S2RDF

RDF Querying with SPARQL on Spark

Alexander Schätzle, Martin Przyjaciel-Zablocki, Simon Skilevic, Georg Lausen
University of Freiburg, Germany

schaetzle@informatik.uni-freiburg.de
http://dbis.informatik.uni-freiburg.de/S2RDF

VLDB 2016 – New Delhi, India
06.09.2016
Motivation

- **Semantic Web has arrived in real-world applications**
  (not only academia & research)
  - Web-scale semantic datasets require scalable querying capabilities

- **Hadoop has become de-facto standard**
  for Big Data applications
  - Great potential for synergy benefits
    (i.e. RDF as "just another data source")
  - However, Hadoop lacks native support
    to process and query semantic data (RDF)

- **Challenges for SPARQL-on-Hadoop:**
  - Not designed for RDF querying from scratch
  - Lack of rich indexes (performance relies on scale-out parallelism)

  ➡️ Query performance is a major issue
RDF & SPARQL

RDF (Graph)

```turtle
@base <http://ex.org/resource> .
@prefix : <http://ex.org/property> .

<A> :follows <B> .
<B> :follows <C> .
<B> :follows <D> .
<C> :follows <D> .
<C> :likes  <I2> .
<A> :likes  <I1> .
<A> :likes  <I2> .
```

SPARQL

```sparql
prefix : <http://ex.org/property>

SELECT * WHERE {
?x :likes  ?w .
?x :follows ?y .
?y :follows ?z .
?z :likes  ?w 
}
```

mapping

<table>
<thead>
<tr>
<th>?x</th>
<th>?y</th>
<th>?z</th>
<th>?w</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;A&gt;</td>
<td>&lt;B&gt;</td>
<td>&lt;C&gt;</td>
<td>&lt;I2&gt;</td>
</tr>
</tbody>
</table>
Relational Representation of RDF

Triples Table (TT)

```spoc
@base <http://ex.org/resource> .
@prefix : <http://ex.org/property> .

<A> :follows <B> .
<B> :follows <C> .
<B> :follows <D> .
<C> :follows <D> .
<A> :likes <I1> .
<A> :likes <I2> .
```

Vertical Partitioning

```spoc
@base <http://ex.org/resource> .
@prefix : <http://ex.org/property> .

<VP_likes>
S   O
A   I1
A   I2
C   I2

<VP_follows>
S   O
A   B
A   C
B   D
C   D
```

Problem:
No (or limited) support for indexes in Hadoop! → very inefficient

Mimics the effect of an index on predicate (p)

Problem:
Skewed distribution → small vs. large tables
Vertical Partitioning (VP)

SELECT * WHERE {
  ?x :likes ?w .
  ?x :follows ?y .
  ?y :follows ?z .
  ?z :likes ?w }

Vertical Partitioning (VP)

VP_{follows}

<table>
<thead>
<tr>
<th>S</th>
<th>O</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>B</td>
</tr>
<tr>
<td>B</td>
<td>C</td>
</tr>
<tr>
<td>B</td>
<td>D</td>
</tr>
<tr>
<td>C</td>
<td>D</td>
</tr>
</tbody>
</table>

VP_{likes}

<table>
<thead>
<tr>
<th>S</th>
<th>O</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>I1</td>
</tr>
<tr>
<td>A</td>
<td>I2</td>
</tr>
<tr>
<td>C</td>
<td>I2</td>
</tr>
</tbody>
</table>

dangling tuples

dangling tuples

dangling tuples
Extended Vertical Partitioning (ExtVP)

Key Idea of ExtVP

\[ \text{VP}_{\text{follows}} \cdot o = \text{VP}_{\text{likes}} \cdot s \]

4

3

\[ \text{VP}_{\text{follows}} \equiv \text{VP}_{\text{likes}} \]

no dangling tuples!

1

1

no dangling tuples!

\[ \text{ExtVP}^{OS}_{\text{follows}} | \text{likes} \]

\[ \text{ExtVP}^{SO}_{\text{likes}} | \text{follows} \]
ExtVP – Correlations

Correlations

\[
\begin{array}{ccc}
V_P^{\text{follows}} & & \\
S & O &  \\
A & B &  \\
B & C &  \\
B & D &  \\
C & D &  \\
\end{array}
\]

\[
\begin{array}{ccc}
V_P^{\text{likes}} & & \\
S & O &  \\
A & I1 &  \\
A & I2 &  \\
C & I2 &  \\
\end{array}
\]

\[
\begin{array}{ccc}
?x & \text{follows} & ?y \\
?y & \text{likes} & ?z \\
\end{array}
\]

\[
?x \ \text{follows} \ ?y \ \text{likes} \ ?z
\]

ExtVP

\[
\begin{align*}
V_P^{\text{follows}} & \times V_P^{\text{likes}} \\
O = S & \\
\end{align*}
\]

\[
\begin{align*}
E_x t V_P^{OS} & \text{follows | likes} \\
\end{align*}
\]

\[
\begin{align*}
V_P^{\text{likes}} & \times V_P^{\text{follows}} \\
S = O & \\
\end{align*}
\]

\[
\begin{align*}
E_x t V_P^{SO} & \text{likes | follows} \\
\end{align*}
\]

