The case against specialized graph engines

Jing Fan, Adalbert Gerald Soosai Raj, and Jignesh M. Patel
University of Wisconsin – Madison
Motivation

• Graph analytics is now common
• Response = new specialized graph engines
Motivation

- Graph analytics is now common
- Response = new specialized graph engines

graphs, 18%

others, 82%
Motivation

• Graph analytics is now common
• Response = new specialized graph engines

Question: Is graph processing that different from other types of data processing?

Our Answer: No. Can be subsumed by “traditional” relational processing
What is appealing about these new engines?

- Vertex-centric API
- Easy to write graph programs
- Higher programmer productivity
Graph API: Giraph

Vertex Centric:

do {

  foreach vertex in the graph {
    receive_messages();
    mutate_vertex_value();
    if (send_to_neighbors()) {
      send_messages_to_neighbors();
    }
  }
}

} until (has_converged() || reached_limit())
Example: Shortest path

Input Graph

Computation & Communication Pattern

Iteration 1
Example: Shortest path

Input Graph

Computation & Communication Pattern

Iteration 2
Example: Shortest path

Input Graph

Iteration 3

Computation & Communication Pattern
1. **Gather** values (from neighbors)
2. **Apply** updates to local state
3. **Scatter** signals to your neighbors
What is appealing about these new engines?

- Vertex-centric API
- Easy to write graph programs
- Higher programmer productivity
But ...

• Can we build a similar vertex-centric simple API?
• ... and then map it to SQL, with good performance

Advantages:
• Already have SQL in the enterprise stack
• Huge advantage to “one size fits many”
  • $O(N^2)$ headaches when maintaining $N$ specialized systems
  • Economies of scale

The GRAIL API
Example: Shortest path

Input Graph

Vertex

<table>
<thead>
<tr>
<th>id</th>
<th>val</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>∞</td>
</tr>
<tr>
<td>B</td>
<td>∞</td>
</tr>
<tr>
<td>C</td>
<td>∞</td>
</tr>
<tr>
<td>D</td>
<td>∞</td>
</tr>
</tbody>
</table>

Edge

<table>
<thead>
<tr>
<th>src</th>
<th>dest</th>
<th>val</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>B</td>
<td>1</td>
</tr>
<tr>
<td>A</td>
<td>C</td>
<td>2</td>
</tr>
<tr>
<td>B</td>
<td>D</td>
<td>2</td>
</tr>
<tr>
<td>C</td>
<td>D</td>
<td>3</td>
</tr>
</tbody>
</table>

Iteration 1

<table>
<thead>
<tr>
<th>id</th>
<th>val</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0</td>
</tr>
<tr>
<td>B</td>
<td>1</td>
</tr>
<tr>
<td>C</td>
<td>2</td>
</tr>
<tr>
<td>D</td>
<td>∞</td>
</tr>
</tbody>
</table>

Iteration 2

<table>
<thead>
<tr>
<th>id</th>
<th>val</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0</td>
</tr>
<tr>
<td>B</td>
<td>1</td>
</tr>
<tr>
<td>C</td>
<td>2</td>
</tr>
<tr>
<td>D</td>
<td>3</td>
</tr>
</tbody>
</table>

Iteration 3

<table>
<thead>
<tr>
<th>id</th>
<th>val</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0</td>
</tr>
<tr>
<td>B</td>
<td>1</td>
</tr>
<tr>
<td>C</td>
<td>2</td>
</tr>
<tr>
<td>D</td>
<td>3</td>
</tr>
</tbody>
</table>
DECLARE @flag int;
SET @flag = 1;

SELECT vertex.id, 2147483647 AS val INTO next
FROM vertex;

CREATE TABLE message(id int, val int);
INSERT INTO message values(1,0);

WHILE (@flag != 0)
BEGIN

SELECT message.id AS id, MIN(message.val) AS val INTO cur
FROM message
GROUP BY message.id;

DROP TABLE message;
SELECT cur.id AS id, cur.val AS val INTO update
FROM cur, next
WHERE cur.id = next.id AND cur.val < next.val;

UPDATE next
SET next.val = update.val
FROM update, next
WHERE next.id = update.id;

