Using semijoin programs to solve traversal queries in graph databases

Norbert Martínez, David Domínguez

June 22, 2014 – Snowbird, Utah, USA
# Motivation

## State-of-the-art in the management of large graphs

<table>
<thead>
<tr>
<th>Technology</th>
<th>Solutions</th>
<th>Querying systems</th>
</tr>
</thead>
<tbody>
<tr>
<td>RDF</td>
<td>Virtuoso RDF–3X</td>
<td>SPARQL</td>
</tr>
<tr>
<td></td>
<td>...</td>
<td></td>
</tr>
<tr>
<td>Distributed frameworks</td>
<td>Pregel</td>
<td>API</td>
</tr>
<tr>
<td></td>
<td>GraphLab</td>
<td>DSL (e.g. Green-Marl)</td>
</tr>
<tr>
<td></td>
<td>...</td>
<td></td>
</tr>
<tr>
<td>Graph Databases</td>
<td>Sparksee (DEX)</td>
<td>API (e.g. BluePrints)</td>
</tr>
<tr>
<td></td>
<td>Neo4J</td>
<td>Gremlin</td>
</tr>
<tr>
<td></td>
<td>Titan</td>
<td>Cypher</td>
</tr>
<tr>
<td></td>
<td>...</td>
<td>GQL</td>
</tr>
<tr>
<td></td>
<td>...</td>
<td></td>
</tr>
</tbody>
</table>
Motivation

• Querying graph databases
  - lack of standards
  - very efficient proprietary APIs for procedural languages
  - DSL or languages that combine data flows of results from APIs

• Open challenges for GDB querying systems
  - define an algebra of graph query operations
  - optimization of graph query plans
  - generic solution to solve different graph queries
    - edge traversals
    - attribute filters and aggregation
    - graph pattern matching
    - graph algorithms
• An algebra with a set of operations to solve graph queries in Sparksee
  - adapted from the relational algebra
    - select, semijoin, project, group, sort, ...
  - extensions
    - foreach: collection-oriented procedures (parallel)
    - while-do: iterative (recursive) programs
  - data structures in the form of key-value pairs
  - compatible with the current Sparksee API
• Optimization of graph query plans
  - classical database query optimization techniques
  - new specific for graph patterns and graph traversals
  - take advantage of Sparksee's compressed bitmap
Sparksee graph representation

**OBJECTS**

**NODES**
- Person
- Post
- Tag

**EDGES**
- hasTag
- likes
- knows
- hasCreator
- hasInterest

**RELATIONSHIPS**

**TAILS: hasTag**
- edge
- node
- edges

**HEADS: hasTag**
- edge
- node
- edges

**TAILS: likes**
- edge
- node
- edges

**HEADS: likes**
- edge
- node
- edges

**TAILS: knows**
- edge
- node
- edges

**HEADS: knows**
- edge
- node
- edges

**TAILS: hasCreator**
- edge
- node
- edges

**HEADS: hasCreator**
- edge
- node
- edges

**TAILS: hasInterest**
- edge
- node
- edges

**HEADS: hasInterest**
- edge
- node
- edges

**ATTRIBUTES**

**ATTRIBUTE: name**
- id
- string
- oids

**ATTRIBUTE: language**
- id
- string
- oids

**ATTRIBUTE: date**
- id
- timestamp
- oids

**ATTRIBUTE: content**
- id
- string
- oids

**ATTRIBUTE: image**
- id
- string
- oids

**OV_ATTR_LANGUAGE**

<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>ca</td>
</tr>
<tr>
<td>4</td>
<td>en</td>
</tr>
<tr>
<td>5</td>
<td>ca</td>
</tr>
</tbody>
</table>

**VOS_ATTR_LANGUAGE**

<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>{3, 5}</td>
</tr>
</tbody>
</table>

*Sparsity Technologies — Powering Extreme Data*
Sparksee APIs

NODES(T) ::= \textit{select} (\textit{scan} ("VOS\_LABELS"), k=T)

NEIGHBORS\_OUT(N, E) ::= \textit{semijoin} (N, \textit{scan} ("OUT\_" || E), k=k)

DEGREE\_OUT(N, E) ::= \textit{group} (NEIGHBORS\_OUT(N, E), [], [\textit{sum} (\textit{length} (v))])

Graph Algorithms

HUB(N, E) ::= let D = \textit{foreach} X in NODES(N)

\hspace{1cm} \text{DEGREE\_OUT}(X, E)

