Scalable and Efficient Self-Join Processing technique in RDF data

Awny Sayed 1 and Amal Almaqrashi 2

1 Faculty of Science, Minia University, Egypt
Ibri College of Applied Sciences, Sultanate of Oman

2 Ibri College of Applied Sciences, Sultanate of Oman

Abstract

Efficient management of RDF data plays an important role in successfully understanding and fast querying data. Although the current approaches of indexing in RDF Triples such as property tables and vertically partitioned solved many issues; however, they still suffer from the performance in the complex self-join queries and insert data in the same table. As an improvement in this paper, we propose an alternative solution to facilitate flexibility and efficiency in that queries and try to reach to the optimal solution to decrease the self-joins as much as possible, this solution based on the idea of "Recursive Mapping of Twin Tables". Our main goal of Recursive Mapping of Twin Tables (RMTT) approach is divided the main RDF Triple into two tables which have the same structure of RDF Triple and insert the RDF data recursively. Our experimental results compared the performance of join queries in vertically partitioned approach and the RMTT approach using very large RDF data, like DBLP and DBpedia datasets. Our experimental results with a number of complex submitted queries shows that our approach is highly scalable compared with RDF-3X approach and RMTT reduces the number of self-joins especially in complex queries 3-4 times than RDF-3X approach.

Keywords: Semantic Web, RDF, RDF Storing, RDF Indexing, Self-join, Query Processing.

1. Introduction

There are several initiatives to improve the situation and reducing the drawbacks of the current web. One of them is a Semantic Web, which is coined by the W3C founder Tim Berners-Lee in a Scientific American article that is describing the future of the Web [1]. The Semantic Web would give more structure and computer-understandable meaning as well as provide a common framework for data sharing across applications, enterprises, and communities. The core of the Semantic Web is built on the Resource Description Framework (RDF) data model [2] [3] [4]. An RDF store consists of a collection of statements, called triples, of the form (subject, predicate, and object) also known as (subject, property, and value), where subject and predicate are resource URIs and O is either a URI or a literal value.

RDF storage has witnessed numerous research initiatives in varied domains. Despite the best efforts, a scalable, efficient and fast index has eluded researcher’s attention. A typical RDF data-store consist of billions of triples (a triple comprise Subject, Predicate & Object) with extensive and wide range of self-dependencies among the subject and the object field values. These results in recursive self joins with an added cost to the query optimizer [1]. Besides self-joins, unions and null values also create far greater performance related issues. There exist broadly two ways to deal with these issues: either to re-design the RDF data-store from scratch using a new setup for representing the triples along with the modified query engine design or to explore faster and more efficient indexing strategies that provide impeccable query processing times irrespective of scalability. This paper focuses on the second aspect related to a new index design.

The rest of the paper is organized as follows. Section 2 introduces the research efforts approaches to store and query RDF data efficiently. Section 3 provides the details of our proposed solution (RMTT) for storing and querying RDF data. Finally, section 4 concludes the paper and provides some suggestions to improve the research direction on our methodology as well as reducing join problems.

2. Related Work

In this section, we discuss the state of storing RDF data in Triple table, with an extended look at the property table approach and vertically partitioned approach.

The RDF data stores in a single table which consists of three columns (S, P, O), and each triple has subject, predicate, and object respectively. This approach is called triples table approach [5]. The serious performance issue of this approach is all the triples stored in a single RDF table which requires expensive and complex self joins over the triples table as pointed out in [11, 12, and 13]. Thus, as
queries become more complex the execution and time increase. In addition, it is exceeding the memory size and congestion of the RDF data sets. Nevertheless, this approach has been implemented by systems like Oracle [14], 3store [5], Redland [20], RDFStore [21] and rdfDB [22]. The research community later introduces an alternative solution for improving the triples table and minimize the number of self-joins issues. An alternative methodology to the previous is the property table approach [6].

