A survey of RDF storage approaches

David C. Faye, Olivier Curé, Guillaume Blin

A survey of RDF storage approaches

David C. FAYE[*-**] — Olivier CURE[**] — Guillaume BLIN[**]

[*] Université Gaston Berger de Saint-Louis
BP 234 Saint-Louis
SENEGAL
david-celestin.faye@ugb.edu.sn

[**] Université Paris-Est
LIGM - UMR CNRS 8049
FRANCE
guillaume.blin@univ-mlv.fr | ocure@univ-mlv.fr

ABSTRACT. The Semantic Web extends the principles of the Web by allowing computers to understand and easily explore the Web. In recent years RDF has been a widespread data format for the Semantic Web. There is a real need to efficiently store and retrieve RDF data as the number and scale of Semantic Web in real-word applications in use increase. As datasets grow larger and more datasets are linked together, scalability becomes more important. Efficient data storage and query processing that can scale to large amounts of possibly schema-less data has become an important research topic. This paper gives an overview of the features of techniques for storing RDF data.

RÉSUMÉ. Le Sémantique étend les principes du Web en permettant aux ordinateurs de comprendre et d'explorer le Web de façon intelligente. Ces dernières années, RDF s'est largement répandu comme format de données du Web Sémantique. Il y a un réel besoin de stocker et de rechercher de façon efficiente les données RDF vu que le nombre et la taille des données du Web Sémantique utilisées dans les applications du monde réél est en progression continue. Etant donné que les sources de données sont de plus en plus volumineuses et de plus en plus liées, le passage à l'échelle devient plus que necessaire. Le stockage efficace de données et le traitement de requête à l'échelle de grande sources de données souvent sans schémas est devenu un sujet de recherche très important. Cet article présente les caractéristiques des différentes propositions pour le stockage de données RDF.

KEYWORDS : Semantic Web, RDF data storage.

MOTS-CLÉS : Web Sémantique, stockage de données RDF.

Received, December 12, 2011,
Accepted, February 7, 2012

ARIMA Journal, vol. 15 (2012), pp. 11-35.

1. Introduction

The Semantic Web extends the principles of the Web from documents to data and needs a unique way of specifying data and data relationships to bring machine-processable data to the World Wide Web. The goal is to allow computers understanding and easily exploring the Web. A standardized way to achieve this goal is to associate to each object of the data an URI identifier and provide descriptions of it in the so-called *schema*, *vocabularies, ontologies* or *data dictionaries.*

In recent years RDF has been a widespread data format for the Semantic Web. RDF is a graph-based data format which is schema-less, thus unstructured, and self-describing, meaning that the labels of the graph within the graph describe the data itself. The production of unstructured data in the semantic web domain is being wide-ranging in social semantic domains where often a fixed schema is not available a priori. This prevalence of RDF data is due to the variety of the underlying graph-based model, i.e. almost any type of data can be expressed in this format including relational and XML data. The variety of the data manipulated makes it hard to maintain a fixed schema. In the other side, the dynamic graph-structured character of Semantic Web is a hard issue for many traditional approaches like those of data indexing and querying.

There is a real need of efficient tools for storing and querying knowledge using the ontologies and the related resources. In this context, the annotation of unstructured data has become a necessity in order to increase the efficiency of query processing. Efficient data storage and query processing that can scale to large amounts of possibly schema-less data has become an important research topic. Consequently, a lot of work to improve the state of the art of semantic web proposals has been done. The proposed approaches usually rely on (object-) relational database technology or on main-memory virtual machine implementations, while employing a variety of storage schemes.

The goal of this survey is to briefly present and compare, in their theoretical aspects, a set of RDF storage tools designed for large-scale Semantic Web applications. We will give an overview of general system features while providing the interested reader with useful references to additional informative material. It should be stressed that the set of tools presented here is not exhaustive and that we do not make a detailed quantitative analysis of system performance, since this would require extensive comparative experiments.

Our contributions in this survey are essentially 1) an up to date taxonomy of RDF storage layouts, 2) the definition of a set of typical characteristics for RDF storing solution which provides a uniform comparison description framework and 3) the comparative study of a set of RDF storage solutions based on the characteristics mentioned previously. One should notice that, due to space consideration, we would not consider several related topics that fall outside of the scope of this survey; such as distributed RDF storage, query language characteristics.

The remainder of this paper is organized as follows. In section 2, we present the required background, the criteria we will take into account when presenting the RDF storage tools, a taxonomy of RDF storage layout and a set of related surveys. Section 3, presents on a set of non-native RDF storage techniques that are DBMS-based. In section 4, we introduce the native RDF storage proposals focusing on indexing techniques specific to the RDF data model. We will conclude by making a comparative study of a set of RDF storage solutions based on the features defined.

2. Background and preliminaries

In this section, we present the required background and the features we will take into account when presenting the RDF storage tools.

The design of a traditional database is guided by the discovery of *regularity* or *uniformity*. The principle of regularity is a standardization of design relying on an abstract view of the world, where exceptions to the rule are not taken into account, since they are considered as insignificant in the design of an advantageous structured schema. The popularity of relational database management systems (RDBMS) is due to their ability to support many data management problems dealt by applications. However the a priori uniformity required by the relational model can lead to hardness when modeling a not static world such as Semantic Web data.

The primary goal of RDF is to handle non regular or *unstructured* data. The research community has early recognized that there is an increasing amount of data that is insufficiently structured to support traditional database techniques, but does contain a sufficiently regular structure exploitable in the formulation and execution of queries [54].

2.1. RDF logical data model

The data manipulated in the Semantic Web is usually described as a set of specific resources each provided with a pair $(property, value)$. These descriptions are often called *triples* or *statements* and represent the data stored in Semantic Web applications. RDF is a logical data model consisting of triples of the form $\langle$s,p,o$\rangle$ where s, p and o are resp. called the subject, property and object of the triple.

Triple. A triple is an element $\langle$s,p,o$\rangle$, typically interpreted as a statement where "object o stands in relationship p with subject s". Hence, the first element s is called the subject of the triple, p is called the predicate, and o is called the object. The signature of an RDF triple corresponds to $(\mathtt{U} \cup \mathtt{B}) \times \mathtt{U} \times (\mathtt{U} \cup \mathtt{B} \cup \mathtt{L})$ where U, B and L are possibly infinite sets of respectively URI resources identifying the nodes, blank nodes (a form of existentially quantified variable) and RDF literals.

Triple store. The RDF graph model is used to describe this set of triples as a graph. Such data model is usually called *triple store* or RDF *database*.

The vocabulary of the data is defined implicitly by the data using the Concepts and Abstract Syntax specification published by the W3C[67]. A base semantics is defined rigorously in the RDF Semantics specification[68]. RDF Schema (RDFS)[69] is a standard built on top of RDF that allows developers to define standardized vocabularies. RDFS aims to provide a basic shared interpretation of RDF data to applications. Full details of the RDF data model can be found in the W3C standards [43, 57].

A piece of RDF graph describing athletics competitions is provided in Figure 1. For ease some edges have not been depicted. The relations *subclass* (sc), *subproperty* (sp), $type$, *domain* (dom) and $range$ are taken from the standardized RDFS vocabulary. For example, the triple $(BlankaVlasic, jumps, HighJump)$ means that $Vlasic$ is a high jumper. Moreover, it shows that in RDF specifications, schemas and data can be described at the same level therefore blurring the traditional separation between data and metadata. Note that the edges and nodes may share some labels; e.g. $jump$ is both a node and an edge label.

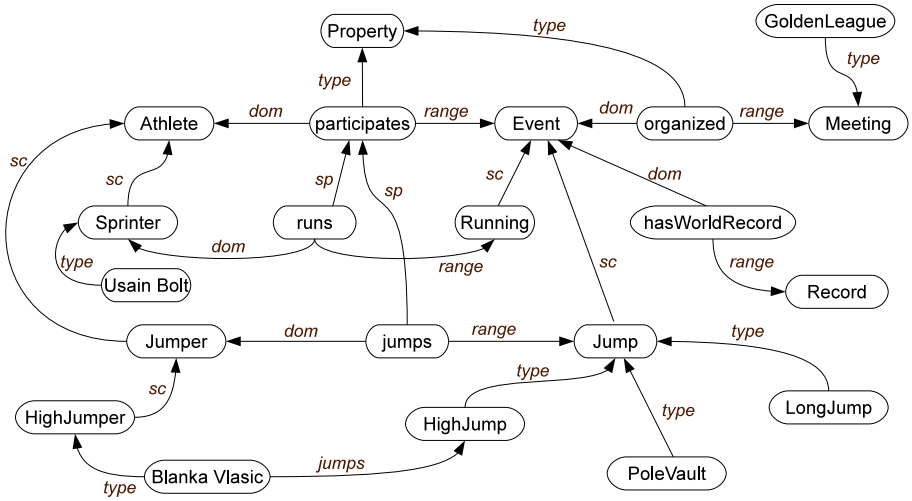

Figure 1. *A piece of* `RDF` *graph describing athletics competitions.*

The research community has early recognized the natural flexibility and expressivity of triples. Indeed, triples consider both objects and relationships as first-class citizens; thus, allowing on-the-fly generation of data. The power of `RDF` relies on the flexibility in representing arbitrary structure without a priori schemas. Each edge in the graph is a single *fact*, a single *statement*, similar to the relationship between a single cell in a relational table and its row's primary key. `RDF` offers the ability to specify concepts and link them together into an graph of data.

