Big Learning with Graphs

Joseph Gonzalez
jegonzal@cs.cmu.edu

Yucheng Low
Aapo Kyrola
Haijie Gu
Danny Bickson
Arthur Gretto
Carlos Guestrin
Alex Smola
Joe Hellerstein
David O’Hallaron
Guy Blelloch
The Age of Big Data

“...growing at 50 percent a year...”

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

Big Graphs
• **Graphs encode relationships** between:

  People          Products          Ideas
  Facts           Interests

• **Big: billions of vertices** and **edges** and rich metadata
Big graphs present exciting new opportunities ...
Big-Graphs are Essential to Data-Mining and Machine Learning

- Identify influential people and information
- Find communities
- Target ads and products
- Model complex data dependencies
Big Learning with Graphs

Understanding and using large-scale structured data.
Examples
PageRank (Centrality Measures)

• Iterate:

\[ R[i] = \alpha + (1 - \alpha) \sum_{(j,i) \in E} \frac{1}{L[j]} R[j] \]

• Where:

– \( \alpha \) is the random reset probability
– \( L[j] \) is the number of links on page \( j \)

\[ R[5] = \alpha + (1 - \alpha) \left( \frac{1}{3} R[1] + \frac{1}{1} R[4] \right) \]

Label Propagation (Structured Prediction)

- **Social Arithmetic:**
  - 50% What I list on my profile
  - 40% Sue Ann Likes
  - 10% Carlos Like
  
  I Like: 60% Cameras, 40% Biking

- **Recurrence Algorithm:**
  \[ Likes[i] = \sum_{j \in \text{Friends}[i]} W_{ij} \times Likes[j] \]
  - iterate until convergence

- **Parallelism:**
  - Compute all \( Likes[i] \) in parallel

---

Collaborative Filtering: Independent Case

- Lord of the Rings
- Star Wars IV
- Star Wars I
- Harry Potter
- Pirates of the Caribbean
Collaborative Filtering: Exploiting Dependencies

What do I recommend???

Women on the Verge of a Nervous Breakdown
The Celebration
City of God
Wild Strawberries
La Dolce Vita
Matrix Factorization
Alternating Least Squares (ALS)

Iterate:

\[ u_i = \arg \min_w \sum_{j \in N[i]} (r_{ij} - m_j \cdot w)^2 \]

\[ m_j = \arg \min_w \sum_{i \in N[j]} (r_{ij} - u_i \cdot w)^2 \]

http://dl.acm.org/citation.cfm?id=1424269
Many More Algorithms

• Collaborative Filtering
  – Alternating Least Squares
  – Stochastic Gradient Descent
  – Tensor Factorization
  – SVD

• Structured Prediction
  – Loopy Belief Propagation
  – Max-Product Linear Programs
  – Gibbs Sampling

• Semi-supervised ML
  – Graph SSL
  – CoEM

• Graph Analytics
  – PageRank
  – Single Source Shortest Path
  – Triangle-Counting
  – Graph Coloring
  – K-core Decomposition
  – Personalized PageRank

• Classification
  – Neural Networks
  – Lasso
  ...

...
Graph Parallel Algorithms

Dependency Graph

Local Updates

Iterative Computation

My Interests

Friends Interests
What is the right tool for Graph-Parallel ML

Map Reduce
- Feature Extraction
- Cross Validation
- Computing Sufficient Statistics

Map Reduce?
- Collaborative Filtering
- Graph Analytics
- Structured Prediction
- Clustering
Why not use *Map-Reduce* for *Graph Parallel* algorithms?
Data Dependencies are Difficult

- Difficult to express dependent data in Map Reduce
  - Substantial data transformations
  - User managed graph structure
  - Costly data replication
Iterative Computation is Difficult

• System is not optimized for iteration:
Map-Reduce for Data-Parallel ML

• Excellent for large data-parallel tasks!

