Giraph: Production-grade graph processing infrastructure for trillion edge graphs
Motivation

Recommended Pages
Get updates from your favorite businesses and brands.

Search for Pages...

Oakland International Airport
Airport Terminal
Michael Kong and 8 other friends were here.

Facebook Analog Research Laboratory
Organization
Carol Pai was here.

Sawa Sushi
Sushi Restaurant
Agata Zielińska and 3 other friends were here.

Fuki Sushi at Facebook
Sushi Restaurant
Alex Feinberg and 10 other friends were here.
Apache Giraph

- Inspired by Google’s Pregel but runs on Hadoop
- “Think like a vertex”
- Maximum value vertex example
Giraph on Hadoop / Yarn

- Giraph
- MapReduce
- YARN

- Hadoop 0.20.X
- Hadoop 0.20.203
- Hadoop 1.X
- Hadoop 2.0.X
public class PageRankComputation extends BasicComputation<LongWritable, DoubleWritable, FloatWritable, DoubleWritable> {
    public void compute(Vertex<LongWritable, DoubleWritable, FloatWritable> vertex, Iterable<DoubleWritable> messages) {
        if (getSuperstep() >= 1) {
            double sum = 0;
            for (DoubleWritable message : messages) {
                sum += message.get();
            }
            vertex.getValue().set(DoubleWritable((0.15d / getTotalNumVertices()) + 0.85d * sum));
        }
        if (getSuperstep() < 30) {
            sendMsgToAllEdges(new DoubleWritable(getVertexValue().get() / getNumOutEdges()));
        } else {
            voteToHalt();
        }
    }
}
Giraph compared to MPI

Giraph’s vertex centric API is higher-level (and narrower) than MPI

- Vertex message queues vs process-level messaging
- Enforced BSP model
- Data distribution, checkpointing handled by Giraph
- Giraph aggregators are user-level MPI_Allreduce reductions

Java (Giraph) vs C/C++ (MPI)

Scheduled with Hadoop clusters

Easy integration with other Hadoop pipelines
Apache Giraph data flow

**Loading the graph**
- Input format
  - Split 0
  - Split 1
  - Split 2
  - Split 3
- Worker 0: Load/Send Graph
- Worker 1: Load/Send Graph

**Compute / Iterate**
- In-memory graph
  - Worker 0: Part 0
  - Worker 1: Part 1
  - Part 2
  - Part 3
- Compute/Send Messages
- Compute/Send Messages

**Storing the graph**
- Worker 0: Part 0
  - Part 1
- Worker 1: Part 2
  - Part 3
- Output format
  - Part 0
  - Part 1
  - Part 2
  - Part 3
Pipelined computation

Master “computes”
- Sets computation, in/out message, combiner for next super step
- Can set/modify aggregator values
Use case
Affinity propagation

Frey and Dueck “Clustering by passing messages between data points”
Science 2007
Organically discover exemplars based on similarity
3 stages

Responsibility $r(i,k)$
- How well suited is $k$ to be an exemplar for $i$?

Availability $a(i,k)$
- How appropriate for point $i$ to choose point $k$ as an exemplar given all of $k$’s responsibilities?

Update exemplars
- Based on known responsibilities/availabilities, which vertex should be my exemplar?

* Dampen responsibility, availability

\[ r \leftarrow \alpha \cdot r_{old} + (1 - \alpha) \cdot r_{new} \]
\[ a \leftarrow \alpha \cdot a_{old} + (1 - \alpha) \cdot a_{new} \]
Responsibility

\[ r(i, k) \leftarrow s(i, k) - \max_{k' \neq k} \left\{ a(i, k') + s(i, k') \right\} \]

Every vertex \( i \) with an edge to \( k \) maintains responsibility of \( k \) for \( i \)
Sends responsibility to \( k \) in ResponsibilityMessage (senderid, responsibility(i,k))
Availability

$$a(i, k) \leftarrow \min \left( 0, r(k, k) + \sum_{i' \notin \{i, k\}} \max(0, r(i', k)) \right)$$

$$a(k, k) \leftarrow \sum_{i' \neq k} \max(0, r(i', k))$$

Vertex sums positive messages
Sends availability to i in AvailabilityMessage (senderid, availability(i,k))
Update exemplars

\[ \text{exemplar}(i) \leftarrow \max(a(i, k) + r(i, k)) \]

