12 June 2011

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

IIF INSTITUT FÜR INFORMATIK FREIBURG

UNI FREIBURG

Overview

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

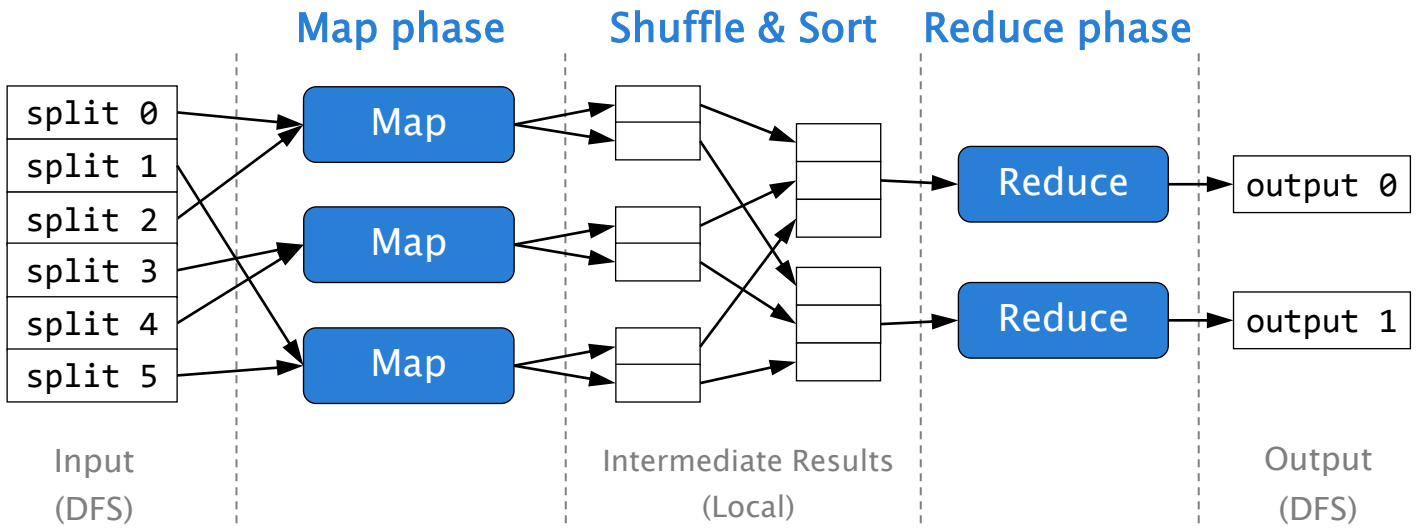
Motivation

- **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 -> 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]

Pig Latin (3)

- Important Operators of Pig Latin

FOREACH: Apply Processing on every Tuple of a Bag

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

FILTER: Delete unwanted Tuples of a Bag

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

[OUTER] JOIN: Join two or more Bags

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

UNION: Combine two or more Bags

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

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

```
result = ORDER input BY field1 ;
```

[more]