The property approach deformatized the RDF table that stored in a flattened format. Furthermore, it is classified into two types which are property class table and clustered property table. The clustered table contains clustered of properties that tend to be defined together. Whereas the property class table exploits the type property of subjects to cluster similar sets if subject together in the same table [7]. The most important advantage for representing the property tables is that they can reduce subject-subject self joins of the triples table. However, this approach may not fit well the RDF data because of unstructured data and missing properties. In an interpretation, not all properties will be defined for all subjects and that perhaps led to many NULLs value which increases the overhead in the memory space. Another problem with the property table is the abundance of multi-valued attributes found in RDF data which cause further complexity and with combine data from several tables the issue of improving the performance of self-joins queries maybe become poor. In summary, property tables are rarely used due to their complexity and inability to handle multi-valued attributes. However, this approach has been used by tools like Sesame [23], Jena2 [12], RDFSuite [24] and 4store [25].

Abadi et al. [7] proposed vertically partitioned approach as an alternative solution to the property table to speed up queries and minimize its limitations and using a fully decomposed storage model (DSM) [10]. In this approach, an RDF table is rewritten into \( n \) two-column tables, where \( n \) is the number of unique properties. Moreover, the first column is subject and the second is an object.

One of the primary benefits of vertical partitioning is the support for rapid subject-subject joins. This feature is achieved by sorting the tables via subject as we mentioned above each binary table has subject and object columns. The tables being sorted by subject, one has a way to use fast merge joins to reconstruct information about multiple properties for subsets of subjects.

The experiments in these papers [8] and [9] showed that the vertical partitioning approach also performs poorly for querying RDF data and slow insertion, because of the multi property tables. In a spite, the vertical partitioning approach supports multi valued attributes and heterogeneous records. In addition, it is eliminating the subjects that don’t define a particular property. Clearly, it reduces the NULLs value through that elimination. Over the past decade, a fair number of RDF storage systems has been developed and implement the previous approaches, including Sesame [23], YARS [26], Kowari system [27], Virtuoso [28], RDF-3X [29], Hexastore [13], BitMat [30] etc. The following gives a simple overview on some of the previously mentioned systems. YARS system [26] (Yet Another RDF Store) combines methods from Information Retrieval and Databases to allow for better query answering performance over RDF data. It stores RDF data persistently by using six B+ tree indices. It not only stores the subject, the predicate and the object, but also the context information about the origin of the data. To speed up keyword queries, the lexicon keeps an inverted index on string literals to allow fast full-text searches. In each B+ tree, the key is a concatenation of the subject, predicate, object and context. The six indices constructed cover all the possible access patterns of quads in the form \((s, p, o, c)\) where \(c\) is the context of the triple \((s, p, o)\). This representation allows fast retrieval of all triple access patterns.

RDF-3X [29] is an RDF storage system with advanced indexes and query optimization that eliminates the need of physical database design by the use of exhaustive indexes for all permutations of subject-property-object triples. Neumann et al. use a potentially huge triples table, with their own storage implementation underneath (as opposed to using an RDBMS). They overcome the problem of expensive self-joins by creating a suitable set of indexes. All the triples are stored in a compressed clustered B+ tree.

Hexastore [13] takes also a similar approach to YARS. The framework is based on the idea of main-memory indexing of RDF data in a multiple-index framework. The RDF data is indexed in six possible ways, one for each possible ordering of the three RDF elements by individual columns. The representation is based on any order of significance of RDF resources and properties and can be seen as a combination of vertical partitioning [7] and multiple indexing approaches [31].

