Journal on Big Data
DOI:10.32604/jbd.2020.010358
Article

An Improved Distributed Query for Large-Scale RDF Data

Aoran Li[1], Xinmeng Wang[1], Xueliang Wang[4] and Bohan Li[1,2,3,*]

[1]College of Computer Science and Technology, Nanjing University of Aeronautics and Astronautics, Nanjing, 211106, China
[2]Key Laboratory of Safety-Critical Software, Ministry of Industry and Information Technology, Nanjing, 211106, China
[3]Collaborative Innovation Center of Novel Software Technology and Industrialization, Suzhou, 215000, China
[4]School of Data Science and Technology, Heilongjiang University, Harbin, 150080, China
[*]Corresponding Author: Bohan Li. Email: bhli@nuaa.edu.cn
Received: 20 April 2020; Accepted: 24 August 2020

Abstract: The rigid structure of the traditional relational database leads to data redundancy, which seriously affects the efficiency of the data query and cannot effectively manage massive data. To solve this problem, we use distributed storage and parallel computing technology to query RDF data. In order to achieve efficient storage and retrieval of large-scale RDF data, we combine the respective advantage of the storage model of the relational database and the distributed query. To overcome the disadvantages of storing and querying RDF data, we design and implement a breadth-first path search algorithm based on the keyword query on a distributed platform. We conduct the LUBM query statements respectively with the selected data sets. In experiments, we compare query response time in different conditions to evaluate the feasibility and correctness of our approaches. The results show that the proposed scheme can reduce the storage cost and improve query efficiency.

Keywords: RDF; distributed query; HBase; query optimization

1 Introduction

In previous studies, researchers used simple sampling for data analytics. The arrival of big data opens up a new era of data value [1], and dealing with large-scale data analysis becomes a hot topic. The basic elements of a knowledge graph are interconnected entities and corresponding attributes, which can be expressed with subject-predicate-object. Typically, we use RDF (Resource Description Framework) to represent the storage of this ternary relationship. The efficient management of RDF data is at the center of knowledge graph and semantic web development. The traditional centralized relational database structure is rigid and fixed, and the search results are distinct. As RDF data has no fixed pattern, it cannot guarantee low data redundancy, which seriously affects the efficiency of data queries and cannot effectively manage massive data. For example, the query efficiency of a 3-store system [2] and RDF-3x [3–4] is far lower than that of big data distributed queries. Therefore, in order to solve the above problems, more and more solutions based on distributed systems are applied to data storage and query, which is of great significance to solve the explosive growth of data management problems. However, how to efficiently manage the growing RDF data and conduct large-scale data analysis is still the focus of many scholars. This paper combines big data and knowledge graph by using distributed system storage and calculation and proposes a breadth-first path search algorithm based on keyword queries under the distributed platform, so as to successfully solve data extraction, fusion, storage, calculation, and other problems.

This paper is organized as following. We introduce the background and significance of the research and the concepts of RDF and in Section 1 and describe the related work on RDF data storage and querying in Section 2, including RDF, SPARQL, Hadoop and HBase. We outline the relational database

storage model and distributed storage model in Section 3. We introduce the whole idea of SPARQL structured query and analyze the keyword query scheme in detail and on the basis of distributed RDF data storage model. We give the implementation process of the breadth-first path search algorithm based on the keyword query in Section 4. Section 5 verifies the correctness and efficiency of the above scheme by experiments. In Section 6, we draw our conclusions.

Our contributions are as follows:

• We design an RDF data storage model based on a distributed system, which can be extended by adding cluster nodes.

• We propose a breadth-first search based on keyword query algorithm on the basis of RDF storage and query framework of HBase, which realizes parallel queries efficiently.

• We test the LUBM datasets. By comparing the sample size of triples and query response time, it is shown that our query algorithm is more efficient than H2RDF in processing huge datasets queries.

