ISSN 2278-3091
Volume 8, No.3, May - June 2019
International Journal of Advanced Trends in Computer Science and Engineering
Available Online at http://www.warse.org/IJATCSE/static/pdf/file/ijatcse62832019.pdf
https://doi.org/10.30534/ijatcse/2019/62832019

RDFMongo: A MongoDB Distributed and Scalable RDF management system based on Meta-model

Mouad Banane[1], Abdessamad Belangour[2]
[1]Hassan II University, Morocco, mouad.banane-etu@etu.univh2c.ma
[2]Hassan II University, Morocco, belangour@gmail.com

Abstract
In recent years, the growing volume of RDF data requires scalable, efficient, and more robust systems. To meet this need we find NoSQL databases which are distributed, scalable and powerful systems for large data management. In this paper we present an RDF data store scalable and efficient for large RDF data management based on MongoDB, which is a distributed and document-oriented NoSQL database, MongoDB is the most used NoSQL database in the world [1]. We propose a storage schema that stores RDF triples (subject, predicate, object) in JSON documents to ensure an RDF triple index structure based on JSON-LD. For querying data we transform SPARQL queries to MongoDB language queries through the use of meta-models. The results of the experiments obtained indicate that our approach works well compared to a large volume of RDF data, and show that our system outperforms existing distributed RDF data stores based on NoSQL.

Key words : RDF, MongoDB, SPARQL, Big Data, Semantic Web.

1. INTRODUCTION

In the last decades, the volume of web data stored and processed is increasing day after day with a remarkable speed. The management and processing of this large amount of web data is considered a challenge that exceeds the capabilities of traditional data management systems. RDF[2] is a data format developed and proposed by the World Wide Web Consortium (W3C). It is intended to describe the data in a formal way, especially the metadata which is the concept of the Semantic Web.

Many research efforts have been devoted to the development of distributed and scalable RDF data management systems. On the other hand, we find the existence of a technology called NoSQL offering distributed, scalable and more robust database management systems dedicated specifically to the management of Big Data phenomenon. Using these NoSQL management systems like MongoDB, we can develop a scalable and more robust system that can handle a large amount of RDF data.

We present in this paper an RDF data store based on MongoDB[3] which is a document-oriented NoSQL database, the architecture of this approach contains two steps the first is the storage of RDF data in MongoDB using the Linked Data format JSON-LD[4]. The second step is to use the two meta-models SPARQL and MongoDB and realize the transformation between these two meta-models for the translation of a SPARQL query into MongoDB. The main contributions of this paper can be summarized as follows:

- Present a MongoDB RDF storage using JSON-LD.
- Present a SPARQL to MongoDB QL query translation based on the meta-model.

The remainder of this paper is structured as follow: Section II exposes some existing related works that propose an RDF management system based on NoSQL. Section III describing the RDF technology, MongoDB system, as well as describing the query language of each one SPARQL and MongoDB query language and present the model driving engineering approach. Section IV presents our main contribution and we show the result of experiments. Finally, in section V we conclude this work and we suggest some future researches directions in this topic.

2. RELATED WORK

There have been several approaches to building a scalable RDF store. Khadilkar et al. [5] describe Jena-HBase a distributed RDF data store based on HBase, and use the Jena framework to query RDF data. Papailiou and al. propose H2RDF [6] it is distributed RDF triple store based on HBase[7], for the index structure of this approach it indexes the following three triple patterns: SPO, POS, OSP (O for object, S for subject and P for predicate) and at the query level H2RDF uses the MapReduce framework. Banane and al [8] present in this work an overview and a study for the management of massive RDF data according to the four models of NoSQL technology oriented graph, document oriented, key/value oriented and column oriented. A solution based on the Cassandra [9] database is CumulusRDF [10] this triplestore RDF has an index structure of four triple patterns and for querying this approach uses Sesame. Based on the Accumulo[11] column-based NoSQL system the triplestore Rya [12] is a scalable RDF triplestore capable of handling a very large volume of data. Rya's index structure is: SPO, POS, OSP to manage this data Rya uses the OpenRDF Sesame[14] framework. in [21] the authors present an approach of transforming complex SPARQL queries into a Hive query language program using the principle of meta-models.

