PowerGraph: Distributed Graph-Parallel Computation on Natural Graphs

J. E. Gonzales, Y. Low, H. Gu, D. Bickson, Carnegie Mellon University
C. Guestrin, University of Washington
Introduction

• New framework for distributed graph paralleled computation on natural graphs
• Transition from big data to big graphs
• Graphs are ubiquitous...

• Graphs encode relationships between:
  People    Products    Ideas
  Facts    Interests

❤ Billions of vertices and edges and rich metadata
Graphs are essential for Data-Mining and Machine Learning

• They help us identify influential people and information
• Find communities
• Target ads and products
• Model complex data dependencies
Problem: Existing distributed graph computation systems perform poorly on Natural Graphs

• Example: PageRank on Twitter Follower Graph

40M Users
1.4 Billion Links

Order of magnitude by exploiting properties of Natural Graphs

Runtime Per Iteration
Properties of the Natural Graphs

Power-Law Degree Distribution

More than $10^8$ vertices have one neighbor.

Top 1% of vertices are adjacent to 50% of the edges!

AltaVista WebGraph
1.4B Vertices, 6.6B Edges

Number of Vertices

Degree
Challenges of Natural Graphs

• Sparsity structure of natural graphs presents a unique challenge to efficient distributed graph-parallel computation

• Hallmark property: most vertices have relatively few neighbours while a few have many neighbours
Power-Law Degree Distribution

More than $10^8$ vertices have one neighbor.

Top 1% of vertices are adjacent to 50% of the edges!

Power-Law Degree Distribution

“Star Like” Motif

High-degree Vertices

Followers
Properties of the Natural Graphs

• Difficult to Partition
  – Power-Law graphs do not have low-cost balanced cuts
  – Traditional graph-partitioning algorithms perform poorly on Power-Law Graphs
PowerGraph

• Split High-Degree vertices:

• Introduction of new abstraction:

♥ EQUIVALENCE on Split Vertices ♥
How do we program graph computation?

• Graph-Parallel Abstraction
  – A user-defined Vertex-program runs on each vertex

• Pregel
  – Graph constrains interact using messages

• GraphLab
  – Graph constrains interact through shared state

• Parallelism: run multiple vertex program at the same time
PageRank Algorithm

• Example: The popularity of a user depends on the popularity of her followers, which depends on the popularity of their followers.

\[
R[i] = 0.15 + \sum_{j \in Nbrs(i)} w_{ji} R[j]
\]

• Update ranks in parallel

• Iterate process until convergence
Pregel PageRank

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);
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;
Challenges of High-Degree Vertices

- A lot of iterating over our neighborhood
- Pregel: many messages
- GraphLab: Touches a large number of states
Pregel Message Combiners on Fan-IN

- User defines commutative associative message operations:
Pregel Struggles with Fan-OUT

- Fan-OUT: Broadcast sends many copies of the same message to the same machine
GraphLab Ghosting

Changes to master are synced to ghosts
Fan-IN and Fan-Out performance

(c) Power-law Fan-In Comm.
(d) Power-law Fan-Out Comm.
Graph Partitioning

• Graph parallel abstractions rely on partitioning:
  – Minimize communication
  – Balance computation and storage

• Both GraphLab and Pregel resort to random partitioning on natural graphs
  – They randomly split vertices over machines

Theorem 5.1. If vertices are randomly assigned to $p$ machines then the expected fraction of edges cut is:

$$\mathbb{E} \left[ \frac{|Edges Cut|}{|E|} \right] = 1 - \frac{1}{p}. \quad (5.1)$$

10 Machines => 90% of edges cut
100 Machines => 99% of edges cut
In Summary

- GraphLab and Pregel are not well suited for computation of natural graphs
- Challenges of high-degree vertices
- Low quality partitioning
Main idea of PowerGraph

• GAS decomposition: distribute vertex – programs
  – Move computation to data
  – Parallelize high-degree vertices
• Represents three conceptual phases of a vertex-program:
  – Gather
  – Apply
  – Scatter
PowerGraph Abstraction

• Combines the best features from both Pregel and GraphLab
  – From GraphLab it borrows the data-graph and shared memory view of computation
  – From Pregel it borrows the commutative, associative gather concept
GAS Decomposition

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

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

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

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

**Scatter**
Update adjacent edges and vertices.

*User Defined:*
- \( \text{Scatter}(Y', Y') \rightarrow \)

