Optimization Approaches for SPARQL and Storage Schemes for RDF - An Experimental Analysis -

Abschlusskolloquium Graduiertenkolleg 806/3 “Mathematische Logik und Anwendungen”

September 12, 2008

Speaker: Michael Schmidt
(joint work with T. Hornung, N. Kuechlin, G. Lausen, and C. Pinkel)
Table of Content

• Introduction to RDF and SPARQL
• Motivation
• The SP²Bench Framework
• Experimental Analysis of RDF Storage Schemes
RDF and SPARQL

- **RDF**
  - Data format for encoding information/knowledge
  - W3C Recommendation since February 2004
- **SPARQL**
  - “SPARQL Protocol and Query Language” for RDF
  - Declarative Language
  - W3C Recommendation since January 2008
The RDF Data Format

- RDF Triples encode knowledge facts
- Each triple of the form \((\text{subject}, \text{predicate}, \text{object})\)
- Encodes binary relation \(\text{predicate}\) between \(\text{subject}\) and \(\text{object}\)
- Example:

  \[(\text{Book1, title, “DBMS”})\]
The RDF Data Format

- Three basic sets of elements
  - \( U \) - URIs (Uniform Resource Identifiers)
  - \( B \) - Blank Nodes
  - \( L \) - Literals

- RDF triple \( t \in (U \cup B) \times U \times (U \cup B \cup L) \)

\[(\text{Book1}, \text{title}, "\text{DBMS}")\]
The RDF Data Format

• An RDF Database is a set of triples
  \[ D \subseteq (U \cup B) \times U \times (U \cup B \cup L) \]

• Example:

\[
\{ (\text{Book1, type, Book}), (\text{Book1, title, “DBMS”}), \\
(\text{Book1, author, Person1}), (\text{Book1, author, Person2}), \\
(\text{Book1, issued, “2002”}), (\text{Person1, name, “J. Gehrke”}), \\
(\text{Person2, name, “R. Ramakrishnan”}), \\
(\text{Inproc1, type, Inproceedings}), \\
(\text{Inproc1, title, “BOAT - optimistic decision tree construction”}), \\
(\text{Inproc1, author, Person1}), (\text{Inproc1, author, Person2}), \\
(\text{Inprocl, author, Person3}), (\text{Person3, name, “V. Ganti”}) \}
\]
RDF Graph Representation

- RDF Databases can be represented by directed, acyclic graphs
SPARQL Query Language

- Let $V$ be a set of variables \{ $?x_1, $?x_2, $?x_3, ... \}

- Basic construct: triple patterns of the form $t \in (U \cup B \cup V) \times (U \cup V) \times (U \cup B \cup L \cup V)$

- Example:

  $(?e, title, ?title)$

  also written as:

  $?e title ?title$
SPARQL Query Language

Inproceeding

Book

Inprocl

Book I

BOAT - optimistic [...] DBMS

author

author

author

author

Inprocl

author

name

name

name

J. Gehrke

R. Ramakrishnan

V. Ganti

?e title ?title

author

issued 2002

Inproceeding

Book

?e title ?title

Inproceeding

Book
SPARQL Query Language

Inproceeding

Book

Inprocl

Book1

DBMS

2002

BOAT - optimistic [...]

Person3

Person2

Person1

J. Gehrke

R. Ramakrishnan

V. Ganti

?e title ?title
SPARQL Query Language

\[
S = \{\begin{array}{ll}
\text{?e} & \text{Inproc1, ?title -> “BOAT …”} \\
\text{?e} & \text{Book1, ?title -> “DBMS”}
\end{array}\}
\]
SPARQL Query Language

- Operators can be used to combine simple triple patterns to more complex queries
- **SELECT**: projection for variables
- **AND**: (denoted as “.”): join over shared variables
- **UNION**: computes union of two result sets
- **OPTIONAL**: optional selection of components
- **FILTER**: filters for a specified condition
SELECT ?yr
SELECT ?yr
Motivation

• Efficient evaluation of SPARQL is a non-trivial task

• SPARQL evaluation is $\text{PSPACE}$-complete (even for many subfragments)

• Homogeneous data format poses potential for severe bottlenecks (as we will discuss later)

• Several optimization approaches have been made, but use their own, user-defined experimental setting for verification
Motivation

• Need for a SPARQL benchmark framework that
  • ... poses various challenges to SPARQL engines
  • ... helps to identify strengths/weaknesses of approaches
  • ... allows to compare optimization approaches
  • $SP^2Bench$ Framework satisfies these needs
• Data generator for arbitrarily large bibliographic models in RDF format
• Set of meaningful and challenging benchmark queries
SP²Bench Queries

• Vary in a broad range of characteristics
  • Meaningful requests on top of the data
  • Different operator constellation, RDF access patterns, and complexity
  • Result size (small, large, linear, ...)
  • Number of variables
  • ...