3. PRELIMINARIES

In this section, we present a brief overview of the technologies used in this paper such as RDF, MongoDB, and Model Driven Engineering.

3.1 RDF

RDF the Resource Description Framework is a standard formalism adopted by the W3C for the representation of knowledge on the Web. It provides interoperability between applications that exchange information on the Web and makes it understandable to machines. RDF increases the ease of automatic processing of Web resources. It can be used to annotate documents written in non-structured languages or as an interface for documents written in languages with equivalent semantics. The syntax of RDF is based on XML. XML provides syntax for encoding data while that RDF provides a mechanism describing their meaning. One of the goals of RDF is to make it possible to specify the semantics of data based on XML in a standardized and interoperable way.

An RDF document is a set of triples of the form <resource, property, value> with:

- A resource is an entity accessible on the Internet via a URI, it can be an HTML or XML document, an image, a web page, part of a web page . . .,
- A property defines a binary relation between a resource and a value, thus making it possible to associate semantic information with a resource,
- A value is a resource or a literal value (string of characters).

An RDF declaration specifies the value of a property of a resource. It can be described as a property (resource, value). The elements of these triples can be URIs, literals or variables. This set of triples can be represented in a natural way by a multi-graph oriented label where the elements appearing as resources or values are the vertices and each triple is represented by an arc whose origin is its resource and the destination its value. The following figure 1 shows an example of the RDF graph triples.

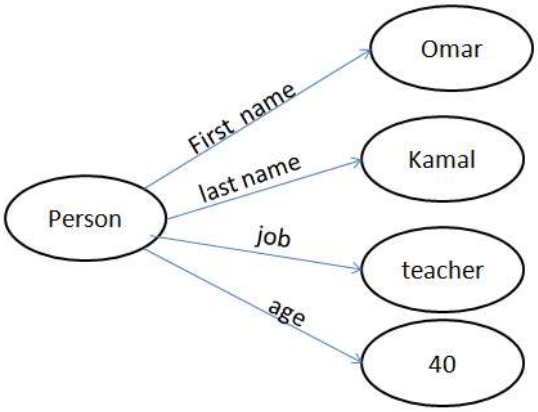

Figure 1: RDF Graph example

3.2 MongoDB

Recently, The rise of big data has followed the evolution of storage and data processing systems with the advent of cloud computing and data science [21,23]. MongoDB is a cross-platform document-oriented database management system. Classed as a NoSQL database, MongoDB avoids the traditional relational database structure in favor of JSON documents with dynamic schemas, making it easier and faster to integrate data into certain types of applications. MongoDB is free and open-source software, MongoDB stores its data in the same format as a JSON document. To be more exact, it's the binary version of the JSON called BSON. That is, a kind of giant dictionary of keys and values. These data can then be exploited by javascript, directly integrated into MongoDB, but can also be exploited by other languages. A JSON document is simply a set of keys and values whose notation is as follows:

```
{
"first_name": "Omar",
"last_name": "Kamal",
"job":"teacher",
"age": 40
}
```

In this example, *first_name* is the key, *Omar* is the value.

Table 1: SQL terminology and concepts and the corresponding MongoDB terminology

SQL terminology	MongoDB terminology
database	database
table	collection
row	document , BSON document
column	field
index	index
table joins	$lookup

3.3 Model driven engineering

Model Driven Engineering (MDE)[14] has emerged as a result of a long history of software engineering. It has considerably contributed to the mastery of the complexity of the distributed IS and the rise in abstraction thanks to the models. In the MDE era, the model has become the unifying concept and is at the center of the IS modeling process. The principle of the MDE is the rise in abstraction thanks to the models and allows the software engineers to evade the technical details of the implementation to focus initially on business models independent of all platforms of executions.

