gives the meaning that the resource S is in a relationship through P with the object O. Objects can be either resources or literal values. In the example of figure 2, we have for example the triple (ex:Jabir,ex:teach,ex:java).

```
@prefix rdfs : <http://www.w3.org/2000/01/rdf-schema#>.
@prefix rdf : <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
@prefix xsd : <http://www.w3.org/2001/XMLSchema#>.
@prefix ex : <https://www.mySite.ma/example#>.

ex:teachingactivity rdf:type rdfs:Class .
ex:unistaff rdf:type rdfs:Class .
ex:course rdfs:subClassOf ex:teachingactivity .
ex:teach rdf:type rdf:Property;
      rdfs:domain ex:lecturer;
      rdfs : range ex:course .
ex:weekhours rdf:type rdf:Property;
      rdfs:domain ex:course;
      rdfs : range xsd:integer .
ex:java rdf:type ex:course ;
      ex: weekhours 5.
ex:Jabir rdf:type ex:unistaff ;
      ex:teach ex :java .
```

Figure 2: RDF Example in N3-Notation

2.2. RDFS and OWL

RDFS offers constructs to describe elements of an RDF graph in a meta-model. The statements in the RDFS meta-model are also expressed as RDF triples. The meta-model declares the classes of resources and predicates used in the RDF graph. Ranges and domains of predicates can also be given in the meta-model. RDFS also offers the possibility of creating hierarchies between classes using constructs such as “subClassOf”. For example, in the example of Figure 3, the class “ex:course” is declared as a subclass of “ex:teachingactivity”.

OWL extends RDFS with many semantic constructs allowing the definition of more expressive RDF graphs and offering more reasoning possibilities on them. OWL meta-models are also expressed in RDF which makes the reasoning based on description logic easier. As examples of constructs in OWL we mention ‘ObjectProperty’ and ‘DatatypeProperty’ for the definition of types of predicates, and ‘AllValuesFrom’, ‘SomeValuesFrom’, ‘ComplementOf’ and ‘DisjointWith’ for constraints on domains and ranges of predicates. OWL also provides constructs for the creation of new types from other types as well as constructs for properties on predicates, for example, if they are invertible, symmetric or transitive.

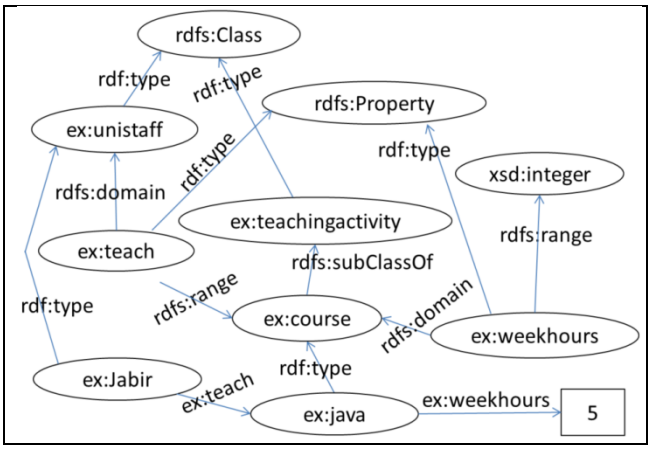

Figure 3: RDF Graph Example

2.3. SPARQL

To query RDF data, the query language SPARQL has been proposed and standardized by the W3C. SPARQL is very similar to SQL and can perform complex joins of various RDF data graphs in the same query. Figure 4 gives a simple example for looking after who teaches "java".

```
Prefix ex: <https://www.mySite.ma/example#>
SELECT ?x
WHERE
   ?x ex:teach ex:java .
```

Figure 4: SPARQL example

Beyond SELECT queries to extract information from RDF data, SPARQL also offers ASK queries that return either true if the query condition is satisfied and false if it is not satisfied, and CONSTRUCT to add new triplets to RDF graphs as results of such queries, as well as DESCRIBE queries that extract information about a resource. SPARQL queries can also handle aggregations and may contain optional clauses with optional conditions as well as a FILTER clause to further filter their results.

3. Categorization approach

The categorization approach we are using is mainly based on the context in which RDF data is used. Within this context the following elements are considered:

- The storage technique used: We mainly focus on its adaptation to RDF model and for which environment it is used. With environment we consider the use of the solution on only one machine or on a cluster of machines and if the solution is for use in Cloud, P2P or desktop context.
- Nature of destination devices: we handle the case of using RDF data either in constrained devices, desktops or clusters.
- System scalability: we especially take into account the separation of solutions dependently on data volumes to be processed.
- Data organization: This point is very important since SPARQL queries may pose many challenges related among others to join and sub-graph processing especially when RDF data are scattered among various graphs or stored in multiple files or in multiple nodes.

4. Native versus Non-Native Triplestores

Native RDF data systems are those systems that are built from scratch only for the purpose of handling RDF data without relying on any existent data management solution. This means that the solutions associated with such native stores are implemented independently of any existing specific database engine for the storage or querying of any kind of data. To achieve their tasks, native stores may however be built using functionalities of the file system under hand and of course existing programming languages such as C, C++ and Java.

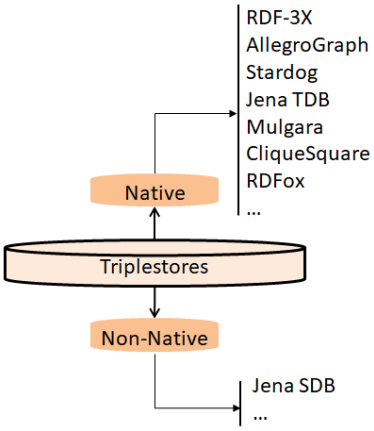

Figure 5: Catégorisation "Natifs / Non-Natifs"

In contrary, non-native triplestores are those stores that rely on already existing data management solutions such as, for example, relational, XML, NoSQL database management systems or also Big Data technologies for data processing such as HBase or Pig. Figure 5 illustrates the considered "native/non-native" categorization.

4.1. Non-Native triplestores

As examples of non-native triplestores we have Jena SDB (https://jena.apache.org), triplestores that are based on existing classical relational database systems and triplestores that are based on NoSQL database systems. The category of relational triplestores is treated in section 6.1 and the category of non-relational triplestore is considered in section 6.2.

The Jena framework is implemented in Java. It has been continuously updated since its launching in the year 2000. Jena uses a data structure called model to represent an RDF graph with associated methods to manipulate its nodes which can be resources, blank nodes or literals. Also Jena creates triples as instances of the Statement class.

Jena also comes with a reasoning module for inferencing based on some RDFS as well as OWL constructs and also based on rules that are defined by users. Furthermore, a Jena server called Fuseki is also provided for SPARQL querying over HTTP.

Jena SDB uses Jena APIs and JDBC for handling RDF data in a relational database system. It will be further detailed in the category of single-table relational triplestores category.

4.2. Native triplestores

As already mentioned, in contrast to non-native stores which are setup to run on top of other existing database processing solutions, native stores are built specially for the RDF model to provide persistent storage with own database implementation solutions. Examples of such native store are RDF-3X [4], AllegroGraph (https://allegrograph.com), Stardog (http:// stardog.com), Jena TDB [5], Mulgara (http://www.mulgara.org), RDFox (http:// www.cs.ox.ac.uk/ isg/ tools/RDFox) and CliqueSquare [6], [7].

AllegroGraph store uses RDF-XML and N-Triples to load the triples. The implemented query language is SPARQL, however external programming APIs can be used to find datasets matching specific triples.

CliqueSquare uses the distributed file system of Hadoop for storing data and its MapReduce implementation for the processing of RDF data.

5. Memory-Based versus Disk-Based Triplestores

Memory based triplestores, also called in-memory databases, rely on main memory for data storage. As the memory access is faster than disk access, these triplestores allow quick access to data and faster query execution. Memory based triplestores show therefore best performance since entire datasets are in memory.

Figure 6 shows the two considered categories which are presented next.