Data-Parallel → Graph-Parallel

**Map Reduce**

- Feature Extraction
- Cross Validation
- Computing Sufficient Statistics

**MPI/Pthreads**

- Collaborative Filtering
- Graph Analytics
- Structured Prediction
- Clustering

21
We could use ....

Threads, Locks, & Messages

“low level parallel primitives”
Threads, Locks, and Messages

• Graduate students repeatedly solve the same parallel design challenges:
  – Implement and debug complex parallel system
  – Tune for a specific parallel platform
  – Six months later the conference paper contains:
    “We implemented _______ in parallel.”

• The resulting code:
  – is difficult to maintain
  – is difficult to extend
  • couples learning model to parallel implementation
Addressing Graph-Parallel ML

• We need alternatives to Map-Reduce

Map Reduce

- Feature Extraction
- Cross Validation
- Computing Sufficient Statistics

Pregel

- Collaborative Filtering
- Graph Analytics
- Structured Prediction
- Clustering
Pregel Abstraction

• User-defined **Vertex-Program** on each vertex
• Vertex-programs interact along edges in the **Graph**
  – Programs interact through Messages
• **Parallelism**: Multiple vertex programs run simultaneously
The Pregel Abstraction

Vertex-Programs communicate through messages

```c
void Pregel_PageRank(i, msgs) :
    float total = sum(m in msgs)

    R[i] = \beta + (1-\beta)*total

    foreach(j in out_neighbors[i]) :
        SendMsg(nbr, R[i] * w_{ij})
```

26
Pregel is Bulk Synchronous Parallel

Compute

Communicate

http://dl.acm.org/citation.cfm?id=1807184
Open Source Implementations

- Giraph: http://incubator.apache.org/giraph/
- Golden Orb: http://goldenorbos.org/
- Stanford GPS: http://infolab.stanford.edu/gps/

An asynchronous variant:

- GraphLab: http://graphlab.org/
Tradeoffs of the BSP Model

• Pros:
  – *Graph Parallel*
  – Relatively easy to implement and reason about
  – *Deterministic execution*

• Cons:
  – User must architect the movement of information
    • Send the correct information in messages
  – Bulk synchronous abstraction inefficient
Curse of the Slow Job

Curse of the Slow Job

- Assuming runtime is drawn from an exponential distribution with mean 1.

![Curse of the Slow Job Graph](http://www.www2011india.com/proceeding/proceedings/p607.pdf)
Bulk synchronous parallel model *provably inefficient* for some graph-parallel tasks
Example:
Loopy Belief Propagation (Loopy BP)

• Iteratively estimate the “beliefs” about vertices
  – Read in messages
  – Updates marginal estimate (belief)
  – Send updated out messages
• Repeat for all variables until convergence

Bulk Synchronous Loopy BP

• Often considered embarrassingly parallel
  – Associate processor with each vertex
  – Receive all messages
  – Update all beliefs
  – Send all messages

• Proposed by:
  – Brunton et al. CRV’06
  – Mendiburu et al. GECC’07
  – Kang, et al. LDMTA’10
  – …
Sequential Computational Structure
Hidden Sequential Structure
Hidden **Sequential** Structure

- **Running Time:**

\[
\frac{2n \text{ Messages Calculations}}{p \text{ Processors}} \times (n \text{ Iterations to Converge}) = \frac{2n^2}{p}
\]

- Time for a single parallel iteration
- Number of Iterations
Optimal Sequential Algorithm

<table>
<thead>
<tr>
<th>Bulk Synchronous</th>
<th>Running Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>$2n^2/p$</td>
</tr>
<tr>
<td>-</td>
<td>$p \leq 2n$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Forward-Backward</th>
<th>Gap</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>$2n$</td>
</tr>
<tr>
<td>-</td>
<td>$p = 1$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Optimal Parallel</th>
<th>-</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>$n$</td>
</tr>
<tr>
<td>-</td>
<td>$p = 2$</td>
</tr>
</tbody>
</table>
The Splash Operation