As a storage language, `RDF` has several advantages [56]. First, it is possible to link different data sources together by adding a few additional triples specifying relationships between the concepts. This would be more difficult in the case of an RDBMS in which a schema realignment or matching may be necessary. Then, `RDF` offers a great deal of flexibility due to the variety of the underlying graph-based model (*i.e.* almost any type of data can be expressed in this format with no needs for data to be present). There is no restriction on the graph size, as opposed to RDBMS field where schema must be concise. This a significant gain when the structure of the data is not well known in advance. Last, any kind of knowledge can be expressed in `RDF`, authorizing extraction and reuse of knowledge by various applications.

Consequently `RDF` offers a very useful data format, for which efficient management is needed. This becomes an hard issue for application dealing with `RDF` and known as `RDF` (or Triple) Stores, due to the irregularity of the data. `RDF` Stores must allow the following fundamental operations on repository of `RDF` data: performing a query, updating, inserting (assertion), and deleting (retraction) triples.

2.2. The problem of indexing `RDF` graphs

The goal of data indexing is to ease the search of and access to data at any given time. This is done by creating a data structure called *index* and providing faster access to the data. Accessing data is determined by the physical storage device being used. Indexing could potentially provide large increases in performance for large-scale analysis of unstructured data. Additionally the implementation of the chosen index must be suitable in

terms of index construction time and storage utilization. Furthermore, the search is done by using the concept of *Basic Graph Patterns* which are essentially conjunctive queries designed for the RDF data model

Basic Graph Pattern. Let $\mathcal{U}$ be a possibly infinite set of URI resources identifying the nodes. $\mathcal{U}$ is a set of atoms (e.g., Unicode strings). Let $\mathcal{V}$ be an enumerable set of variables, disjoint from $\mathcal{U}$. A *a triple pattern* (TP) is an element of $(\mathcal{U} \cup \mathcal{V}) \times (\mathcal{U} \cup \mathcal{V}) \times (\mathcal{U} \cup \mathcal{V})$. In other words, a TP is a triple(typically interpreted as a statement) in which roles (i.e., subjects, predicates, and objects) may be either atoms or variables. A basic graph pattern (BGP) is a *conjunction* $(s_1, p_1, o_1) \wedge \ldots \wedge (s_m, p_m, o_m)$ of one or more TPs.

BGPs are essentially conjunctive queries designed for the RDF data model. Based on the co-occurrence of variables and atoms and the different TPs composing a BGP, it is possible to make joins between TPs of a BGP [7]. There are six native BGP join types: *subject-subject*, *subject-predicate*, *subject-object*, *predicate-predicate*, *predicate-object*, and *object-object* joins.

In order to execute a given query against an RDF graph, an RDF store retrieves data for every triple pattern and joins it along the query edges. The efficiency of retrieval depends on the physical data organization and indexing. However the efficiency of joins may be determined by the join implementation and query optimization strategies. Thus, a problem faced by the RDF data store, once physically organizing and indexing data, is the following: *How to build an index data structure over an RDF graph to support efficient evaluation of graph patterns in it?*

The bottleneck in the querying of large RDF datasets is performing joins and unions. Most queries involve many such joins because an RDF dataset is a collection of simple three column tuples. Thus, reducing the needs and cost of joins and unions in query implementations is vital to support scalable query access.

When querying RDF data, seeking is no longer limited to matching keywords against documents. Instead, structured queries can be processed against web resources. In this regard, conjunctive queries represent an important way of querying RDF data, which essentially consist in a set of triple patterns of the form (s, p, o), where p is a predicate and s and o are variables or constants. These conjunctive queries have high practical relevance because they are capable of expressing a large portion of relational queries. An important majority of query languages used in practice fall into this fragment.

In order to query the stored data, one can use the following query languages: RDF Query Language (RQL) [41], RDF Data Query Language (RDQL) [60] and finally the W3C Recommendation SPARQL Protocol and RDF Query Language (SPARQL) [57].

Large-scale analysis of unstructured data has received a lot of attention recently with native or non-native solution for RDF data storage and indexing. The non-native solutions make usually the mapping onto a DBMS and then use the indexing techniques proposed in this field, while the native ones tend to design multiple indexing schemes closer to the RDF data model. Most of native systems depend on building efficient auxiliary indexes on the RDF data and using them to improve the overall query performance.

Let us now introduce the framework we adopt for describing these proposals.

2.3. Criteria for describing the proposals

The purpose of this section is to present the framework we have adopted for the survey of the different proposals of RDF data storage. The management of RDF data at large scale includes technical challenges for the storage layout, indexing, and query processing. Indeed, the lack of global schema and the variety of predicates represent the major problems for physical database design. Additionally, by the fine-grained modeling of RDF data (i.e having triples instead of entire records or entities), queries with a large number of joins will inherently participate to a large part of the workload. Nevertheless, the join attributes are much less predictable than in a relational setting [55]. In this paper, we focus on the storing and indexing of RDF data. The criteria falling into this evaluation framework are focused on the theoretical characteristics of the tools rather than their implementation details. We consider the following five facets.

First, we consider the **data storage mechanism** which refers to the system used to store the data. It can be an (object-)relational database technology or based on main-memory virtual machine implementations, while employing a variety of storage schemes.

Second, we consider the **inference support** that indicates whether the proposed approach allows arbitrary deduction rules for inferring new knowledge and not simply the recording of facts. There is a growing need for efficient queries that handle inference and deductive reasoning as more complex ontologies are developed. Note that, to the graph RDF applies a field of logic called *description logics*. This last provides a rigorous mathematical foundation enabling valid and consistent reasoning analogous to Relational Theory in Relational DBMS. On the other hand, RDF Schema (RDFS) as a lightweight ontology language, is gaining popularity and, consequently, tools for scalable RDFS inference and querying are needed. Note that, to address this issue of inferring new triples from a knowledge base, there exist two approaches. One can either, pre-compute them (at *compile-time*) a priori and then store inferred knowledge before querying. This approach has the advantage of avoiding the re-evaluation of the deduction rules for every query, but incurs a storage overhead and makes data updates harder. Additionally, it may be too expensive and broad to attempt to infer all possible knowledge. Otherwise, one can compute them on demand (i.e. at *run-time*) a posteriori and let the knowledge bases support inference as part of the query processing. This approach has less storage requirements, but its scalability is limited by the main memory space that is required for the run-time deductive rules computations. The choice between the two strategies may depend if the tool is concerned with query performance or with the performance of adding/updating knowledge to the database.

Third, we consider the **update support** that represents the efficiency of the proposal in modifying the contents of the underlying RDF storage system. Indeed, the data in a triple store needs to be manipulated - triples added, modified and deleted. One may argue that most RDF Stores are query intensive, even read-only ones. However, in some cases it may be desirable to load the RDF database in an incremental manner or to support updates such as inserting a new triple for annotating existing data.

Fourth, we consider the **scalability** which refers to the availability of the triple manipulated by Semantic Web applications to be several millions order of magnitude. Thus, Triple Stores must work on the scale of the Web where there might be complex graphs of relationships, including references to other resources. It is therefore necessary to Triple Stores to deal with large numbers of triples (millions).

Last, we consider the **network distribution** that relates to the distributivity of the RDF data over a cluster. Centralized RDF Triple Stores have limitations on both their failure tolerance and scalability. When managing remote resources or graphs, triple stores must handle network failure, bandwidth and system usage, autonomy and the volatility of data sources.

The general tend of our criteria is to evaluate the function flexibility of the triple stores proposals, since they are dealing with a very general descriptive format.

2.4. Taxonomy of RDF storage layout

An efficient RDF storage schema should offer both scalability in its data management performance and variety in its data storage, processing and representation. To meet the storage and querying needs of large scale RDF stores, numerous systems have been developed. The related work about RDF data management systems can be subdivided into two categories depending on their ability to be RDF model compliant: the *native* (resp. *non-native*) that are (resp. are not) RDF model compliant [62]. An classification of RDF data storage approaches is proposed in Figure 2.