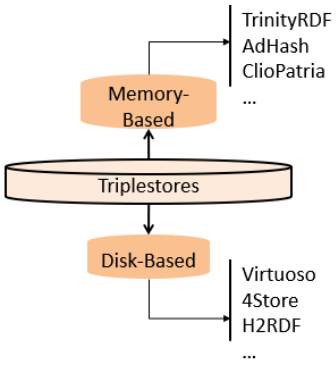

Figure 6: "Memory / Disc" Categorization

5.1. Memory-based triplestores

As the name indicates, main-memory-based triplestores fully load RDF data in main memory to do processing on it. Jena TDB, TrinityRDF [8], AdHash [9], ClioPatria [10] and ScalaRDF [11] are examples of memory based triplestores.

TrinityRDF allows the store of trillions of triples. It represents entities as graph nodes while the relations are represented as graph edges. Trinity supports parallel computing and handles massive number of in-memory objects as well as complex data with large schemas; however, it does not guarantee serialization for concurrent threads.

AdHash uses the principle of applying lightweight hash partitioning to distribute the triples by using a hashing according to subjects in order to limit the data communication costs for join queries. AdHash elaborates this by monitoring the data access patterns and gradually redistributing and replicating the accessed data. By increasing the in parallel executed join operations, AdHash improves the queries execution time.

5.2. Disk-based triplestores

The triplestores in this category interact with RDF data through programs loading from disk the portions of data each time when they are needed. In this category we have of course those triplestores that use engines of relational database systems for processing RDF data such as Virtuoso [12] and 4store (https://github.com/4store/4store).

We also have Big Data RDF processing solutions that rely on Hadoop or Spark frameworks for managing RDF data and which will be presented in section 8.

6. Relational versus Non-Relational Categorization

During the first years of the semantic web, the focus was mainly on the use of relational database (RDB) systems for the storage and processing of RDF data on one hand for their dominance and on the other hand for the aim to benefit from associated during years developed technologies with respect to efficient data processing as well as to users APIs.

However, such use of these relational systems still face many challenges such as the need for efficient solutions to reduce the added time complexities due to the need of translating SPARQL queries into SQL ones. Also, there are still some difficulties faced by the semantic web world for the use of object oriented based application frameworks and programming languages. Furthermore, the dynamicity of RDF data generally poses a challenging problem to relational database designers since relational schemas generally rely on static schemas to model the tables of their databases.

6.1. Relational Triplestores

Relational RDF stores use relational database (RDB) systems to store and query RDF data. Figure 7 presents the categories of such stores which will be detailed next.

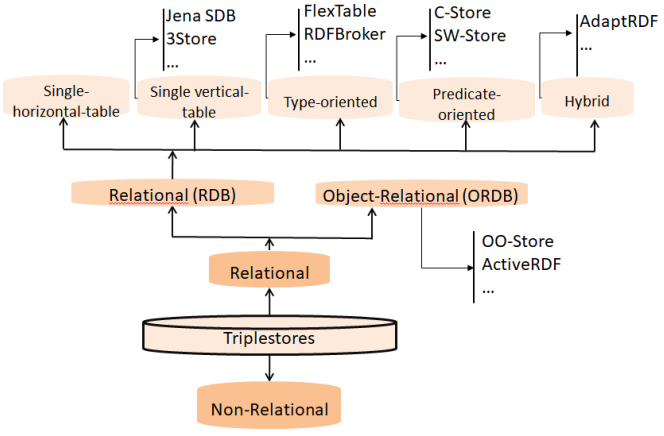

Figure 7: "Relational Triplestores" Category

Relational triplestores provide various advantages due to the technologies developed over decades for relational database management systems (RDBMs). Worth mentioning with respect to RDF data storage and processing is the indexing strategies (e.g.,

hashing, B/B+ trees) offered by RDBMs and query optimization techniques based on relational algebra operators as well as value typing. Also relational triplestores allow easy data integration of relational databases into RDF models or of other data sources with the use of existing data mapping and conversion techniques for transforming and storing such data sources into relational databases. A further positive point of relational triplestores is the possibility to use exiting data analytics tools (e.g., machine learning, business intelligence) developed for RDBMs. However extensions in this sense are still to be considered in the context of the nature of RDF data.

Relational triplestores also suffer from the limitation related to the high processing costs due to RDF data loading in RDBMs and also to the need of translating SPARQL queries into SQL ones for data processing. Another drawback of RDF stores is that they are in majority centralized solutions which let them not to be adequate for massive RDF data management. A further negative point of relational triplestores is the lack of user involvement to use equivalent functionalities that are already offered to SQL users such as creation of indexes or programming interfaces.

6.1.1. Non-Object Relational stores

Since the beginning of the semantic web, various solutions to store RDF in classical non-object relational database (RDB) systems have been proposed. They mainly depend on how the RDF triples are distributed with the appropriate relational schemas. In the following we present the main sub-categories of RDB triplestores and their RDB used schemas to manage RDF data.

6.1.1.1. Single vertical-table RDB triplestores

This category contains those relational stores that store triples in a single table with a column for subjects, a column for predicates, a column for objects and possibly a column for graphs to which triples belong. In this category we have the triplestores Jena SDB, 3Store [13], 4Store [14], Sesame [15] and Hexastore [16].

Jena-SDB which is an RDB triplestore can be used with a large number of RDBs which let it benefit from the indexing capabilities provided by RDB. Applications may use JDBC connector to store RDF triples in Jena SDB. The use of Jena-SDB is only recommended when it is necessary to layer on an existing SQL deployment. However, if explicit transactions support is required, Jena SDB reuses the transaction model from the underlying relational database.

3Store has a query engine developed using the C language and it is implemented on top of MySQL. Hash tables are used respectively for resources, literals and graphs to encode such objects. Triples are stored in a single table that stores for each component of a triple its associated hash code that is used as a reference key to its entry in the associated table. RDF data can be accessed via RDQL based on an apache server interface and a query engine that translates RDQL query into SQL query.

4Store runs on a cluster. Its design is based on 3Store. Data in 4Store is stored as quads (G:graph, S,P,O) where G is the graph to which the triple '(S,P,O)' belongs. Triples in 4Store are partitioned using hashing on the identifiers of subjects. This partitioning strategy can however lead to nodes that are heavily loaded with data than the others and may therefore lead to high query processing time costs.

6.1.1.2. Single-horizontal-table RDB stores

The systems of this category use a single table, also called predicate table, with a column for subjects and a column for each possible predicate. Each resource is then stored with all values of its associated predicates in one line of the table and NULL values for the other predicates. Therefore, this approach could lead to lines in the table with many NULL values resulting in large processing times.

6.1.1.3. Predicate-oriented RDB triplestores

These systems associates a relation with two columns with each predicate for its (subject, object) pairs [17]. C-Store [18] and SW-Store [19] are examples of such stores.

An advantage of this storing approach is that it is easy to implement and resolve the NULL values problem of single-horizontal-table. However this approach has the disadvantage that it comes with a huge number of tables which involves large number of joins in query execution.

6.1.1.4. Type-oriented RDB triplestores

These systems associate a relation with each RDF resource type (i.e., for each class of objects) with one column for subjects of this common class and a column for each predicate associated with such subjects. Triples with subjects not belonging to any class are stored together separately in a table of three columns respectively associated with subjects, predicates and objects. FlexTable [20] and RDFBroker [21] are examples of such stores.

RDFBroker is based on occurring predicates signatures in RDF data. The set of predicates that occur with a resource in the triples is called its signature. These signatures together build the signature of the graph. A signature graph is then constructed with nodes being the signatures, and an edge from one signature to a second one means that the first signature is a subset of the second one. RDFBroker then creates for each signature in the graph a table with a first column for the subjects appearing with the signature predicates in the RDF data triples and one column for each of these predicates. Triples are then put in the suited table according to the signature of their subjects. Such a strategy does remedy to the problem of NULLs posed by the single-horizontal-table approach.

