SPARQLGX:

Efficient Distributed Evaluation of SPARQL with Spark

Damien Graux, Louis Jachiet, Pierre Genevès, Nabil Layaïda

Tyrex Team, France
http://tyrex.inria.fr

ISWC – October 20th, 2016

Motivations

Context

- Large amounts of RDF data $\Longrightarrow$ distribution

Motivations

Context

- Large amounts of RDF data $\Longrightarrow$ distribution
- Extract quickly information from them using SPARQL

Motivations

Context

- Large amounts of RDF data $\implies$ distribution
- Extract quickly information from them using SPARQL
- In addition, we want to be *e.g.* resilient, or parsimonious

Motivations

Context

- Large amounts of RDF data $\implies$ distribution
- Extract quickly information from them using SPARQL
- In addition, we want to be *e.g.* resilient, or parsimonious

Cluster Computing Frameworks

- Provide an interface with implicit data parallelism and fault-tolerance
- Offer a set of low-level functions *e.g.* map, join, collect . . .

Motivations

Context

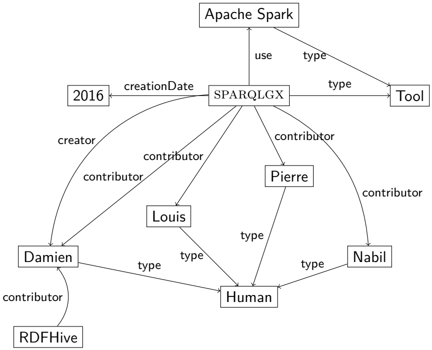
- Large amounts of RDF data $\implies$ distribution
- Extract quickly information from them using SPARQL
- In addition, we want to be *e.g.* resilient, or parsimonious

Cluster Computing Frameworks

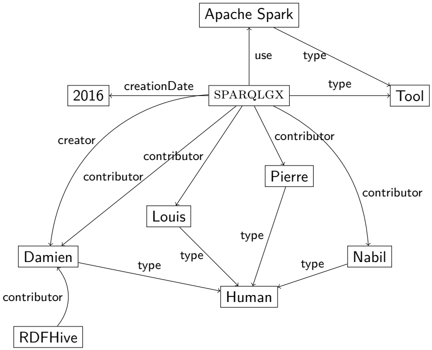
- Provide an interface with implicit data parallelism and fault-tolerance
- Offer a set of low-level functions *e.g.* map, join, collect ...

Apache Spark [Zaharia *et al.* 2012]
HDFS

Graph and Triples, an example

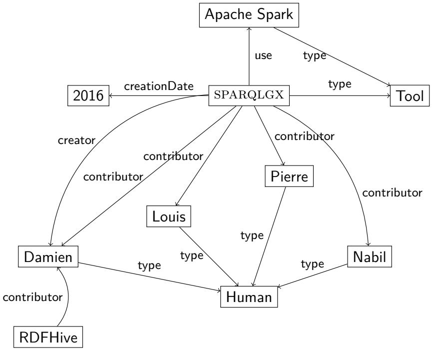


Graph and Triples, an example



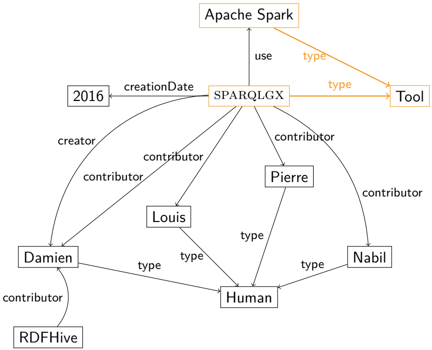
sparqlgx	type	tool
sparqlgx	creationDate	2016
sparqlgx	use	spark
spark	type	tool
Damien	type	human
Louis	type	human
Pierre	type	human
Nabil	type	human
sparqlgx	creator	Damien
RDFHive	contributor	Damien
sparqlgx	contributor	Damien
sparqlgx	contributor	Louis
sparqlgx	contributor	Pierre
sparqlgx	contributor	Nabil