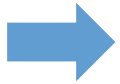
3. PigSPARQL

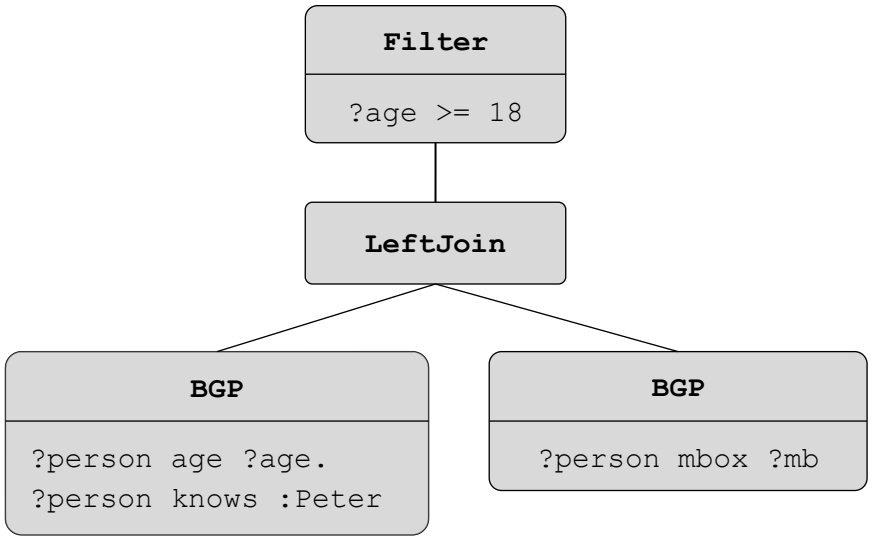
- Mapping SPARQL to Pig Latin

SPARQL Algebra Tree

- 1. Step
  - Convert SPARQL Query into SPARQL Algebra Tree

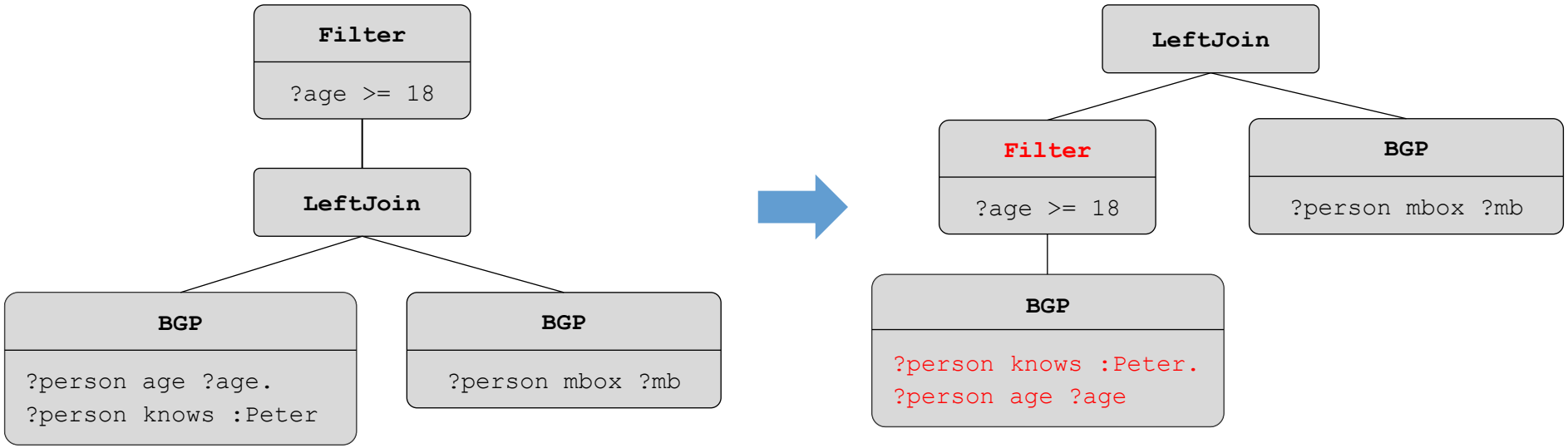
```
PREFIX :     <http://example.org/>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT *
WHERE {
  ?person foaf:age ?age .
  ?person foaf:knows :Peter
  OPTIONAL {
    ?person foaf:mbox ?mb
  }
  FILTER (?age >= 18)
}
```





