PowerGraph

Distributed Graph-Parallel Computation on Natural Graphs

Joseph Gonzalez

Joint work with:

- Yucheng Low
- Haijie Gu
- Danny Bickson
- Carlos Guestrin

Carnegie Mellon University
Graphs are ubiquitous..
• **Graphs encode relationships** between:

  People
  Products
  Ideas

  Facts
  Interests

• **Big:** billions of vertices and edges and rich metadata
Graphs are Essential to Data-Mining and Machine Learning

- Identify influential people and information
- Find communities
- Target ads and products
- Model complex data dependencies
Natural Graphs
Graphs derived from natural phenomena
Problem:

Existing *distributed* graph computation systems perform poorly on **Natural Graphs**.
PageRank on Twitter Follower Graph

Natural Graph with 40M Users, 1.4 Billion Links

Runtime Per Iteration

- Hadoop
- GraphLab
- Twister
- Piccolo

Order of magnitude by exploiting properties of Natural Graphs

Hadoop results from [Kang et al. '11]
Twister (in-memory MapReduce) [Ekanayake et al. ‘10]
Properties of Natural Graphs

Power-Law Degree Distribution
Power-Law Degree Distribution

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

High-Degree Vertices

AltaVista WebGraph
1.4B Vertices, 6.6B Edges
Power-Law Degree Distribution

“Star Like” Motif

President Obama

Followers
Power-Law Graphs are Difficult to Partition

- Power-Law graphs do not have **low-cost** balanced cuts \cite{Leskovec08,Lang04}
- Traditional graph-partitioning algorithms perform poorly on Power-Law Graphs. \cite{Abou-Rjeili06}
Properties of Natural Graphs

High-degree Vertices
Low Quality
Power-Law Degree Distribution
• Split **High-Degree** vertices

• New Abstraction $\rightarrow$ **Equivalence** on Split Vertices
How do we *program* graph computation?

“Think like a Vertex.”

-Malewicz et al. [SIGMOD’10]
The Graph-Parallel Abstraction

• A user-defined **Vertex-Program** runs on each vertex

• **Graph** constrains interaction along edges
  – Using **messages** (e.g. **Pregel** [PODC’09, SIGMOD’10])
  – Through shared state (e.g., **GraphLab** [UAI’10, VLDB’12])

• **Parallelism**: run multiple vertex programs simultaneously
Example

What’s the popularity of this user?

Depends on the popularity of her followers

Depends on the popularity their followers

Popular?
PageRank Algorithm

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

- Update ranks in parallel
- Iterate until convergence
The Pregel Abstraction

Vertex-Programs interact by sending messages.

Pregel_PageRank(i, messages) :

// Receive all the messages
total = 0
foreach (msg in messages) :
  total = total + msg

// Update the rank of this vertex
R[i] = 0.15 + total

// Send new messages to neighbors
foreach (j in out_neighbors[i]) :
  Send msg(R[i] * w_{ij}) to vertex j

Malewicz et al. [PODC’09, SIGMOD’10]
The GraphLab Abstraction

Vertex-Programs directly read the neighbors state

GraphLab_PageRank(i)

// Compute sum over neighbors
total = 0
foreach (j in in_neighbors(i)):
  total = total + R[j] * w_{ji}

// Update the PageRank
R[i] = 0.15 + total

// Trigger neighbors to run again
if R[i] not converged then
  foreach (j in out_neighbors(i)):
    signal vertex-program on j

Low et al. [UAI’10, VLDB’12]
Challenges of High-Degree Vertices

- 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)
Communication Overhead for High-Degree Vertices

Fan-In vs. Fan-Out
Pregel **Message Combiners on Fan-In**

- User defined *commutative associative* (+) message operation:
Pregel Struggles with **Fan-Out**

- **Broadcast** sends many copies of the same message to the same machine!
Fan-In and Fan-Out Performance

- PageRank on synthetic Power-Law Graphs
  - Piccolo was used to simulate Pregel with combiners

![Graph showing Total Communication (GB) vs. Power-Law Constant α for High Fan-Out and High Fan-In Graphs](image)

