An Experimental Comparison of RDF Data Management Approaches in a SPARQL Benchmark Scenario

7th International Semantic Web Conference
Karlsruhe, Germany
October 28, 2008

Speaker: Michael Schmidt
joint work with T. Hornung, N. Küchlin, G. Lausen, and C. Pinkel
Motivation

• Efficient evaluation of SPARQL is a non-trivial task

• SPARQL evaluation is $\text{PSPACE}$-complete

• 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
Contributions

**SPARQL Performance Benchmark SP²Bench**

- Data Generator + Benchmark Queries
- Queries pose various challenges to SPARQL engines
- Allows us to compare optimization approaches
- Available online at

http://dbis.informatik.uni-freiburg/index.php?project=SP2B
Contributions

Part II
- Evaluation of existing RDF management approaches
  - Focus on translations into relational context

Part III
- Comparison to native engine, relational setting
- Several new findings
  - Limitations of existing evaluation approaches
  - Severe gap to native relational data processing
SP²Bench Scenario

- Domain: DBLP bibliographic data
- Contains bibliographic entities such as articles, journals, proceedings, inproceedings...
- DBLP fits „RDF philosophy“
- RDF designed for representing meta data
- Many social-world distributions found in DBLP

Part I – The SP²Bench SPARQL Performance Benchmark
Data with Real-world Characteristics

#instances per year for each document type

real DBLP
vs.
approx. in SP²Bench
Data with Real-world Characteristics

• Other characteristics that we consider
  • Citation system
    • Incoming citations per publication (follows a power law distribution)
    • Outgoing citations per publication
  • Structure of documents
  • ...

Part I – The SP²Bench SPARQL Performance Benchmark
SP²Bench SPARQL Queries

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

Part I – The SP²Bench SPARQL Performance Benchmark
Storage Schemes for RDF

• Focus of this work: 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

Triples(subject, predicate, object)

<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>
<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

```
SELECT ?author
WHERE {
}
```

“Select all book authors”

```
SELECT T2.object AS author
FROM Triples T1,
     Triples T2
WHERE
    T1.predicate="type" AND
    T1.object="Book" AND
    T2.predicate="author" AND
    T1.subject=T2.subject
```
Triple Table Approach

- Main disadvantage: resulting queries typically contain self-joins over table Triples
### Dictionary 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>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

#### Dictionary encoding

<table>
<thead>
<tr>
<th>ID</th>
<th>val</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Book1</td>
</tr>
<tr>
<td>2</td>
<td>type</td>
</tr>
<tr>
<td>3</td>
<td>Book</td>
</tr>
<tr>
<td>4</td>
<td>title</td>
</tr>
<tr>
<td>5</td>
<td>“DBMS”</td>
</tr>
<tr>
<td>6</td>
<td>issued</td>
</tr>
<tr>
<td>7</td>
<td>2002</td>
</tr>
<tr>
<td>8</td>
<td>author</td>
</tr>
<tr>
<td>9</td>
<td>Person1</td>
</tr>
<tr>
<td>10</td>
<td>Person2</td>
</tr>
<tr>
<td>11</td>
<td>name</td>
</tr>
<tr>
<td>12</td>
<td>J. Gehrke</td>
</tr>
</tbody>
</table>

#### Triples

<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>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

#### Dictionary

<table>
<thead>
<tr>
<th>ID</th>
<th>val</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Book1</td>
</tr>
<tr>
<td>2</td>
<td>type</td>
</tr>
<tr>
<td>3</td>
<td>Book</td>
</tr>
<tr>
<td>4</td>
<td>title</td>
</tr>
<tr>
<td>5</td>
<td>“DBMS”</td>
</tr>
<tr>
<td>6</td>
<td>issued</td>
</tr>
<tr>
<td>7</td>
<td>2002</td>
</tr>
<tr>
<td>8</td>
<td>author</td>
</tr>
<tr>
<td>9</td>
<td>Person1</td>
</tr>
<tr>
<td>10</td>
<td>Person2</td>
</tr>
<tr>
<td>11</td>
<td>name</td>
</tr>
<tr>
<td>12</td>
<td>J. Gehrke</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

