International Journal of Computing and Digital Systems
ISSN (2210-142X)
Int. J. Com. Dig. Sys. 9, No.2 (Mar-2020)
http://dx.doi.org/10.12785/ijcds/090211

A New System for Massive RDF Data Management Using Big Data Query Languages Pig, Hive, and Spark

Mouad Banane[1] and Abdessamad Belangour[1]

[1] *Hassan II University, Casablanca, Morocco*

Received 29 Sep. 2019, Revised 21 Dec. 2019, Accepted 30 Dec. 2019, Published 01 Mar. 2020

Abstract: The era of big data has emerged. The volume of generated data has never been greater. Massive quantities of data are stored on a huge number of servers that are inter-connected and share their storage space. Computation methods have been developed to perform computation operations directly on these machines, previously used mainly for storage. Tools such as Hive, Pig, and Spark provide the means for data query and analysis but are not suitable for Semantic Data. For this kind of data, a specialized tool called SPARQL is dedicated to query semantic data represented by the Resource Description Framework or RDF. The aim of our work is to transform a given SPARQL query into a Hive program, a Pig program or a Spark script according to the user's choice. To achieve this goal, we propose a Model-Driven Approach which consists of creating a metamodel for each of these tools, to define a mapping between SPARQL metamodel on one hand and each of the previous Big Data query languages (Pig, Hive, and Spark). The transformation is then performed using Atlas Transformation Language or ATL. We conducted that an experiment on three datasets containing a large volume of distributed RDF data on a powerful server cluster to validate our approach.

Keywords: Semantic Web,RDF, SPARQL, Big Data, Hive, Pig, Spark.

1. Introduction

Storing information from the semantic web implies being able to potentially manage very large volumes of data. Hence the need to opt for a necessarily distributed solution to be able to scale up. The problem encountered at present is that the benchmarks for RDF are intended for centralized architectures and do not take into account certain aspects specific to the resolution of queries on a distributed system. Indeed, the main advantage of distributed storage, which is to more easily allow scaling up, is offset by constraints linked to the algorithm applied to retrieve data from the nodes constituting the distributed system and then aggregate them.

On the other hand, with the advent of digital technology and smart devices, a huge amount of digital data is generated every day. This sharp increase in data, both in size and form, is mainly due to social networks that allow millions of users to share information express and disseminate their ideas and opinions on a topic, and show their attitudes towards content. All of these actions stored on social media generate a massive body of opinion that provides an opportunity for automated data mining and analysis systems to determine Internet user trends. Several researchers have shown a keen interest in using this information to predict human behavior in areas as diverse as medicine, politics, marketing, and so on.

It is in this context that we are witnessing the emergence of a new massive data management system called Hadoop. Hadoop is an open source framework developed in Java, part of the projects of the Apache Foundation. It has been designed to:

- Store very large volumes of data;
- Support data in various formats, structured, semi-structured or unstructured.

Hadoop is based on a set of machines that form a Hadoop cluster. Each machine is called a node. It is the addition of storage and processing capabilities of its nodes that ensures a large storage system and computing power. The storage system is called the Hadoop Distributed File System (HDFS) [20]. The computing power is based on the MapReduce parallel programming paradigm [21]. There is the reason to believe that MapReduce will become the normal mode of data processing in the digital age and therefore that Hadoop will become the default tool for data processing. The problem is that the MapReduce is a very low level language, that is to say, very close to the machine; it implies that the developer knows how to interact with the cluster, which can be very

E-mail: mouad.banane-etu@etu.univh2c.ma, belangour@gmail.com

difficult for a new developer in the world of parallel processing, or business users. One of the ways to simplify MapReduce development and Hadoop, in general, is to provide what is called an abstraction language. An abstraction language is a syntax language that is relatively close to human language and allows you to express business problems in the form of simple queries. The abstraction comes from the fact that when the user expresses his need in the form of a request, this request is transformed below into machine instructions. Thus, the abstraction language is a layer that masks the complexity of expressing problems directly in low-level language as a developer would. The higher the level of abstraction offered by the language, the further away from the machine and the easier it is for users. The Apache foundation currently provides three abstraction languages for MapReduce: Hive, Pig and Cascading. These three languages, designed for a non-developer audience, make it possible to express MapReduce jobs in a programming style similar to that of SQL, familiar to users. Subsequently, these languages transform written requests into MapReduce Jobs that are submitted to the cluster for execution. Overall, Hive offers a higher level of abstraction language that Pig and Pig offer higher level abstraction than Cascading. In this article, we will only study Hive and Pig. Cascading is too low for business analysts only familiar with SQL.

The rest of this document is organized as follows. In section 2 we present the research work, and in the third section, we discuss the technologies used in this work, such as the RDF standard, the Pig, Hive, Spark languages, and model-driven engineering. Section 4 presents the architecture of our system, then we illustrate in Section 5, the results of the experiments. The last section concludes the work and presents our future work.

2. Related Work

Numerous research studies have proposed methods for querying semantic Web data stored in RDF format to meet various needs. A study [2] evaluates approaches to managing large volumes of RDF data based on Big Data technologies. In the RDFSpark[3,4]. Spark is used to execute complex SPARQL queries on massive RDF data. And for querying RDF data stored in an RDFMongo triplestore [5], we can use the query language of MongoDB to handle this RDF data.

The work presented in [6] proposes a query system for Linked Data which uses the equivalence relationships between resources of different databases to propagate the search for results of a query. This approach first queries a known database of Linked Data and then uses the owl: sameAs relationships of its resources to other databases to find other results. This system fully uses the “Linked” nature of Linked Data. Unlike our method, this approach does not require having a list of Linked Data databases to query as long as the starting database is sufficiently linked in Linked Data. It nevertheless encounters latency time problems when querying other databases, while our method is designed to limit these latency times.

The SPARQL 1.1 standard defines the SERVICE keyword making it possible to specify a particular endpoint for parts of the body of a federated SPARQL request, [7,8]. Similar to the previous work, the paper [9] proposes a system for optimizing the sending of federated SPARQL queries by parallelizing the processing of queries to each database. This system divides a request into different sub-requests according to their selectivity and distributes them to the different databases according to a heuristic based on their latency times.

In [10], the authors present the MapReduce programming model and its implementation for massive data processing. In this model, while the map function filters the data, the reduce function aggregates them. MapReduce jobs are divided into two sets of tasks, map tasks and reduce tasks, which are distributed across a set of servers. This allows developers, without any experience with parallel and distributed programming, to easily use such a system. With all these advantages MapReduce is quite complicated hence the emergence of new Big Data data processing tools such as Hive [11], Pig [12], and Spark [13], these tools provide an intermediate layer between User and MapReduce since a Pig program, or Hive is finally transforming into a MapReduce job.

The existing work that deals with the processing of SPARQL queries on Hadoop / Big Data is grouped in [14], this survey discusses and compares these systems by measuring the loading time and the execution time.

