ISSN 2278-3091
Volume 8, No.4, July – August 2019
International Journal of Advanced Trends in Computer Science and Engineering
Available Online at http://www.warse.org/IJATCSE/static/pdf/file/ijatcse68842019.pdf
https://doi.org/10.30534/ijatcse/2019/68842019

# Querying massive RDF data using Spark

**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**

Semantic web technologies are increasingly used in different domains. The core technology of the Semantic Web is the RDF standard. Today with the growth of RDF data it requires systems capable of handling these large volumes of data and responding to very complex queries at the join level. With the increase in RDF data volumes available, many research efforts have been made to allow for distributed and efficient evaluation of SPARQL queries. In this paper, we propose a new solution based on Apache Spark for massive querying and RDF data. This new system allows the processing of complex SPARQL queries on large volumes of RDF data stored in the Hadoop file system or a NoSQL database management system. Our approach translates the SPARQL query of the user to a Spark script in order to take advantage of the speed of processing and the performance of Spark compared to other existing systems, thanks to an experiment carried out on RDF datasets.

**Key words :** SPARQL, Apache Spark, RDF, MapReduce, Big Data.

## 1. INTRODUCTION

Hadoop is a framework that will allow the processing of massive data on a cluster ranging from one to several hundred machines. Hadoop[9] is written in Java and was created by Doug Cutting and Michael Cafarella in 2005 (after creating the Lucene search engine, Doug was working for Yahoo on his Nutch web crawling project). Hadoop will manage the data distribution at the heart of the cluster machines, their possible failures but also the aggregation of the final processing. The architecture is of the "Share nothing" type: no data is processed by two different nodes even if the data are distributed over several nodes (principle of a primary node and secondary nodes).

Hadoop is composed of four elements:

- Hadoop Common: set of utilities used by other elements,
- Hadoop Distributed File System (HDFS): a distributed file system for persistent data storage.
- Hadoop YARN: a framework for resource management and processing planning,
- Hadoop MapReduce: a distributed processing framework based on YARN.

HDFS is a Java file system used to store structured and unstructured data on a set of servers. It is a distributed, expandable and portable system developed by the creator of Hadoop inspired by the system developed by Google (GoogleFS). Written in Java, it has been designed to store very large volumes of data on a large number of machines equipped with local hard drives. HDFS relies on the native OS file system to present a unified storage system based on a set of heterogeneous disk and file systems.

MapReduce[1] is a parallel processing framework, created by Google for its web search engine. It is a framework that allows the de-composition of an important query into a set of smaller queries that will each produce a subset of the final result: this is the Map function. All the results are processed (aggregation, filter): this is the Reduce function. MapReduce is ideal for batch processing, but it is not iterative by default. A batch type architecture makes it possible to process a set of input data until the source is exhausted. As long as data are available, the processing will continue and we will have a coherent and accessible result only at the end of the processing.

In order to avoid this tunnel effect it is possible to split the input data and this is where the incremental notion is important. It will allow the new data to be taken into account without the need to reprocess all the data already processed. A perfect example of Batch Big Data is Map Reduce in its Hadoop version. The data is first selected by a main and often unique processing. The data is distributed between different nodes in order to be processed.

Once the data processed by the set of nodes a process performs the global operations:

- Sorting,
- Aggregation,

This type of processing has advantages such as simplicity of implementation, but it has drawbacks like:

- Processing time,
- Data arriving during processing are not taken into account.

The outline of the paper is as follows: Section 2 exposes some existing related works in this topic. Section 3 describes Spark. Section 4 presents our main contribution. Section 5 evaluates and analyzes our approach (RDFSpark) with Lehigh University Benchmark. Finally. We conclude in Section 6.