2 Related Work

How to store and query RDF data is the focus of many researchers when designing RDF data management systems. Currently, existing RDF data storage methods mainly include RDF data storage based on memory, relational database, object-oriented database, and distributed cluster [5]. The main methods to query RDF data include local query, RDF graph query based on parallel platform and index query. Research demonstrates that the efficiency of storage and query of relational databases in processing massive data is lower than that of distributed databases [6]. Therefore, more and more researchers begin to use the massive storage and parallel computing of distributed clusters to query large-scale RDF datasets. For example, Nikolaos Papaiiliou proposed H2RDF distributed system based on HBase and MapReduce in 2012 [7]. At the same time, some mature RDF management systems have been developed, such as Jena [8], Sesame [9], RDF-3X [10–12], etc. Query method based on a distributed system has gradually become the mainstream method of large-scale RDF data queries.

2.1 RDF and SPARQL

The resource description framework is an infrastructure framework specified by W3C (World Wide Web Consortium) [13] for describing structured data and their interrelationships. It is mainly composed of syntax specification and expression statements. The core data model consists of three object types: Resource, attribute, and statement. RDF datasets consist of resources identified by a unique URL and intended to be readable by a computer. As is shown in Fig. 1, the RDF document can be used to describe "https://www.runoob.com//rdf". The structure of RDF triples <resource-relational-resource> can describe complex entity flexibly and easily and provide a framework and a common classification and query method for different metadata elements. RDF can designate practice tables for Web events, information about Web pages, and content for search engines. At present, research on RDF data management mainly includes two aspects: How to store RDF data effectively and how to query RDF data efficiently. We implement SPARQL and other query statements based on the RDF data storage model and further propose a breadth-first path search algorithm based on keyword queries.

SPARQL (Simple Protocol and RDF Query Language) is a standard query language published by W3C for querying RDF data [14]. Its basic format and syntax structure are similar to SQL. Most RDF data management systems currently support SPARQL queries. Most forms of SPARQL queries contain a number of triple patterns called a basic graph pattern [15]. Triple Pattern is the most basic matching unit, and its structure is similar to RDF Triple storage. The SPARQL query is basically divided into three steps. The first step is constructing the basic graph pattern. The second step is matching to the subgraph that satisfies the requirements of the graph pattern. Finally, the result is bound to the corresponding variable. SPARQL also provides four different forms of queries: SELECT, ASK, DESCRIBE, and CONSTRUCT. At present, implementing the SPARQL query of massive RDF data on a distributed platform is the main research direction. A number of SPARQL based extension tools such as C-SPARQL [16], BimSPARQL

[17] emerged. In this paper, we query the RDF data with SPARQL statements and test it with LUBM, a standard test data set, on a distributed Hadoop system.

```
<?xml version="1.0"?>

<RDF>
 <Description about="https://www.runoob.com//rdf">
 <author>Jan Egil Refsnes</author>
 <homepage>https://www.runoob.com/</homepage>
 </Description>
</RDF>
```

Figure 1: An example of an RDF document

2.2 Hadoop and HBase

Hadoop is an open-source distributed computing project that originated in Google's clustering system. Because of Hadoop's low cost and high fault tolerance, researchers often develop and run applications for processing massive data on distributed platforms. For example, Ding et al. [18–19] mainly use a distributed system to realize large-scale data queries. The most important design of the Hadoop framework is a distributed file system HDFS and distributed computing framework MapReduce. At present, the RDF data query based on the distributed Hadoop platform needs MapReduce for batch processing and parallel computation. MapReduce divides job processing into four stages: sharding, Map, sorting and Reduce. Fig. 2 shows the execution flow diagram of MapReduce. In recent years, a lot of research work using the MapReduce framework for the RDF data query. For example, [20–21], these data query systems have flexibility and scalability, but it is difficult to provide interactive queries because the MapReduce framework is limited to the fixed format of input data (key, value). In order to solve the above problems, this paper focuses on RDF storage and RDF query, analyzes and compares storage container selection in the storage system, and proposes a path algorithm based on BFS in query retrieval.