PigSparql [15] translates complex SPARQL queries at the level of algebraic presentations such as the syntax tree, and the algebra tree, into a program of the Latin Pig language, this program finally is translated into MapReduce jobs. Note that a SPARQL query is addressed to the algebra part and that the expression of the SPARQL algebra is interpreted as a tree, this expression will be evaluated upwards through an optimizer. The SPARQL query processing time concerns the size of the RDF data proportionally. The work [16] presents a comparative study of SPARQL query management systems in a distributed environment using NoSQL databases management systems such as HBase [17], Cassandra[21,22], and MongoDB.

San et al. [17] present a distributed and scalable RDF triplestore based on the HBase database, the RDF triples are stored in column format, and to manage these data RDF San and al proposes a new MapReduce strategy for SPARQL BGP processing which stands for Basic Graph Pattern, this strategy is suitable for the storage scheme in HBase. To process a typical BGP, this technique uses several MapReduce jobs. In each job, it uses a greedy method to first select the join key and then eliminate several triple patterns. Mammo et al.[18] present a comparative study of two presto and Hive systems to measure the performance of each in the processing of large RDF data.

In RDF databases, resources are described by their links to other resources and their links to literal values. The semantics of RDF databases are therefore contained in these relationships. However, there is a gap between the structured representation that the user perceives and the physical representation in an RDF database. Querying a SimplePARQL query [19] is done via a transparent rewriting of several SPARQL queries through the endpoints of the Linked Data databases. These multiple SPARQL queries, necessary to determine the inaccurate elements of the user-defined SimplePARQL query, have a direct impact on the execution time. With this solution, users build structured queries in an intuitive way and without requiring prior knowledge of the base vocabulary and IRIs (Internationalized Resource Identifier). Among the areas that require powerful tools to handle large volumes of Semantic Web data we find the recommendation systems [31], a recommendation system based entirely on SPARQL named RecSPARQL was introduced in [32]. The proposed tool extends the syntax and semantics of SPARQL to allow flexible and generic collaborative filtering and a recommendation based on RDF graphs. In [33], the authors present an event recommendation system based on Linked Data and the diversity of users. A semantic extension of the SVD+++ model named SemanticSVD+++ is presented in [34] it integrates semantic categories of items in the model.

3. BACKGROUND

We present in this section the different tools and technologies used in this work as: SPARQL, Apache Hive, Apache Pig, Spark and Model Engineering.

A. Semantic Web: RDF & SPARQL

Thanks to the efforts of the World Wild Web Consortium (W3C), the information available on the web can be processed automatically by machines, not by humans. The idea is to make the Web intelligent, where information will no longer be stored but understood by machines to provide users with relevant answers. Several languages have been developed as part of the Semantic Web and most of these languages are based and use XML syntax. OWL [25] and RDF [26] are the most important languages of the Semantic Web, they are based on XML. RDF increases the ease of automatic processing of web resources. The RDF is the first W3C standard for enriching web-based resources with detailed descriptions. Descriptions can be characteristic of resources, such as the author or the content of a website. These descriptions are metadata. Enriching the Web with metadata allows the development of what is called the Semantic Web. RDF is also used to represent semantic graphs corresponding to a specific knowledge modeling.

- RDF is a language developed by the (W3C) to put a semantic layer on the Web. It allows connection of web resources using directed and tagged arc. The structure of RDF documents is complex. An RDF document is a set of <subject, predicate, object> triples. In addition, the predicate (also called property) links the subject (resource) to the object (value). Thus, the subject and the object are nodes of the graph connected by a directed edge of the subject towards the object. Nodes and arcs belong to "resource" types. A resource is identified by a unified resource identifier (URI).
- SPARQL [27] is the standard language for querying semantic graphs. The SPARQL language has gradually become the reference language for querying RDF datasets. SPARQL has become an official recommendation of the (W3C) dedicated to the interrogation of semantic graphs. SPARQL was designed to handle complex query structures.

B. Hive

Hive is an IT infrastructure similar to the Data Warehouse that provides query and aggregation services for very large volumes of data stored on a distributed HDFS file system. Hive provides an SQL-based query language called Hive Query Language (HiveQL), which is used to address queries to data stored on the HDFS. HiveQL also enables advanced users / developers to integrate Map and Reduce features directly to their queries to cover a wider range of data management issues. When you write a query in HiveQL, this query is transformed into a MapReduce job and submitted to the JobTracker for execution by Hive.

C. Pig

Pig is a runtime environment for interactive data flows under Hadoop. It is composed of 2 elements:

- A data flow expression language called the Latin Pig;
- And an interactive environment for executing these data flows;

The language offered by Pig, the Latin Pig, is roughly similar to a Scripting language such as Perl, Python, or Ruby. However, it is more specific than the latter and describes itself better on the term "data flow language". It makes it possible to write queries in the form of sequential flows of source data to obtain "target" data under Hadoop in the manner of an ETL. These streams are then transformed into MapReduce functions which are finally submitted to the jobtracker for execution. To put it simply, Pig is the Hadoop ETL. Programming in Pig Latin amounts to describing as independent but nested streams how data is loaded, transformed, and aggregated using specific Pig instructions called operators. The mastery of these operators is the key to mastering programming in

Latin Pig, especially since they are not numerous in relation to the Hive for example.

D. *Spark*

Before explaining what Spark is, remember that for an algorithm to run on multiple nodes of a Hadoop cluster, it must be parallelizable. Thus, an algorithm is said to be "scalable" if it is parallelizable (and thus can benefit from the scalability of a cluster). Hadoop is an implementation of the MapReduce calculation model. The problem with MapReduce is that it is built on a Direct Acyclic Graph model. In other words, the sequence of MapReduce operations runs in three direct and straightforward sequential phases (Map -> Shuffle -> Reduce), no phase is iterative (or cyclic). The direct acyclic model is not suitable for certain applications, especially those that reuse data across multiple operations, such as most statistical iterative algorithms, for the most part, and interactive data analysis queries. Spark is a response to these limitations; it is a calculation engine that performs distributed processing in memory on a cluster. In other words, it is a distributed in-memory calculation engine. Compared to the MapReduce that works in batch mode, the Spark calculation model works in interactive mode, ie, mounts the data in memory before processing it and is therefore very suitable for Machine Learning processing.

E. *Model Driven Engineering*

Since the beginning of software engineering, the size and complexity of the software developed have been growing faster and faster, while the constraints of development time, quality, maintenance, and evolution are always stronger. In this context, software engineering techniques are constantly evolving to manage the complexity and ensure the quality of the software product. These techniques are grouped under Model-Driven Engineering (MDE)[28]. This term reflects the evolution of the software development process from "contemplative" use to "productive" use of models. Where models were used as elements of design, discussion or documentation, the idea of MDE is to use them as an input to the development process. In practice, this requires formalizing the models to make them usable by the machine and to produce programs for processing models. In the terminology of the MDE, these programs are grouped under the term model transformations. The two originalities of the MDE are, on the one hand, more formal models and, on the other hand, model transformation programs. In order to ensure the quality of the development process, and therefore the quality of the software produced, it is necessary to ensure the quality of the models and the correction of the transformations used.

