Graphs On Databases

Alekh Jindal

NEDB 2014

Supervisors

Collaborate

work

sabbatical

work

work

sabbatical

work

work
Relational DATA → Relational DATABASE
Streaming DATA -> Streaming DATABASE
Graph $\rightarrow$ Graph

DATA DATABASE
DATABASE

DATA
APPLICATIONS

DATABASE

Logical Data Independence

Physical Data Independence

DATA
Barriers to “Graphs on Databases”

- Graphs in relational model
- Graph operations in SQL
- Expressing iterative graph queries
- Efficient graph analytics performance
- Ease-of-use
Graphs in Relational Model

Nodes

<table>
<thead>
<tr>
<th>id</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
</tr>
</tbody>
</table>

Edges

<table>
<thead>
<tr>
<th>fromId</th>
<th>toId</th>
<th>weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
Graphs Operations in SQL

- **Node access**
  
  ```sql
  Select * From Nodes Where Id=ID
  ```

- **Neighborhood access**
  
  ```sql
  Select * From Edges Where fromId=ID
  ```

- **Parallel neighborhood access**
  
  ```sql
  Select * From Edges Group By fromId
  ```

- **1-hop neighbors**
  
  ```sql
  Select * From Edges e1,Edges e2 Where e1.tId=e2.fId
  ```
Example: Shortest Paths

```sql
UPDATE Nodes AS node SET value = new_node.value
FROM (SELECT e.told AS Id, min(n1.value+1) AS value
      FROM Nodes AS n1, Edges AS e, Nodes AS n2
      WHERE n1.Id = e.fromId AND n2.Id = e.told
      GROUP BY e.told, n2.value
      HAVING min(n1.value+1) < n2.value
    ) AS new_node
WHERE node.Id = new_node.Id;
```
Example: Shortest Paths

```
UPDATE Nodes AS node SET value = new_node.value
FROM (SELECT e.told AS Id, min(n1.value+1) AS value
     FROM Nodes AS n1, Edges AS e, Nodes AS n2
     WHERE n1.Id = e.fromId AND n2.Id = e.told
     GROUP BY e.told, n2.value
     HAVING min(n1.value+1) < n2.value
     ) AS new_node
WHERE node.Id = new_node.Id;
```
Iterative Graph Queries

- Driver program: UDF / Stored Procedure
- Three Things:
  - initialization
  - actual graph query (in a loop)
  - termination condition
Example: Shortest Paths

Initialization:
1. Set the value of start node to 0
2. Set the value of all other node to inf

Loop:
The shortest paths SQL

Termination Condition:
No more nodes to Update

```
UPDATE Nodes AS node SET value=new_node.value
FROM(
    SELECT e.told AS Id, min(n1.value+1) AS value
    FROM Nodes AS n1, Edges AS e, Nodes AS n2
    WHERE n1.Id=e.fromId AND n2.Id=e.told
    GROUP BY e.told, n2.value
    HAVING min(n1.value+1) < n2.value
) AS new_node
WHERE node.Id = new_node.Id;
```
Efficient Graph Analytics

- Three SQL Databases:
  - row store
  - column store
  - main-memory store

- Two Graph Databases:
  - transactional graph database
  - graph analytics system

- Two queries: PageRank, Shortest Paths

- Social network dataset from snap.stanford.edu/data
PageRank

Time (seconds)

Graph Database
Main-memory Database
Row Store Database
Apache Giraph
Column Store Database

Twitter
GPlus
LiveJournal
Shortest Paths

Time (seconds)

- Twitter
- GPlus
- LiveJournal

Graph Database
Main-memory Database
Row Store Database
Apache Giraph
Column Store Database
Ease-of-Use

**SQL**

```
UPDATE Nodes AS node SET value=new_node.value
FROM(
    SELECT e.told AS Id, min(n1.value+1) AS value
    FROM Nodes AS n1, Edges AS e, Nodes AS n2
    WHERE n1.Id=e.fromId AND n2.Id=e.told
    GROUP BY e.told, n2.value
    HAVING min(n1.value+1) < n2.value
) AS new_node
WHERE node.Id = new_node.Id;
```

