A Big Data Solution To Process Semantic Web Data Using The Model Driven Engineering Approach

Mouad Banane, Abdessamad Belangour

Abstract: The digital transformation of companies and organizations leads to an evolution of databases towards Big Data. Our work is part of this transformation and concerns more specifically the mechanisms for processing semantic data stored on a Hadoop Big Data or NoSQL platform. Generally, for querying semantic Web data we have to use SPARQL language, and Big Data languages like Apache Spark, are not dedicated to processing this type of data. In this paper, we present a new model engineering approach for analyzing and transforming complex SPARQL queries into Spark scripts. To automate this transformation process, we used the MDE approach that provides a formal framework for the transformation and mapping mechanisms between two languages. From a SPARQL metamodel, we propose transformation rules to generate, in fine, a Spark script intended for a Hadoop / NoSQL platform. We introduce a logic level intermediate schema to limit the impact of Spark releases. An experimentation of the transformation process was carried out on three RDF databases.

Index Terms: Big Data, Semantic Web, SPARQL, RDF, Spark.

1 INTRODUCTION

THanks to the efforts of the World Wilde Web Consortium community, 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. The OWL and RDF 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. SPARQL is the standard language for querying semantic graphs, this solution is not suitable for large semantic graphs. In this paper, we present a new model engineering approach for analyzing and transforming complex SPARQL queries into Spark scripts. We then analyze theoretically the efficiency of our method, and we calculate the dominant parameters of the system. Finally, we implement the whole system on the Big Data platform using data from LUBM Benchmark [1], DBpedia [2], and SP2Bench [3]. In the next section, we discuss previous work in the literature and then clarify our purpose and the tools used, and present our system in section 3, our system is evaluated in section 4. Finally, the discussion in section 5.

2 RELATED WORK

Several previous works used Big Data technologies for Semantic Web data management such as: RDFMongo [4] it is a complete RDF data management solution based on the MongoDB database, in this solution the data is stored in MongoDB, and for querying this data the user request written in SPARQL will be translated into a program of MongoDB query Language, Our second system [5] is an approach based on the principle of metamodels, it allows the processing of Complex SPARQL queries using a Big Data Query Tool called Hive. We also find a system based on Apache Spark [6] to handle large volumes of RDF data. Other research works like [7] and [8] are comparative studies and surveys of semantic Web systems based on Big Data tools. Big data and its processing tools are developing rapidly, reaching and affecting more and more domains. Skourletopoulos et al. [9] and Hashem et al. [10] explored the opportunities, challenges, and techniques of Big Data and their relationship with cloud technologies, such as Big Data-As-A-Service. Besides, they identified some data analysis challenges such as scalability, availability, data integrity, and data transformation. Li et al. [11] gave an overview of the techniques and systems used for Big Data storage and resource management such as Distributed File System, Distributed Databases, Access Interface, and Query Language. They also briefly presented three main problems in this area:

- Large scale storage.
- The management complexity caused by the heterogeneity of the data.
- The requirement in terms of performance and reliability of storage.
- Improving the query index in distributed systems.
- Storage and processing in real-time and streaming of Big Data.
- Complex event processing.

To highlight the challenges of storage, Begoli [12] published a state-of-the-art review of architectures and platforms for large-scale data analysis and knowledge discovery from Data. Mazumder [13] exposed concepts, common techniques, and models in Big Data such as storage and service management as well as the Hadoop ecosystem. They also proposed implementations, use cases (data ingestion, data mining, information creation, data consumption, archiving and data purging) and mapping them to various Big tools and platforms. data. The process of extracting data is as important as the data itself. Thus, the need to analyze and qualify data collected from various sources is one of the main drivers of Big Data analysis tools. For this, several contributions have been made to discuss and propose improvements to current analysis techniques. As one of the main success factors of Big Data is the ability to manage constraints in real-time, Liu,