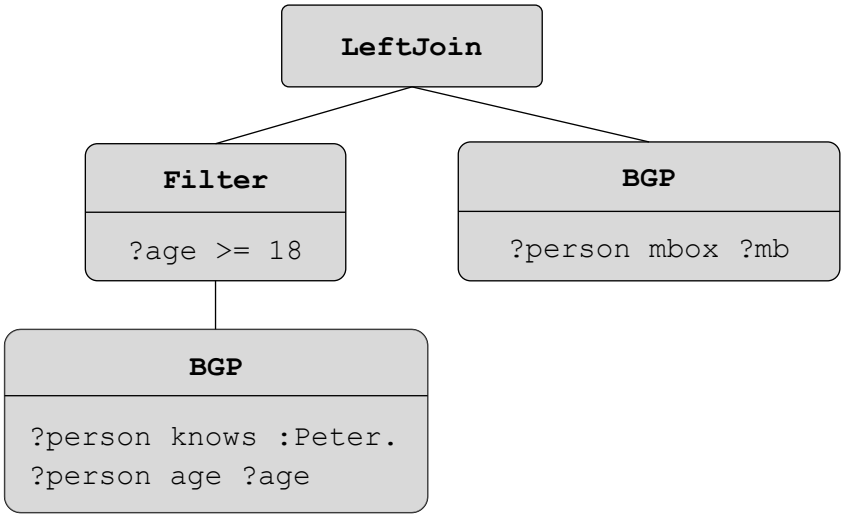
SPARQL Algebra Optimization

- 2. Step
  - Triple Pattern Reodering, Filter Pushing, Filter Substitution



Mapping to Pig Latin

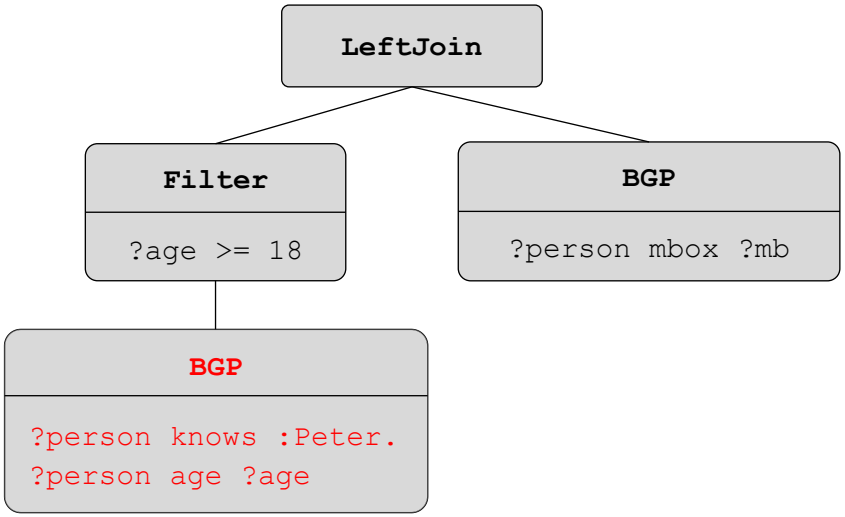
- 3. Step
  - Translate Algebra Tree to Pig Latin Program




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

Mapping to Pig Latin (2)

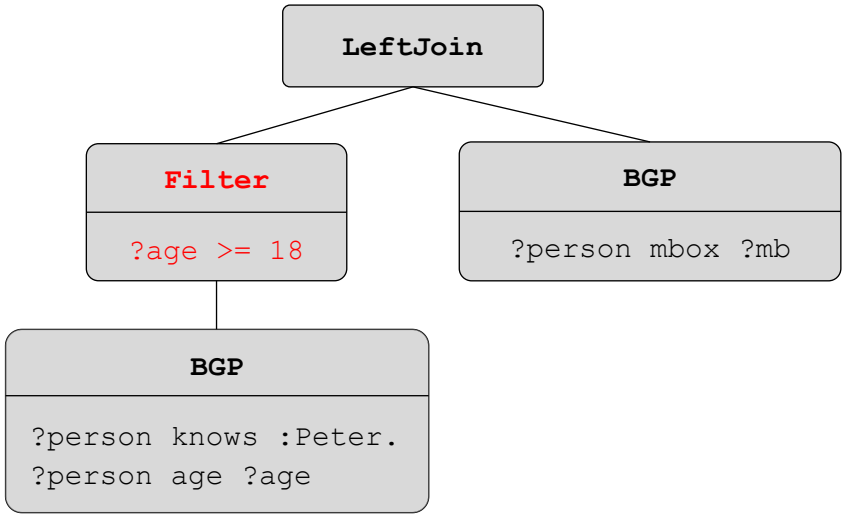
- 3. Step
  - Translate Algebra Tree to Pig Latin Program




```
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 ;
```

Mapping to Pig Latin (3)