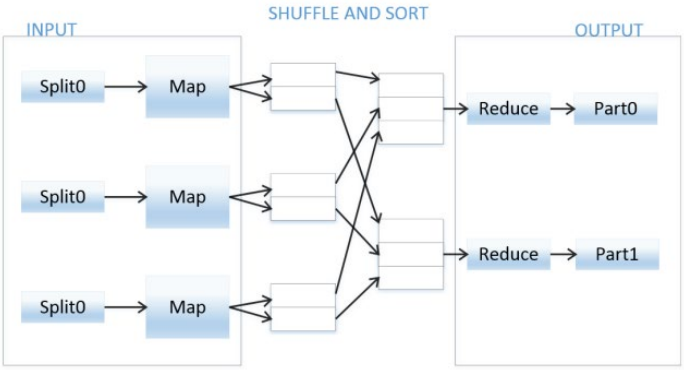

Figure 2: Execution flow diagram of MapReduce

HBase is a distributed database based on the Google big data model [22], which can be used to process massive structured or unstructured data and provide random and real-time read-write access to large-scale data [23]. Compared with the traditional relational database, HBase determines value by row key, column name, and timestamp, so it does not need to query by row, but only needs to scan related columns when querying, so it greatly improves the query efficiency and query performance. But HBase abandons atomicity, Consistency, Isolation, Durability and other features of traditional relational databases. Fig. 3 shows a storage schema instance of the HBase table. Myung et al. [24] store the form of

RDF data preserving triples directly in HDFS and propose greedy selection and two connection selection strategies to increase the efficiency of MapReduce. This approach is inefficient because there is no index. Therefore, in this paper, we design an RDF data storage model with HBase technology. According to the ontology described by OWL, create two index tables for each class to store the class and property information in the ontology. In addition, the problem of null and multi-valued attributes in the traditional relational attribute table storage is well solved by utilizing the characteristics of column storage and sparsity in the HBase.

```
Row Key: KeyID {
 Column Family 1 {
     Column 1:  T1 Value1
 }
 Column Family 2 {
     Column2:   T2 Value2     T3 Value3
     Column3:   T4 Value4
 }
}
```

Figure 3: A storage schema instance of the HBase table

3 Storing RDF

Recently, the RDF data storage model mainly includes relational database storage and distributed data storage. This section compares the two storage methods. Owing to the poor scalability of database storage, we adopt a distributed clustering approach to store RDF data in the experiments in Section 5.

3.1 Relational Database Storage

Relational database to store RDF data is a common tool of storage, widely used in various kinds of system programming [25]. Relational database storage has the characteristics of simple operation, easy to understand and complete data structure, but poor scalability. It includes four types of relational storage: horizontal, vertical, decomposition, and hybrid. Current relational RDF stores are difficult to meet the need for efficient management of large-scale RDF data and are unable to efficiently handle complex queries. The inherent mismatch between relational and RDF models further impedes the development of storage, so more and more scholars begin to use parallel or distributed storage management.

3.2 Distributed Storage

Due to the increasing amount of RDF data, the traditional node storage method is no longer suitable for massive data. Therefore, researchers begin to study storage methods based on distributed clusters. In 2010, Sun J proposed a distributed storage method using index tables [26]. In 2013, Papailiou N proposed an H2RDF storage scheme and specified the query order of triple pattern according to the greedy strategy [27]. The distributed storage methods can effectively improve the speed of loading data and the efficiency of reasoning query on the basis of ensuring the correctness of data storage. The storage process of the distributed database based on HBase is divided into five stages: uploading RDF data set to HDFS, pre-processing and parsing the data set to get triples, encoding strings, loading RDF data to HBase, and finally designing a reasonable table structure to store RDF data. Fig. 4 shows an example of abstract RDF storage. HBase stores data by column. Therefore, distributed storage can improve the scalability of data management. It is the most common method to deal with large-scale data sets.

