Large-Scale Distributed Machine Learning

Carlos Guestrin

Joseph Gonzalez  Yucheng Low  Aapo Kyrola  Haijie Gu  Joseph Bradley  Danny Bickson
Needless to Say, We Need Machine Learning for Big Data

6 Billion Flickr Photos
28 Million Wikipedia Pages
1 Billion Facebook Users
72 Hours a Minute YouTube

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

How will we design and implement parallel learning systems?
Part 1
ASYNCHRONOUS DATA-PARALLEL ALGORITHMS
Sparse Regression

\[ y \approx w_0 + aw_1 + a^2w_2 + a^3w_3 + \ldots \]

\[ = Aw \]

target

basis functions

weights

LASSO: find sparse weight vector \( w^* \)

\[
\min_w F(w) \\
F(w) = \|y - Aw\|^2 + \lambda \|w\|_1
\]

least squares

sparsity inducing regularizer

- Fundamental machine learning task
- Huge number of applications (many thousands of papers)
  - Computational biology, computer vision, compressed sensing...

[Tibshirani, 1996]
Shooting: Stochastic Coordinate Descent (SCD)  

While not converged

- Choose random coordinate $j$
- Optimize $w_j$  
  (closed-form minimization)

Lasso: $\min_{w} F(w)$  
where  
$$F(w) = \frac{1}{2} \| A w - y \|_2^2 + \lambda \| w \|_1$$
Coordinate Descent for LASSO
(aka Shooting Algorithm)

- **Repeat until convergence**
  - **Pick a coordinate** \( j \) at (random or sequentially)
  - **Set:**
    \[
    \hat{w}_\ell = \begin{cases} 
    (c_\ell + \lambda)/a_\ell & c_\ell < -\lambda \\
    0 & c_\ell \in [-\lambda, \lambda] \\
    (c_\ell - \lambda)/a_\ell & c_\ell > \lambda
    \end{cases}
    \]
  - **Where:**
    \[
    a_\ell = 2 \sum_{j=1}^{N} (h_\ell(x_j))^2 \\
    c_\ell = 2 \sum_{j=1}^{N} h_\ell(x_j) \left( t(x_j) - (w_0 + \sum_{i \neq \ell} w_i h_i(x_j)) \right)
    \]
Analysis of SCD [Shalev-Shwartz, Tewari ’09/’11]

- Theorem: With iterations $T$, expected error decreases as:

$$O\left(\frac{d \gamma \|w^*\|^2}{T}\right)$$

- For $d$ dimensions, and optimum $w^*$
- For (coordinate-wise) strongly convex functions ($\Delta w = \delta_{wj} e_j$):
  $$F(w + \Delta w) \leq F(w) + |\Delta w| (\nabla F(w))_j + \frac{\gamma |\Delta w|^2}{2}$$
- For LASSO $\gamma=1$, for Logistic Regression $\gamma=1/4$

Great rate...
but gets expensive in high dimensions
Shotgun: Data-Parallel SCD

[Bradley, Kyrola, Bickson, G. ‘11]

While not converged

- On each of \( P \) processors
  - Choose random coordinate \( j \)
  - Optimize \( w_j \) (as in Shooting)

---

**Is coordinate descent inherently sequential?**

Lasso:

\[
\min_{w} F(w) \quad \text{where} \quad F(w) = \frac{1}{2} \| Aw - y \|_2^2 + \lambda \| w \|_1
\]
Is SCD inherently sequentially?

**Lasso:** \( \min_w F(w) \) where \( F(w) = \| Xw - y \|_2^2 + \lambda \| w \|_1 \)

**Coordinate update:**
\[
\begin{align*}
  w_j & \leftarrow w_j + \delta w_j \\
  \text{(closed-form minimization)}
\end{align*}
\]

**Collective update:**
\[
\Delta w = \begin{pmatrix}
  \delta w_i \\
  0 \\
  0 \\
  0 \\
  \delta w_j \\
  0
\end{pmatrix}
\]
Is SCD inherently sequential?