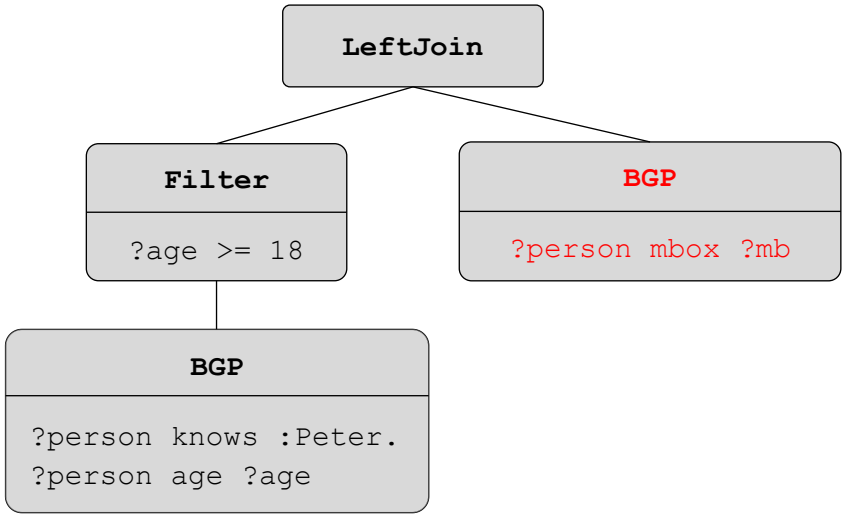
- 3. Step
  - Translate Algebra Tree to Pig Latin Program




```
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)

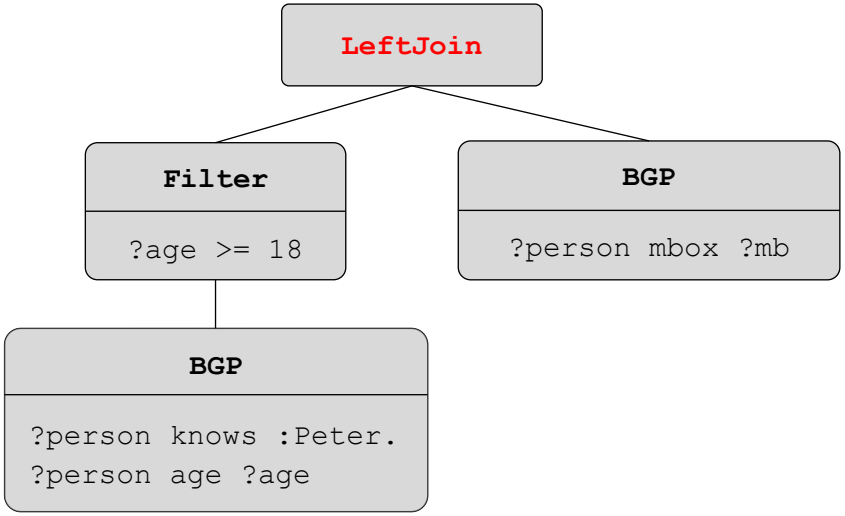
- 3. Step
  - Translate Algebra Tree to Pig Latin Program




