PowerGraph: Distributed Graph-Parallel Computation on Natural Graphs

Gonzales et al.

James Trever
What are Graphs?

Graphs are everywhere and used to encode relationships
So what are they used for?

- Targeted ads
- Natural Language Processing
- Identifying influential people and information
Natural Graphs
Graphs derived from real world phenomena
Challenges with Natural Graphs

Power-Law Degree Distribution
<table>
<thead>
<tr>
<th>#</th>
<th>Name (Screen Name)</th>
<th>Location</th>
<th>URL</th>
<th>Followers</th>
<th>Following</th>
<th>Updates</th>
<th>Joined</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>KATY PERRY @katyperry</td>
<td></td>
<td><a href="http://t.co/TUWZkUEN3o">http://t.co/TUWZkUEN3o</a></td>
<td>76,701,097</td>
<td>157</td>
<td>6,691</td>
<td>81 months ago</td>
</tr>
<tr>
<td>2.</td>
<td>Justin Bieber @justinbieber</td>
<td></td>
<td><a href="http://t.co/2B1YApLM35">http://t.co/2B1YApLM35</a></td>
<td>68,765,135</td>
<td>237,436</td>
<td>29,760</td>
<td>80 months ago</td>
</tr>
<tr>
<td>3.</td>
<td>Barack Obama @BarackObama</td>
<td>Washington, DC</td>
<td><a href="http://t.co/O5Woad92z1">http://t.co/O5Woad92z1</a></td>
<td>65,259,895</td>
<td>639,547</td>
<td>14,163</td>
<td>Details...</td>
</tr>
<tr>
<td>4.</td>
<td>Taylor Swift @taylorswift13</td>
<td></td>
<td><a href="http://t.co/AJTSTRgs35">http://t.co/AJTSTRgs35</a></td>
<td>64,195,315</td>
<td>240</td>
<td>3,976</td>
<td>84 months ago</td>
</tr>
<tr>
<td>5.</td>
<td>YouTube @YouTube</td>
<td>San Bruno, CA</td>
<td><a href="http://t.co/F3flcfn8Vf">http://t.co/F3flcfn8Vf</a></td>
<td>55,801,499</td>
<td>896</td>
<td>14,982</td>
<td>97 months ago</td>
</tr>
</tbody>
</table>
Graph-Parallel Abstraction

- A Vertex-Program, designed by the user, runs on every vertex
- Vertex-Programs interact with one another along their edges
- Multiple Vertex-Programs are run simultaneously
Challenges with Natural Graphs

- Power-Law Graphs are very difficult to partition/cut
- Often incurs a large communication or storage overhead
Existing Systems

Pregel
&
GraphLab
Pregel

- Bulk Synchronous Message Passing Abstraction
- Uses messages to communicate with other vertices
- Waits until all vertex programs have finished before starting the next “super step”
- Uses message combiners

```java
Message combiner(Message m1, Message m2) :
    return Message(m1.value() + m2.value());
void PregelPageRank(Message msg) :
    float total = msg.value();
    vertex.val = 0.15 + 0.85*total;
    foreach(nbr in out_neighbors) :
        SendMsg(nbr, vertex.val/num_out_nbrs);
```
Pregel

Fan-In

Fan-Out
GraphLab

- Asynchronous Distributed Shared-Memory Abstraction
- Vertex-Programs have shared access to distributed graph with data stored on each vertex and edge and can access the current vertex, adjacent edges and adjacent vertices irrespective of edge direction
- Vertex-Programs have the ability to schedule other vertices’ execution in the future

```c
void GraphLabPageRank(Scope scope) :
    float accum = 0;
    foreach (nbr in scope.in_nbrs) :
        accum += nbr.val / nbr.nout_nbrs();
    vertex.val = 0.15 + 0.85 * accum;
```
GraphLab

Machine 1

Machine 2

GraphLab Ghosting
Challenges with Natural Graphs

- Sequentially process edges
- Sends many messages (Pregel)
- Touches a large fraction of graph (GraphLab)
- Edge meta-data too large for single machine
- Asynchronous Execution requires heavy locking (GraphLab)
- Synchronous Execution prone to stragglers (Pregel)
PowerGraph
PowerGraph

- GAS Decomposition
  - Distribute Vertex-Programs
  - Parallelise high degree vertices