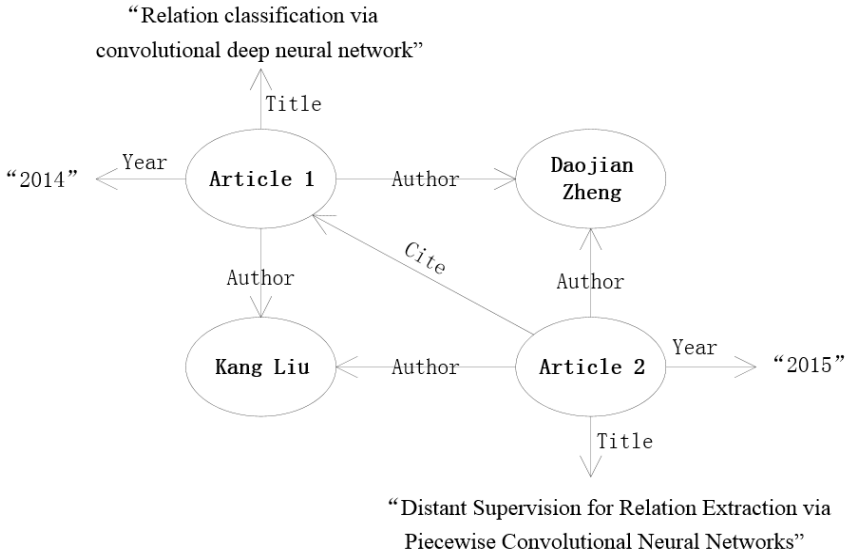

Figure 4: RDF storage by example

4 Retrieving RDF

4.1 SPARQL Query

SPARQL [28] is a structured language for querying RDF data. At present, a large number of researchers use SPARQL to conduct structured queries in a distributed environment. This query is similar to SELECT statements in SQL. Fig. 5 is a simple example of the SPARQL query. The SPARQL query statements contain the triple matching pattern, and the subgraph matching pattern corresponding to the triple matching pattern is shown in Fig. 6. In addition, to support for SELECT queries, SPARQL includes keywords such as DISTINCT, LIMIT, and ORDER BY, which further improves query efficiency. SPARQL query methods mainly include triple pattern query, basic graph pattern query, and complex query. A common SPARQL query consists of multiple triple queries, each of which takes the form < subject-predicate-object >. The query pattern of RDF data is mainly based on the basic graph pattern. For example, in 2009, Zhu [29] demonstrated that the graph structure storage of RDF data has the advantage of avoiding reconstruction and direct mapping. However, because the graph database does not provide the SPARQL query interface, researchers need to perform complex language conversion when searching, which limits the development of SPARQL language to some extent.

```
PREFIX ontoweb:<http://www.ontoweb.cn>
SELECT ?id ?age
WHERE
{
    ?student ontoweb:name  'Li Lei' .
    ?student ontoweb:id ?id .
    ?student ontoweb:age ?age .
}
```

Figure 5: An example of the SPARQL query

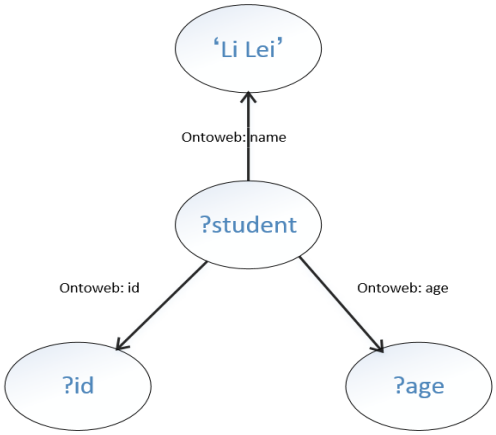

Figure 6: A subgraph matching pattern

4.2 Keyword Query

Suppose G = <r1, (<S1,P1,Q1>,...,<Sx,Px,Qx>)>, ... ,<rn,(<S1,P1,Q1>, ... , <Sy,Py,Qy>)>}.

Algorithm 1: BFSClassBasedPath(Graph $G1$)
Input: Enter the keywords for each class ClassWords = $\{cw_1,cw_2,...,cw_n\}$,
graph relation G_r = {Class, Relation}.
Output: SubGraph G
Initialize $Start[n]$, $End[n]$ //Path that starts or ends with this class
$G \leftarrow$ InitializeGraph($G1$)
While $c_i \in$ Class
$Start[c_i] \leftarrow$ BFS(G_r,c_i)
$End[c_{ie}] \leftarrow$ Reverse($Start[c_i]$) //c_{ie} represents the last node of the path
end while
$n_{cw1} \leftarrow$ SetUnion(Start[cw_1],End[cw_1]) //SetUnion(a,b) means union
for $n_i \in n_{cw1}$ do
for $n_{cwi} \leftarrow$ SetUnion($Start[cw_i]$,$End[cw_i]$) and $i=1...n$ do
if $n_i \in n_{cwi}$ then
$G \leftarrow (n_i, p_{cwi})$
end if
end for
end for
return G.