Iftikhar and Xie [14], Zheng et al. [15] and Mohamed and Al-Jaroodi [16] have addressed this topic by providing an overview of the current state of the tools that can handle Big Data. Liu, Iftikhar, and Xie [14] presented an analysis of open source real-time processing technologies with a focus on real-time architectures. Zheng et al. [15] discussed the challenges of Big Data and in particular those of real-time processing. They also presented a multilevel storage model and some deployment methods to meet Big Data requirements in real-time and heterogeneity. Mohamed and Al-Jaroodi [16] presented some technical challenges to real-time applications in Big Data. Also, they provided a performance analysis and some Big Data requirements in real-time. Other researchers have focused on the comparison of Big Data processing tools. Indeed, Lopez, Lobato, and Duarte [17] have described, analyzed and compared three main open-source distributed flow processing platforms such as Spark, Flink, and Storm. They provided experimental performance results focused on throughput and parallelism in a threat detection application in network traffic. In a study conducted by Lu et al. [18], a repository of modern distributed flow calculation frameworks is proposed. This benchmark covers performance, fault recovery capacity and sustainability of Big data frameworks. Nevertheless, this study provides a result only for Storm and Spark. Hess et al. [19] proposed an overview and a comparison of four flow processing platforms, one platform more (Samza) than the study conducted by Lopez, Lobato, and Duarte [17]. However, the comparison is based only on the architecture and the responsibilities of the system components and does not allow to conclude to superiority in terms of efficiency. In a study by Lu et al [18], a benchmark based on error recovery capability as well as sustainability for modern distributed flow computing tools is presented. The paper provides a set of comparison results between Storm and Spark. Liu, Iftikhar, and Xie [14] presented real-time/near real-time open-source processing technologies while focusing on their architectures and platforms. Yadranjiaghdam, Pool, and Tabrizi [20] presented tools for real-time analysis of big data and classified different studies according to the tools used and the type of application. They focused on applications related to surveillance, the environment, social media, and health care. Tsai et al. [21] discussed solutions for large data analysis. Besides, they exposed some research directions and questions about open techniques and platforms in this area of research. Gong, Morandini, and Sinnott [22] proposed a benchmarking and implementation of SMASH, a generic and highly scalable Cloud solution, for processing large-scale traffic data. Katal, Wazid, and Goudar [23] compared three frameworks (Storm, Hadoop, Drill) through the following characteristics: owner, workload, source code, low latency, and complexity. Also, the issues and challenges of managing Big Data were discussed. In the study by Almeida and Bernardino [24], the authors compared six open source Big Data platforms: Apache Mahout, MOA, Project R, Vowpal Wabbit, and GraphLab, according to the programming paradigm and programming language. interface, type of data and algorithms supported. Urmila [25] introduced and compared Hive, Pig, and MapReduce for Big Data analysis. The comparison is based on the type of language, the user interface, the available algorithms and the scale of data supported in each tool. Some research has focused on NoSQL (Not Only Structured Query Language). Indeed, Corbellini et al. [26] compared NoSQL storage systems based on their types (key-value, column, document, and row databases). The comparison is based on API, language and persistence. Nevertheless, it does not allow us to choose the database adapted to the application needs. This study covers more NoSQL databases than other studies such as the one conducted by Dede et al. [27]. However, Moniruzzaman and Hossain [28] covered fewer NoSQL databases than Corbellini et al. [26] but cited more comparison attributes such as integrity (atomicity, consistency, isolation, durability ...), indexing (geospatial index, secondary indexes) and distribution (replication, scalability horizontal ...). It is important to note that Hadoop is not the only attractive platform for Big Data, there is also Apache Spark. While both tools are sometimes considered competitors, it is often accepted that they work even better when combined. Apache Spark has become increasingly suited as a high-level project for Big Data analysis, so many research tends to focus on improving it. For example, Gulzar et al. [29] created a tool called BigDebug that provides real-time interactive debug primitives for massive data processing in Spark. NetSpark, an improved Spark framework is introduced by Li, Chen, and Xu [11]. This Framework reduces the execution time of a Spark task by combining network buffer management, hardware-supported Remote Direct Memory Access (RDMA) technology, and optimization. Serialization of the data. A strategy, called MPTE (Multiple Phases Time Estimation), was presented by Yang et al. [30] to reduce the impact of "straggler machines". In addition, the scheduling of backup tasks has been improved by designing a new task scheduler. Spark uses a fast calculation in memory to process the data. However, in-memory processing can cause a problem of volatility, failure, or lack of a Resilient Distributed Datasets (RDD) that will cause Spark to recalculate all the missing RDDs on the lineage. A long lineage will also increase the cost in time and the use of the driver's memory analyzing the lineage. For this, Zhu, and Hu [31] presented an automatic checkpoint algorithm to solve Spark's "long lineage" problem with little impact on overall performance. Other studies have designed new Spark-based frameworks to make big data analysis more powerful. For example, Yan et al. [32] designed TR-Spark to deal with transient resource issues. This framework can run as a secondary background on transient resources and makes Spark-based applications more efficient. The design of this new framework is based on two principles: the stability of resources and the planning of the reduction of the size of the data. The combination of these principles allows TR-Spark to adapt to the stability of the infrastructure. To better make business decisions, Park et al. [33] relied on Spark to provide a goal-oriented Big Data analytics framework. The latter was experienced on the shipping decision.

