# Evaluation Criteria for RDF Triplestores with an Application to Allegrograph

Khadija Alaoui[1], Mohamed Bahaj[2]
LITEN Lab, Faculty of Sciences and Techniques
Hassan I University
Settat, Morocco

***Abstract*—Since its launching as the standard language of the semantic web, the Resource Description Framework RDF has gained an enormous importance in many fields. This has led to the appearance of a variety of data systems to store and process RDF data. To help users identify the best suited RDF data stores for their needs, we establish a list of evaluation and comparison criteria of existing RDF management systems also called triplestores. This is the first work addressing such topic for such triplestores. The criteria list highlights various aspects and is not limited to special stores but covers all types of stores including among others relational, native, centralized, distributed and big data stores. Furthermore, this criteria list is established taking into account relevant issues in accordance with triplestores tasks with respect to the main issues of RDF data storage, RDF data processing, performance, distribution and ease of use. As a study case we consider an application of the evaluation criteria to the graph RDF triplestore AllegroGraph.**

***Keywords*—RDF; RDFS; SPARQL; triplestore; big data; NoSQL; AllegroGraph**

## I. Introduction

The primary goal of the W3C (World Wide Web Consortium) standardized ontology language RDF (Resource Description Framework, [24]) and its query language SPARQL (SPARQL Protocol and RDF Query Language, [25]) is to enrich the Web with semantics by structuring data through linking. This goal was set up with the aim to transform the web from a web of documents to a web of intelligent data in order to allow applications to easily extract semantics from data. With the web of documents, there is a difficulty to intelligently follow the semantics of the data because of the lack of structure in the documents content ([9]). For these reasons, there has been a massive use of RDF for publishing data on the web during the last decade. The use of RDF has paved the way for new features and use by scientists and businesses. RDF has indeed been used for modeling and publishing of data in various fields such as health services [3], smart city services [7], Internet of Things [8] and Geography Information Systems (GIS) [23]. This use of RDF has also been accompanied by a rapid development of a multitude of data management systems, also called triplestores, for the storage and processing of RDF data. In the first years of RDF, storage and processing solutions for RDF data were developed based on the use of relational based management systems because of the successful developments of such systems that had been reached over many years. However, these relational solutions present many limitations because of multiple problems such as, among others, SPARQL to SQL (Structured Query Language) query conversion overhead for RDF data querying, complex joins processing imposed by the relational schema proposals for modeling RDF data, integration of other data sources and the handling of big amounts of data. To come up with solutions to the relational problems with regards to RDF data handling, various RDF data management systems have been proposed during the past decade ranging from NoSQL (Not only SQL) based systems through native triplestores to Big Data solutions.

The aim of this work is to give a complete list of evaluation and comparison criteria for RDF management systems. To this end, we first give a summarized categorization of existing triplestores while considering the motivations behind their use for handling RDF data. We identify the benefits of each identified category of systems and the challenges they are facing. In a second step, we establish and motivate an extended evaluation criteria list for triplestores taking into account their associated categorization and relevant aspects with respect to their tasks for handling RDF data.

With the established criteria list, we aim to provide users with detailed insights of the various RDF management systems and comparison aspects with regards to the various relevant issues of dealing with RDF data. Users will be able to differentiate between RDF management systems and identify the best suited triplestore to their data for their specific use cases.

Contrary to existing comparison works that mainly focus on response times of query processing for a limited number of RDF storage systems (e.g. [29], [32], [13]) our list of evaluation criteria for triplestores considers a large variety of aspects. Indeed, based on the categorization details we are considering, various issues related among others to storage models, data organization and data recovery, query processing, query optimization, concurrency, dynamicity, scalability, reasoning, data integration, data exchange, data portability, scalability, visualization and support of analytical functionalities. The detailed criteria list provides users with means to focus on the triplestores aspects that better fulfill their objectives while comparing triplestores. Users can indeed choose the right criteria to identify the drawbacks or the positive aspects of these triplestores.

The following sections are structured as follows. Section 2 presents the W3C standards RDF, RDFS (RDF Schema, [10]), OWL (Web Ontology Language, [18]) and SPARQL as well as a summary about triplestores categories. Sections 3 to 7 present

the main categories of comparison and evaluation criteria with motivations behind their associated criteria. Section 8 discusses the case of Allegrograph and Section 9 concludes this work.