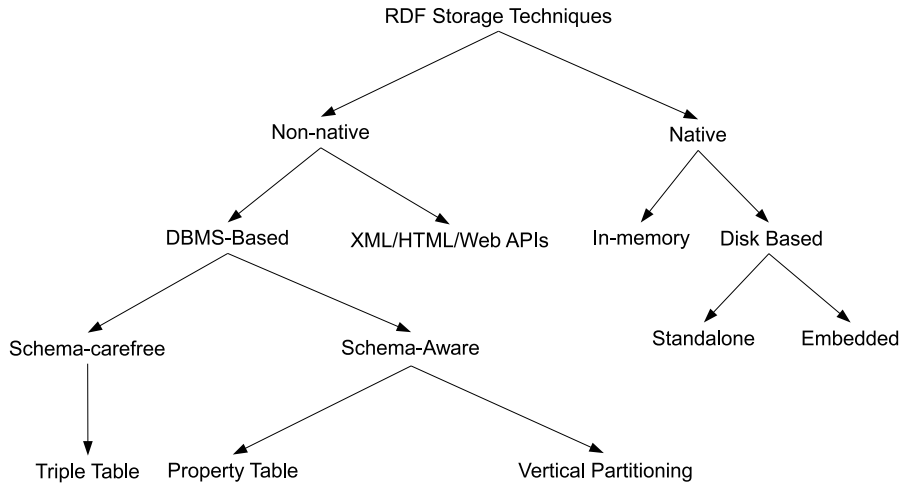

Figure 2. *A classification of RDF data storage approaches*

The **non-native storage** solutions make use of Database systems or others related systems to store RDF data permanently. Among others, let us cite GRDDL, Microformat, RSS feeds.

GRDDL (Gleaning Resource Descriptions from Dialects of Languages) [21] is a technique for obtaining RDF data from XML documents and in particular XHTML pages. The same applies to microformat[1] which is a web-based approach to semantic markup that seeks to re-use existing HTML/XHTML tags to convey metadata and other attributes, in web pages and other contexts that support (X)HTML, such as RSS. This approach allows information intended for end-users (such as contact information, geographic coordinates, calendar events) to also be automatically processed by software. Syndicated feeds as RSS

1. http://microformats.org/

2.0(most commonly expanded as *Really Simple Syndication*) is are another source. RSS is an XML-based format that can be used in different ways for content distribution including full or summarized text, plus metadata such as publishing dates and authorship. Note that, when using the name RSS, the speaker may be referring to any of the following versions of Web content syndication: *RDF Site Summary* (RSS 0.9, RSS 1.0), *Rich Site Summary* (RSS 0.91, RSS 1.0) or *Really Simple Syndication* (RSS 2.0).

Finally, efficient storage of RDF data in DBMS providing a SPARQL end point has already been discussed in the literature with different physical organization techniques, namely the *schema-carefree* and the *schema-aware*. In *schema-carefree*, a single table is used for storing both RDF/S schema and resource descriptions under the form of triples $\langle subject, predicate, object \rangle$. This approach is also called *Triple Table*. In *schema-aware*, unlike the previous representation that is quite straightforward, the schemas properties are used to split the triple table into different tables based on the RDFS schema properties or classes. We can distinguish two major schemas, namely *property table* and the *vertical partitioning* approach.

The **native storage** solutions provide a way to store RDF closer to the data model, eschewing the mapping to a DBMS. It uses the triple nature of RDF as an asset. These systems can be broadly classified as *persistent disk-based* and *main-memory- based* systems. The *persistent disk-based* storage is a way to store RDF data permanently on file system. These implementations may use well known index structures, such as B-Tree. One should notice that accessing the disk slows the search process to an unacceptable level. We then distinguish the *standalone* and the *embedded* representations. On one hand, *standalone* RDF native representations can be stored, transmitted and processed by their own means without referring to a specific host. For example, an in-memory Document Object Model (DOM) representation of an RDF/XML document can be built by using a SAX parser. Among the existing solutions, one can mention N3, N-triples, Turtle, RDF/XML and Trix.

Notation 3 (N3) [15] is a very complex language used to store RDF-Triples, which was issued in 1998. It is a language which is a compact and readable alternative to RDF's XML syntax that allow greater expressiveness. N-Triples [65] is a recommendation of W3C, published in 2004. It is a line-based plain text serialization format for RDF graphs based on a simplified version of N3 in order to reduce its complexity. The Terse RDF Triple Language (Turtle) [11] is an extension N-Triples which syntax is also used to define graph patterns in the query language SPARQL. RDF/XML [66] defines an XML syntax for representing RDF-Triples. TriX [17] is an experimental alternative serialization for expressing RDF triples in XML. It aims at providing a highly normalized consistent XML representation for RDF graphs, allowing the effective use of generic XML tools such as XSLT and XQuery.

On the other hand, *embedded* RDF native representations are part of a specific application/framework and only usable through this last. Their representation is only defined on the context of this framework. Here, the triples are produced by applying a transformation [33]. Among the existing solutions, one can mention RDFa, eRDF and SMW.

RDFa (RDF in attributes)[32] is a W3C recommendation that adds a set of attribute level extensions to XHTML for embedding rich metadata within web documents. The RDF data model mapping enables its use for embedding RDF triples within XHTML docu-

ments. It also enables the extraction of RDF model triples by compliant user agents. eRDF[2] (embedded RDF) is a syntax for writing HTML in such a way that the information in the HTML document can be extracted (with an eRDF parser or XSLT stylesheet) into RDF. It is a simplified approach for semantically annotating data in web sites. It allows most of the RDF model to be embedded without attempting to fulfill it. Semantic MediaWiki[3] (SMW) is a semantic wiki engine that enables users to add semantic data to wiki pages. This data can then be used for better searching, browsing, and exchanging of information. Most of the annotations that occur in SMW corresponds to simple ABox statements in OWL Description Logic.

The *in-memory* storage of RDF data allocates a certain amount of the available main memory to store the whole RDF graph structure. Like the persistent disk based storage, this approach relies on research results in the database domain (e.g., indices or efficient processing) and multiple indexing based techniques. When working on RDF data stored in main memory, some of the most time-consuming operations are the loading and parsing of the RDF file, but also the creation of suitable indexes. Therefore, an RDF Store must have a memory efficient data representation that leaves enough space for the operation of search algorithms. Some proposals falling into this category are, for example, Jena[50], Hexastore[71], and Bitmat[8] among others. An illustration of the compliance of the different solutions presented above to RDF data model is provided in Figure 3 which is adapted from [33].

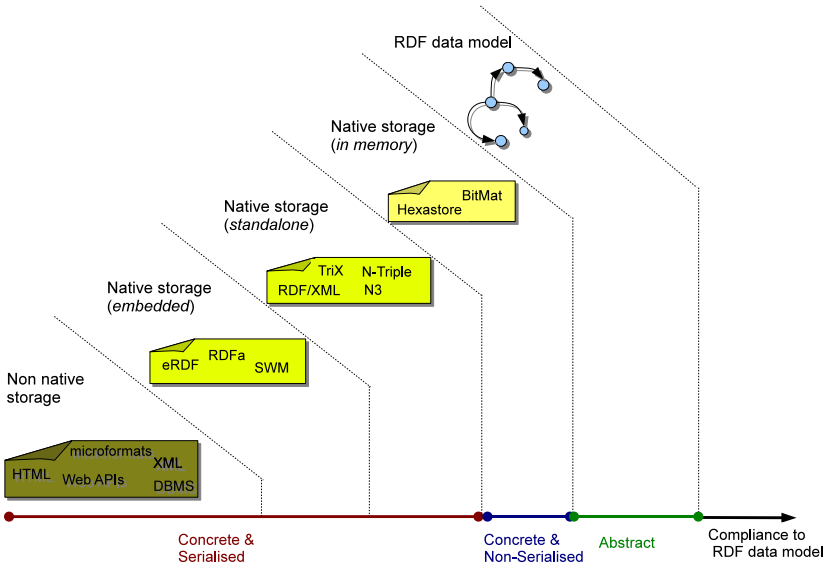

Figure 3. *Compliance of RDF storage layout to RDF data model*

2. http://research.talis.com/2005/erdf/wiki/Main/RdfInHtml
3. http://semantic-mediawiki.org/wiki/Semantic_MediaWiki

2.5. Related surveys

In the last few years, a number of technical reports and papers focused on RDF storage solutions have been published. Among them, we will described a set of previously published surveys.

In [47], Magkanaraki *et al.* described a wide range of tools in terms of storing, accessing and querying ontologies. The tools and query languages were described across a variety of features with performance figures, based on those provided by the tools. This survey defines and uses similar criteria to ours, such as query language, implementation language, inference support, API, export data format and scalability/performance properties. The survey considers at how the tools are appropriate for ontological applications rather than for more simpler Semantic Web.

In [12], Beckett considers open-source RDF storage systems and defines a set of criteria such as programming language and system, APIs, capacity/performance, query languages and inferencing. Despite the quality of this survey, first, the standards and the solutions have evolved quite significantly since 2002 and, moreover, other frameworks like multiple indexing and compliant RDF storage approaches have appeared. We will present this updates in the present contribution. In [10], Barstow uses a similar set of criteria. Nevertheless, information about these criteria are given for some storing systems, but no real comparisons have been performed.

In 2004, in the context of the SIMILE research project, a survey of open source RDF storage system was done by Lee [45]. It provides a complete performance evaluation for a set of chosen tools but does not clearly define additional criteria for the evaluation. In [62], Stegmaier *et al.* evaluated a selected set of RDF databases that support the SPARQL query language by the following means: general features such as details about software producer and license information, architectural and efficiency comparison of the interpretation of SPARQL queries on a scalable test dataset. Beckett *et al.* [14] extensively studied the use of RDBMS for semantic web storage, the issues and the schemas used.

