PigSPARQL:
A Translation from SPARQL to Pig Latin

14th BTW conference on Database Systems for Business, Technology and Web

Alexander Schätzle
Martin Przyjaciel-Zablocki
Thomas Hornung
Georg Lausen

Albert-Ludwigs-University of Freiburg
Databases & Information Systems
Agenda

1. Motivation
2. Foundations
3. Translation
4. Evaluation
5. Summary
1. 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
  - Social Graphs can also be represented in RDF

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

- **Our Approach:** Distributed execution of SPARQL queries using MapReduce
2. Foundations – RDF

- **RDF Triple**
  - Expressions of the form `<subject, predicate, object>`
  - `<subject>` has property `<predicate>` with value `<object>`
- **RDF Graph**
  - A RDF dataset consists of a set of RDF Triples and can be interpreted as directed labeled graph

```reasoning
@prefix : <http://example.org/> .
@prefix foaf: <http://xmlns.com/foaf/0.1/> .

:Peter foaf:mbox "peter@abc.de" .
:John foaf:knows :Bob .
:John foaf:knows :Sarah .
:John foaf:knows :Peter .
:John foaf:age "27" .
:Bob foaf:knows :Peter .
:Bob foaf:age "32" .
:Bob foaf:mbox "bob@abc.de" .
:Sarah foaf:knows :Peter .
:Sarah foaf:age "17" .

peter@abc.de

Sarah

Bob

Peter

John

17

27

32

knows

age

mbox
```
2. Foundations – SPARQL

- W3C Recommendation for RDF Query Language
- Graph Pattern
  - A SPARQL query defines a Graph Pattern that is applied to an RDF Graph
  - The result set is a mapping from variables to resources in the RDF Graph

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

- W3C Recommendation for RDF Query Language
- Graph Pattern
  - A SPARQL query defines a Graph Pattern that is applied to an RDF Graph
  - The result set is a mapping from variables to resources in the RDF Graph

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

- W3C Recommendation for RDF Query Language
- Graph Pattern
  - A SPARQL query defines a Graph Pattern that is applied to an RDF Graph
  - The result set is a mapping from variables to resources in the RDF Graph

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

- Framework introduced by Google in 2004
  - Automatic parallelization of computations
  - Fixed two-stage process: Map -> Reduce -> Map -> ...

- Distributed File System
  - Fault tolerance by replication
  - Large Files / Write Once – Read Many

- Hadoop
  - Well-known Open Source Implementation
  - Used by Yahoo, Facebook, Amazon, IBM, Last.fm, ...
2. Foundations – 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
2. Foundations – Pig Latin

- **Data model of Pig Latin**
  - Flexible, fully nested
  - Main Data type: **Tuple** → 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' → {('Sarah')}
  - 'age' → 24
2. Foundations – Pig Latin

- **Important Operators of Pig Latin**

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

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

  **[OUTER] JOIN:** Join two or more Bags
  \[
  \text{result} = \text{JOIN left BY field1 [LEFT OUTER], right BY field2 ;}
  \]

  **UNION:** Combine two or more Bags
  \[
  \text{result} = \text{UNION bag1, bag2 ;}
  \]

  **ORDER:** Order a Bag by the specified field(s)
  \[
  \text{result} = \text{ORDER input BY field1 ;}
  \]

more
3. Translation of SPARQL (1)

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 .
  ?person foaf:age ?age
  FILTER (?age >= 18)
  OPTIONAL {
    ?person foaf:mbox ?mb
  }
}

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

2. Step

- Translate Algebra–Tree into Pig Latin Program

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

LeftJoin

<table>
<thead>
<tr>
<th>Filter</th>
<th>BGP</th>
</tr>
</thead>
<tbody>
<tr>
<td>?age &gt;= 18</td>
<td>?person mbox ?mb</td>
</tr>
</tbody>
</table>

BGP

<table>
<thead>
<tr>
<th>BGP</th>
</tr>
</thead>
<tbody>
<tr>
<td>?person knows :Peter.</td>
</tr>
<tr>
<td>?person age ?age</td>
</tr>
</tbody>
</table>
3. Translation of SPARQL (2)

2. Step

- Translate Algebra–Tree into 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 ;
3. Translation of SPARQL (2)

2. Step
  - Translate Algebra–Tree into 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 ;
```
3. Translation of SPARQL (2)

2. Step
   - Translate Algebra–Tree into 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 ;

BGP2 = FOREACH indata GENERATE s AS person, o AS mb ;
```

Diagram:
- **LeftJoin**
  - **Filter**
    - ?age >= 18
  - **BGP**
    - ?person mbox ?mb
  - **BGP**
    - ?person knows :Peter.
    - ?person age ?age
3. Translation of SPARQL (2)

2. Step
   - Translate Algebra-Tree into Pig Latin Program

```pig
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() ;
```
4. Evaluation

- Query 2 of SP²Bench Performance Benchmark
- 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
```
5. Summary

- **PigSPARQL**: SPARQL query evaluation with MapReduce
- Nearly all features of SPARQL 1.0 (not only BGPs)
- Evaluated 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!
References


Backup Slides

MapReduce
SPARQL Graph Pattern
Pig Latin – Data Model
Pig Latin – Operators
MapReduce (2)

- Steps of a MapReduce execution

Map-Phase

- Input (HDFS)
  - Split 0
  - Split 1
  - Split 2
  - Split 3
  - Split 4
  - Split 5

Intermediate Results (Local)

Shuffle-Phase

- Map
- Map
- Map

Reduce-Phase

- Reduce
- Reduce
- Output (HDFS)
  - Out 0
  - Out 1
MapReduce (3)

- Signature of a Map-Function
  - `map(in_key, in_value) -> (out_key, intermediate_value) list`

- Signature of a Reduce-Function
  - `reduce(out_key, intermediate_value list) -> out_value list`
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
Pig Latin – Operators (1)

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

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

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

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

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

Ex: \( \text{result} = \text{JOIN} \text{ left BY field1 [LEFT OUTER], right BY field2} ; \)

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

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

Outer
## Pig Latin – Operators (3)

**UNION:**  
Ex: result = UNION bag1, bag2;

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

Ex: result = UNION bag1, bag2;

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

**ORDER:**  
Ex: result = ORDER input BY field1;

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

Ex: result = ORDER input BY field1;

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

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

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