## 2. RELATED WORK

Many research efforts have been devoted to developing a large RDF data management system such as Jena-HBase [2], it is an evolutionary RDF based triplestore based on HBase[8] database storage which a basic management system column-oriented NoSQL data, Jena-HBase uses Jena for querying RDF data, another system also based on HBase is H2RDF [3] which is a fully distributed RDF data store, H2RDF combines the MapReduce Framework with a store distributed data NoSQL, so for storage this system uses HBase and for the processing of these data it uses MapReduce. we also mention that the H2RDF system has two features that allow efficient processing of simple SPARQL queries and also multiple joins on an unlimited number of triple RDF's through joining algorithms that perform joins based on the selectivity of the query for reduce the processing. Other researchers use for example the NoSQL database Cassandra[10] as: Ladwig and al. Providing with CumulusRDF [4] the nested key-values RDF data store and distributed as the underlying storage component for a linked data server, CumulusRDF provides functionality to process linked data through HTTP lookups. Or the authors of CumulusRDF have developed two index schemes for RDF to support both linked data retrieval and basic triple searching based on the model. The work [14] presents an overview of the RDF data management systems based on the NoSQL databases according to the different models: column-oriented, document-oriented, key-value oriented and graph-oriented. Cudre-Mauroux and al present [5] a comparison of NoSQL stores for RDF data processing. This work describes only four NoSQL stores that are: 4store, Jena-HBase, Hive-HBase, and CumulusRDF, this work also compares their key features through running standard RDF benchmark tests on a cloud infrastructure using two deployment modes. : On a single machine and distributed mode. A recent comparison [11] of RDF stores based on NoSQL, and recently in [12] the authors propose RDF data storage technique, for efficient processing of SPARQL queries using two distributed computing engines Spark and Drill [13]. Semantic Web applications generate huge amounts of data every day. RDF databases or triplestores are not scalable, on the other hand the big data systems guarantee us these scalability options, high data availability, and the high performance of the system, from where helped to integrate the Semantic Web technologies in a Big Data environment[20], the first work was a preliminary study [14] of the use of NoSQL database management systems as a new RDF database, then we proposed a survey [11] approaches that propose the use of Big Data technologies such as NoSQL or even Hadoop Distributed File System (HDFS) for managing large volumes of triple RDFs. After having carried out these state-of-the-art works, our work is oriented towards the proposition of our system which has an added value compared to the existing solutions, based on our two studies presented above, we proposed SPARQL2Hive [11] a new solution for processing complex SPARQL queries on Apache Hive based on Model-Driven Engineering, this system manages RDF data stored either in HDFS or a NoSQL database, the user writes its queries in SPARQL which is going to be in accordance with our SPARQL metamodel, then SPARQL2Hive transforms this request into a Hive program, thanks to the transformation and mapping between the two metamodels SPARQL and Hive. In recent years, MongoDB, the NoSQL database management system is the number 1 system and the most used in the world, and it is not suitable to manage semantic web data, based on this result we started a new proposal RDFMongo [19] which a complete solution for the storage and processing of semantic data based on MongoDB, the RDF triples are transformed into JSON document is stored in MongoDB, and for the querying of these data we have implemented a mapping algorithm that transforms SPARQL queries into MongoDB Query Language queries.

## 3. SPARK

Apache Spark[6] is an open source framework for Big Data processing built on the basis of Hadoop MapReduce to perform sophisticated analysis and designed for speed and ease of use. It was originally developed by UC Berkeley University in 2009 and passed Open Source as an Apache project in 2010.

**Spark ecosystem**

Next to Spark's main APIs, the ecosystem contains additional libraries that allow you to work in the field of Big Data analysis and machine learning. Among these libraries are Streaming, SQL, MLlib and GraphX.

### 3.1 Spark Streaming

It can be used for real-time flow processing. It relies on a micro-batch processing mode and uses an abstraction to effectively reuse data in a large family of applications called RDD for Resilient Distributed Dataset. RDDs are fault tolerant and offer parallel data structures that allow users to:

1. Persistently store intermediate data in memory.
2. Control their partitioning to optimize the location of the data.
3. Manipulate the data, using a large set of operators.

### 3.2 Spark SQL

It exposes Spark datasets, via a JDBC type API, and executes SQL type queries, using traditional BI and visualization tools. Spark SQL can extract, transform and load data in different formats and expose them for ad hoc queries.

### 3.3 Spark MLib

MLlib is a machine learning library, which contains all the classic learning algorithms and algorithms, such as classification, regression, clustering, collaborative filtering, dimension reduction, in addition to the underlying optimization primitives.

### 3.4 Spark GraphX

GraphX is the new API, still in alpha version, for graph processing and graph parallelization. GraphX extends Spark's RDDs by introducing the Resilient Distributed Dataset Graph, an oriented multi-graph with properties attached to nodes and arcs. To support these processes, GraphX exposes a set of basic operators, such as Sub-graph, JoinVertices, and AggregateMessages. In addition, GraphX includes an ever-growing collection of algorithms to simplify graph analysis tasks.