- Generalize the optimal chain algorithm:

  1) Grow a BFS Spanning tree with fixed size
  2) Forward Pass computing all messages at each vertex
  3) Backward Pass computing all messages at each vertex

Prioritize Computation

Challenge = Boundaries

Synthetic Noisy Image

Vertex Updates

Algorithm identifies and focuses on hidden sequential structure
Comparison of Splash and Pregel Style Computation

Limitations of bulk synchronous model can lead to \textit{provably} inefficient parallel algorithms.
The Need for a New Abstraction

Need: Asynchronous, Dynamic Parallel Computations

- Data-Parallel
  - Map Reduce
    - Feature Extraction
    - Cross Validation
    - Computing Sufficient Statistics

- Graph-Parallel
  - BSP, e.g., Pregel
    - Graphical Models
      - Gibbs Sampling
      - Belief Propagation
      - Variational Opt.
    - Semi-Supervised Learning
      - Label Propagation
      - CoEM
    - Collaborative Filtering
      - Tensor Factorization
    - Data-Mining
      - PageRank
      - Triangle Counting
The GraphLab Goals

- Designed specifically for ML
  - Graph dependencies
  - Iterative
  - Asynchronous
  - Dynamic

- Simplifies design of parallel programs:
  - Abstract away hardware issues
  - Automatic data synchronization
  - Addresses multiple hardware architectures

Know how to solve ML problem on 1 machine

Efficient parallel predictions
Data Graph

Data associated with vertices and edges

Graph:
• Social Network

Vertex Data:
• User profile text
• Current interests estimates

Edge Data:
• Similarity weights
Update Functions

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

Update function applied (asynchronously) in parallel until convergence

Many schedulers available to prioritize computation
The scheduler determines the order that vertices are updated.

The process repeats until the scheduler is empty.
Ensuring Race-Free Code

How much can computation overlap?
Need for Consistency?

- Higher Throughput (#updates/sec)
- No Consistency
- Potentially Slower Convergence of ML
Consistency in Collaborative Filtering

GraphLab guarantees consistent updates

User-tunable consistency levels trades off parallelism & consistency

Netflix data, 8 cores
The GraphLab Framework

Graph Based
Data Representation

Update Functions
User Computation

Scheduler

Consistency Model
Bayesian Tensor Factorization
Gibbs Sampling
Dynamic Block Gibbs Sampling
Matrix Factorization
Lasso
Belief Propagation
CoEM
PageRank
SVM
GraphLab
Alternating Least Squares
Linear Solvers
LDA
Gibbs Sampling
SVD
Splash Sampler
Bayesian Tensor Factorization
...Many others...
GraphLab vs. Pregel (BSP)

PageRank (25M Vertices, 355M Edges)

- 51% updated only once
Never Ending Learner Project (CoEM)

<table>
<thead>
<tr>
<th></th>
<th>Number of CPUs</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hadoop</td>
<td>95 Cores</td>
<td>7.5 hrs</td>
</tr>
<tr>
<td>GraphLab</td>
<td>16 Cores</td>
<td>30 min</td>
</tr>
<tr>
<td>Distributed</td>
<td>32 EC2 machines</td>
<td>80 secs</td>
</tr>
</tbody>
</table>

0.3% of Hadoop time
The Cost of the Wrong Abstraction

Log-Scale!
Thus far...

GraphLab1 provided exciting scaling performance

But...

We couldn’t scale up to Altavista Webgraph 2002
1.4B vertices, 6.7B edges
Natural Graphs
Assumptions of **Graph-Parallel** Abstractions

**Idealized Structure**

- *Small* neighborhoods
  - Low degree vertices
- Similar degree
- Easy to partition

**Natural Graph**

- *Large* Neighborhoods
  - High degree vertices
- *Power-Law* degree distribution
- *Difficult to partition*
Natural Graphs ➔ Power Law

Top 1% of vertices is adjacent to 53% of the edges!