Update Edge Data & Activate Neighbors.
PageRank in PowerGraph

\[ R[i] = 0.15 + \sum_{j \in \text{Nbrs}(i)} w_{ji} R[j] \]

```python
PowerGraph_PageRank(i):
    Gather(j -> i): return w_{ji} * R[j]
    sum(a, b): return a + b;

Apply(i, \Sigma): R[i] = 0.15 + \Sigma

Scatter(i -> j):
    if R[i] changed then trigger j to be recomputed
```
Example
New Approach to Partitioning

- Rather than cut edges:
  - Must synchronize many edges
  - CPU 1
  - CPU 2

- We cut vertices:
  - Must synchronize a single vertex
  - CPU 1
  - CPU 2

New Theorem: For any edge cut we can construct a vertex cut which requires strictly less communication and storage.
Constructing Vertex-Cuts

• Evenly assign edges to machines
  – Minimize machines spanned by each vertex
• Assign each edge as it is loaded
  – Touch each edge only once
• Three distributed approaches:
  – Random Edge Placement
  – Coordinated Greedy Edge Placement
  – Oblivious Greedy Edge Placement
Random Edge Placement

- Uniquely assigned to one machine
- Balanced cut
Greedy Vertex-Cuts

• Place edges on machines which already have the vertices in that edge.
• If more machines have the same vertex, place edge on less loaded machine
Greedy Vertex-Cuts

- Greedy minimizes the expected number of machines spanned
- Coordinated
  - Requires coordination to place each edge
  - Slower: higher quality cuts
- Oblivious
  - Approx. greedy objective without coordination
  - Faster: lower quality cuts
Partitioning Performance

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.
Partitioning Performance

(a) Actual Replication

(b) Effect of Partitioning

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

• Supports three execution modes:
  – Synchronous: Bulk-Synchronous GAS Phases
  – Asynchronous: Interleave GAS Phases
  – Asynchronous + Serializable: Neighbouring vertices do not run simultaneously

• Delta Caching
  – Accelerate gather phase by caching partial sums for each vertex
Implementation and Evaluation

• Technical details:
  – Experiments were performed on a 64 node cluster of Amazon EC2 Linux instances
  – Each instance has two quad core Intel Xeon X5570 processor with 23GB RAM and is connected via 10 GigE Ethernet
  – PowerGraph was written in C++ and compiled with GCC 4.5
System Design

• Built on top of
  – MPI/TCP-IP
  – Pthreads
  – HDFS
• Uses HDFS for Graph input and output
• Fault-tolerance is achieved by check-poining
  – Snapshot time <5 sec. for twitter network
## Implemented Algorithms

<table>
<thead>
<tr>
<th>Collaborative Filtering</th>
<th>• Graph Analytics</th>
</tr>
</thead>
<tbody>
<tr>
<td>- Alternating Least Squares</td>
<td>- PageRank</td>
</tr>
<tr>
<td>- Stochastic Gradient Descent</td>
<td>- Triangle Counting</td>
</tr>
<tr>
<td>- SVD</td>
<td>- Shortest Path</td>
</tr>
<tr>
<td>- Non-negative MF</td>
<td>- Graph Coloring</td>
</tr>
<tr>
<td></td>
<td>- K-core Decomposition</td>
</tr>
<tr>
<td>Statistical Inference</td>
<td>• Computer Vision</td>
</tr>
<tr>
<td>- Loopy Belief Propagation</td>
<td>- Image stitching</td>
</tr>
<tr>
<td>- Max-Product Linear Programs</td>
<td>• Language Modeling</td>
</tr>
<tr>
<td>- Gibbs Sampling</td>
<td>- LDA</td>
</tr>
</tbody>
</table>
Results

**Topic Modeling**

- English language Wikipedia
  - 2.6M Documents, 8.3M Words, 500M Tokens
  - Computationally intensive algorithm

### Million Tokens Per Second

- 100 Yahoo! Machines: Specifically engineered for this task
- 64 cc2.8xlarge EC2 Nodes: 200 lines of code & 4 human hours
Triangle Counting on The Twitter Graph
Identify individuals with strong communities.
Counted: 34.8 Billion Triangles

Hadoop
[WWW’11]
1536 Machines
423 Minutes

PowerGraph
64 Machines
1.5 Minutes
282 x Faster

More results
Thank you for your attention!

http://graphlab.org
Some of the slides were taken from the talk by J. E. Gonzalez, available on the website: https://www.usenix.org/conference/osdi12/technical-sessions/presentation/gonzalez