PigSPARQL
Mapping SPARQL to Pig Latin

3rd International Workshop on Semantic Web Information Management (SWIM 2011)

Alexander Schätzle
Martin Przyjaciel–Zablocki
Georg Lausen

University of Freiburg
Databases & Information Systems
Overview

1. Motivation
2. Framework
3. PigSPARQL
4. Evaluation
5. Summary
Motivation

- **Semantic Web**
  - Amount of Semantic Data increases steadily
  - RDF is W3C standard for representing Semantic Data

- **Social Networks**
  - > 500 million active users in Facebook
  - Interacting with >900 million objects (pages, groups, events)
  - Social Graphs can also be represented in RDF

But how can we execute SPARQL queries on very large RDF datasets with billions of triples?

- **Our Approach:** Distributed execution of SPARQL queries using MapReduce
MapReduce

- **Automatic parallelization of computations**

- **Distributed File System**
  - Commodity hardware → Fault tolerance by replication
  - Very large files / write–once, read–many pattern

- **Apache Hadoop**
  - Well–known Open–Source implementation
Pig Latin

- **Properties of Pig Latin (Yahoo!)**
  - „High-Level“ Language for Data Analysis with Hadoop
  - Automatic Translation into MapReduce-Jobs
  - Link between User & MapReduce

- **Utilize Advantages of MapReduce**
  - Parallelization done by the System
  - Good Fault Tolerance & Scalability

- **Avoid Drawbacks of MapReduce**
  - „Low-Level“ to implement & hard to maintain
  - No Primitives like JOIN or GROUP
Pig Latin (2)

- **Data model of Pig Latin**
  - Flexible, fully nested
  - Main Data type: **Tuple** \(\rightarrow\) Sequence of fields
  - A field of a Tuple can be of any Data type, i.e.

 Atom: 'Bob' or 24  
Tuple: ('John', 'Doe')  
Bag: 
  - ('Bob', 'Sarah')  
  - ('Peter', ('likes', 'football'))  
Map:  
  - 'knows' \(\rightarrow\) {'Sarah'}  
  - 'age' \(\rightarrow\) 24
Important **Operators** of Pig Latin

**FOREACH:** Apply Processing on every Tuple of a Bag

```pig
result = FOREACH input GENERATE field1*field2 AS mul ;
```

**FILTER:** Delete unwanted Tuples of a Bag

```pig
adults = FILTER persons BY age >= 18 ;
```

**[OUTER] JOIN:** Join two or more Bags

```pig
result = JOIN left BY field1 [LEFT OUTER], right BY field2;
```

**UNION:** Combine two or more Bags

```pig
result = UNION bag1, bag2 ;
```

**ORDER:** Order a Bag by the specified field(s)

```pig
result = ORDER input BY field1 ;
```
3. PigSPARQL

Mapping SPARQL to Pig Latin
1. Step

Convert SPARQL Query into SPARQL Algebra Tree

PREFIX : <http://example.org/>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT *
WHERE {
    ?person foaf:knows :Peter
    OPTIONAL {
        ?person foaf:mbox ?mb
    }
    FILTER (?age >= 18)
}
2. Step

- Triple Pattern Reordering, Filter Pushing, Filter Substitution

```sparql
BGP
?person knows :Peter

BGP
?person mbox ?mb

LeftJoin

Filter
?age >= 18

BGP
?person knows :Peter.
?person age ?age

BGP
?person mbox ?mb
```
3. Step
   - Translate Algebra Tree to Pig Latin Program

```pig
indata = LOAD 'pathToFile' USING rdfLoader() AS (s,p,o) ;
```

```
LeftJoin

Filter
  ?age >= 18

BGP
  ?person mbox ?mb

BGP
  ?person knows :Peter.
  ?person age ?age
```
Mapping to Pig Latin (2)

3. Step
- Translate Algebra Tree to Pig Latin Program

```
indata = LOAD 'pathToFile' USING rdfLoader() AS (s,p,o) ;

f1 = FILTER indata BY p=='foaf:knows' AND o==':Peter' ;
t1 = FOREACH f1 GENERATE s AS person ;
f2 = FILTER indata BY p=='foaf:age';
t2 = FOREACH f2 GENERATE s AS person, o AS age ;
j1 = JOIN t1 BY person, t2 BY person ;
BGP1 = FOREACH j1 GENERATE
    t1::person AS person, t2::age AS age ;
```

```
LeftJoin

Filter
?age >= 18
BGP
?person mbox ?mb
```

```
BGP
?person knows :Peter.
?person age ?age
```
Mapping to Pig Latin (3)

3. Step

Translate Algebra Tree to Pig Latin Program

indata = LOAD 'pathToFile' USING rdfLoader() AS (s,p,o) ;

f1 = FILTER indata BY p=='foaf:knows' AND o==':Peter' ;
t1 = FOREACH f1 GENERATE s AS person ;
f2 = FILTER indata BY p=='foaf:age' ;
t2 = FOREACH f2 GENERATE s AS person, o AS age ;
j1 = JOIN t1 BY person, t2 BY person ;
BGP1 = FOREACH j1 GENERATE
   t1::person AS person, t2::age AS age ;