4. System Architecture

The design of our system contains three parts: the source metamodel part, the target metamodel, and the transformation between these two metamodels. The source metamodel that is unique is the SPARQL metamodel, but for the target metamodel, we are going to create a generic metamodel that acts as a metamodel of Big Data query language, three metamodels of Apache Hive, Apache Pig and Spark. The transformation step makes it possible to take in input the SPARQL request and transform it by using the ATL[29] transformation language into a Hive, Pig program or a Spark script according to the user's choice.

A. *Hive metamodel*

Hive allows you to write queries in a language inspired by SQL and called HiveQL. These queries are transformed into MapReduce jobs. To work, just define a schema that is associated with the data. This schema gives the names and types of the columns and structures the information into tables that can be used by HiveQL. A Hive SELECT query contains the following clauses: WHERE, Having, Group By, and Order By. The figure 1 below shows our Hive metamodel.

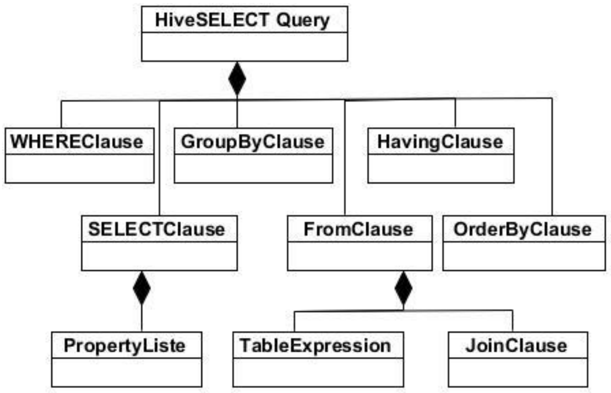


Figure 1. Proposed Hive metamodel

B. *Pig metamodel*

Pig makes it possible to write useful processes on data, without suffering the complexity of Java. The goal is to make Hadoop accessible to non-computer scientists: physicists, statisticians, mathematicians. . . Pig proposes a scripting language called "Pig Latin". This language is called "Data Flow Language". Its instructions describe processes on a data stream. Conceptually, it looks like a Unix tube; each command modifies the flow of data that passes through it. Pig Latin also makes it possible to build much more varied and non-linear treatments. Pig translates Pig Latin programs into MapReduce jobs and integrates results into the flow.

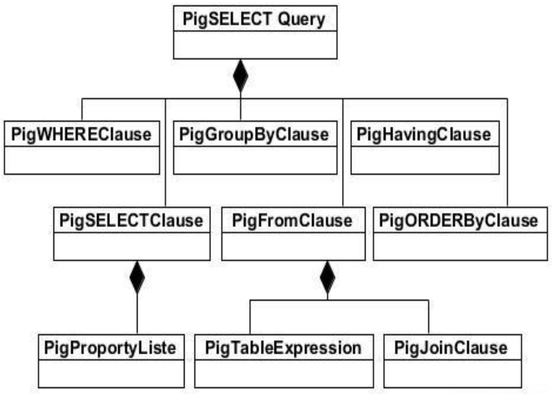


Figure 2. Proposed Pig metamodel

C. Spark metamodel

The main object of Spark is the RDD: Resilient Distributed Dataset. It is a device for processing a collection of data by robust parallel algorithms. A RDD does not really contain data, only a treatment. This abstraction makes it possible to manipulate data distributed on several machines as simply as centralized data. Spark relies on the MapReduce paradigm to offer algebraic data manipulation operations (selection, projection, grouping, etc.) that transform RDDs into other RDDs. An RDD can thus be defined as a sequence of algebraic operations and can be recalculated if necessary. This allows Spark to keep data in memory according to requests and to guarantee their recovery in the event of a failure. Unlike Hadoop and Hive, Spark stores intermediate results in memory and moves them to disk only when necessary. Memory storage allows Spark to avoid congestion due to disk I/O, especially for the intermediate results of a map task. The notion of RDD also allows Spark to define the partitioning of data at the time of their creation.

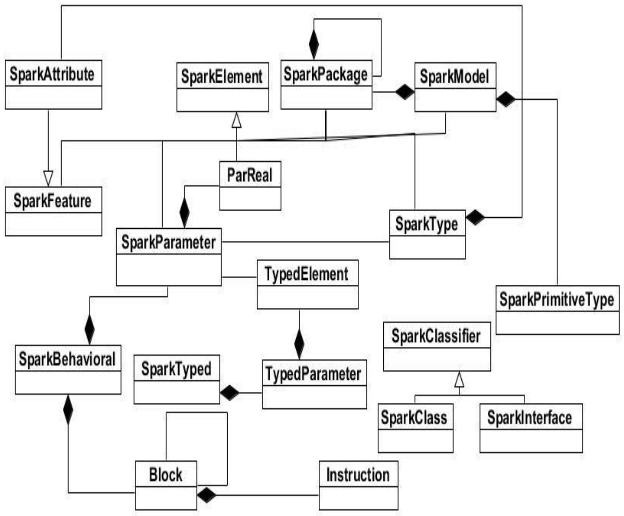


Figure 3. Proposed Spark metamodel

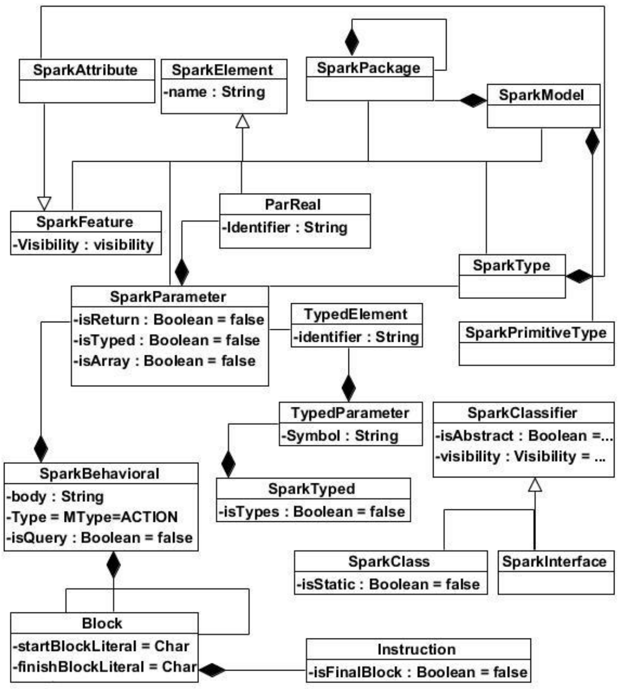


Figure 4. Proposed Spark metamodel with meta-class properties

D. Proposition of Big Data query Language metamodel

To realize a metamodel big data query, we noticed that the queries of all these Big Data querying systems are transformed in the background to the MapReduce Jobs, hence the idea of proposing a generic metamodel which will be the metamodel of MapReduce. Figure 5 presents the proposal of Big Data query language metamodel.