3.3.1 The models

The central concept of the MDA is the notion of a model. Whatever the scientific discipline considered, a model is an abstraction of a system built for a specific purpose. It is said that the model represents the system (programs, computer applications, etc.). A model is an abstraction in that it contains a restricted set of information about the system it represents. It is built for a specific purpose and the information it contains is chosen to be relevant to the user that will be made of the

model. A model is often presented in the form of a combination of diagrams and texts. The text can be written with a modelling language or with a natural language.

In order to make a model usable (productive), it is necessary to specify the language in which it is expressed. In addition, this language must be clearly defined so that the models are handled by the machines. We use for this a meta-model. A meta-model is a model that defines the expression language of a model that is the modeling language [15]. The concept of a meta-model makes it possible to define the characteristics common to a set of models. A meta-model represents a formal specification of an abstraction (abstract syntax), generally consensual and normative of a modeling language.

A model is linked to its meta-model by a relation of conformity. A model is said to conform to a meta-model if all elements of the model are defined by the meta-model. This notion of conformance is essential to model engineering, but it is not new: a text conforms to a grammar, a JAVA program conforms to the Java language, and an XML document conforms to its DTD / XML Schema.

3.3.2 Meta-meta-model

In the same way that it is necessary to have a meta-model to interpret a model, to be able to interpret a meta-model it is necessary to have a description of the language in which it is written: a model for the meta-models. It is natural that this particular model is designated by the term meta-meta-model. By its position in the hierarchy of use, the choice of the meta-meta-model is very important, because it will depend on all the meta-models and models defined thereafter.

In order to avoid the problem of defining meta-meta-models (and thus avoid having to define a meta-meta-meta-model), the idea generally adopted is to design the meta-meta-models. so that they are self-descriptive, that is, self-defining. The MDA approach defines the MOF as the only meta-meta-model for the definition of different meta-models.

Figure 2 presents the different relationships that exist between models, meta-models, and meta-meta-models. We distinguish in this figure 2 the three levels of modelling M1, M2 and M3 respectively corresponding to the model, meta-model and meta-meta-model. The M0 level is real world ie represents the system for example, a JAVA code of an application, the level M1 is the model of this system for example, a UML model of this application, this model UML must be compliant with the UML standard ie. the M2 level and all the meta-models have a single meta-meta-model which the MOF is the last level M3 since it describes itself ie. MOF is the meta-model of MOF.

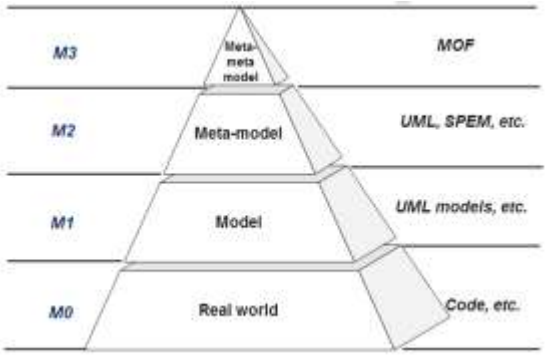

Figure 2: Relationships between models, meta-models, and meta-meta-models

The central concept of the MDA is the notion of a model. Whatever the scientific discipline considered, a model is an abstraction of a system built for a specific purpose. It is said that the model represents the system (programs, computer applications, etc.). A model is an abstraction in that it contains a restricted set of information about the system it represents. It is built for a specific purpose and the information it contains is chosen to be relevant to the user that will be made of the model. A model is often presented in the form of a combination of diagrams and texts. The text can be written with a modelling language or with a natural language.

4. SYSTEM ARCHITECTURE

In this section, we describe the architecture of our system, Figure 3 presents an overview of this architecture that is based on MongoDB: the NoSQL[22] and document-oriented data management system, the choice of MongoDB is for several reasons, first since 2015 , according to db-engines [1], MongoDB is the first in the ranking of the most popular NoSQL database management systems. Then the majority of approaches that offer NOSQL-based RDF data management use column-oriented NoSQL systems such as HBase.