**Lasso:*** \( \min_w F(w) \) where \( F(w) = \|Xw - y\|_2^2 + \lambda \|w\|_1 \)

**Lemma:** If \( X \) is normalized s.t. \( \text{diag}(X^TX) = 1 \),

\[
F(w + \Delta w) - F(w) \\
\leq - \sum_{i_j \in P} (\delta w_{i_j})^2 + \sum_{i_j, i_k \in P, \ j \neq k} (X^T X)_{i_j, i_k} \delta w_{i_j} \delta w_{i_k}
\]

Can be positive or negative \( \otimes \)

"Positive" progress

**Key term!**
(Measures "correlation" between features...)

"interference" between updates
Theorem: Shotgun Convergence

Assume $P < d / \rho + 1$

where $\rho = \text{largest eigenvalue of } A^T A$

Then: can achieve linear speed ups with up to $P$ processors

Nice case: Uncorrelated features
$\rho = 1 \Rightarrow P_{\text{max}} = d$

Bad case: Correlated features
$\rho = d \Rightarrow P_{\text{max}} = 1$ (at worst)

$E[ F(w^{(T)}) ] - F(w^*) \leq \frac{d \left( \frac{1}{2} \| w^* \|_2^2 + F(w^{(0)}) \right)}{TP}$

final - opt objective

iterations # parallel updates
Experiments Match Theory!

Shotgun outperforms LASSO solvers

(Shooting, SGD, L1_LS Parallel, FPC_AS, Spa_RSA, GPSR_BB, Hard_L0)

2-10x on wide range of real data

Mug32, single pixel camera dataset
Key Proof Technique

Parallel optimization problem

Potential interference between updates

Guarantee based on bounding magnitude of interference
Stepping Back...

- **Stochastic coordinate ascent (SCD)**
  - Optimization: *Pick a coordinate* $j$, find $\arg\min_{w_j} F(w)$
  - Parallel SCD: *Pick* $p$ *coordinates and update at once*
  - **Issue:** Updates may interfere on *$p$* coordinates
  - **Solution:** Bound possible interference using spectral norm

- **Natural counterpart:** Stochastic gradient descent (SGD)
  - Optimization: *Pick a data point and take a small gradient step on all coordinates*
  - Parallel: *Pick* $p$ *data points and update at once*
  - **Issue:** Updates may interfere on *all* coordinates
  - **Solution:** Bound interference using sparsity of data points
Stochastic Gradient Descent

- Coordinate descent updates one coordinate $w_j$, using all data points

- Stochastic gradient descent updates all coordinates, using one data point $x^{(i)}$:

$$w^{(t+1)} \leftarrow w^{(t)} + \eta \nabla F(w; x^{(i)})$$
Parallel Stochastic Gradient Descent

Each processor does update using a different data point

Weight vector \( \mathbf{w} \):

\[
\mathbf{w}(t+1) \leftarrow \mathbf{w}(t) + \eta \nabla F(\mathbf{w}; x_i) \\
\mathbf{w}(t+1) \leftarrow \mathbf{w}(t) + \eta \nabla F(\mathbf{w}; x_j) \\
\mathbf{w}(t+1) \leftarrow \mathbf{w}(t) + \eta \nabla F(\mathbf{w}; x_k) \\
\mathbf{w}(t+1) \leftarrow \mathbf{w}(t) + \eta \nabla F(\mathbf{w}; x_\ell)
\]

Different data points

Risk versus coordinate descent: SGD could interfere on all coordinates simultaneously
Parallel SGD with No Locks

- Each processor in parallel:
  - Pick data point $i$ at random
  - For $j = 1 \ldots d$:

$$w_j^{(t+1)} \leftarrow w_j^{(t)} + \eta \left( \nabla F(w; x^{(i)}) \right)_j$$