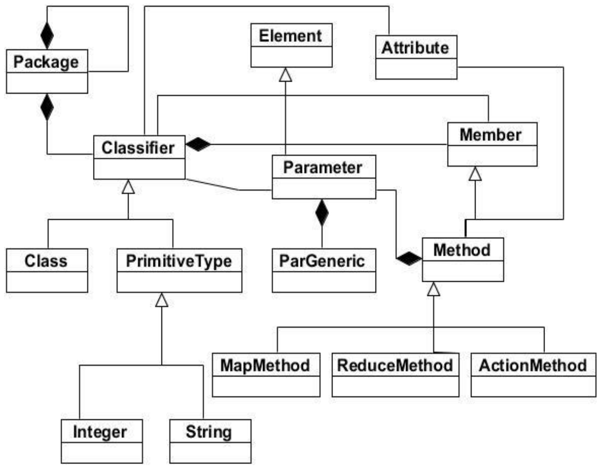


Figure 5. Proposed Big Data query Language metamodel

E. SPARQL metamodel

To build a SPARQL metamodel it is necessary to know its general structure of the requests, first remark is that this structure is similar to the structure of language SQL, secondly we there are three types of requests SPARQL: SELECT, CONSTRUCT, and ASK, we begin by the SELECT query since it is the most used and the most important this query can extract RDF data according to conditions specified in the WHERE clause, so it is a query, second query a constructive query, CONSTRUCT allows to generates a new RDF graph that completes the queried graph. Each SELECT query contains the following clauses: SELECT, FROM, and the condition clause WHERE.

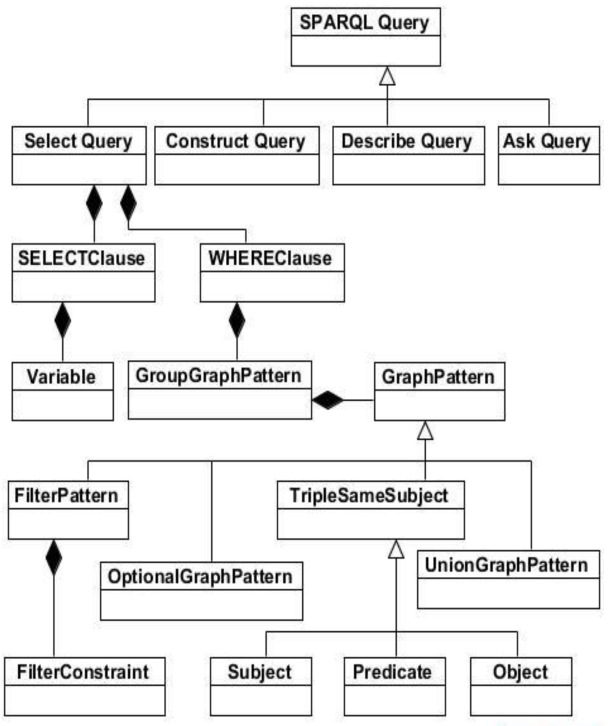
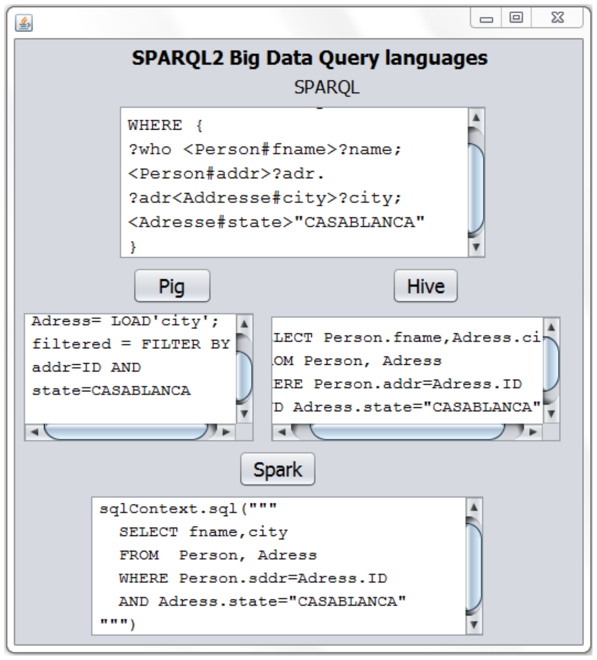


Figure 6. Proposed SPARQL metamodel

The following figure 7 illustrates an example of the execution of our system which transforms a SPARQL query into a program of PigLatin, HiveQL, and a Spark script.

The choice is left to the user between the three languages PigLatin, HiveQL, and Spark. The system can simultaneously generate the translation of the SPARQL query into these three Big Data languages, always according to the choice of the user.



```
WHERE {
?who <Person#fname>?name;
<Person#addr>?adr.
?adr<Addresse#city>?city;
<Adresse#state>"CASABLANCA"
}
```

```
Adress= LOAD'city';
filtered = FILTER BY
addr=ID AND
state=CASABLANCA
```

```
LECT Person.fname,Adress.ci
OM Person, Adress
ERE Person.addr=Adress.ID
D Adress.state="CASABLANCA"
```

```
sqlContext.sql("""
  SELECT fname,city
  FROM  Person, Adress
  WHERE Person.sddr=Adress.ID
  AND Adress.state="CASABLANCA"
""")
```

Figure 7. Example of converting a SPARQL query to Pig, Hive, and Spak.

5. Validation & Experiences

A. SPARAQL2Hive experiences

In order to measure the performance of our approach, we used 3 instances of the LUBM[30] Benchmark. The nine LUBM queries are run on these three datasets of different sizes to better analyze the SPARQL2Hive system. In the first part, we present the configuration and context of our experiences, the version of Hive, and the details of the datasets. Then we will analyze the results obtained. Finally, we will evaluate the impact of the size of the sample database on the quality of the model transformation and we discuss the results of this evaluation which are presented graphically and show the efficiency of the SPARQL2Hive system.

SPARQL2Hive is implemented on the Hadoop 3.xy version and the Hive 3.1.0 version on a machine with a 2.3 GHz Intel Xeon processor, this machine can store up to 4 TB of hard disk storage and RAM storage of 16 GB. LUBM1, LUBM2 and LUBM5 these three datasets used in this experiment, they have the following triplets number: 138 million triples, 275M and 689M and the sizes of these three datasets are: 11.4 GB, 22, 77 GB and 56, 8 GB. The results obtained for the loading time of these three games to give are presented in the table 1:

TABLE I. Loading time for LUBM datasets

Dataset	LUBM1	LUBM2	LUBM5
Loading Time(ms)	1,26	3,05	7,9

Table 2 illustrates the results of running the 14 LUBM queries on the three instances of this Benchmark.