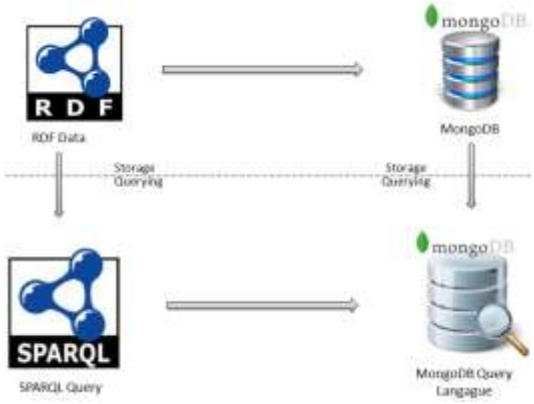

Figure 3: RDF Graph example

4.1 Storage Approach

Storing the RDF triples in a document-oriented database is a very difficult operation, since the triple RDF contains 3 elements and the JSON format is of a key/value pair structure.

To store the RDF data in the MongoDB database, you need

to convert the triple RDF T that contains the elements (Subject S, Predicate P, Object O), for this reason, we transform the RDF data into JSON-LD format through JSON-LD RDF API[16]. JSON-LD JavaScript Object Notation for Linked Data is a method for encoding linked data using JSON. The goal is to provide a simple way for developers to turn existing data into JSON to JSON-LD. This allows you to serialize data in the same way as with traditional JSON. JSON-LD is a recommendation of the World Wide Web Consortium and, therefore, is considered a standard.

4.2 Querying Approach

For querying RDF data, we will use a SPARQL query translator in MongoDB queries. For this we have two choices the first is the use of xR2RML as in [17], this approach consists of the following steps illustrated in Figure 4. First the transformation of the SPARQL query into an abstract query through the xR2RML mapping, then our abstract SPARQL query will be translated into an abstract MongoDB query that will be passed through a set of optimization techniques in order to finally have our MongoDB query.

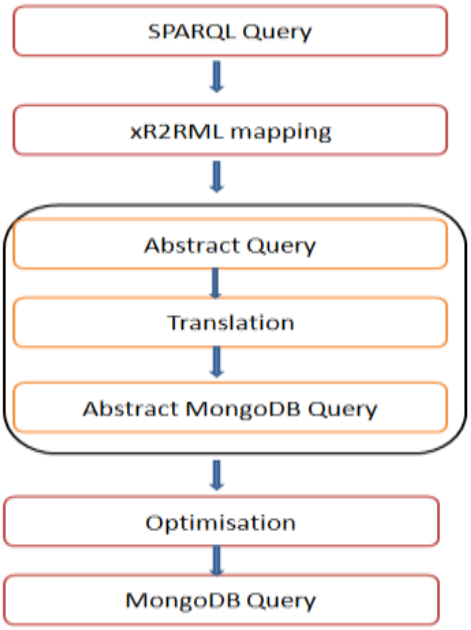

Figure 4: SPARQL to MongoDB mapping using xR2RML

The second technique of this translation is based on the meta-models approach, firstly we realize a SPARQL meta-model and a MongoDB meta-model, then we propose the transformation between these two meta-models, to realize this transformation we used the Atlas transformation language (ATL)[18]. The meta-model SPARQL language is illustrated in Figure 5. In the following, we describe the components of a SPARQL query necessary for the realization of this meta-model.

4.2.1 SPARQL Meta-model

An Ask, Select, Construct and Describe request are SPARQL queries, and the Select query contains the Select clause and the Where clause, or the Select clause contains only one or more Variable as long as the Where clause consists of a *GraphPattern* that can be either an *OptionalGraphPattern*, a *FilterPattern*, a *UnionGraphPattern* or a *TripleSameSubject* that is composed of a subject, predicate, and an object. The following figure 5 illustrates our proposed SPARQL meta-model.

