A Main Memory Index Structure to Query Linked Data

Olaf Hartig
http://olafhartig.de/foaf.rdf#olaf
@olafhartig

Frank Huber

Database and Information Systems Research Group
Humboldt-Universität zu Berlin
The Issue

- ContactInfoPhillipe (Query No. 36)
- UnsetPropsPhillipe (Query No. 37)
- 2ndDegree1Phillipe (Query No. 38)
- 2ndDegree2Phillipe (Query No. 39)
- IncomingPhillipe (Query No. 40)

<table>
<thead>
<tr>
<th>hit rate</th>
<th>query execution time (in seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0,2</td>
<td>query execution time (in seconds)</td>
</tr>
<tr>
<td>0,4</td>
<td>query execution time (in seconds)</td>
</tr>
<tr>
<td>0,6</td>
<td>query execution time (in seconds)</td>
</tr>
<tr>
<td>0,8</td>
<td>query execution time (in seconds)</td>
</tr>
<tr>
<td>1</td>
<td>query execution time (in seconds)</td>
</tr>
</tbody>
</table>

- no reuse
- given order

ContactInfoPhillipe
- hit rate
- query execution time
- number of query results

UnsetPropsPhillipe
- hit rate
- query execution time
- number of query results

2ndDegree1Phillipe
- hit rate
- query execution time
- number of query results

2ndDegree2Phillipe
- hit rate
- query execution time
- number of query results

IncomingPhillipe
- hit rate
- query execution time
- number of query results
The Issue

Descriptor objects in the query-local dataset after query execution:

- ContactInfoPhillipe (Query No. 36): 172
- UnsetPropsPhillipe (Query No. 37): 533
- 2ndDegree1Phillipe (Query No. 38)
- 2ndDegree2Phillipe (Query No. 39)
- IncomingPhillipe (Query No. 40)
What data structure do we use to physically represent the query-local dataset?
Outline

1. Requirements + Existing Work

2. Data Structures

3. Evaluation
Requirements

• (Consecutively) build and use ad hoc collections of many small sets of RDF triples

• Four main operations:
  • *Find* … matching triples for a triple pattern in all descriptor objects
  • *Add*, *Remove*, *Replace* … descriptor objects

• Support of concurrent access (i.e. isolation)

• *Non*-relevant properties:
  • Querying descriptor objects individually is not necessary
  • No need to write data back to the Web
  • ACID properties not required for complete queries
Requirements

- (Consecutively) build and use ad hoc collections of many small sets of RDF triples
- Four main operations:
  - Find … matching triples for a triple pattern in all descriptor objects
  - Add, Remove, Replace … descriptor objects
- Support of concurrent access (i.e. isolation)

- Non-relevant properties:
  - Querying descriptor objects individually is not necessary
  - No need to write data back to the Web
  - ACID properties not required for complete queries
Existing Work

- Disk based storage solutions for RDF data
  - Unsuitable due to very costly I/O operations
- Main memory based data structures in the literature
  - Focus on a large, single set of RDF triples
  - Optimized for complete graph pattern queries or path queries
- Main memory based data structures in RDF frameworks
  - Focus on Jena, ARQ and NG4J
  - Inefficient (see evaluation)
Outline

1. Requirements + Existing Work

2. Data Structures

3. Evaluation
Hash-Based Index for RDF Data

Logical representation

- Dictionary:
  - Two-way mapping between RDF terms and numerical identifiers

- 6 hash tables:
  - Each hash table contains all ID-encoded triples
  - Efficient support for all types of triple patterns

*Similar to Harth and Decker, 2005*
Hash-Based Index for RDF Data

Logical representation

Physical representation

- **Dictionary:**
  - Two-way mapping between RDF terms and numerical identifiers

- **6 hash tables:**
  - Each hash table contains all ID-encoded triples
  - Efficient support for all types of triple patterns

\[ t_{id} = (id_s, id_p, id_o) \]