TABLE II. SYSTEM RUNTIME FOR LUBM QUERIES (MS)

Queries	LUBM1	LUBM2	LUBM5
Q1	481	537	752
Q2	429	516	641
Q3	535	583	633
Q4	509	621	627
Q5	743	797	851
Q6	657	720	773
Q7	678	736	794
Q8	179	216	201
Q9	129	130	142
Q10	181	237	252
Q11	121	135	150
Q12	83	103	126
Q13	376	405	451
Q14	325	361	404

We compare our SPARQL to Hive system with Jena by always using the three datasets LUBM1, LUBM2, LUBM5, generally on the majority of the queries; SPARQL2hive is more powerful than Jena at the runtime of LUBM Benchmark queries. Figures 8, 9, 10, 11, 12, 13, 14,15, 16, 17,18,19,20 and 21 show the results of this comparison for LUBM queries.

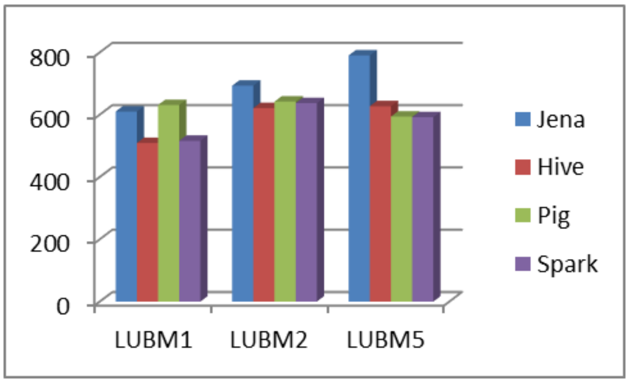


Figure 8. LUBM Q1 runtime(ms)

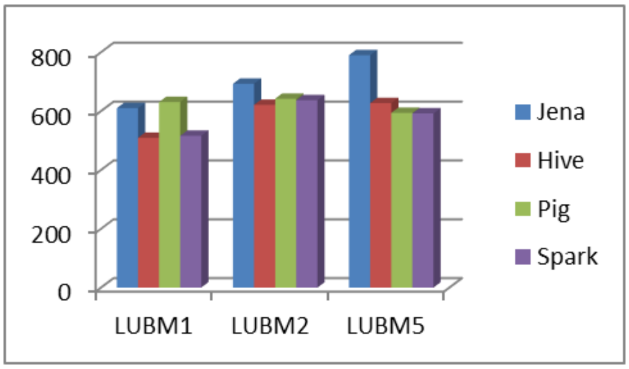


Figure 9. LUBM Q2 runtime(ms)

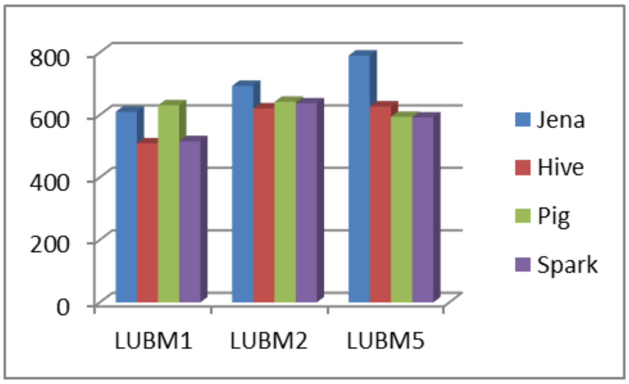


Figure 10. LUBM Q3 runtime(ms)

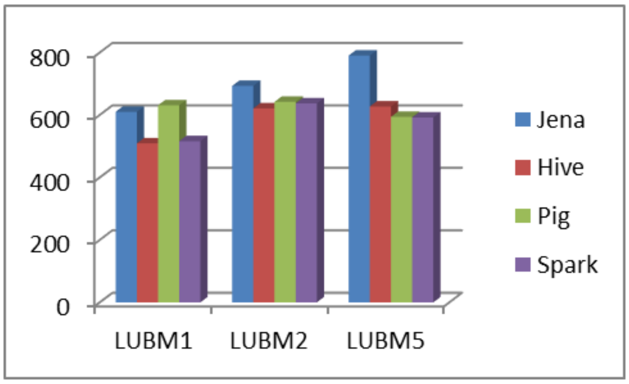


Figure 11. LUBM Q4 runtime(ms)

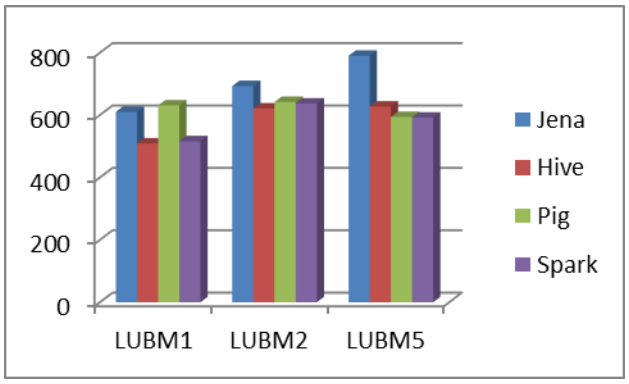


Figure 12. LUBM Q5 runtime(ms)

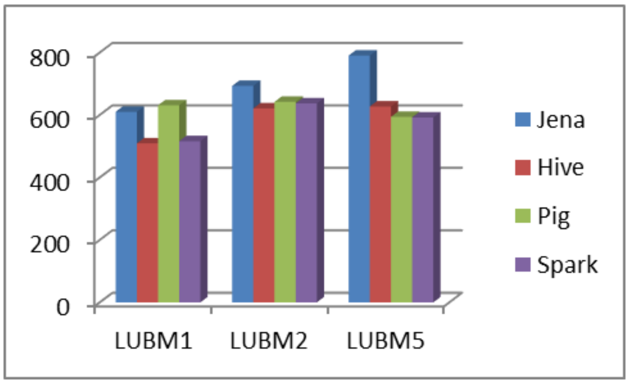


Figure 13. LUBM Q6 runtime(ms)

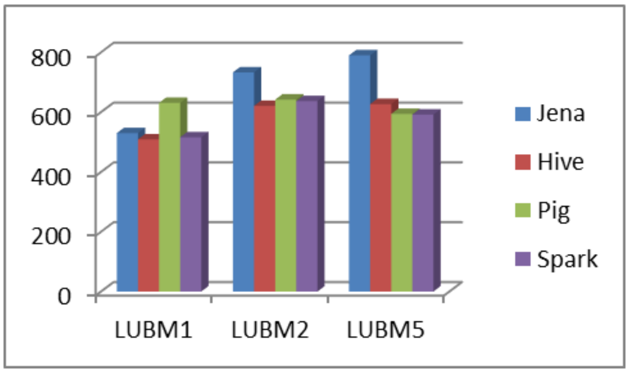


Figure 14. LUBM Q7 runtime(ms)

