Paper Narrative:
SW-Store: a vertically partitioned DBMS for Semantic Web data Management

Group 4
NIPUN GARG 4282567
SURABHI MITHAL 4282643
Introduction

The paper is an analysis of the storage mechanisms available for RDF data which exist currently and proposes a new mechanism for storage of the data. The authors run various real world queries to compare and contrast the performance of the existing techniques and the proposed scenario. An analysis of the reason for the performance of various storage systems is done. Further, a new column oriented database called SW-Store is introduced which is based on the concepts outlined in the paper.

Problems addressed

The paper outlines the need for effective storage, management and retrieval of RDF data. The problems highlighted in this paper are:-

1. Vast data is available on the internet but it resides in silos of information which are not linked. The Semantic web concept addresses this problem by integrating this data. But, applying this concept to the real world applications has issues related to scalability and performance.
2. Resource description Framework (RDF) is the data model for semantic web and forms an important part of the concept. The current storage procedures for RDF do not perform well due to the nature and size of the data.
3. Typically the format for RDF data exchange is XML but it is generally parsed to produce series of triples of the form \(< subject, property, object >\). These triples are then stored in a relation database. This table becomes really large and when queries are executed there are a lot of self joins which lead to poor performance.
4. Traditional row oriented databases do not perform well with this kind of data and there is a need for a new mechanism to store data.

This paper put forward some good ideas of how to solve the questions addressed above. The authors' ideas can overcome the limitations which exist currently and provide a better way of storing and analyzing RDF data.

This is relevant to the course as semantic web is an area of research and the concept is expanding to geospatial semantic web. The study of geospatial semantic web is to combine the Semantic Web and geospatial data by developing effective geospatial ontology. An example is: http://www.geonames.org/ontology/documentation.html

Contributions

The paper has many good contributions to the field of semantic web which include:

1. The analysis of current mechanisms storing RDF data in databases and highlighting the limitations they have in terms of scalability and performance.
2. Introduction of a new concept of vertically partitioning RDF data to improve query performance and a mechanism to use a column-oriented database with the vertical partitioning approach to improve performance and increase simplicity.
3. The performance evaluation of the new and existing techniques with a real world and appropriate example and a good explanation of the results are provided. The results show that the new technique outperforms the existing mechanism by a significant magnitude.
4. A new column oriented database SW-store is proposed which is based on storing vertically partitioned RDF data in an effective manner. The basic structure of this database is explained with some examples.
The most important contribution is proposal to store RDF data as vertically partitioned on a column oriented database which improves the performance of queries drastically over traditional methods. The research group tested the solution they provide with a publicly available library dataset and a collection of queries generated from a web-based user interface for browsing RDF content.

**Key Concepts and Ideas**

One of the current techniques for resolving the performance problems related to querying RDF data is a *property table*. The main idea behind property table is to combine frequently occurring related properties into a new table of the structure <subject, property 1, property 2...>. A data clustering algorithm is used to find out the related properties. This type of property table is called *clustered property table*. The limitation with this type of property table is that a property can occur at most only in one property table. So the author has described another way to create property tables. This is called "*property class table*". This approach creates clusters based on subject’s type property and one property can occur in multiple property tables. An example of this idea is shown in Figure 1.

Though this reduces the number of self joins (subject-object joins) needed to answer queries there are drawbacks associated with this approach. Some of the major drawbacks include:

- **NULLs in data**: As all the properties are not defined for each occurrence there are a lot of NULLs in the data which consume a lot of space.
- **Multi Valued attributes**: RDF data has a lot of multivalued attributes, which are not effectively represented in case of property tables.
- **Clustering complexity**: As the basis of the property tables is clustering of related attributes the accuracy of the clustering affects the performance of this storage mechanism. The author has later proved that when clustering is not done accurately the performance degrades rapidly.

**Table 1** Some sample RDF data and possible property tables

<table>
<thead>
<tr>
<th>Subj.</th>
<th>Prop. type</th>
<th>Obj. type</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID1</td>
<td>&quot;title&quot;</td>
<td>&quot;BookType&quot;</td>
</tr>
<tr>
<td>ID2</td>
<td>&quot;author&quot;</td>
<td>&quot;Person&quot;</td>
</tr>
<tr>
<td>ID3</td>
<td>&quot;copyright&quot;</td>
<td>&quot;Date&quot;</td>
</tr>
<tr>
<td>ID4</td>
<td>&quot;title&quot;</td>
<td>&quot;Artist&quot;</td>
</tr>
<tr>
<td>ID5</td>
<td>&quot;language&quot;</td>
<td>&quot;English&quot;</td>
</tr>
</tbody>
</table>

