GraphiQL
Graph Intuitive Query Language for Relational Databases

Alekh Jindal

Amol Deshpande
work
University of Maryland

Sam Madden
work
Mike Stonebraker

IEEE BigData 2014

Supervisors
work
collaborate
sabbatical
work
work

work

at
Talking on
GraphiQL
Graph Intuitive Query Language for Relational Databases

Alekh Jindal
IEEE BigData 2014
Talking on
Collaborate

Amol Deshpande
University of Maryland
Sabbatical

Sam Madden
Supervisors

Mike Stonebraker

MIT
Expensive!
Graph Analysis =

Graph Algorithms +

Store
Extract
Preprocess
Update
Failover
Postprocess
Graph Analysis =

Relational Database

Graph Algorithms

Store
Extract
Preprocess
Update
Failover
Postprocess

“Counting Triangles with Vertica”
“Scalable Social Graph Analytics Using the Vertica Analytic Platform,”
“Graph Analysis: Do We Have to Reinvent the Wheel?”
“Query Optimization of Distributed Pattern Matching,”
“GraphX: A Resilient Distributed Graph System on Spark,”
“Vertexica: Your Relational Friend for Graph Analytics!”
Problem!
Alekh Jindal
GraphiQL
Graph Intuitive Query Language for Relational Databases
Sam Madden
Mike Stonebraker
Amol Deshpande
MIT
University of Maryland
IEEE BigData 2014
Talking on
Supervisors
work
work
work
sabbatical
work
work
Mike Stonebraker

SQL
SELECT
COUNT
UPDATE
FROM
GROUP BY
SUM
WHERE
Redundant Effort
Optimizations?
GraphiQL
Key Features

• Graph view of relational data; the system takes care of mapping to the relational world

• Inspired from PigLatin: right balance between declarative and procedural style language

• Key graph constructs: looping, recursion, neighborhood access

• Compiles to optimized SQL
Graph Table

GraphiQL

SQL

Graph Table

Relational Table

Alekh Jindal

Sam Madden

Mike Stonebraker

MIT

University of Maryland

IEEE BigData 2014

Supervisors

collaborate

work

sabbatical

work
### Graph Table

<table>
<thead>
<tr>
<th>Graph Elements</th>
<th>id</th>
<th>weight</th>
<th>type</th>
</tr>
</thead>
<tbody>
<tr>
<td>node1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>node2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>edge1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>edge2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>edge3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>node3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>node4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>node5</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Incoming:**
- node8
- node9
- edge7
- edge8
- edge9

**Outgoing:**
- node6
- node7
- edge4
- edge5
- edge6
Graph Table Definition

- **Create**
  ```sql```
  CREATE GRAPHTABLE g AS
  NODE (p1,p2,..)
  EDGE (q1,q2,..)
  ```sql```

- **Load**
  ```sql```
  LOAD g AS
  NODE FROM graph_nodes DELIMITER d
  EDGE FROM graph_edges DELIMITER d
  ```sql```

- **Drop**
  ```sql```
  DROP GRAPHTABLE g
  ```sql```
Graph Table Manipulation

- **Iterate**
  
  \[
  \text{FOREACH element in } g \\
  \text{[WHILE condition]}
  \]

- **Filter**

  \[
  g' = g(k_1=v_1, k_2=v_2, \ldots, k_n=v_n)
  \]

- **Retrieve**

  \[
  \text{GET expr}_1, \text{expr}_2, \ldots, \text{expr}_n \\
  \text{[WHERE condition]}
  \]

- **Update**

  \[
  \text{SET variable TO expr} \\
  \text{[WHERE condition]}
  \]

- **Aggregate**

  \[
  \text{SUM, COUNT, MIN, MAX, AVG}
  \]
## Nested Manipulation

<table>
<thead>
<tr>
<th>outer</th>
<th>inner</th>
<th>Iterate</th>
<th>Aggregate</th>
<th>Retrieve</th>
<th>Update</th>
</tr>
</thead>
<tbody>
<tr>
<td>Iterate</td>
<td></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Aggregate</td>
<td></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Retrieve</td>
<td></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Update</td>
<td></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>
Example 1: PageRank

FOREACH n IN g(type=N)
    SET n.pr TO new_pr
Example 1: PageRank

FOREACH n IN g(type=N)
    SET n.pr TO 0.15/num_nodes + 0.85*SUM(pr_neighbors)
Example 1: PageRank

FOREACH n IN g(type=N)
    SET n.pr TO 0.15/num_nodes + 0.85*SUM(
        FOREACH n' IN n.in(type=N)
            GET pr_n'
    )
Example 1: PageRank