3 SYSTEM ARCHITECTURE

In this section, we present the architecture of our system, this architecture which consists of three parts the first is the proposal of a SPARQL Metamodel, the second is to propose also a Metamodel is the Spark Metamodel, and in the third part we realize a transformation between these two meta-models using a transformation language called QVT (Query View Transformation) [34]. A SPARQL query Q that conforms to the SPARQL metamodel will be transformed/mapped into a Spark script S, this Spark script conforms to the Spark metamodel.

$$Spark\ script\ S = trans_{QVT}(SPARQL\ query\ Q)$$

Figure 1 illustrates the architecture of our system, as well as the relationships between models and metamodels.

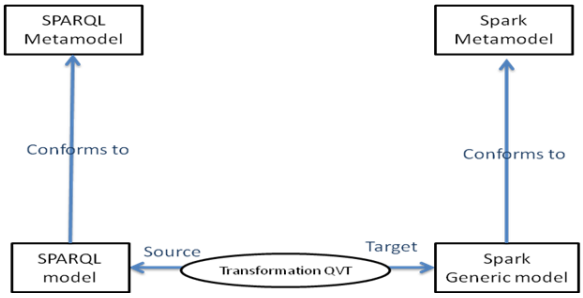

Fig.1 *System Architecture*

3.1 SPARQL metamodel

In particular, SPARQL does not have a FROM construct to define the relations we are going to query. By default, any relation present in the database can be queried. A "relationship" does not have to be declared explicitly and can be any URI. SPARQL is therefore by default much more open than SQL, since we are not limited to using the resources defined in our database instance. SPARQL [35] is the language recommended by the W3C for querying RDF graphs. SPARQL 1.0 is the first recommendation for querying RDF data. It defines requests of the form SELECT, ASK, DESCRIBE and CONSTRUCT. A query of the form SELECT returns the list of variable associations of the query to resources or literals for which there is a matching of the WHERE clause graph of the query with the RDF graph queried. A SPARQL query of the ASK form makes it possible to ask boolean response requests on an RDF graph; it allows to ask if a pairing exists between the query graph and the RDF graph; it controls the search for a match between the graph of its WHERE clause and the RDF graph questioned. A SPARQL query of the form CONSTRUCT produces a new RDF graph by replacing the variables of the graph of the CONSTRUCT clause with the values for which the query graph of the WHERE clause matches with the RDF graph queried. A SPARQL query of the DESCRIBE form makes it possible to request the RDF description of a resource present in the stored RDF data; its result is a set of RDF triples describing the targeted resource. The WHERE clause of the SPARQL queries of the form SELECT, ASK or CONSTRUCT contains a graph pattern constructed as a conjunction of triples where a variable can take the place of a resource or a literal. It can also be a union of graphs, some parts of the query graph can be declared optional, filters can be applied to the solutions of a pairing:

- A UNION B brings together the solutions of graphs A and B;
- A OPTIONAL B makes it possible to return the solutions of A, whether there is a solution with B or not;
- A FILTER B makes it possible to restrict the solutions of A using a Boolean expression B.
- Operators make it possible to modify the sequence of the solutions:
- DISTINCT eliminates duplicates;
- ORDER BY allows you to sort the solutions;
- LIMIT allows cutting the number of solutions by limiting the number of returned solutions;
- OFFSET allows cutting the number of solutions by indicating to begin the solutions generated after the indicated number of solutions.

Figure 2 shows our proposed SPARQL metamodel.

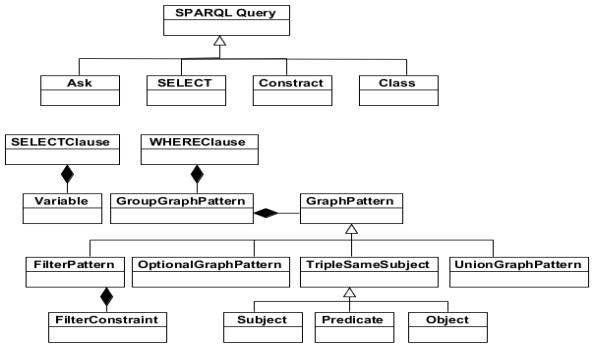

Fig.2 *System Architecture*

3.1 Spark metamodel

In this section, we present our Spark metamodel proposition shown in Figure 3. A Spark package is composed of Spark models, and a Spark Classifier can be either a Spark Class or a Spark Interface. A Spark Bihavioral is composed of Spark Parameters and Blocks, the blocks can contain one or more instructions.

