GraphFrames: Graph Queries in Apache Spark SQL

Ankur Dave
UC Berkeley AMPLab

Joint work with Alekh Jindal (Microsoft), Li Erran Li (Uber), Reynold Xin (Databricks), Joseph Gonzalez (UC Berkeley), and Matei Zaharia (MIT and Databricks)
GraphFrames (2016)

2009 Spark

2013 Apache Spark + GraphX

2016 Apache Spark + GraphFrames

Relational Queries

+ Graph Algorithms

+ Graph Queries
Graph Algorithms vs. Graph Queries

Graph Algorithms

PageRank

Alternating Least Squares

Graph Queries
Graph Algorithms vs. Graph Queries

Graph Algorithm: PageRank

Graph Query: Wikipedia Collaborators

<table>
<thead>
<tr>
<th>Editor 1</th>
<th>Editor 2</th>
<th>Article 1</th>
<th>Article 2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Graph Algorithms vs. Graph Queries

Graph Algorithm: PageRank

// Iterate until convergence
wikipedia.pregel(
    sendMsg = { e =>
        e.sendToDst(e.srcRank * e.weight)
    },
    mergeMsg = _ + _,
    vprog = {
        (id, oldRank, msgSum) =>
        0.15 + 0.85 * msgSum
    }
)

Graph Query: Wikipedia Collaborators

wikipedia.find(
    "(u1)-[e11]->(article1);
    (u2)-[e21]->(article1);
    (u1)-[e12]->(article2);
    (u2)-[e22]->(article2)"
).select("*",
    "e11.date - e21.date".as("d1"),
    "e12.date - e22.date".as("d2"))
    .sort("d1 + d2".desc).take(10)
Separate Systems

Graph Algorithms

\[ \text{GraphX} \]

\[ \text{APACHE GIRAPH} \]

\[ \text{GraphLab} \]

Graph Queries

\[ \text{neo4j} \]

\[ \text{TITAN} \]

\[ \text{OrientDB} \]
Problem: Mixed Graph Analysis

Raw Wikipedia

Text Table

<table>
<thead>
<tr>
<th>Article</th>
<th>Text</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Hyperlinks

Edit Graph

PageRank

Frequent Collaborators

Vandalism Suspects

User | Article
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Edit Table

User | Article
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

XML

< / >
Solution: GraphFrames

Graph Algorithms ➔ GraphFrames API ➔ Spark SQL ➔ Pattern Query Optimizer ➔ Graph Queries
GraphFrames API

• Unifies graph algorithms, graph queries, and DataFrames
• Available in Scala, Java, and Python

```scala
class GraphFrame {
  def vertices: DataFrame
  def edges: DataFrame

  def find(pattern: String): DataFrame
  def registerView(pattern: String, df: DataFrame): Unit

  def degrees(): DataFrame
  def pageRank(): GraphFrame
  def connectedComponents(): GraphFrame

  ...
}
```
Implementation

Query String → Parsed Pattern → Logical Plan

Graph–Relational Translation
View Selection

Materialized Views

Logical Plan → Optimized Logical Plan

Join Elimination and Reordering

Spark SQL → DataFrame Result

Graph Algorithms → GraphX
Graph–Relational Translation

Existing Logical Plan
Output: A, B, C

Vertex Table

Edge Table
Materialized View Selection

GraphX: Triplet view enabled efficient message-passing algorithms
Materialized View Selection

**GraphFrames**: User-defined views enable efficient graph queries
Join Elimination

Unnecessary join

\[
\text{SELECT src, dst FROM edges INNER JOIN vertices ON src = id;}
\]

can be eliminated if tables satisfy referential integrity, simplifying graph-relational translation:

\[
\text{SELECT src, dst FROM edges;}
\]
Join Reordering

Example Query

Left-Deep Plan

Bushy Plan

User-Defined View
Evaluation

Faster than Neo4j for *unanchored* pattern queries

Triangle query on 1M edge subgraph of web-Google. Each system configured to use a single core.
Evaluation

Approaches performance of GraphX for graph algorithms using Spark SQL whole-stage code generation

PageRank Performance

Per-iteration runtime, s

GraphFrames    GraphX   Naïve Spark

Per-iteration performance on web-Google, single 8-core machine. Naïve Spark uses Scala RDD API.
Evaluation

Registering the right views can greatly improve performance for some queries

<table>
<thead>
<tr>
<th>View</th>
<th>Query</th>
<th>Size in Google graph</th>
</tr>
</thead>
<tbody>
<tr>
<td>2-cycle</td>
<td>(a) -&gt; (b) -&gt; (a)</td>
<td>1,565,976</td>
</tr>
<tr>
<td>V</td>
<td>(c) &lt;-&gt; (a) -&gt; (b)</td>
<td>67,833,471</td>
</tr>
<tr>
<td>Triangle</td>
<td>(a) &lt;-&gt; (b) -&gt; (c) -&gt; (a)</td>
<td>28,198,954</td>
</tr>
<tr>
<td>3-cycle</td>
<td>(a) -&gt; (b) -&gt; (c) -&gt; (a)</td>
<td>11,669,313</td>
</tr>
</tbody>
</table>
Future Work

• Suggest views automatically
• Exploit attribute-based partitioning in optimizer
• Code generation for single node
Try It Out!

Released as a Spark Package at:

https://github.com/graphframes/graphframes

Thanks to Joseph Bradley, Xiangrui Meng, and Timothy Hunter.

ankurd@eecs.berkeley.edu