Dampens availabilities and scans edges to find exemplar \( k \)
Updates self-exemplar
Master logic

if (exemplars agree they are exemplars && changed exemplars < Δ) then halt, otherwise continue
Performance & Scalability
Example graph sizes

Twitter 255M MAU (https://about.twitter.com/company), 208 average followers (Beevolve 2012)
→ Estimated >53B edges

Facebook 1.28B MAU (Q1/2014 report), 200+ average friends (2011 S1)
→ Estimated >256B edges
## Faster than Hive?

<table>
<thead>
<tr>
<th>Application</th>
<th>Graph Size</th>
<th>CPU Time Speedup</th>
<th>Elapsed Time Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>Page rank (single iteration)</td>
<td>400B+ edges</td>
<td><strong>26x</strong></td>
<td><strong>120x</strong></td>
</tr>
<tr>
<td>Friends of friends score</td>
<td>71B+ edges</td>
<td><strong>12.5x</strong></td>
<td><strong>48x</strong></td>
</tr>
</tbody>
</table>
Apache Giraph scalability

Scalability of workers (200B edges)

Seconds

# of Workers

50 100 150 200 250 300

0 125 250 375 500

Giraph

Ideal

Scalability of edges (50 workers)

Seconds

# of Edges

1E+09 7E+10 1E+11 2E+11

Giraph

Ideal
Trillion social edges page rank

Minutes per iteration

<table>
<thead>
<tr>
<th>Date</th>
<th>Minutes</th>
</tr>
</thead>
<tbody>
<tr>
<td>6/30/2013</td>
<td>4</td>
</tr>
<tr>
<td>6/2/2014</td>
<td>3</td>
</tr>
</tbody>
</table>

Improvements
- GIRAPH-840 - Netty 4 upgrade
- G1 Collector / tuning
Graph partitioning
Why balanced partitioning

Random partitioning == good balance
BUT ignores entity affinity
Balanced partitioning application

Results from one service:
Cache hit rate grew from 70% to 85%, bandwidth cut in 1/2
Balanced label propagation results

* Loosely based on Ugander and Backstrom. Balanced label propagation for partitioning massive graphs, WSDM '13
Leveraging partitioning

Explicit remapping
Native remapping
  • Transparent
  • Embedded
Explicit remapping

**Original graph**

<table>
<thead>
<tr>
<th>Id</th>
<th>Edges</th>
</tr>
</thead>
<tbody>
<tr>
<td>San Jose</td>
<td>(Chicago, 4)</td>
</tr>
<tr>
<td></td>
<td>(New York, 6)</td>
</tr>
<tr>
<td>Chicago</td>
<td>(San Jose, 4)</td>
</tr>
<tr>
<td></td>
<td>(New York, 3)</td>
</tr>
<tr>
<td>New York</td>
<td>(San Jose, 6)</td>
</tr>
<tr>
<td></td>
<td>(Chicago, 3)</td>
</tr>
</tbody>
</table>

**Partitioning Mapping**

<table>
<thead>
<tr>
<th>Id</th>
<th>Alt Id</th>
</tr>
</thead>
<tbody>
<tr>
<td>San Jose</td>
<td>0</td>
</tr>
<tr>
<td>Chicago</td>
<td>1</td>
</tr>
<tr>
<td>New York</td>
<td>2</td>
</tr>
</tbody>
</table>

**Remapped graph**

<table>
<thead>
<tr>
<th>Id</th>
<th>Edges</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>(1, 4)</td>
</tr>
<tr>
<td></td>
<td>(2, 6)</td>
</tr>
<tr>
<td>1</td>
<td>(0, 4)</td>
</tr>
<tr>
<td></td>
<td>(2, 3)</td>
</tr>
<tr>
<td>2</td>
<td>(0, 6)</td>
</tr>
<tr>
<td></td>
<td>(1, 3)</td>
</tr>
</tbody>
</table>