6.1.1.5. Hybrid triplestores

Hybrid triplestores consist of stores that use combination of previous approaches. Within this category, we principally have stores that cluster the predicates that appear together with respect to a clustering criterion. For each cluster of predicates, a table is created with columns represented by the cluster predicates and a column for the subjects, and triples with these predicates are put together in this so-called property table. To store the triples with the remaining predicates, which are not clustered, a single-vertical-table is used.

AdaptRDF [22] is an example of hybrid triplestores. It uses a single vertical table as well as property tables. First all triples are put in the vertical table and with respect to the queries load property tables are created to partition the vertical table and further dynamically adjusted with respect to predicates based on a

mathematical process which considers the query workloads over time.

6.1.2. Object Relational Stores

Object relational databases (ORDBs) offer the possibility to encapsulate data within objects. They also give methods to serialize objects, generally as key value associations, or compound values. ORDBs are specially needed in fields with complex data objects and objects interactions with data represented as a collection of objects.

Conversion and storage techniques from RDF into ORDBs have mainly been inspired from previous similar works on the processing of XML documents in ORDBs. One solution in this sense is the one proposed with its prototype in [23]. In this solution, for each document a model instance of a class Model is created to manage the elements of the document. A class Resource is defined to instantiate a resource or a predicate and a class Literal is used to instantiate literals. All these three classes are subclasses of a class Node which gathers common attributes. Also a class Statement is provided for instantiating triples with Node-components. Using these classes, the various elements of the RDF document are scanned and stored as objects using the methods of the class Model.

Also worth mentioning are OO-Store [24] which is proposed as a prototype implementation for the processing of RDF data based on ORDBs, and ActiveRDF [25] which also comes with programming elements for the management of RDF data.

All object oriented relational RDF solutions with their OO programming constructs have the advantage of being open for possible further extensions to interact with the widely developed object oriented solutions either for data engineering or data accessing (e.g., UML, Spring, hibernate, ..) as well as with other programming languages. Also ORDB triplestores do profit from the advantages offered by RDBMs. Ways are however still needed to extend such triplestores with object oriented graph APIs for an object oriented programming perspective within the context of RDF data.

6.2. Non-Relational triplestores

Some of the key factors that have motivated the looking for other types of RDF processing systems other than relational ones are of course the limitations of RDB systems with respect to the variability that needs dynamic and flexible schemas other than static RDB schemas.

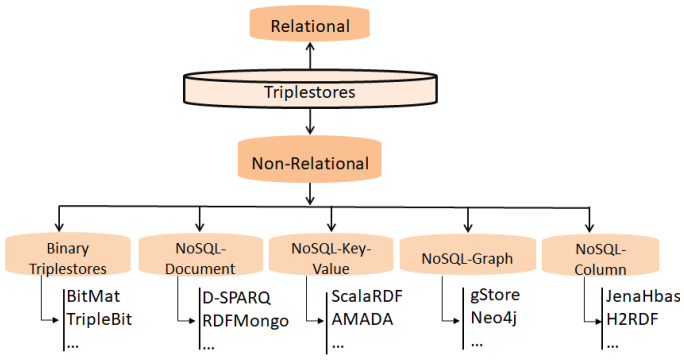

Figure 8: Category of Non-Relational Triplestores

Figure 8 illustrates the categorization of such systems whose sub-categories are presented in the following subsections.

6.2.1. Binary Triplestores

The class of binary stores consists of triplestores that use bits to encode RDF triples. BitMat [26] and TripleBit [27] are two examples of such stores.

In BitMat each line of the matrix is associated with one subject and each column is associated with one predicate. Each entry in the i-th line and j-th column of the matrix is a sequence of bits from the set {0,1} with only one bit 1 whose position k representing the presence of the triple (i-th subject, j-th predicate, k-th object). BitMat benefits from the use of 0-1 sequences for representing RDF triples to use them to compress the RDF data. Querying of RDF data is done in two steps in BitMat. Candidate matches are derived from the bit matrix in first step and the exact matches are returned in a second step. Though the advantages offered by the bit representation and the possible compression on it, this technique still needs however to tackle the problems faced by insertion or deletion of RDF triples.

6.2.2. NoSQL-Document triplestores

Document stores, like MongoDB and CouchDB, do use documents to persist data. A document is organized as a collection of fields where each field is associated with a set of values. Each one of the fields could be used as an index for data retrieval.

The RDF triplestores D-SPARQ [28] and RDFMongo [29] are examples of document-oriented store that use MongoDB.

Another NoSQL document solution for processing RDF data which we call CouchbaseRDF was presented in [30] to store RDF data in Couchbase (https://www.couchbase.com) which is a JSON-based document store.

6.2.3. NoSQL-Key-Value triplestores

The category of NoSQL-key-value triplestores consists of those RDF stores that use a NoSQL key-value database system for storing and querying RDF triples. NoSQL key-value database systems store data as collections of key/value pairs and offer get(key) and put(key, value) access methods to read and write data. Redis (REmote DIctionary Server - https:// github.com/redis/redis) and DynamoDB (https:// aws.amazon.com/fr/dynamodb) are examples of key-value NoSQL database management systems. Redis is an in-memory data management system. DynamoDB offers various characteristics such as replication and buck up of data and the possibility of its integration in web applications.

An example of NoSQL-key-value RDF stores using Redis is ScalaRDF which is also a distributed and memory-based store. As an example of triplestores using DynamoDB we have AMADA [31].

6.2.4. NoSQL-Graph triplestores

Graph oriented triplestores simply use the graph representation of RDF data and store these data as directed graphs where the nodes are either resources or literals and an edge starting from a node n1 to a node n2 is libeled with a predicate p to mean that (n1,p,n2) is a triple.

In the family of NoSQL-graph triplestores we have gStore [32], Dydra (http://dydra.com), AllegroGraph, BlazeGraph (http://blazegraph.com) and S2X (SPARQL on Spark with GraphX) [33].

The triplestore gStore is a centralized graph-oriented solution that uses bit-encoding to encode RDF triples as well as to encode SPARQL queries in the same way. Codes of SPARQL queries are then simply matched to the encoding list of RDF data. gstore has also been extended to a distributed solution called gStoreD.

AllegroGraph is a high performance triplestore that is continuously updated and extended. For data retrieval, it organizes data in Repositories, associates an identifier with each triple, and stores each triple as a quad composed of the values for subject, predicate, object and the graph to which the triple belongs. Furthermore, it uses all combinations of these 4 components to which the identifier is added as default indexes. A query in Allegrograph is first analyzed to determine the indexes that may be involved by the query. The actual indexes used by the query are dynamically identified in a second processing step. Allegrograph also supports reasoning and transaction management.

6.2.5. NoSQL-Column triplestores

The list of column triplestores comprises among others "Jena-HBase" [34], H2RDF+ [35], CumulusRDF [36] and Rya [37].

The triplestore "Jena-HBase" is built on top of HBase column store and is discussed in the Hadoop-nonnative Big Data triplestores category. H2RDF+ and CumulusRDF use the Cassandra column database. The triplestore Rya uses Accumulo.

7. Centralized versus Distributed Categorization

Various RDF stores have been designed to ensure efficient and scalable RDF query processing in a centralized way. Centralized systems manage the storage and querying on a single node. Hence, their main advantage is that they handle all operations locally. However they face the inconvenient of limited resources due to the using of a single machine.

Distributed triplestores use multiple machines for the storage and querying of RDF data. They have therefore the capability of handling large amounts of data.

Both categories with their characteristics are presented in the following subsections, respectively.

7.1. Centralized triplestores

Centralized triplestores use a single machine to handle RDF data. The centralization is of course with respect to data storage as well as SPARQL processing. Figure 9 presents the sub-categories of the centralized triplestores category.

As their name suggest, the main drawback of centralized triplestores is the lack of scalability and fault tolerance.

7.1.1. Desktop triplestores

With desktop triplestores we mean those RDF management systems that run on single desktop machine such as RDF-3X and Hexastore.