F1 = FILTER BGP1 BY age >= 18 ;
Mapping to Pig Latin (4)

3. Step

- Translate Algebra Tree to Pig Latin Program

```sql
indata = LOAD 'pathToFile' USING rdfLoader() AS (s,p,o) ;

f1 = FILTER indata BY p=='foaf:knows' AND o==':Peter' ;
t1 = FOREACH f1 GENERATE s AS person ;
f2 = FILTER indata BY p=='foaf:age';
t2 = FOREACH f2 GENERATE s AS person, o AS age ;
j1 = JOIN t1 BY person, t2 BY person ;
BGP1 = FOREACH j1 GENERATE
t1::person AS person, t2::age AS age ;

F1 = FILTER BGP1 BY age >= 18 ;

f1 = FILTER indata BY p=='foaf:mbox' ;
BGP2 = FOREACH indata GENERATE s AS person, o AS mb ;
```
3. Step

Translate Algebra Tree to Pig Latin Program

```
indata = LOAD 'pathToFile' USING rdfLoader() AS (s,p,o) ;

f1 = FILTER indata BY p=='foaf:knows' AND o==':Peter' ;
t1 = FOREACH f1 GENERATE s AS person ;
f2 = FILTER indata BY p=='foaf:age';
t2 = FOREACH f2 GENERATE s AS person, o AS age ;
j1 = JOIN t1 BY person, t2 BY person ;

BGP1 = FOREACH j1 GENERATE
t1::person AS person, t2::age AS age ;

F1 = FILTER BGP1 BY age >= 18 ;

f1 = FILTER indata BY p=='foaf:mbox' ;
BGP2 = FOREACH indata GENERATE s AS person, o AS mb ;

lj = JOIN F1 BY person LEFT OUTER, BGP2 BY person ;

LJ1 = FOREACH lj GENERATE F1::person AS person,
     F1::age AS age, BGP2::mb AS mb ;

STORE LJ1 INTO 'pathToOutput' USING resultWriter() ;
```
Optimizations

- **SPARQL Algebra**
  - Filter Optimization (Pushing, Splitting, Substitution)
  - Triple Pattern Reordering by Selectivity

- **Algebra Translation**
  - Delete unnecessary Data as early as possible
  - Multi-Joins to reduce the number of Joins

- **Data Representation**
  - Vertical Partitioning of the RDF Data by Predicate
4. Evaluation

SP²Bench SPARQL Benchmark
SP²Bench Query 2

- Native Translation needs 8 Joins + 1 Outer Join
- **Optimizations:**
  - Multi-Join reduces the number of Joins
  - Vertical Partitioning reduces input size

WHERE {
  ?inproc rdfs:seeAlso ?ee .
  ?inproc dcterms:issued ?yr
  OPTIONAL {
    ?inproc bench:abstract ?abstract
  }
}
ORDER BY ?yr
SP²Bench Query 6

- Unsafe Filter Condition (not well designed)
- Implements a (closed world) negation
- **Optimizations:**
  - Vertical Partitioning reduces input size

Query 6

```sparql
SELECT ?yr ?name ?doc
WHERE {
  ?class rdfs:subClassOf foaf:Document .
  ?author foaf:name ?name
  OPTIONAL {
    ?class2 rdfs:subClassOf foaf:Document .
    ?doc2 dc:creator ?author2
    FILTER (?author=?author2 && yr2 < yr)
  }
  FILTER (!bound(?author2))
}
```

### 4. Evaluation Query 6

- **Visual Representation:**
  - Graph showing time (in hours) vs. RDF triples (in Millions)
  - Bar charts for HDFS Bytes Read, HDFS Bytes Written, and Reduce Shuffle Bytes in GB (800M / 1600M RDF triples)

#### Graph:
- **Axes:**
  - X-axis: RDF triples (in Millions)
  - Y-axis: Time (in hours)

#### Bar Charts:
- **Legend:**
  - Q6 (800M)
  - Q6 VP (800M)
  - Q6 VP (1600M)

<table>
<thead>
<tr>
<th>RDF Triples (in Millions)</th>
<th>Q6 (800M)</th>
<th>Q6 VP (800M)</th>
<th>Q6 VP (1600M)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>200</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>400</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>600</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>800</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1000</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1200</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1400</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1600</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

**Table:**
- **Columns:**
  - HDFS Bytes Read
  - HDFS Bytes Written
  - Reduce Shuffle Bytes
- **Data:**
  - Q6 (800M):
    - HDFS Bytes Read: 451 GB
    - HDFS Bytes Written: 682 GB
    - Reduce Shuffle Bytes: 258 GB
  - Q6 VP (800M):
    - HDFS Bytes Read: 905 GB
    - HDFS Bytes Written: 798 GB
    - Reduce Shuffle Bytes: 367 GB
  - Q6 VP (1600M):
    - HDFS Bytes Read: 258 GB
    - HDFS Bytes Written: 183 GB
    - Reduce Shuffle Bytes: 367 GB

