Graph + Relational Design

Graph Analysis Tasks Vary Widely
- Different types of Graph Queries
- Continuous Queries / Real-Time Analysis
- Batch Graph Analytics
- Machine Learning
- Many different ways to deal with graph data
- Graph Databases (Neo4j, OrientDB, RDF stores)
- Distributed Batch Analysis Frameworks (Giraph, GraphX, GraphLab)
- In-Memory Systems (Ligra, Green-Marl, X-Stream)
- Many research prototypes/custom indexes

1. Graph Frontend, Graph Backend
   - RDF, Property Graph databases
   - Focus on graph queries
   - Issue: Requires a complete buy-in

2. Graph Frontend, Relational Backend
   - Some XML/RDF Property Graph Database (e.g., SQLGraph, Titan)
   - Graph analytics frameworks (Vertica, Giraph)
   - Issue: Requires complete buy-in; limited expressive power for analytics

3. Relational Frontend, Graph Engine
   - Using a graph engine to efficiently process SQL queries over relational databases
   - Mostly to tackle queries involving long series of joins.

4. Graph + Relational Frontend, Relational Backend (GraphGen)
   - Aster Graph, SAP Graph Engine
   - Enterprises have existing relational databases with rigid schemas
   - Need to enable analysis of the hidden graphs within them
   - While continuing to support SQL queries/analytics

GraphGen

GraphGenQL
- Declarative specification of GraphViews over the database
- User specifies Nodes and Edges
- CoAuthors GraphView

CREATE GRAPHVIEW CoAuthors AS
  Node((ID, name) :- Author((ID, name)),
  Edge(ID1, ID2, wt=$degNY($pub)) :-
    AuthorPub((ID1, pub),
    AuthorPub((ID2, pub), Author((ID1), name),
    Edge((ID1, ID2), pub), AuthorPub((ID2, pub),
    Edge((ID1, ID2), pub),
    AuthorPub((ID2, pub), Author((ID1), name)),
    Edge((ID1, ID2), pub),
    AuthorPub((ID2, pub), Author((ID1), name))
WHERE Author((X)) AS
  Node(X, name) :- Author((X, name)),
  Node(ID, name) :- AuthorPub((ID, pub),
  Edge((ID1, ID2), pub), AuthorPub((ID1, pub),
  AuthorPub((ID2, pub), Author((ID1), name),
  Edge((ID1, ID2), pub), AuthorPub((ID2, pub),
  Author((ID1), name)),
  Edge((ID1, ID2), pub),
  AuthorPub((ID2, pub), Author((ID1), name)))
WITH Edges as ...

GraphGenQL for relational databases with 
- Edge-centric views
- Node-centric views

Specifying Analysis Tasks
- Vertex-centric works for many situations but not very expressive
  We plan to support:
  - Direct access to graph views for complex programs
  - Dataflow-based DSL (build upon languages like Socialite)

Opportunities and Challenges

Where to Execute Tasks?
- Depending on workload, rate of updates, rate of queries...
- Usually in-memory is faster, but ETL may not be worth it
- Other issues: Large output joins, and selectivity estimation errors associated with them.

Key Challenge: Develop accurate cost models, workload monitoring tools and optimization techniques
- How much of the graph do we pre-compute / materialize?
- Incremental view maintenance for graph-views may prove challenging

Query Rewriting
- Auto-generated SQL can consist of many blocks
- Many ways of writing equivalent SQL queries
- Optimization can be challenging
- Could define edges using the MTH clause, or as a VED

Optimizing the Extraction of Multi-Graph Views
- Typically one would need to generate a separate query for each graph
- We employ a technique called result-tagging

Preiminary Experiments

<table>
<thead>
<tr>
<th>Query</th>
<th>DBS1</th>
<th>DBS2</th>
<th>MySQL</th>
<th>PostgreSQL</th>
</tr>
</thead>
<tbody>
<tr>
<td>With (at edges)</td>
<td>1</td>
<td>1.62</td>
<td>NA</td>
<td>13.8</td>
</tr>
<tr>
<td>With (at the end)</td>
<td>53.028</td>
<td>2.99</td>
<td>NA</td>
<td>37.8</td>
</tr>
<tr>
<td>View (at edges)</td>
<td>1.054</td>
<td>2.07</td>
<td>3.01</td>
<td>15.6</td>
</tr>
<tr>
<td>View (at the end)</td>
<td>51.92</td>
<td>77.13</td>
<td>538.19</td>
<td>35.99</td>
</tr>
<tr>
<td>On Base Table</td>
<td>46.45</td>
<td>74.878</td>
<td>678.87</td>
<td>36.16</td>
</tr>
</tbody>
</table>

Triangle Counting (small); time in seconds

<table>
<thead>
<tr>
<th>Query</th>
<th>DBS1</th>
<th>MySQL</th>
<th>PostgreSQL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Using With (in edge)</td>
<td>48.05</td>
<td>NA</td>
<td>477.56</td>
</tr>
<tr>
<td>Using With (at the end)</td>
<td>2404</td>
<td>NA</td>
<td>1499.33</td>
</tr>
<tr>
<td>Using View (in edges)</td>
<td>44.72</td>
<td>357.59</td>
<td>811</td>
</tr>
<tr>
<td>Using View (at the end)</td>
<td>2377.61</td>
<td>&gt;3600</td>
<td>1774.54</td>
</tr>
<tr>
<td>Directly On Base Table</td>
<td>2348</td>
<td>&gt;3600</td>
<td>1790</td>
</tr>
</tbody>
</table>

Triangle Counting (large); time in seconds

Query WRITERS rewrites lead to significant differences in performance
- In-memory execution is usually faster but when its warranted

Dataset Sizes
- small: 100,000, 1.639, 55.436
- large: 500,000, 15,741, 529.434

Preliminary Results

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Triangle Counting</th>
<th>Triangle Pattern</th>
<th>ETL</th>
</tr>
</thead>
<tbody>
<tr>
<td>small</td>
<td>0.169</td>
<td>0.001</td>
<td>2.049</td>
</tr>
<tr>
<td>large</td>
<td>6.723</td>
<td>0.015</td>
<td>17.52</td>
</tr>
</tbody>
</table>

Patterns Matching where area='ML' (large); time in seconds