The query efficiency of retrieval methods based on the RDF graph is generally low, so the query method based on keywords has become a hot research topic in the field of graph query. At present, there are two main algorithms for querying RDF graphs: one is to query each subgraph through graph partitioning strategy, and the other is to map a key summary graph in the RDF graph. The query results of the RDF graph are obtained by path query on a key summary graph. Keyword query on RDF datasets is divided into two steps: Firstly, the mapping relationship between the keyword and the RDF graph structure is established; secondly, the breadth-first search is applied to RDF elements in the mapping relationship. The result is a query subgraph that contains all the elements that match the keyword requirements. On the basis of the structured query, this paper analyzes the keyword query method, makes

a further improvement, and proposes a breadth-first path search algorithm based on keyword queries. The query algorithm is mainly divided into two steps. First, the path at the beginning of category Ci will be stored in the Start[n] array, and the path at the end of category Ci will be stored at the End[n], and all paths of the entity class will be initialized. Second, the breadth-first path search is carried out on the input path. Finally, the mapping of the keyword query to the structured query is realized. Algorithm 1 shows the pseudo-code of the breadth-first search based on keywords. A large number of experiments show that the coverage is wider, the query efficiency is higher by using the keyword query method. Our BFS algorithm is based on keywords as shown in Algorithm 1. The principle of breadth first algorithm is first in, first out. After each class Ci is queued, it needs to access all its adjacent classes. The time complexity is O(n). The time complexity of breadth first module is O(n+e). The time complexity of union module is O(n2). The complexity of our BFS algorithm is O(n2+n+e), that is O(n2).

5 Experiments

In this section, we conduct the storage model and retrieval query algorithm presented above on the datasets. This experiment designs a distributed cluster environment. The simulation environment is configured with Intel(R) Core(TM) i7 2.60 GHz CPU, 8GB of memory and 500 GB of hard disk space. The development environment is Java SE 1.8, Hadoop version 2.7.3 and HBase version 1.4.9. The experimental data set is the LUBM standard test set. We verify the query algorithm for different query methods in LUBM, select the running time as the index, analyze and compare the change of query performance of keyword query algorithm before and after optimization, and compare it with traditional algorithm. The experimental data set is the LUBM standard test set. We use query methods in LUBM to verify our new scheme, select the running time as the evaluation index. We also analyze and compare the change of query performance of the keyword query before and after majorization.

5.1 Datasets

This experiment uses the LUBM (Lehigh University Benchmark) [30] test set. LUBM provides both data sets of different sizes and 14 standard query statements. Tab. 1 is a characteristic description of the queries.

Table 1: Characteristics of the LUBM queries

Query	Volume	Complexity	Selectivity
1,3,4,10,11, 13	Low	Low	High
2	High	High	-----
5	Low	High	Low
9	High	High	-----
6,14	High	Low	High
7,8,12	High	Low	-----

Table 2: Experimental data sets

Data Set	Number of Universities	Number of Triples	Number of Entities
D(1)	1	102722	20K
D(5)	5	735845	130K
D(10)	10	1355893	260K
D(50)	50	7032612	14M
D(100)	100	13543642	28M
D(200)	200	26725428	43M

In the experiment, we use UBA's data generator to generate six different sizes of RDF data sets. Di represents a data set containing (i) schools. The selected data sets correspond to the number of 1, 5, 10, 50, 100 and 200 schools respectively. Each test set is shown in Tab. 2 The generated data is stored in HDFS and then loaded into the class's storage table with a MapReduce. Since some of the query statements in LUBM are designed to test the reasoning ability of the engine, we select typical statements for testing.