• ...
Storage Schemes for RDF

- Idea: translation into Relational context and evaluation of queries with conventional SQL database systems
- We consider two different approaches
  - Simple Triple Table Approach
  - Vertical Partitioning
Triple Table Approach

- Simple and straightforward storage scheme for RDF data
- All data stored in a single relation
  $\text{Triples}(\text{subject, predicate, object})$

<table>
<thead>
<tr>
<th>subject</th>
<th>predicate</th>
<th>object</th>
</tr>
</thead>
<tbody>
<tr>
<td>Book 1</td>
<td>type</td>
<td>Book</td>
</tr>
<tr>
<td>Book 1</td>
<td>title</td>
<td>“DBMS”</td>
</tr>
<tr>
<td>Book 1</td>
<td>issued</td>
<td>“2002”</td>
</tr>
<tr>
<td>Book 1</td>
<td>author</td>
<td>Person 1</td>
</tr>
<tr>
<td>Book 1</td>
<td>author</td>
<td>Person 2</td>
</tr>
<tr>
<td>Person 1</td>
<td>name</td>
<td>“J. Gehrke”</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
**Triple Table Approach**

- Systematic SPARQL-to-SQL rewriting to evaluate SPARQL queries on top of the triples table

SPARQL-to-SQL translation:

```sql
SELECT T3.object AS yr
FROM Triples T1,
     Triples T2,
     Triples T3
WHERE
  T1.predicate = "type" AND
  T1.object = "Book" AND
  T2.predicate = "title" AND
  T2.object = "DBMS" AND
  T3.predicate = "issued" AND
  T1.subject = T2.subject AND
  T2.subject = T3.subject
```
Triple Table Approach

- Main disadvantage: Resulting queries typically contain many self-joins

```sql
SELECT T3.object AS yr
FROM Triples T1,
     Triples T2,
     Triples T3
WHERE
  T1.predicate = “type” AND
  T1.object = “Book” AND
  T2.predicate = “title” AND
  T2.object = “DBMS” AND
  T3.predicate = “issued” AND
  T1.subject = T2.subject AND
  T2.subject = T3.subject
```
Dictionary Encoding

- URIs and Literals tend to be long strings
- Save space and accelerate joins through encoding

### Triples

<table>
<thead>
<tr>
<th>subject</th>
<th>predicate</th>
<th>object</th>
</tr>
</thead>
<tbody>
<tr>
<td>Book1</td>
<td>type</td>
<td>Book</td>
</tr>
<tr>
<td>Book1</td>
<td>title</td>
<td>“DBMS”</td>
</tr>
<tr>
<td>Book1</td>
<td>issued</td>
<td>“2002”</td>
</tr>
<tr>
<td>Book1</td>
<td>author</td>
<td>Person1</td>
</tr>
<tr>
<td>Book1</td>
<td>author</td>
<td>Person2</td>
</tr>
<tr>
<td>Person1</td>
<td>name</td>
<td>“J. Gehrke”</td>
</tr>
</tbody>
</table>

### Dictionary

- **ID** | **val**
  - 1     | Book1   
  - 2     | type     
  - 3     | Book     
  - 4     | title    
  - 5     | “DBMS”   
  - 6     | issued   
  - 7     | 2002     
  - 8     | author   
  - 9     | Person1  
  - 10    | Person2  
  - 11    | name     
  - 12    | “J. Gehrke” 

### Dictionary encoding

```
<table>
<thead>
<tr>
<th>subject</th>
<th>predicate</th>
<th>object</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>1</td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>1</td>
<td>8</td>
<td>9</td>
</tr>
<tr>
<td>1</td>
<td>8</td>
<td>10</td>
</tr>
<tr>
<td>9</td>
<td>11</td>
<td>12</td>
</tr>
</tbody>
</table>
```

...
Vertical Partitioning

- Set up one table for each distinct property (predicate) in the data
- Per table, store all tuples with the respective predicate
Vertical Partitioning

- Systematic SPARQL-to-SQL rewriting also exists for this scheme
- Approach very similar to this for the Triple Table approach
- Resulting queries contain joins over the (typically smaller) predicate tables instead of the whole Triples table
Efficient Merge Joins

**Query:** Select all book authors

```sql
SELECT a.object
FROM type t, author a
WHERE
  t.object="Book" AND
  t.subject=a.subject
```

<table>
<thead>
<tr>
<th>subject</th>
<th>object</th>
</tr>
</thead>
<tbody>
<tr>
<td>Book1</td>
<td>Book</td>
</tr>
<tr>
<td>Book2</td>
<td>Book</td>
</tr>
<tr>
<td>Book3</td>
<td>Book</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>subject</th>
<th>object</th>
</tr>
</thead>
<tbody>
<tr>
<td>Book1</td>
<td>Person1</td>
</tr>
<tr>
<td>Book1</td>
<td>Person2</td>
</tr>
<tr>
<td>Book2</td>
<td>Person2</td>
</tr>
<tr>
<td>Book2</td>
<td>Person3</td>
</tr>
<tr>
<td>Book3</td>
<td>Person4</td>
</tr>
<tr>
<td>Book3</td>
<td>Person5</td>
</tr>
<tr>
<td>Book3</td>
<td>Person6</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
Efficient Merge Joins