Altavista Web Graph: 1.4B Vertices, 6.7B Edges
High Degree Vertices are Common

“Social” People

Popular Movies

Hyper Parameters

Common Words

α
θ
Z
W

B

 Docs

LDA

Words

Obama

Netflix

Harry Potter

θ
Z
w

B
α
Problem:
High Degree Vertices Limit Parallelism

- Edge information too large for single machine
- Touches a large fraction of graph (GraphLab 1)
- Produces many messages (Pregel)
- Sequential Vertex-Updates

- Asynchronous consistency requires heavy locking (GraphLab 1)
- Synchronous consistency is prone to stragglers (Pregel)
Problem:
High Degree Vertices $\rightarrow$ High Communication for Distributed Updates

Natural graphs do not have low-cost balanced cuts
[Leskovec et al. 08, Lang 04]

Popular partitioning tools (Metis, Chaco,...) perform poorly
[Abou-Rjeili et al. 06]

Extremely slow and require substantial memory
Both GraphLab1 and Pregel proposed Random (hashed) partitioning for Natural Graphs.

For $p$ Machines:

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

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

GraphLab1 and Pregel are not well suited for natural graphs

- Poor performance on high-degree vertices
- Low Quality Partitioning
Distribute a single vertex-update
- Move computation to data
- Parallelize high-degree vertices

Vertex Partitioning
- Simple online approach, effectively partitions large power-law graphs
Factorized Vertex Updates

Split update into 3 phases

Parallel Sum

Locally apply the accumulated $\Delta$ to vertex

Data-parallel over edges

Data-parallel over edges

Gather

Apply($\Delta$, $Y$)

Locally apply the accumulated $\Delta$ to vertex

Update neighbors

Scatter

Apply($\Delta$, $Y$)

Locally apply the accumulated $\Delta$ to vertex
PageRank in GraphLab2

\[ R[i] = \beta + (1 - \beta) \sum_{(j,i) \in E} w_{ji} R[j] \]

**PageRankProgram**

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

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

**Apply** \((i, \Sigma)\) : \(R[i] = \beta + (1 - \beta) \times \Sigma\)

**Scatter** \((i \rightarrow j)\) :
  
  if \((R[i] \text{ changes})\) then \text{activate}(j)
Distributed Execution of a GraphLab2 Vertex-Program

Gather

Apply

Scatter
Minimizing Communication in GraphLab2

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]
Minimizing Communication in GraphLab2: Vertex Cuts

A vertex-cut minimizes # machines per vertex

Percolation theory suggests Power Law graphs can be split by removing only a small set of vertices [Albert et al. 2000]

Small vertex cuts possible!
Constructing Vertex-Cuts

- **Goal:** *Parallel graph partitioning on ingress*
- GraphLab 2 provides three **simple** approaches:
  - **Random Edge Placement**
    - Edges are placed randomly by each machine
      - Good theoretical guarantees
  - **Greedy Edge Placement with Coordination**
    - Edges are placed using a shared objective
      - Better theoretical guarantees
  - **Oblivious-Greedy Edge Placement**
    - Edges are placed using a local objective
Random Vertex-Cuts

• Randomly assign edges to machines

Machine 1

Machine 2

Machine 3

Balanced Cut

Y: Spans 3 Machines
Z: Spans 2 Machines
Filled Circle: Spans only 1 machine
Random Vertex Cuts vs Edge Cuts

Memory and Comm. Reduction w. Vertex Cuts

Number of Machines
Greedy Vertex-Cuts

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

• **Derandomization**: Minimizes the expected number of machines spanned by each vertex.

• **Coordinated**
  – Maintain a shared placement history (DHT)
  – Slower but higher quality

• **Oblivious**
  – Operate only on local placement history
  – Faster but lower quality
Partitioning Performance

Twitter Graph: 41M vertices, 1.4B edges

Oblivious balances partition quality and partitioning time.
Beyond Random Vertex Cuts!