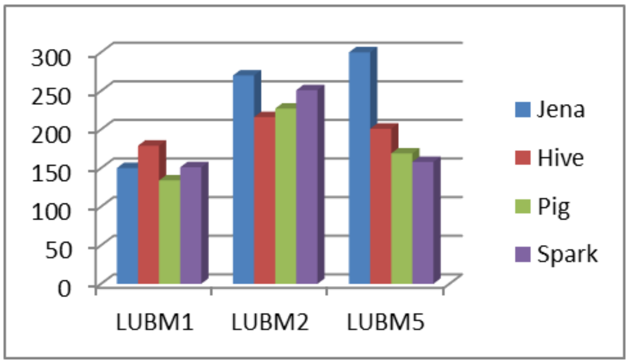


Figure 15. LUBM Q8 runtime(ms)

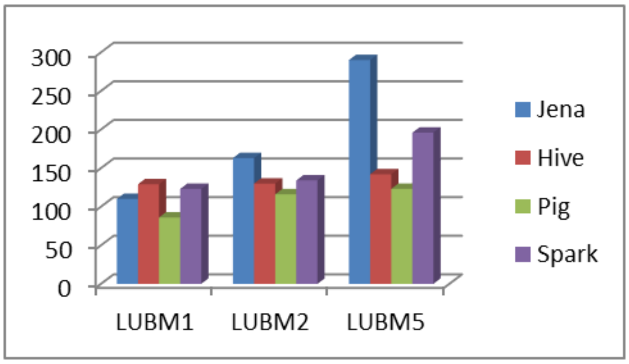


Figure 16. LUBM Q9 runtime(ms)

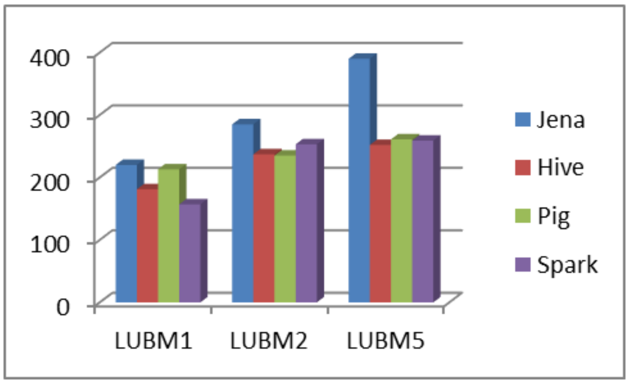


Figure 17. LUBM Q10 runtime(ms)

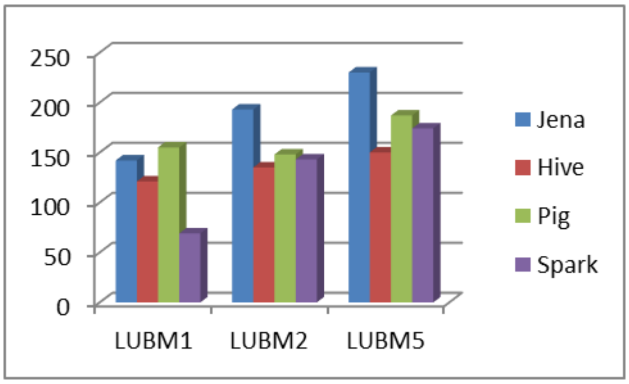


Figure 18. LUBM Q11 runtime(ms)

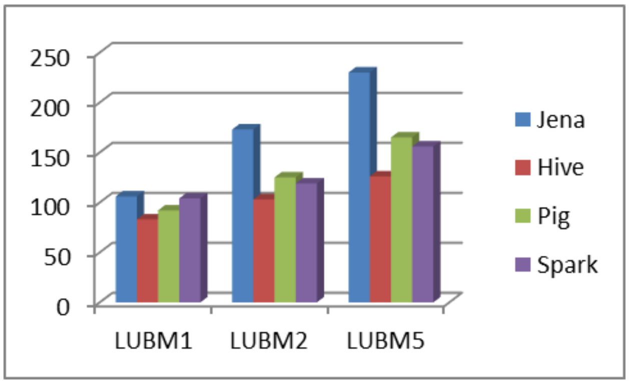


Figure 19. LUBM Q12 runtime(ms)

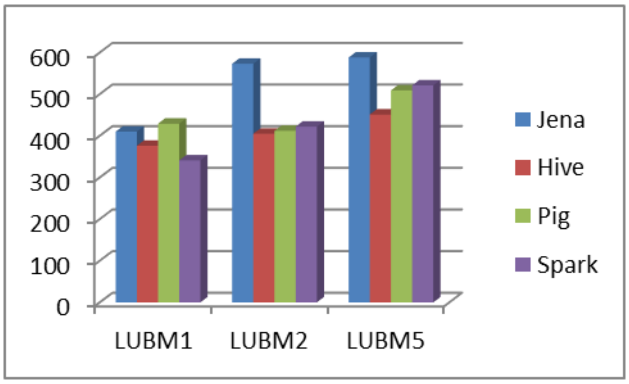


Figure 20. LUBM Q13 runtime(ms)

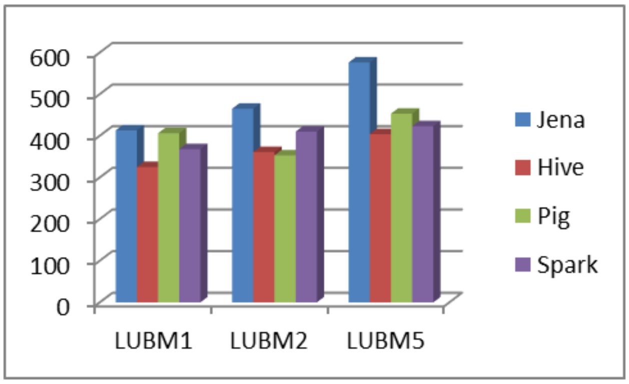


Figure 21. LUBM Q14 runtime(ms)

We can conclude from the analysis of the previous results that SPARQL2Hive is scalable system, robustness and fault tolerant. These results show the effectiveness of SPARQL2Hive when the RDF data volume is very important. SPARQL2Hive does not take a lot of time to load the data. Because it performs a simple translation of a given SPARQL query to a HiveQL program. Compared to the Jena framework, whose operation becomes a little complicated because the request goes through a set of steps, which takes a lot of time, especially for loading data, preparing data for recovery, much more that Jena uses a lot of resources such as RAM.

B. Big Data query languages experiences

Now, we experimentally evaluate the efficiency and scalability of our approach using the three Apache Hive, Apache Pig, and Spark systems.