As a consequence, the current approaches of RDF stores that we discuss above triple table, property table approach and vertically partitioned approach, they still suffer from weak data locality which affects the storage and query performance costs. As an improvement of this paper which led to a stronger data locality, we proposed the Recursive Mapping of Twin Tables (RMTT) approach.
3. RMTT Approach

Recursive Mapping of Twin Tables (RMTT) is an approach to partition the RDF Triple which includes <subject, predicate, and object> triples into two tables that have the same structure of RDF Triple. In addition, this algorithm depends on the following idea whereas the subjects of the second table are subset of the objects in the first table; however, the vice versa isn’t. In contrast, the subjects in the first table are not subset of the objects in the second table and vice versa. So, $s_2 \subseteq o_1$ and $o_2 \not\subseteq s_1$ whereas (s1 and o1 are subject and object of the first table) (s2 and o2 are subject and object of the second table). In contrast, s1 $\not\subseteq$ o2 and o2 $\not\subseteq$ s1. Moreover, the insertion of data depends on two direction (s1 $\rightarrow$ o2) AND (s2 $\rightarrow$ o1).

3.1 Pseudocode

The following figure explains the pseudo code that used to create our RMTT approach.

<table>
<thead>
<tr>
<th>Input:</th>
<th>The main RDF table (T) which has three columns (S,P,O)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Output:</td>
<td>The new twin table based on the main RDF table</td>
</tr>
</tbody>
</table>

- **Step 1:** Identify/create/recreate two empty tables T1 and T2 with the same structure which have three columns and dynamic rows.
- **Step 2:** Define four empty HashSet to store the object’s values and another to store the subject’s values of the first table, and the same for the second table (s1HashSet, s2HashSet, o1HashSet, o2HashSet).
- **Step 3:** Fetch the first row from the main RDF table.
- **Step 4:**
  - **Step 4.1:** If the main RDF table has next row
    - Go to Step 6...
  - ELSE
    - Go to Step 6...

- IF:
  - The object’s set contains the next subject’s value OR the subject’s set contains the next object’s value.
    - Go to Step 5...
  ELSE
    - Insert the row in the first table and store the current object’s value in the set also store the current subject’s value in the second set of the first table.

- **Step 4.1:**
  - **Step 4.1.1:** If the main RDF table has next row
    - Go to Step 5...
  - ELSE
    - Go to Step 6...

- **Step 5:**
  - **Step 5.1:** If the main RDF table has next row
    - Go to Step 5...
  - ELSE
    - Go to Step 6...

- ELSE:
  - If the subject’s set equals to zero, insert the row in the second table and store the current subject’s value in the set also store the current object’s value in the set also store the subject’s value in the second set of the second table.

- **Step 5.1:**
  - **Step 5.1.1:** If the main RDF table has next row
    - Go to Step 5...
  - ELSE
    - Go to Step 6...

- ELSE:
  - If the object’s set contains the next subject’s value OR the subject’s set contains the next object’s value.
    - Go to Step 4...

- **Step 6:** STOP/EDIT (if there is no data)

Figure 1: RMTT algorithm

This pseudocode describes how the algorithm will be going and divides the main RDF Triple into two table’s recursively. Firstly, it defines two empty tables which have the same structure of the main RDF table. In addition, identify two empty sets for the first table to store the subject’s values and object’s values and the same for the second table. Then, fetch the first triple from the main and apply the RMTT algorithm. Note that, in this approach, inserting data done through two conditions which are the next subject’s value in the first table is exist in all the previous object values or not, or the second condition is the next object’s value in the first table is exist in all the previous subject values or not. If it already exists go to the next table and insert data and follow the same steps above. However, insert data in the next triple in the same table and repeat the previous steps until the main triple has no data. The following DBLP example applies the RMTT algorithm which we proposed and gives you the final structure of the twin table as it shows in Figure 4.

3.2 Example

An example describes a magazine which publishes two articles. Each article has a title, year of publishing and it is written by one or more authors. An author has a name and some personal details which led to another branch such as affiliation or works in any university that has a name and place. For instance, Figure 2 is an example about al-Quds magazine published articles B1 and B2. Whereas B1 has a title Data Web which published in 2007 by Tom Lara. The author works at a University of Malta that is in Malta city. Figure 2 describes more details about this example. We must keep in mind that the RDF stores are increasing dynamically.
Figure 3: The main RDF table (MagazineTable)

Figure 4: the Twin table of RDF Data

3.3 Query Analysis

Query performance can be analyzed to improve it by viewing query execution plans or by manipulating of the queries. The goal of query performance is to minimize the response time of queries and to make the best use of the server's resources by minimizing the network traffic, disk I/O, and the efficiency of CPU time. The following tests queries from different levels to demonstrate the evaluation. These queries evaluate on the main RDF table as well as in the new twin tables as figures 3 and 4 were shown.