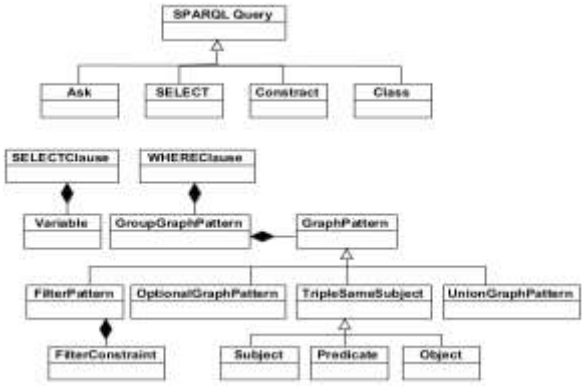

Figure 5: Proposed SPARQL meta-model

4.2.2 MongoDB Meta-model

A MongoDB query can be an insertion query: Insert, modify query Update, Delete query or a selection query: find, this find query contains two elements: query and projection, and each of these two elements is composed of one or more operators, so the projection also contains one or more fields. The figure 6 shows the proposed meta-model for a MongoDB language find query.

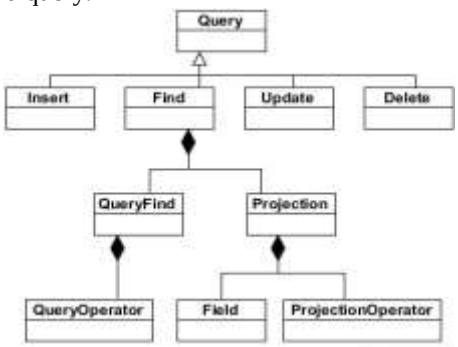

Figure 6: Proposed MongoDB meta-model

The figure 7 illustrates our approach in the levels model of model driven engineering, our system that transforms the SPARQL queries to MongoDB queries takes models using SPARQL and MongoDB models, these SPARQL and MongoDB query models conform respectively to the meta-models of SPARQL and MongoDB and these two meta-models are also conform to the MOF meta-model.

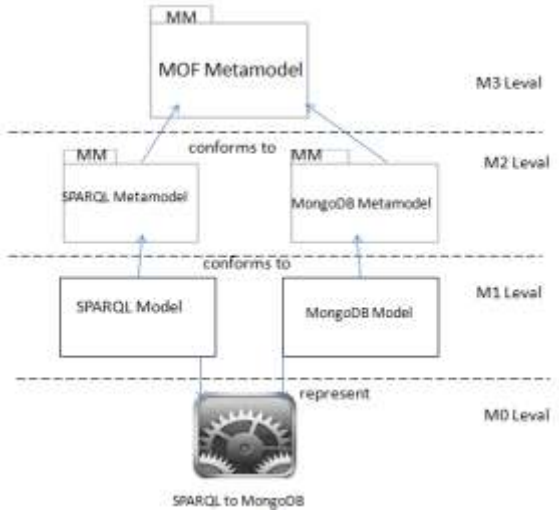

Figure 7: SPARQL to MongoDB in MDE architecture

4.3 Transformation

Given a source model in the language is SPARQL, and a target model in the language is MongoDB, it is in this step to develop a mapping of the concepts of SPARQL to those of MongoDB (eg a SELECT clause corresponds to a find() clause of MongoDB). Therefore, the techniques of meta-modeling presented above are used to establish a comprehensive and generic basis of rules.

The transformation rules are established between the source meta-model and the target meta-model, that is to say between the set of concepts of the source model and that of the target model. The transformation process takes as input a model conforming to the source meta-model and outputs another model conforming to the target meta-model.

To define a model transformation, we can use a non-formal language, an action language to represent the transformation algorithm, or a well-defined model mapping language. Conscious of the need for a well defined and standardized language for the expression of the rules of model transformations, to realize this transformation we can use specific languages like QVT[19] and ATL (ATLAS Transformation Language). Object Management Group (OMG) offers QVT for the standardization of the transformation process. This standard requires that model transformations be precisely defined in terms of the relationship between a source meta-model and a target meta-model and that both meta-models are all MOF [9] compliant. In our case, we used the ATL language.