## II. Semantic Web Standards and Related Work

In this section we present aspects of the semantic web standards RDF, RDFS, OWL and SPARQL ([24], [10], [18], [25]) as well as of associated existing management systems that help in guiding the identification of evaluation criteria for such systems. We also give an overview of research works that deal with comparison and evaluation of triplestores.

### A. RDF and SPARQL

RDF semantic language revolutionized the research domain of creation, engineering and processing of ontologies for sharing information on the web. It uses a flexible model where statements in RDF are simply modeled as a set of triples having the form of (S,P,O):=(Subject, Predicate, Object) where a subject represents a resource, an object can be either resource or a literal value and the relation between the Subject and Object is expressed by the Predicate. An object may also be a set of either resources or literals grouped together using RDF grouping constructs such as "RDF:bag", "RDF:seq" for an ordered list or "RDF:list". Literal values may have a type and XML types may be used as types of literals.

RDF data can be presented in different formats: XML, Turtle, N-Triples and the N3 (Notation 3). Fig. 1 gives an RDF example using N3 and XML formats. RDF resources and predicates may be endowed with URIs (Uniform Resource Identifiers) to separate data into groups and to allow linkage between graphs to get a web of data.

With the RDF representation of data in form of triples such data can be considered as an oriented graph where nodes are either resources or literals and edges are labeled with predicates. There could be of course more than one edge between two nodes of the graph.

As mentioned above, the W3C standardized query language of RDF data is SPARQL (SPARQL Protocol and RDF Query Language, [25]). A SPARQL query has a SELECT clause and a WHERE clause and may have a FILTER clause to filter the results according to some conditions. In the SELECT clause, attributes to look for are given as variables and these variables are used as substitutes of either subjects, predicates or objects in the triples to look for in the WHERE clause.

```
@prefix fsts: <http://www.fsts.ma/studies#> .
fsts:MachL a fsts:Course .
fsts:MachL fsts:coursename "Machine Learning".
```

**(a)** N3 Format

```
<rdf:RDF xmlns:rdf= "http://www.w3.org/1999/02/22-rdf-syntax-ns#"
            xmlns:fsts="http://www.fsts.ma/studies#">
<rdf:Description rdf:about=”http://www.fsts.ma/studies#MachL”>
 <fsts:coursename>
  Machine Learning</fsts:coursename>
</rdf:Description>
</rdf:RDF>
```

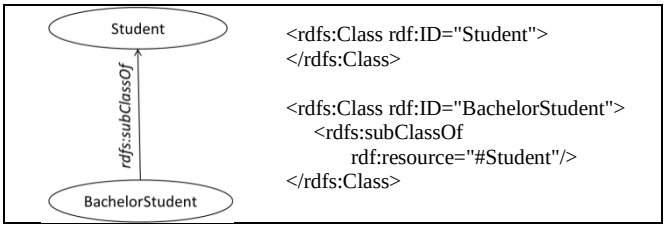
**(b)** XML Format

Fig 1. Example of an RDF Triple.

### B. Schema Languages RDFS and OWL

The RDF schema language RDFS [10]) is the meta-language for RDF data. Statements in RDFS are also RDF triples. RDFS allows RDF resources to be grouped into classes, and allows the declaration of subclasses, properties, subproperties and domains and ranges for properties. An example is given in Fig. 2 where "BachelorStudent" is declared as a subclass of the class "Student".

Built on top of RDFS, OWL (Web Ontology Language [18]) extends RDFS by adding concepts of classes and properties equivalence, resources equality, symmetric properties, disjoint properties and cardinalities.




**(a)** Graph representation **(b)** XML format

Fig 2. Class Hierarchy in RDFS.

OWL uses "ObjectPropertyDomain" and "DataPropertyDomain" to specify the domains of an object property and a data property. It also offers other inference constructs such as "owl:sameAs", "owl:inverseOf" and "owl:TransitiveProperty". Such OWL constructs have the advantage to induce inheritance between classes and similarity between properties and therefore allow reasoning over data through inference.

### C. RDF Triplestores

Over the two past decades several systems for the storage and the processing of RDF data have been developed. Those systems called triplestores can be classified into several categories according to the aspects considered for data management [1]. The criteria we are giving in the following section take into account the category of the triplestore chosen for handling RDF data.

