Scalable Semantic Version Control
for Linked Data Management

Claudius Hauptmann, Michele Brocco and Wolfgang Wörndl
Technische Universität München
2nd Workshop on Linked Data Quality
at ESWC 2015 (June 1st, 2015 - Portorož, Slovenia)
Version Control for Linked Data - Goal

- Scalable (reduce disk space-, CPU- and memory consumption, network traffic and disk I/O)
- Semantic (data about versioning accessible via SPARQL queries and OWL reasoners)
Query Types

• Cross-version queries ("Which triples were modified last month that are related to Portorož?"")
• Targeted queries ("What did we know about Portorož in version X ?")
Update Types

• Creation of branches
• Modification of triples (working on a branch)
• Review and correction of modifications
• Commit of modifications (creation of a version)
• Merging of branches
• Deletion of branches
Storage Strategies for Versioned Triples

- Version-based
- Delta-based
- Hybrid
- Hypergraph-based
- Partial Order Index
Delta-Based Storage Schema
Partial On-Demand Reconstruction of Historic Versions

Diagram:

- triple 1
- commit 1
- triple 2
  - +
  - commit 2
  - triple 3
    - -
    - commit 4
    - triple 3
      - +
      - commit 5
Partial On-Demand Reconstruction of Historic Versions

SELECT ?object WHERE {
    GRAPH <http://graph1/branches/branch1> {
    }
}

SELECT ?object WHERE {
    GRAPH ?identifier {
    }
    GRAPH <http://graph1> {
        ?branchCommit versioning:hasPrevious* ?addCommit .
        FILTER NOT EXISTS {
            ?branchCommit versioning:hasPrevious* ?deleteCommit .
            ?deleteCommit versioning:hasPrevious* ?addCommit .
        }
    }
}

Query Engine Optimization

• Arbitrary length path operator is inefficient (many lookup operations)
• Replace slow operators by new operators, that are optimized for partial on-demand version reconstruction
• Use in-memory indices for commits
• Cache indices
Step 1: Loading commits into in-memory index

- Caching for queries on same graph
Step 2: Planning Commit Graph Traversal

• Caching for queries on same graph and same version
Step 3: Loading changes for chunk of triples

• Load relationships between triples and commits:
  • Commit-triple index
  • Triple-commit index
Step 4: Traverse Commit Graph + Test Triples

- Start traversal at commit specified in query
- Get changes for commit from index
  - Check if add or delete
  - Save result for triple
  - Remove triple from indices
- If indices not empty go to next commit and repeat
Evaluation - Dataset

- DBpedia class assertions (version 2014)
- 28,031,852 triples
- 3 datasets: 280,319, 28,032 and 2,804 commits (triples equally distributed over commits, 1 branch, no deletes)
- Base line: query with 3 arbitrary length path operators
- Repeated 100x (base line 10x), no caching
- Test queries:

  SELECT * WHERE { <http://dbpedia.org/resource/Slovenia> ?p ?o }  

  SELECT * WHERE { ?s ?p <http://schema.org/Country> }
## Evaluation - Response Time Query 1 (7 results)

<table>
<thead>
<tr>
<th>#commits</th>
<th>2,804</th>
<th>28,032</th>
<th>280,319</th>
</tr>
</thead>
<tbody>
<tr>
<td>Baseline</td>
<td>30,537 ms</td>
<td>304,474 ms</td>
<td>3,061,018 ms</td>
</tr>
<tr>
<td>Optimized</td>
<td>126 ms</td>
<td>318 ms</td>
<td>3,110 ms</td>
</tr>
<tr>
<td>Loading commits</td>
<td>15.15 ms</td>
<td>162.5 ms</td>
<td>2,352 ms</td>
</tr>
<tr>
<td>Creating Plan</td>
<td>0.65 ms</td>
<td>11.6 ms</td>
<td>178 ms</td>
</tr>
<tr>
<td>Loading changes</td>
<td>14.87 ms</td>
<td>15.6 ms</td>
<td>16 ms</td>
</tr>
<tr>
<td>Traversal</td>
<td>0.39 ms</td>
<td>4.7 ms</td>
<td>60 ms</td>
</tr>
</tbody>
</table>
# Evaluation - Response Time Query 2 (3108 results)

<table>
<thead>
<tr>
<th>#commits</th>
<th>2,804</th>
<th>28,032</th>
<th>280,319</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Baseline</strong></td>
<td>57.009 ms</td>
<td>607.924 ms</td>
<td>OutOfMemoryError</td>
</tr>
<tr>
<td><strong>Optimized</strong></td>
<td>792 ms</td>
<td>1,188 ms</td>
<td>5,910 ms</td>
</tr>
<tr>
<td>Loading commits</td>
<td>10.95 ms</td>
<td>160.6 ms</td>
<td>2,326 ms</td>
</tr>
<tr>
<td>Creating Plan</td>
<td>0.60 ms</td>
<td>11.9 ms</td>
<td>175 ms</td>
</tr>
<tr>
<td>Loading changes</td>
<td>610.57 ms</td>
<td>616.5 ms</td>
<td>649 ms</td>
</tr>
<tr>
<td>Traversal</td>
<td>13.54 ms</td>
<td>160.6 ms</td>
<td>2,000 ms</td>
</tr>
</tbody>
</table>
Conclusion and Outlook

• Conclusion: Delta-based storage strategies can be used for datasets with millions of triples and thousands of versions for targeted queries by partial on-demand version reconstruction and query engine optimization.

• Outlook:
  • Optimization of implementation
  • Evaluation with established benchmarks
  • Evaluation of caching strategies
  • Evaluation with hybrid storage strategies
  • Integration into existing systems (e.g. r43ples)