### The benefit of Spark

Spark has several advantages over other technologies like Hadoop and Storm. First, Spark offers a comprehensive and unified Framework to address Big Data processing needs for various data-sets, different in nature (text, graph) as well as source type (in Batch or real-time mode). Then, Spark allows applications on Hadoop [16,18] Clusters to be executed up to 100 times faster in memory, 10 times faster on the disk. It allows you to quickly write applications in Java, Scala or Python and includes a game of over 80 high-level operators. In addition, it can be used interactively to query data from a Shell command window. Finally, in addition to the "map" and "reduce" functions, Spark supports SQL queries and data flow and offers machine learning and graph-oriented processing functions. Developers can use these capabilities separately or by combining them into a complex processing chain.

### Micro-batch processing

With the arrival of Spark for Streaming, the concept of micro-Batchs has attracted the attention of developers. This consists of splitting into smaller processes, real-time streams to be processed, at intervals between 500 ms and 5000 ms. Spark implements the concept of micro-Batchs in its operation. On the other hand, Spark should not be considered as a real-time stream processing engine. This is indeed the biggest difference between Spark and Storm. Storm, thanks to its Trident API, is able to support Batch Standards and micro-Batch processing.

## 4. SYSTEM ARCHITECTURE

To handle the growing size of RDF datasets that arrives up to billions of triple RDFs, we find the MapReduce Framework that Google developed for the parallel processing of very large datasets. Indeed writing programs with MapReduce is technically difficult and takes a lot of time. We use Spark because Spark allows applications on Hadoop Clusters to be executed up to 100 times faster in memory, 10 times faster on the disk, in addition to the "map" and "reduce" functions, Spark supports SQL queries and data flow and offers machine learning and graph-oriented processing functions. On the other hand, Apache Spark is an open source Big Data processing framework built to perform sophisticated and very fast analysis as MapReduce. We now present our approach which is a framework for translating SPARQL queries into Spark Jobs as an intermediate layer between SPARQL and Hadoop MapReduce. This abstraction layer ensures interoperability and compatibility to future Hadoop changes since it leaves our approach independent of the Hadoop version, see Figure 1.

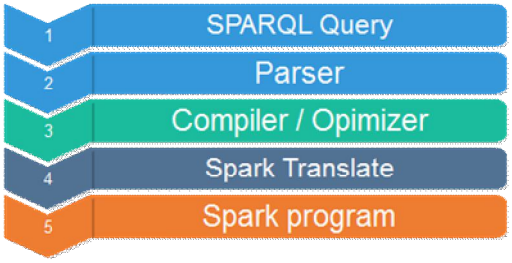


**Figure 1:** Modular Translation Process

## 5. EXPERIMENTS

In this section, we show that our system allows the evaluation of SPARQL queries on several billion triple RDF distributed over several nodes. We also compare our solution to other state-of-the-art solutions. We thus demonstrate the performances obtained by allowing the participants to reproduce the results themselves through different scenarios directly competing with several state-of-the-art solutions.
In order to demonstrate how our system can be a scalable RDF system, we describe an experiment. We discuss selected systems; in addition, we describe the experiment conditions, present the results, and discuss them. In this experiment, we wanted to evaluate the scalability of our solution for this reason, we used the Lehigh University Benchmark (LUBM) [7], and we used four instances of this Benchmark LUBM1, LUBM2, LUBM3, LUBM4. (Table 1).

**Table 1:** A Datasets summary

| Dataset | Number of Triples | Size |
|---|---|---|
| LUBM1 | 1,316,993 | 0.11GB |
| LUBM2 | 6,890,933 | 0.583GB |
| LUBM3 | 133.000.000 | 22 GB |
| LUBM4 | 1,334,000,000 | 213 GB |

We compare the performance of our system:"RDFSpark" with three other systems: Jena-HBase [2], a horizontally expandable triple RDF store that uses HBase for storage, CumulusRDF, the Cassandra-based RDF triplestore proposed by Ladwig et al [4], which uses an indexing scheme based on the NoSQL model, and H2RDF [3] which is also based on HBase. The code Jena-HBase and H2RDF are open-source and we used the default configurations for the code.