FOREACH n IN g(type=N)
  SET n.pr TO 0.15/num_nodes + 0.85*SUM(
    FOREACH n’ IN n.in(type=N)
      GET n’.pr/COUNT(n’.out(type=N))
  )
Example 1: PageRank

\[
\text{FOREACH iterations IN [1:10]} \\
\quad \text{FOREACH n IN g(type=N)} \\
\quad \quad \text{SET n.pr TO } 0.15/\text{num\_nodes} + 0.85*\text{SUM(} \\
\quad \quad \quad \text{FOREACH n' IN n.in(type=N)} \\
\quad \quad \quad \quad \text{GET n'.pr/COUNT(n'.out(type=N))} \\
\quad \quad \)} \]
Example 1: PageRank

FOREACH iterations IN [1:10]
  FOREACH n IN g(type=N)
    SET n.pr TO 0.15/num_nodes + 0.85*SUM(
      FOREACH n’ IN n.in(type=N)
        GET n’.pr/COUNT(n’.out(type=N))
    )
Example 2: SSSP

FOREACH n IN g(type=N)
    SET n.dist TO min_dist
Example 2: SSSP

```
FOREACH n IN g(type=N)
SET n.dist TO MIN(n.in(type=N).dist)+1 AS dist'
WHERE dist' < n.dist
```
Example 2: SSSP

\[
\text{WHILE updates > 0} \\
\text{FOREACH } n \text{ IN } g(\text{type}=N) \\
\text{updates =} \\
\text{SET } n.\text{dist} \text{ TO } \text{MIN}(n.\text{in}(\text{type}=N).\text{dist})+1 \text{ AS dist'} \\
\text{WHERE dist'} < n.\text{dist}
\]
Example 2: SSPP

SET g(type=N).dist TO inf
SET g(type=N,id=start).dist TO 0
WHILE updates > 0
    FOREACH n IN g(type=N)
        updates =
            SET n.dist TO MIN(n.in(type=N).dist)+1 AS dist'
        WHERE dist’ < n.dist
GraphiQL Compiler

- Graph Table manipulations to relational operators:
  - filter → selection predicates
  - iterate → driver loop
  - retrieve → projections
  - update → update in place
  - aggregate → group-by aggregate

- Graph Tables to relational tables:
  - mapping
GraphiQL Compiler

\[
g(\text{type=}N) \mapsto N \\
g(\text{type=}E) \mapsto E \\
g(\text{type=}N).\text{out}(\text{type=}E) \mapsto N \bowtie E \\
g(\text{type=}E).\text{out}(\text{type=}E) \mapsto E \bowtie E \\
g(\text{type=}N).\text{out}(\text{type=}N) \mapsto N \bowtie E \bowtie N \\
g.\text{out}.\text{in} = g.\text{in} \\
g.\text{in}.\text{out} = g.\text{out}
\]
Example: SSSP

SET g(type=N).dist TO inf
SET g(type=N,id=start).dist TO 0
WHILE updates > 0
    FOREACH n IN g(type=N)
        updates =
        SET n.dist TO
        MIN(n.in(type=N).dist)+1 AS dist'
        WHERE dist' < n.dist

\[\text{updateCount}>0 \ (\]
\[n.d\text{ist} \leftarrow \sigma_{n.d\text{ist}>dist'} (\]
\[\gamma_{\min(n'.d\text{ist})+1 (\]
\[\Gamma_{n.id (\]
\[N \Join E \Join N' \]
\[) \] \] \) \) \) \)
GraphiQL Optimizations

- De-duplicating graph elements
- Selection pushdown
- Cross-product as join
- Pruning redundant joins
Performance
Performance

Machine:
2GHz, 24 threads, 48GB memory, 1.4TB disk
Performance

Machine:
2GHz, 24 threads, 48GB memory, 1.4TB disk

Dataset:
Small: 81k/1.7m directed; 334k/925k undirected
Large: 4.8m/68m directed; 4m/34m undirected
Performance - small graph

Time (seconds)

Apache Giraph
GraphiQL

12x Speedup!
Performance - large graph

- Apache Giraph
- GraphiQL

4.3x Speedup!
Summary

• Several real world graph analysis are better off in relational databases

• We need both the graph as well as relational view of data

• GraphiQL introduces Graph Tables to allows users to think in terms of graphs

• Graph Table supports recursive association, nested manipulations, and SQL compilation

• GraphiQL allows users to easily write a variety of graph analysis
Thanks!
Other Languages

Imperative languages: e.g. Green Marl
XPath: e.g. Cypher, Gremlin
Datalog: e.g. Socialite
SPARQL: Teradata blog
Procedural language: e.g. Vertex-centric