ISBN 978-981-14-1684-2
Proceedings of 2019 the 9th International Workshop on Computer Science and Engineering
(WCSE 2019 SUMMER)
Hong Kong, 15-17 June, 2019, pp. 638-642
doi: 10.18178/wcse.2019.06.094

A Cloud-based Storage and Retrieval Solution for RDF Data

Sun Yuxiang and Yongju Lee [+]

School of Computer Science and Engineering
Kyungpook National University
Daegu 41566, Korea

Abstract. In recent years, RDF (Resource Description Framework) has been widely recognized as a standard data storage format. There is a real issue that how to store and retrieve RDF data efficiently, because it is the foundation of all semantic-based application development. In this paper, we propose a novel storage and retrieval solution based on cloud and R*-tree. Different to existing approaches, we use cloud-based storage approach to compress RDF data, meanwhile R*-tree is adapted to retrieve compressed data. The advantage is that it not only reduces the local storage pressure but also improves the retrieval performance compare to existing approaches. Because we separate the storage and retrieval and adapt dictionary-based compression approaches, in terms of security and flexibility, our solution is better than existing approaches.

Keywords: Cloud-based Storage, RDF Compression, Linked Data, Indexed Approach, Join Algorithm, R*-tree, Sorted Lists

1. Introduction

RDF is a logical data model for Linked Data and a graph-based data format. It consists of triples and each triple includes subject, predicate and object. It offers a very useful data format to describe relationships between data items. As one of many storage languages, it has several advantages. Due to special structural features, it is easy to link with different datasets. That is one of the main reasons that it is recognized by the semantic research communities. Because of the flexibility of structure, the query of data is not limited to single dataset or table. Compared to traditional data search it provides better performance in accuracy and search range. It means that we need to design a novel storage approach to replace traditional relational database. From the storage structure, it is mainly divided into centralized approaches, distributed approaches and index approaches.

The centralized approach collects data from several datasets or government agencies and stores them in a local storage registry. The using of the registry greatly improve query performance. However, there are also several shortages. First, as data volumes grow, because all data are stored in local space it will increase local storage pressure. Second, we cannot guarantee that local information is up-to-date, because we cannot confirm whether the endpoint data has updated, this point is fatal for application development.

The distributed approach queries RDF data by using multiple SPARQL endpoints. There are two advantages. First, the decentralized access structure can greatly reduce the local storage pressure. Second, all queried data is up-to-date. These two points perfectly address the shortages of centralized approaches. Meanwhile, since all endpoints must be accessed via network, we cannot predict in terms of security and network delay. In addition, we cannot guarantee that all publishers provide SPARQL endpoints are reliable.

The indexed approach can well solve issues of two approaches that mentioned above. Because of the special structure of RDF data, we can easily convert them into three-dimensional points by hash function.

[+] Corresponding author. Tel.: +82-53-950-7285; fax: +82-53-957-4846.
E-mail address: yongju@knu.ac.kr

Therefore, queries based on RDF data can be viewed as point search in three-dimensional space. The indexed approach supports efficient join query processing by adapting auxiliary indexes based on MBB (minimum bounding box) approximation to pruning unnecessary data. We will describe it in Section 3.

In order to reduce local storage pressure, we compare different compression approaches. In this study, we adapt the dictionary-based cloud compression approach. This approach not only achieves efficient data compression, but also separates real values and logical relations of RDF data. Real values are saved to the cloud using the dictionary-based compression approach. Logical relations are saved to personal servers using an adjacency-lists compression approach. This storage structure greatly improves security. The detail structure of the cloud will be described in Section 3.

2. Related Work

2.1. Storage approach

We collate and classify some research results about the storage of RDF data in the last decade. Like centralized approaches represented by Yars [1] and RDF-3X [2]. Distributed approaches represented by DARQ [3] and SemWIQ [4]. Indexed approaches represented by MIDAS-RDF [5] and QTree [6].

The Yars approach used six B+ tree indices to store RDF data, it not only stores triples but also the context information about the origin of data. It adapts inverted index to speed up keyword queries. The six B+ trees covers all access patterns; therefore, it also supports single lookup and complex join query. Theoretically, it can achieve better query performance, however, it performs worse on storage performance. Although the Kowari [7] uses a hybrid structure of AVL and B trees replace B+ trees for multiple indexing, the basis storage structure has not changed.