\hspace{2cm} in \textit{semijoin} (D, \textit{group} (D, [], [\textit{max} (v)]), v=v)

BFS(N, E, K) ::= \textit{with} R=N

\hspace{1cm} \textit{do} S=R,

\hspace{2cm} R=NEIGHBORS\_OUT(S, E)

\hspace{1cm} \textit{until} \textit{semijoin} (R, S, k<>k)

\hspace{1cm} \textit{steps} K

\hspace{1cm} \textit{return} R
<table>
<thead>
<tr>
<th>Operation</th>
<th>Q1</th>
<th>Q2</th>
<th>Q3</th>
<th>Q4</th>
<th>Q5</th>
<th>Q6</th>
<th>Total</th>
<th>%</th>
</tr>
</thead>
<tbody>
<tr>
<td>EQUIJOIN</td>
<td>16</td>
<td>6</td>
<td>4</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>29</td>
<td>11.2%</td>
</tr>
<tr>
<td>FOREACH</td>
<td>3</td>
<td>4</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td>8</td>
<td>3.1%</td>
</tr>
<tr>
<td>GROUP</td>
<td>4</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>9</td>
<td>3.5%</td>
</tr>
<tr>
<td>LIMIT</td>
<td>1</td>
<td>1</td>
<td></td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>5</td>
<td>1.9%</td>
</tr>
<tr>
<td>PRODUCT</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td>9</td>
<td>3.5%</td>
</tr>
<tr>
<td>PROJECT</td>
<td>12</td>
<td>7</td>
<td>5</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>27</td>
<td>10.4%</td>
</tr>
<tr>
<td>SCAN</td>
<td>23</td>
<td>14</td>
<td>7</td>
<td>6</td>
<td>7</td>
<td>6</td>
<td>63</td>
<td>24.2%</td>
</tr>
<tr>
<td>SELECT</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td>17</td>
<td>6.5%</td>
</tr>
<tr>
<td>SEMIJOIN</td>
<td><strong>30</strong></td>
<td><strong>13</strong></td>
<td><strong>11</strong></td>
<td><strong>7</strong></td>
<td><strong>10</strong></td>
<td><strong>10</strong></td>
<td><strong>81</strong></td>
<td><strong>31.2%</strong></td>
</tr>
<tr>
<td>SORT</td>
<td>1</td>
<td>1</td>
<td></td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>5</td>
<td>1.9%</td>
</tr>
<tr>
<td>UNION</td>
<td>1</td>
<td>3</td>
<td>1</td>
<td></td>
<td>1</td>
<td>1</td>
<td>7</td>
<td>2.7%</td>
</tr>
</tbody>
</table>
Friends and friends of friends

Forum membership and posts

*Sparsity Technologies — Powering Extreme Data*
semijoin(R, S, R.x = S.y):

- R.x is an oid (key \(k\))
  - usually R is a persistent KVP and R.x is indexed
  - S.y is a collection of oids: bitmap union and test

- R.x is a collection of oids (values \(v\))
  - in general R is a temporary result inside the query pipeline
  - bitmap intersections

<table>
<thead>
<tr>
<th>semijoin</th>
<th>Q1</th>
<th>Q2</th>
<th>Q3</th>
<th>Q4</th>
<th>Q5</th>
<th>Q6</th>
<th>Total</th>
<th>%</th>
</tr>
</thead>
<tbody>
<tr>
<td>k=k</td>
<td>23</td>
<td>6</td>
<td>2</td>
<td>4</td>
<td>1</td>
<td>2</td>
<td>38</td>
<td>46.9%</td>
</tr>
<tr>
<td>k=v</td>
<td>7</td>
<td>7</td>
<td>8</td>
<td>3</td>
<td>6</td>
<td>5</td>
<td>36</td>
<td>44.5%</td>
</tr>
<tr>
<td>v=k</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td>2</td>
<td>4</td>
<td>4.9%</td>
</tr>
<tr>
<td>v=v</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>3.7%</td>
</tr>
</tbody>
</table>
Conclusions

• Our proposal is a compromise between:
  - the relational model and the noSQL approach
  - key-value pairs storages and graph querying systems

• The main operation is the semijoin:
  - widely used in database technology for distributed systems
  - semijoin program optimization has been studied in detail
  - specific semijoin optimizations on compressed collections of oids

• Procedural extensions for parallel subquery execution and recursion
  - used to simulate graph analytical frameworks for the computation of complex graph algorithms
Thanks!

Q&A