**Compute output**

<table>
<thead>
<tr>
<th>Id</th>
<th>Distance</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>6</td>
</tr>
</tbody>
</table>

**Reverse partition mapping**

<table>
<thead>
<tr>
<th>Alt Id</th>
<th>Id</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>San Jose</td>
</tr>
<tr>
<td>1</td>
<td>Chicago</td>
</tr>
<tr>
<td>2</td>
<td>New York</td>
</tr>
</tbody>
</table>

**Final compute output**

<table>
<thead>
<tr>
<th>Id</th>
<th>Distance</th>
</tr>
</thead>
<tbody>
<tr>
<td>San Jose</td>
<td>0</td>
</tr>
<tr>
<td>Chicago</td>
<td>4</td>
</tr>
<tr>
<td>New York</td>
<td>6</td>
</tr>
</tbody>
</table>
Native transparent remapping

Original graph

<table>
<thead>
<tr>
<th>Id</th>
<th>Edges</th>
</tr>
</thead>
<tbody>
<tr>
<td>San Jose</td>
<td>(Chicago, 4)</td>
</tr>
<tr>
<td></td>
<td>(New York, 6)</td>
</tr>
<tr>
<td>Chicago</td>
<td>(San Jose, 4)</td>
</tr>
<tr>
<td></td>
<td>(New York, 3)</td>
</tr>
<tr>
<td>New York</td>
<td>(San Jose, 6)</td>
</tr>
<tr>
<td></td>
<td>(Chicago, 3)</td>
</tr>
</tbody>
</table>

Partitioning Mapping

<table>
<thead>
<tr>
<th>Id</th>
<th>Group</th>
</tr>
</thead>
<tbody>
<tr>
<td>San Jose</td>
<td>0</td>
</tr>
<tr>
<td>Chicago</td>
<td>1</td>
</tr>
<tr>
<td>New York</td>
<td>2</td>
</tr>
</tbody>
</table>

Compute - shortest paths from “San Jose”

Final compute output

<table>
<thead>
<tr>
<th>Id</th>
<th>Distance</th>
</tr>
</thead>
<tbody>
<tr>
<td>San Jose</td>
<td>0</td>
</tr>
<tr>
<td>Chicago</td>
<td>4</td>
</tr>
<tr>
<td>New York</td>
<td>6</td>
</tr>
</tbody>
</table>

Original graph with group information

<table>
<thead>
<tr>
<th>Id</th>
<th>Group</th>
<th>Edges</th>
</tr>
</thead>
<tbody>
<tr>
<td>San Jose</td>
<td>0</td>
<td>(Chicago, 4)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(New York, 6)</td>
</tr>
<tr>
<td>Chicago</td>
<td>1</td>
<td>(San Jose, 4)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(New York, 3)</td>
</tr>
<tr>
<td>New York</td>
<td>2</td>
<td>(San Jose, 6)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(Chicago, 3)</td>
</tr>
</tbody>
</table>
Native embedded remapping

Original graph

<table>
<thead>
<tr>
<th>Id</th>
<th>Edges</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>(1, 4)</td>
</tr>
<tr>
<td></td>
<td>(2, 6)</td>
</tr>
<tr>
<td>1</td>
<td>(0, 4)</td>
</tr>
<tr>
<td></td>
<td>(2, 3)</td>
</tr>
<tr>
<td>2</td>
<td>(0, 6)</td>
</tr>
<tr>
<td></td>
<td>(1, 3)</td>
</tr>
</tbody>
</table>

Partitioning Mapping

<table>
<thead>
<tr>
<th>Id</th>
<th>Mach</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
</tr>
</tbody>
</table>

Compute - shortest paths from "San Jose" -

<table>
<thead>
<tr>
<th>Id</th>
<th>Distance</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>6</td>
</tr>
</tbody>
</table>