---

**Part II – Experimental Setting**

**Triples**

**Dictionary**
Vertical Partitioning

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

<table>
<thead>
<tr>
<th>subject</th>
<th>object</th>
</tr>
</thead>
<tbody>
<tr>
<td>Book1</td>
<td>Book</td>
</tr>
<tr>
<td>Inproc1</td>
<td>Inproceeding</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>Inproc1</td>
<td>Person1</td>
</tr>
<tr>
<td>Inproc1</td>
<td>Person2</td>
</tr>
<tr>
<td>Inproc1</td>
<td>Person3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>subject</th>
<th>object</th>
</tr>
</thead>
<tbody>
<tr>
<td>Person1</td>
<td>“J. Gehrke”</td>
</tr>
<tr>
<td>Person2</td>
<td>“R. Ramakrishnan”</td>
</tr>
<tr>
<td>Person3</td>
<td>“V. Ganti”</td>
</tr>
</tbody>
</table>

Vertical Partitioning

- Systematic SPARQL-to-SQL rewriting to evaluate SPARQL queries on top of the predicate tables, similar to the Triple Table approach

```
SELECT ?author
WHERE {
}
```

“Select all book authors”
**Part II – Experimental Setting**

**Merge Joins (Vertical Partitioning)**

```
SELECT au.object AS author
FROM type ty,
     author au
WHERE
  ty.object="Book" AND
  ty.subject=au.subject
```

“Select all book authors”

<table>
<thead>
<tr>
<th>type</th>
<th>author</th>
</tr>
</thead>
<tbody>
<tr>
<td>subject</td>
<td>object</td>
</tr>
<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>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>

Part II – Experimental Setting

Merge Joins (Vertical Partitioning)

```
SELECT au.object AS author
FROM type ty, author au
WHERE
  ty.object="Book" AND
ty.subject=au.subject
```

“Select all book authors”

Efficient evaluation by merging subject columns when data physically sorted by (subject,object)!

Finding: merge joins also possible in Triple Table scenario when physically sorting data by (predicate,subject,object)!

### Vertical Partitioning

<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>Book3</td>
<td>Person4</td>
</tr>
<tr>
<td>Book3</td>
<td>Person5</td>
</tr>
<tr>
<td>Book3</td>
<td>Person6</td>
</tr>
</tbody>
</table>

### Triple Table Approach

<table>
<thead>
<tr>
<th>subject</th>
<th>predicate</th>
<th>object</th>
</tr>
</thead>
<tbody>
<tr>
<td>Book1</td>
<td>author</td>
<td>Person1</td>
</tr>
<tr>
<td>Book1</td>
<td>author</td>
<td>Person2</td>
</tr>
<tr>
<td>Book2</td>
<td>author</td>
<td>Person2</td>
</tr>
<tr>
<td>Book3</td>
<td>author</td>
<td>Person4</td>
</tr>
<tr>
<td>Book3</td>
<td>author</td>
<td>Person5</td>
</tr>
<tr>
<td>Book3</td>
<td>author</td>
<td>Person6</td>
</tr>
</tbody>
</table>

see also: L. Sidirourgos, R. Gocalves, M. Kerstin, N. Nes, and S. Manegold: Column-store Support for RDF Data Management: not all swans are white. In VLDB’08.
Experimental Setting

- Scenario **TR**: Simple Triple Table approach
  - Data physically sorted by \((\text{predicate}, \text{subject}, \text{object})\)
  - Secondary index for remaining permutations of \text{subj.}, \text{pred.}, \text{obj.}
  - Combined with Dictionary Encoding
- Scenario **VP**: Vertical Partitioning
  - Data physically sorted by \((\text{subject}, \text{object})\)
  - Secondary Index for \((\text{object}, \text{subject})\)
  - Combined with Dictionary Encoding
Experimental Setting

- Scenario **SP**: Sesame native SPARQL engine
  - No RDF/SPARQL-to-SQL translation necessary
  - Provided Sesame all possible indices on RDF data
- Scenario **RS**: Purely relational model of the scenario
  - Encoding designed using ERM DB modeling techniques
  - Using flat tables for publications, venues, persons, etc.
  - Queries: semantically equivalent SQL queries on top of the relational model
Settings Summary

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

MonetDB mserver v5.5.0, using the new algebra frontend

Sesame v2.0 coupled with its native SAIL

- Intel2 DuoCore 2.13GHz CPU, 3GB DDR2 RAM, Ubuntu v7.10 gutsy
- Generated Documents: 10k, 50k, 250k, 1M, 5M, and 25M triples
- 30min/query timeout, 2GB main memory limit, report on avg. over 3 runs
Return the year of publication of the journal with the title ‘Journal 1 (1940)’.