- Assume atomicity of sum operation for a coordinate:

$$w_j \leftarrow w_j + \alpha$$

**Key to proof of bounded interference:**
Assume data points are sparse ➔
update interferes at most on a few coordinates
Shared Memory versus Distributed Memory

- **Shared memory**: all machines can access same memory space
  
  Weight vector $w$: 
  
  - Much harder to implement Shotgun or Hogwild!, because of need to synchronize parameters across machines
    - Synchronization can be extremely slow
DHT: Distributed memory that looks like shared memory from the programmer’s perspective

- Easy to program
- Guarantees consistency of values read/written
- Only really efficient when “large” objects are written/read
- In ML, an “object” is a parameter, just a double ➔ standard DHTs are too slow
Parameter Servers \( (\text{e.g., Smola et al.}) \)

- A parameter server is a **Lazy DHT** with **commutative-associative operations**, e.g., \( w_j \leftarrow w_j + \alpha \)

Each machine has a local view of global DHT

Eventually:

Parameter servers only guarantee eventual consistency

But, often good enough for many distributed learning procedures
Summary of Part 1

- Shotgun/Hogwild! solve distributed optimization by ignoring dependencies in problem

- Key proof method: bound interference in updates

- Implement in distributed settings using parameter servers
Part 2
ASYNCHRONOUS GRAPH-PARALLEL ALGORITHMS
DATA PARALLEL versus GRAPH PARALLEL

abstractions
Data Parallelism (MapReduce)

Solve a huge number of independent subproblems
“A white elephant is a valuable but burdensome possession of which its owner cannot dispose and whose cost (particularly cost of upkeep) is out of proportion to its usefulness or worth.” Wikipedia

Everyone knows has limitations, nobody happy, but what to do next???
MapReduce for Data-Parallel ML

Excellent for large data-parallel tasks!

Data-Parallel

MapReduce

Feature Extraction
Cross Validation
Computing Sufficient Statistics

Is there more to Machine Learning?
What is this an image of?
It’s next to this...
The Power of Dependencies

where the value is!
Flashback to 1998

First Google advantage: a Graph Algorithm & a System to Support it!
It’s all about the graphs...
Graphs encode the relationships between:

- People
- Products
- Ideas
- Facts
- Interests

**Big:** 100 billions of vertices and edges and rich metadata

- Facebook (10/2012): 1B users, 144B friendships
- Twitter (2011): 15B follower edges
Examples of Graphs in Machine Learning
Label a Face and Propagate

grandma
Pairwise similarity not enough...
Propagate Similarities & Co-occurrences for Accurate Predictions

Probabilistic Graphical Models

grandma

similarity edges

co-occurring faces further evidence
Collaborative Filtering: Exploiting Dependencies

Latent Factor Models
Non-negative Matrix Factorization

What do I recommend???

Women on the Verge of a Nervous Breakdown
The Celebration
Wild Strawberries
La Dolce Vita
Estimate Political Bias

Semi-Supervised & Transductive Learning
Topic Modeling

LDA and co.
Machine Learning Pipeline

Data
- images
- docs
- movie ratings

Extract Features
- faces
- important words
- side info

Graph Formation
- similar faces
- shared words
- rated movies

Structured Machine Learning Algorithm
- belief propagation
- LDA
- collaborative filtering

Value from Data
- face labels
- doc topics
- movie recommend.
Parallelizing Machine Learning

Data

Extract Features

Graph Formation

Structured Machine Learning Algorithm

Value from Data

Graph Ingress
mostly data-parallel

Graph-Structured Computation
graph-parallel

GraphLab
ML Tasks Beyond Data-Parallelism

Data-Parallel

Map Reduce

Feature Extraction

Cross Validation

Computing Sufficient Statistics

Graph-Parallel

Graphical Models

Gibbs Sampling

Belief Propagation

Variational Opt.

Semi-Supervised Learning

Label Propagation

CoEM

Collaborative Filtering

Tensor Factorization

Graph Analysis

PageRank

Triangle Counting
Example of a Graph-Parallel Algorithm
What’s the rank of this user?