```
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 ;
```

Mapping to Pig Latin (5)

- 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

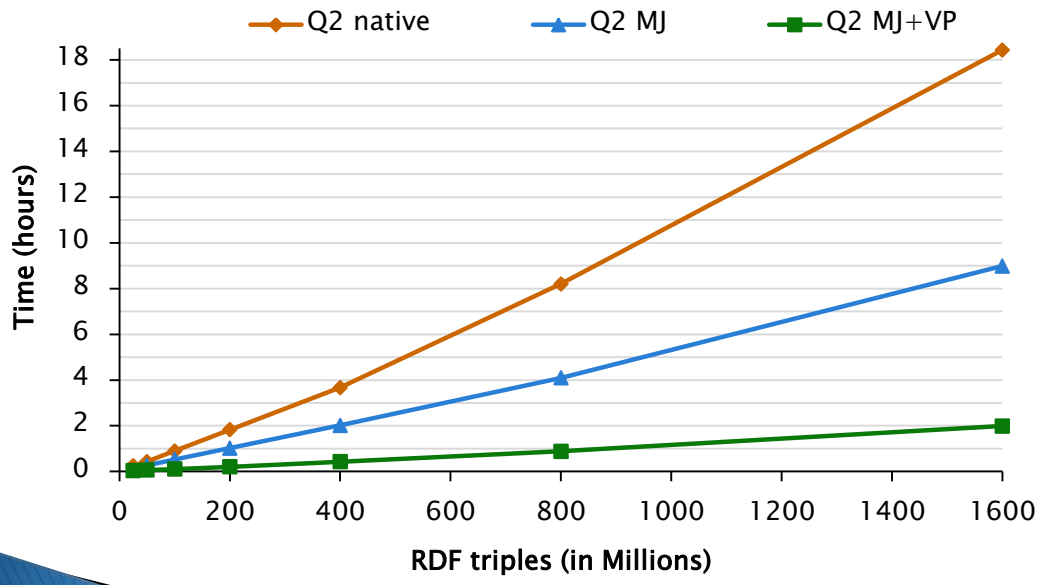
4. Evaluation

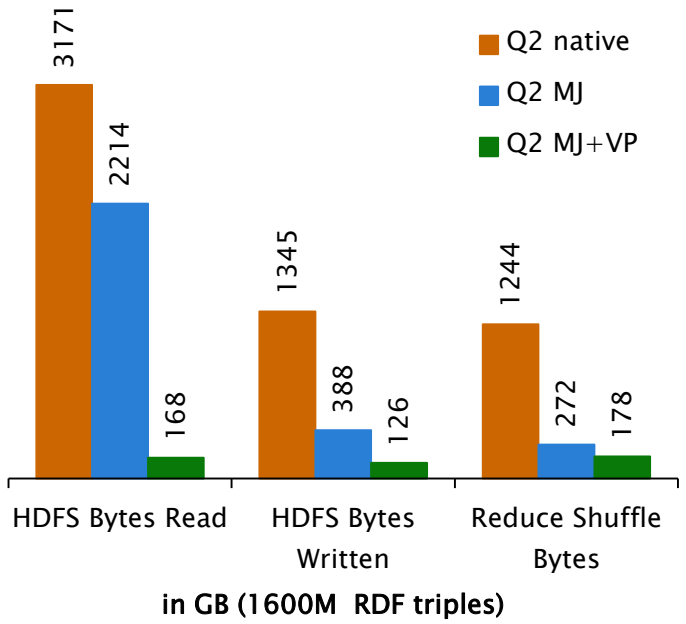
- SP²Bench SPARQL Benchmark

SP²Bench Query 2

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