RDF management systems can be broadly classified as being relational or non-relational, native or non-native, centralized or distributed and memory or disc based, as well as Map-Reduce based or not relying on Map-Reduce for the case of big RDF data.

Relational RDF stores are solutions that exploit relational database systems to store RDF data. However, the dynamicity of the RDF data is generally not guaranteed by these triplestores. Object relational stores on the other hand provide the link between classic relational databases and object databases. Non-relational RDF stores are those stores that do not rely on relational database systems for handling RDF data. Native triplestores are those systems designed solely for the purpose of handling RDF data. Some of them are disk-based stores (e.g., 4Store [17]) and others are main-memory-based stores (e.g., Cliopatra [35]). NoSQL triplestores are those RDF solutions that use column, document, Key-value or graph

NoSQL databases for handling RDF data. Among NoSQL triplestores we have CumulusRDF that is based on Cassandra ([21]) and SHARD ([28]).

RDF triplestores are also categorized as either centralized or distributed stores. Although centralized triplestores ensure efficient and scalable RDF query processing in a centralized way, they show limitations in storing and processing large amount of data.

RDF management systems can further categorized in cloud based triplestores (e.g., 4Store [17], Amada [4], [11]), mobile solutions designed for mobile devices (e.g., RDF on the Go [22]), and P2P solutions (e.g., Rya [26], Atlas [20], Statustore [33], RAPID+ [27]). Another category of RDF management systems consists of Big Data triplestores that either use Hadoop Map-Reduce (e.g., SHARD [28], HadoopRDF [19], RAPID+ [27], PigSPARQL [30]) or other frameworks such as Spark framework (e.g., S2RDF [31], PRoST [12]).

To be noticed is that a triplestore may belong to one or more of the given categories. The comparison and evaluation criteria given in following sections also considers the categorization of triplestores. Fig. 3 summarizes the list criteria and the classes they belong to.

### D. Related Work

As already mentioned, this is the first times a research paper addresses the topic of evaluation and comparison criteria for RDF management systems. Many works mainly dealt with the comparison of some triplestores only with respect to either the amount of RDF data they can store, the loading times of such data or the execution times of SPARQL queries on these data. This is done for example for the comparison of some Big Data and some NoSQL RDF in [6].

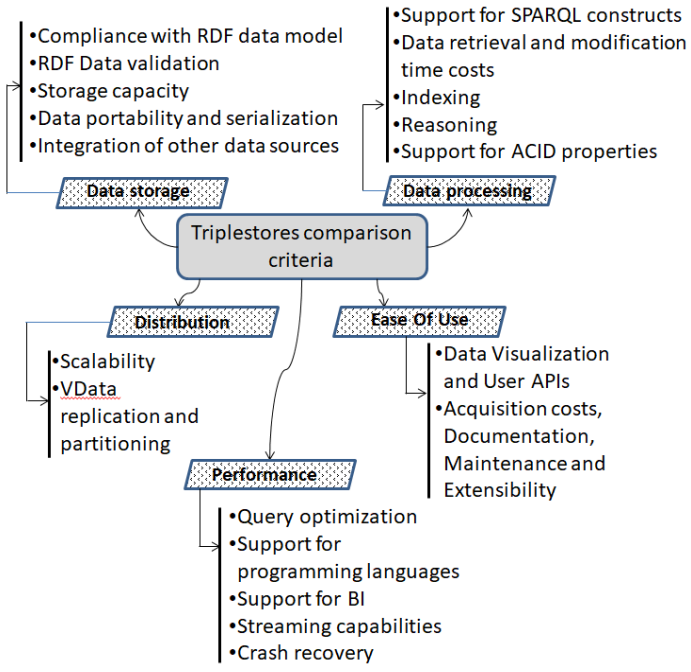



Fig 3. Comparison Criteria with Associated Categories.

Also such type of comparison has also been done in the context of the specific application domain of smart city services, RDF data loading times and query response times were compared in [7] principally for some NoSQL and relational triplestores using data benchmarks related to smart city services.

## III. Criteria Related to RDF Data Storage

In this section we list some important criteria dealing with the capabilities of triplestores to handle RDF data storage. Such criteria involve the respect of RDF data model, RDF data validation, storage capacity, Data portability and serialization and integration of other data sources.

### A. Compliance with RDF Data Model

RDF storage solutions have to preserve the flexibility and dynamicity of RDF data. The "-Subject, Object, Predicate" data model and the graph structure of RDF data is beneficial for querying the semantic information and also for adding new predicates without the need to change the schema. It also allows partitioning of the data for the efficient storage and processing of the queries.