![Bar chart showing reduction in runtime for PageRank, Collaborative Filtering, and Shortest Path with Random, Oblivious, and Greedy methods.](chart.png)
From the Abstraction to a System
GraphLab Version 2.1 API (C++)

Sync. Engine
  Fault Tolerance
Async. Engine
  Distributed Graph
  Map/Reduce
  Ingress

MPI/TCP-IP Comms
  PThreads
  Boost
  Hadoop/HDFS

Linux Cluster Services (Amazon AWS)

Carnegie Mellon

Select Lab
Triangle Counting in Twitter Graph

Total: 34.8 Billion Triangles

40M Users
1.2B Edges

Hadoop results from [Suri & Vassilvitskii '11]

Hadoop
1536 Machines
423 Minutes

GraphLab
64 Machines, 1024 Cores
1.5 Minutes
LDA Performance

- All English language Wikipedia
  - 2.6M documents, 8.3M words, 500M tokens

- LDA state-of-the-art sampler (100 Machines)
  - *Alex Smola*: 150 Million tokens per Second

- GraphLab Sampler (64 cc2.8xlarge EC2 Nodes)
  - **100 Million Tokens per Second**
  - Using only 200 Lines of code and 4 human hours
PageRank

40M Webpages, 1.4 Billion Links

Hadoop results from [Kang et al. '11]
Twister (in-memory MapReduce) [Ekanayake et al. ‘10]
How well does GraphLab scale?

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

1.4B Webpages, 6.6 Billion Links

11 Mins

1B links processed per second
30 lines of user code

1024 Cores (2048 HT)

4.4 TB RAM
GraphLab
Release 2.1
available now
Apache 2 License
GraphLab easily incorporates external toolkits
Automatically detects and builds external toolkits
Graph Processing

Extract knowledge from graph structure

- Find communities
- Identify important individuals
- Detect vulnerabilities

Algorithms

- Triangle Counting
- Pagerank
- K-Cores
- Shortest Path

Coming soon:

- Max-Flow
- Matching
- Connected Components
- Label propagation
Collaborative Filtering

**Understanding Peoples Shared Interests**

- Target advertising
- Improve shopping experience

**Algorithms**

- ALS, Weighted ALS
- SGD, Biased SGD

Proposed:

- SVD++
- Sparse ALS
- Tensor Factorization
Graphical Models

Probabilistic analysis for correlated data.

- Improved predictions
- Quantify uncertainty
- Extract relationships

Algorithms

- Loopy Belief Propagation
- Max Product LP

Coming soon:

- Gibbs Sampling
- Parameter Learning
- $\ell_1$ Structure Learning
- $M^3$ Net
- Kernel Belief Propagation
Structured Prediction

• Input:
  – Prior probability for each vertex
  – Edge List
  – Smoothing Parameter (e.g., 2.0)

• Output: posterior

<table>
<thead>
<tr>
<th>User Id</th>
<th>Pr(Conservative)</th>
<th>Pr(Not Conservative)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.8</td>
<td>0.2</td>
</tr>
<tr>
<td>2</td>
<td>0.5</td>
<td>0.5</td>
</tr>
<tr>
<td>3</td>
<td>0.3</td>
<td>0.7</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>User Id</th>
<th>Pr(Conservative)</th>
<th>Pr(Not Conservative)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.7</td>
<td>0.3</td>
</tr>
<tr>
<td>2</td>
<td>0.3</td>
<td>0.7</td>
</tr>
<tr>
<td>3</td>
<td>0.1</td>
<td>0.8</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
Computer Vision (CloudCV)

Making sense of pictures.