In this contribution, we explore the existing Semantic Web data storage systems by taking into account the latest proposals in the field like the multi-indexing framework focusing on indexing techniques specific to RDF data model.

3. Non-native RDF data storage : DBMS based approaches

A set of techniques have been proposed for storing RDF data in relational databases. Currently, this is widely considered to be the best performing approach for their persistent data store due to the great amount of work achieved on making it efficient, extremely scalable and robust. Efficient storage of RDF data has already been discussed in the literature with different physical organization techniques such as *triple table*, *property table* and the *vertical partitioning* approaches. The strategy used by each tool depends on if the tool is concerned with query performance or with the performance of adding/updating knowledge to the database.

3.1. Triple table.

The *triple-table* approach is perhaps the most straightforward mapping of RDF into a RDBMS. Each RDF statement of the form $(subject, property, object)$ is stored as a triple in one large table with a three-columns schema (i.e. a column for the subject, predicate, and object resp.). Indexes are then added for each of the columns in order to make joins less expensive.

However, since the collection of triples are stored in one single RDF table, the queries may be very slow to execute. Indeed, when the number of triples scales, the RDF table may exceed main memory size. Additionally, simple *statement-based* queries can be satisfactorily processed by such systems, although they do not represent the most important way of querying RDF data. Nevertheless, RDF triples store scales poorly because complex queries with multiple triple patterns require many self-joins over this single large table as pointed out in [4, 71, 44]. The triple table approach has been used by systems like Oracle[19], 3store[28], Redland[13], RDFStore[58] and rdfDB[26].

3.2. Property table.

The property table technique has been introduced later on for improving RDF data organization by allowing multiple triple patterns referencing the same subject to be retrieved without an expensive join. In this model, RDF tables are physically stored in a representation closer to traditional relational schemas in order to speed up the queries over the triple stores[72, 19]. In this approach, each named table includes a subject and several fixed properties. The main idea is to discover clusters of subjects often appearing with the same set of properties. A variant of the property table, named *property-class table* [72, 74], uses the "rdf:type" property of subjects to cluster similar sets of subjects together in the same table.

The immediate consequence is that self-joins on the subject column can be avoided. However, the property table technique has the drawback of generating many NULL values since, for a given cluster, not all properties will be defined for all subjects. This is due to the fact that RDF data may not be very structured. A second disadvantage of property table is that multi-valued attributes, that are furthermore frequent in RDF data, are hard to express. In a data model without a fixed schema like RDF, its common to seek for all defined properties of a given subject, which, in the property table approach, requires scanning all tables.

Note that, in this approach, adding properties requires also to add new tables; which is clearly a limitation for applications dealing with arbitrary RDF content. Thus schema flexibility is lost and this approach limits the benefits of using RDF. Moreover, queries with triples patterns that involve multiple property tables are still expensive because they may require many union clauses and joins to combine data from several tables. This consequently complicates query translation and plan generation. In summary, property tables are rarely used due to their complexity and inability to handle multi-valued attributes.

This approach has been used by tools like Sesame[74], Jena2[4], RDFSuite[5] and 4store[29].

3.3. Vertical partitioning

The vertical partitioning approach suggested in swStore[2] by Abadi *et al.* is an alternative to the property table solution that speeds up queries over a triple store providing similar performance while being easier to implement. In this approach, RDF data are vertically partitioned using a fully decomposed storage model (DSM)[22]. Each triple table is divided into n two column tables where n is the number of unique properties in the data. In each of these tables, the first column contains the *subject* and the second column the *object* value of that subject. The tables are stored, by using a column-oriented DBMS[63] (i.e., a DBMS designed especially for the vertically partitioned case, as opposed to a row-oriented DBMS, gaining benefits of compressibility and performance), as collections of columns rather than collections of rows. The goal is to avoid reading entire row into memory from disk, like in row-oriented databases, if only a few attributes are accessed per query. Consequently, in column oriented databases only those columns relevant to a query will be read. Note that the approach creates materialized views for frequent joins. Furthermore, the authors suggest that the object columns of tables in their scheme can also be optionally indexed (e.g., using an unclustered B+ tree), or a second copy of the table can be created clustered on the object column

One of the primary benefits of vertical partitioning is the support for rapid *subject-subject* joins. This benefit is achieved by sorting the tables via subject. The tables being sorted by subject, one has a way to use fast merge joins to reconstruct information about multiple properties for subsets of subjects. Note that index-all approach is a poor way to simulate a column-store.

The vertical partitioning approach offers a support for multi-valued attributes. Indeed, if a subject has more than one object value for a given property, each distinct value is listed in a successive row in the table for that property. For a given query, only the properties involved in that query need to be read and no clustering algorithm is needed to divide the triples table into two-column tables.

Note that inserts can be slow in vertically partitioned tables since multiple tables need to be accessed for statement about the same subject. In [61], an independent evaluation of the techniques presented in [2], the authors pointed out potential scalability problems for the vertical partitioning approach when the number of properties in an RDF data-set is high. With a larger number of properties, the triple store solution manages to outperform the vertically partitioned approach. Note that there is no inference support in swStore.

As a first step to an efficient RDF storage road map, roStore proposes an intermediate ontology-guided approach which extends the vertical partitioning approach. roStore proposes a solution that outperforms the approach of swStore when reasoning over property hierarchies is necessary. The proposal is based on a set of semantic query rewriting rules to improve query performance by reasoning over the ontology schema of the RDF triples.

In this solution, there is a single table for each property hierarchy. Those tables have three columns (one for each element of the RDF triple). The rest of the tables follow the pattern design of the vertical partitioning approach. A direct consequence of this design is to reduce the number of tables for ontologies containing several property hierarchies, e.g. ontologies in the medical domain like OpenGalen. The reduction of the number of property tables has a big impact on the performance of queries requiring joins over properties of the same property hierarchy due to the generation and maintenance of less

relations than in swStore. Hence performance of an important number of queries requiring inferences on property hierarchies are improved since less joins are needed.

The aim of this approach is to try to analyze the efficiency of a compromise approach where less partitions are used. Intuitively, such physical organization will take benefits of requiring less joins in practical queries.

4. Native RDF data storage : Multiple indexing framework

4.1. Overview

Most of these approaches eschew the mapping to an RDBMS and focus instead on indexing techniques specific to RDF data model. They propose methods for rearranging data in memory or in given database systems such that query processing can be performed more efficiently compared to straightforward approaches like triple tables. They are motivated by the fact that using a traditional RDBMS for RDF data storage results in propagating RDBMS deficiencies such as inflexible schemas whereas avoiding these limitations is, arguably, one of the major reason for adopting the RDF data model [44]. The corresponding proposals aim to be closer to the query model of the Semantic Web.

The idea of multi-indexing is based on the fact that queries bound on property value are not necessarily the most interesting or popular type of queries encountered in real-world Semantic Web applications. Due to the triple nature of RDF data, the goal is to handle equally the following type of queries: *triples having the same property*, *triples having the same subject* and *finally list of subjects or properties related to a given object*.

For achieving this goal, these approaches maintain a set of six indices covering all possible access schemes an RDF query may require. These indexes are PSO, POS, SPO, SOP, OPS, and OSP (P stands for property, O for object and S for subject). These indices materialize all possible orders of precedence of the three RDF elements. At first sight, such a multiple-indexing would result into a combinatorial explosion for an ordinary relational table. Nevertheless, it is quite practical in the case of RDF data [71]. The approach does not treat property attributes specially, but pays equal attention to all RDF items. In the next section, we will present thirteen systems which appeared since 2005.

4.2. Multiple-indexing frameworks

The **YARS** [30] system combines methods from Information Retrieval and Databases to allow for better query answering performance over RDF data. It stores RDF data persistently by using six B+ tree indices. It not only stores the subject, the predicate and the object, but also the context information about the origin of the data. Each element of the corresponding quad (i.e., 4-uplet) is encoded in a dictionary storing mappings from literals and URIs to object IDs (OIDs-stored as number identifiers for compactness). To speed up keyword queries, the lexicon keeps an inverted index on string literals to allow fast full-text searches. In each B+ tree, the key is a concatenation of the subject, predicate, object and context. The six indices constructed cover all the possible access patterns of quads in the form $\langle$`s, p, o, c`$\rangle$ where `c` is the context of the triple $\langle$`s, p, o`$\rangle$. This representation allows fast retrieval of all triple access patterns. Thus, it is also oriented towards simple statement-based queries and has limitations for efficient processing of more

complex queries. The proposal sacrifices space and insertion speed for query performance since, to retrieve any access pattern with a single index lookup, each triple is encoded in the dictionary six times, in different sorting order. Note that inference is not supported.

The **Kowari** system[73] is an Open Source transaction safe, purpose-built database for the storage, retrieval and analysis of metadata. The system uses an approach similar to YARS. Indeed, the RDF statements are also stored as quads in which the first three items form a standard RDF triple and the fourth describes in which model the statement appears. The approach also uses six different orderings of quad elements acting as a compound index, and independently contains all the statements of the RDF store. In this ordering, the four quad elements can be arranged such that any collection of one to four elements can be used to find any matching statement or group of statements. However, Kowari uses an hybrid of AVL and B trees instead of B+ trees for multiple indexing purpose. Kowari solution also envisions simple statement-based queries like YARS.