TABLE III. PIG, HIVE, AND SPARK RUNTIME USING LUBM1, LUBM2, AND LUBM3

	Hive			**Pig**			**Spark**		
	LUBM1	***LUBM2***	***LUBM5***	***LUBM1***	***LUBM2***	***LUBM5***	***LUBM1***	***LUBM2***	***LUBM5***
Q1	481	537	752	511	752	817	437	614	803
Q2	429	516	641	463	621	502	369	536	511
Q3	535	583	633	502	619	688	548	592	673
Q4	509	621	627	631	642	594	516	637	592
Q5	743	797	851	670	803	692	671	786	664
Q6	657	720	773	664	761	685	654	728	687
Q7	678	736	794	682	718	727	628	705	730
Q8	179	216	201	134	227	169	151	251	108
Q9	129	130	142	86	116	123	123	134	216
Q10	181	237	252	213	235	261	157	253	259
Q11	121	135	150	155	148	187	69	143	174
Q12	83	103	126	92	125	165	104	119	156
Q13	376	405	451	429	412	509	341	422	521
Q14	325	361	404	406	353	453	368	410	423

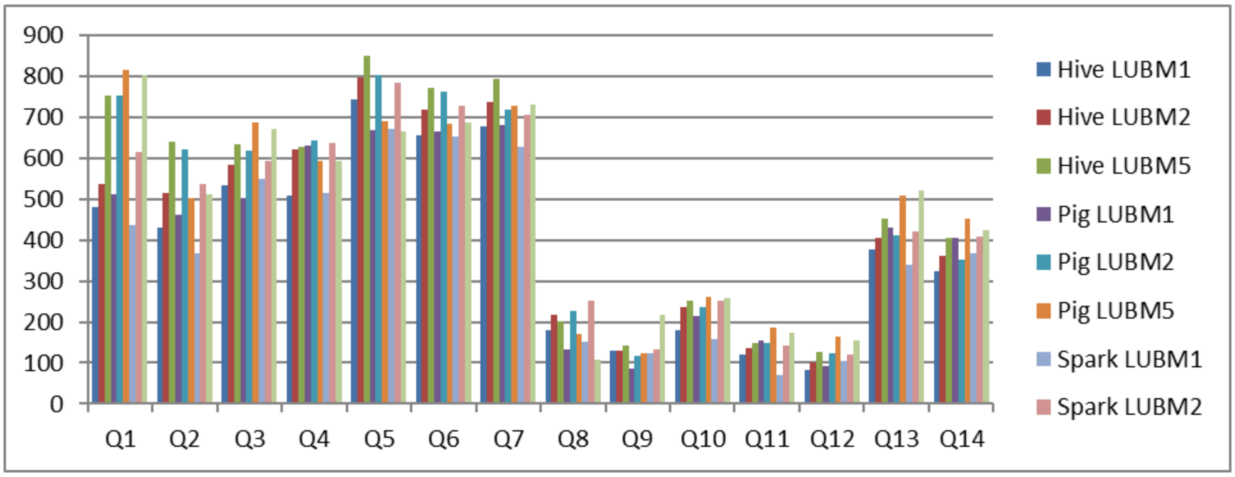


Figure 22. Pig, Hive, and Spark Runtime using LUBM1, LUBM2, and LUBM3

We note by analyzing the results of the experiments that the three systems are efficient for the management of large volumes of RDF data, and the results are very similar for the three systems and the three instances of LUBM Benchmark, with a small note for LUBM1 Hive and a bit faster than Pig and Spark for Q1, Q4, and Q14 queries versus Spark, which is good for some queries running on the LUBM2 dataset and the same for Apache Pig on the LUBM5 instance. We can conclude from this analysis that the choice between the three Apache Hive, Apache Pig, and Spark systems generally is not very important because the results obtained almost the same for all three systems on a dataset of the RDF data. Note also that the query execution time that contains complex joins is reduced compared to older systems like Jena, thanks to the two Big Data principles ie. data storage distribution and parallel processing of these data. Languages such as PigLatin, HiveQL, and Spark have been proposed, with the notable objective of expressing more powerful operators (for example joins). These operators remain executable in a MapReduce context, much like SQL is executable in a system based on file browsing. Finally, recently, systems offering richer alternatives to Hadoop have started to emerge. The main motivation is to provide support for algorithms that work by iteration. This is the case for a large number of techniques in data mining which progressively refine a result until obtaining an optimal solution. MapReduce is (was) very poorly suited to this type of execution. Systems like Spark or Flink are major advances from this point of view. Spark allows you to write complex treatments composed of several Map and Reduce phases. We can also do this with YARN, but the data from each phase must be stored on HDFS, to be reused immediately after in the next phase. It takes a lot of time and space. YARN jobs take a long time to launch and execute. There are considerable latencies. On the contrary, Spark makes much better use of the central memory of the machines in the cluster and manages the sequence of tasks itself. The treatments can be written in several languages: Scala, Java, and Python. But this is not the case for Hive and Pig.

CONCLUSION & FUTUR WORKS

Recently, the storage of large amounts of RDF data is achieved using Big Data technologies like Hadoop and NoSQL systems. To manipulate this data we will have to use the SPARQL language above Hadoop and NoSQL like the Apache Pig, Apache Hive and Spark systems. In this paper, we presented a model-driven engineering-based approach for the transformation of SPARQL queries into an Apache Pig Latin program, Apache HiveQL program, or a Spark script. In future works we will study an area of application of our solution for example e-Learning and recommendation systems, it is a case study of the use of our system. In addition, As we have seen in this work, the RDF language still poses many performance issues at the moment, mainly for reasoning over large volumes of knowledge. These efficiency needs are found in the processing of semantic data flows, where time is just as important. There are many methods to try to solve them, most of them still in development. One of the solutions studied concerns the use of RDF summaries: by considering an RDF graph, it is possible to reduce the volume of information it contains, while retaining maximum precision, in order to be able to manipulate it in a way more optimized. These summaries can be generated in different ways depending on the systems. there is a need to examine large volumes of data. Without effective treatment methods, the result can be imprecise and very long to obtain: it is, therefore, imperative to use methods allowing to obtain the essential of the relevant elements within a set of structured information.

REFRENCES

[1] Banane, Mouad, Allae Erraissi, and Abdessamad Belangour. "SPARQL2Hive: An approach to processing SPARQL queries on Hive based on meta-models." In 2019 8th International Conference on Modeling Simulation and Applied Optimization (ICMSAO), pp. 1-5. IEEE, 2019.

[2] 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.

[3] Banane, Mouad, and Abdessamad Belangour. "RDFSpark: a new solution for querying massive RDF data using spark." International Journal of Engineering & Technology 8, no. 3 (2019).