```
SELECT ?inproc ?author ?booktitle ?title
       ?proc ?ee ?page ?url ?yr ?abstract
WHERE {
  ?inproc rdf:type bench:Inproceedings .
  ?inproc dc:creator ?author .
  ?inproc bench:booktitle ?booktitle .
  ?inproc dc:title ?title .
  ?inproc dcterms:partOf ?proc .
  ?inproc rdfs:seeAlso ?ee .
  ?inproc swrc:pages ?page .
  ?inproc foaf:homepage ?url .
  ?inproc dcterms:issued ?yr
  OPTIONAL {
    ?inproc bench:abstract ?abstract
  }
}
ORDER BY ?yr
```

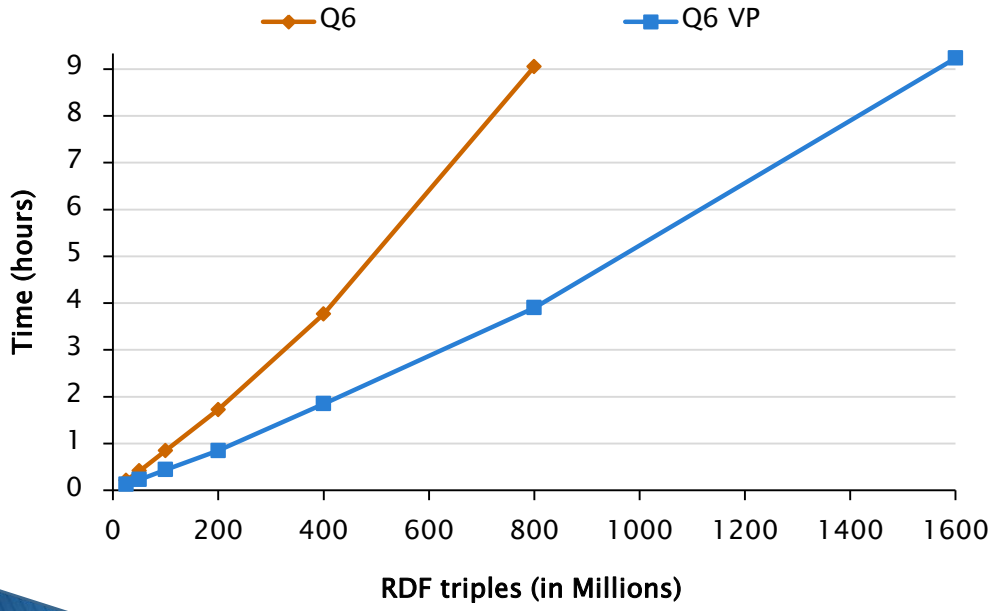


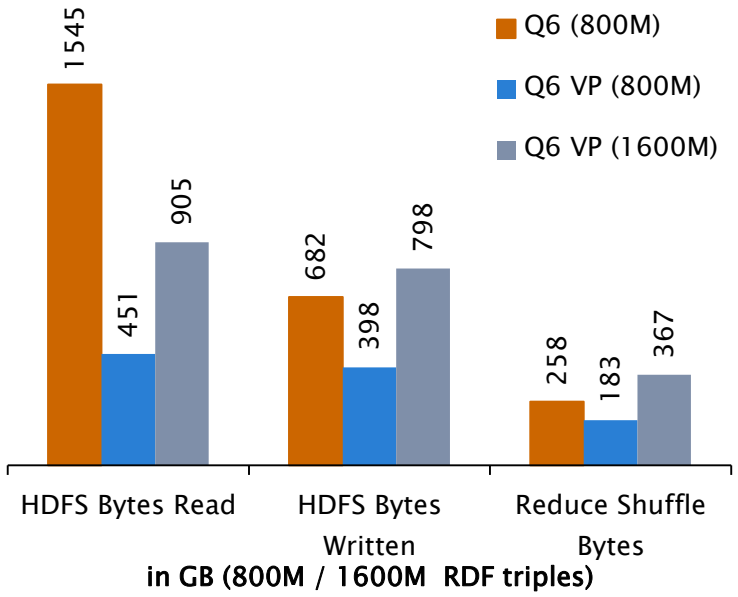


SP²Bench Query 6

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

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





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

b) Pig Latin – Operators

FOREACH: Apply Processing on every Tuple of a Bag

Ex: result = FOREACH input GENERATE field1*field2 AS mul ;

input

field1	field2
2	3
4	7

⇨

result

mul
6
28

FILTER: Delete unwanted Tuples of a Bag

Ex: adults = FILTER persons BY age >= 18 ;

persons

name	age
Bob	21
Sarah	17

⇨

adults

name	age
Bob	21

b) Pig Latin – Operators (2)

[OUTER] JOIN: Join two or more Bags

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

left

field1
a
b

⋈

right

field1	field2
4	a
7	a

⇨

result

left:: field1	right:: field1	right:: field2
a	4	a
a	7	a

left

field1
a
b

⋈ Outer

right

field1	field2
4	a
7	a

⇨

result

left:: field1	right:: field1	right:: field2
a	4	a
a	7	a
b		

b) Pig Latin – Operators (3)

UNION: Ex: result = UNION bag1, bag2 ;

bag1

field1	field2
a	1

U

bag2

field1	field2
b	3

⇨

result

field1	field2
a	1
b	3

ORDER: Ex: result = ORDER input BY field1 ;

input

field1	field2
3	a
1	b

⇨

result

field1	field2
1	b
3	a