### B. RDF Data Validation

For triplestores it is also necessary that they provide the possibility for users to validate their RDF data against the constraints and the structures they provide in associated RDFS/OWL schemas. Through validation, not only data conformity with such schemas will be guaranteed but also data exchange and integration will be facilitated.

### C. Storage Capacity

The storage capacity for RDF data management systems refers to the possible amount of RDF triples such systems can store and handle. AllegroGraph can handle RDF datasets with more than 1 trillion RDF triples. The Stardog triplestore can handle up to 50 billion triples [31], and GraphDB and Virtuoso triplestores cab handle up to 15 billion [34]. Such information are naturally of great importance for RDF users because of its crucial role in choosing the best suited triplestore for managing their RDF data.

### D. Data Portability and Serialization

Data portability would give the opportunity for users to exchange information and content between the services. This requires representation portability mechanisms to be implemented in triplestores. Among such mechanisms, at least export functionalities of RDF data into portable formats such as XML or Json formats are of great importance. In this way, exported RDF data will be machine-understandable and extensible. Furthermore, switching from a triplestore to another one can be easily done.

### E. Integration of Other Data Sources

Integration functionalities expected from a triplestore concern above all adding new RDF graphs into the triplestore as well as merging graphs. Also adding non RDF data source is of great importance to allow interoperability with other database systems that are not RDF based. Many existing transformation techniques of other non RDF data sources such

as UML, relational and XML already exist and can be incorporated into triplestores to realize such interoperability.

## IV. RDF Data Processing Criteria

### A. Support for SPARQL Constructs

Without SPARQL support by a triplestore such a triplestore will of course be useless. Triplestores should offer SPARQL querying to extract the desired information in an efficient way while providing support for all constructs of SPARQL 1.1. It is an important requirement to efficiently process queries, especially interactively. Also querying with the use of SPARQL should be possible also for massive amounts of data.

### B. Data Retrieval and Modification Time Costs

When considering triplestores, we consider the data, its size and how it is processed. The first thing to consider is how long the triplestore needs to load the data.

Another point to consider is the storage and the retrieval time of the data. Generally, native triplestores are more efficient than existing relational database based triplestores because of the difficulty they face when trying to map the graph based models to SQL.

### C. Indexing

The main objective of data indexing is to sort data in order to make its querying easier and faster. Indexing plays an important role especially when managing large amount of data to increase the performance for a large-scale analysis. Indeed, though indexing involves some space overhead, it lets focus only on the portions of data involved by the analysis so that loading these data can be faster and memory space and execution time will be reduced.

The major problem faced by RDF data stores, is how they can build an index data structure over RDF triples. Because of the performance problems related to loading RDF files, or creating suitable indexes, an RDF triplestore must also provide a memory efficient data representation that leaves enough space for the operation of SPARQL querying algorithms.

With regards to indexing, both automatic indexing through the system and the possibility for users to set indexes on specific resources or literal values of triples are of great importance. The former solution will let users not care about indexation and the latter will give them the possibility to index items dependently of their needs.

Triplestores that are relying on relational database management systems have naturally profited from indexing techniques these systems offer.

### D. Reasoning

Reasoning allows inferring logical consequences and checking the consistency of a database. It allows a better interpretation and processing of the information for the users.

As mentioned above, RDFS and OWL offer constructs (e.g., “rdfs:subClassOf”, “rdfs:subPropertyOf”, “owl:sameAs”, “owl:inverseOf”, “owl:TransitiveProperty”) for modeling the relations between RDF classes or properties to better structure RDF data in order to avoid problems related for example to redundancies, updates or deletion. However, structuring of information using such constructs will have no sense if the system does not have algorithms for an automatic reasoning that can infer, with the use of such constructs, the hidden information which is implicitly deducible from RDFS/OWL schemas. Concerning AllegroGraph, it allows RDFS reasoning with its built-in reasoned as well as temporal reasoning.

### E. Support for ACID Properties

The well-known properties of atomicity, consistency, isolation and durability are of course of great importance for transactions handling [15]. RDF systems that use relational database management systems to store RDF triples have profited from implementations of these properties in these systems. However there is still a lack for support of such properties in non-relational triplestores. Users should therefore be aware of supported properties to ensure that operations of transactions are performed in the right sequence to avoid problems related to inconsistencies, to incomplete executions of such operations or to conflicting operations.