**Pregel**

```
void compute(vector<float> messages){
    // get the minimum distance
    float mindist = id==START_NODE ? 0 : DBL_MAX;
    for(vector<float>::iterator it = messages.begin();
        it != messages.end(); ++it)
        mindist = min(mindist,*it);
    // send messages to all edges if new minimum is found
    float vvalue = getVertexValue();
    if(mindist < vvalue){
        modifyVertexValue(mindist);
        vector<int> edges = getOutEdges();
        for(vector<int>::iterator it = edges.begin();
            it != edges.end(); ++it)
            sendMessage(*it, mindist+1);
    }
    // halt
    voteToHalt();
}
```
Ease-of-Use

SQL

<table>
<thead>
<tr>
<th>id</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>fromId</th>
<th>told</th>
<th>weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Pregel

Nodes

Edges

1 2
2 4
2 5
3 2
5 1

5
1
2
3
4
Vertex-centric Query Interface

**APPLICATION**

Logical Data Independence

**DATABASE**

Logical Data Independence

**DATA**

Physical Data Independence

**Vertex Programs**

Pregel-style API:
- getMessages()
- getEdges()
- sendMessage()
- voteToHalt(), etc.

**Vertex UDF**

Invokes the vertex program if:
- the vertex is active, or
- the vertex has incoming messages

**Coordinator**

Synchronizes supersteps
Redistributes Messages

**Vertex (V), Edge (E), Message (M)**
Vertex-centric Query Interface

**APPLICATION**

- Logical Data Independence
- Batching
- No in-place Updates

**DATABASE**

- Physical Data Independence
- Union
  - Vertex (V), Edge (E), Message (M)

**COORDINATOR**

- Synchronizes supersteps
- Redistributes Messages

**VERTEX UDF**

- Invokes the vertex program if:
  - the vertex is active, or
  - the vertex has incoming messages

**VERTEX PROGRAMS**

- Pregel-style API:
  - getMessages()
  - getEdges()
  - sendMessage()
  - voteToHalt(), etc.
PageRank (Vertex)

- Twitter: 421.5 seconds
- GPlus: 2,071.0 seconds
- LiveJournal: 218.1 seconds

Bar chart showing the time (in seconds) for different datasets and platforms:

- Main-memory Database: Twitter 421.5, GPlus 2,071.0, LiveJournal 218.1
- Apache Giraph: Twitter 47.0, GPlus 53.5, LiveJournal 218.1
- Column Store Database: Twitter 10.9, GPlus 47.7, LiveJournal 335.5
Shortest Paths (Vertex)

<table>
<thead>
<tr>
<th>Platform</th>
<th>Twitter</th>
<th>GPlus</th>
<th>LiveJournal</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time (seconds)</td>
<td>121.0</td>
<td>50.8</td>
<td>7,950.1</td>
</tr>
<tr>
<td>Main-memory DB</td>
<td>43.7</td>
<td>23.8</td>
<td>115.5</td>
</tr>
<tr>
<td>Apache Giraph</td>
<td>23.8</td>
<td></td>
<td>146.3</td>
</tr>
<tr>
<td>Column Store</td>
<td>10.6</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Main-memory Database
- Apache Giraph
- Column Store Database
Vertex-centric interface allows...

- Connected Components
- Random Walks with Restart
- Stochastic Gradient Descent
- Or, other message Passing Algorithms

... right within the database system!
Advantages of “Graphs on Databases”

- Running arbitrary SQL queries
- Pre- and post- processing of data
- Updates are trivial
- ACID for free
- Don’t need to deal with Yet-Another-System!
Summary

- Graph analytics can be mapped to relational queries (plus UDFs)
- SQL systems can offer very good performance over relational queries
- We can extend SQL systems to provide more graph-natural query interfaces