Depends on rank of who follows her

Depends on rank of who follows them...

Loops in graph → Must iterate!
PageRank Iteration

Iterate until convergence:
“My rank is weighted average of my friends’ ranks”

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

- \( \alpha \) is the random reset probability
- \( w_{ji} \) is the prob. transitioning (similarity) from j to i
Properties of Graph Parallel Algorithms

Dependency Graph

Local Updates

Iterative Computation

My Rank

Friends Rank
Addressing Graph-Parallel ML

Data-Parallel

Map Reduce

Feature Extraction
Cross Validation
Computing Sufficient Statistics

Graph-Parallel

Graph-Parallel Abstraction

Graphical Models
Gibbs Sampling
Belief Propagation
Variational Opt.

Semi-Supervised Learning
Label Propagation
CoEM

Collaborative Filtering
Tensor Factorization

Data-Mining
PageRank
Triangle Counting
Graph Computation:

*Synchronous v. Asynchronous*
Bulk Synchronous Parallel Model: Pregel (Giraph)

[Valiant ‘90]
Problem:

*Bulk synchronous parallel systems can be highly inefficient*
BSP Systems Problem: Curse of the Slow Job
Bulk synchronous parallel model provably inefficient for some ML tasks
Analyzing Belief Propagation

[Gonzalez, Low, G. ‘09]

Asynchronous Parallel Model (rather than BSP) fundamental for efficiency
Asynchronous Belief Propagation

Challenge = Boundaries

Synthetic Noisy Image

Cumulative Vertex Updates

Many Updates

Few Updates

Graphical Model

Algorithm identifies and focuses on hidden sequential structure
BSP ML Problem:
Synchronous Algorithms can be **Inefficient**

[Theorem]: Bulk Synchronous BP $O(#\text{vertices})$ slower than Asynchronous BP

Efficient parallel implementation was painful, painful, painful...
The Need for a New Abstraction

Need: Asynchronous, Dynamic Parallel Computations

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

- Graph-Parallel
  - GraphLab
  - Carnegie Mellon
  - 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
The **GraphLab Goals**

Know how to solve ML problem on 1 machine

Efficient parallel predictions
POSSIBILITY
Data Graph

Data associated with vertices and edges

Graph:
• Social Network

Vertex Data:
• User profile text
• Current interests estimates

Edge Data:
• Similarity weights
How do we program graph computation?

“Think like a Vertex.”

-Malewicz et al. [SIGMOD’10]
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
Update Function Example: Connected Components

**Initialize:**
Assign component id to vertex id

**Update(v):**
v.component = min(self & neighbor components)
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
MORE SLIDES ABOUT CONSISTENCY???
The GraphLab Framework

Graph Based
Data Representation

Scheduler

Update Functions
User Computation

Consistency Model
Bayesian Tensor Factorization
Gibbs Sampling
Dynamic Block Gibbs Sampling
CoEM
Belief Propagation
Lasso
LDA
Gibbs Sampling
K-Means
Linear Solvers
SVD
Splash Sampler
Bayesian Tensor Factorization
PageRank
SVM
Matrix Factorization
...Many others...
**Never Ending Learner Project (CoEM)**

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Hadoop</td>
<td>95 Cores</td>
<td>7.5 hrs</td>
</tr>
<tr>
<td>Distributed</td>
<td>32 EC2 machines</td>
<td>80 secs</td>
</tr>
<tr>
<td>GraphLab</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

0.3% of Hadoop time

2 orders of mag faster ➡ 2 orders of mag cheaper
ML algorithms as vertex programs
Asynchronous execution and consistency models
Thus far...

GraphLab 1 provided exciting scaling performance

But...