## V. Performance Criteria

### A. Query Optimization

The query optimizer as a component of triplestores, attempts to find the best way to execute a given query efficiently. It simplifies the query and removes redundant computation. In [5] a methodology using the BGPs and OPTIONALs query optimization techniques for the queries with a mix of UNION and FILTER clauses is proposed.

In term of query optimization, relational based RDF triplestores offer better solutions due to the efforts done on making relational query processing efficient over the last three decades.

### B. Support for Programming Languages

It is also to consider if the triplestore serves most modern programming languages (e.g., Java, C++, C#, Python). Within the associated programming APIs RDF Formats and SPARQL query languages should also be supported.

AllegroGraph, for example, offers a Java and Python APIs that implement most of the Sesame and Jena interfaces to access RDF data. It also provides the possibility to Lisp programmers to interact with its RDF repositories.

### C. Support for BI

Nowadays, we deal with a huge amount of data and businesses are aware that analyzing and processing those data can generate new opportunities and improvements of the processes. Business intelligence (BI) tools are therefore to be supported by triplestores in order to provide analytical functionalities for users to analyze their data and extract useful information from these data.

### D. Streaming Capabilities

Real time processing of data is becoming of high importance, due to the increased sources of real time data (e.g. weather sensors, social networks, IoT tools).

The ability for a triplestore to provide support for streaming will have a crucial role in applications. To this end, the

triplestore should support SPARQL querying to be done dynamically over the streams with results given as a continuous streams.

### E. Crash Recovery

A important requirement that is be considered, when considering comparison criteria, is the robustness of triplestores toward the system components failure while processing RDF data and the ability to restore the accidently loosed, deleted or corrupted data.

We consider here, as an example, HadoopRDF ([19]) which provides an architecture that stores triples on HDFS. It replicates the triples on multiple machines and decomposes a user query into partial queries with an independent evaluation of these queries without any communication overhead between the partitions.

## VI. Criteria for Distributed Triplestores

### A. Data Replication and Partitioning

Due to the quick increase of the scale of RDF data, various distributed storage systems have been developed. For such systems partitioning and replication capabilities while handling RDF data is necessary to distribute both data and processing among RDF nodes. For the distribution of data, partitioning techniques should be efficient enough to achieve a reasonable query processing performance together with efficient data transfers between the nodes.

There are two types of data partitioning in existing RDF stores. The first type is the static graph partitioning, which creates partitions with a minimum of edges. The second type is the workload aware partitioning, which faces however the complex problem of choosing the right decisions regarding space and workload [2].

On the other hand, replication refers to the storage of the same data in several different locations. Of course, such replication requires the availability of synchronization mechanisms between the data sources to guarantee consistency between the replicated data. To this end, good strategies are to be provided by triplestores to select the RDF data to be replicated, to control the storage availability and to handle data changes related to updates, insertion or deletion. For example, the distributed triplestore DREAM [16] does not partition data over nodes but simply replicates the whole data in every node, which necessitate updating the same data each time changes occur.

### B. Scalability

A distributed triplestore should have the ability to scale either vertically with the possibility to add data resources to the nodes, or horizontally with the possibility to add more nodes to the system. This is a relevant property for handling large amounts of data that are gathered from various sources as well as for integrating data from classical databases (e.g., XML, relational or file systems).

Because of the graph nature of RDF data, good strategies are needed to achieve both arts of scalability in order to achieve efficient SPARQL search, delete or update queries. Indeed, such queries may involve complex joins of subgraphs and therefore an extra time complexity. The development of Big Data technologies and frameworks (e.g., Hadoop, Map Reduce and Spark) has also favored the development of various scalable triplestores based on such technologies (e.g., SHARD [28], HadoopRDF [19], PRoST [12] and CliqueSquare [14]).

## VII. Ease of Use Criteria

### A. Data Visualization and User APIs

With APIs (Application Programming Interfaces) we mainly mean those APIs that make it easier for triplestores users to interact with their data easily to query their RDF data and to have their data presented in a user friendly way. The list should also include APIs for programming languages or for the use existing RDF/SPARQL programming packages such as the use of Jena.

Visualization of RDF in several ways has also to be taken into account for the understanding of different RDF data structures. Principally, a triplestore, because of the nature of RDF data, should support RDF data presentation in form of graphs.