*Similar to Harth and Decker, 2005*
Hash-Based Index for RDF Data

- Two-way mapping between RDF terms and numerical identifiers

- 6 hash tables:
  - Each hash table contains all ID-encoded triples
  - Efficient support for all types of triple patterns

\[ t_{id} = (id_s, id_p, id_o) \]

*Similar to Harth and Decker, 2005*
Individual Indexing

- Logical representation
- Physical representation

**Idea:** Index each descriptor object separately

**Implementation of the four operations:**

- *Add, Remove, and Replace* are straightforward
- *Find* requires iterating over all indexes
Individual Indexing

• Idea: Index each descriptor object separately

• Implementation of the four operations:
  • Add, Remove, and Replace are straightforward
  • Find requires iterating over all indexes
Combined Indexing

Logical representation

- Idea: Use a single index for all descriptor objects

- $src$ – maps each triple to a set of descriptor object IDs

Physical representation
Combined Indexing

Logical representation

Physical representation

- **Idea:** Use a single index for all descriptor objects

- $src$ – maps each triple to a set of descriptor object IDs

$t_{id} = (id_s, id_p, id_o)$
Combined Indexing

- Logical representation
- Physical representation

- **Idea:** Use a single index for all descriptor objects
- **src** – maps each triple to a set of descriptor object IDs

\[ t_{id} = (id_s, id_p, id_o) \]

\[ src(t_{id}) = \{,\} \]
Quad Indexing

- Idea: Use a single quad index for all descriptor objects
  - $quad = \text{ID-encoded triple} + \text{descriptor object ID}$

$q = (\text{id}_s, \text{id}_p, \text{id}_o, )$
Outline

1. Requirements + Existing Work

2. Data Structures

3. Evaluation
Experiment Setup

Does this affect the overall execution time for link traversal based query executions?
Experiment Setup

Does this affect the overall execution time for link traversal based query executions?

- Simulation of the Web of Data
  - Linked Data server publishes BSBM dataset (scal. factor: 50)
  - Adjusted BSBM queries link to the simulation server

- Experiment:
  - Sequence of 200 query mixes
  - Reuse of the query-local dataset for the whole sequence
  - IndIR, CombIR, and QuadIR (as presented), engine: SQUIN
  - NamedGraphSetImpl (NG4J/Jena), engine: SemWeb Client
Execution Time

Overall number of descriptors in the queried dataset vs. execution time in seconds for different implementations:
- NG4J (SWCLib)
- IndIR, m=4
- CombIR, m=12
- CombQuadIR, m=12
Execution Time

- NG4J (SWCILib)
- IndIR, m=4
- CombIR, m=12
- CombQuadIR, m=12
Summary

- Three hash index based data structures:
  - Individually indexing
  - Combined indexing
  - Quad indexing

- Findings:
  - A single index improves query performance significantly
  - Smaller load times with quads

- Also for other use cases of ad hoc storing of Linked Data
  - Consecutively retrieved from remote sources
  - Used for immediate local processing
Combined Indexing

Logical representation

- Physical representation

- **Idea:** Use a single index for all descriptor objects
  - *src* – maps each triple to a set of descriptor object IDs

- **Implementation of the four operations requires:**
  - *status* – maps each descriptor object ID to a status (BeingIndexed, Indexed, ToBeRemoved, BeingRemoved)
  - *Find* reports a triple *t* only if: \( \exists d \in src(t) : status(d) = \text{Indexed} \)
Implementation

- Available as Free Software (http://squin.org)
- Hash tables with $n = 2^m$ buckets
- Hash functions:
  - $h_S(i_s, i_p, i_o) = i_s \& \text{bitmask}^m$
  - $h_{SP}(i_s, i_p, i_o) = (i_s \cdot i_p) \& \text{bitmask}^m$
  - etc.
- Which $m$?*
  - $m = 4$ for the individual indexes
  - $m = 12$ for the combined index
  - $m = 12$ for the quad index