Once specified and expressed, the rules require a runtime engine to run. This engine takes as input the source model and meta-model, the target meta-model, as well as the transformation model (the transformation rules written in the ATL transformation language, based on the correspondences between the two source and target meta-models.) and its meta-model (representing the ATL transformation language grammar) and outputs the target model. The transformation engine can proceed either by interpretation or by compilation.

A SPARQL search query can contain more than the clause SELECT the following clauses: WHERE, FILITER, ORDER BY and LIMIT and for MongoDB we find the clauses find, Sort, Limit. Table 1 illustrates the SPARQL language clauses with their corresponding clauses in MongoDB's query language.

Table 2: Corresponding syntax of SPARQL and MongoDB

SPARQL	MongoDB query
SELECT	db.collection.find
WHERE	WHERE
FILTER	FILTER
ORDER BY	Sort
LIMIT	LIMIT

Table 3: Example of queries conversion from SPARQL to MongoDB

Query	SPARQL Syntax	MongoDB Syntax
Retrieve the list of persons	**SELECT * FROM** person	db.person.find()
Get first name and age of all persons.	**SELECT** ?first_name, ? age **FROM** person	db.person.find({ }, { first_name: 1, age: 1, _id: 0 })
Get information from persons with as first name OMAR and they are 45 years old	**SELECT** * **FROM** person **WHERE**?first_name = "OMAR" **AND** ?age = 45	db.person.find({ first_name: "OMAR", age: 45 })
Retrieve information from persons with an age greater strictly than 20 and less than or equal to 45	**SELECT** * **FROM** person **WHERE** ?age > 20 **AND** ?age <= 45	db.person.find({ age: { $gt: 20, $lte: 45 } })

5. EVALUATION

In this section, we describe the configuration and the experimental setup, then we present the performance evaluation of our MongoDB based RDF store, as well as the comparison with other existing approaches that are also based on NoSQL technologies.

5.1 Configuration & Experiments Setup

For the implementation and testing our system is based on MongoDB is specially version 4.0, with Intel 2.4G processor for the hard disk can hold up to 4TB and a memory of 8GB.

5.2 Performance evaluation

For the evaluation of the system, we use the Lehigh University Benchmark (LUBM) [20] which is the most benchmark used to test the performance of RDF data management systems. We use three instances of this benchmark which are LUBM1, LUBM2 and LUBM5. The

following table shows our three datasets with their number of universities, their number of triples and their size.

Table 4: LUBM datasets informations

Dataset	Number of Universities	Number of Triples	Size
LUBM1	1000	138M	11,4 GB
LUBM2	2000	276M	22,77 GB
LUBM5	5000	689M	56,8 GB

5.3 Comparison with other existing systems

We present in Table 5 the results of this experiment using the LUBM benchmark queries on the Jena-HBase, Rya and our MongoDB based systems, the results are presented on the three instances of the LUBM Benchmark.

Table 5: Experiments using LUBM queries on Jena-HBase, Rya, and our MongoDB based system

Dataset	LUBM1			LUBM2			LUBM5		
System / Queries	Jena-HBase	Rya	MongoDB	Jena-HBase	Rya	MongoDB	Jena-HBase	Rya	MongoDB
Q1	61	48	44	228	145	147	414	440	398
Q2	182	5127	120	2322	13783	2000	14565	23611	14705
Q3	220	42	63	2549	55	52	5543	61	74
Q4	508	780	524	2498	825	793	5616	764	729
Q5	619	2946	602	4645	3039	3106	11513	3216	4015
Q6	483	254	237	4367	1518	1580	11161	3538	3389
Q7	479	603	463	2639	607	712	7031	636	605
Q8	767	10026	1225	3006	10384	2763	5939	10851	5549
Q9	635	3403	756	7752	25677	8541	19972	46026	17852
Q10	99	21	34	1053	137	184	2338	139	172
Q11	49	65	42	51	72	50	62	85	59
Q12	63	484	68	79	465	83	124	471	134
Q13	201	115	132	2391	117	128	5175	516	480
Q14	189	217	170	2938	1397	1305	7872	3834	3607