**Table 2:** Query execution time on LUBM1

| | Q1 | Q2 | Q3 | Q4 | Q5 | Q6 | Q7 | Q8 | Q9 |
|---|---|---|---|---|---|---|---|---|---|
| **Jena-HBase** | 324 | 6549 | 325 | 480 | 339 | 576 | 236 | 754 | 9766 |
| **H2RDF** | 430 | 7857 | 560 | 543 | 587 | 550 | 321 | 506 | 11276 |
| **CumulusRDF** | 308 | 5801 | 311 | 452 | 305 | 496 | 376 | 467 | 8744 |
| **RDFSpark** | 165 | 2765 | 176 | 265 | 187 | 298 | 188 | 345 | 3870 |

**Table 3:** Query execution time on LUBM2

| | Q1 | Q2 | Q3 | Q4 | Q5 | Q6 | Q7 | Q8 | Q9 |
|---|---|---|---|---|---|---|---|---|---|
| **Jena-HBase** | 205 | 2495 | 175 | 207 | 180 | 298 | 185 | 322 | 3874 |
| **H2RDF** | 831 | 7857 | 481 | 543 | 507 | 563 | 377 | 509 | 7800 |
| **CumulusRDF** | 308 | 5801 | 311 | 851 | 305 | 492 | 341 | 752 | 8732 |
| **RDFSpark** | 414 | 3458 | 219 | 407 | 289 | 479 | 250 | 447 | 7332 |

**Table 4:** Query execution time on LUBM3

| | Q1 | Q2 | Q3 | Q4 | Q5 | Q6 | Q7 | Q8 | Q9 |
|---|---|---|---|---|---|---|---|---|---|
| **Jena-HBase** | 461 | 4860 | 358 | 451 | 239 | 378 | 406 | 521 | 5008 |
| **H2RDF** | 478 | 6750 | 568 | 594 | 587 | 554 | 377 | 508 | 6907 |
| **CumulusRDF** | 342 | 7801 | 311 | 454 | 305 | 496 | 376 | 467 | 8744 |
| **RDFSpark** | 365 | 2764 | 176 | 238 | 187 | 298 | 188 | 302 | 3821 |

**Table 5:** Query execution time on LUBM4

| | Q1 | Q2 | Q3 | Q4 | Q5 | Q6 | Q7 | Q8 | Q9 |
|---|---|---|---|---|---|---|---|---|---|
| **Jena-HBase** | 584 | 7857 | 566 | 613 | 675 | 1830 | 304 | 585 | 8902 |
| **H2RDF** | 409 | 6004 | 890 | 543 | 587 | 1512 | 379 | 506 | 8254 |
| **CumulusRDF** | 324 | 6549 | 325 | 480 | 339 | 776 | 236 | 754 | 9766 |
| **RDFSpark** | 433 | 7857 | 566 | 543 | 583 | 523 | 359 | 568 | 5670 |

**Table 6:** Loading time LUBM queries

| | LUBM1 | LUBM2 | LUBM3 | LUBM4 |
|---|---|---|---|---|
| Q1 | 251 | 6536 | 325 | 480 |
| Q2 | 430 | 7857 | 566 | 543 |
| Q3 | 308 | 5801 | 382 | 627 |
| Q4 | 267 | 2765 | 478 | 699 |
| Q5 | 252 | 6601 | 461 | 489 |
| Q6 | 423 | 7859 | 504 | 543 |
| Q7 | 307 | 5801 | 311 | 651 |
| Q8 | 165 | 2771 | 476 | 735 |
| Q9 | 263 | 6632 | 329 | 1280 |

For the Load Time: the evaluation shows that the load time in-creases linearly with the size of the data, as expected. The RDFSpark loading process uses Spark Streaming provided by Apache Spark to speed up the acquisition. The acquisition process is parallelized between the servers, using as much as possible all the servers. Figures 2, 3, 4, and 5 illustrate the results of the execution of the nine LUBM Benchmark queries, according to the four datasets used LUBM1, LUBM2, LUBM3, and LUBM4.

Our distributed RDF storage solution using Apache Spark to evaluate SPARQL queries and store data through Hadoop File System (HDFS) infrastructures. This system relies on a SPARQL query translator to a sequence of instructions executable by Spark by adopting evaluation strategies. The results of this adopted strategy are presented in the following figures 2, 3, 4 and 5.

