Sempala

Interactive SPARQL Query Processing on Hadoop

Alexander Schätzle, Martin Przyjaciel-Zablocki, Antony Neu, Georg Lausen
University of Freiburg, Germany

ISWC 2014 - Riva del Garda, Italy
Motivation

- **Semantic Web has arrived in real-world applications**
  (not only academia & research)
  - Web-scale semantic data makes single machine solutions infeasible

- **One Solution:** Design a specialized clustered infrastructure
  - E.g. Virtuoso, 4store, ...

- **Our Idea:**
  - Hadoop ecosystem has become de-facto standard for Big Data applications
  - Why not use it for Semantic Web purposes as well?
  - **2 main reasons:**
    1) Web-scale semantic data requires solutions that scale out
    2) Industry has settled on Hadoop (or Hadoop-style) architectures

  ➞ superior cost-benefit ratio compared to specialized infrastructures
Motivation

- **SPARQL-on-Hadoop**
  - Existing solutions are mostly based on MapReduce
  - Scale very well to billions of triples (or more)
  - But MapReduce is batch and thus inherently slow
  - Good for unselective ETL-like queries with many results

- **SPARQL queries are often explorative and ad-hoc**
  - Typically selective thus returning only a few results
    - Runtimes in the order of hours are not satisfying!
  - But interactive runtimes are virtually impossible to achieve with MapReduce

- **Need for interactive SPARQL-on-Hadoop**
  - Following the trend of interactive SQL-on-Hadoop
### SPARQL-on-Hadoop query engine

- Especially designed with **ad-hoc style selective** queries in mind
- Built on top of **Impala**, an MPP SQL query engine for Hadoop
- RDF data stored in HDFS using columnar storage format (**Parquet**)

![Diagram of Sempala processing flow](image.png)
Single Unified Property Table

- Parquet is a column-oriented storage format ➔ optimized for wide tables while selecting only a few columns on request
- Sparse columns cause little to no storage overhead in Parquet
- No clustering, no joins for star pattern queries, eases query formulation
- Duplication strategy for multi-valued properties
- **Single Unified Property Table**
  - Parquet is a *column-oriented* storage format
    → optimized for wide tables while selecting only a few columns on request
  - *Sparse columns* cause little to no storage overhead in Parquet
  - No clustering, no joins for *star pattern* queries, eases query formulation
  - Duplication strategy for multi-valued properties

```
<table>
<thead>
<tr>
<th>subject</th>
<th>author: string</th>
<th>title: string</th>
<th>pages: int</th>
<th>cite: string</th>
<th>erdoesNr: int</th>
</tr>
</thead>
<tbody>
<tr>
<td>Article1</td>
<td>Paul_Erdoes</td>
<td>&quot;Title 1&quot;</td>
<td>12</td>
<td>文章1</td>
<td>1</td>
</tr>
<tr>
<td>Article1</td>
<td>Alice</td>
<td>&quot;Title 1&quot;</td>
<td>12</td>
<td>文章1</td>
<td>1</td>
</tr>
<tr>
<td>Article2</td>
<td>Paul_Erdoes</td>
<td>&quot;Title 2&quot;</td>
<td>8</td>
<td>文章2</td>
<td>2</td>
</tr>
<tr>
<td>Paul_Erdoes</td>
<td>Alice</td>
<td>(&quot;Title 1&quot;, 2)</td>
<td>(12, 2)</td>
<td>(Article2, 2)</td>
<td>0 1</td>
</tr>
</tbody>
</table>
```

- run-length encoding
- + compression
- + dictionary encoding
Sempala – Query Compiler

- Overall workflow from SPARQL to Impala SQL

```sparql
SELECT * 
  OPTIONAL { ?a :erdosNr ?e FILTER(?e < 3) }
  FILTER(?p > 10) }
ORDER BY DESC(?p)
```
Overall workflow from SPARQL to Impala SQL

```
SELECT *
    OPTIONAL { ?a :erdoesNr ?e FILTER(?e < 3) }
    FILTER(?p > 10) }
ORDER BY DESC(?p)
```

1. Algebra Tree

2. Filter
   - ?p > 10

3. BGP
   - ?s :pages ?p
   - ?s :author ?a

4. BGP
   - ?a :erdoesNr ?e

5. Left Join
   - ?e < 3

6. ORDER BY
   - DESC ?p

7. ORDER BY
   - DESC ?p
Sempala – Query Compiler

- Overall workflow from SPARQL to Impala SQL

SPARQL

```
SELECT *
    OPTIONAL { ?a :erdosesNr ?e FILTER(?e < 3) }
    FILTER(?p > 10) }
ORDER BY DESC(?p)
```

Algebra Tree

```
ORDER BY
DESC ?p
```

```
Left Join
?e < 3
```

```
Filter
?p > 10
```

```
BGP
?a :erdosesNr ?e
```