OS

SO

SS

OO
ExtVP – Correlations

Correlations

<table>
<thead>
<tr>
<th>S</th>
<th>O</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>B</td>
</tr>
<tr>
<td>B</td>
<td>C</td>
</tr>
<tr>
<td>B</td>
<td>D</td>
</tr>
<tr>
<td>C</td>
<td>D</td>
</tr>
</tbody>
</table>

ExtVP follows

VP follows

OS

| VP follows | \( o = s \) |
\[
\text{ExtVP}^{\text{OS}}_{\text{follows} | \text{likes}}
\]

SO

| VP likes | \( s = o \) |
\[
\text{ExtVP}^{\text{SO}}_{\text{likes} | \text{follows}}
\]

SS

| VP follows | VP likes |
\[
\text{ExtVP}^{\text{SS}}_{\text{follows} | \text{likes}}
\]

OO

?x follows ?y

?z likes ?x

?y follows ?x

likes

?z
ExtVP – Correlations

Correlations

<table>
<thead>
<tr>
<th>S</th>
<th>O</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>B</td>
</tr>
<tr>
<td>B</td>
<td>C</td>
</tr>
<tr>
<td>B</td>
<td>D</td>
</tr>
<tr>
<td>C</td>
<td>D</td>
</tr>
</tbody>
</table>

VP\_follows

?x follows ?y

VP\_likes

?x likes ?z

?y follows ?x likes ?z

S2RDF: RDF Querying with SPARQL on Spark

ExtVP\_OS

VP\_follows \(\times\) VP\_likes

\(o = s\)

\(ExtVP\_OS\) follows | likes

ExtVP\_SO

VP\_likes \(\times\) VP\_follows

\(s = o\)

\(ExtVP\_SO\) likes | follows

ExtVP\_SS

VP\_follows \(\times\) VP\_likes

\(s = s\)

\(ExtVP\_SS\) follows | likes

ExtVP\_OO

VP\_likes \(\times\) VP\_follows

\(s = s\)

\(ExtVP\_SS\) likes | follows
ExtVP – Correlations

Correlations

<table>
<thead>
<tr>
<th>S</th>
<th>O</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>B</td>
</tr>
<tr>
<td>B</td>
<td>C</td>
</tr>
<tr>
<td>B</td>
<td>D</td>
</tr>
<tr>
<td>C</td>
<td>D</td>
</tr>
</tbody>
</table>

?x follows ?y
?z likes ?y

OS

ExtVP_{follows} \times \text{VP}_{likes}

VP_{follows} \times \text{VP}_{likes}

SS

ExtVP_{follows} \times \text{VP}_{likes}

OO

ExtVP_{follows} \times \text{VP}_{likes}
Basic Idea:

- Deduce a subquery for every triple pattern
- Use the smallest ExtVP table for every subquery (i.e. with best selectivity)
- Join subquery results

### SPARQL

```
SELECT * WHERE {
?x :likes ?w .
?x :follows ?y .
?y :follows ?z .
?z :likes ?w }
```

### ExtVP Table Selection

<table>
<thead>
<tr>
<th>Table</th>
<th>Size</th>
<th>SF</th>
</tr>
</thead>
<tbody>
<tr>
<td>VP_{follows}</td>
<td>4 tuples</td>
<td>1.00</td>
</tr>
<tr>
<td>ExtVP_{SO}^{follows \mid follows}</td>
<td>3 tuples</td>
<td>0.75</td>
</tr>
<tr>
<td>ExtVP_{OS}^{follows \mid likes}</td>
<td>1 tuple</td>
<td>0.25</td>
</tr>
</tbody>
</table>

### SQL

```
SELECT s AS x, o AS w
FROM ...
```

```
SELECT s AS x, o AS y
FROM ...
```

```
SELECT s AS y, o AS z
FROM ExtVP_{OS}^{follows \mid likes}
```

```
SELECT s AS z, o AS w
FROM ...
```
ExtVP – Query Processing

Join Order Optimization:

- **Heuristics**: smaller join inputs lead to smaller join outputs
- Order join sequence by size of ExtVP tables (smallest first) but avoid cross products
- Prioritize triple patterns with more bound components (i.e. WHERE clause in subquery)

**SPARQL**

```
SELECT * WHERE {
?x :likes ?w .
?x :follows ?y .
?y :follows ?z .
?z :likes ?w }
```

**SQL**

```
SELECT s AS x, o AS w
FROM VPlikes
```

```
SELECT s AS x, o AS y
FROM ExtVP_SSfollows|likes
```

```
SELECT s AS y, o AS z
FROM ExtVP_OSfollows|likes
```

```
SELECT s AS z, o AS w
FROM ExtVP_SOlikes|follows
```