Final compute output

Original graph with mapping embedded in Id

<table>
<thead>
<tr>
<th>Top bits machine, Id</th>
<th>Edges</th>
</tr>
</thead>
<tbody>
<tr>
<td>0, 0</td>
<td>(1, 4)</td>
</tr>
<tr>
<td></td>
<td>(2, 6)</td>
</tr>
<tr>
<td>1, 1</td>
<td>(0, 4)</td>
</tr>
<tr>
<td></td>
<td>(2, 3)</td>
</tr>
<tr>
<td>0, 2</td>
<td>(0, 6)</td>
</tr>
<tr>
<td></td>
<td>(1, 3)</td>
</tr>
</tbody>
</table>

Not all graphs can leverage this technique, Facebook can since ids are longs with unused bits.
# Remapping comparison

<table>
<thead>
<tr>
<th>Pros</th>
<th>Explicit</th>
<th>Native Transparent</th>
<th>Native Embedded</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Can also add id compression</td>
<td>• No application change, just additional input parameters</td>
<td>• Utilize unused bits</td>
<td></td>
</tr>
<tr>
<td>Cons</td>
<td>• Application aware of remapping</td>
<td>• Additional memory usage on input</td>
<td>• Application changes Id type</td>
</tr>
<tr>
<td></td>
<td>• Workflow complexity</td>
<td>• Group information uses more memory</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Pre and post joins overhead</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Partitioning experiments

345B edge page rank

Seconds per iteration

<table>
<thead>
<tr>
<th></th>
<th>Random</th>
<th>47% Local</th>
<th>60% Local</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>160</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>120</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>80</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>40</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>
Message explosion
Avoiding out-of-core

Example: Mutual friends calculation between neighbors
1. Send your friends a list of your friends
2. Intersect with your friend list

1.23B (as of 1/2014)
200+ average friends (2011 S1)
8-byte ids (longs)
= 394 TB / 100 GB machines

3,940 machines (not including the graph)
WOAH..
Superstep splitting

Subsets of sources/destinations edges per superstep

* Currently manual - future work automatic!
Giraph in production

Over 1.5 years in production
Hundreds of **production** Giraph jobs processed a week
  • Lots of untracked experiments
30+ applications in our internal application repository
Sample production job - 700B+ edges
Job times range from minutes to hours
GiraphicJam demo
Giraph related projects

Welcome to Grafos.ml:
Tools for large-scale Machine Learning and Graph Analytics

- Okapi ML Library
- Real-Time Giraph
- User-friendly tools
- Join us!

Graft: The distributed Giraph debugger
Giraph roadmap
The future
Scheduling jobs

Snapshot automatically after a time period and restart at end of queue.
**Democratize Giraph?**

### Higher level primitives
- Filter
- Aggregating Neighbor Values (ANV)
- Local Update of Vertices (LUV)
- Update Vertices Using One Other Vertex (UVUOV)
- Updates vertex values by using a value from one other vertex (not necessarily a neighbor)
- Form Supervertices (FS)
- Aggregate Global Value (AGV)

### Graph traversal language
- (i.e. Gremlin)

```java
// calculate basic collaborative filtering for vertex 1
m = [:];
g.v(1).out('likes').in('likes').out('likes').groupCount(m)
m.sort{-it.value}
```

```java
// calculate the primary eigenvector (eigenvector centrality) of a graph
m = [:]; c = 0;
g.V.as('x').out.groupCount(m).loop('x'){c++ < 1000}
m.sort{-it.value}
```

### Implement lots of algorithms?
- ./run-page-rank
  - input pages
  - output page_rank_output
- ./run-mutual-friends
  - input friendlist
  - output pair_count_output
- ./run-graph-partitioning
  - input vertices_edges
  - output vertex_partition_list
Future work

Investigate alternative computing models
  • Giraph++ (IBM research)
  • Giraphx (University at Buffalo, SUNY)

Performance
Applications
Our team

Pavan Athivarapu
Avery Ching
Sergey Edunov
Maja Kabiljo
Sambavi Muthukrishnan