The commercial system **Virtuoso** [24] stores quads combining a graph to each triple $\langle$s, p, o$\rangle$. It, thus, conceptually stores the quads in a triples table expanded by one column. The columns are g for graph, p for predicate, s for subject and o for object. While technically rooted in an RDBMS, it closely follows the model of YARS but with fewer indices. The quads are stored in two covering indices, g, s, p, o and o, g, p, s, where the URI's are dictionary encoded. Several further optimizations are added, including bitmap indexing. In this approach, the use of fewer indices tips the balance slightly towards insertion performance from query performance, but still favors query one.

RDF-3X [55] is an RDF storage system with advanced indexes and query optimization that eliminates the need of physical database design by the use of exhaustive indexes for all permutations of subject-property-object triples. Neumann *et al.* use a potentially huge triples table, with their own storage implementation underneath (as opposed to using an RDBMS). They overcome the problem of expensive self-joins by creating a suitable set of indexes. All the triples are stored in a compressed clustered B+ tree. The triples are sorted lexicographically in the B+ tree. The triple store is compressed by replacing long string literals in the triples IDs using a *mapping dictionary*. The system supports both individual update operations and entire batches updates.

Hexastore [71] takes also a similar approach to YARS. The framework is based on the idea of main-memory indexing of RDF data in a multiple-index framework. The RDF data is indexed in six possible ways, one for each possible ordering of the three RDF elements by individual columns. The representation is based on any order of significance of RDF resources and properties and can be seen as a combination of vertical partitioning [2] and multiple indexing approaches [30]. Two vectors are associated with each RDF element, one for each of the others two RDF elements (e.g., [subject,property] and [subject,object]). Moreover, lists of the third RDF element are appended to the elements in these vectors. Hence, a sextuple indexing schema is created. As Weiss *et al.* point out in [71], the values for O in PSO and SPO are the same. So in reality, even though six tables are created only five copies of the data are really computed, since the object columns are duplicated. To limit the amount of storage needed for the URIs, Hexastore uses the typical dictionary encoding of the URIs and the literals, i.e. every URI and literal is assigned a unique numerical identifier. Hexastore provides efficient single triple pattern lookups, and also allows fast merge-joins for any pair of two triple patterns. However, space requirement of Hexastore is five time the space required for storing statement in a triples table. Hexastore favors query performance over insertion time passing over applications

that require efficient statement insertion. Updates and insertions operations affect all six indices, hence can be slow. Note that Hexastore does not provide inference support. Recently, in [70], Weiss *et al.* proposed an on-disk index structure/storage layout so that Hexastore performance advantages can be preserved. Additionally to their experimental evaluations, they show empirically that, in the context of RDF storage, their vector storage schema provides significantly lower data retrieval times compared to B trees.

The system **RDFCube** [49] is a three-dimensional hash index designed for RDFPeers [16] which is a distributed RDF repository that efficiently search RDF triples. Each triple is stored by specifying its subject, predicate, or object as a key. The RDFCube storage schema consists of set of cubes of the same size called *cells*. Each of this cell contains a bit called existence flag indicating the presence or absence of triples mapped into the cell. During the processing of a query, by checking the existence flags of cells into which candidate answer triples are mapped, it is possible to know the existence of the triples before actually accessing remote nodes where the candidate answer triples are stored. This information helps reducing the amount of data that is transfered among nodes when processing a join query since it is possible to narrow down the candidate triples by using AND operator between existence flags bits and transfer only the actual present candidate triples. However, using a DHT(*Distributed Hash Table*) for the indexation suffers from some problems such as freshness of data and security.

BitMat [8] is a main-memory based bit-matrix structure for representing a large set of RDF triples with the idea to make the representation compact. Each RDF triple is considered as a 3-dimensional entity which conceptually gives rise to a single universal table holding all RDF triples. This last can be horizontally partitioned into multiple fragments based on the usage requirements. BitMat can be viewed as a 3-dimensional bit-cube, in which each cell is a bit representing a unique triple and denoting the presence or absence of that triple. For representing the bit-cube in memory, it is flattened in a 2-dimensional bit matrix. There are six ways of flattening a bit-cube into a BitMat. Each structure contributes to more efficient particular set of single-join queries. To deal with the inherent sparsity of BitMat, this latter is maintained as an array of bit-rows, where each row is a collection of all the triples having the same subject. The underlying goal is to represent large RDF triple-sets with a compact in-memory representation and supporting a scalable multi-join query execution. These queries are processed using bitwise AND, OR operations on the BitMat rows and the resulting triples are returned as another BitMat. BitMat is designed to be mainly a read-only RDF triple storing system. Dynamic insertion or deletion of RDF triples is not supported at present.

BRAHMS [39] is an main-memory based RDF storage system, specifically designed to support fast semantic association discovery (finding paths between two nodes in a RDF graph) in large RDF databases for which it uses graph algorithms like *depth-first-search* and *breadth-first-search*. BRAHMS has not been designed for modifications of RDF, but only for querying them. It employs six indexes i.e. two per dimension (e.g. subject dimension ordered on [predicate, object] and [object, predicate]) to speed up these queries. The triples of instance resources are indexed as follows $[subject \rightarrow object, predicate]$, $[object \rightarrow subject, predicate]$ and $[predicate \rightarrow subject, object]$. These indexes are needed for a fast retrieval of node neighborhoods, as well as for merging them during the semantic association discovery process.

The **RDFJoin** [51] project provides several new features built on top of previous cutting edge research including vertical partitioning [3] and sextuple indexing [71], namely

Hexastore. Hexastore is a main-memory solution whereas RDFJoin proposes a persistent column-store database storage for these tables with the primary goal to reduce the need and cost of joins and unions in query implementations. Indeed, it also use the six possible indexes on ⟨s, p, o⟩ using three tables: PS-O, SO-P and PO-S. These tables are indexed on both the first two columns so they provide all possible six indexes, while insuring that only one copy of the third column is stored. By keeping three separate triples tables and normalizing the identification numbers, RDFJoin allows subject-object and object-object joins to be implemented as merge joins as well. RDFJoin uses conversion tables closely matching the dictionary encoding of the vertical partitioning and Hexastore projects and the auxiliary mappings tables of the BitMat project. All the third column tuples are stored in a bit vector, and hash indexing based on the first two columns is provided. This reduces space and memory usage and improves the performance of both joins and lookups. For example, the PS-O table has columns $Property$, $SubjectID$ and $ObjectBitVector$ where $ObjectBitVector$ is a bit vector with the bits corresponding to all the object ID that appears in a triple with this property and subject. This also applies for the SO-P and the PO-S tables. Thus, all of the RDF triples in the dataset can be rendered from any of these tables. Additionally, execution of subject-subject, subject-object and object-object joins are done and stored as binary vectors into tables called *join tables*. This task is performed one time for any RDF dataset during the preprocessing stage to avoid overhead. Then, the results are stored in the relational database where they are quickly accessible. Indeed, RDFJoin stores much of its data as binary vectors and implements joins and conditions as binary set operations. This implementation provides significant performance improvement over storing each triple as a unique tuple. Let us remark that RDFJoin do support insertion of new RDF triples, but does not allow direct updates or deletions of triples in the database. Moreover, there is no suppport for inference in RDFJoin.

RDFKB [52] (Resource Description Framework Knowledge Base) is a relational database system for RDF datasets supporting OWL inference rules and knowledge management. The solution is implemented and tested using column-store and the RDFJoin [51] technology. It supports inference at data storage time rather than as part of query processing. All known inference rules are applied to the dataset to determine all possible knowledge. The core of the RDFKB design is that for each RDF triple, all possible additional RDF triples are inferred, stored and made accessible to queries. Mc Glothlin *et al.* made the choice to store redundant information. At query execution time, there is information about any knowledge related to the query, and this can be used to limit the scope of the inference search. Queries against inferred data are simplified and performance is increased. However, inferring all possible knowledge may be very expensive and the performance penalty can be high as the vocabulary is increased since more triples are persisted and loaded into memory. Moreover, to handle a query that do not need inference, the architecture builds a second copy of all tables, including all triples from the dataset but ignoring all triples added by the inference rules. The trade-off of this approach are added storage time, increased storage space requirements and increased memory consumption [52]. Note that, the architecture of RDFJoin tends to minimize the costs of these trade-off. RDFKB provides support for adding new triples. Moreover, inference is computed once a triple is added with the effect to increase the amount of time required to add and store new triples. On the other hand, it does not support transactions that delete or change triples in the dataset.