The DARQ is a query engine for federated SPARQL queries. It supports transparent query access to multiple, distributed endpoints. The DARQ mainly includes four stages parsing, query planning, optimization and query execution. Despite of decomposing a query into many sub-queries to improve query performance, we still cannot solve the issue of security and delay.

The MIDAS-RDF is a distributed index structure based on k-d tree [8]. It can process various pattern queries in hops logarithmic to the number of peers. It achieves better performance by using labeling schemes replaced traditional iterative procedures.

2.2. Compression approaches

In [9], the author summarizes three basic compression approaches for RDF data. Table 1 shows different compression approaches in detail.

Table 1: Basic Compression Approaches

Approaches	Description
Direct Compression	Compression with gzip, bzip2, ppmdi
Adjacency Lists	Convert RDF data to adjacency lists
Dictionary + Triples	Split data into dictionary and triples

The direct compression uses three well-known techniques such as gzip, bzip2 and ppmdi. In theory, we do not change the structure of data, only to improve compression efficiency by using different algorithm. The advantage is simple, but we cannot obtain satisfactory compression performance. The adjacency lists compression focuses on the repeatability of data. For example, it can be descripted like $S \rightarrow [(P_1, ObjList_1), \cdots (P_K, ObjList_K)]$. The advantage is that we can reduce the appearance of repetitive data. However, compared to direct compression, we need cost more time to change the structure of data. The dictionary + triples split the data into dictionary of elements and the triples substituting for each element. Triples can be represented by adjacency lists based on hash values. The advantage is that we separate the real values and logical relation from RDF data, it provides theoretical support for the implementation of cloud-based compression structure. In next session, the cloud-based compression structure will be described in detail.

3. Storage & Retrieval Solution

3.1. Cloud-based storage platform

Figure 1 shows the cloud-based compression platform. The abbreviation of DPL is Data Parsing Layer, the primary function is to parse different types of RDF data from users. If the user has registered in registry database, then the parsed triples are stored into dictionary database. This process is called DPSS (Data Parsing and Storage Service). Another important service is DFRS (Data Forwarding and Retrieval Service). While the cloud platform received query requests from users, the triple to be queried will be parsed in DPL and converted to hash values in dictionary database under the premise of already registered. Finally, the query request will be forwarded to personal server by DFL (Data Forwarding Layer). We designed a cache structure to cache data between registry database and dictionary database. The goal is to reduce delay in the cloud, it is a memory-based key-value database. We will compare the compression performance in the experiment evaluation and analysis session.

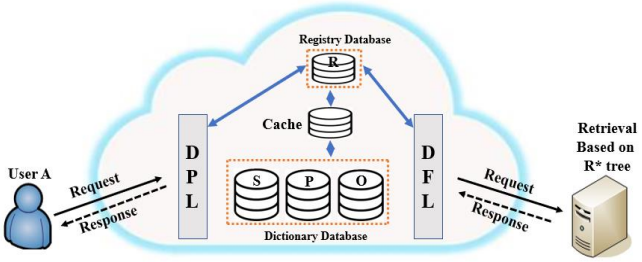

Fig. 1: Cloud-based Compression Platform.

3.2. Retrieval based on R*-tree

We described the structure and function of the cloud compression platform in Session 3.1. In this session, we will address the retrieval structure based on R*-tree with sorted lists (see Figure 2). Different to our previous study [10], we replace linked lists with sorted lists, meanwhile we adopt binary search to retrieve data with sorted lists. The retrieved data is stored into memory-based database. Therefore, the join query processing completely based on memory. For example, when we want to query (s, p, ?o) (?s,p,?o), first we retrieve from R*-tree that match (s, p, ?o) and (?s,p,?o) and obtain target sorted lists r3 and r5, then we obtain triples that satisfy our conditions in r3 and r5 by binary search algorithm. Finally, the results will be inserted into Set function and Map function. The result is obtained by retrieving whether keys in the Map function coincide with the values in the Set function.