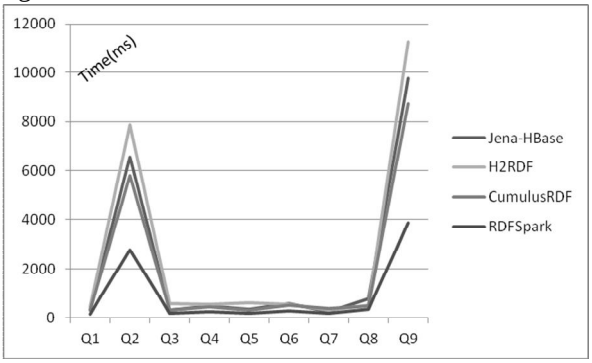


**Figure 2:** LUBM1 queries execution time

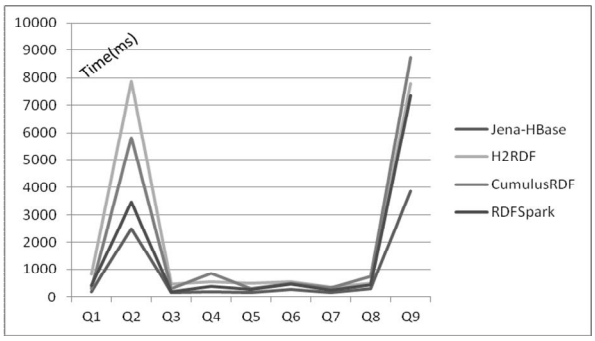


**Figure 3:** LUBM2 queries execution time

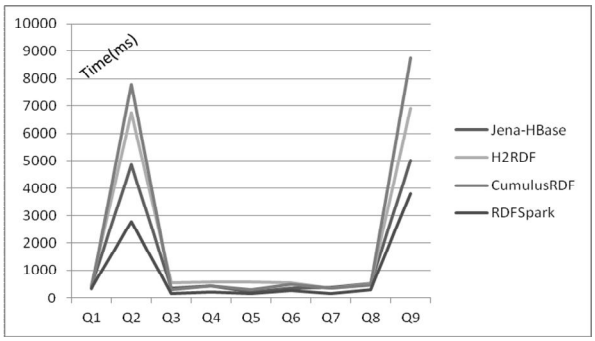


**Figure 4:** LUBM3 queries execution time

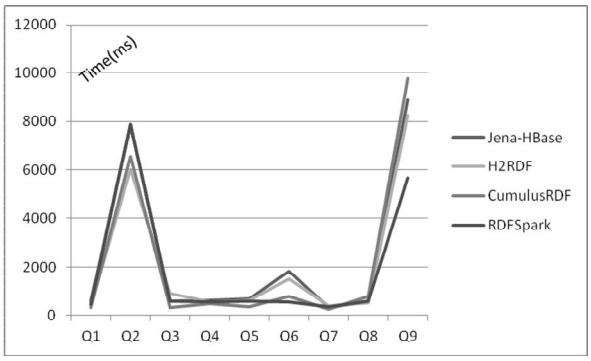


**Figure 5:** LUBM4 queries execution time

From these results, our experiments firstly confirm the efficiency of RDFSpark, and for the processing time of a SPARQL query T with respect to the size of the data S, we notice that there is an almost linear scalability of this factor T / S. Thanks to the advantages of Spark, these evaluations show that RDFSpark is an efficient system for handling complex SPARQL queries against the processing of these queries by using MapReduce or a query language of NoSQL databases. Another interesting point is the step of optimizing RDFSpark, the series of optimization techniques reduces both the amount of data to be processed and the processing time of the corresponding SPARQL request.

## 6.CONCLUSION

In this paper, we have seen the features of Apache Spark in data processing and analysis. Spark positions itself against traditional MapReduce implementations like Apache Hadoop or we have proposed RDFSpark, a new approach for scalable execution of Apache Spark-based SPARQL queries to applications based on extracting information from very large RDF data volumes. To do this, we designed and implemented a translation of SPARQL queries to Spark program. The resulting Spark program is translated into a MapReduce job sequence and run in parallel on a Hadoop cluster. It is also possible to combine the Spark processing types with Spark SQL, Spark Machine Learning and Spark Streaming, thanks to different Spark integration modes and adapters.