3.3.1 Queries

Query (1): This is a simple triple query. It aims to retrieve the title of the article B1. With consider the table MagazineTable (S,P,O) which refers to the main RDF table as it is shown in figure 3. The output of execution this query is Data Web.

```
SELECT T.O
FROM MagazineTable T
WHERE T.S = ':B1' AND T.P = ' :Title' ;
```

Query (2): This is a path query with one join. It aims to find all the authors of the article B2.

```
-------------------In Main RDF Table ------------------------------
SELECT T2.O
FROM MagazineTable T1 , MagazineTable T2
```

The output of execution this query is Bob Hacker.

```
-------------------In Twin Table -------------------------------
SELECT T1.S, T1.O
FROM TABLE2 T1 , TABLE1 T2
WHERE (T1.S = T2.O OR T2.S = T1.O) AND ( T1.P = 'author') AND (T2.P = 'affiliation') AND T2.O = (select T2.S from TABLE2 T2 where T2.O = 'University of Malta')
```

The output of execution this query is Bob Hacker.

Query (3): This is a path query with two joins. It aims to list all the articles which have authors from University of Malta and their authors.

```
-------------------In Main RDF Table ------------------------------
SELECT T3.S, T3.O
FROM MagazineTable T1 , MagazineTable T2 , RDF_TRIPLE T3
```

The output of execution this query is B1 and the author is A1.

```
-------------------In Twin Table -------------------------------
SELECT T2.O
FROM TABLE2 T1 , TABLE1 T2
```

The output of execution this query is Bob Hacker.

Query (4): This is a simple triple query. It aims to retrieve the title of the article B1. With consider the table MagazineTable (S,P,O) which refers to the main RDF table as it is shown in figure 3. The output of execution this query is Data Web.

```
SELECT T.O
FROM MagazineTable T
WHERE T.S = ':B1' AND T.P = ' :Title' ;
```

The output of execution this query is Bob Hacker.
Query (4): This is a star query which has four joins. It aims to display a list of authors who work on Cyprus and their articles.

The output of execution this query is A2, the articles B1 and B2.

SELECT T2.S,T2.O from TABLE1 T1, TABLE2 T2 where (T2.S=T1.O OR T1.S=T2.O) AND (T2.P=':Author' OR T2.P=':City') AND T1.O= (select T2.S from TABLE2 T2 where T2.O =(select T1.S from TABLE1 T1 where T1.O = 'Cyprus'))
The output of execution this query is A2, the articles B1 and B2.

4. Experimental Results

This section studies the performance of RMTT approach on a heterogeneous experimental setup comprising real-world RDF datasets from different areas of knowledge. We study query performance against RDF-3X triple (which based on one large triple table) and RMTT approach (which based on two small join tab tables), evaluating the number joins and the query response time. We compare our results with a highly-efficient store (RDF 3X).

4.1 Datasets

The datasets that we use in our experimental are: CAS-IBRI (holds information about staff, faculty, students, departments), DBLP (Provides information about computer science journals and proceedings), DBpedia (is the semantic evaluation of Wikipedia). The main characteristics of these datasets are described in the following table:

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Size(MB)</th>
<th>#Triples</th>
<th>#Subjects</th>
<th>#Predicates</th>
<th>#Objects</th>
</tr>
</thead>
<tbody>
<tr>
<td>CAS-IBRI</td>
<td>130.5</td>
<td>1,950.0</td>
<td>292,360</td>
<td>25</td>
<td>309,233</td>
</tr>
<tr>
<td>DBLP</td>
<td>7,635.60</td>
<td>111,343,500</td>
<td>7,534,606</td>
<td>20</td>
<td>18,214,612</td>
</tr>
<tr>
<td>DBpedia</td>
<td>25,423.60</td>
<td>218,345,10</td>
<td>17,423,222</td>
<td>35,456</td>
<td>61,654,706</td>
</tr>
</tbody>
</table>

Table 1: dataset descriptions

4.2 Query Performance

We submitted number of complex queries against RMTT approach and RDF-3X approach focusing on query response time.

