From “Think Like a Vertex” to “Think Like a Graph”

Yuanyuan Tian, Andrey Balmin, Severin Andreas Corsten, Shirish Tatikonda, John McPherson
Large Scale Graph Processing

- Graph data is everywhere and growing rapidly!
  - 126 million blogs
  - 50 billion Web pages
  - 90 trillion emails
- Analyzing graph data is increasingly important.
  - Retail: customer segmentation and recommendation
  - Consumer Insights: key influencer analysis
  - Telco: churn prediction
  - Biology & Health Care: disease propagation analysis
  - Government: Terrorists detection
- What is the right programming model for large-scale graph processing?
Existing Graph Processing Systems

- Divide input graphs into partitions

- Employ **vertex-centric** programming model
  - Programmers “think like a vertex”
    - Operate on a vertex and its edges
    - Communication to other vertices
      - Message passing (e.g. Pregel/Giraph)
      - Scheduling of updates (e.g. GraphLab)
“Think like a vertex” → “Think like a graph”

- **Partition**: A collection of vertices
- **Computation**: A vertex and its edges
- **Communication**: 1-hop at a time
  e.g. A → B → D

→

- **“Think like a graph”**
- **Partition**: A proper subgraph
- **Computation**: A subgraph
- **Communication**: Multiple-hops at a time
  e.g. A → D

![Graph Diagram]

<table>
<thead>
<tr>
<th>Partition</th>
<th>Vertex</th>
<th>Edge List</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>A</td>
<td>B</td>
</tr>
<tr>
<td></td>
<td>B</td>
<td>D</td>
</tr>
<tr>
<td>P2</td>
<td>C</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td>D</td>
<td>E</td>
</tr>
<tr>
<td>P3</td>
<td>E</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td>F</td>
<td>A</td>
</tr>
</tbody>
</table>

![Subgraph Diagram]

- **Subgraph G1**: A → B → D
- **Subgraph G2**: C → A → D
- **Subgraph G3**: E → A → D

- **Subgraph G4**: E → D
Graph-Centric Programming Model

- Expose subgraphs to programmers
  - Internal vertices vs boundary vertices
    - Information exchange between internal vertices of a partition is immediate
    - Messages are only sent from boundary vertices of a partition to internal vertices of a different partition
Advantages of graph-centric model

- Any algorithm expressed in the vertex-centric model can be expressed in the graph-centric model
- Allow lower-level access for algorithm-specific optimizations
  - Use of existing off-the-shell graph algorithms on subgraphs
  - Local asynchronous computation
  - Natural translation of existing graph-centric parallel algorithms
- Provide sufficiently high-level abstraction for easy of use
Example: Weakly Connected Component (WCC)

compute() on each subgraph

- If 0th superstep
  - Sequential WCC on subgraph
  - Send labels of boundary vertices to their corresponding internal vertices

- Else
  - Use the received messages to update the labels of vertices in the subgraph
  - Merge connected components
  - For boundary vertices with label change, send labels to their corresponding internal vertices
Hybrid Execution Model

- Can we keep the simple vertex-centric programming model but still improve performance?
  - Key: Differentiate internal messages and external messages
  - What messages can be used in local computation?
    - Vertex-centric model
      - Only messages (external and internal) from previous superstep
    - Hybrid model
      - External messages from previous superstep
      - Internal messages from previous + current superstep (local asynchronous computation)
  - Only apply to a limited set of graph algorithms
Giraph++: A New Graph Processing System

- Built on top of Apache Giraph
- Support vertex-centric model (VM), graph-centric model (GM) & hybrid model (HM) in the same Giraph++ system
- Contributed to Apache Giraph Project
  - Planned in future Giraph release
A Peek of Performance Results

- 10-node cluster
  - Per node: 32GB RAM, 8 cores, 7 workers
  - Network: 1Gbit

- Dataset: 4 web graphs
  - # nodes: 19million ~ 428million
  - # edges: 298million ~ 1.0billion

- Significant improvement in execution time and network messages, especially with better graph partitioning strategy
  - Up to $62\times$ speedup in execution time!
  - Up to $200\times$ reduction in network messages!
Conclusion

“Think like a vertex” → “Think like a graph”

- Take advantage of local graph structure in a partition
- Enable complex and flexible graph algorithms
- Can be exploited to various graph applications
- Bring significant performance improvement, especially with better graph partitioning strategy
- A valuable complement to existing vertex-centric model