We couldn’t scale up to Altavista Webgraph 2002
1.4B vertices, 6.7B edges
Natural Graphs
Problem:

Existing *distributed* graph computation systems perform poorly on **Natural Graphs**
**Achilles Heel:** Idealized Graph Assumption

**Assumed...**

Small degree → Easy to partition

**But, Natural Graphs...**

Many high degree vertices (power-law degree distribution) → Very hard to partition
Power-Law Degree Distribution

High-Degree Vertices: 1% vertices adjacent to 50% of edges

AltaVista WebGraph
1.4B Vertices, 6.6B Edges
High Degree Vertices are Common

“Social” People

Popular Movies

Hyper Parameters

Common Words

θ

α

β

Z

W

Docs

LDA

Words

Users

Netflix

Movies

Obama

β
Power-Law Degree Distribution

“Star Like” Motif

President Obama

Followers
Problem:
High Degree Vertices $\implies$ 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
Random Partitioning

- Both GraphLab 1, Pregel, Twitter, Facebook,... rely on Random (hashed) partitioning for Natural Graphs

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

10 Machines → 90% of edges cut
100 Machines → 99% of edges cut!

All data is communicated... Little advantage over MapReduce
In Summary

GraphLab 1 and Pregel are not well suited for natural graphs

- Poor performance on high-degree vertices
- Low Quality Partitioning
SCALABILITY
**Common Pattern for Update Fncs.**