These set of fourteen queries starts with simple query and gradually are complex; we used the three types of datasets explained in table 1. The query performance time is evaluated for both our RMTT approach and RDF-3X approach. From table 2, it’s clear that RMTT is faster than RDF-3X. For DBLP dataset RMTT performs a factor of 1-2 faster than RDF-3X, and for Dpedia performs 3-4 faster than RDF-3X approach. For CAS-IBri dataset the queries response time are almost the same, because the characteristics of this type of data has no more self-joins.

Table 2: queries with response time with different data size and the number of joins

![Figure 5: response times for a) CAS-ibri b) DBLP c) DBpedia](image)
For scalability issue, we evaluate the number of self-joins (if its) using the DBpedia dataset against RMTT and RDF-3X approaches. Table 2 and figure 5 shows that for the first seven queries there is no difference between the two approaches. From query 8 to query 10 the difference in the number of self-joins between the two approaches are very small. From query 11 until query 14 which is more complex our proposed RMTT approach reduced the number of self-joins more than 30% compared with RDF-3X approach. This means the performance of our improves with more complex queries and the size of the data.

5. Conclusion and Future Work

In this paper, we have proposed the RMTT approach that comes as an alternative solution for the current approaches triple table, property table approach and vertically partitioned approach. Because they still suffer from weak data locality which affects the storage and query performance costs. RMTT approach is just a partitioning step for making high performance and efficient management of RDF data. In the future work, we try to demonstrate a good indexing mechanism which is suitable to RMTT approach. That works together with partitioning step and query the evaluation to achieve efficient and scalable management of RDF store.

Appendix

Query: 1

PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX ub: <http://www.lehigh.edu/~zhp2/2004/0401/univ-bench.owl#>
SELECT ?X
WHERE
{?X rdf:type ub:GraduateStudent .
?X ub:takesCourse http://www.Department0.University0.edu/GraduateCourse0}

Query: 2

PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX ub: <http://www.lehigh.edu/~zhp2/2004/0401/univ-bench.owl#>
WHERE
{?X rdf:type ub:GraduateStudent .
?Y rdf:type ub:University .
?Z rdf:type ub:Department .
?X ub:undergraduateDegreeFrom ?Y}

Query: 3

PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX ub: <http://www.lehigh.edu/~zhp2/2004/0401/univ-bench.owl#>
SELECT ?X
WHERE
{?X rdf:type ub:Publication .
?X ub:publicationAuthor http://www.Department0.University0.edu/AssistantProfessor0}

Query: 4

PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX ub: <http://www.lehigh.edu/~zhp2/2004/0401/univ-bench.owl#>
SELECT ?X, ?Y1, ?Y2, ?Y3
WHERE
{?X rdf:type ub:Professor .
?X ub:undergraduateDegreeFrom ?Y .
?X ub:telephone ?Y3}

Query: 5

PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX ub: <http://www.lehigh.edu/~zhp2/2004/0401/univ-bench.owl#>
SELECT ?X
WHERE
{?X rdf:type ub:Person .
?X ub:memberOf <http://www.Department0.University0.edu>}

Query: 6

PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX ub: <http://www.lehigh.edu/~zhp2/2004/0401/univ-bench.owl#>
SELECT ?X WHERE {?X rdf:type ub:Student}
Query:9

PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX ub: <http://www.lehigh.edu/~zhp2/2004/0401/univ-bench.owl#>
WHERE
{?X rdf:type ub:Student .
 ?Y rdf:type ub:Department .
 ?X ub:emailAddress ?Z}