Graph and Triples, an example



?s type tool
?g type human
?s contributor ?g

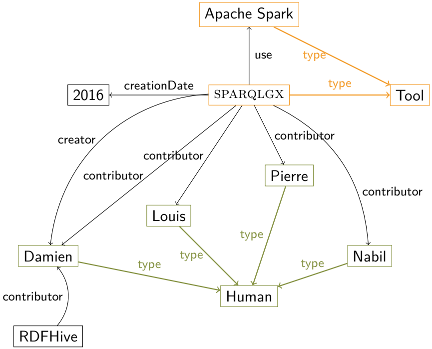
Graph and Triples, an example



?s type tool
?g type human
?s contributor ?g

?s: SPARQLGX, Apache Spark

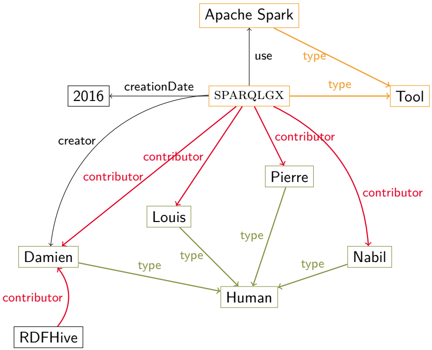
Graph and Triples, an example



?s type tool
?g type human
?s contributor ?g

?s: SPARQLGX, Apache Spark
?g: Damien, Louis, Pierre, Nabil

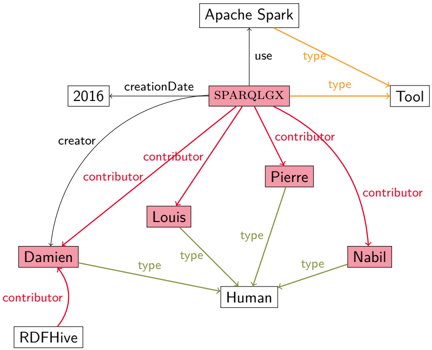
Graph and Triples, an example



?s type tool
?g type human
?s contributor ?g

?s: SPARQLGX, Apache Spark
?g: Damien, Louis, Pierre, Nabil
(?s,?g): (SPARQLGX,Damien), (SPARQLGX,Louis), (SPARQLGX,Pierre), (SPARQLGX,Nabil),(RDFHive,Damien)

Graph and Triples, an example



?s type tool
?g type human
?s contributor ?g

?s: SPARQLGX, Apache Spark
?g: Damien, Louis, Pierre, Nabil
(?s,?g): (SPARQLGX,Damien), (SPARQLGX,Louis), (SPARQLGX,Pierre), (SPARQLGX,Nabil),(RDFHive,Damien)

Solution (?s,?g): (SPARQLGX,Damien), (SPARQLGX,Louis), (SPARQLGX,Pierre), (SPARQLGX,Nabil)

Vertical Partitioning [Abadi *et al.* 2007]

RDF *predicates* carry the semantic information, thereby:

- Limited number of distinct predicates *e.g.* few hundreds [Gallego *et al.* 2011]
- Predicates rarely variable in queries [Gallego *et al.* 2011]

Vertical Partitioning [Abadi *et al.* 2007] Storage Model

dataset

sparqlgx	type	tool
sparqlgx	creationDate	2016
sparqlgx	use	spark
spark	type	tool
Damien	type	human
Louis	type	human
Pierre	type	human
Nabil	type	human
RDFHive	contributor	Damien
sparqlgx	creator	Damien
sparqlgx	contributor	Damien
sparqlgx	contributor	Louis
sparqlgx	contributor	Pierre
sparqlgx	contributor	Nabil

type.txt

sparqlgx	tool
spark	tool
Damien	human
Louis	human
Pierre	human
Nabil	human

contributor.txt

RDFHive	Damien
sparqlgx	Damien
sparqlgx	Louis
sparqlgx	Pierre
sparqlgx	Nabil

creationDate.txt

sparqlgx	2016

use.txt

sparqlgx	spark

creator.txt

sparqlgx	Damien

SPARQL Translation Process

Dealing with one TP ...

- `textFile` to access relevant files
- `filter` to keep matching triples

SPARQL Translation Process

Dealing with one TP ...

- `textFile` to access relevant files
- `filter` to keep matching triples

?s type tool .

```
textFile("type.txt")
 .filter{case(s,o)=>o.equals("tool")}
 .map{case(s,o)=>s}
```

SPARQL Translation Process

... with a conjunction of TPs

- Translate each TP
- Join them one by one

SPARQL Translation Process

?s type tool .
?g type human .
?s contributor ?g

SPARQL Translation Process

?s type tool .
?g type human .
?s contributor ?g

```
tp1=sc.textFile(``type.txt'')
  .filter{case(s,o)=>o.equals(``tool'')}
  .map{case(s,o)=>s}
  .keyBy{case(s)=>s}
```

SPARQL Translation Process

?s type tool .
?g type human .
?s contributor ?g

```
tp1=sc.textFile(``type.txt'')
  .filter{case(s,o)=>o.equals(``tool'')}
  .map{case(s,o)=>s}
  .keyBy{case(s)=>s}
tp2=sc.textFile(``type.txt'')
  .filter{case(g,o)=>o.equals(``human'')}
  .map{(g,o)=>g}
  .keyBy{case(g)=>g}
```