TripleT [25] is a Three-way Triple Tree secondary-memory indexing technique that facilitate flexible and efficient join evaluation on RDF data. The proposal pays attention

to data locality to avoid a piece of data to appear in multiple locations, spanning multiple data structures and consequently negatively impacting storage and query processing costs. Consequently, the index is built over the atoms occurring in the data set, rather than over whole triples occurring in the data set. Moreover, the atoms are indexed regardless of the roles (i.e., subjects, predicates, or objects) they play in the triples of the data set. The B+ tree secondary-memory data structure [20] is used to implement the various indexing techniques considered.

iStore [64] is an approach for data partitioning and join processing, which uses a structure index automatically built from the data. This index, while being basically a compact representation of the data graph, can act as a schema, enabling the effective browsing and querying of schema-less Web data. The structure index is a basic graph which can be computed for general `RDF` graphs to capture different structure patterns exhibit by the data. The vertices of this graph represent groups of data elements that are similar in structure – where "structure" refers to the set of incoming and outgoing connections. Thus, the schema properties are used to perform vertical partitioning, i.e. vertices are mapped to physical tables. This is done to obtain a contiguous storage of data elements that are structurally similar. The processing of a given query begins by matching it against the structure index in order to identify groups of data satisfying the overall query structure (in other words, to filter candidates through structure-level processing). Then, these data groups are retrieved and joined. This needs to be performed only for some parts of the query. In fact, structure-level processing helps to prune the query and then processing the pruned query using standard data-level operations. The whole query structure is taken into account for the retrieval of candidate data group instead of retrieving data for every single triple pattern using a vertical table. Finally, structure-level may be very helpful since, in the case where no candidates can be found in the structure index, data-level processing can be completely skipped. Compared to Vertical Partitioning (VP), where triples with the same properties are grouped together, the partitioning proposed by iStore results in the contiguous storage of triples that have the same structure.

Parliament [44] describes a storage and indexing schema based on linked lists and memory-mapped files with a storage structure composed of three parts: the *resource table*, the *statement table*, and the *resource dictionary.* The resource table is a single file of fixed-length records (sequentially numbered with numbers serving as ID of the corresponding resources), each of which representing a single resource or literal. This allows direct access to a record given its ID via simple array indexing. Each record has eight components :

– three statement ID fields representing the first statements that contain this resource as a subject, predicate, and object, respectively;

– three count fields containing the number of statements using this resource as a subject, predicate, and object, respectively;

– an offset used to retrieve the string representation of the resource;

– bit-field flags encoding various attributes of the resource.

The statement table, similarly to the resource table, is a single file of fixed-length records (with an ID per statement), each of which represents a single statement. To provide a one-to-one, bidirectional mapping between a resource and its resource ID, a dictionary is used. The first component of this dictionary is the mapping from a resource to its associated identifier. This portion of the dictionary uses Berkeley DB to implement a B-tree whose keys are the resources' string representations and whose values are the cor-

responding resource ID's. The second half of the dictionary is the reverse lookup from a resource ID to The current approach stores each string representation twice. The resource table, the statement table, and the string representations file are stored and accessed via a "memory mapping" independent of the index structure of the store. The mechanism for accessing files is optimized and keeps frequently accessed pages in memory. This scheme is designed to balance insertion performance, query performance, and space usage.[44] Parliament is designed as an embedded triple store and does not include a SPARQL or other query language processor.

5. Summary of some RDF storage proposals

Table 1 summarizes the storage schemes implemented by existing RDF/S stores. The summary is based on the criteria we have defined for describing the different proposals. Others approaches exist, like the distributed RDF repositories, but they are beyond the scope of our paper. Figure 4 indicates the influences between the different proposals over the time.

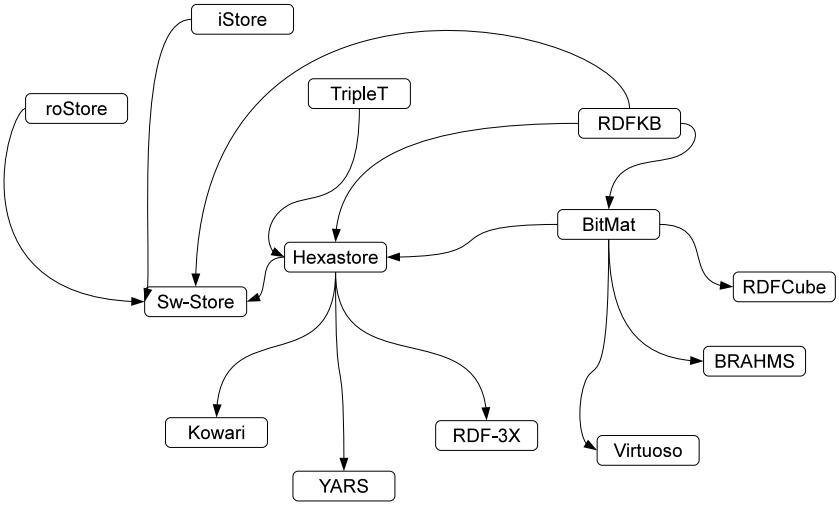

Figure 4. *Evolution of native RDF store. Edges indicates influences*

6. Conclusion and discussion

This paper presents a new taxonomy of RDF storage layouts, which builds on the previous surveys of RDF storage approaches and improve them. We have reviewed some of the recent proposals related to storing semantic web data, pointing which part of the taxonomy space they cover. Analysis of the state-of-art has shown many facets.

The proposed approaches usually rely on disk based storage(e.g. (object-) relational database technology, files) or on main-memory virtual machine implementations, while employing a variety of storage schemes. Most of the tools, especially the first proposals,

Store	Storage scheme					Storage support			Query Language	Update Support	Inference Support		Scala-bility	Distri-bution
	TT	PT	VP	HPP	MI	DBMS	File	In RAM			compile time	run time		
Jena2		√				√		√	RDQL/SPARQL	√		√		
RDFSuite		√				√			RQL	√	√	√	1,8 M	
Sesame		√				√	√	√	RQL	√	√		0,4 M	
3store	√					√			RDQL/SPARQL		√	√		
rdfDB	√					√		√	SquishQL-like	√	√	√		
RDFStore	√					√	√	√	SPARQL/RDQL		√	√		
Redland	√					√			SPARQL/RDQL	√				
AllegroGraph							√		SPARQL					√
sw-Store			√			√			SPARQL				55M	
roStore				√		√			SPARQL			√	1,3M	
4store		√				√			SPARQL	√	√			√
YARS					√		√	√	N3 extension	√			2,8M	
Kowari					√	√			iTQL/RDQL					
Hexastore					√	√		√	SPARQL				6M	
RDFJoin					√	√			SPARQL				44M	
RDFKB					√	√			-	√	√		44M	
BitMat					√			√	SPARQL-like				47M	
TripleT					√			√	-				6M	
RDF-3X					√	√			SPARQL	√			51M	
iStore					√		√		-				6M	
Parliament					√		√		-	√	√	√	4,5M	
Brahms					√			√	-				6M	
Virtuoso					√	√			SPARQL	√		√	1068M	√
RDFCube					√				-				0,1M	√

Table 1. *Characteristics of proposed RDF storage layout. TT=Triple Table, PT=Property Table, VP= Vertically Partitioning, HPP=Hierarchical Property Partitioning*

use relational technology - (O)RDBMSs - to store data and most of their associated query languages are triple based. However, in recent literature, the proposal focus on indexing techniques specific to RDF data model and rearranging data in-memory for efficient query processing.

The DBMS-based techniques relying on schema information, such as for data partitioning, largely improve RDF query processing. However, a schema, cannot always be found for Web data. In most of these systems, RDF data is decomposed into a large number of single statements that are directly stored in relational or hash tables. An immediate consequence of this approach is that, simple statement-based queries can be satisfactorily processed by such systems. A statement-based query lacks one or two parts of a triple, and the answer is a set of resources that complement the missing parts. However statement-based queries do not represent the most important way of querying RDF data[71]. Since more complex queries, involving multiple filtering steps are frequent, theses approaches are not efficient for such queries. In addition, most of the proposed triple stores have limitations either on their scalability or the specialization of their architecture for special kind of queries, or both. The memory based storage, or the mapping of the triple-based format to a relational database are surely serious limitations for the traditional approaches. In fact, with with the representation in giant triples table, serious scalability problems appear and they loose the opportunity to optimize triple storage, retrieval, and updates to the graph-based format.

To overcome some limitations of the triple centric approach a set of systems such as Jena create out of RDF data a set of *property tables* gathering together information about multiple properties over a list of subjects. However, these schemes lack of performance for queries that cannot be answered from a single property table, but need to combine data from several tables. Additionally, these systems have to manage sparse representation with many NULL values in the formed property tables because of the relational-like structure imposed on semi-structured RDF data, which leads to a significant computational overhead as opposed in the denser representations. Still, this approach did not escape the scalability defects.

The vertical partitioning model is one of the most saillant proposal in the set of non-native RDF storage proposals based on DBMS and focusing on taming the scalability drawback. This scheme is oriented towards answering queries in which the property resource is bound, or, otherwise, the search is limited to only a few properties. In fact, the two-column tables used by [2] are themselves a special variation of property tables too because they are related to the *multi-valued property tables* introduced in [72] ; namely, the latter also store single properties with subject and object columns[71]. In this respect, the most significant novelty of the vertical partitioning approach has been to integrate such two-column property tables into a column-oriented DBMS.