---

PigSPARQL: Mapping SPARQL to Pig Latin 19
5. Summary

PigSPARQL: Key Points
Summary

- **PigSPARQL**: SPARQL execution with MapReduce

- All features of SPARQL 1.0 (not only BGP)

- Evaluation with a SPARQL specific performance benchmark (more complex queries than LUBM)

- Linear scaling behavior with up to 1.6 Billion triples

- **Future Work**: SPARQL 1.1
Thanks for your attention! Questions?
Backup Slides

a) SPARQL Graph Pattern
b) Pig Latin – Operators
a) SPARQL Graph Pattern

- **Basic Graph Pattern**
  - Finite set of **Triple Patterns** concatenated with AND (.)
  - A Triple Pattern is an RDF Triple with variables

- **Graph Pattern**
  - A Basic Graph Pattern is a Graph Pattern
  - If P and P' are Graph Patterns, then \{P\}.{P'}, \{P\} UNION \{P'\} and \{P\} OPTIONAL \{P'\} are also Graph Patterns
  - If P is a Graph Pattern and R is a Filter Condition, then
    P FILTER (R) is also a Graph Pattern
  - If P is a Graph Pattern, u an URI and ?v a variable, then
    GRAPH u \{P\} and GRAPH ?v \{P\} are also Graph Pattern
## Pig Latin – Operators

**FOREACH:** Apply Processing on every Tuple of a Bag  
Ex: \( \text{result} = \text{FOREACH input GENERATE field1*field2 AS mul} ; \)

<table>
<thead>
<tr>
<th>input</th>
<th>result</th>
</tr>
</thead>
<tbody>
<tr>
<td>field1</td>
<td>mul</td>
</tr>
<tr>
<td>field2</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>6</td>
</tr>
<tr>
<td>3</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>28</td>
</tr>
<tr>
<td>7</td>
<td></td>
</tr>
</tbody>
</table>

**FILTER:** Delete unwanted Tuples of a Bag  
Ex: \( \text{adults} = \text{FILTER persons BY age} \geq 18 ; \)

<table>
<thead>
<tr>
<th>persons</th>
<th>adults</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>name</td>
</tr>
<tr>
<td>age</td>
<td>age</td>
</tr>
<tr>
<td>Bob</td>
<td>Bob</td>
</tr>
<tr>
<td>21</td>
<td>21</td>
</tr>
<tr>
<td>Sarah</td>
<td></td>
</tr>
<tr>
<td>17</td>
<td></td>
</tr>
</tbody>
</table>
**b) Pig Latin – Operators (2)**

**[OUTER] JOIN:** Join two or more Bags

Ex: result = JOIN left BY field1 [LEFT OUTER], right BY field2;

<table>
<thead>
<tr>
<th>left</th>
<th>right</th>
<th>result</th>
</tr>
</thead>
<tbody>
<tr>
<td>field1</td>
<td>field1</td>
<td>left:: field1</td>
</tr>
<tr>
<td>a</td>
<td>4</td>
<td>a</td>
</tr>
<tr>
<td>b</td>
<td>7</td>
<td>a</td>
</tr>
<tr>
<td></td>
<td></td>
<td>right:: field1</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>a</td>
</tr>
<tr>
<td></td>
<td>7</td>
<td>a</td>
</tr>
<tr>
<td></td>
<td></td>
<td>right:: field2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>a</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>a</td>
</tr>
<tr>
<td></td>
<td></td>
<td>a</td>
</tr>
</tbody>
</table>

Outer
b) Pig Latin – Operators (3)

**UNION:**

Ex: result = UNION bag1, bag2;

<table>
<thead>
<tr>
<th>bag1</th>
<th></th>
<th>bag2</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>field1</td>
<td>field2</td>
<td>field1</td>
<td>field2</td>
</tr>
<tr>
<td>a</td>
<td>1</td>
<td>b</td>
<td>3</td>
</tr>
</tbody>
</table>

\[ \text{result} = \text{bag1} \cup \text{bag2} \]

<table>
<thead>
<tr>
<th>result</th>
</tr>
</thead>
<tbody>
<tr>
<td>field1</td>
</tr>
<tr>
<td>a</td>
</tr>
<tr>
<td>b</td>
</tr>
</tbody>
</table>

**ORDER:**

Ex: result = ORDER input BY field1;

<table>
<thead>
<tr>
<th>input</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>field1</td>
<td>field2</td>
</tr>
<tr>
<td>3</td>
<td>a</td>
</tr>
<tr>
<td>1</td>
<td>b</td>
</tr>
</tbody>
</table>

\[ \text{result} = \text{ORDER} \text{input} \text{BY} \text{field1} \]

<table>
<thead>
<tr>
<th>result</th>
</tr>
</thead>
<tbody>
<tr>
<td>field1</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>3</td>
</tr>
</tbody>
</table>