SPARQL Translation Process

?s type tool .
?g type human .
?s contributor ?g

```
tp1=sc.textFile(''type.txt'')
  .filter{case(s,o)=>o.equals(''tool'')}
  .map{case(s,o)=>s}
  .keyBy{case(s)=>s}
tp2=sc.textFile(''type.txt'')
  .filter{case(g,o)=>o.equals(''human'')}
  .map{(g,o)=>g}
  .keyBy{case(g)=>g}
tp3=sc.textFile(''contributor.txt'')
  .keyBy{case(s,g)=>(s,g)}
```

SPARQL Translation Process

?s type tool .
?g type human .
?s contributor ?g

```
tp1=sc.textFile(``type.txt'')
  .filter{case(s,o)=>o.equals(``tool'')}
  .map{case(s,o)=>s}
  .keyBy{case(s)=>s}
tp2=sc.textFile(``type.txt'')
  .filter{case(g,o)=>o.equals(``human'')}
  .map{(g,o)=>g}
  .keyBy{case(g)=>g}
tp3=sc.textFile(``contributor.txt'')
  .keyBy{case(s,g)=>(s,g)}

bgp=tp1.cartesian(tp2).values
  .keyBy{case(s,g)=>(s,g)}
  .join(tp3).value
```

Join Order

To minimize size of intermediate results, we try:

1. Avoiding cartesian product
2. Exploiting statistics on data

Join Order

Initial:

```
?s type tool .
?g type human .
?s contributor ?g
```

New:

```
?s contributor ?g
?s type tool .
?g type human
```

```
tp1=sc.textFile(``contributor.txt'')
  .keyBy{case(s,g)=>s}
tp2=sc.textFile(``type.txt'')
  .filter{case(s,o)=>o.equals(``tool'')}
  .map{case(s,o)=>s}
  .keyBy{case(s)=>s}
tp3=sc.textFile(``type.txt'')
  .filter{case(s,o)=>o.equals(``human'')}
  .map{case(g,o)=>g}
  .keyBy{case(g)=>g}

bgp=tp1.join(tp2).values
  .keyBy{case(s,g)=>(g)}
  .join(tp3).value
```

Two SPARQL Evaluators

SPARQLGX Advantages:

- Vertcal Partitioning provides natural compression and indexing
- Statistics on data

SDE Advantages:

- Dealing with dynamic data
- Evaluating one single SPARQL query

Experimental Performances

Experiments

- Cluster of 10 nodes with 17GB of RAM each
- LUBM & WatDiv

Competitors

- Selection criteria: HDFS-based, OpenSource, Popular and Recent
- Two types of evaluators:
    - Conventional (with preprocessing): RYA, CliqueSquare and S2RDF
    - Direct: PigSPARQL, and RDFHive

Experimental Performances

Datasets

Dataset	Number of Triples	Original File Size on HDFS
Watdiv1k	109 million	46.8 GB
Lubm1k	134 million	72.0 GB
Lubm10k	1.38 billion	747 GB

Summary

1. SPARQLGX answers all WatDiv/LUBM queries unlike many competitors
2. SPARQLGX is the fastest among those capable of answering all queries
3. SDE outperforms other direct evaluators
4. SDE is even sometimes faster than conventional ones

Detailed results at: `http://tyrex.inria.fr/sparql-comparative/`

Conclusion

We provide:

- SPARQLGX
- SDE

They are:

- Efficient
- Available from: `https://github.com/tyrex-team/sparqlgx`

Thank you.

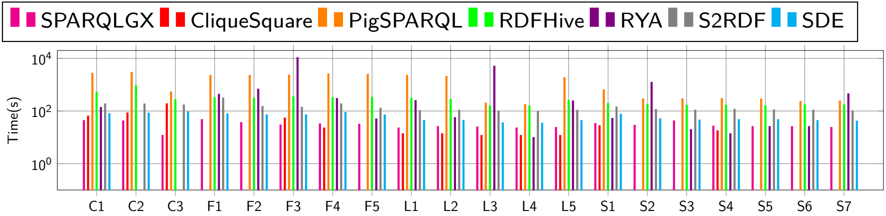
Statistics Rewriting

Selectivity

- Selectivity of an element located at pos is: either its occurrence number at pos if it is a constant or the total number of triples if it is a variable.
- Selectivity of a TP is the min of its element selectivities.

We just sort the TPs of a BGP in ascending order of their selectivities.

Query Response Time with WatDiv1k



Query Response Time with Lubm1k