**Property Table**

<table>
<thead>
<tr>
<th>Subj.</th>
<th>Type</th>
<th>Title</th>
<th>copyright</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID1</td>
<td>&quot;BookType&quot;</td>
<td>&quot;NY&quot;</td>
<td>2001</td>
</tr>
<tr>
<td>ID2</td>
<td>&quot;CDType&quot;</td>
<td>&quot;ABC&quot;</td>
<td>1985</td>
</tr>
<tr>
<td>ID3</td>
<td>&quot;BookType&quot;</td>
<td>&quot;Book&quot;</td>
<td>NULL</td>
</tr>
<tr>
<td>ID4</td>
<td>&quot;DVDType&quot;</td>
<td>&quot;DE&quot;</td>
<td>NULL</td>
</tr>
<tr>
<td>ID5</td>
<td>&quot;CDType&quot;</td>
<td>NULL</td>
<td>2004</td>
</tr>
</tbody>
</table>

**Left-Over Triples**

<table>
<thead>
<tr>
<th>Subj.</th>
<th>Prop. type</th>
<th>Obj. type</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID1</td>
<td>&quot;author&quot;</td>
<td>&quot;Person&quot;</td>
</tr>
<tr>
<td>ID2</td>
<td>&quot;language&quot;</td>
<td>&quot;English&quot;</td>
</tr>
<tr>
<td>ID3</td>
<td>&quot;title&quot;</td>
<td>&quot;Artist&quot;</td>
</tr>
</tbody>
</table>

**Clustering Example**

```
SELECT C.obj
FROM TRIPLES AS A,
     TRIPLES AS B,
     TRIPLES AS C
WHERE A.subj = B.subj,
     AND A.obj = C.subj,
     AND A.prop = 'copyright',
     AND B.prop = 'author',
     AND C.prop = 'title'
```

**Figure 1:** RDF triples and Property tables

*Source:* SW-Store: a vertically partitioned DBMS for Semantic Web data management
The authors propose a storage mechanism which involves vertical partitioning of data and further storing this vertically partitioned data into a column oriented database. A two column table for each unique property in the RDF data is created where the first column contains subjects and the second column contains the object values for that property. The table is sorted on the subject for quick access. The advantages of this approach are the following:

- Multivalued attributes can be handled efficiently as each value can now form a new row in the two column table.
- Elimination of NULLs: If the value for a particular property is null, it can be eliminated completely instead of storing nulls.
- Clustering of properties is no longer needed.
- The number of unions is less as compared to the property class approach in case when the query does not restrict on one class type.

The authors explain the choice of a column oriented database by listing some advantages of storing data in a column oriented database. Some of these are: no wastage of bandwidth as projections on data happen before it is pulled into main memory as opposed to after in case of row oriented store, record header is stored in separate columns thus reducing the tuple width and letting us choose different compression techniques for each column.

The authors propose SW-store which is a column oriented DBMS optimized for storing RDF triples after partitioning them vertically. The Key concepts in this DBMS include:-

- Storage system: - Some properties can be stored as a single column table having one to one mapping with the table containing subjects. Depending on the volume of the sparse data appropriate compression scheme are applied.
- Query engine: - Lack of tree structure in query plan for column oriented databases can lead to problems where there can be mismatch of rate of consumption of data. This is overcome by ensuring that the graph is still rooted with a single node with no parents and the parents request the data at the same rate.
- Overflow tables and Query translation: - As the database is compressed and vertically partitioned, to overcome the limitation of slow insert an overflow table is maintained. Queries go both to overflow tables and vertical partitions and later on merged.
- Materialized joins: - Subject object joins can be removed using materialized paths. An example is storing multiple properties as one property to prevent multiple joins for frequent queries.

**Dataset, Software and hardware**

The research group uses the dataset taken from the publicly available Barton Libraries dataset provided by the Simile Project at MIT (http://simile.mit.edu/rdf-test-data/barton). They do some pre-processing of the dataset to improve the quality of data namely elimination of duplicates and removing data with very long literals.

The set of queries they run is fixed and is based on a browsing session of Longwell, a UI built by Simile group for querying the library dataset.

Hardware: - The system they use for evaluating has the following configuration:

- 3.0 GHz Pentium IV
- RedHat Linux
- 2 GB of RAM, 1MB L2 cache, and a 3-disk, 750 GB striped RAID array. The disk can read data at 150–180 MB/sec.

**Software:** - The following databases are used for evaluation
- PostGres SQL database for evaluation of triples technique, property tables and vertical portioning approach.
- C-Store a open source column oriented store is used by them for checking the proposed vertical portioned and column oriented store.

**Validation Methodology**

The authors use an accurate and interesting validation methodology which involves seven queries that are executed during the use of Longwell. The user of Longwell selects a few specific entities to focus on and queries execute on the backend RDF store. The queries are listed on the end of this narrative.

These queries are then executed on the all the existing mechanisms and the new techniques proposed namely:-
1. Triple data store (subject, property, object table with no improvements on Postgres).
2. Property tables (on Postgres)
4. Vertically partitioned data in a column oriented store (C-Store).

**Strengths of this methodology include:**

- Real world data and effective coverage of the different query scenarios which are easy to understand.
- Comparison of all the techniques which exist and the proposed technique in an effective medium which is not biased towards ant particular approach.
- The inclusion of special/practical scenarios and revaluating the results is not very hard using this methodology.