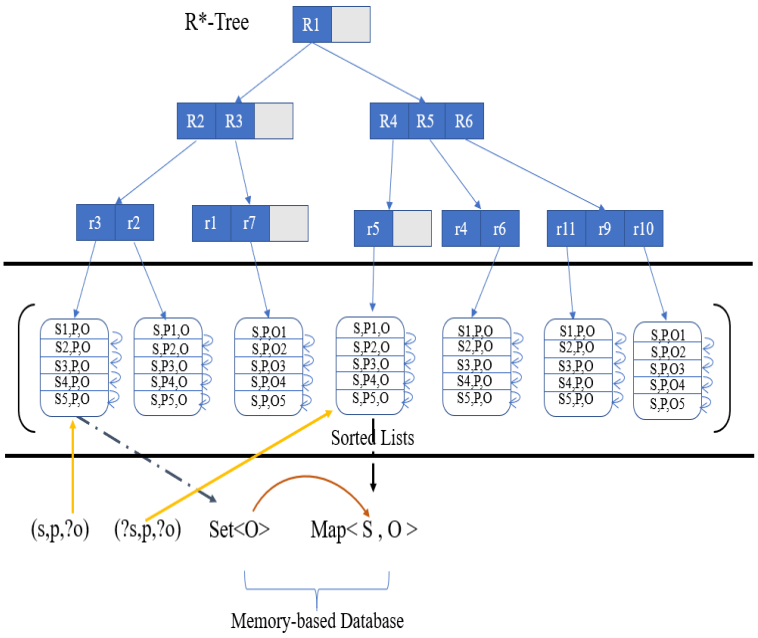

Fig. 2: Retrieval Structure Based on R*-tree

```
QT: query triple patterns
RS: Result Sets
BS: Binary Search
SL: Sorted Lists
RA: Array of results
SQM: Single Query Mode
DQM: Double Query Mode
FRS: Final Result Sets
If (QT in R*-Tree) {
   for (int i=0; I < QT.Lenght; i++) {
       SL = Sort (Retrieve (QT[i]));
       If (QT[i] == SQM) {
           RS = BS(SL) match QT[i];
           RA[i] = Set.add(RS);
       } else if (QT[i] == DQM) {
           RS = BS(SL) match QT[i];
           RA[i]= Map.put(RS);
       }
   }
}
#operation in memory-based database
For (int i=0; i < RA.Length; i+2) {
   FRS = RA[I] match RA[i+1];
}
```

Fig. 3: Join Algorithm

4. Join Algorithm

We briefly introduced join query in Session 3.2. In this session, for further understand the principles of join query, we adapt short code to describe it. There are three popular join algorithms such as Nested Loops join, Sort-Merge join and Hash join. In order to adapt the sorted lists retrieval approach, we made some modify in traditional Nested Loops join algorithms.

First, we confirm whether the query data meets R*-tree in line 9 of Figure 3. Second, we retrieve all data from R*-tree and sort them according to query modes. For example, if we want to query the object of triples, we should sort the hash value of subject or predicate. According to the number of queries, it is divided into SQM (Single Query Mode) and DQM (Double Query Mode), the mean of SQM is that we know any two terms of triples and want to query the third term. DQM is the same way. Finally, results will be store in RA. We can obtain results by Nested Loops operation. This step is done in memory-based database, because the performance of memory-based database has been certified in [11].

5. Performance Comparison

5.1. Experimental data

In order to verify the accuracy of join query, we randomly insert two sets of triples that meet join query condition into three datasets to compare their join query times by querying the same data. Our operation system is windows 10 professional version, memory is 8G and the processor frequency is 3.60GHz.

In order to ensure the reliability of the experimental results, we downloaded three different datasets. The DBpedia[1] dataset contains RDF information extracted from Wikipedia. The DrugBank[2] dataset contains information on drugs and drug targets. The LinkedGeoData[3] dataset is a large spatial knowledge base which has been derived from OpenStreetMap for the Semantic Web. Table 2 shows the detailed data information.

Table 2: Characteristics of Experimental Data

Datasets	Size (MB)	# of Triples	# of Subjects	# of Predicates	# of Objects
DBpedia	3.94	31,050	4,008	23	16,644
DrugBank	144	766,920	19,693	119	274,864
LinkedGeoData	327	2,207,295	552,541	1,320	1,017,242

5.2. Compression performance

In Table 3, Direct Compression, Adjacency Lists, Dictionary + Triples and cloud-based compression are referred as DC, AL, DT and CC. We described four compression approaches in previous session. We can know that the performance of cloud-based approach is the best. Because the dictionary cannot be compressed, the performance of DT is the worst. In spite of this, it provided the theoretical support for our cloud-based compression approach. We greatly reduce the local storage pressure by managing dictionaries in the cloud. We only need to store compressed triples in local storage space.