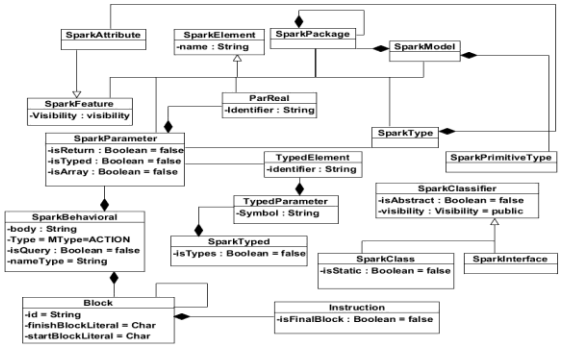

Fig.3 *Proposed Spark metamodel*

3.3 Transformation

The transition between the different models presented before is done through the execution of transformations. To transform a model SPARQL to model Spark, we have to use a transformation language such as ATL [36], and QVT [37]. ATL is an open-source language, widely used in the industry. Its implementation is robust and powerful. But ATL is not a norm. Its competitor has the advantage of being a standard of the OMG Object Management Group, the organization that has notably made UML, MDA,... This is QVT (Query View Transform). QVT allows you to normalize a way to express correspondences (transformations) between languages defined with MOF (Meta Object Facility). This language provides mechanisms for expressing Queries (Query Q) to filter and select elements of a model (including selecting the source elements of a transformation), and to propose a mechanism for creating views (Views V) that are models deduced from another to reveal specific aspects. It also allows formalizing a way of describing transformations (Transformations T). There are two types of QVTs: QVTd

(declarative, QVTr Relation and QVTc Core), equivalent to the ATL "matched rules" and QVT Operational (imperative), equivalent to "called rules" ATL. The QVT language could be the ultimate "solution" for transforming models. Only here, as very often in the standards, QVT suffers from a great complexity. The QVTo version starts to be implemented while QVT-Relation is in the embryonic state. Figure 4 below shows a simple example of QVT code.

```
modeltype SPARQLSpark uses SPARQL('http:///SPARQL.ecore');

transformation NewTransformation(in source:SPARQL, out target:Spark);

main() {
      source.rootObjects()[Root]->map Root2Root();
}

mapping Root :: Root2Root() : Root {
      element += self.element->select(SELECT |
            SELECT.oclIsKindOf(Query))[Query]->map Query2Script();
}

mapping Query :: Query2Script() : Script
      when {
            self.id > 0
      }
      {
      result.id := self.id;
      result.Load := self.SELECT + " Column!";
}
```

Fig.4 *Part of QVT code*

4 EXPERIENCE & VALIDATION

For the validation of our approach, we realize experiments using our system, we also give a comparison of the results obtained with the two famous existing systems Sesame and Jena.

4.1 Configuration and Test Environment

We describe in this section the configuration of the test environment and tools used, we performed two tests, the first on a single machine mode of 4TB disk storage and 16GB RAM storage, the second test on a Distributed mode this cluster of three server disk storage 4TB disk storage and 16GB RAM storage. Our experiments are based on three different datasets: LUBM Benchmark, DBpedia, and SP2Bench. For tools: use Apache Spark 2.4.3, with Hadoop version 3. Apache Jena version 3.12.0. and for Sesame, we used the Sesame API 2.x. the Sesame API for handling RDF in Java program requires the recovery of several JAVA JARs. The Java code example below: Figure 5 illustrates initializing a Repository, loading RDF data, writing RDF to a file, and querying a remote Sesame server or public SPARQL endpoint.

```
Repository repository = new SailRepository(new MemoryStore());
repository.initialize();

File fileToAdd = new File("/mouad/dataset/DBpedia.rdf");
repository.getConnection().add(
    fileToAdd,
    RDF.NAMESPACE,
    RDFFormat.forFileName(fileToAdd.getName())
);
repository.getConnection().export(new RDFXMLWriter(new FileOutputStream("/DBpedia.rdf")));
Repository sesameServer = new HTTPRepository("http://localhost:8080/openrdf-sesame","Test1");
sesameServer.initialize();

Repository dbpedia = new HTTPRepository("http://dbpedia.org/sparql");
dbpedia.initialize();
```

