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

"One Size Fits All": An Idea Whose Time Has Come and Gone

Michael Stonebraker
Computer Science and Artificial Intelligence Laboratory, M.I.T., and StreamBase Systems, Inc.
stonebraker@csail.mit.edu

Uğur Çetintemel
Department of Computer Science
Brown University, and StreamBase Systems, Inc.
ugur@cs.brown.edu

The house always wins
Motivation

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

\[
\begin{align*}
&\text{do } \{ \\
&\quad \text{foreach vertex in the graph } \{ \\
&\quad\quad \text{receive\_messages()}; \\
&\quad\quad \text{mutate\_vertex\_value()}; \\
&\quad\quad \text{if (send\_to\_neighbors()) } \{ \\
&\quad\quad\quad \text{send\_messages\_to\_neighbors()}; \\
&\quad\quad \} \\
&\quad \}\} \\
&\quad \text{until (has\_converged() || reached\_limit())}
\end{align*}
\]
Example: Shortest path

Input Graph

Computation & Communication Pattern

Iteration 1
Example: Shortest path

Input Graph

Computation & Communication Pattern

Iteration 2

(A, 1)
(B, 2)
(C, 2)
(D, ∞)

(V_A, 0)
(V_B, 1)
(V_C, 2)
(V_D, 3)

(D, 3)
(D, 5)
Example: Shortest path

Input Graph

Computation & Communication Pattern

Iteration 3
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
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

Vertex

<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>∞</td>
</tr>
<tr>
<td>C</td>
<td>∞</td>
</tr>
<tr>
<td>D</td>
<td>∞</td>
</tr>
</tbody>
</table>

Next

<table>
<thead>
<tr>
<th>id</th>
<th>val</th>
</tr>
</thead>
<tbody>
<tr>
<td>D</td>
<td>3</td>
</tr>
</tbody>
</table>

Message

<table>
<thead>
<tr>
<th>id</th>
<th>val</th>
</tr>
</thead>
<tbody>
<tr>
<td>B</td>
<td>1</td>
</tr>
<tr>
<td>C</td>
<td>2</td>
</tr>
</tbody>
</table>

Iteration 2

Vertex

<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>

Next

<table>
<thead>
<tr>
<th>id</th>
<th>val</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>2</td>
</tr>
</tbody>
</table>

Message

<table>
<thead>
<tr>
<th>id</th>
<th>val</th>
</tr>
</thead>
<tbody>
<tr>
<td>D</td>
<td>3</td>
</tr>
<tr>
<td>D</td>
<td>5</td>
</tr>
</tbody>
</table>

Iteration 3

Vertex

<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>

Next

<table>
<thead>
<tr>
<th>id</th>
<th>val</th>
</tr>
</thead>
<tbody>
<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>

Message

<table>
<thead>
<tr>
<th>id</th>
<th>val</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></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;
END

SELECT @flag = COUNT(*) FROM message;

The Grail API

Initialize

Initialize the message table

Aggregate the messages

Update and generate messages for the next iteration

Stop when no new msgs.

VertexValType: INT
MessageValType: INT
InitiateVal : INT_MAX
InitialMessage : (1, 0)
CombineMessage: MIN(message)
UpdateAndSend: update=cur.val<getVal()
if (update) {
  setVal(cur.val)
send(out, cur.val+1)
}
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

Initialize
Initialize the message table
Aggregate the messages
Create an update table and only consider updated vertices
Update the next table
Generate the message table for the next iteration
Stop when there are no new messages
For single source shortest path

- **Receive messages**
  - \( \text{cur} \leftarrow \gamma_{id,F_0(val)} \) (message)

- **Mutate value**
  - \( \text{next} \leftarrow u \pi_{\text{next.id},F_1(\text{other.val})} \text{other} \bowtie_{\text{id}} \text{next} \)

- **Send messages**
  - \( \pi_{\text{edge.B},F_2(\text{other.val}, \text{edge.val})} \text{other} \bowtie_{\text{other.id}= \text{edge.A}} \text{edge} \)

**Aggregate function** (can be a UDAF)
- **Scalar computation** (can be a UDF)
  - **Scalar computation** (can be a UDF)
  - **Join attributes control the direction**

- **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