## REFERENCES

1. Al-Hamami, Alaa Hussein, and Ali Adel Flayyih. "**Enhancing Big Data Analysis by using Map-reduce Technique**." *Bulletin of Electrical Engineering and Informatics* 7.1 (2018): 113-116...
2. Khadilkar, V., Kantarcioglu, M., Thuraisingham, B., & Castagna, P. (n.d.). "**Jena-HBase: A Distributed, Scalable and Efficient RDF Triple Store**", 1–4.
3. Papailiou, N., Konstantinou, I., Tsoumakos, D., & Koziris, N. (2012). "**H2RDF : Adaptive Query Processing on RDF Data**" *in the. Proceedings of the 21st International Conference Companion on World Wide Web, 397–400.* http://doi.org/10.1145/2187980.2188058.
4. Günter Ladwig and Andreas Harth. "**CumulusRDF: linked data management on nested key-value stores**". *In The 7th International Workshop on Scalable Semantic Web Knowledge Base Systems (SSWS 2011)*, page 30, 2011.
5. Cudr, P., Haque, A., Harth, A., Keppmann, F. L., Miranker, D. P., Sequeda, J. F., & Wylot, M. (n.d.). "**NoSQL Databases for RDF : An Empirical Evaluation**", 310–325.
6. Karau, H., & Warren, R. (n.d.). "**High performance Spark : best practices for scaling and optimizing Apache Spark**".2017
7. Y. Guo, Z. Pan and J. Heflin, "**LUBM: a benchmark for OWL knowledge base systems**," *Web Semantics: Science, Services and Agents on the World Wide Web*, vol 3, Oct. 2005, pp.158-182. https://doi.org/10.1016/j.websem.2005.06.005
8. Dimiduk, N., & Khurana, A. (2013). **HBase in action. Manning**. Book
9. White, T. (Tom E. . (n.d.). **Hadoop : the definitive guide**.
10. Brown, M. (n.d.). "**Learning Apache Cassandra : build an effi-cient, scalable, fault-tolerant, and highly-available data layer into your application using Cassandra**". Book.

11. Banane, M. and Belangour, A., 2018, July. "**A Survey on RDF Data Store Based on NoSQL Systems for the Semantic Web Applications**". *In International Conference on Advanced Intelligent Systems for Sustainable Development* (pp. 444-451). Springer, Cham. https://doi.org/10.1007/978-3-030-11928-7_40
12. Hassan, M., & Bansal, S. K. (2018). "**RDF data storage tech-niques for efficient SPARQL query processing using distributed computation engines**". *Proceedings - 2018 IEEE 19th International Conference on Information Reuse and Integration for Data Science*, IRI 2018, 323–330. http://doi.org/10.1109/IRI.2018.00056
13. Apache Drill - Schema-free SQL for Hadoop, NoSQL and Cloud Storage. [Online]. Available: https://drill.apache.org/. [Accessed: 05-10-2018].
14. Banane M., Belangour A., El Houssine L. (2019)" **Storing RDF Data into Big Data NoSQL Databases**". In: *Mizera-Pietraszko J., Pichappan P., Mohamed L. (eds) Lecture Notes in Real-Time Intelligent Systems. RTIS* 2017. Advances in Intelligent Systems and Computing, vol 756. Springer, Cham https://doi.org/10.1007/978-3-319-91337-7_7
15. Nair, Lekha R., Sujala D. Shetty, and Siddhant Deepak Shetty. "**Streaming big data analysis for real-time sentiment based targeted advertising**." *International Journal of Electrical and Computer Engineering 7.1* (2017): 402. https://doi.org/10.11591/ijece.v7i1.pp402-407
16. Erraissi, A., & Belangour, A. (2018). **Data sources and ingestion big data layers: meta-modeling of key concepts and features**. *International Journal of Engineering & Technology*, 7(4), 3607-3612. https://doi.org/10.2139/ssrn.3185342
17. Banane, Mouad, and Abdessamad 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, no. 4 (2019). https://doi.org/10.14569/IJACSA.2019.0100474
18. Erraissi Allae, and Abdessamad Belangour. « **Hadoop Storage Big Data Layer: Meta-Modeling of Key Concepts and Features** ». *International Journal of Advanced Trends in Computer Science and Engineering* 8, nº 3 (2019): 646‑53. https://doi.org/10.30534/ijatcse/2019/49832019
19. Mouad Banane, and Abdessamad Belangour. « **RDFMongo: A MongoDB Distributed and Scalable RDF management system based on Meta-model**». *International Journal of Advanced Trends in Computer Science and Engineering* 8, nº 3 (2019): 734 – 741. https://doi.org/10.30534/ijatcse/2019/62832019
20. Mouad Banane, and Abdessamad Belangour. « **An Evaluation and Comparative study of massive RDF Data management approaches based on Big Data Technologies**». *International Journal of Emerging Trends in Engineering Research.* 7, nº 7 (2019): 48 - 53. https://doi.org/10.30534/ijeter/2019/03772019