*see paper
Experiment Setup

- Comparison without link traversal based query execution
- Compared data structures:
  - Our implementation of IndIR, CombIR, CombQuadIR
  - NamedGraphSetImpl in NG4J (Jena)
  - DatasetGraphMap in ARQ (Jena)
- Berlin SPARQL Benchmark (BSBM)
  - BSBM datasets partitioned into query-local datasets
  - BSBM (v2.0) query mixes executed over these datasets
Required Memory

<table>
<thead>
<tr>
<th>BSBM scaling factor</th>
<th>number of descriptor objects</th>
<th>overall number of triples</th>
</tr>
</thead>
<tbody>
<tr>
<td>50</td>
<td>2,599</td>
<td>22,616</td>
</tr>
<tr>
<td>100</td>
<td>4,178</td>
<td>40,133</td>
</tr>
<tr>
<td>150</td>
<td>5,756</td>
<td>57,524</td>
</tr>
<tr>
<td>200</td>
<td>7,329</td>
<td>75,062</td>
</tr>
<tr>
<td>250</td>
<td>9,873</td>
<td>97,613</td>
</tr>
<tr>
<td>300</td>
<td>11,455</td>
<td>115,217</td>
</tr>
<tr>
<td>350</td>
<td>13,954</td>
<td>137,567</td>
</tr>
<tr>
<td>500</td>
<td>18,687</td>
<td>190,502</td>
</tr>
</tbody>
</table>
Execution Time

<table>
<thead>
<tr>
<th>pc (BSBM scaling factor)</th>
<th>number of descriptor objects</th>
<th>overall number of triples</th>
</tr>
</thead>
<tbody>
<tr>
<td>50</td>
<td>2,599</td>
<td>22,616</td>
</tr>
<tr>
<td>100</td>
<td>4,178</td>
<td>40,133</td>
</tr>
<tr>
<td>150</td>
<td>5,756</td>
<td>57,524</td>
</tr>
<tr>
<td>200</td>
<td>7,329</td>
<td>75,062</td>
</tr>
<tr>
<td>250</td>
<td>9,873</td>
<td>97,613</td>
</tr>
<tr>
<td>300</td>
<td>11,455</td>
<td>115,217</td>
</tr>
<tr>
<td>350</td>
<td>13,954</td>
<td>137,567</td>
</tr>
<tr>
<td>500</td>
<td>18,687</td>
<td>190,502</td>
</tr>
</tbody>
</table>
Load Time

<table>
<thead>
<tr>
<th>pc (BSBM scaling factor)</th>
<th>BSBM scaling factor</th>
<th>number of descriptor objects</th>
<th>overall number of triples</th>
</tr>
</thead>
<tbody>
<tr>
<td>50</td>
<td>2,599</td>
<td>22,616</td>
<td></td>
</tr>
<tr>
<td>100</td>
<td>4,178</td>
<td>40,133</td>
<td></td>
</tr>
<tr>
<td>150</td>
<td>5,756</td>
<td>57,524</td>
<td></td>
</tr>
<tr>
<td>200</td>
<td>7,329</td>
<td>75,062</td>
<td></td>
</tr>
<tr>
<td>250</td>
<td>9,873</td>
<td>97,613</td>
<td></td>
</tr>
<tr>
<td>300</td>
<td>11,455</td>
<td>115,217</td>
<td></td>
</tr>
<tr>
<td>350</td>
<td>13,954</td>
<td>137,567</td>
<td></td>
</tr>
<tr>
<td>500</td>
<td>18,687</td>
<td>190,502</td>
<td></td>
</tr>
</tbody>
</table>
Execution Time

![Graph showing execution time with query mix on the x-axis and execution time in seconds on the y-axis. The graph compares different index structures: NG4J (SWClLib), IndIR, m=4, CombIR, m=12, and CombQuadIR, m=12.](image-url)
These slides have been created by Olaf Hartig

http://olafhartig.de

This work is licensed under a Creative Commons Attribution-Share Alike 3.0 License (http://creativecommons.org/licenses/by-sa/3.0/)