SPARQL (original benchmark query)

```
SELECT ?yr
WHERE {
  ?journal dc:title "Journal 1 (1940)".
  ?journal dcterms:issued ?yr
}
```

SQL/VP query without dictionary encoding (marginally modified)

```
SELECT T3.object AS yr
FROM type ty, title ti, issued is
WHERE ty.object="bench:Journal" AND
  ti.object="Journal 1 (1940)" AND
  ty.subject=ti.subject AND
  ti.subject=is.subject
```

SQL/TR query without dictionary encoding (marginally modified)

```
SELECT T3.object AS yr
FROM Triples T1, Triples T2, Triples T3
WHERE T1.predicate="rdf:type" AND
  T1.object="bench:Journal" AND
  T2.predicate="dc:title" AND
  T2.object="Journal 1 (1940)" AND
  T3.predicate="dcterms:issued" AND
  T1.subject=T2.subject AND
  T1.subject=T3.subject
```

All translations and SP²Bench data generator available online at
http://dbis.informatik.uni-freiburg/index.php?project=SP2B
Part III – Experimental Results

#Triples: S1=10k / S2=50k / S3=250k / S4=1M / S5=5M / S6=25M
Part III – Experimental Results

Experimental Results Q4

Select the names of all distinct pairs of article authors that have published in the same journal.

SPARQL (original benchmark query)

SELECT DISTINCT ?name1 ?name2
WHERE {
  ?article1 rdf:type bench:Article.
  ?article2 rdf:type bench:Article.
  ?article1 dc:creator ?author1.
  ?author1 foaf:name ?name1.
  ?article1 swrc:journal ?journal.
  ?article2 swrc:journal ?journal.
  FILTER (?name1<?name2)
}

SQL/Triple Table without dictionary encoding (marginally modified)

SELECT DISTINCT
  T4.object AS name1, T6.object AS name2
FROM Triples T1, Triples T2, ..., Triples T8
WHERE
  T1.predicate="rdf:type" AND
  T1.object="bench:Article" AND
  T2.predicate="rdf:type" AND
  T2.object="bench:Article" AND
  T3.predicate="dc:creator" AND
  T4.predicate="foaf:name" AND
  T5.predicate="dc:creator" AND
  T6.predicate="foaf:name" AND
  T7.predicate="swrc:journal" AND
  T8.predicate="swrc:journal" AND
  T1.subject=T3.subject AND
  T1.subject=T7.subject AND
  T2.subject=T5.subject AND
  T2.subject=T8.subject AND
  T3.object=T4.subject AND
  T5.object=T6.subject AND
  T7.object=T8.object AND
  T4.object<T6.object
#Triples: $S_1=10k$ / $S_2=50k$ / $S_3=250k$ / $S_4=1M$ / $S_5=5M$ / $S_6=25M$
SELECT DISTINCT ?title
WHERE {
  ?class rdfs:subClassOf foaf:Document.
  ?doc2 dcterms:references ?bag2
  OPTIONAL {
    ?class3 rdfs:subClassOf foaf:Document.
    ?bag3 ?member3 ?doc
  }
  OPTIONAL {
    ?class4 rdfs:subClassOf foaf:Document.
    ?bag4 ?member4 ?doc3
  }
  FILTER (!bound(?doc4))
} FILTER (!bound(?doc3))

Encoded as:
Return the titles of all cited papers for which none of the citing papers is not cited.
Return the titles of all papers that have been cited at least once, but not by any paper without citations.

Problem when translating into VP: Unbound predicates require large unions over all predicate tables; in contrast, query can be easily translated into TR scheme.
Part III – Experimental Results

#Triples:   S1=10k / S2=50k / S3=250k / S4=1M / S5=5M / S6=25M
Conclusion

- Optimizers of RDBMs often not laid out for the specific challenges that arise in the context of processing SW data.
- Vertical Partitioning not a general solution: Limitations for queries with unbound predicates, non subject-subject joins, and in general more complex queries.
- Triple Store with \((\text{predicate,subject,object})\) physical sort order often competitive to VP, since data is arranged in the same way on disk.
- Typically gap of one order of magnitude compared to relational data processing yet on small documents, increasing with document size.

