SPARQLGX:
Efficient Distributed Evaluation of SPARQL with Spark

Damien Graux, Louis Jachiet, Pierre Genevès, Nabil Layaida

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

Context

- Large amounts of RDF data $\Rightarrow$ distribution
Motivations

Context

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

Context

- Large amounts of RDF data $\Rightarrow$ 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 $\Rightarrow$ 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 $\rightarrow$ 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

D. Graux et al.  
SPARQLGX – https://github.com/tyrex-team/sparqlgx  
3/11
Graph and Triples, an example

`s type tool
?g type human
?s contributor ?g
Graph and Triples, an example

\[
\text{?s type tool} \\
\text{?g type human} \\
\text{?s contributor ?g}
\]

?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

<table>
<thead>
<tr>
<th>dataset</th>
<th>sparqlgx</th>
<th>type</th>
<th>tool</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>sparqlgx</td>
<td>creationDate</td>
<td>2016</td>
</tr>
<tr>
<td></td>
<td>spark</td>
<td>use</td>
<td>spark</td>
</tr>
<tr>
<td>Damien</td>
<td>type</td>
<td>human</td>
<td></td>
</tr>
<tr>
<td>Louis</td>
<td>type</td>
<td>human</td>
<td></td>
</tr>
<tr>
<td>Pierre</td>
<td>type</td>
<td>human</td>
<td></td>
</tr>
<tr>
<td>Nabil</td>
<td>type</td>
<td>human</td>
<td></td>
</tr>
<tr>
<td>RDFHive</td>
<td>contributor</td>
<td>Damien</td>
<td></td>
</tr>
<tr>
<td>sparqlgx</td>
<td>creator</td>
<td>Damien</td>
<td></td>
</tr>
<tr>
<td>sparqlgx</td>
<td>contributor</td>
<td>Damien</td>
<td></td>
</tr>
<tr>
<td>sparqlgx</td>
<td>contributor</td>
<td>Damien</td>
<td></td>
</tr>
<tr>
<td>sparqlgx</td>
<td>contributor</td>
<td>Louis</td>
<td></td>
</tr>
<tr>
<td>sparqlgx</td>
<td>contributor</td>
<td>Pierre</td>
<td></td>
</tr>
<tr>
<td>sparqlgx</td>
<td>contributor</td>
<td>Nabil</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>type.txt</th>
<th>sparqlgx</th>
<th>tool</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>sparqlgx</td>
<td>tool</td>
</tr>
<tr>
<td></td>
<td>spark</td>
<td>tool</td>
</tr>
<tr>
<td>Damien</td>
<td>human</td>
<td></td>
</tr>
<tr>
<td>Louis</td>
<td>human</td>
<td></td>
</tr>
<tr>
<td>Pierre</td>
<td>human</td>
<td></td>
</tr>
<tr>
<td>Nabil</td>
<td>human</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>creationDate.txt</th>
<th>sparqlgx</th>
<th>2016</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>use.txt</th>
<th>sparqlgx</th>
<th>spark</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>creator.txt</th>
<th>sparqlgx</th>
<th>Damien</th>
</tr>
</thead>
<tbody>
<tr>
<td>RDFHive</td>
<td>Damien</td>
<td></td>
</tr>
<tr>
<td>sparqlgx</td>
<td>Damien</td>
<td></td>
</tr>
<tr>
<td>sparqlgx</td>
<td>Louis</td>
<td></td>
</tr>
<tr>
<td>sparqlgx</td>
<td>Pierre</td>
<td></td>
</tr>
<tr>
<td>sparqlgx</td>
<td>Nabil</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>contributor.txt</th>
<th>RDFHive</th>
<th>Damien</th>
</tr>
</thead>
<tbody>
<tr>
<td>sparqlgx</td>
<td>Damien</td>
<td></td>
</tr>
<tr>
<td>sparqlgx</td>
<td>Louis</td>
<td></td>
</tr>
<tr>
<td>sparqlgx</td>
<td>Pierre</td>
<td></td>
</tr>
<tr>
<td>sparqlgx</td>
<td>Nabil</td>
<td></td>
</tr>
</tbody>
</table>
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

\[
\text{?s type tool .} \\
\text{?g type human .} \\
\text{?s contributor ?g}
\]
SPARQL Translation Process

\( ?s \text{ type tool .} \)
\( ?g \text{ type human .} \)
\( ?s \text{ contributor } ?g \)

\[
\text{tp1=sc.textFile(''type.txt'')}
\text{.filter\{case(s,o)=&gt;o.equals(''tool'')\}}
\text{.map\{case(s,o)=&gt;s\}}
\text{.keyBy\{case(s)=&gt;s\}}
\]
?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}
```

```text
```
?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

```scala
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:

- Vertical 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

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Number of Triples</th>
<th>Original File Size on HDFS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Watdiv1k</td>
<td>109 million</td>
<td>46.8 GB</td>
</tr>
<tr>
<td>Lubm1k</td>
<td>134 million</td>
<td>72.0 GB</td>
</tr>
<tr>
<td>Lubm10k</td>
<td>1.38 billion</td>
<td>747 GB</td>
</tr>
</tbody>
</table>

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

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

The diagram shows the query response time for various systems, with different colored bars representing different systems. The x-axis represents the queries labelled from C1 to S7, and the y-axis represents the time in seconds on a logarithmic scale ranging from 10^0 to 10^4.
Query Response Time with Lubm1k

![Bar chart showing query response times for various systems.](https://github.com/tyrex-team/sparqlgx)