Fig. 5 *MapReduce JAVA code example for initializing a Repository, loading RDF data, and writing RDF to a file and querying a Sesame server.*

4.2 Experiments results

In this section, we present the results of experiments conducted on three Benchmark LUBM, DBpedia, and SP2Bench. Figure 6 illustrates the execution time result of a SPARQL query on a centralized mode of a single server and figure 7 presents the result of the execution time of this SPARQL query on a distributed mode (cluster of servers). In the figures our system is named Big DataSW.

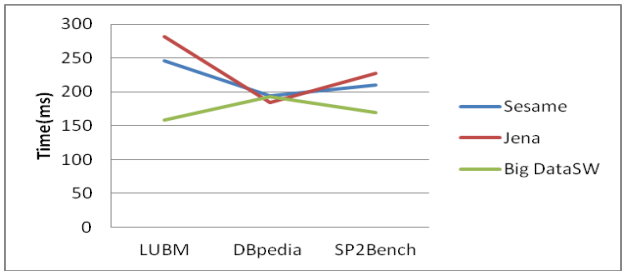

Fig. 6 *Runtime on one machine*

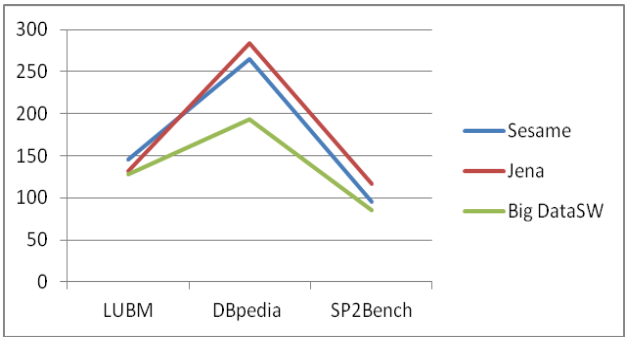

Fig. 7 *Runtime on a cluster of distributed machines*

5 DISCUSSION

We analyze these results; we can notice that there is a difference between the centralized mode (a single machine) and the distributed mode (cluster of the machines). The three solutions: Sesame, Jena, and our systems give similar results for running on a single server. But for the distributed mode we notice the big difference between the three systems, Our system reduces the execution time of a SPARQL query for the three datasets used in this experiment LUBM, DBpedia, and SP2Bench. Thanks to the use of Spark, the execution time of complex SPARQL queries using our system has been reduced, the answers are optimal at the join level because our system uses an indexing scheme that is transformed directly from the SPARQL query, this query that conforms to our Spark metamodel. Another very important factor is that Spark is more than 100 times very fast than MapReduce, so executing a Spark script will be very fast due to the execution of a MapReduce Job. To validate the choice of Spark, we found that the method used by Spark to process the data makes it much faster than MapReduce. While MapReduce works in stages, Spark can work on all data at one time. The MapReduce Job Sequence looks like this: it reads the data at the cluster level, it executes an operation, writes the results at the cluster level, reads the cluster-level updated data again, executes the cluster next operation, he writes the new results at the cluster level, and so on. On the contrary, Spark performs all data analysis operations in memory and near real-time. Spark reads the data at the cluster level, performs any necessary analysis, writes the results at the cluster level. It is important to note that Spark can run on several file and database systems, including HDFS (Hadoop Distributed File

System) [38]. Spark is a step ahead of MapReduce because it handles most of its operations in memory, copying datasets from a physical storage system to much faster RAM. For its part, MapReduce [39] writes and reads data from the hard disk. If disk access can take several milliseconds to access 1 MB of data, the data access rates in memory go below the millisecond. Another advantage of Spark on MapReduce, its relative ease of use and its flexibility. Finally, to conclude, the fundamental difference between Hadoop MapReduce and Spark is that Spark writes data in RAM, not on disk. This has several important consequences on the speed of calculation processing as well as on the global architecture of Spark.

6 CONCLUSION

Big Data technologies are capable of handling large volumes of data, but are not dedicated to managing semantic web data that is typically stored in RDF standard format. To query this data we must use the SPARQL querying standard. In this paper, we present a new model engineering approach for analyzing and transforming complex SPARQL queries into Spark scripts. To automate this transformation process, we used the MDE approach that provides a formal framework for the transformation and mapping mechanisms between two languages.