With regards to APIs, relational databases based triplestores have largely profited of existing APIs developed for relational database systems.

### B. Acquisition costs, Documentation, Maintenance and Extensibility

Two other points to consider are the product costs and the learning costs that are associated of a triplestore and its implementation.

Also the development conditions of a triplestore are also to be considered, together with its documentation, maintenance, accessibility and performance.

As stated, it is important to check how long a triplestore is used, and to also get an overview of possible updates, releases development and dedicated extensibility mechanisms. This will provide an idea about the triplestore, if it is an individual initiative, an active or a non-active project, if it is dependent to a third party application and if it is an open source system.

It is also important to consider, if the store is brightly used, in which domains it is used and how long it is being in use. These factors play an important role in the decision regarding the adoption of such a triplestore or not.

## VIII. Case Study: AllegroGraph

It is absolutely evident that an evaluation of triplestore should be done in the context of its comparison with other stores belonging to its category using associated established criteria. However for specific applications, the triplestore could also be compared with stores not belonging to its category and in this case such comparison needs to only be conducted with respect to some specific criteria pertaining to the specific use in applications. Both types of comparison will lead to further research papers and constitute one of our future perspectives.

However, to illustrate the application of the established list of criteria, we discuss in this section the case of AllegroGraph

triplestore with the NoSQL triplestore XX and with the Big Data triplestores HadoopRDF. As mentioned, a thorough comparison of AllegroGraph with other triplestores from Graph stores and other types of stores using the aforementioned criteria will be the subject of another research paper.

AllegroGraph is an efficient RDF native graph database that uses disk storage, which allows it to scale up to billions of triplets. It was developed to meet RDF standards and It has been continuously further improved since its appearance in 2004. It also offers interfaces for many programming languages such as Java, Python, Ruby, C#, and Scala.

Inference is also supported by AllegroGraph under two angles. On one hand, AllegroGraph offers the so-called "dynamic RDFS++ reasoned" that implements a set of RDFS inference rules and also OWL2resoner. The first reasoner generates inferred triples during inference execution without saving inferred triples. However, the OWL2 reasoner adds generated triples to the considered triples database.

AllegroGraph also has components for the analysis of social network and geospatial data. It also supports visual generation of SPARQL queries as well as visualization of graphs using Gruff. A free, developer and enterprise versions of Allegrograph with storage capacities of respectively 5, 50 and 50+ million triples are provided for users.

In comparison with other Graph oriented triplestores and even to other kind of RDF stores, AllegroGraph fulfills by far many of the criteria mentioned above. We can say that many of such RDF management systems are still at their infancy phase since they are still limited to RDF storage and SPARQL processing functionalities.

For example, HadoopRDF is a Big Data triplestore [19] that uses the Hadoop file system for the distributed storage of RDF data in a cluster of nodes and Map Reduce framework for SPARQL query processing. In comparison of HadoopRDF with AllegroGraph, HadoopRDF also shows a high failure tolerance and reliability. Indeed, Hadoop based triplestores can be easily implemented on clusters of so called commodity computers and the cluster can continue functioning after node failure. Therefore HadoopRDF can also handle very large amounts of RDF data. With regards to RDF querying, processing of SPARQL queries is done in HadoopRDF efficiently since it partitions the RDF data not in a single file but in a set of small files and Map Reduce jobs are simply run on small portions that are of concern [19].

Apart from RDF data modeling compliance, storage and querying, HadoopRDF has not been further developed since its appearance and show strong limitations with respect to the other criteria already listed in comparison with AllegroGraph. However, because of the Hadoop architecture of HadoopRDF, HadoopRDF can also be easily extended and further yields other research perspectives. Indeed, this fact will let HadoopRDF benefice from the analytical technologies and APIs already developed within the framework of Hadoop.

## IX. Conclusion

We have established a list of criteria for the comparison and evaluation of RDF triplestores. To achieve this task, we provided a methodology relying on the identification of expected key characteristics for triplestores. This is done by categorizing the criteria according to: - RDF data storage (e.g., Compliance with RDF data model, RDF Data validation, Storage capacity, Data portability and serialization, Integration of other data sources), - RDF data processing (e.g., support for SPARQL constructs, data retrieval and modification times, indexing, reasoning, support for ACID properties), - performance (e.g., query optimization, support for programming languages, support for BI, streaming capabilities, crash recovery), - distribution (e.g., data replication, scalability), - and ease of use (e.g., user APIs, visualization, acquisition costs, documentation, maintenance and extensibility). As an illustration of the criteria list, we considered the case of AllegroGraph triplestore and showed that AllegroGraph fulfills many of these criteria.