**GraphLab_PageRank(i)**

```plaintext
// 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**
- **Apply Update to Vertex**
- **Scatter Signal to Neighbors & Modify Edge Data**
**GAS Decomposition**

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

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

**Scatter**
Update adjacent edges and vertices.
Many ML Algorithms fit into GAS Model

graph analytics, inference in graphical models, matrix factorization, collaborative filtering, clustering, LDA, ...
Discovering **Influencers** in Social Networks

Triangles measure both “popularity” of vertex & “cohesiveness” of vertex’s community.

- **High degree** with **Fewer Triangles** indicates a **Weaker Community**.
- **Many triangles** suggest a **Stronger Community**.

![Diagram](image)
Gather/Apply/Scatter Triangle Counting

Gather:

My neighbors are:

Apply: Store this list

Scatter:

I’m neighbors with:

Triangle!!

I’m neighbors with:
## Triangle Counting on Twitter (2010)

### Popular People

<table>
<thead>
<tr>
<th>Degree</th>
<th>Name</th>
<th>Followers</th>
</tr>
</thead>
<tbody>
<tr>
<td>3081108</td>
<td>Britney Spears</td>
<td></td>
</tr>
<tr>
<td>2997653</td>
<td>Ashton Kutcher</td>
<td></td>
</tr>
<tr>
<td>2679666</td>
<td>Ellen DeGeneres</td>
<td></td>
</tr>
<tr>
<td>2653045</td>
<td>Barack Obama</td>
<td></td>
</tr>
<tr>
<td>2450768</td>
<td>CNN Breaking News</td>
<td></td>
</tr>
<tr>
<td>1994945</td>
<td>Oprah Winfrey</td>
<td></td>
</tr>
<tr>
<td>1959765</td>
<td>Twitter</td>
<td></td>
</tr>
<tr>
<td>1885917</td>
<td>Ryan Seacrest</td>
<td></td>
</tr>
<tr>
<td>1844123</td>
<td>SHAQ</td>
<td></td>
</tr>
</tbody>
</table>

### Popular People With Strong Communities

<table>
<thead>
<tr>
<th>Triangles / Following</th>
<th>Name</th>
<th>Followers</th>
</tr>
</thead>
<tbody>
<tr>
<td>6449985</td>
<td>Women's Wear Daily</td>
<td></td>
</tr>
<tr>
<td>5.962666e+06</td>
<td>wefollow</td>
<td></td>
</tr>
<tr>
<td>5607368</td>
<td>Stephen Colbert</td>
<td></td>
</tr>
<tr>
<td>5272692</td>
<td>Jonas Brothers</td>
<td></td>
</tr>
<tr>
<td>4483823</td>
<td>Rev Run</td>
<td></td>
</tr>
<tr>
<td>3564740</td>
<td>Defamer.com</td>
<td></td>
</tr>
<tr>
<td>3207562</td>
<td>You Look Great</td>
<td></td>
</tr>
<tr>
<td>2.936561e+06</td>
<td>Oprah Winfrey</td>
<td></td>
</tr>
<tr>
<td>2.488950e+06</td>
<td>Al Gore</td>
<td></td>
</tr>
<tr>
<td>2.474015e+06</td>
<td>CNN Breaking News</td>
<td></td>
</tr>
</tbody>
</table>
Factorized Belief Propagation

- **Gather**: Accumulates product of *in messages*

- **Apply**: Updates central belief

- **Scatter**: Computes out messages & schedules neighbors as needed
Collaborative Filtering (via Alternating Least Squares)

Goal: discover latent categories for users & movies
Factorized Collaborative Filtering Updates

Apply:
Compute user’s new factor weights

Gather:
sum over movies, product of ratings & factor weights
(and a little more info)

Iterate over users & movies
Distributed Execution of a GL2 PowerGraph Vertex-Program

Gather

Apply

Scatter

Machine 1

Machine 2

Machine 3

Machine 4
Minimizing Communication in GL2 PowerGraph: Vertex Cuts

GL2 PowerGraph includes novel vertex cut algorithms

Provides order of magnitude gains in performance

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!
PageRank on the Twitter Follower Graph
Natural Graph with 41M Users, 1.4 Billion Links

Communication

- GraphLab 1
- Pregel
- PowerGraph

Running time

- GraphLab 1
- Pregel
- PowerGraph

32 Nodes x 8 Cores (EC2 HPC cc1.4x)
From the Abstraction to a System
GraphLab Version 2.1 API (C++)

Sync. Engine: Fault Tolerance
Async. Engine

Map/Reduce
Ingress

Distributed Graph

MPI/TCP-IP Comms
PThreads
Boost
HDFS

Linux Cluster Services (Amazon AWS)

Graph Analytics
Graphical Models
Computer Vision
Clustering
Topic Modeling
Collaborative Filtering

Graph Analyticx
Graphical Models
Computer Vision
Clustering
Topic Modeling
Collaborative Filtering

GraphLab Version 2.1 API (C++)

Sync. Engine: Fault Tolerance
Async. Engine

Map/Reduce
Ingress

Distributed Graph

MPI/TCP-IP Comms
PThreads
Boost
HDFS

Linux Cluster Services (Amazon AWS)
Triangle Counting on Twitter Graph
34.8 Billion Triangles

Hadoop [WWW’11]
1636 Machines
423 Minutes

GL2
64 Machines
15 Seconds

PowerGraph

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
Topic Modeling (LDA)

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

Smola et al.

Specifically engineered for this task

64 cc2.8xlarge EC2 Nodes

200 lines of code & 4 human hours