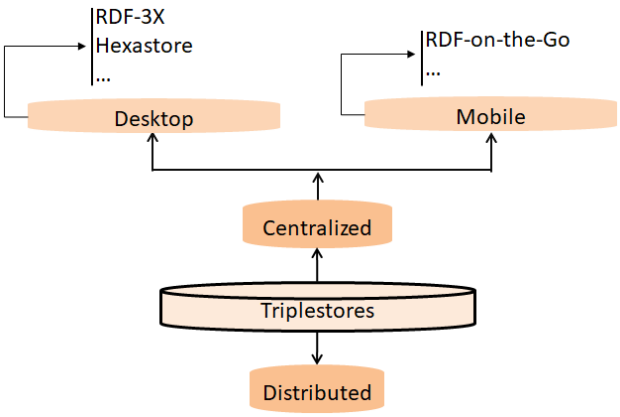

Figure 9: Category of Centralized triplestores

Hexastore combines the relational vertical representation approach with indexing capability to ensure fast querying of RDF triples. Indeed it uses each possible combination of the components "subject", "predicate" and "object" for indexing.

7.1.2. Mobile triplestores

This category of RDF stores consists of course of triplestores built especially for managing RDF data in mobile devices such as RDF-on-the-Go [38]. The flexibility and simplicity of the RDF data model make it as a good candidate for data interaction within and between such mobile devices.

RDF-on-the-Go is a full-edged RDF storage system that allows RDF storage and SPARQL query processing for mobile devices. RDF-on-the-Go relies on Jena and the Semantic Web Toolkit ARQ. It stores triples using the Berkeley DB. Its indexing strategy is based on the use of R-Trees.

7.2. Distributed RDF Triplestores

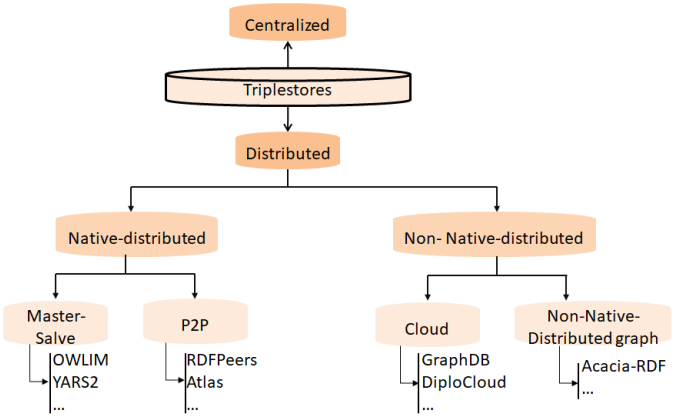

Figure 10: Categorization of Distributed Triplestores

Distributed triplestores are of course those systems that use more than one node to manage RDF data. The distribution concerns either the task of storage alone, the task of query processing alone or both tasks. Data distribution needs choosing efficient RDF data partition strategies that are also in accordance with the data retrieval modes chosen for querying the RDF data in order to achieve rapid RDF data manipulations. Issues involved are mainly related to data partition, data exchange between nodes, processing load partition and failure handling. The speed of RDF

data processing is mainly influenced by such issues. The strategies to address such issues have to be well chosen to better control the communications between nodes which can lead to high costs and to minimize data processing times within single nodes.

Figure 10 illustrates the distributed categories of triplestores.

7.2.1. Native-distributed triplestores

Native-distributed systems are considered here with respect to the distribution only and are those triplestores that come with their own distributing approaches for both storage distribution and query processing distribution.

7.2.1.1. Master-Slave native-distributed RDF stores

This category is composed of those triplestores that are built independently of any data management already existing solution and follow the master-slave distribution principle where RDF data management is in control of a master node that distribute management tasks to slave nodes. Examples of such triplestores are Virtuoso Cluster Edition, OWLIM [39], YARS2 [40], TriAD (Triple Asynchronous Distributed) [41].

OWLIM with its variants is developed with the java programming language and is a native RDF store. Its variant SwiftOWLIM is rather a memory based centralized triplestore. Its cluster version BigOwlim is distributed and contrary to other distributed triplestores handles deletion and insertion of RDF data more efficiently with the help of its indexing and partitioning strategy. It is currently developed under the new name of GraphDB (http://graphdb.ontotext.com) which also belongs to the category of cloud triplestores.

YARS2 is a native distributed RDF store. It proposes distributed indexing methods and three forms of indexes: Keyword index, six quad indexes and Join indexes.

TriAD also uses a classical master-slave architecture with a direct communication through the asynchronous exchange of messages. TriAD uses METIS graph partitioning with respect to subject and objects and associated combinations of indexes. Queries in TriAD are optimized using a summary graph that takes in consideration the result of the partitioning in order to execute queries directly only on concerned parts of the RDF graph.

7.2.1.2. P2P triplestores

P2P (peer to peer) defines a distributed model for a network of computers in which computers, also called nodes, play an equal autonomous role with regards to responsibility in the network and share their resources with the other nodes. In a P2P system, there is no single master node for managing the distribution traffic between the nodes. Computing services, data management and networking are offered in a decentralized way and are therefore not controlled centrally like in master-slaves networks. Beyond this decentralization, both fault-tolerance and scalability are the main advantages of P2P systems.

Examples of P2P based RDF data management systems are RDFPeers [42], Atlas [43], Edutella [44], RDFCube [45], GridVine [46] and UniStore [47].

The main problem faced by P2P triplestores is how to get a balanced distribution of RDF data between nodes for an efficient retrieval and querying of data and in order to avoid that some of nodes get heavily loaded with data more than other nodes.

Hashing is a common indexing solution that is used for distributing and tracking RDF data. Triplestores do however differ in their adopted hashing strategies. The hashing does of course guide the distribution but dependently of the used hashing method it however may lead to imbalances of load between nodes. In this case, the strategy is generally completed with a local split procedure at each node to achieve a uniform distribution of RDF data and therefore to a balanced querying of the RDF data. Once exceeding a threshold of stored data a node launches its split procedure to achieve a uniform data distribution.

Another crucial task for a P2P store is the maintenance of the hashing information during the processes of data suppression, update and insertion.

Apart from this burden caused by hashing tasks, generally speaking P2P triplestores beyond scalability show robustness with respect to fault tolerance and the advantage of not being centralized controlled.

7.2.2. Nonnative-distributed triplestores

The nonnative-distributed triplestores, as the name suggests, rely on existing distribution frameworks for the processing of RDF data. On one hand, we have those triplestores that use cloud solutions that are presented next, and on the other hand, we have triplestores that are relying on Big Data frameworks which are presented in section 8.

7.2.2.1. Cloud triplestores

During the last years, cloud computing has acquired more interest by users due to its flexibility, costs and availability of computing resources. Indeed numerous cloud computing providers have evolved and are offering numerous computing software and making available powerful machines to users. Furthermore, cloud computing has many advantages such as hiding from users all the complexity of distribution and handling of problems related to fault tolerance or others. Within the framework of RDF data management, numerous triplestores relying on cloud solutions have also been developed. Among these we have GraphDB (http:// graphdb. ontotext.com), AMADA, H2RDF [48], Rya, Stratustore [49] and DiploCloud [50].

GraphDB is an RDF database system that runs on the AWS cloud. It provides easy on-demand access for semantic metadata.

DiploCloud represents an RDF graph is generated on three main structures, namely RDF molecule clusters, template list and key index.

7.2.2.2. Nonnative-Distributed graph triplestores

This category is constituted of those RDF systems that use graph oriented solutions for RDF data management in a distributed scenario.

Acacia-RDF [51] is an example of such triplestores. It also has implementation of various algorithms for handling graphs. Furthermore, it can also be run on a single node. Acacia-RDF relies on the graph database solution Acacia and is programmed in the language X10. It can also be used as a centralized triplestore.

8. Big Data Triplestores

In recent years, various solutions have emerged for the processing of huge amounts of data with the use of clusters made up simply by commodity computers. Such solutions are also offering programming tools for accessing and processing the large data scattered in their distributed file systems based on well-defined frameworks.