Table 3: Comparison with Compression Rate

Datasets	DC	AL	DT		CC
			Dictionary	Triples	
DBpedia	9.37%	7.49%	86.47%	4.29%	4.29%
DrugBank	7.72%	6.72%	23.64%	1.91%	1.91%
LinkedGeoData	6.26%	6.26%	45.83%	3.63%	3.63%

5.3. Query performance

As depicted in Figure 2, the join query is divided into two parts; the first part is to obtain results from R*-tree, the second part is to store results, and the join operation is performed by adopting the join algorithm in Figure 3. Figure 4 shows the overall query performance including the first part and second part. We can

[1] http://downloads.dbpedia.org/2016-04/

[2] https://www.drugbank.ca/releases/latest

[3] https://hobbitdata.informatik.uni-leipzig.de/LinkedGeoData/Downloads.linkedgeodata.org/releases

find that R*-tree obtains better performance than existing mainstream approaches. On the one hand, the spatial structure of R*-tree is very conducive to RDF data indexing; it benefits from the utilization of memory-based database.

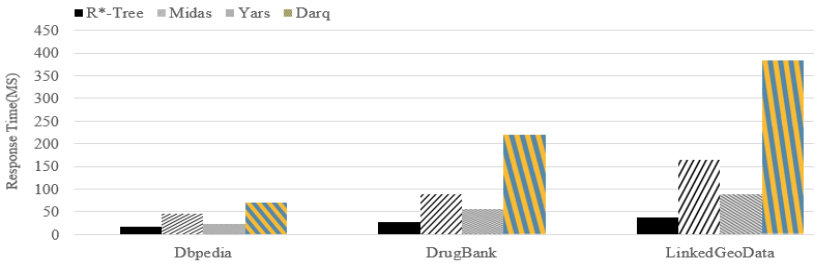

Fig. 4: Performance of Join Query

6. Conclusions

Storage and retrieval capabilities are the foundation of application development. We not only achieved efficient storage performance but also improved the security of data, even if the cloud platform is hacked. It is difficult to obtain complete and valuable information without acquiring the logical relationship between them. Because all processing is done via network, we must consider network delay. Nonetheless, with the development of 5G networks, this issue must be addressed. This paper demonstrates that the utilization of R*-tree obtains better retrieval performance than another approaches. In the future work, we will continue to improve the retrieval performance of R*-tree by trying different join algorithms.

7. Acknowledgments

This research was supported by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education (No. 2016R1D1B02008553).

8. References

[1] A. Harth, Decker S. Optimized index structures for querying RDF from the Web. *Proc. of the 3rd Latin American Web Congres*. 2005, pp. 71-81.

[2] T. Neumann, G.Weikum. RDF-3X: A RISC-style engine for RDF. *Proc. of the 34th International Conference on Very Large Data Bases*. 2008, pp. 647-659.

[3] B. Quilitz, U. Leser. Querying distributed RDF data sources with SPARQL. *Proc. of the 5th European Semantic Web Conference*. Vol. 5021, Lecture Notes in Computer Science. 2008, pp. 524-538.

[4] A. Langegger, W. Wob, M. Blochl. A semantic middleware for virtual data integration on the Web. *Proc. of the 5th European Semantic Web Conference*. Vol. 5021, Lecture Notes in Computer Science. 2008, pp. 493-507.

[5] G. Tsatsanifos, D. Sacharidis, T. Sellis. On enhancing scalability for distributed RDF/S stores. *Proc. of the 14th International Conference on Extending Database Technology*. Uppsala, Sweden. 2011, pp. 141-152.

[6] A. Harth, K. Hose, M. Karnstedt, A. Polleres, K. Satler, J. Umbrich. Data summaries for on-demand queries over Linked Data. *Proc. of the 19th International Conference on World Wide Web*. 2010. pp. 411-420.

[7] D. Wood, P. Gearon, T.Adams. Kowari: A Platform for semantic Web storage and analysis. *Proc. of the XTech 2005 Conference*

[8] J. L. Bentley. Multidimensional binary search trees used for associative searching. *Communications of the ACM*. 1975, **18** (9).

[9] J. Fernandez, C. Gutierrez, M. Martinez-prieto. RDF compression: Basic approaches. *Proc. of the WWW 2010*. 2010, pp. 1091-1092.

[10] Y. Lee, Y.Sun. Hybrid index structured on MBB approximation for Linked Data. *Proc. of the International Conference on Intelligent Computing and Applications*. 2018.

[11] W. Puangsaijai, S. Puntheeranurak. A comparative study of relational database and key-value database for Big Data applications, *Proc. of the 5th International Electrical Engineering Congress*. 2017.