Graph Analytics using Vertica Relational Database

Alekh Jindal - Samuel Madden, Malú Castellanos - Meichun Hsu
Introduction

- High demand for graph analytics
- Popularity of distributed graph computing systems
  - Vertex-centric systems: Pregel, Giraph, GraphLab
- Question:

  Are traditional relational database systems not good enough for graph analytics?
Introduction

Limitations of distributed graph computing systems:

- Data is initially collected and stored in a relational database
- Graph processing is slow for very large graphs
  - Users have to choose a subgraph to run the algorithm
- Preparation might include operations that relational databases are optimized for.
  - Pre-processing or post-processing
- Some graph algorithms compute aggregates over a large neighbourhood
  - Hard to express in vertex-centric systems
Goal

- Show how vertex-centric graph processing can be translated, optimized and run on Vertica
  - SSSP, PageRank, Connected Components

- Compare Performance with two vertex-centric distributed systems for graph analysis (Giraph and GraphLab)

- Vertica → Enterprise column-store database management system
  - Supports parallel processing
Vertex-Centric Model

- The user provides a `vertex.compute` function (UDF):
  - The UDF will be executed at each node.
  - Will update the node’s state.
  - And communicate the changes to the neighbours.
Giraph Physical Plan

1. Input Superstep: Workers reading the data, building “Server Data stores”

2. Intermediate step: Run UDF, shuffle messages, wait for everyone, synchronize.

3. Output Superstep: Produce the output.
Giraph Logical Plan

Same query plan but in relational logic:

1. V join E
2. (V join E) join M: messages from previous superstep
3. Run UDF
4. Produce new state for vertex (V') and messages for the next superstep (M').
Overview

● Translation to SQL queries
● Query Optimization
● Query Execution
● Extending Vertica
Translation to SQL

1) Eliminate the message table

1. Giraph logical query plan

2. Pushing down the vertexCompute UDF

3. Replacing $M$ by $V \mathbin{\wedge} E$
Translation to SQL

2) Translate vertex compute function

SSSP : $\text{vertexCompute} \quad \rightarrow \quad \sigma_{d' < V_1.d} (\Gamma_{d'} = \min(V_2.d + 1))$

```
UPDATE vertex AS v SET v.d = v'.d
FROM (SELECT v1.id, MIN(v2.d+1) AS d
      FROM vertex AS v1, edge AS e, vertex AS v2
      WHERE v2.id = e.from_node AND v1.id = e.to_node
      GROUP BY e.to_node, v1.d
      HAVING MIN(v2.d+1) < v1.d
     ) AS v'
WHERE v.id = v'.id;
```
Query Optimizations

1) Update Vs. Replace

```
CREATE TABLE vertex_prime AS
    SELECT v.id, ISNULL(v’.d, v.d) AS d
    FROM vertex AS v LEFT JOIN (  
        SELECT v1.id AS id, MIN(v2.d+1) AS d  
        FROM vertex AS v1, edge AS e, vertex AS v2  
        WHERE v2.id=e.from_node AND v1.id=e.to_node  
        GROUP BY e.to_node, v1.d  
        HAVING MIN(v2.d+1) < v1.d  
    ) AS v’  
    ON v.id = v’.Id;
```
2) Incremental Evaluation

```sql
CREATE TABLE v_update_prime AS
    SELECT v1.id, MIN(v2.d+1) AS d
    FROM v_update AS v2, edge AS e, vertex AS v1
    WHERE v2.id = e.from_node AND v1.id = e.to_node
    GROUP BY e.to_node, v1.d
    HAVING MIN(v2.d+1) < v1.d;

DROP TABLE v_update;
ALTER TABLE v_update_prime RENAME TO v_update;

CREATE TABLE vertex_prime AS
    SELECT v.id, ISNULL(v.update.d, v.d) AS value
    FROM vertex AS v LEFT JOIN v_update
    ON v.id = v.update.id;

DROP TABLE vertex; ALTER TABLE vertex_prime RENAME TO vertex;
```
Query Optimizations

2) Join Elimination

Join Elimination in PageRank
Query Execution

● Physical Design
  ○ Encoding and compression, sort orders, multiple table projections

● Join Optimization
  ○ Join directly over compressed data, choose from hash join and merge join

● Query Pipelining
  ○ Avoids materializing intermediate output and repeated access to disk

● Intra-query Parallelism
  ○ Process subgraphs in parallel across cpu cores using GroupBy
Query Execution Plan of SSSP

Different from Giraph execution pipeline:

1. Filter unnecessary tuples as early as possible.
2. Fully pipelines the execution flow.
3. Picks the best join execution strategy.
Extending Vertica

- Running unmodified vertex programs
  - As table UDFs without translating to relational operators
Extending Vertica

- Avoiding Intermediate Disk I/O
  - Load and store graph in shared memory, higher memory footprint
Experiments

Setup:

- Cluster of 4 machines
- 48 GB memory
- 1.4 TB Disk

Dataset:

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Nodes</th>
<th>Edges</th>
</tr>
</thead>
<tbody>
<tr>
<td>Directed</td>
<td>Twitter-small</td>
<td>81,306</td>
<td>1,768,149</td>
</tr>
<tr>
<td></td>
<td>LiveJournal</td>
<td>4,847,571</td>
<td>68,993,773</td>
</tr>
<tr>
<td></td>
<td>Twitter</td>
<td>41,652,230</td>
<td>1,468,365,182</td>
</tr>
<tr>
<td>Undirected</td>
<td>YouTube</td>
<td>1,134,890</td>
<td>2,987,624</td>
</tr>
<tr>
<td></td>
<td>LiveJournal-undir</td>
<td>3,997,962</td>
<td>34,681,189</td>
</tr>
</tbody>
</table>
Experiments

Data Preparation:

<table>
<thead>
<tr>
<th>Metric</th>
<th>Dataset</th>
<th>Vertica</th>
<th>GraphLab</th>
<th>Giraph</th>
</tr>
</thead>
<tbody>
<tr>
<td>Upload Time (sec)</td>
<td>LiveJournal</td>
<td>45.927</td>
<td>15.621</td>
<td>12.049</td>
</tr>
<tr>
<td></td>
<td>Twitter</td>
<td>916.421</td>
<td>472.358</td>
<td>267.799</td>
</tr>
<tr>
<td>Disk Usage (GB)</td>
<td>LiveJournal</td>
<td>0.423</td>
<td>3.030</td>
<td>3.030</td>
</tr>
<tr>
<td></td>
<td>Twitter</td>
<td>9.964</td>
<td>73.140</td>
<td>73.140</td>
</tr>
</tbody>
</table>

Runtime:
Experiments

Memory Usage (PageRank):
Experiments

In memory Graph Analysis:
Experiments

Mixed Graph and Relational Analysis:

<table>
<thead>
<tr>
<th>Query</th>
<th>Dataset</th>
<th>Vertica</th>
<th>Giraph</th>
<th>SpeedUp</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sub-graph Projection &amp; Selection</td>
<td>PR</td>
<td>55.6</td>
<td>954.6</td>
<td>17.2</td>
</tr>
<tr>
<td></td>
<td>SSSP</td>
<td>101.3</td>
<td>405.5</td>
<td>4.0</td>
</tr>
<tr>
<td>Graph Analysis Aggregation</td>
<td>PR</td>
<td>643.9</td>
<td>1089.7</td>
<td>1.7</td>
</tr>
<tr>
<td></td>
<td>SSSP</td>
<td>279.8</td>
<td>349.9</td>
<td>1.3</td>
</tr>
<tr>
<td>Graph Joins</td>
<td>PR+SSSP</td>
<td>927.0</td>
<td>1435.9</td>
<td>1.5</td>
</tr>
</tbody>
</table>

More Complicated Graph Processing:

<table>
<thead>
<tr>
<th>Query</th>
<th>Dataset</th>
<th>Vertica</th>
<th>Giraph</th>
</tr>
</thead>
<tbody>
<tr>
<td>Strong Overlap</td>
<td>Youtube</td>
<td>259.56</td>
<td>230.01</td>
</tr>
<tr>
<td></td>
<td>LiveJournal-undir</td>
<td>381.05</td>
<td>out of memory</td>
</tr>
<tr>
<td>Weak Ties</td>
<td>Youtube</td>
<td>746.14</td>
<td>out of memory</td>
</tr>
<tr>
<td></td>
<td>LiveJournal-undir</td>
<td>1,475.99</td>
<td>out of memory</td>
</tr>
</tbody>
</table>
Conclusion

- Vertica can be tuned to offer good end-to-end performance on graph queries (because it is optimized for scans, joins and aggregates).

- Users can trade memory with reduced I/O cost in iterative graph analysis.

- Relational databases can combine graph processing with relational analysis as pre-processing or post-processing steps.

- Features of relational databases can be combined with graph processing systems and it might be a good idea to stitch these systems together.
Thank you for your attention.