**SQL (optimized)**

```
SELECT s AS z, o AS w
FROM ExtVP_SOlikes|follows
```

```
SELECT s AS y, o AS z
FROM ExtVP_OSfollows|likes
```

```
SELECT s AS x, o AS y
FROM ExtVP_SSfollows|likes
```

```
SELECT s AS x, o AS w
FROM ExtVP_SOlikes|follows
```

```
SELECT s AS x, o AS w
FROM VPlikes
```
Evaluation

**Setup**
- ExtVP implemented on top of Spark SQL (omitting OO correlations)
- Cluster of 10 machines (1 master + 9 worker)
- Cloudera’s Distribution of Hadoop CDH 5.4 (based on Hadoop 2.6.0)
- Spark 1.3.0 (20 GB memory for each executor)

**Datasets**
- Waterloo SPARQL Diversity Test Suite (WatDiv) → generated datasets up to ~ 1 billion triples
- YAGO dump (YAGO2s 2.5.3) → real-world dataset with ~ 245 million triples

**Competitors**
- PigSPARQL
- SHARD → (based on MapReduce)
- H2RDF+ → (based on HBase)
- Sempala → (based on Impala)
WatDiv Basic Testing Use Case:

- 20 query templates in 4 groups: star (S), linear (L), snowflake (F), complex (C)
- AM per query group for ~ 1 billion triples

<table>
<thead>
<tr>
<th></th>
<th>S2RDF</th>
<th>Sempala</th>
<th>H2RDF+</th>
<th>PigSPARQL</th>
<th>SHARD</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>38</td>
<td>266</td>
<td>1956</td>
<td>571</td>
<td>5046</td>
</tr>
<tr>
<td>L</td>
<td>51</td>
<td>131</td>
<td>709</td>
<td>1500</td>
<td>4427</td>
</tr>
<tr>
<td>F</td>
<td>128</td>
<td>310</td>
<td>4815</td>
<td>2300</td>
<td>2303</td>
</tr>
<tr>
<td>C</td>
<td>153</td>
<td>432</td>
<td>9500</td>
<td>2333</td>
<td>5500</td>
</tr>
</tbody>
</table>
ExtVP – Selectivity Threshold

### Observation:
- Selectivity of ExtVP tables is skewed (some reduce much, others little)
- Unselective ExtVP tables cost a lot of storage
- Unselective ExtVP tables do not improve performance that much

→ Introduce a selectivity threshold for ExtVP tables

<table>
<thead>
<tr>
<th></th>
<th># triples</th>
<th># tables</th>
<th># tuples</th>
<th>overhead</th>
</tr>
</thead>
<tbody>
<tr>
<td>WatDiv</td>
<td>1091 M</td>
<td>2129</td>
<td>13059 M</td>
<td>~ 11n</td>
</tr>
<tr>
<td>YAGO</td>
<td>245 M</td>
<td>6588</td>
<td>2619 M</td>
<td>~ 10n</td>
</tr>
</tbody>
</table>

Cost of ExtVP

input size (n) ExtVP size
### S2RDF: RDF Querying with SPARQL on Spark

**ExtVP – Selectivity Threshold**

<table>
<thead>
<tr>
<th>SF TH</th>
<th>#tables (%)</th>
<th>#tuples (%)</th>
<th>HDFS size (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>WatDiv</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.00</td>
<td>86 (0.04)</td>
<td>1091 M (0.08)</td>
<td>6.6 GB (0.09)</td>
</tr>
<tr>
<td>0.25</td>
<td>1275 (0.60)</td>
<td>3316 M (0.25)</td>
<td>18.4 GB (0.26)</td>
</tr>
<tr>
<td>0.50</td>
<td>1609 (0.76)</td>
<td>5889 M (0.45)</td>
<td>32.1 GB (0.46)</td>
</tr>
<tr>
<td>0.75</td>
<td>1887 (0.89)</td>
<td>9480 M (0.73)</td>
<td>51.3 GB (0.73)</td>
</tr>
<tr>
<td>1.00</td>
<td>2129 (1.00)</td>
<td>13059 M (1.00)</td>
<td>70.3 GB (1.00)</td>
</tr>
</tbody>
</table>

- **SF TH = 0.25**: 95% of best performance benefit
- Overhead (in tuples) reduced: ~11n → ~2n
- **SF TH = 0.25**: 99% of best performance benefit
- Overhead (in tuples) reduced: ~10n → ~2n
- **Overall**: achieves almost same performance
- Uses only ~3n tuples (i.e. ~2n overhead)
Thank you for your attention!

Questions?
References


Evaluation

WatDiv Basic Testing Use Case:

- 20 query templates in 4 groups: star (S), linear (L), snowflake (F), complex (C)
- AM per query for ~ 1 billion triples
WatDiv Increasing Linear Use Case:

- 3 query types: user bound, retailer bound, unbound
- Increasing linear path length (5 up to 10)
Selectivity Threshold:

- For WatDiv Basic Testing use case (per query type)