[4] Mouad Banane, and Abdessamad Belangour. « Querying massive RDF data using Spark». International Journal of Advanced Trends in Computer Science and Engineering 8, nº 4 (2019): 1481 - 1486.

[5] Mouad Banane, and Abdessamad Belangour. « RDFMongo: A MongoDB Distributed and Scalable RDF management system based on Meta-model». International Journal of Ad-vanced Trends in Computer Science and Engineering 8, nº 3 (2019): 734 – 741.

[6] Hartig, Olaf, Christian Bizer, and Johann-Christoph Freytag. "Executing SPARQL queries over the web of linked data." In International Semantic Web Conference, pp. 293-309. Springer, Berlin, Heidelberg, 2009.

[7] Prud'hommeaux, E. and Aranda, C. B. (2013). SPARQL 1.1 federated query. W3C recommendation, W3C. http://www.w3.org/ TR/sparql11-federated-query/.

[8] Solbrig, Harold R., Eric Prud'hommeaux, Grahame Grieve, Lloyd McKenzie, Joshua C. Mandel, Deepak K. Sharma, and Guoqian Jiang. "Modeling and validating HL7 FHIR profiles using semantic web Shape Expressions (ShEx)." Journal of biomedical informatics 67 (2017): 90-100.

[9] Wang, Xin, Thanassis Tiropanis, and Hugh C. Davis. "Lhd: Optimising linked data query processing using parallelisation." (2013).

[10] Dean, J. and Ghemawat, S. (2004). MapReduce : Simplified data processing on large clusters. In Sixth Symposium on Operating System Design and Implementation, pages 137–150.

[11] Du, Dayong. Apache Hive Essentials: Essential techniques to help you process, and get unique insights from, big data. Packt Publishing Ltd, 2018.

[12] Wadkar, Sameer, Madhu Siddalingaiah, and Jason Venner. Pro Apache Hadoop. Apress, 2014.

[13] Karau, Holden, and Rachel Warren. High performance Spark: best practices for scaling and optimizing Apache Spark. " O'Reilly Media, Inc.", 2017.

[14] Banane, Mouad, and Abdessamad Belangour. "A Survey on RDF Data Store Based on NoSQL Systems for the Semantic Web Applications." International Conference on Advanced Intelligent Systems for Sustainable Development. Springer, Cham, 2018.

[15] Schätzle, Alexander, Martin Przyjaciel-Zablocki, and Georg Lausen. "PigSPARQL: Mapping SPARQL to pig latin." Proceedings of the International Workshop on Semantic Web Information Management. ACM, 2011.

[16] Banane, Mouad, Abdessamad Belangour, and Labriji El Houssine. "Storing RDF data into big data NoSQL databases." First International Conference on Real Time Intelligent Systems. Springer, Cham, 2017.

[17] Sun, Jianling, and Qiang Jin. "Scalable rdf store based on hbase and mapreduce." 2010 3rd international conference on advanced computer theory and engineering (ICACTE). Vol. 1. IEEE, 2010.

[18] Mammo, Mulugeta, and Srividya K. Bansal. "Distributed sparql over big rdf data: A comparative analysis using presto and mapreduce." 2015 IEEE International Congress on Big Data. IEEE, 2015.

[19] Djebali, Sonia, and Thomas Raimbault. "SimplePARQL: a new approach using keywords over SPARQL to query the web of data." Proceedings of the 11th International Conference on Semantic Systems. ACM, 2015.

[20] Dhruba Borthakur. 2008. HDFS architecture guide. HADOOP APACHE Proj. Httphadoop Apache Orgcommondocscurrenthdfs.

[21] Jeffrey Dean and Sanjay Ghemawat. 2010. MapReduce: a flexible data processing tool. Commun. ACM 53, 1 (2010), 72–77.

[22] Dimiduk, N., Khurana, A., Ryan, M. H., & Stack, M. (2013). HBase in action. Shelter Island: Manning.

[23] Aniceto, R., Xavier, R., Guimarães, V., Hondo, F., Holanda, M., Walter, M. E., & Lifschitz, S. (2015). Evaluating the Cassandra NoSQL database approach for genomic data persistency. International journal of genomics, 2015.

[24] Chodorow, Kristina. MongoDB: the definitive guide: powerful and scalable data storage. " O'Reilly Media, Inc.", 2013.

[25] Sirin, Evren, Bijan Parsia, Bernardo Cuenca Grau, Aditya Kalyanpur, and Yarden Katz. "Pellet: A practical owl-dl reasoner." Web Semantics: science, services and agents on the World Wide Web 5, no. 2: 51-53. Elsevier.

[26] Özsu, M. Tamer. "A survey of RDF data management systems." Frontiers of Computer Science 10.3 (2016): 418-432. Elsevier.

[27] Schätzle, A., Przyjaciel-Zablocki, M., Skilevic, S., & Lausen, G. (2016). S2RDF: RDF querying with SPARQL on spark. Proceedings of the VLDB Endowment, 9(10), 804-815.

[28] Schmidt, Douglas C. "Model-driven engineering." COMPUTER-IEEE COMPUTER SOCIETY- 39.2: 25.

[29] Jouault, F., Allilaire, F., Bézivin, J., & Kurtev, I. (2008). ATL: A model transformation tool. Science of computer programming, 72(1-2), 31-39. Elsevier.

[30] 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. Elsevier.

[31] Lee, C. I., Hsia, T. C., Hsu, H. C., & Lin, J. Y. (2017, April). Ontology-based tourism recommendation system. In 2017 4th International Conference on Industrial Engineering and Applications (ICIEA) (pp. 376-379). IEEE.

[32] Ayala, V. A. A., Przyjaciel-Zablocki, M., Hornung, T., Schätzle, A., & Lausen, G. (2014, June). Extending sparql for recommendations. In Proceedings of Semantic Web Information Management on Semantic Web Information Management (pp. 1-8). ACM.

[33] Khrouf, H., & Troncy, R. (2013, October). Hybrid event recommendation using linked data and user diversity. In Proceedings of the 7th ACM conference on Recommender systems (pp. 185-192). ACM.

[34] Rowe, Matthew. "SemanticSVD++: incorporating semantic taste evolution for predicting ratings." 2014 IEEE/WIC/ACM International Joint Conferences on Web Intelligence (WI) and Intelligent Agent Technologies (IAT). Vol. 1. IEEE, 2014.

Mouad Banane is a PhD at the Faculty of Sciences Ben M‘Sik, Laboratory of Information Technology and Modeling (LTIM), Hassan II University of Casablanca,
Morocco. His research fields: Model Driven Engineering, Semantic Web, Big Data and Analytics and Internet of Thing .

Abdessamad Belangour is a professor of higher education at the Faculty of Sciences Ben M‘Sik from Laboratory of Information Technology and Modeling (LTIM), Hassan II University of Casablanca, Morocco His major research interest is on Systems engineering Model Driven Engineering, Big Data, and Internet of Thing.