More high-degree vertices
GraphLab Ghosting

• Changes to master are synced to ghosts
• Changes to neighbors of high degree vertices creates substantial network traffic
Fan-In and Fan-Out Performance

• PageRank on synthetic Power-Law Graphs
• GraphLab is \textit{undirected}

![Graph showing Fan-In and Fan-Out Performance](image-url)

- Total Comm. (GB)
- Power-Law Constant alpha
- More high-degree vertices

- Pregel Fan-In
- Pregel Fan-Out
- GraphLab Fan-In/Out
Graph Partitioning

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

Machine 1

Machine 2

Data transmitted across network \( O(\# \text{ cut edges}) \)
Random Partitioning

- Both GraphLab and Pregel resort to random (hashed) partitioning on natural graphs

\[ E \left[ \frac{|Edges\ Cut|}{|E|} \right] = 1 - \frac{1}{p} \]

- 10 Machines \(\rightarrow\) 90% of edges cut
- 100 Machines \(\rightarrow\) 99% of edges cut!
In Summary

GraphLab and Pregel are not well suited for natural graphs

• Challenges of high-degree vertices
• Low quality partitioning
PowerGraph

• GAS Decomposition: distribute vertex-programs
  – Move computation to data
  – Parallelize high-degree vertices

• Vertex Partitioning:
  – Effectively distribute large power-law graphs
A Common Pattern for Vertex-Programs

GraphLab_PageRank(i)

// Compute sum over neighbors
total = 0
foreach (j in in_neighbors(i)):
  total = total + R[j] * w_{ji}

// Update the PageRank
R[i] = 0.1 + total

// Trigger neighbors to run again
if R[i] not converged then
  foreach (j in out_neighbors(i))
    signal vertex-program on j

Gather Information About Neighborhood

Update Vertex

Signal Neighbors & Modify Edge Data
GAS Decomposition

**G**ather (Reduce)
Accumulate information about neighborhood

*User Defined:*
- Gather($Y$) → $\Sigma$
- $\Sigma_1 + \Sigma_2 \to \Sigma_3$

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

*User Defined:*
- Apply($Y$, $\Sigma$) → $Y'$

**S**catter
Update adjacent edges and vertices.

*User Defined:*
- Scatter($Y'$) → $Y''$

Parallel Sum $\to \Sigma$

Update Edge Data & Activate Neighbors
PageRank in PowerGraph

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

\( \text{Gather}( j \rightarrow i ) : \text{return } w_{ji} \times R[j] \)

\( \text{sum}(a, b) : \text{return } a + b; \)

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

\( \text{Scatter}( i \rightarrow j ) : \)

if \( R[i] \) changed then trigger \( j \) to be recomputed
Distributed Execution of a PowerGraph Vertex-Program

Gather

Apply

Scatter
Minimizing Communication in PowerGraph

Communication is linear in the number of machines each vertex spans

A vertex-cut minimizes machines each vertex spans

Percolation theory suggests that power law graphs have good vertex cuts. [Albert et al. 2000]
New Approach to Partitioning

• Rather than cut edges:

New Theorem:

For any edge-cut we can directly construct a vertex-cut which requires strictly less communication and storage.

CPU 1

Must synchronize a single vertex

CPU 2
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

• Propose three **distributed** approaches:
  – *Random* *Edge Placement*
  – *Coordinated Greedy* *Edge Placement*
  – *Oblivious Greedy* *Edge Placement*
Random Edge-Placement

• Randomly assign edges to machines

Machine 1

Machine 2

Machine 3

Balanced Vertex-Cut

Y  Spans 3 Machines

Z  Spans 2 Machines

Not cut!
Analysis Random Edge-Placement

• Expected number of machines spanned by a vertex:

Twitter Follower Graph
41 Million Vertices
1.4 Billion Edges

Accurately Estimate Memory and Comm. Overhead
Random Vertex-Cuts vs. Edge-Cuts

- Expected improvement from vertex-cuts:

![Graph showing reduction in communication and storage vs. number of machines]
Greedy Vertex-Cuts

- Place edges on machines which already have the vertices in that edge.
Greedy Vertex-Cuts

• **De-randomization** $\rightarrow$ greedily minimizes the expected number of machines spanned