5.2 Evaluation

We conduct LUBM's three typical query statements Q1, Q5, and Q6 on six datasets, and compare them with the traditional H2RDF system. The average response time is the average execution time of each query running 5 times on different data sets. The response time of the LUBM query statement is shown in Tab. 3, in milliseconds. Experimental results show that under the Q1 simple query, compared with the traditional H2RDF system, this scheme has no obvious advantage in response time, and the response time varies little when the data sets multiply. For Q5, which is less selective but has a complex reasoning relationship, the response time increases faster as the data sets grow. For Q6 with high selectivity, the response time is the slowest and the growth is the fastest. Compared with the H2RDF system, we prove that both the proposed distributed storage model and the breadth-first path search based on keywords are efficient and feasible.

Table 3: Performance comparison for queries

Query Runtimes (in msec)						
	Q1		*Q5*		*Q6*	
	H_2RDF	BFS-RDF	H_2RDF	BFS-RDF	H_2RDF	BFS-RDF
D(1)	7044	7519	168078	173589	173456	188650
D(5)	8702	9203	176640	180727	178494	218231
D(10)	9678	10230	207684	230387	224845	370044
D(50)	15834	17092	280461	310351	402567	482323
D(100)	26481	28515	496487	459879	673578	594602
D(200)	32541	34288	934856	892456	1130894	995446

6 Conclusion

Both Traditional data storage and structured query have a long execution time and slow response time in the distributed scenario, which cannot meet the requirements of querying the massive RDF Data. We analyze and compare the relational database storage with distributed one, and develop an improved distributed storage scheme based on HBase. Furthermore, we compare the SPARQL structured query and keyword query and propose a breadth-first path search strategy. The experiment indicates the feasibility and efficiency of this scheme by comparing average response time and query efficiency in the LUBM dataset. Our future work will investigate the keyword query method, the mapping relationship between keywords and RDF elements. The conversion rate and accuracy of keywords still need to be further improved.

Funding Statement: This work is supported in part by National Natural Science Foundation of China (61728204), Innovation Funding (NJ20160028, NT2018027, NT2018028, NS2018057), Aeronautical Science Foundation of China (2016551500), State Key Laboratory for smart grid protection and operation control Foundation, Association of Chinese Graduate Education (ACGE).

Conflicts of Interest: The authors declared that there is no conflict of interest for this paper.

References

[1] M. Ko and W. Choi, "A distributional inference for cross-lingual undefined entities linking," *Converg,* vol. 4, no. 2, pp. 23–28, 2013.

[2] S. Harris and N. Gibbins, "3-store: Efficient bulk RDF storage," Island: CEUR-W S Press, vol. 2, no. 3, pp. 1–15, 2003.

[3] T. Neumann and G. Weikum, "The RDF-3X engine for scalable management of RDF data," *The VLDB Journal*, vol. 19, no. 1, pp. 91–113, 2010.

[4] T. Neumann and G. Weikum. "x-rdf-3x: Fast querying, high update rates, and consistency for RDF databases," in *Proc. of the VLDB Endowment*, vol. 3, no. 1, pp. 256–263, 2010.

[5] D. Janke, "Study on data placement strategies in distributed RDF stores," University of Koblenz and Landau, Germany, 2020.

[6] C. Franke, S. Morin and A. Chebotko, "Distributed semantic web data management in HBase and MySQL cluster," in *2011 IEEE Int. Conf. on Cloud Computing (CLOUD)*, pp. 105–112.

[7] N. Papailiou, I. Konstantinou and D. Tsoumakos, "H2RDF: Adaptive query processing on RDF data in the cloud," *International Conference on World Wide Web*, ACM, pp. 397–400, 2012.S

[8] J. J. Carroll, I. Dickinson and C. Dollin, "Jena: Implementing the semantic web recommendations," in *Pro. of the 13th Int. World Wide Web Conf. on Alternate Track Papers & Posters*, ACM, pp. 74–83, 2004.

[9] Aduna, Sesame, 2003. [Online]. Available: http://www.openrdf.org.html.

[10] T. Neumann and G Weikum, "RDF-3X: A RISC-style engine for RDF," *Proceedings of the VLDB Endowment*, vol. 1, no. 1, pp. 647–659, 2008.