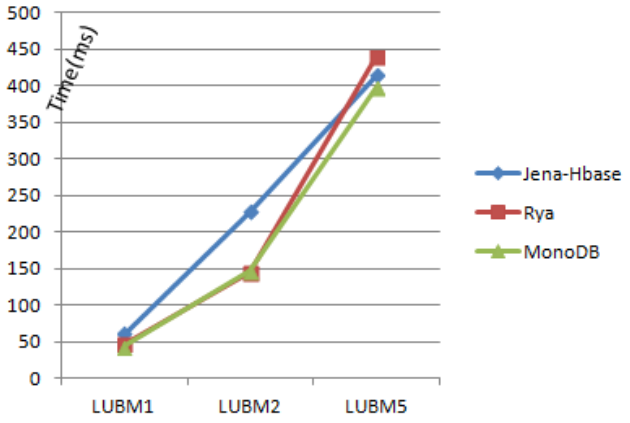

Figure 8: execution time for Q1 LUMB Benchmark

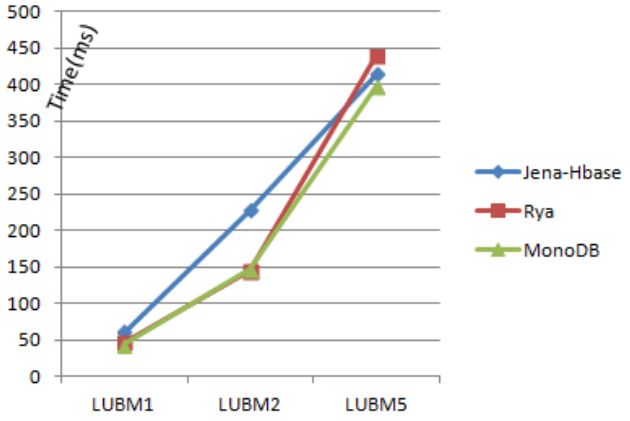

Figure 9: execution time for Q11 LUMB Benchmark

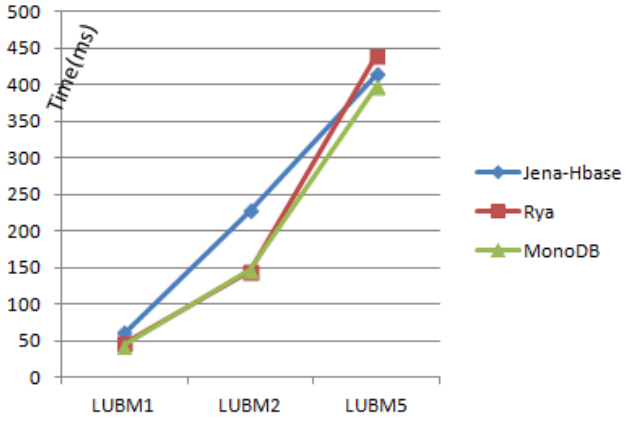

Figure 10: execution time for Q11 LUMB Benchmark

6. CONCLUSION

The growing volume of RDF data requires efficient management of this huge data, we have proposed in this paper a scalable and very powerful RDF store implemented on a document-oriented NoSQL database named MongoDB. The results of the experiments obtained show the effectiveness of our approach compared to existing systems. This approach based primarily on two steps first is for storage that consists of transforming the RDF triples into a JSON document using the JSON-LD format, and for the second ie. Querying is to transform a given SPARQL query into a MongoDB query using the meta-model approach.