SELECT edge.dest AS id, update.val + 1 AS val INTO message
FROM update, edge
WHERE edge.src = update.id;

DROP TABLE cur;
DROP TABLE update;

SELECT @flag = COUNT(*) FROM message;
END

The Grail API
1. VertexValType: INT
2. MessageValType: INT
3. InitiateVal: INT_MAX
4. InitialMessage: (1, 0)
5. CombineMessage: MIN(message)
6. UpdateAndSend: update = cur.val < getVal()
   if (update) {
      setVal(cur.val)
      send(out, cur.val+1)
   }
7. End: NO_MESSAGE
DECLARE @flag int;
SET @flag = 1;

SELECT vertex.id, 2147483647 AS val INTO next FROM vertex;

CREATE TABLE message(id int, val int);
INSERT INTO message values(1,0);

WHILE (@flag != 0)
BEGIN
    SELECT message.id AS id, MIN(message.val) AS val INTO cur FROM message GROUP BY message.id;
    DROP TABLE message;
    SELECT cur.id AS id, cur.val AS val INTO update FROM cur, next WHERE cur.id = next.id AND cur.val < next.val;
    UPDATE next SET next.val = update.val FROM update, next WHERE next.id = update.id;
    SELECT edge.dest AS id, update.val + 1 AS val INTO message FROM update, edge WHERE edge.src = update.id;
    DROP TABLE cur;
    DROP TABLE update;
    SELECT @flag = COUNT(*) FROM message;
END
<table>
<thead>
<tr>
<th>Vertex Centric</th>
<th>Relational Algebra</th>
</tr>
</thead>
<tbody>
<tr>
<td>Receive messages</td>
<td>$\text{cur} \leftarrow \gamma_{id,F_0(val)}(\text{message})$</td>
</tr>
<tr>
<td>Mutate value</td>
<td>$\text{next} \leftarrow u \pi_{\text{next.id,F}<em>1(\text{other.val})} \text{other} \bowtie</em>{id} \text{next}$</td>
</tr>
<tr>
<td>Send messages</td>
<td>$\pi_{\text{edge.B,F}<em>2(\text{other.val,edge.val})} \text{other} \bowtie</em>{\text{other.id}=\text{edge.A}} \text{edge}$</td>
</tr>
</tbody>
</table>

**Aggregate function** (can be a UDAF)

**Scalar computation** (can be a UDF)

**Scalar computation** (can be a UDF)

**Join attributes control the direction**

For single source shortest path:

- **min**
- **sum**
- **identity**
- **Outgoing edges**
Grail: Implementation and Evaluation

**Test Machine (single node)**
- Dual 1.8GHz Xeon E2450L
- 96GB of main memory

**Compare with**
- Giraph (v.1.1.0)
- GraphLab (v 2.2): sync and async

**Dataset**

<table>
<thead>
<tr>
<th>Dataset</th>
<th>#nodes</th>
<th>#edges</th>
<th>size</th>
</tr>
</thead>
<tbody>
<tr>
<td>web-google (GO)</td>
<td>9K</td>
<td>5M</td>
<td>71MB</td>
</tr>
<tr>
<td>com-Orkut (OR)</td>
<td>3M</td>
<td>117M</td>
<td>1.6GB</td>
</tr>
<tr>
<td>Twitter-10 (TW)</td>
<td>41.6M</td>
<td>1.5B</td>
<td>24GB</td>
</tr>
<tr>
<td>uk-2007-05 (UK)</td>
<td>100M</td>
<td>3.3B</td>
<td>56GB</td>
</tr>
</tbody>
</table>

**Queries**
- Single source shortest-path
- Page Rank
- Weakly connect components
Grail is slower than GraphLab for the smallest datasets, ... but catches up as the dataset size grows, ... and can handle the largest datasets, while the other systems fail.
Summary: Graph Analytics on RDBMS

- Simple API (Grail) addresses the programmer productivity issue.
- Produces far more robust and deployable solutions than specialized graph engines.
- Interesting physical schema design and optimization issues.
The general case against GraphDB Inc.

Figure 1: Fraction of jobs per application type.

Figure 2: Fraction of users per application type.
Thanks!

David DeWitt  Jae Young Do  Alan Halverson  Ian Rae