The criteria will play an important role in supporting users to make accurate decisions for the adoption of the appropriate triplestore that best suit their objectives and will help in identifying the strength and weaknesses of existing triplestores.

This research work is as far as we know the first work that addresses comparison and evaluation criteria for triplestores. Because of the increasing use of RDF in many application domains, the established list of comparison and evaluation criteria will surely pave the way for more research works that deal with further improvements of the functioning of existing triplestores or with the development of new ones.

## References

[1] K. Alaoui. "A Categorization of RDF Triplestores," Proc. Smart City Applications, SCA-2019, October 2–4, 2019, Casablanca, Morocco, ACM, ISBN 978-1-4503-6289-4/19/10, DOI 10.1145/3368756.3369047, 2019.

[2] A. Al-Ghezi and L. Wiese, "Adaptive workload-based partitioning and replication for RDF graphs" Database and Expert Systems Applications, 2018.

[3] S. Anand and A. Verma, "Development of Ontology for Smart Hospital and Implementation using UML and RDF," IJCSI Int. J. of Computer Science Issues, Vol. 7, Issue 5, 2010.

[4] A. Aranda-Andujar, F. Bugiou, J. Camacho-Rodriguez, D. Colazzo, F. Goasdoué, Z. Kaoudi, and I. Manolescu, "Amada: Web data repositories in the Amazon cloud," in Proc. 21st Int. Conf. on Information and knowledge Management, CIKM"12, maui, 29 Octpber-02 November 2012, ACM, pp. 2749-2751, 2012.

[5] M. Atre, "Algorithms and analysis for the SPARQL constructs," arXiv:1805.08037v3 [cs.DB] 23 May 2018.

[6] M. Banane and A. Belangour, "An Evaluation and Comparative study of massive RDF Data management approaches based on Big Data Technologies," Int. J. of Emerging Trends in Engineering Research, vol. 7, no. 7, July 2019.

[7] P. Bellini and P. Nesi, "Performance assessment of RDF graph databases for smart city services," J. Vis. Lang. Comput. 2018, 45, 24–38.

[8] M. Bermudez-Edo, T. Elsaleh, P. Barnaghi, and K. Taylor, "IoT-Lite: A Lightweight Semantic Model for the Internet of Things," in 2016 International IEEE Conferences on Ubiquitous Intelligence Computing, Advanced and Trusted Computing, Scalable Computing and Communications, Cloud and Big Data Computing, Internet of People, and Smart World Congress, July 2016, pp. 90–97.

[9] T. Berners-Lee, J. Hendler, and Ora Lassila. “The Semantic Web,” Scientific American, pages 29–37, May 2001.

[10] D. Brickley and R.V. Guha, "RDF Schema 1.1, W3C Recommendation," https://www.w3.org/TR/rdf-schema/, 2014.

[11] F. Bugiou, F. Goasdoué, Z. Kaoudi, and I.Manolescu, “RDF data management in the Amazon cloud,” in ICDT/EDBT Workshops 2012.

[12] M. Cossu, M. Färber, and G. Lausen, "PRoST: Distributed Execution of SPARQL Queries Using Mixed Partitioning Strategies," in Proc. of the 21st International Conference on Extending Database Technology (EDBT), March 26-29, 2018.

[13] J. V. F. Dombeu, and R. Kwuimi, “Semantic data storage in information systems,” African J. Of Information Systems, 2018.

[14] F. Goasdoué, Z. Kaoudi, I. Manolescu, J.A. Quiané-Ruiz, and S. Zampetakis, “CliqueSquare: Flat plans for massively parallel RDF queries,” in Proc. IEEE 31st International Conference on Data Engineering, 2015, pp. 771–782.

[15] T. Haerder, A. Reuter, "Principles of Transaction-oriented Database Recovery.," ACM Comput. Surv. 15, Nr. 4, 1983, pp. 287–317, doi:10.1145/289.291

[16] M. Hammoud, D.A, Rabbou, and R. Nouri, “DREAM: Distributed RDF Engine with Adaptive Query Planner and Minimal Communication,” VLDB Endowment, 2015.