SELECT a.object
FROM type t, author a
WHERE
  t.object = 'Book' AND
  t.subject = a.subject

Result: {(Person1)}
### Efficient Merge Joins

**SELECT** `a.object`  
**FROM** `type t, author a`  
**WHERE**  
`t.object = "Book"`  
`t.subject = a.subject`

Result: `{(Person1), (Person2)}`
Efficient Merge Joins

SELECT a.object
FROM type t, author a
WHERE
  t.object="Book" AND
  t.subject=a.subject

Result: {(Person1), (Person2)}
Efficient Merge Joins

```
SELECT a.object
FROM type t, author a
WHERE t.object="Book" AND t.subject=a.subject
```

Result: {(Person1), (Person2), (Person2)}
Efficient Merge Joins

```
SELECT a.object 
FROM type t, author a 
WHERE 
  t.object=“Book” AND 
  t.subject=a.subject
```

Result: { (Person1), (Person2), (Person2), (Person3) }
Efficient Merge Joins

```
SELECT a.object
FROM type t, author a
WHERE
  t.object = "Book" AND
  t.subject = a.subject
```

Result: \{(Person1), (Person2), (Person2), (Person3)\}
SELECT a.object
FROM type t, author a
WHERE
  t.object="Book" AND
  t.subject=a.subject

Result: {(Person1), (Person2), (Person2), (Person3), (Person4)}
Efficient Merge Joins

```
SELECT a.object
FROM type t, author a
WHERE
  t.object="Book" AND
  t.subject=a.subject
```

Result: 

```
{ (Person1), (Person2), (Person2), (Person3), (Person4), (Person5) }
```
Efficient Merge Joins

```
SELECT a.object
FROM type t, author a
WHERE
  t.object="Book" AND
  t.subject=a.subject

Result: {(Person1), (Person2), (Person2), (Person3), (Person4), (Person5), (Person6)}
```
Experimental Setting

- **Scenario TR:**
  - Simple Triple Table Approach
  - Physically sorted by \((predicate, subject, object)\)
  - Combined with Dictionary Encoding

- **Scenario VP:**
  - Vertical Partitioning as described before
  - Combined with Dictionary Encoding
Experimental Setting

- Scenario **RS**:  
  - Purely relational model of the scenario  
  - Using flat tables for docs, authors, citations, etc.  
  - Designed using ERM DB modeling techniques

- Scenario **SP**:  
  - Sesame native SPARQL engine  
  - No RDF/SPARQL-to-SQL translation necessary
Settings Summary

- TR: Triple Table Approach
- VP: Vertical Partitioning
- RS: Purely Relational Schema
- SP: SPARQL Engine Sesame

- Intel2 DuoCore 2.13GHz CPU, 3GB DDR2 RAM
- OS: Ubuntu v7.10 gutsy Linux
- 30min timeout per query, 2GB main memory limit

MonetDB mserver v5.5.0, using the new algebra frontend

Sesame v2.0 coupled with its native SAIL
Select all distinct pairs of article author names that have published in the same journal.
#Triples: S1=10k / S2=50k / S3=250k / S4=1M / S5=5M / S6=25M
Experimental Results Q6

Return, for each year, the set of publications authored by persons that have not published in a year before

```
SELECT ?yr ?name ?doc
WHERE {
  ?class rdfs:subClassOf foaf:Document.
  ?author foaf:name ?name
  OPTIONAL {
    ?class2 rdfs:subClassOf foaf:Document.
    ?doc2 dc:creator ?author2
    FILTER (?author=?author2 && ?yr2<?yr)
  }
  FILTER (!bound(?author2))
}
```
#Triples:  S1=10k / S2=50k / S3=250k / S4=1M / S5=5M / S6=25M
Conclusion

- Experiments identified previously unknown drawbacks
- Vertical Partitioning not a general solution
- Triple Store with different physical sort order competitive to Vertical Partitioning
- Gap of at least one order of magnitude compared to relational data processing, increasing with doc size

New storage schemes and query evaluation approaches need to be developed, to bring forward the evaluation of SPARQL queries
Selected References

G. Lausen, M. Meier, and M. Schmidt. SPARQLing Constraints for RDF. In EDBT, 2008.

A. Chebotko et al.. Semantics Preserving SPARQL-to-SQL Query Translation for Optional Graph Patterns. TR-DB-052006-CLJF.
D.J. Abadi et al.. Using the Barton libraries dataset as an RDF benchmark. TR MIT-CSAIL-TR-2007-036, MIT.
M. Ley. DBLP Database. http://www.informatik.uni-trier.de/ley/db/.

MonetDB: http://monetdb.cwi.nl/
Sesame: http://www.openrdf.org/