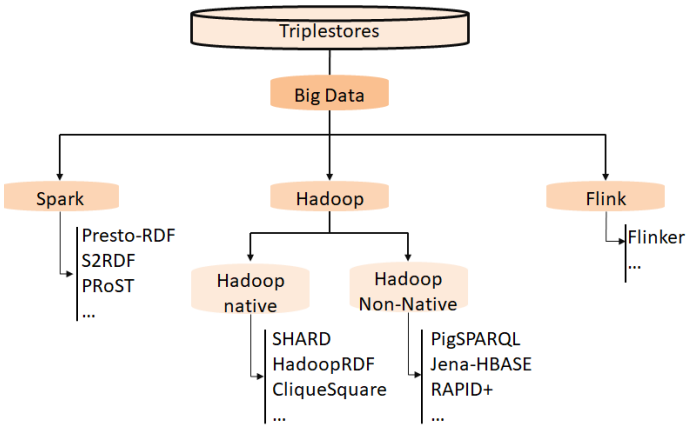

Figure 11: Category of Big Data Triplestores

The categorization of Big Data triplestores we are giving here is made with respect to the Big Data processing solution used by each one of these triplestores. More precisely, we distinguish between those triplestores that are based on Hadoop, Spark or Flink. The adoption of each of these systems by a triplestore will be clarified in the associated category subsection taking of course the characteristics of the system considered. The sub-categories within the Big Data category are presented in the following subsections and are illustrated in Figure 11.

8.1. Hadoop triplestores

The Hadoop triplestores are triplestores that are built on Hadoop HDFS (Hadoop Distributed File System) and Hadoop MapReduce programming framework for the storage and processing of RDF data.

In the following subsections we give the associated subcategories and highlight the main principles on which the storage structure and querying are based. Three subcategories are identified taken into account if they are relying on a direct use of HDFS and MapReduce completely or only partly with the use of other intermediary solutions.

8.1.1. Hadoop-native triplestores

Native HDFS-MapReduce triplestores are not relying on any already existing solution that uses Hadoop either for storing or querying data. They are built from scratch for the use of the HDFS file system to store RDF graphs and MapReduce for execution of SPARQL queries.

In the category of Hadoop-native triplestores we have SHARD [52], HadoopRDF [53] and CliqueSquare.

HadoopRDF stores RDF data triples into HDFS based on a predicate-oriented partitioning and performs decomposes queries respectively in MapReduce jobs. It keeps as many joins as possible in each job to reduce the number of jobs. This strategy can lead to high time costs especially when the value of predicates are unknown and multiple files have to be uploaded in this case to process queries.

SHARD is also a Hadoop-triplestore that distinguish itself by the subject oriented RDF data storage and an iterative query processing which is also subject-oriented. For each subject it stores all its triples with their predicates and objects in one line. For processing a query, it creates a pattern matching job for each triple pattern in the query and executes a join with the result computed up to this job. This strategy leads of course to enormous running times.

8.1.2. Hadoop-nonnative triplestores

With the category of Hadoop-nonnative triplestores, we mean those triplestores that directly use other existing HDFS/MapReduce general data management solutions for the handling of RDF data. Examples of RDF stores in this category are PigSPARQL [54] and RAPID+ [55], [56], “Jena-HBase” and “Hive+HBase” [57].

The triplestore “Hive+HBase”, for example, uses functionalities of HBase that uses HDFS for managing data and Hive that also offers a data warehousing module.

The reliance of Hadoop-nonnative triplestores on other existing Hadoop data storage and processing existing solutions is an advantage of such triplestores since such solutions are for use in a general context and offer therefore to the triplestores possible ways for further development with components related for example to integration of other data sources and for incorporating other functionalities related to data analytics and also to transaction management.

However, the major drawback for both Hadoop-native and Hadoop non-native triplestores is the high communication costs because of unavoidable disk input and output operations during the execution of the task of MapReduce jobs phases when dealing with massive RDF data. In the case of Hadoop-nonnative triplestores, the translation of SPARQL queries to the query languages of the engines on these triplestores rely also adds extra costs.

8.2. Spark based triplestores

Spark's solution is based on storing processed data and intermediate results in main memories of computing nodes and keeping a history of the computations for recovering lost data in case of failures. This let Spark enhances speed since the switching to disk is not frequent as it is in the case for Hadoop MapReduce executions. At the base of computation, Spark uses the so called Resilient Distributed Datasets (RDDs) which are collections of data partitioned into chunks distributed on the computing nodes and kept as much as possible in main memory. Such RDDs are represented as Java objects.

Spark also provides a module for SQL. SQL querying is done on RDDs which enables fast querying through the parallel computation offered by Spark across the nodes while benefiting from the use of memory to store RDDs. SQL querying on external data like Hive data is also done by loading such data into Spark RDDs.

Spark is used by the triplestores SPARQLGX [58], S2RDF [59], SPARQL-Spark [60], PRoST [61], TripleRush [62] and Presto-RDF [63].

The triplestore S2RDF ("SPARQL on Spark for RDF") tries to minimize times of query processing by reducing the amounts of data to keep in memory. For this, it uses a schema for RDF data that extends the predicate-oriented partitioning schema already presented in subsection 6.1.1.3 with additional pre-computed tables. The main idea behind this schema is to reduce the size of data to be loaded into memory when dealing with joins within the queries to be processed. This has the advantage of avoiding input-output hard disk operations since spark keeps data in memory for programs execution. For two distinct predicates tables T1 and T2, S2RDF pre-computes and stores into HDFDS three semi-join tables of those (s,o) pairs of T1 for which, respectively, s is a subject in the second, o is a subject in T2 and s is an object in T2. A limitation of S2RDF is the need for additional functionalities for the automatic launching of an efficient updating of the semi-join tables each time a deletion of some existing triples or an insertion of new ones happen.

With regards to the aforementioned characteristics of Spark based triplestores, such triplestores have the advantage over Hadoop ones of largely reducing RDF data processing costs since the input-output operations are largely reduced due to the fact that RDF data and intermediary data is mainly kept partitioned in memories of processing nodes during the processing stages.

8.3. Flink based triplestores

Flink is natively developed for data streaming and offers massive real time streaming functionalities. It also offers APIs for data mining operations on streams. Flink can be considered as a Big Data engine for event streaming while Spark can be considered as a Big Data engine for micro-batch streaming.

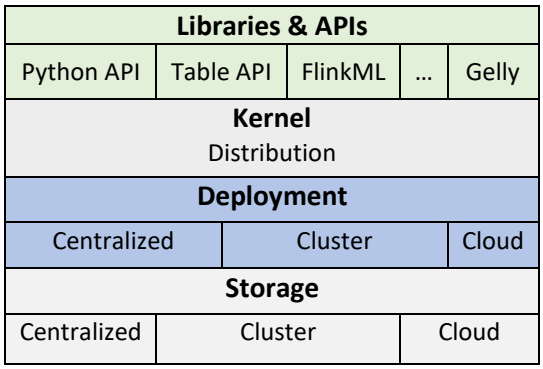

Figure 12: Flink architecture

Flink is developed in Java and Scala and provides an API for the processing of graphs called Gelly. Components of Flink are presented in Figure 12.

FLINKer [64] is an example of a triplestore that is based on Flink and provides therefore RDF data streaming. In FLINKer, Gelly graphs are built from RDF triples and then loaded in the Flink system to be handled. This graph representation is a strong positive point of FLINKer since it allows easy graph partitioning and distribution of data processing among nodes. For RDF data querying FLINKer uses Flink data processing operators on Gelly graphs to generate query optimization plans based on Flink parallelization contract programming approach (PACT) for a parallel execution.

Though these advantages of FLINKer, it still needs adding some functionality for more user involvement with regards to possible extensions of FLINKer with APIs for data representation based on Gelly graphs and for data analytics purposes. Also, FLINKer lacks elements for transaction management.

