Enabling fine-grained HTTP caching of SPARQL query results

Gregory Todd Williams
willig4@cs.rpi.edu
@kasei

Jesse Weaver
weavej3@cs.rpi.edu
@jrweave
Overview

• Motivation for (HTTP) caching SPARQL
• Related work
• Modifying RDF indexing structures
• Connecting SPARQL with modified indexes
• Algorithmic sketch
• Evaluation results
Why HTTP caching?

- Standard protocol used for accessing SPARQL endpoints
- Caching support built into protocol
- Caching support widely implemented in client libraries
Caching Motivation

• HTTP provides primitives to support caching: Last-Modified and ETags

• Can these primitives be used for SPARQL query answering?

• Many SPARQL queries are executed over and over\(^1\); caching can make this much more efficient

---

Framework

- Client Cache
- ISP Cache
- SPARQL Endpoint
- Triple Store
Example

GET /sparql?query=...

HTTP/1.1 200 OK
Last-Modified: Mon, 26 Sep 2011 18:00:00 GMT
...

GET /sparql?query=...
If-Modified-Since: Mon, 26 Sep 2011 18:00:00 GMT

HTTP/1.1 304 Not-Modified
Fine-grained?

• Can achieve this behavior with a modification timestamp on the database
• However, this wouldn’t be “fine-grained”
• Desire this behavior even if database has been updated between requests so long as the data of interest hasn’t changed
Example Query 1

# Q1: get names/emails of Eve’s friends
SELECT * WHERE {
    <#eve> foaf:knows ?friend .
    ?friend foaf:name ?name .
}

HTTP/1.1 200 OK
Last-Modified: Mon, 26 Sep 2011 18:00:00 GMT
Example Update

# U1: insert a depiction image
# nothing to do with Eve’s friends' names/emails
INSERT DATA {
  <#eve> foaf:depiction <eve.jpg> .
}

HTTP/1.1 204 No Content
Example Query 2

# Q2: should have exactly the same results as Q1
SELECT * WHERE {
    <#eve> foaf:knows ?friend .
    ?friend foaf:name ?name .
}

HTTP/1.1 304 Not-Modified
Basic Idea

- Query results should appear to be last modified as close to the actual change of “relevant data” as possible (without being earlier)
- Allows caches to use previous results so long as the relevant data hasn’t changed
Related Work

- Martin, Unbehauen, and Auer (ESWC2010\textsuperscript{2}) developed caching system for SPARQL

What is ‘relevant data’?

- **Triple** patterns
- (**Quad** patterns)
- (**Graph** names)
What is ‘relevant data’?

• **Triple patterns**

• *(Quad patterns)*

• *(Graph names)*

```sql
SELECT * WHERE {
  <#eve> rdf:type foaf:Person .
  GRAPH <SocialNetwork> {
    <#eve> foaf:knows ?friend 
  }
  GRAPH ?g {
  }
}
```
What is ‘relevant data’?

• **Triple** patterns
• (**Quad** patterns)
• (**Graph** names)

```
SELECT * WHERE {
  <#eve> rdf:type foaf:Person .
  GRAPH <SocialNetwork> {
    <#eve> foaf:knows ?friend 
  }
  GRAPH ?g {
    ?friend foaf:name ?name ;
    foaf:mbox ?email .
  }
}
```

<table>
<thead>
<tr>
<th>subject</th>
<th>predicate</th>
<th>object</th>
<th>graph</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;#eve&gt;</td>
<td>rdf:type</td>
<td>foaf:Person</td>
<td>(default)</td>
</tr>
<tr>
<td>&lt;#eve&gt;</td>
<td>foaf:knows</td>
<td>?friend</td>
<td>&lt;SocialNetwork&gt;</td>
</tr>
<tr>
<td>?friend</td>
<td>foaf:name</td>
<td>?name</td>
<td>?g</td>
</tr>
<tr>
<td>?friend</td>
<td>foaf:mbox</td>
<td>?email</td>
<td>?g</td>
</tr>
</tbody>
</table>
How can we recognize “relevant data”? 
Search trees in semantic web systems

- Use of search trees to index RDF data is widespread (notable exceptions: true graph databases, bitmap indexes)
- Allow efficient access for range of access patterns
- [YARS] uses six indices that cover all quad access patterns: spog, pog, ogs, gsp, gp, os
- [Hexastore], [RDF-3X] use six indices to cover all triple access patterns and all orderings: spo, sop, pso, pos, osp, ops
- [4store] trades full indexing for storage space/simplicity: g, ps, po
Search Tree Augmentation

• Modify search tree indexes to maintain modification time (mtime) for each tree node

• Update nodes on insertion/deletion to ensure mtime of a node is greater than or equal to the mtime of all of its descendants