Impala SQL

```
CREATE TABLE result AS SELECT q1.s, q1.p, q1.a, q2.e
FROM ( SELECT DISTINCT subject AS s, pages AS p, author AS a FROM propTable
    WHERE author IS NOT NULL AND pages > 10 ) q1
    LEFT OUTER JOIN
    ( SELECT DISTINCT subject AS a, erdoesNr AS e FROM propTable
    WHERE erdoesNr IS NOT NULL ) q2
ON ( q1.a = q2.a AND q2.e < 3 )
ORDER BY p DESC
```
BPG processing in Sempala

- Decompose BGP into disjoint triple groups having the same subject
- Use a join-free subquery for every triple group and join the results (join group)

```
?s title ?t .
?s cite ?c .
?c author Paul_Erdos .
?c pages ?p
```
BPG processing in Sempala

- Decompose BGP into disjoint triple groups having the same subject
- Use a join-free subquery for every triple group and join the results (join group)

\[
\text{SPARQL:}
\begin{align*}
&s \text{ title } ?t . \\
&s \text{ cite } ?c . \\
?c \text{ author Paul_Erdos .} \\
?c \text{ pages } ?p
\end{align*}
\]

- triple group \(tg_1\)
- triple group \(tg_2\)
BPG processing in Sempala

- Decompose BGP into disjoint triple groups having the same subject
- Use a join-free subquery for every triple group and join the results (join group)
**BPG processing in Sempala**

- Decompose BGP into disjoint **triple groups** having the same subject
- Use a join-free **subquery** for every triple group and join the results (**join group**)

---

**SPARQL**

```
?s title ?t .
?s cite ?c .
?c author Paul_Erdoes .
?c pages ?p
```

**Impala SQL**

```
SELECT DISTINCT sub AS s, title AS t, cite AS c FROM propTable
WHERE title IS NOT NULL AND cite IS NOT NULL
```

```
SELECT DISTINCT sub AS c, pages AS p FROM propTable
WHERE author = 'Paul_Erdoes' AND pages IS NOT NULL
```
**BPG processing in Sempala**

- Decompose BGP into disjoint *triple groups* having the same subject
- Use a join-free *subquery* for every triple group and join the results (join group)

**SPARQL**

\[
\text{SELECT } q1.s, q1.t, q2.c, q2.p \text{ FROM}
( \text{SELECT DISTINCT sub AS } s, \text{ title AS } t, \text{ cite AS } c \text{ FROM propTable}
  \text{ WHERE title IS NOT NULL AND cite IS NOT NULL } ) q1
\text{ JOIN}
( \text{SELECT DISTINCT sub AS } c, \text{ pages AS } p \text{ FROM propTable}
  \text{ WHERE author = 'Paul_Erdoes' AND pages IS NOT NULL } ) q2
\text{ ON ( } q1.c = q2.c \text{ )}
\]
Evaluation

- Small Cluster with low-end configuration
  - 10 machines, 32 GB RAM and 2 disks each, Gigabit network (Cloudera recommends 256 GB RAM and 12 disks or more)
  - CDH 4.5, Impala 1.2.3

- LUBM and BSBM benchmarks
  - LUBM up to 3K universities
  - BSBM up to 3M products

- Compared Sempala with 4 other Hadoop based systems
  - Hive (same Query Compiler as Sempala but Hive as execution engine)
  - PigSPARQL (built on top of Apache Pig) [1]
  - MapMerge (map-side merge join for SPARQL BGPs) [2]
  - MAPSIN (map-side index nested loop join based on Apache HBase) [3]
Evaluation

- LUBM 3K (sec in log scale)
Evaluation

LUBM 3K (sec in log scale)

- Geometric Mean for selective (star-shaped) queries:
  - Sempala (8.3), Hive (89), PigSPARQL (69.7), MAPSIN (78.1), MapMerge (65.8)
Evaluation

- LUBM 3K (sec in log scale)

- Geometric Mean for more sophisticated queries:
  - Sempala (20.7), Hive (316.6), PigSPARQL (266), MAPSIN (119.5), MapMerge (175.6)
Summary

- **Sempala SPARQL query engine for Hadoop**
  - Built on top of a state-of-the-art MPP SQL query engine (**Impala**)
  - Uses a state-of-the-art columnar storage format (**Parquet**)
  - **SPARQL 1.0** (not only BGPs)

- **Evaluation**
  - Outperforms existing Hadoop based systems by an **order of magnitude**
  - Interactive runtimes for **selective queries** (selective ≠ simple)
    (order of seconds, not minutes or even hours)

- **Future Work**
  - Refinement of data layout (nested data structures → Impala 2.1)
  - **SPARQL 1.1** features (e.g. subqueries, aggregations)


Evaluation

- **LUBM 3K (sec in log scale)**

![Graph showing query execution times](image)

- **Geometric Mean for unselective queries:**
  - Sempala (63.5), Hive (166), PigSPARQL (52.4), MAPSIN (101.4), MapMerge (24.5)
  - Runtime of Sempala dominated by storing millions of records in result table