A Framework for Machine Learning and Data Mining in the Cloud

Vicky Papavasileiou, UCSD
Slides adapted from SelectLab at Carnegie Mellon
Big Data is Everywhere

6 Billion
Flickr Photos

28 Million
Wikipedia Pages

900 Million
Facebook Users

72 Hours a Minute
YouTube

“...growing at 50 percent a year...”

“... data a new class of economic asset, like currency or gold.”
Big Learning

How will we **design** and **implement** *Big* learning systems?
Shift Towards Use Of Parallelism in ML

- GPUs
- Multicore
- Clusters
- Clouds
- Supercomputers

• Graduate students repeatedly solve the same parallel design challenges:
  – Race conditions, distributed state, communication...

• Resulting code is very specialized:
  – difficult to maintain, extend, debug...

Avoid these problems by using high-level abstractions
Graphs are Everywhere

Social Network

Collaborative Filtering

Probabilistic Analysis

Text Analysis
Properties of Computation on Graphs

Dependency Graph

Local Updates

Iterative Computation

My Interests

Friends Interests
Why not use Map-Reduce for Graph Parallel Algorithms?
Data Dependencies

• Map-Reduce does not efficiently express dependent data
  – User must code substantial data transformations
  – Costly data replication
MapAbuse: Iterative MapReduce

System is not optimized for iteration:

![Diagram showing MapReduce system with iterations illustrating startup and disk penalties.](image-url)
The GraphLab Framework

Graph Based Data Representation

Update Functions User Computation

Iterative computation
Netflix Collaborative Filtering

Ideal

D=100

D=20

# machines

# machines

Speedup

Runtime(s)

Hadoop

MPI

GraphLab
Data Graph

A graph with arbitrary data (C++ Objects) associated with each vertex and edge.

Graph:
- Social Network

Vertex Data:
- User profile text
- Current interests estimates

Edge Data:
- Similarity weights
Update Functions

User-defined program: applied to a **vertex** and transforms data in **scope** of vertex

```plaintext
Pagerank(scope)
// Update the current vertex data
vertex.PageRank = \alpha
ForEach inPage:
    vertex.PageRank += (1 - \alpha) \times inPage.PageRank
// Reschedule Neighbors if needed
if vertex.PageRank changes then
    reschedule_all_neighbors;
```

**Pagerank** is a program that updates the PageRank of a vertex and reschedules its neighbors if its PageRank changes.
Natural Graphs
Graphs derived from natural phenomena
Problem:

Existing *distributed* graph computation systems perform poorly on **Natural Graphs**.
Power-Law Degree Distribution

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

High-Degree Vertices

AltaVista WebGraph
1.4B Vertices, 6.6B Edges
Graph Partitioning

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

Data transmitted across network $O(\# \text{ cut edges})$
Power-Law Graphs are Difficult to Partition

• Power-Law graphs do not have low-cost balanced cuts [Leskovec et al. 08, Lang 04]
• Traditional graph-partitioning algorithms perform poorly on Power-Law Graphs. [Abou-Rjeili et al. 06]
• GAS Decomposition: distribute vertex-programs
  – Move computation to data
  – Parallelize high-degree vertices

• Vertex Partitioning:
  – Effectively distribute large power-law graphs
PageRank on Twitter Follower Graph

Natural Graph with 40M Users, 1.4 Billion Links

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

Order of magnitude by exploiting properties of Natural 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
GAS Decomposition

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

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

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

*User Defined:*
- **Apply**($Y$, $\Sigma$) $\rightarrow \overset{*}{Y}$

**Scatter**
Update adjacent edges and vertices.

*User Defined:*
- **Scatter**($Y'$) $\rightarrow \overset{*}{Y}'$

Parallel Sum
$\Rightarrow \Sigma$

Update Edge Data & Activate Neighbors
PageRank in PowerGraph

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

PowerGraph_PageRank(i)

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

sum(a, b) : return a + b;

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

Scatter( i \rightarrow j ) :
if \ R[i] changed then trigger \ j \ to be recomputed
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.

•

Must synchronize a single vertex
Distributed Execution of a PowerGraph Vertex-Program

Gather
Apply
Scatter

Machine 1
Machine 2
Machine 3
Machine 4
Modes of execution

• Synchronous
  – Follows Bulk Synchronous Processing model
  – Computation happens in super-steps: First, all vertices gather, then all vertices apply and then all vertices scatter

• Asynchronous
  – GAS phases are interleaved
  – Consistency achieved through locking neighborhood
  – Non-deterministic
  – High throughput
Implementations in GraphLab

PageRank
Topic Modeling

• Latent Dirichlet Model
  – Assigns topics to tokens (occurrence of word in document)
  – Assigns a distribution over topics to each document and a distribution over words to each topic
  – Assumptions:
    • Words in the same document are topically related.
    • Documents that share common terms are topically related.

• Approximate Inference Algorithm
  – Collapsed Gibbs Sampler by Griffiths and Steyvers “Finding Scientific Topics”
Topic Modeling in GraphLab

• Bipartite graph

• Vertex data: Vector of topic counts

• Edge data: tokens (#occurrence of word in doc) and latent topic assignments for that token
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