[11] T. Neumann and G. Weikum. "The RDF-3X engine for scalable management of RDF data," *The VLDB Journal*, vol. 19, no. 1, pp. 91–113, 2010.

[12] T. Neumann and G. Weikum, "X-RDF-3X: fast querying, high update rates, and consistency for RDF databases," in *Proc. of the VLDB Endowment*, vol. 3, no. 1–2, pp. 256–263, 2010.

[13] G. Klyne, J. J. Carroll and B. McBride, "Resource description framework (RDF): Concepts and abstract syntax," *W3C Recommendation*, 2004.

[14] S. Harris and A. Seaborne, "SPARQL 1.1 Query language," *W3C Recommendation*, 2013.

[15] E. Prudhommeaux and A. Seaborne, "SPARQL Query Language for RDF," *W3C Working Draft 4*, 2006.

[16] A. F. Dia, Z. Kazi-Aoul and A. Boly, "C-SPARQL Extension for Sampling RDF Graphs Streams," *Advances in Knowledge Discovery and Management*, 2018.

[17] Z. Chi, B. Jakob and D. V. Bauke, "BimSPARQL: Domain-specific functional SPARQL extensions for querying RDF building data," *Semantic Web*, pp. 1–27, 2018.

[18] Z. H. Ding, M. Y. Jiang and K. Abraham, "Port-Based Reliability Computing for Service Composition," *IEEE Transactions on Services Computing*, vol. 5, no. 3, pp. 422–436, 2012.

[19] Z. B. Zheng, X. M. Wu and Y. L. Zhang, "QoS Ranking Prediction for Cloud Services," *IEEE Transactions on Parallel and Distributed Systems*, vol. 24, no. 6, pp. 1213–1222, 2013.

[20] A. Schäftzle, M. Przyjaciel-Zablocki, T. Hornung and G. Lausen, "PigSPARQL: A SPARQL query processing baseline for big data," in *Proc. of the ISWC 2013 Posters & Demonstrations Track*, pp. 241–244, 2013.

[21] X. Zhang, L. Chen, Y. Tong and M. Wang, "EAGRE: towards scalable I/O efficient SPARQL query evaluation on the cloud," in *2013 IEEE 29th Int. Conf. on Data Engineering (ICDE)*, pp. 565–576, 2013.

[22] Z. Li, "Hadoop-Based Big Data Computing Technologies," *E-Science Technology & Application*, 2012.

[23] M. Somnath and S. Alberto, "Chapter One-Fast execution of RDF queries using Apache Hadoop," *Advances in Computers*, vol. 119, pp. 1–33, 2020.

[24] J. Myung, J. Yeon and S. G. Lee. "SPARQL basic graph pattern processing with iterative MapReduce," in *Pro. of the 2010 Workshop on Massive Data Analytics on the Cloud*, no. 6, pp. 1–6, 2010.

[25] H. Mahmoudinasab and S. Sakr, "AdaptRDF: Adaptive storage management for RDF databases," *International Journal of Web Information Systems*, vol. 8, no. 2, pp. 234–250, 2012.

[26] D. J. Kim, J. H. Shin and K. S. Hong, "Scalable RDF store based on HBase and MapReduce," in *Int. Conf. on Advanced Computer Theory and Engineering*, vol. V1, pp. 633–636, 2010.

[27] N. Papailiou, I. Konstantinou and D. Tsoumakos, "H2RDF: High-performance distributed joins over large-scale RDF graphs," in *IEEE Int. Conf. on Big Data*, pp. 255–263, 2013.

[28] J. Perez, M. Arenas and C. Gutierrez. "Semantics and complexity of SPARQL," *ACM Transactions on Database Systems*, vol. 34, no. 3, pp. 1–16, 2009.

[29] K. Zhu, X. Wang and Y. Liu. "A new query expansion method based on query logs mining," *International Journal on Asian Language Processing*, 2009.

[30] Y. Guo, Z. Pan and J. Heflin. "LUBM: A benchmark for OWL knowledge base systems," *Web Semantics Science Services & Agents on the World Wide Web*, vol. 3, no. 2-3, pp. 158–182, 2005.