• Use mtime of lowest common ancestor (LCA) of tree items to indicate “effective” mtime of relevant data
mtime update algorithm

- When an item is inserted or deleted, update the leaf node mtime
- Propagate mtime updates to root
- This has associated cost, but does not change complexity and is “free” for counted or append-only B+ trees
B+ Trees

Triples:
1. a p x
2. a p y
3. a q z
4. x q w
5. x p a
Triples:
① a p x
② a p y
③ a q z
④ x q w
⑤ x p a
B+ Trees

Triples:
① a p x
② a p y
③ a q z
④ x q w
⑤ x p a
B+ Trees

Triples:

① a p x
② a p y
③ a q z
④ x q w
⑤ x p a

③ a p b
④ a q x
⑤ b p a
B+ Trees

Triples:
1. a p x
2. a p y
3. a q z
4. x q w
5. x p a

Diagram:
```
   a q x  b p c
   .  .  .  .

   a p b  a q x  b p a  b p c
   4  4
```

Tuesday, October 25, 11
B+ Trees

Triples:
1. a p x
2. a p y
3. a q z
4. x q w
5. x p a

Diagram:

```
  1         2
 a p x a p y
  3         4
 a q z x q w
  5
```

```
  4
 a p b a q x
  5
 b p a b p c c q z
```
B+ Trees

\[ \text{a q x} \quad \text{c q z} \]

\[ \text{a p b} \quad \text{a q x} \quad \text{b p a} \quad \text{b p c} \quad \text{c q z} \]
B+ Trees

• Pattern: { <a> <p> ?o }

```
<table>
<thead>
<tr>
<th>a</th>
<th>q</th>
<th>x</th>
</tr>
</thead>
<tbody>
<tr>
<td>b</td>
<td>p</td>
<td>a</td>
</tr>
</tbody>
</table>
```

```
<table>
<thead>
<tr>
<th>c</th>
<th>q</th>
<th>z</th>
</tr>
</thead>
<tbody>
<tr>
<td>b</td>
<td>p</td>
<td>c</td>
</tr>
</tbody>
</table>
```

Tuesday, October 25, 11
B+ Trees

- Pattern: \{ <a> <p> ?o \} 
- mtime of LCA is 4
B+ Trees

• Pattern: \{ <a> <p> ?o \} \\
• mtime of LCA is 4
B+ Trees

- Pattern: \{ <a> <p> ?o \} 
- mtime of LCA is 4
- Updates are certainly irrelevant after time 4
B+ Trees

• Pattern: \{ <a> <p> ?o \} \\
• mtime of LCA is 4 \\
• Updates are certainly irrelevant after time 4 \\
• (updates are actually irrelevant after time 2)
Choices...

- Choice of search tree type affects mtime precision (e.g. using tries can improve precision)
- For B+ trees, trade-off between branching factor and mtime precision
- Effective mtime (mtime of LCA) will always be safe w.r.t. caching
Query algorithm

- Extract list of access operations (triple patterns) from query
- For each access operation
  - Choose appropriate index for access
  - Probe index for LCA of all nodes containing matching data
  - Retrieve mtime of LCA
- Effective mtime of relevant data is max mtime over all access operations
- Compare effective mtime with cache validator (If-Modified-Since header), or return it as Last-Modified value
Evaluation

• Implemented prototype SPARQL system with 6 mtime-augmented, covering B+ tree indexes

• Used modified Berlin SPARQL Benchmark (BSBM) with a non-uniform (Pareto; power-law) query distribution

• Modified BSBM test driver to support caching headers and maintain results cache
Read-only Throughput

BSBM Explore (Percent Change, QMpfH)

alpha

1MT

5MT

1 0.1 0.3 0.5 1 2 4

1 10 100 1000
Cost of Caching Code

QMpH, BSBM Explore+Update 5MT

alpha

0.1 0.3 0.5 1 2 4

No caching  Cache unused in client
### Comparison with previous work

<table>
<thead>
<tr>
<th></th>
<th>Our work</th>
<th>Martin, et al.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Coupling</td>
<td>high</td>
<td>low</td>
</tr>
<tr>
<td>Portable</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Decentralized</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Lightweight cache</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>management</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Lightweight query</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>performance</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Summary

- Relatively minor changes to RDF indexing structures can enable HTTP-level caching of query results.
- Fine-grained nature of indexing can allow high-precision.
- On non-uniform workloads, SPARQL results caching can provide significant performance improvements.
Future Work

• Larger evaluation with varied bandwidth/latency and more accurate query distribution

• Maintaining mtime under different entailment regimes

• Integrate caching into more fully-featured SPARQL implementation
Thank You

Questions?