REFERENCES

1. **DB-Engines Ranking - popularity ranking of database management systems**. [Online]. https://db-engines.com/en/ranking. [Consulted on: 09-04 -2019].
2. The Resource Description Framework (RDF) and its Vocabulary Description Language RDFS
3. K. Chodorow, **MongoDB: The Definitive Guide: Powerful and Scalable Data Storage** -.
4. **JSON-LD - JSON for Linking Data** . [Online]. Avalable on: https://json-ld.org/. [Consulted on: 28-07-2018]
5. V. Khadilkar, M. Kantarcioglu, B. Thuraisingham, et P. Castagna, **Jena-HBase: A Distributed, Scalable and Efficient RDF Triple Store** , p. 4
6. N. Papailiou, I. Konstantinou, D. Tsoumakos, et N. Koziris, **H2RDF: adaptive query processing on RDF data in the cloud**. , 2012, p. 397. tps://doi.org/10.1145/2187980.2188058
7. **HBase in Action** , Manning Publications. Book
8. M. Banane, A. Belangour, et L. E. Houssine, **Storing RDF Data into Big Data NoSQL Databases**, *in Lecture Notes in Real-Time Intelligent Systems*, 2017, p. 69‑ 78. https://doi.org/10.1007/978-3-319-91337-7_7
9. Learning Apache Cassandra : build an efficient, scalable, fault-tolerant, and highly-available data layer into your application using Cassandra. Book
10. Ladwig et A. Harth, **CumulusRDF: Linked Data Management on Nested Key-Value Stores** , *The 7th International Workshop on Scalable Semantic Web Knowledge Base Systems (SSWS 2011)*. Vol. 30. 2011.
11. Apache Accumulo. [Online]: https://accumulo.apache.org/. [Consulted on: 31-01-2019].
12. R.Punnoose, A. Crainiceanu, et D. Rapp, **Rya: a scalable RDF triple store for the clouds** , *Proceedings of the 1st International Workshop on Cloud Intelligence*. ACM, 2012. https://doi.org/10.1145/2347673.2347677
13. J. Broekstra, A. Kampman, et F. van Harmelen, **Sesame: An Architecture for Storing and Querying RDF Data and Schema Information**, *International semantic web conference. Springer, Berlin, Heidelberg* p. 16.
14. M.Banane, A.Belangour, **New Approach based on Model Driven Engineering for Processing Complex SPARQL Queries on Hive** *International Journal of Advanced Computer Science and Applications(IJACSA)*, 10(4), 2019. https://doi.org/10.14569/IJACSA.2019.0100474
15. About the Meta Object Facility Specification Version 2.4.2 . [Online]. https://www.omg.org/spec/MOF/2.4.2/About-MOF/. [Consulted : 09-11-2018].
16. JSON-LD RDF API. [Online] : https://json-ld.org/spec/latest/json-ld-rdf/.
17. F. Michel, L. Djimenou, C. F. Zucker, et J. Montagnat, **xR2RML: Non-Relational Databases to RDF Mapping Language** , p. 35

18. F. Jouault et I. Kurtev, **Transforming Models with ATL** , in *Satellite Events at the MoDELS 2005 Conference*, vol. 3844, J.-M. Bruel, Éd. Berlin, Heidelberg: Springer Berlin Heidelberg, 2016, p. 128‑138.. https://doi.org/10.1007/11663430_14
19. I. Kurtev, **State of the Art of QVT: A Model Transformation Language Standard** , in *Applications of Graph Transformations with Industrial Relevance*, vol. 5088.
20. Guo, Yuanbo, Zhengxiang Pan, and Jeff Heflin. **LUBM: A benchmark for OWL knowledge base systems**. *Web Semantics: Science, Services and Agents on the World Wide Web* 3.2-3: 158-182. https://doi.org/10.1016/j.websem.2005.06.005
21. Ahamad, D., Akhtar, M., & Hameed, S. A. (2019). **A Review and Analysis of Big Data and MapReduce** International Journal of Advanced Trends in Computer Science and Engineering, 8(1), 2–4. https://doi.org/10.30534/ijatcse/2019/01812019
22. Erraissi Allae, et Abdessamad Belangour. **Data Sources and Ingestion Big Data Layers: Meta-Modeling of Key Concepts and Features**. International Journal of Engineering, s. d., 7.
23. B.Manoj, K.V.K.Sasikanth, M.V.Subbarao Prakash, V. J. (2018). **Analysis of Data Science with the use of Big Data**. International Journal of Advanced Trends in Computer Science and Engineering, 7(6), 5–8. https://doi.org/10.30534/ijatcse/2018/02762018