- Recognizing people
- Medical imaging
- Enhancing images

Algorithms

- Image stitching
- Feature extraction

Coming soon:

- Person/object detectors
- Interactive segmentation
- Face recognition
Clustering

Identify groups of related data

- Group customer and products
- Community detection
- Identify outliers

Algorithms

- K-Means++
- Coming soon:
  - Structured EM
  - Hierarchical Clustering
  - Nonparametric *-Means
Topic Modeling

Extract meaning from raw text

Algorithms
- LDA Gibbs Sampler
- Coming soon:
  - CVB0 for LDA
  - LSA/LSI
- Correlated topic models
- Trending Topic Models

Improved search
Summarize textual data
Find related documents
GraphChi: Going small with GraphLab

Solve huge problems on small or embedded devices?

Key: Exploit non-volatile memory (starting with SSDs and HDs)
GraphChi – disk-based GraphLab

Novel Parallel Sliding Windows algorithm

- Fast!
- Solves tasks as large as current distributed systems
- Minimizes disk seeks
  - Efficient on both SSD and hard-drive
- Multicore Asynchronous execution
Triangle Counting in Twitter Graph

- 40M Users
- 1.2B Edges
- Total: 34.8 Billion Triangles

**Hadoop**
- 1536 Machines
- 423 Minutes

**GraphChi**
- 59 Minutes, 1 Mac Mini!

**GraphLab**
- 64 Machines, 1024 Cores
- 1.5 Minutes

Hadoop results from [Suri & Vassilvitskii '11]
Release 2.1 available now
http://graphlab.org

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

GraphChi 0.1 available now
http://graphchi.org

Select Lab

Carnegie Mellon
Open Challenges
Dynamically Changing Graphs

• **Example:** *Social Networks*
  – New users $\rightarrow$ New Vertices
  – New Friends $\rightarrow$ New Edges

• How do you adaptively maintain computation:
  – Trigger computation with changes in the graph
  – Update “interest estimates” only where needed
  – Exploit asynchrony
  – Preserve consistency
Graph Partitioning

• How can you quickly place a large data-graph in a distributed environment:

• Edge separators fail on large power-law graphs
  – Social networks, Recommender Systems, NLP

• Constructing vertex separators at scale:
  – No large-scale tools!
  – How can you adapt the placement in changing graphs?
Graph Simplification for Computation

• Can you construct a “sub-graph” that can be used as a proxy for graph computation?

• See Paper:
  – *Filtering: a method for solving graph problems in MapReduce.*
    • [http://research.google.com/pubs/pub37240.html](http://research.google.com/pubs/pub37240.html)
Concluding BIG Ideas

• Modeling Trend: Independent Data → Dependent Data
  – Extract more signal from noisy structured data
• Graphs model data dependencies
  – Captures locality and communication patterns
• Data-Parallel tools not well suited to Graph Parallel problems
• Compared several Graph Parallel Tools:
  – Pregel / BSP Models:
    • Easy to Build, Deterministic
    • Suffers from several key inefficiencies
  – GraphLab:
    • Fast, efficient, and expressive
    • Introduces non-determinism
  – GraphLab2:
    • Addresses the challenges of computation on Power-Law graphs
• Open Challenges: Enormous Industrial Interest
Fault Tolerance
Checkpoint Construction

Pregel (BSP)

Synchronous Checkpoint Construction

GraphLab

Asynchronous Checkpoint Construction
Checkpoint Interval

- **Tradeoff:**
  - **Short** $T_i$: Checkpoints become too costly
  - **Long** $T_i$: Failures become too costly
Optimal Checkpoint Intervals

• Construct a first order approximation:

\[ T_i \approx \sqrt{2T_c T_{mtbf}} \]

• Example:
  – 64 machines with a per machine MTBF of 1 year
    • \( T_{mtbf} = \frac{1 \text{ year}}{64} \approx 130 \text{ Hours} \)
  – \( T_c = \) of 4 minutes
  – \( T_i \approx \) of 4 hours

From: [http://dl.acm.org/citation.cfm?id=361115](http://dl.acm.org/citation.cfm?id=361115)