New storage schemes and query evaluation approaches necessary, to bring forward the evaluation of SPARQL queries!

Thank you for your attention!

W3C: Resource Description Framework (RDF). http://www.w3.org/RDF/.
Abadi, D.J., Marcus, A., Madden, S., Hollenbach, K.J.: Using the Barton libraries dataset as an RDF benchmark. TR, MIT.
CWI Amsterdam: MonetDB. http://monetdb.cwi.nl/.
Cyganiak, R.: A Relational Algebra for SPARQL. TR, HP Bristol.
Additional Resources

- Benchmark Requirements
- Data generator implementation
- Query characteristics summary
- Distribution of outgoing citations
- Triple table approach with physical 
  \( (\text{subject}, \text{predicate}, \text{object}) \) sort order
- Purely relational scheme
Benchmark Requirements

- **Relevance:** test typical operations within the benchmark domain
- **Scalability:** support tests on different data sizes
- **Portability:** possible execution on different platforms, applicability to different systems
- **Understandability:** since otherwise, it will not be accepted in practice

Data Generator Implementation

- Technical challenges to data generator
- Efficient generation of large data sets (scales linearly to document size, constant memory)
- Deterministic (random functions with fixed seed)
- Incremental data generation
- Platform independent
- Physical Database Size
## Query Characteristics

<table>
<thead>
<tr>
<th>Category</th>
<th>Construct</th>
<th>Q1</th>
<th>Q2</th>
<th>Q3</th>
<th>Q4</th>
<th>Q5</th>
<th>Q6</th>
<th>Q7</th>
<th>Q8</th>
<th>Q9</th>
<th>Q10</th>
<th>Q11</th>
</tr>
</thead>
<tbody>
<tr>
<td>Operators</td>
<td>And</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Union</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Filter</td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Optional</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Solution Modifiers</td>
<td>Distinct</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Limit</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Offset</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Order by</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Data Access</td>
<td>Blank</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Literal</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td></td>
<td>URI</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
</tr>
</tbody>
</table>
Relevance and Understandability

- Data with real-world characteristics

---

Probability for a paper having $x$ citations

- real DBLP
- vs.
- approx. in SP$^2$Bench
**Merge Joins (Triple Table)**

```
SELECT T2.object AS author
FROM Triples T1, Triples T2,
WHERE
  T1.predicate="type" AND
  T1.object="Book" AND
  T2.predicate="author" AND
  T1.subject=T2.subject
```

"Select all book authors"

No efficient join evaluation possible when data is physically sorted by (subject,predicate,object)!
The Relational Scheme RS

- Reference(fk_from, fk_to)
- Document(ID, address, booktitle, ISBN, ...)
- Venue(ID, fk_document, ...)
- Publication(ID, fk_document, ...)
- Editor(fk_person, fk_document)
- Author(fk_person, fk_publication)
- Person(ID, name)
## Physical Database Size (incl. Indizes)

<table>
<thead>
<tr>
<th>#triples in document</th>
<th>SP</th>
<th>TR</th>
<th>VP</th>
<th>RS</th>
</tr>
</thead>
<tbody>
<tr>
<td>10k</td>
<td>3 MB</td>
<td>3 MB</td>
<td>6 MB</td>
<td>4 MB</td>
</tr>
<tr>
<td>50k</td>
<td>14 MB</td>
<td>5 MB</td>
<td>8 MB</td>
<td>5 MB</td>
</tr>
<tr>
<td>250k</td>
<td>69 MB</td>
<td>18 MB</td>
<td>20 MB</td>
<td>13 MB</td>
</tr>
<tr>
<td>1M</td>
<td>277 MB</td>
<td>63 MB</td>
<td>58 MB</td>
<td>42 MB</td>
</tr>
<tr>
<td>5M</td>
<td>1376 MB</td>
<td>404 MB</td>
<td>271 MB</td>
<td>195 MB</td>
</tr>
<tr>
<td>25M</td>
<td>6928 MB</td>
<td>2395 MB</td>
<td>1168 MB</td>
<td>913 MB</td>
</tr>
</tbody>
</table>