Therefore, the two-colum tables shares most of the disadvantages of those property-table solutions, for example, poorly handling queries that have unknown property values, i.e. in cases where a query does not restrict on property value, or when the value of the property is bound during query execution namely at runtime. All two-column tables will have to be queried and the results combined with either complex union clauses, or through joins[71]. In [27], the authors shows that the column store approach is especially suitable for data that needs to be optimized for read-only access. Still, this vertical partitioning model is strictly property-oriented. Arguably, property-bound queries are not necessarily

the most representative way of querying RDF data. Hence, the methodology becomes unsuitable or suboptimal for general queries, in which properties may not be bound.

While being straightforward in *schema carefree* approach (the triple table approach), schema evolution is hard in the *schema-aware* approach because, for example, the addition (deletion) of a new property requires at least the addition (deletion) of a table. This can result in a significant overhead when managing a potentially large number of tables constructed from huge RDF/S schemata. However, in the *schema carefree* approach type information are lost because all property values are usually stored as strings in the object attribute.

The idea of multi-indexing is based on the fact that queries bound on property value are not necessarily the most interesting or popular type of queries encountered in real-world Semantic Web applications. The goal is to handle equally, because of the triple nature of RDF data, the following type of queries : *triples having the same property*, *triples having the same subject*, and *finally list of subjects or properties related to a given object*. For achieving this goal these approaches maintain a set of six indices covering all possible access schemes an RDF query may require. At first sight, such a multiple indexing would result into a combinatorial explosion for an ordinary relational table, it is quite practical in the case of RDF data[71]. The approach does not treat property attributes specially, but pays equal attention to all RDF items. The approach can be main-memory based or disk-based.

The main-memory based approaches allocate a certain amount of the available main memory to store the whole RDF graph structure. When storing RDF data, the memory usage restriction requires to use a compact representation of the triples, nodes, their values, and storing only the most necessary structural data. On the other hand, the speed requirement demands creating suitable indexes for fast access and search. Beyond optimizing in-memory indices, most of the systems still make use of traditional data structures, such as B-trees, when it comes to on-disk storage. B-tree has become the dominant index structure in database systems and B-trees are a well understood data structure and have good properties regarding inserts and deletions. In [70] the authors claim that in the case of triple-stores the usage of B-trees is actually highly detrimental to query performance

Centralized RDF triple stores have limitations on both their failure tolerance and scalability. Therefore, RDF query processing in a P2P environment is an important issue. Realization of extremely large improvements in scalability will inevitably require a move towards clustered stores.

In [46], it has been reported that native stores reduce load and update time due to physical organization and indexes that are more tailored to RDF. DB-based solutions provide better query optimization, due to the great amount of work achieved on making relational query processing efficient. There has been a lot of work to improve the state of the art of RDF data storage and it seems there is no single approach, but rather a combination of different concepts that makes up the state of the art in RDF data management. In particular, vertical partitioning [2] is the candidate for physical data organization, multiple indexes [30, 71] enable fast lookup, and tailored query plans [55] result in fast performance for complex join processing.

Finally, we believe that there should not be a unique generic solution to RDF storage and that depending on the data itself, the underlying ontology, application queries, better performance may be obtain by considering alternative and several dedicated approaches.

7. References

[1] DANIEL ABADI, SAMUEL MADDEN, MIGUEL FERREIRA , "Integrating compression and execution in column-oriented database systems", ,*SIGMOD '06: Proceedings of the 2006 ACM SIGMOD international conference on Management of data*, Chicago, IL, USA, 2006.

[2] DANIEL ABADI, ADAM MARCUS, SAMUEL MADDEN, KATE HOLLENBACH, "Scalable semantic web data management using vertical partitioning", *VLDB '07: Proceedings of the 33rd international conference on Very large data bases, Vienna, Austria*, 2007.

[3] DANIEL J. ABADI, ADAM MARCUS , SAMUEL MADDEN, KATE HOLLENBACH, "SW-Store: a vertically partitioned DBMS for Semantic Web data management", *,VLDB J.*2009.

[4] KEVIN WILKINSON, CRAIG SAYERS, HARUMI A. KUNO, DAVE REYNOLDS, "Efficient RDF Storage and Retrieval in Jena2", *SWDB*,2003.

[5] SOFIA ALEXAKI, VASSILIS CHRISTOPHIDES, GREG KARVOUNARAKI , DIMITRIS PLEXOUSAKIS , KARSTEN TOLLE, "The ICS-FORTH RDFSuite: Managing Voluminous RDF Description Bases", *2nd International Workshop on the Semantic Web (SemWeb'01)*,Hongkong, 2001.

[6] RENZO ANGLES, CLAUDIO GUTIERREZ, " Querying rdf data from a graph database perspective", *In Proceedings of the Second European Semantic Web Conference*,2005.

[7] MARCELO ARENA, CLAUDIO GUTIERREZ, JORGE PÉREZ , " Foundations of RDF Databases", *Reasoning Web. Semantic Technologies for Information Systems: 5th International Summer School 2009, Brixen-Bressanone, Italy*,2009.

[8] MEDHA ATRE, JAGANNATHAN SRINIVASAN , JAMES A. HENDLER, " BitMat: A Main Memory RDF Triple Store", Technical Report, 2009.

[9] VALERIE BÖNSTRÖM, ANNIKA HINZE , HEINZ SCHWEPPE , "Storing RDF as a Graph ", *LA-WEB '03: Proceedings of the First Conference on Latin American Web Congress*,Washington, DC, USA,2003.

[10] ART BARSTOW, " Survey of RDF/Triple Data Stores", http://www.w3.org/2001/05/rdf-ds/DataStore, 2001.

[11] DAVID BECKETT, "Turtle-terse rdf triple language ", http://www.dajobe.org/2004/01/turtle/,November, 2007.

[12] DAVE BECKETT, "Scalability and Storage: Survey of Free Software / Open Source RDF storage systems ", *SWAD-Europe* 10.1,2002.

[13] DAVID BECKETT, "The design and implementation of the redland RDF application framework ", *WWW '01: Proceedings of the 10th international conference on World Wide Web*,New York, NY, USA,2001.

[14] DAVE BECKETT , JAN GRANT, "Mapping Semantic Web Data with RDBMSes ", *SWAD-Europe*,January, 2003.

[15] TIM BERNERS-LEE, "Notation 3 ", http://www.w3.org/DesignIssues/Notation3,March, 2006.

[16] MIN CAI , MARTIN FRANK, " RDFPeers: a scalable distributed RDF repository based on a structured peer-to-peer network", *WWW '04: Proceedings of the 13th international conference on World Wide Web*,New York, NY, USA, 2004.

[17] JEREMY CARROLL , PATRICK STICKLER, " TriX : RDF Triples in XML", HP Labs,2004.

[18] RICK CATTELL " Scalable SQL and NoSQL data stores", *SIGMOD Rec.*,New York, NY, USA, 2011.

[19] EUGENE INSEOK CHONG , SOURIPRIYAC DAS, GEORGE EADON, JAGANNATHAN SRINIVASAN, "An efficient SQL-based RDF querying scheme ", *VLDB '05: Proceedings of the 31st*

international conference on Very large data bases ,Trondheim, Norway, 2005.

[20] DOUGLAS COMER, "The ubiquitous B-tree ";, *ACM Computing Surveys*,1979

[21] CONNOLLY, "Gleaning Resource Descriptions from Dialects of Languages(GRDDL) ", http://www.w3.org/TR/2007/WD-grddl-20070302/,2007.

[22] GEORGE P. COPELAND, SETRAG N. KHOSHAFIAN , "A decomposition storage model ", *SIGMOD '85: Proceedings of the 1985 ACM SIGMOD international conference on Management of data*,New York, NY, USA, 1985.

[23] LUPING DING, KEVIN WILKINSON, CRAIG SAYERS, HARUMI KUNO, " Application-Specific Schema Design for Storing Large RDF Datasets", *In First Intl Workshop on Practical and Scalable Semantic Systems*,2003.

;

[24] ORRI ERLING , IVAN MIKHAILOV, "RDF Support in the Virtuoso DBMS. ", *Conference on Social Semantic Web*,2007.

[25] GEORGE H.L. FLETCHER, PETER W. BECK, " Scalable indexing of RDF graphs for efficient join processing", *CIKM '09: Proceeding of the 18th ACM conference on Information and knowledge management*,New York, NY, USA, 2009.

[26] R. V. GUHA , , "rdfDB : An RDF Database ", http://www.guha.com/rdfdb/,2000.

[27] STAVROS HARIZOPOULO , VELEN LIANG , DANIEL J. ABADI, SAMUEL MADDEN, , " Performance tradeoffs in read-optimized databases", *VLDB '06: Proceedings of the 32nd international conference on Very large data bases*,Seoul, Korea, 2006.

[28] STEPHEN HARRIS , NICHOLAS GIBBINS, " 3store: Efficient Bulk RDF Storage", 2003.

[29] S HARRIS, N LAMB , N SHADBOL , "4store: The design and implementation of a clustered rdf store ", *In SSWS2009: Proceedings of the 5th International Workshop on Scalable Semantic Web Knowledge Base Systems*, 2009.

