PowerGraph
Distributed Graph-Parallel Computation on Natural Graphs
Lingyun (Luke) Li
Large-scale graph-structured computation

- Central to tasks ranging from targeted advertising to natural language processing
- Billions of vertices, edges and extremely rich data

Social Media  Science  Advertising  Web

- Facebook
- Twitter
- ACM
- Amazon
- Netflix
- Google
Existing Graph-Parallel Abstractions

- A sparse graph $G = \{V,E\}$
- A vertex-program $Q$ which is executed in parallel on each vertex $v \in V$
- $Q(v)$ interact with neighboring instances $Q(u)$ where $(u, v) \in E$
- Communication through shared-state in GraphLab, or messages in Pregel
Existing Graph-Parallel Abstractions

- GAS: three conceptual phases of a vertex-program: Gather, Apply, and Scatter
- Constrain the interaction of vertex program to a graph structure to enable the optimization of data-layout and communication
- Rely on each vertex having a small neighborhood to maximize parallelism and effective partitioning to minimize communication
Natural Graphs

- Commonly found in the real-world
- Highly skewed power-law degree distributions
- Poor performance on existing distributed graph computation systems
Skewed Power-Law Degree Distribution

- Most vertices have relatively few neighbors while a few have many neighbors
- Star like graph

![Graph showing a skewed power-law degree distribution](image)

- More than $10^8$ vertices have one neighbor.
- High-Degree Vertices

Altavista WebGraph
1.4B Vertices, 6.6B Edges
Challenges of Natural Graphs

- Partitioning
  - GraphLab and Pregel depend on graph partitioning to minimize communication and ensure work balance.
  - Performs poorly on power-law graphs
Challenges of Natural Graphs

- Work imbalance
- Communication
  - Communication asymmetry
  - Generate and send many identical messages
Challenges of Natural Graphs

- **Storage**
  - Locally store the adjacency information for each vertex
  - Storage linear in degree of vertex

- **Computation**
  - No parallelism within individual vertex-programs
  - Limiting scalability on high-degree vertices
PowerGraph

- GAS decomposition to distribute a single vertex-program over multiple machines
- Vertex partitioning: effectively distribute large power-law graphs
- Eliminates the degree dependence of the vertex-program
GAS Decomposition

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

*User Defined:*
- \( \text{Gather}(\mathbf{Y}) \to \Sigma \)
- \( \Sigma_1 + \Sigma_2 \to \Sigma_3 \)

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

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

**Scatter**
Update adjacent edges and vertices.

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

Update Edge Data & Activate Neighbors
PageRank

\[
gather(D_u, D_{(u,v)}, D_v):
\]
\[
    \text{return } D_v.\text{rank} / \#\text{outNbrs}(v)
\]

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

\[
\text{apply}(D_u, acc):
\]
\[
    rnew = 0.15 + 0.85 \ast acc
\]
\[
    D_u.\text{delta} = (rnew - D_u.\text{rank}) / \#\text{outNbrs}(u)
\]
\[
    D_u.\text{rank} = rnew
\]

// scatter_nbrs: OUT_NBRS

\[
\text{scatter}(D_u, D_{(u,v)}, D_v):
\]
\[
    \text{if}(|D_u.\text{delta}| > \varepsilon) \text{ Activate}(v)
\]
\[
    \text{return } \text{delta}
\]
Distribution of A PowerGraph Vertex-Program

Machine 1
Master

Machine 2
Mirror

Machine 3
Mirror

Machine 4
Mirror

Gather
Distribution of A PowerGraph Vertex-Program

Gather

Machine 1

\[ \Sigma_1 + \Sigma_2 + \Sigma_3 + \Sigma_4 \]

Machine 2

Machine 3

Machine 4

Mirror

Mirror
Distribution of A PowerGraph Vertex-Program
Distribution of A PowerGraph Vertex-Program
Distribution of A PowerGraph Vertex-Program
Balanced Vertex-Cut

- Evenly assign edges to machines
- Store edge only once
- Edge data do not need to be communicated

- Allow vertices to span multiple machines
- Changes to a vertex must be copied to all the machines it spans
- **Storage and network overhead depend on the number of machines spanned by each vertex.**

- Theorem: For any edge-cut, we can establish a vertex cut that requires strictly less communication and storage
Balanced Vertex-Cut

- Random
- Greedy
  - Assign edge \((u, v)\) to the machine that already contains vertex \(u\) or \(v\)
  - Assign to least loaded machine if there are multiple choices to ensure work balance
  - Two implementations:
    - Coordinated:
      - Coordination between machines
      - Higher quality cuts
      - Slower
    - Oblivious
      - No coordination
      - Low quality cuts
      - Faster
Balanced Vertex-Cut

(a) Replication Factor (Twitter)  (b) Ingress time (Twitter)
Abstraction Comparison: Work Imbalance

(a) Power-law Fan-In Balance

(b) Power-law Fan-Out Balance
Abstraction Comparison: Communication Volume

(c) Power-law Fan-In Comm.  (d) Power-law Fan-Out Comm.
Abstraction Comparison: Runtime

(a) Power-law Fan-In Runtime
(b) Power-law Fan-Out Runtime
Summary

- **Problem**: Computation on large-scale Nature Graphs is challenging
  - High-degree vertices
  - Low quality edge-cut partition

- **Solution**: PowerGraph
  - GAS Decomposition: distribute vertex programs
  - Balanced Vertex-Cut: partition natural graphs
  - Outperforms existing Graph-Parallel systems
Other Contributions

- A delta caching procedure which allows computation state to be dynamically maintained
- A theoretical characterization of network and storage
- A high-performance open-source implementation of the PowerGraph abstraction
- A comprehensive evaluation of three implementations of PowerGraph on a large EC2 deployment using real-world MLDM applications
Thank you!