Large-scale Linked Data Management

Marko Grobelnik, Andreas Harth (Günter Ladwig), Dumitru Roman
Big Linked Data Tutorial Semantic Days 2012
Large-Scale Linked Data Management (Andreas)

- Motivation
- Preliminaries
  - Apache Cassandra
  - CumulusRDF
- Storage Layouts
  - Storage Model
  - Hierarchical Layout
  - Flat Layout
- Evaluation
- Conclusion
MOTIVATION
Linked Data Storage and Retrieval at Scale

Size

TB

Distributed

GB

Single machine

MB

Index Lookups

SPARQL

Reasoning

Algorithmic complexity

CumulusRDF (Apache Cassandra)

CloudSPARQL

BigData, 4Store, YARS2

SAOR

Jena TBD

Sesame

OWLIM

Pellet, HermiT

Marko Grobelnik, Andreas Harth (Günter Ladwig), Dumitru Roman, Big Linked Data
Linked Data Management

- RDF data accessible via HTTP lookups
- Many datasets cover descriptions of millions of entities
- Publishers often use full-fledged triple stores
  - Complex query processing capabilities not necessary for Linked Data lookups
- Trend towards specialized data management systems tailored for specific use cases
- Distributed key-value stores
  - Simple (often nested) data model
  - No (expensive) joins
  - High availability and scalability
- We investigate applicability of key-value stores for managing and publishing Linked Data
Linked Data Lookups

- Dereferencing URI $t$ should return RDF graph describing $t$
  - Exact content is only lightly specified
- Common practice (e.g. DBpedia) is to return
  - all triples with the given URI as subject and
  - some triples with the given URI as object
- Other options
  - Only triples with the given URI as subject
  - Concise Bounded Descriptions

User Agent

Server

GET RDF

GET RDF

http://www.bbc.co.uk/music/artists/191cba6a-b83f-49ca-883c-02b20c7a9dd5#artist

http://www.bbc.co.uk/music/artists/191cba6a-b83f-49ca-883c-02b20c7a9dd5.rdf
Triple Patterns

- A triple pattern is an RDF triple that may contain variables instead of RDF terms in any position
  
  ?s dbpprop:birthPlace dbpedia:Karlsruhe .
  
  or
  
  ?s foaf:name ?o .
  
- Linked Data Lookup on $t$ translates into two triple patterns lookups
  
  - $(t \ ? \ ?)$
  
  - $(? \ ? \ t)$

- At least three indexes to cover all possible triple patterns (with prefix lookups)

<table>
<thead>
<tr>
<th>Patterns</th>
<th>Index</th>
</tr>
</thead>
<tbody>
<tr>
<td>$? \ ? \ ?$</td>
<td>Any</td>
</tr>
<tr>
<td>$s \ ? \ ?$</td>
<td>SPO</td>
</tr>
<tr>
<td>$? \ p \ ?$</td>
<td>POS</td>
</tr>
<tr>
<td>$? \ o \ ?$</td>
<td>OSP</td>
</tr>
<tr>
<td>$s \ p \ ?$</td>
<td>SPO</td>
</tr>
<tr>
<td>$? \ p \ o$</td>
<td>POS</td>
</tr>
<tr>
<td>$s \ ? \ o$</td>
<td>OSP</td>
</tr>
<tr>
<td>$s \ p \ o$</td>
<td>Any</td>
</tr>
</tbody>
</table>
Apache Cassandra

- Open source data management system
- Distributed key-value store (DHT-based)
  - Nested key-value data model
  - Schema-less
- Decentralized
  - Every node in the cluster has the same role
  - No single point of failure
- Elastic
  - Throughput increases linearly as machines are added with no downtime
- Fault-tolerant
  - Data can be replicated
CumulusRDF

Client
Client
Client
...
Client

Application Server (Tomcat)

HTTP

CumulusRDF Webapp

Thrift

Cassandra Nodes

Cassandra
Storage

Cassandra
Storage

Cassandra
Storage
CumulusRDF Functionality

- Distributed deployment to enable scale (more data and also more clients) by adding more machines (via Cassandra)