9. Stores for Constrained Devices

Micro computing has made it possible to integrate programmable modules with memories for data storage in devices with reduced capacity. Various devices with such modules have been developed in recent years for various applications (edge devices, sensors, etc.). The integration of RDF processing systems has also become possible for such small peripherals despite their limited memory capacity. In the category of triplestores for constrained devices we have µRDF store [65], RDF4Led [66] and Wiselib [67].

The µRDF store was developed with the aim to make the exchange and treatment of RDF data possible in the world of "internet-of-things" (IoT). It was tested for micro-controllers with memories ranging from 8 to 64 kB and with an internet connection. The tests include the storage of RDF data as well as SPARQL querying using basic SPARQL constructs.

RDF4Led, on the contrary, addresses RDF data exchange for lightweight edge devices. Such devices are largely common in IoT as well as in Cloud computing. The RDF4Led built-in system comprises a physical storage with an indexing strategy of triples, an intermediary buffering unit and a query engine. Efficiency of RDF4Led has been proven for devices with some hundreds of Mbytes in main memory and with a storage capacity of 16 GBytes.

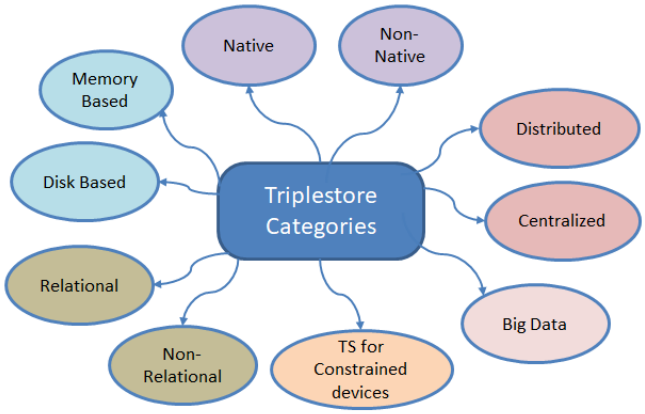

Figure 13: Main Categories of Triplestores

10. Comparison with Related Works

We notice that most existing works concentrate on a specific type of triplestores for reviewing or categorizing triplestores principally with limited characteristics or for comparing query processing times. We principally mention the works in [68] for the case of relational stores, in [69] for NoSQL stores, in [70] for P2P stores and in [71] for Big Data stores.

Contrary to these works, our approach comes with a consistent and detailed categorization with a focus on the storage and query processing characteristics. Figure 13 presents the main categories. As already mentioned, some of existing triplestores can be part of several categories.

For other issues related to detailed comparison criteria for RDF stores we refer to our work in [72].

11. Conclusion

The enormous acceptance of RDF in many fields has led to the development of various triplestores for the management of RDF data with each triplestore exhibiting its own characteristics. The variety of triplestores is of course a result of the variety of application use cases and of the various characteristics of data to be handled. Such characteristics are mainly related to data variety, to the volumes of data and to the data management tools and technologies. In this work we gave an extensive categorization of existing triplestores while identifying, for each established category its associated key features that make it to be treated separately, and presenting its underlying RDF data processing capabilities. We mainly focused on the data processing techniques used by the systems of each category as well as the modes of their deployment for the processing of RDF data and queries. The list of the different categories of triplestores is indeed established according to destination machines if they are for constrained devices, for desktops or for clusters, as well as depending on the technologies on which they are based: relational, non-relational, Cloud, P2P or Big Data. The categorization is also illustrated by reviewing within each category its representative RDF triplestores while highlighting advantages and disadvantages of the technology on which they are based in the context of RDF data characteristics and giving some suggestions for possible extensions.

With the given categorization, users will be able to identify the best suited triplestores for their use cases. Also, triplestores designers will be able to adequately focus on the relevant features to consider for the challenging task of design and development of RDF stores or to identify possible extensions of existing stores dependently on the targeted data management types and the tools at hand.

Conflict of Interest

The authors declare no conflict of interest.

12. References

[1] K. Alaoui, M. Bahaj, "Semantic oriented data modeling for enterprise application engineering using semantic web languages," International Journal of Advanced Trends in Computer Science and Engineering, 9(3), 3229–3236, 2020, doi:10.30534/ijatcse/2020/116932020.

[2] K. Alaoui, M. Bahaj, "Semantic oriented data modeling based on RDF, RDFS and OWL," Advanced Intelligent Systems for Sustainable Development (AI2SD'2019), **4** - Advanced Intelligent Systems for Applied Computing Sciences, M. Ezzyani (Ed.), Springer AISC 1105, 411–421, 2020, doi:10.1007/978-3-030-36674-2_42.

[3] K. Alaoui, "A categorization of RDF triplestores," Smart City Applications, SCA-2019, October 2–4, 2019, Casablanca, Morocco, ACM International Conference Proceeding Series, 2019, doi:10.1145/3368756.3369047.

[4] T. Neumann, G. Weikum, "RDF-3X: a RISC-style engine for RDF," VLDB, 1, 647–659, 2008, doi:10.1145/1453856.1453927.

[5] K. Wilkinson, C. Sayers, H. Kuno, D. Reynolds, "Efficient RDF storage and retrieval in jena2," Proceedings of the 1st International Conference on Semantic Web and Databases, SWDB 2003, 120–139, 2003.

[6] F. Goasdoué, Z. Kaoudi, I. Manolescu, J. Quiané-Ruiz, S. Zampetakis, "CliqueSquare: Efficient Hadoop-based RDF query processing," BDA'13 - Journées de Bases de Données Avancées, Oct 2013, Nantes, France. 2013. <hal-00867728>, https://hal.inria.fr/hal-00867728/document, 2013.

[7] F. Goasdoué, Z. Kaoudi, I. Manolescu, J.A. Quiane-Ruiz, S. Zampetakis, "CliqueSquare: Flat plans for massively parallel RDF queries," ICDE, 771–782, 2015, doi:10.1109/ICDE.2015.7113394.

[8] K. Zeng, J. Yang, H. Wang, B. Shao, Z. Wang, "A distributed graph engine for web scale RDF data," VLDB, **6**(4), 265-276, 2013, doi:10.14778/2535570.2488333.

[9] R. Harbi, I. Abdelaziz, P. Kalnis, N. Mamoulis, "Evaluating SPARQL queries on massive RDF datasets," 1848–1851, 2015, doi:10.14778/2824032.2824083.

[10] J. Wielemaker, W. Beek, M. Hildebrand, J. van Ossenbruggen, "ClioPatria: A SWI-Prolog Infrastructure for the Semantic Web," Semantic Web, **7**(5), 529–541, 2016, doi:10.3233/SW-150191.

[11] C. Hu, X. Wang, R. Yang, T. Wo, "ScalaRDF: a distributed, elastic and scalable in-memory RDF triple store," 22nd International Conference on Parallel and Distributed Systems, IEEE, 2016, doi:10.1109/ICPADS.2016.0084.

[12] O. Erling, I. Mikhailov, "RDF Support in the Virtuoso DBMS," In: Pellegrini T., Auer S., Tochtermann K., Schaffert S. (eds) Networked Knowledge - Networked Media. Studies in Computational Intelligence, 221. Springer, Berlin, Heidelberg, 2009, doi:10.1007/978-3-642-02184-8_2.

[13] S. Harris, N. Gibbins, "3store: efficient bulk RDF storage," First International Workshop on Practical and Scalable Semantic Systems, 2003.

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

[15] J. Broekstra, A. Kampman, F. van Harmele, "Sesame: A generic architecture for storing and querying RDF and RDF schema," The Semantic Web — ISWC 2002, (Editors: I. Horrocks and J. Hendler), Lecture Notes in Computer Science, 2342. Springer, 2002, doi:10.1007/3-540-48005-6_7.

[16] C. Weiss, P. Karras, A. Bernstein, "Hexastore: Sextuple indexing for semantic web data management," VLDB'08, August, 2008, Auckland, New Zealand, 2008 VLDB, ACM, 2008, doi:10.5167/uzh-8938.