**Weaknesses of this methodology are:**

- Avoiding queries involving unrestricted property problem which are particularly prevalent for vertical partitioned scenarios
- The methodology uses clustering technique for property table’s, accuracy of which is assumed very high by the authors.
- With the different setting i.e. use of a different underlying databases the performance may differ.

**Reasons for choosing this methodology**

The author chooses this methodology because:-

- The paper describes the storage and effective retrieval of RDF data and the most logical way to test database performance is to see the response time of complex real world queries.
- An unstructured real world data set is already available publically with an effective UI to query it. This helped the authors design the execution flow simulating a real world flow.
Evaluation of Results:

The following chart summarizes the results obtained by the authors executing the validation methodology.

Assumptions by the author

The assumptions made by the author are:-

1. Postgres is assumed to be the best available choice for a row oriented RDBMS because of effective handling of NULLs.
2. Authors assume that queries that do not restrict on property values are very rare for RDF applications and have not evaluated their technique for these queries.
3. Authors assume that there will be a moderate amount of Insert/Updates on RDF store which will keep the compression and decompression of the data to a minimum.
4. Authors have assumed that both attributes of two column tables in vertically partitioned RDF store are fixed length using dictionary encoding for strings but have ignored the overheads involved for this.

Critique of the assumption (3)

In our point of view, assuming that the amount of Inserts/ Update to the database is moderate to low is unfair. We cannot presume the rate of changes to the database in a real world scenario. In other words, they have restricted the performance evaluation of SW-store.

If this assumption is not there and these operations occur frequently SW-store performance might degrade because the overflow tables will get filled rapidly and the batch operation to update the column oriented store will occur more often degrading the performance as a whole.

Preservation and Revisions to the paper

We would definitely like to preserve the validation methodology and key ideas put forward in this paper. This seems a promising idea especially due to the simplicity and ease of implementation.

Revisions
Authors have not paid much attention to the RDF concept from spatial perspective –
• Schema design - Queries are fired on vertically partitioned tables as well as overflow tables. Owing to the heaviness of spatial data, there should be some spatial indexing like R* TREE or GRID to make these queries faster.

• Restrictive nature - Spatial queries are not restricted to only specific “properties” which is an important assumption on their part. E.g. Landmarks
Tables should be partitioned in a better way rather than just handling one property per table. Taxonomies can be helpful grouping similar properties together.

Also, the paper can be revised to have a section on future work with details on how would they like to proceed with the implementation of SW-store and issues they feel they may face. The paper does not include results from SW-store specifically and it would be good to include results of queries after the initial implementation of SW-store is done to clearly indicate its performance.

Appendix

Query1:
SELECT A.obj, count(*)
FROM triples AS A
WHERE A.prop = "<type>"
GROUP BY A.obj

Query2:
SELECT B.prop, count(*)
FROM triples AS A, triples AS B,
properties AS P
WHERE A.subj = B.subj
AND A.prop = "<type>"
AND A.obj = "<Text>"
AND P.prop = B.prop
GROUP BY B.prop

Query3:
SELECT B.prop, B.obj, count(*)
FROM triples AS A, triples AS B,
properties AS P
WHERE A.subj = B.subj
AND A.prop = "<type>"
AND A.obj = "<Text>"
AND P.prop = B.prop
GROUP BY B.prop, B.obj
HAVING count(*) > 1

Query4:
SELECT B.prop, B.obj, count(*)
FROM triples AS A,
triples AS B,
triples AS C,
properties AS P
WHERE A.subj = B.subj
AND A.prop = "<type>"
AND A.obj = "<Text>"
AND P.prop = B.prop
AND C.subj = B.subj
AND C.prop = "<language>"
AND C.obj = "<language/iso639-2b/fre>"
GROUP BY B.prop, B.obj
HAVING count(*) > 1

**Query5:**
SELECT B.subj, C.obj
FROM triples AS A, triples AS B,
triples AS C
WHERE A.subj = B.subj
AND A.prop = "<origin>"
AND A.obj = "<info:marcorg/DLC>"
AND B.prop = "<records>"
AND B.obj = C.subj
AND C.prop = "<type>"
AND C.obj != "<Text>"

**Query6:**
SELECT A.prop, count(*)
FROM triples AS A,
properties AS P,
( (SELECT B.subj
FROM triples AS B
WHERE B.prop = "<type>"
AND B.obj = "<Text>"))
UNION
(SELECT C.subj
FROM triples AS C,
triples AS D
WHERE C.prop = "<records>"
AND C.obj = D.subject
AND D.prop = "<type>"
AND D.obj = "<Text>"))
) AS uniontable
WHERE A.subj = uniontable.subj
AND P.prop = A.prop
GROUP BY A.prop

**Query7:**
SELECT A.subj, B.obj, C.obj
FROM triples AS A, triples AS B,
triples AS C
WHERE A.prop = "<Point>"
AND A.obj = "end"
AND A.subj = B.subject
AND B.prop = "<Encoding>"
AND A.subj = C.subject
AND C.prop = "<type>"