7 REFERENCES

[1] Y. Guo, 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, no. 2-3 (2005): 158-182.

[2] M. Morsey, Jens Lehmann, Sören Auer, and Axel-Cyrille Ngonga Ngomo. "DBpedia SPARQL benchmark–performance assessment with real queries on real data." In International semantic web conference, pp. 454-469. Springer, Berlin, Heidelberg, 2011.

[3] M. Schmidt, Thomas Hornung, Georg Lausen, and Christoph Pinkel. "SP^ 2Bench: a SPARQL performance benchmark." In 2009 IEEE 25th International Conference on Data Engineering, pp. 222-233. IEEE, 2009.

[4] M. 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.

[5] M. Banane, 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).

[6] M. 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.

[7] Ma, Zongmin, and Li Yan. "Towards Massive RDF Storage in NoSQL Databases: A Survey." In Emerging Technologies and Applications in Data Processing and Management, pp. 263-284. IGI Global, 2019.

[8] M. 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.

[9] G. Skourletopoulos, Constandinos X. Mavromoustakis, George Mastorakis, Jordi Mongay Batalla, Ciprian Dobre, Spyros Panagiotakis, and Evangelos Pallis. "Big data and cloud computing: a survey of the state-of-the-art and research challenges." In Advances in mobile cloud computing and big data in the 5G Era, pp. 23-41. Springer, Cham, 2017.

[10] I. A. T. Hashem, Ibrar Yaqoob, Nor Badrul Anuar, Salimah Mokhtar, Abdullah Gani, and Samee Ullah Khan. "The rise of “big data” on cloud computing: Review and open research issues." Information systems 47 (2015): 98-115.

[11] J. Li, Zheng Xu, Yayun Jiang, and Rui Zhang. "The overview of big data storage and management." In 2014 IEEE 13th International Conference on Cognitive Informatics and Cognitive Computing, pp. 510-513. IEEE, 2014.

[12] E. Begoli. "A short survey on the state of the art in architectures and platforms for large scale data analysis and knowledge discovery from data." In Proceedings of the WICSA/ECSA 2012 Companion Volume, pp. 177-183. ACM, 2012.

[13] S. Mazumder. "Big data tools and platforms." In Big Data Concepts, Theories, and Applications, pp. 29-128. Springer, Cham, 2016.

[14] [X. Liu, Nadeem Iftikhar, and Xike Xie. "Survey of real-time processing systems for big data." In Proceedings of the 18th International Database Engineering & Applications Symposium, pp. 356-361. ACM, 2014.

[15] Z. Xie, Guannan Liu, Junjie Wu, Lihong Wang, and Chunyang Liu. "Wisdom of fusion: Prediction of 2016 Taiwan election with heterogeneous big data." In 2016 13th international conference on service systems and service management (ICSSSM), pp. 1-6. IEEE, 2016.

[16] N. Mohamed, and Jameela Al-Jaroodi. "Real-time big data analytics: Applications and challenges." In 2014 international conference on high performance computing & simulation (HPCS), pp. 305-310. IEEE, 2014.

[17] M. A. Lopez, Antonio Gonzalez Pastana Lobato, and Otto Carlos MB Duarte. "A performance comparison of open-source stream processing platforms." In 2016 IEEE Global Communications Conference (GLOBECOM), pp. 1-6. IEEE, 2016.

[18] R. Lu, Gang Wu, Bin Xie, and Jingtong Hu. "Stream bench: Towards benchmarking modern distributed stream computing frameworks." In 2014 IEEE/ACM 7th International Conference on Utility and Cloud Computing, pp. 69-78. IEEE, 2014.

[19] G. Hesse, and Martin Lorenz. "Conceptual survey on data stream processing systems." In 2015 IEEE 21st International Conference on Parallel and Distributed Systems (ICPADS), pp. 797-802. IEEE, 2015.

[20] B. Yadranjiaghdam, Nathan Pool, and Nasseh Tabrizi. "A survey on real-time big data analytics: applications and tools." In 2016 International Conference on Computational Science and Computational Intelligence (CSCI), pp. 404-409. IEEE, 2016.

[21] C. W. Tsai, Chin-Feng Lai, Han-Chieh Chao, and Athanasios V. Vasilakos. "Big data analytics: a survey." Journal of Big data 2, no. 1 (2015).