[17] D. J. Abadi, A. Marcus, S.R. Madden, K. Hollenbach, "Scalable Semantic Web Data Management Using Vertical Partitioning," 33rd International Conference on Very Large Data Bases, 411–422. VLDB, 2007.

[18] M. Stonebraker, D.J. Abadi, A. Batkin, X. Chen, M. Cherniack, M. Ferreira, E. Lau, A. Lin, S. Madden, E. O'Neil, A. Rasin, N. Tran, S. Zdonik, "C-Store: a column-oriented DBMS," 31st International Conference on Very Large Data Bases, VLDB, 553–564, 2005, doi:10.1145/3226595.3226638

[19] D. Abadi, A. Marcus, S. Madden, K. Hollenbach, "SW-Store: a vertically partitioned DBMS for Semantic Web data management," VLDB Journal **18**(2), 2009, doi:10.1007/s00778-008-0125-y.

[20] Y. Wang, X.Y. Du, J.H. Lu, X.F. Wang, "FlexTable: using a dynamic relation model to store RDF data," 15th International Conference on Database Systems for Advanced Applications, 580–594, 2010, doi:10.1007/978-3-642-12026-8_44.

[21] M. Sintek, M. Kiesel, "RDFBroker: a signature-based high-performance RDF store," 3rd European Semantic Web Conference, 363–377, 2006, doi:10.1007/11762256_28

[22] H. MahmoudiNasab, S. Sakr, "AdaptRDF: adaptive storage management for RDF databases," International Journal Information Systems, **8**(2), 234–250, 2012, doi:10.1108/17440081211241978.

[23] C.-M. Chao, "An object-oriented approach for storing and retrieving RDF/RDFS documents," Tamkang Journal of Science and Engineering, **10**(3), 275-286, 2007, doi:10.6180/jase.2007.10.3.10.

[24] V. Bönström, A. Hinze, H. Schweppe, "Storing RDF as a graph," First Latin American Web Congress (LA-WEB 2003). IEEE, 2003, doi:10.1109/LAWEB.2003.1250279.

[25] E. Oren, R. Delbru, "ActiveRDF: Object-oriented RDF in Ruby," ESWC Workshop on Scripting for the Semantic Web, http://ceur-ws.org/Vol-181/Paper2.pdf, 2006.

[26] M. Atre, J. A. Hendler, "BitMat: A main memory Bit-matrix of RDF triples," 5th International Workshop on Scalable Semantic Web Knowledge Base Systems (SSWS'09). Citeseer, 33. (2009).

[27] P. Yuan, P. Liu, B. Wu, H. Jin, W. Zhang, L. Liu, "TripleBit: A fast and compact system for large scale RDF data," VLDB, **6**(7), 517-528, 2013, doi:10.14778/2536349.2536352.

[28] R. Mutharaju, S. Sakr, A. Sala, P. Hitzler, "D-SPARQ: distributed, scalable and efficient RDF query engine," ISWC (Posters & Demos), 261–264, 2013.

[29] M. Banane, A. Belangour, "RDFMongo: A MongoDB Distributed and Scalable RDF management system based on Meta-model," International Journal of Advanced Trends in Computer Science and Engineering,