- Vertex Partitioning
  - Distribute power-law graphs more efficiently
GAS Decomposition

**Gather (Reduce)**
Accumulate information about neighborhood

- **User Defined:**
  - \( \text{Gather}(\cdot) \rightarrow \Sigma \)
  - \( \Sigma_1 + \Sigma_2 \rightarrow \Sigma_3 \)

\[ \Sigma \]

**Apply**
Apply the accumulated value to center vertex

- **User Defined:**
  - \( \text{Apply}(\mathbf{Y}, \Sigma) \rightarrow \mathbf{Y}' \)

\[ \mathbf{Y}' \]

**Scatter**
Update adjacent edges and vertices.

- **User Defined:**
  - \( \text{Scatter}(\cdot) \rightarrow \cdot \)

\[ \cdot \]

Update Edge Data & Activate Neighbors

---

**Algorithm 1: Vertex-Program Execution Semantics**

**Input:** Center vertex \( u \)

- if cached accumulator \( a_u \) is empty then
  - foreach neighbor \( v \) in gather_nbrs\( (u) \) do
    - \( a_u \leftarrow \text{sum}(a_u, \text{gather}(D_u, D_{(u,v)}, D_v)) \)
  - end

end

\( D_u \leftarrow \text{apply}(D_u, a_u) \)

- foreach neighbor \( v \) in scatter_nbrs\( (u) \) do
  - \((D_{(u,v)}, \Delta a) \leftarrow \text{scatter}(D_u, D_{(u,v)}, D_v)\)
  - if \( a_v \) and \( \Delta a \) are not Empty then
    - \( a_v \leftarrow \text{sum}(a_v, \Delta a) \)
  - else
    - \( a_v \leftarrow \text{Empty} \)
  - end

end
Vertex Partitioning

Edge Cuts

Vertext Cuts
Vertex Partitioning
How the vertices are partitioned

- Evenly assign edges to machines

- 3 different approaches
  - Random edge placement
  - Greedy placement
    - Coordinated edge placement
    - Oblivious edge placement
Random Edge Placements

**Machine 1**

**Machine 2**

**Machine 3**

**Balanced Vertex-Cut**
- \( Y \) Spans 3 Machines
- \( Z \) Spans 2 Machines
- **Not cut!**
Greedy Edge Placements

- Place edges on machines that already have the vertices in that edge
- If there are multiple options, choose the less loaded machine
Greedy Edge Placements

- Minimises the expected number of machines spanned
- Coordinated:
  - Requires coordination to place each edge
  - Slower but has higher quality cuts
- Oblivious:
  - Approximate greedy objective without coordination
  - Faster but lower quality cuts
Experiments - Graph Partitioning

Figure 7: (a) The actual replication factor on 32 machines. (b) The effect of partitioning on runtime.

Figure 8: (a,b) Replication factor and runtime of graph ingress for the Twitter follower network as a function of the number of machines for random, oblivious, and coordinated vertex-cuts.
Experiments - Synthetic Work Imbalance and Communication

Figure 9: Synthetic Experiments: Work Imbalance and Communication. (a, b) Standard deviation of worker computation time across 8 distributed workers for each abstraction on power-law fan-in and fan-out graphs. (b, c) Bytes communicated per iteration for each abstraction on power-law fan-in and fan-out graphs.
Experiments - Synthetic Runtime

Figure 10: Synthetic Experiments Runtime. (a, b) Per iteration runtime of each abstraction on synthetic power-law graphs.
Table 2: Relative performance of PageRank, triangle counting, and LDA on similar graphs. PageRank runtime is measured per iteration. Both PageRank and triangle counting were run on the Twitter follower network and LDA was run on Wikipedia. The systems are reported as number of nodes by number of cores.
Other Features

- 3 different execution modes:
  - Bulk Synchronous
  - Asynchronous
  - Asynchronous Serialisable

- Delta Caching
Critical Evaluation

- Lots of talk of performance, not many tests comparing systems
- Delta caching only briefly touched on
- Future work lacks detail
- Lots of unbacked up claims
- Greedy edge placement not very clear
- No mention of fault tolerance
Bibliography


And his original presentation found here:

http://www.cs.berkeley.edu/~jegonzal/talks/powergraph_osdi12.pptx