[17] S. Harris, N. Lamb, and N. Shadbolt, "4store: The design and implementation of a clustered RDF store." In Proc. Scalable SemanticWeb Knowledge Base Systems - SSWS2009. pp. 94–109, 2009.

[18] P. Hitzler, M. Krötzsch, B. Parsia, P. F. Patel-Schneider, and S. Rudolph, "OWL 2 Web Ontology Language Primer," https://www.w3.org/TR/owl-primer/, 2012.

[19] M. Husain, J. McGlothlin, M.M. Masud, L. Khan, and B.M. Thuraisingham, “Heuristics-Based query processing for large RDF graphs using cloud computing,” IEEE Trans. on Knowl. and Data Eng., 2011.

[20] Z. Kaoudi, M. Koubarakis, K. Kyzirakos, I. Miliaraki, M. Magiridou, and A. Papadakis-Pesaresi, "Atlas: Storing, updating and querying RDF(s) data on top of DHTS," J. of Web Semantics 8 (4), pp. 271–277, 2010.

[21] G. Ladwig and A. Harth, "CumulusRDF: linked data management on nested key-value stores,” SSWS 30, 2011.

[22] D. Le-Phuoc, J. X. Parreira, V. Reynolds, and M. Hauswirth, "RDF on the go: an RDF storage and query processor for mobile devices," In ISWC Posters&Demos. 2010.

[23] G. Mai, K. Janowicz, B. Yan, and S. Scheider, "Deeply integrating linked data with geographic information systems," Transactions in GIS, 230 (3), pp. 579–600, 2019, doi: 10.1111/tgis.12538.

[24] F. Manola, E. Miller, and B. McBride. “RDF 1.1 Primer," http://www.w3.org/TR/rdf-primer/, 2014.

[25] E. Prud’hommeaux and Andy Seaborne. SPARQL Query Language for RDF, W3C Recommendation. http://www.w3.org/TR/rdf-sparqlquery/, 2008.

[26] R. Punnoose, A. Crainiceanu, and D. Rapp, “Rya: A scalable RDF triple store for the clouds,” in 1st Interna=onal Workshop on Cloud Intelligence (in conjunction with VLDB), 2012.

[27] P. Ravindra, H. Kim, and K. Anyanwu, “An intermediate algebra for optimizing RDF graph patern matching on MapReduce”, in ESWC 2011.

[28] K. Rohloff and R. E. Schantz, "High-performance, massively scalable distributed systems using the MapReduce software framework: the SHARD triple-store," In Programming Support Innovations for Emerging Distributed Applications, pp.1-5, October 17-21, Reno, Nevada, 2010.

[29] S. Sankar, A. Sayed, and J. A. Bani-younis, “A schematic analysis on selective-RDF database stores," Int. J. of Computer Applications 86(11), pp. 21-28, January 2014..

[30] A. Schätzle, M. Przyjaciel-Zablocki, and G. Lausen, ”PigSPARQL: mapping SPARQL to Pig Latinn”, in SWIM 2011.

[31] A. Schätzle, M. Przyjaciel-Zablocki, S. Skilevic, and G. Lausen, “S2RDF: RDF querying with SPARQL on Spark,” arXiv Prepr., pp. 804–815, 2015.

[32] M. Schmidt, O. Görlitz, P. Haase, G. Ladwig, A. Schwarte, and T. Tran, ” FedBench: A benchmark suite for federated semantic data query processing," In Aroyo, L., Welty, C., Alani, H., Taylor, J., Bernstein, A., Kagal, L., Noy, N., Blomqvist, E. (eds.) Proceedings of the 10th International Semantic Web Conference (ISWC 2011), Bonn, Germany, October 23-27, 2011, Part I. pp. 585–600. Springer 2011.

[33] R. Stein and V. Zacharias, “RDF on cloud number nine,” in 4th Workshop on New Forms of Reasoning for the Seman=c Web: Scalable and Dynamic, AWS SimpleDB, May 2010.

[34] W3C, https://www.w3.org/wiki/LargeTripleStores.

[35] J. Wielemaker, W. Beek, M. Hildebrand, and J. van Ossenbruggen, "ClioPatria: A SWI-Prolog infrastructure for the semantic web," Semantic Web 7(5), pp. 529-541, 2016.