8(3), ,2019, doi:10.30534/IJATCSE/2019/62832019.
[30] P. Cudré-Maurou, I. Enchev, S. Fundatureanu, P. Groth, A. Haque, A. Harth, F. L. Keppmann, D. Miranker, J. F. Sequeda, M. Wylot, "NoSQL databases for RDF: An empirical evaluation," The Semantic Web – ISWC 2013, (Editors: H. Alani et al.), Lecture Notes in Computer Science, 8219. Springer, 2013, doi:10.1007/978-3-642-41338-4_20.
[31] A. Aranda-Andújar, F. Bugiotti, J. Camacho-Rodríguez, D. Colazzo, F. Goasdoué, Z. Kaoudi, I. Manolescu, "AMADA: web data repositories in the Amazon cloud," 21st ACM International Conference on Information and Knowledge Management, CIKM'12, Maui, HI, USA, 2749–2751, ACM, 2012, doi:10.1145/2396761.2398749.
[32] L. Zou, M. T. Özsu, L. Chen, X. Shen, R. Huang, D. Zhao, "gStore: A graph-based SPARQL query engine," VLDB Journal, 23(4), 565-590, 2014, doi:10.1007/s00778-013-0337-7.
[33] A. Schätzle, M. Przyjaciel-Zablocki, T. Berberich, G. Lausen, "S2X: Graph-Parallel Querying of RDF with GraphX," VLDB Workshop on Big Graphs Online Querying, Big-O(Q). 2015, doi:10.1007/978-3-319-41576-5_12.
[34] V. Khadilkar, M. Kantarcioglu, B. Thuraisingham, P. Castagna "Jena-HBase: A distributed, scalable and efficient RDF triple store," International Semantic Web Conference on Posters & Demonstrations Track (ISWC-PD'12), Volume 914, 85–88, ACM, 2012, doi:10.5555/2887379.2887401.
[35] N. Papailiou, I. Konstantinou, D. Tsoumakos, P. Karras, N. Koziris, "H2RDF+: high-performance distributed joins over large-scale RDF graphs," IEEE International Conference on Big Data, October 2013, doi:10.1109/BigData.2013.6691582.
[36] G. Ladwig, A. Harth, "CumulusRDF: Linked Data Management on Nested Key-Value Stores." 7th International Workshop on Scalable Semantic Web Knowledge Base Systems (SSWS2011) at the 10th International Semantic Web Conference (ISWC2011), 2011.
[37] R. Punnoose, A. Crainiceanu, D. Rapp, "RYA: a scalable RDF triple store for the clouds," International Workshop on Cloud Intelligence. ACM, 4, 2012, doi:10.1145/2347673.2347677.
[38] D. Le-Phuoc, J. X. Parreira, V. Reynolds, M. Hauswirth, "RDF on the go: An RDF storage and query processor for mobile devices," ISWC-PD'10: Proceedings of the 2010 International Conference on Posters & Demonstrations Track - Volume 658, 149–152, 2010.
[39] A Barry Bishop, A. Kiryakov, D. Ognyanoff, I. Peikov, Z. Tashev, R. Velkov. "OWLIM: A Family of Scalable Semantic Repositories," Semantic Web, 2(1):33–42, 2011, doi: 10.3233/SW-2011-0026.
[40] A. Harth, J. Umbrich, A. Hogan, S. Decker, "YARS2: a federated repository for querying graph structured data from the web," in Proc. 6th International Semantic Web Conference, 211–224, 2007, doi:10.1007/978-3-540-76298-0_16.
[41] S. Gurajada, S. Seufert, I. Miliaraki, M. Theobald, "Triad: a distributed shared-nothing rdf engine based on asynchronous message passing," ACM SIGMOD, 2014, doi:10.1145/2588555.2610511.
[42] M. Cai, M. Frank, B. Yan, R. MacGregor, "A subscribable peer-to-peer RDF repository for distributed metadata management," Web Semantics: Science, Services and Agents on the World Wide Web 2, 109–130, 2004, doi:10.1016/j.websem.2004.10.003.
[43] Z. Kaoudi, M. Koubarakis, K. Kyzirakos, I. Miliaraki, M. Magiridou, A. Papadakis-Pesaresi, "Atlas: Storing, updating and querying RDF(s) data on top of DHTS," Journal of Web Semantics **8**(4), 271–277, 2010, doi:10.1016/j.websem.2010.07.001.
[44] W. Nejdl, B. Wolf, C. Qu, S. Decker, M. Sintek, A. Naeve, M. Nilsson, M. Palmér, T. Risch, "EDUTELLA: a P2P networking infrastructure based on RDF," In D. Lassner, D. De Roure, A. Iyengar, editors, Eleventh International World Wide Web Conference, WWW 2002, May 7-11, 2002, Honolulu, Hawaii, 604–615. ACM, 2002, doi:10.1145/511523.511525.
[45] A. Matono, S., Mirza, I. Kojima, "RDFCube: A P2P-based Three-dimensional Index for Structural Joins on Distributed Triple Stores," Databases, Information Systems, and Peer-to-Peer Computing, Trondheim, Norway, Springer, 2006, doi:10.1007/978-3-540-71661-7_31.
[46] K. Aberer, P. Cudre-Mauroux, M. Hauswirth, T.V. Pelt, "GridVine: Building Internet-Scale Semantic Overlay Networks," The Semantic Web – ISWC 2004, **3298**. Springer, 107–121, 2004, doi:10.1007/978-3-540-30475-3_9.
[47] M. Karnstedt, K. Sattler, M. Richtarsky, J. Muller, M. Hauswirth, R. Schmidt, R. John, "UniStore: Querying a DHT-based Universal Storage, 23rd International Conference on Data Engineering, ICDE 200, Istanbul, Turkey, 2007, doi:10.1109/ICDE.2007.369054.
[48] N. Papailiou and I. Konstantinou and D. Tsoumakos, N. Koziris, "H2RDF: Adaptive Query Processing on RDF Data in the Cloud," 21th International Conference on World Wide Web (WWW demo track), Lyon, France, 2012, doi:10.1145/2187980.2188058.
[49] R. Stein, V. Zacharias, "RDF on Cloud Number Nine," Proceedings of the 4th Workshop on New Forms of Reasoning for the Semantic Web: Scalable & Dynamic, 11-23. CEUR Workshop Proceedings, http://ceur-ws.org, 2010.
[50] M. Wylot, P. Cudré-Mauroux. "DiploCloud: Efficient and scalable management of RDF Data in the cloud," Transactions On Knowledge And Data Engineering, 2015, doi:10.1109/TKDE.2015.2499202.
[51] M. Dayarathna, I. Herath, Y. Dewmini, G. Mettananda, S. Nandasiri, S. Jayasena, T. Suzumura, "Introducing Acacia-RDF: An X10-Based Scalable Distributed RDF Graph Database Engine," 2016 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW), IEEE, 2016, doi:10.1109/IPDPSW.2016.31.
[52] K. Rohloff, R.E. Schantz, "High-performance, massively scalable distributed systems using the MapReduce software framework: the SHARD triple-store," Programming Support Innovations for Emerging Distributed Applications, 1-5, October 17-21, Reno, Nevada, 2010, doi:10.1145/1940747.1940751.
[53] M. F. Hussain, J. McGlothlin, M. M. Masud, L. Khan, B. Thuraisingham, "Heuristics-Based Query Processing for Large RDF Graphs Using Cloud Computing," TKDE, **23**(9), 1312 –1327, Sept. 2011, doi:10.1109/TKDE.2011.103
[54] A. Schätzle, M. Przyjaciel-Zablocki, G. Lausen, "PigSPARQL: Mapping SPARQL to Pig Latin," International Workshop on Semantic Web Information Management. SWIM '11, ACM, New York, NY, USA, 2011, doi:10.1145/1999299.1999303.
[55] P. Ravindra, H. Kim, K. Anyanwu, "An intermediate algebra for optimizing RDF graph pattern matching on MapReduce," In G. Antoniou, M. Grobelnik, E. Paslaru Bontas Simperl, B. Parsia, D. Plexousakis, P. De Leenheer, J.Z. Pan (Editors), The Semanic Web: Research and Applications - 8th Extended Semantic Web Conference, ESWC 2011, Heraklion, Crete, Greece, May 29 - June 2, 2011, Proceedings, Part II, 6644 of Lecture Notes in Computer Science, 46–61. Springer, 2011, doi:10.1007/978-3-642-21064-8_4.
[56] H. Kim, P. Ravindra, K. Anyanwu, "From SPARQL to MapReduce: The Journey Using a Nested Triple-Group Algebra," VLDB, **4**(12), 1426–1429, 2011, doi:10.14778/3402755.3402787.
[57] A. Haque, L. Perkins, "Distributed RDF Triple Store Using HBase and Hive," University of Texas at Austin, 139, 2012.
[58] D. Graux, L. Jachiet, P. Genevès, N. Layaïda, "SPARQLGX: efficient distributed evaluation of SPARQL with apache spark," ISWC, 2016, doi:10.1007/978-3-319-46547-0_9.
[59] A. Schätzle, M. Przyjaciel-Zablocki, S. Skilevic, G. Lausen, "S2RDF: RDF querying with SPARQL on spark," VLDB **9**(10), 804–815, 2016, doi:10.14778/2977797.2977806.
[60] H. Naacke, B. Amann, O. Curé, "SPARQL graph pattern processing with apache spark," Fifth International Workshop on Graph Data-Management Experiences and Systems, GRADES 2017, ACM, New York, 2017. , doi:10.1145/3078447.3078448.
[61] M. Cossu, M. Färber, G. Lausen, "PRoST: Distributed Execution of SPARQL Queries Using Mixed Partitioning Strategies," 21st International Conference on Extending Database Technology (EDBT), March 26-29, open proceedings, 2018, doi:10.5441/002/edbt.2018.49.
[62] P. Stutz, M. Verman, L. Fischer, A. Bernstein, "TripleRush: a fast and scalable triple store," 9th International Workshop on Scalable Semantic Web Knowledge Base Systems, Sydney, Australia, 21 October 2013 - 22 October, 2013, doi:10.5167/uzh-80646.
[63] M. Mammo, M. Hassan, S.K. Bansal, "Distributed SPARQL querying over big RDF data using PRESTO-RDF," International Journal of Big Data, **2**(3), 2015, doi:10.29268/stbd.2015.2.3.3.
[64] A. Azzam, S. Kirrane, A. Polleres, "Towards Making Distributed RDF Processing FLINKer," 2018 4th International Conference on Big Data Innovations and Applications (Innovate-Data), 2018, doi:10.1109/Innovate-Data.2018.00009.
[65] V. Charpenay, S. Käbisch, H. Kosch, "RDF Store: Towards Extending the Semantic Web to Embedded Devices," In: The Semantic Web: ESWC 2017 Satellite Events, **10577**, 76-80. Springer International Publishing, Cham, 2017, doi:10.1007/978-3-319-70407-4_15.
[66] A. Le-Tuan, C. Hayes, M. Hauswirth, D. Le-Phuoc, "Pushing the Scalability of RDF Engines on IoT Edge Devices," Sensors, **20**, 2020, doi:10.3390/s20102788.
[67] H. Hasemann, A. Kröller, M. Pagel, "The Wiselib TupleStore: A Modular RDF Database for the Internet of Things," CoRR, abs/1402.7228, 2014.
[68] Z. Ma, M. A. M. Capretz, L. Yan, "Storing massive Resource Description Framework (RDF) data: a survey," The Knowledge Engineering Review, **31**(4), 391–413, 2016, doi:10.1017/S0269888916000217.
[69] K. R. Saikaew, C. Aswamenakul, M. Buranarach, "Design and evaluation of a NoSQL database for storing and querying RDF data," KKU Engineering Journal. **41**, 537-545, 2014, doi:10.14456/kkuenj.2014.38.

[70] I. Filali, F. Bongiovanni, F. Huet, F. Baude, "A Survey of Structured P2P Systems for RDF Data Storage and Retrieval," In A. Hameurlain, J. Küng, and R. Wagner (Editors.): TLDKS III , LNCS 6790, 20–55, Springer, 2011, doi:10.1007/978-3-642-23074-5_2.

[71] M. Banane, A. Belangour, "An Evaluation and Comparative study of massive RDF Data management approaches based on Big Data Technologies," International Journal of Emerging Trends in Engineering Research, **7**(7), 2019, doi:10.30534/ijeter/2019/03772019.

[72] K. Alaoui, M. Bahaj, "Evaluation criteria for RDF triplestores with an application to Allegrograph," International Journal of Advanced Computer Science and Applications (IJACSA), **11**(6), 2020, doi:10.14569/IJACSA.2020.0110653.