• **Coordinated** Edge Placement
  – Requires coordination to place each edge
  – Slower: higher quality cuts

• **Oblivious** Edge Placement
  – Approx. greedy objective without coordination
  – Faster: lower quality cuts
Oblivious balances cost and partitioning time.
Greedy Vertex-Cuts Improve Performance

Greedy partitioning improves computation performance.
Other Features (See Paper)

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

• Delta Caching
  – Accelerate gather phase by **caching** partial sums for each vertex
System Evaluation
System Design

- Implemented as C++ API
- Uses HDFS for Graph Input and Output
- Fault-tolerance is achieved by check-pointing
  - Snapshot time $< 5$ seconds for twitter network
Implemented Many Algorithms

- **Collaborative Filtering**
  - Alternating Least Squares
  - Stochastic Gradient Descent
  - SVD
  - Non-negative MF

- **Statistical Inference**
  - Loopy Belief Propagation
  - Max-Product Linear Programs
  - Gibbs Sampling

- **Graph Analytics**
  - PageRank
  - Triangle Counting
  - Shortest Path
  - Graph Coloring
  - K-core Decomposition

- **Computer Vision**
  - Image stitching

- **Language Modeling**
  - LDA
Comparison with GraphLab & Pregel

• PageRank on Synthetic Power-Law Graphs:

- **Communication**

  - Total Network (GB) vs. Power-Law Constant $\alpha$

- **Runtime**

  - Seconds vs. Power-Law Constant $\alpha$

- PowerGraph is robust to high-degree vertices.
PageRank on the Twitter Follower Graph

Natural Graph with 40M Users, 1.4 Billion Links

**Communication**

<table>
<thead>
<tr>
<th></th>
<th>Total Network (GB)</th>
<th>Seconds</th>
</tr>
</thead>
<tbody>
<tr>
<td>GraphLab</td>
<td>35</td>
<td>70</td>
</tr>
<tr>
<td>Pregel (Piccolo)</td>
<td>15</td>
<td>30</td>
</tr>
<tr>
<td>PowerGraph</td>
<td>5</td>
<td>10</td>
</tr>
</tbody>
</table>

**Runtime**

- GraphLab: 70 seconds
- Pregel (Piccolo): 30 seconds
- PowerGraph: 10 seconds

Reduces Communication

Runs Faster

32 Nodes x 8 Cores (EC2 HPC cc1.4x)
PowerGraph is Scalable

Yahoo Altavista Web Graph (2002):
One of the largest publicly available web graphs

1.4 Billion Webpages, 6.6 Billion Links

7 Seconds per Iter.
1B links processed per second
30 lines of user code
**Topic Modeling**

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

---

**Smola et al.**

Specifically engineered for this task

**PowerGraph**

64 cc2.8xlarge EC2 Nodes

200 lines of code & 4 human hours

---

**Million Tokens Per Second**

0 20 40 60 80 100 120 140 160

100 Yahoo! Machines

---

53
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

Why? Wrong Abstraction →
Broadcast $O(\text{degree}^2)$ messages per Vertex

S. Suri and S. Vassilvitskii, “Counting triangles and the curse of the last reducer,” WWW’11
Summary

• *Problem*: Computation on **Natural Graphs** is challenging
  – High-degree vertices
  – Low-quality edge-cuts

• *Solution*: **PowerGraph System**
  – **GAS Decomposition**: split vertex programs
  – **Vertex-partitioning**: distribute natural graphs

• **PowerGraph** *theoretically* and *experimentally* outperforms existing graph-parallel systems.
Machine Learning and Data-Mining Toolkits

- Graph Analytics
- Graphical Models
- Computer Vision
- Clustering
- Topic Modeling
- Collaborative Filtering

PowerGraph (GraphLab2) System
Future Work

• Time evolving graphs
  – Support *structural changes* during computation

• Out-of-core storage (GraphChi)
  – Support graphs that don’t fit in memory

• Improved Fault-Tolerance
  – Leverage *vertex replication* to reduce snapshots
  – *Asynchronous* recovery
PowerGraph

is GraphLab Version 2.1

Apache 2 License

http://graphlablab.org

Documentation... Code... Tutorials... (more on the way)