- Geographical replication (via Cassandra)

- Write-optimised indices with eventual consistency (via Cassandra)

- Triple pattern lookups (via CumulusRDF index structures)

- Linked Data Lookups (via CumulusRDF index structures)
STORAGE LAYOUTS
**Nested Key-Value Storage Model**

- **Column-only** { row-key : { column : value } }

- **Super columns** { row-key : { supercolumn : { column : value } } }

<table>
<thead>
<tr>
<th>Row</th>
<th>Column key</th>
<th>Column value</th>
</tr>
</thead>
<tbody>
<tr>
<td>$r_0$</td>
<td>$c_{00}$</td>
<td>$v_{00}$</td>
</tr>
<tr>
<td></td>
<td>$c_{01}$</td>
<td>$v_{01}$</td>
</tr>
<tr>
<td></td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>$r_1$</td>
<td>$c_{10}$</td>
<td>$v_{10}$</td>
</tr>
<tr>
<td></td>
<td>$c_{11}$</td>
<td>$v_{11}$</td>
</tr>
<tr>
<td></td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Row</th>
<th>Super column key</th>
<th>Column key</th>
<th>Column value</th>
</tr>
</thead>
<tbody>
<tr>
<td>$r_2$</td>
<td>$sc_{00}$</td>
<td>$c_{000}$</td>
<td>$v_{000}$</td>
</tr>
<tr>
<td></td>
<td>$sc_{01}$</td>
<td>$c_{010}$</td>
<td>$v_{010}$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>$r_3$</td>
<td>$sc_{00}$</td>
<td>$c_{000}$</td>
<td>$v_{000}$</td>
</tr>
<tr>
<td></td>
<td>$sc_{01}$</td>
<td>$c_{010}$</td>
<td>$v_{010}$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
Nested Key-Value Storage Model

- Secondary indexes map column values to rows
  - \{ value : row-key \}

Cassandra limitations

- Entire rows always stored on a single node
- No range queries on row keys
- Columns are stored in specified order and allow for range queries
Hierarchical Layout

- Uses super columns
- RDF terms occupy row, supercolumn and column positions
  - Value is empty
- Three indexes SPO, POS, OSP cover all possible triple pattern
- Example: SPO index

  ```
  SPO: \{ s: \{ p: \{ o: - \} \} \}
  ```

<table>
<thead>
<tr>
<th>Row key</th>
<th>Super column key</th>
<th>Column key</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>dbp:Jaws</code></td>
<td>foaf:name</td>
<td>“Jaws”</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>rdf:type</td>
<td><code>dbp:Film</code></td>
<td>-</td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>dbp:Work</code></td>
<td>-</td>
</tr>
</tbody>
</table>
Flat Layout

- Uses columns only
  - Range queries on column keys allow **prefix lookups**
- Concatenate second & third position to form column key
  - SPO \( \{ s : \{ po : - \} \} \)
  - po is the concatenation of predicate and object
  - For \((sp?)\) we perform a prefix lookup on \(p\) in row with key \(s\)

<table>
<thead>
<tr>
<th>dbp:Jaws</th>
<th>foaf:name “Jaws”</th>
<th>-</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>rdf:type dbp:Film</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>rdf:type dbp:Work</td>
<td>-</td>
</tr>
</tbody>
</table>

Row key | Column key | Value
POS Index

- RDF data is skewed: many triples may share the same predicate (\texttt{rdf:type} is a prime example)
  - \texttt{p} as row key will result in a very uneven distribution
  - Cassandra cannot split rows among several nodes
- We take advantage of Cassandra’s secondary indexes
- Use \texttt{po} as row key
  - \{ \texttt{po} : \{ \texttt{s} : - \} \}
  - Smaller rows, better distribution
  - No range queries on rows key: no prefix lookup!
- In each row we add a special column ‘\texttt{p}’ which has \texttt{p} as its value
  - \{ \texttt{po} : \{ ‘\texttt{p}’ : \texttt{p} \} \}
- Secondary index on column ‘\texttt{p}’ allows retrieval of all \texttt{po} row keys for a given \texttt{p}
EVALUATION
Evaluation