[30] ANDREAS HARTH , STEFAN DECKER, " Optimized Index Structures for Querying RDF from the Web", *LA-WEB '05: Proceedings of the Third Latin American Web Congress*, Washington, DC, USA, 2005.

[31] ANDREAS HARTH, JÜRGEN UMBRICH, AIDAN HOGAN, STEFAN DECKER, "YARS2: A Federated Repository for Querying Graph Structured Data from the Web", *Proceedings of the 6th International Semantic Web Conference and 2nd Asian Semantic Web Conference (ISWC/ASWC2007)*, Busan, South Korea , 2007.

[32] MICHAEL HAUSENBLAS , BEN ADIDA , "RDFa in HTML Overview ", http://www.w3.org/2006/07/SWD/RDFa/,2007.

[33] MICHAEL HAUSENBLAS, WOLFGANG SLANY , DANNY AYERS, " A Performance and Scalability Metric for Virtual RDF Graphs", *in 3 rd Workshop on Scripting for the Semantic Web (SFSW07)*,2007.

[34] JONATHAN HAYES, CLAUDIO GUTIERREZ, " Bipartite Graphs as Intermediate Model for RDF", *In Proc. of the 3th Int. Semantic Web Conference (ISWC), number 3298 in LNCS*,2004;

[35] PATRICK HAYES, , "RDF Semantics ", http://www.w3.org/TR/rdf-mt/ ,February, 2004.

[39] MACIEJ JANIK, KRYS KOCHUT , " BRAHMS: A WorkBench RDF Store and High Performance Memory System for Semantic Association Discovery", *In Fourth International Semantic Web Conference*,2005.

[40] GREGORY KARVOUNARAKIS , SOFIA ALEXAKI, VASSILIS CHRISTOPHIDES , DIMITRIS PLEXOUSAKIS , MICHEL SCHOLL, " RQL: a declarative query language for RDF", *WWW '02: Proceedings of the 11th international conference on World Wide Web*,Honolulu, Hawaii, USA, 2002.

[41] GREGORY KARVOUNARAKIS, SOFIA ALEXAKI, VASSILIS CHRISTOPHIDES, DIMITRIS PLEXOUSAKIS , MICHEL SCHOLL "RQL: a declarative query language for RDF ", *WWW*

'02: Proceedings of the 11th international conference on World Wide Web,New York,Honolulu, Hawaii, USA.

[42] CHRISTOPH KIEFER , ABRAHAM BERNSTEIN, MARKUS STOCKER , " The Fundamentals of iSPARQL: A Virtual Triple Approach For Similarity-Based Semantic Web Tasks", ,

[43] GRAHAM KLYNE , JEREMY J. CARROLL, "Resource Description Framework (RDF): Concepts and Abstract Syntax ", W3C Recommendation ,2004.

[44] DAVE KOLAS , IAN EMMONS , MIKE DEAN, "Efficient Linked-List RDF Indexing in Parliament ", *Proceedings of the 5th International Workshop on Scalable Semantic Web Knowledge Base Systems (SSWS2009)*,2009.

[45] RYAN LEE , " Scalability report on triple store applications", Massachusetts Institute of Technology, 2004.

[46] LI MA , CHEN WANG , JING LU, FENG CAO , YUE PAN, **Yong Yu**, " Effective and efficient semantic web data management over DB2", *SIGMOD '08: Proceedings of the 2008 ACM SIGMOD international conference on Management of data*,Vancouver, Canada, 2008.

[47] AIMILIA MAGKANARAKI, GRIGORIS KARVOUNARAKIS, VASSILIS CHRISTOPHIDES, PLEXOUSAKIS PLEXOUSAKIS , TA TUAN ANH, "Ontology Storage and Querying ", ICS-FORTH, 2002.

[48] AKIYOSHI MATONO, TOSHIYUKI AMAGASA , MASATOSHI YOSHIKAWA , SHUNSUKE UEMURA, "A path-based relational RDF database ", *ADC '05: Proceedings of the 16th Australasian database conference*, Newcastle, Australia, 2005

[49] AKIYOSHI MATONO, SAID PAHLEVI, ISAO KOJIMA , " RDFCube: A P2P-Based Three-Dimensional Index for Structural Joins on Distributed Triple Stores", SpringerLink - Book Chapter Databases, Information Systems, and Peer-to-Peer Computing, 2007.

[50] BRIAN MCBRIDE, " Jena: A Semantic Web Toolkit", *IEEE Internet Computing journal*, num. 6, 2002.

[51] JAMES MCGLOTHLIN , LATIFUR R KHAN, " RDFJoin: A Scalable of Data Model for Persistence and Efficient Querying of RDF Datasets", UTDCS-08-09, 2009.

[52] JAMES P. MCGLOTHLIN, LATIFUR R. KHAN, " RDFKB: efficient support for RDF inference queries and knowledge management", *IDEAS '09: Proceedings of the 2009 International Database Engineering, Applications Symposium*,Cetraro - Calabria, Italy, 2009.

[53] DONALD R. MORRISON , " PATRICIA—Practical Algorithm To Retrieve Information Coded in Alphanumeric", *J. ACM*, 1968.

[54] ANDRAE MUYS, "Building an EnterpriseScale Database for RDF Data", *Seminar. Netymon*,2007.

[55] THOMAS NEUMANN, GERHARD WEIKUM, " RDF-3X: a RISC-style engine for RDF", Proc. VLDB Endowment, 2008.

[56] ALISDAIR OWENS , "An Investigation into Improving RDF Store PerformanceAn Investigation into Improving RDF Store Performance ", PHD Thesis , UNIVERSITY OF SOUTHAMPTON, 2009.

[57] ERIC PRUD'HOMMEAUX, ANDY SEABORNE , " SPARQL Query Language for RDF (Working Draft)", W3C, 2007.

[58] ALBERTO REGGIORI, "RDFStore Perl API for RDF Storage ", 2002.

[59] THORSTEN SCHÜTT, FLORIAN SCHINTKE, ALEXANDER REINEFELD, "Scalaris: reliable transactional p2p key/value store ", *Proceedings of the 7th ACM SIGPLAN workshop on ERLANG*, Victoria, BC, Canada, 2008.

[60] ANDY SEABORNE, " RDQL - A Query Language for RDF", W3C (proposal), 2004.

[61] LEFTERIS SIDIROURGOS, ROMULO GONCALVES, MARTIN KERSTEN, NIELS NES , STEFAN MANEGOLD, " Column-store support for RDF data management: not all swans are

white", *Proc. VLDB Endow.*, 2008.

[62] FLORIAN STEGMAIER, UDO GRÖBNER, MARIO DÖLLER, HARALD KOSCH, GERO BAESE , "Evaluation of Current RDF Database Solutions ", Proceedings 10th International Workshop of the Multimedia Metadata Community on Semantic Multimedia Database Technologies (SeMuDaTe'09), 2009

[63] MIKE STONEBRAKER, DANIEL J. ABADI , ADAM BATKIN, XUEDONG CHEN, MITCH CHERNIACK, MIGUEL FERREIRA, EDMOND LAU, AMERSON LIN , SAM MADDEN, ELIZABETH O'NEIL , PAT O'NEIL , ALEX RASIN, NGA TRAN, STAN ZDONIK, " C-store: a column-oriented DBMS", *VLDB '05: Proceedings of the 31st international conference on Very large data bases*, 2005.

[64] THANH TRAN , GUNTER LADWIG, SEBASTIAN RUDOLPH, "iStore: Efficient RDF Data Management Using Structure Indexes for General graph Structured Data ", Institute AIFB, Karlsruhe Institute of Technology, 2009.

[65] W3C, " RDF Test Cases", W3C, 2004.

[66] W3C, " RDF/XML Syntax Specification (Revised)", W3C, 2004.

[67] W3C, "Resource Description Framework (RDF): Concepts and Abstract Syntax specification", *http://www.w3.org/TR/rdf-concepts/*,February, 2004.

[68] W3C , "RDF Semantics ", W3C,2004.

[69] W3C, " RDF Vocabulary Description Language 1.0: RDF Schema", 2004.

[70] CATHRIN WEISS, ABRAHAM BERNSTEIN " On-disk storage techniques for Semantic Web data - Are B-Trees always the optimal solution?", Proceedings of the 5th International Workshop on Scalable Semantic Web Knowledge Base Systems (SSWS2009), 2009.

[71] CATHRIN WEISS, PANAGIOTIS KARRAS, ABRAHAM BERNSTEIN "Hexastore: sextuple indexing for semantic web data management ", Proc. VLDB Endow., 2008.

[72] KEVIN WILKINSON, " Jena Property Table Implementation", HP Labs, 2006.

[73] DAVID WOOD "Kowari: A Platform for Semantic Web Storage and Analysis ", In XTech 2005 Conference, 2005.

[74] JEEN BROEKSTRA, ARJOHN KAMPMAN , FRANK VAN HARMELEN "Sesame: A Generic Architecture for Storing and Querying RDF and RDF ", 2002.