[22] Y. Gong, Luca Morandini, and Richard O. Sinnott. "The design and benchmarking of a cloud-based platform for processing and visualization of traffic data." In 2017 IEEE International Conference on Big Data and Smart Computing (BigComp), pp. 13-20. IEEE, 2017.

[23] A. Katal, Mohammad Wazid, and R. H. Goudar. "Big data: issues, challenges, tools and good practices." In 2013 Sixth international conference on contemporary computing (IC3), pp. 404-409. IEEE, 2013.

[24] PDC. de Almeida, and Jorge Bernardino. "Big data open source platforms." In 2015 IEEE International Congress on Big Data,

pp. 268-275. IEEE, 2015.
[25] U. R. Pol. "Big data analysis: comparison of hadoop mapreduce, pig and hive." Int. J. Innov. Res. Sci. Eng. Technol 5, no. 6 (2016).
[26] A. Corbellini, Cristian Mateos, Alejandro Zunino, Daniela Godoy, and Silvia Schiaffino. "Persisting big-data: The NoSQL landscape." Information Systems 63 (2017): 1-23.
[27] E. Dede, Madhusudhan Govindaraju, Daniel Gunter, Richard Shane Canon, and Lavanya Ramakrishnan. "Performance evaluation of a mongodb and hadoop platform for scientific data analysis." In Proceedings of the 4th ACM workshop on Scientific cloud computing, pp. 13-20. ACM, 2013.
[28] M. Moniruzzaman, and Syed Akhter Hossain. "Nosql database: New era of databases for big data analytics-classification, characteristics and comparison." arXiv preprint arXiv:1307.0191 (2013).
[29] M. A. Gulzar, Matteo Interlandi, Tyson Condie, and Miryung Kim. "Bigdebug: Interactive debugger for big data analytics in apache spark." In Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering, pp. 1033-1037. ACM, 2016.
[30] Y. Ma, Yulei Wang, Jun Yang, Yiming Miao, and Wei Li. "Big health application system based on health internet of things and big data." IEEE Access 5 (2016): 7885-7897.
[31] H. Yang, Xianyang Liu, Shenbo Chen, Zhou Lei, Hongguang Du, and Caixin Zhu. "Improving Spark performance with MPTE in heterogeneous environments." In 2016 International Conference on Audio, Language and Image Processing (ICALIP), pp. 28-33. IEEE, 2016.
[32] Y. Yan, Yanjie Gao, Yang Chen, Zhongxin Guo, Bole Chen, and Thomas Moscibroda. "Tr-spark: Transient computing for big data analytics." In Proceedings of the Seventh ACM Symposium on Cloud Computing, pp. 484-496. ACM, 2016.
[33] G. Park , Sooyong Park, Latifur Khan, and Lawrence Chung. "IRIS: A goal-oriented big data analytics framework on Spark for better Business decisions." In 2017 IEEE International Conference on Big Data and Smart Computing (BigComp), pp. 76-83. IEEE, 2017.
[34] P. Stevens, "Bidirectional model transformations in QVT: semantic issues and open questions." Software & Systems Modeling 9, no. 1 (2010).
[35] H.Ahuja. "Implementation of FOAF, AIISO and DOAP ontologies for creating an academic community network using semantic frameworks." International Journal of Electrical & Computer Engineering. 9 (2019).
[36] F. Jouault, Freddy Allilaire, Jean Bézivin, and Ivan Kurtev. "ATL: A model transformation tool." Science of computer programming 72, no. 1-2 (2008): 31-39.
[37] M. L.Drago, Carlo Ghezzi, and Raffaela Mirandola. "A quality driven extension to the QVT-relations transformation language." Computer Science-Research and Development 30, no. 1 (2015): 1-20.
[38] M. A. Rahman, J. Hossen, C. Venkataseshaiah, Ck Ho, Kim Geok Tan, Aziza Sultana, M. Z. H. Jesmeen, and Ferdous Hossain. "A Survey of Machine Learning Techniques for Self-tuning Hadoop Performance." International Journal of Electrical and Computer Engineering 8, no. 3 (2018): 1854.
[39] G. Bathla, Himanshu Aggarwal, and Rinkle Rani. "A Novel Approach for Clustering Big Data based on MapReduce." International Journal of Electrical & Computer Engineering (2088-8708) 8, no. 3 (2018).