- System: 4 node cluster on virtualized infrastructure
  - 2 CPUs, 4GB RAM, 80GB disk per node
- Dataset: DBpedia 3.6 subset
  - 120M triples (all w/o multilingual labels)
- Triple pattern queries
  - 1M sampled S, SP, SPO, SO, and O patterns from dataset
  - Output: all matching triples
- Linked Data lookup queries
  - 2M resource lookups from DBpedia logs (1.2M unique)
  - Output: all triples with URI as subject and 10k triples with URI as object

Marko Grobelnik, Andreas Harth (Günter Ladwig), Dumitru Roman, Big Linked Data
## Results – Storage Layout

<table>
<thead>
<tr>
<th>Index</th>
<th>Node 1</th>
<th>Node 2</th>
<th>Node 3</th>
<th>Node 4</th>
<th>Std. Dev.</th>
<th>Max. Row</th>
</tr>
</thead>
<tbody>
<tr>
<td>SPO Hier</td>
<td>4.41</td>
<td>4.40</td>
<td>4.41</td>
<td>4.41</td>
<td>0.01</td>
<td>0.0002</td>
</tr>
<tr>
<td>SPO Flat</td>
<td>4.36</td>
<td>4.36</td>
<td>4.36</td>
<td>4.36</td>
<td>0.00</td>
<td>0.0004</td>
</tr>
<tr>
<td>OSP Hier</td>
<td>5.86</td>
<td>6.00</td>
<td>5.75</td>
<td>6.96</td>
<td>0.56</td>
<td>1.16</td>
</tr>
<tr>
<td>OSP Flat</td>
<td>5.66</td>
<td>5.77</td>
<td>5.54</td>
<td>6.61</td>
<td>0.49</td>
<td>0.96</td>
</tr>
<tr>
<td>POS Hier</td>
<td>4.43</td>
<td>3.68</td>
<td>4.69</td>
<td>1.08</td>
<td>1.65</td>
<td>2.40</td>
</tr>
<tr>
<td>POS Sec</td>
<td>7.35</td>
<td>7.43</td>
<td>7.38</td>
<td>8.05</td>
<td>0.33</td>
<td>0.56</td>
</tr>
</tbody>
</table>

Values in GB

- **SPO Flat**: \{ s : { po : - } \}, OSP
- **POS Sec**: \{ po : \{ ‘p’ : p \} \}
- **SPO Hier**: \{ s : { p : { o : - } } \}, OSP, POS
Results – Pattern Lookups

![Graph showing performance of Cumulus-RDF with flat and hierarchical structures across different client numbers.]

Marko Grobelnik, Andreas Harth (Günter Ladwig), Dumitru Roman, Big Linked Data
Results – Pattern Lookups

The diagram shows the response time [msec] for different patterns under two conditions: Cumulus RDF Flat and Cumulus RDF Hierarchical. The patterns are labeled as S, O, SP, SO, and SPO. The response times are represented as bars with error bars indicating variability. The red bars represent the Cumulus RDF Flat condition, while the green bars represent the Cumulus RDF Hierarchical condition.
Results – Linked Data Lookups

![Graph showing performance comparison between CumulusRDF N-Triples and CumulusRDF RDF/XML under varying numbers of clients.](image)

- **CumulusRDF N-Triples**
- **CumulusRDF RDF/XML**

Marko Grobelnik, Andreas Harth (Günter Ladwig), Dumitru Roman, Big Linked Data
Conclusion

- We evaluated two index schemes for RDF on nested key-value stores to support Linked Data lookups
  - Flat indexing gives best overall results
  - Output format impacts performance (N-Triples v RDF/XML)
- Apache Cassandra is a viable alternative to full-fledged triple stores for Linked Data lookups
- Future work
  - Automatic generation and maintenance of dataset statistics
  - Evaluate insert and update performance
- Get CumulusRDF at [http://code.google.com/p/cumulusrdf](http://code.google.com/p/cumulusrdf)