How well does GraphLab scale?

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

1.4B Webpages, 6.7 Billion Links

7 seconds per iter.

1B links processed per second

30 lines of user code

1024 Cores (2048 HT)  4.4 TB RAM
**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

Challenge:
Random Accesses

Novel GraphChi solution:
Parallel sliding windows method →
minimizes number of random accesses
Triangle Counting on Twitter Graph

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

**Hadoop**
- 1636 Machines
- 423 Minutes

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

**GraphLab2**
- 64 Machines, 1024 Cores
- 15 Seconds

Hadoop results from [Suri & Vassilvitskii '11]
ML algorithms as vertex programs
Asynchronous execution and consistency models

Natural graphs change the nature of computation
Vertex cuts and gather/apply/scatter model
GL2 PowerGraph focused on Scalability at the loss of Usability
PageRank(i, scope) {
  acc = 0
  for (j in InNeighbors) {
    acc += pr[j] * edge[j].weight
  }
  pr[i] = 0.15 + 0.85 * acc
}

Explicitly described operations

Code is intuitive
GraphLab 1

PageRank(i, scope){
    acc = 0
    for (j in InNeighbors) {
        acc += pr[j] * edge[j].weight
    }
    pr[i] = 0.15 + 0.85 * acc
}

Code is intuitive

GL2 PowerGraph

Implicit operation

gather(edge) {
    return edge.source.value * edge.weight
}

merge(acc1, acc2) {
    return accum1 + accum2
}

apply(v, accum) {
    v.pr = 0.15 + 0.85 * acc
}

Implicit aggregation

Need to understand engine to understand code
Great flexibility, but hit scalability wall

Scalability, but very rigid abstraction
(many contortions needed to implement SVD++, Restricted Boltzmann Machines)

What now?
GraphLab
WarpGraph

USABILITY

In a realm all its own... Cadillac
GL3 WarpGraph Goals

Program
Like GraphLab 1

Run Like
GraphLab 2

Machine 1

Machine 2
Fine-Grained Primitives

Expose Neighborhood Operations through Parallelizable Iterators

\[
R[i] = 0.15 + 0.85 \sum_{(j,i) \in E} w[j, i] \times R[j]
\]

PageRankUpdateFunction(Y) {
  Y.pagerank = 0.15 + 0.85 * 
  
}
Expressive, Extensible Neighborhood API

- **MapReduce over Neighbors**
  - Parallel Sum
  - \( Y + Y + \ldots + Y \)

- **Parallel Transform Adjacent Edges**
  - Modify adjacent edges

- **Broadcast**
  - Schedule a selected subset of adjacent vertices

- **DHT Get Keys**
- **DHT Update Keys**
Can express every GL2 PowerGraph program (more easily) in GL3 WarpGraph

But GL3 is more expressive

- Multiple gathers
- Scatter before gather
- Conditional execution

UpdateFunction(v) {
    if (v.data == 1)
        accum = MapReduceNeighs(g, m)
    else ...
}


GL2 PowerGraph:

Fast because **communication** phases are very **predictable**

GL3 WarpGraph:

**Communication** highly **unpredictable**

**Risk:** **High Latency**

(spend all our time waiting for a reply...)

... repeat
Hide Latency

Do Something Else while Waiting
Create 1000s of threads, each running an update function on a different vertex

Performance Bottleneck: Context Switching

Every cycle used in context switching is wasted
(OS context switch is slow requiring 10K-100k cycles)

GL3 WarpGraph: Novel user-mode threading

8M context switches per second
100x faster than OS
PageRank Twitter Graph: 41M Vertices 1.4B Edges

WarpGraph only 25% slower, with much improved programmability
But, here, asynchrony not fundamental for performance

32 Nodes x 16 Cores (EC2 HPC cc2.8x)
Graph Coloring

Twitter Graph: 41M Vertices 1.4B Edges

Asynchrony fundamental here ➔ WarpGraph outperforms PowerGraph with simpler code

GL2 PowerGraph

227 seconds

GL3 WarpGraph

89 seconds

2.5x Faster

32 Nodes x 16 Cores (EC2 HPC cc2.8x)
Usability

Consensus that WarpGraph is much easier to use than PowerGraph

User study size = 2 :-)  

Bigger + Real User Study in Progress,  
as we release new open-source version of GraphLab
New abstraction simplifies writing programs in GraphLab

But you still need to get a cluster, install GraphLab, configure system...
**Launch GraphLab Cluster**

Name: 
Node Configuration | Number: 1 | Type: m2.4xlarge

Options | SSL Certificate

Launch
## Clusters

<table>
<thead>
<tr>
<th>Name</th>
<th>Connect URL</th>
<th>Configuration</th>
<th>Status</th>
<th>Last</th>
</tr>
</thead>
<tbody>
<tr>
<td>demo</td>
<td><a href="https://go.graphlab.com/user/demo/learn">https://go.graphlab.com/user/demo/learn</a></td>
<td>1 Node (m2.4xlarge)</td>
<td>Running</td>
<td>today</td>
</tr>
</tbody>
</table>
```python
In [2]: import GraphLab
gl = GraphLab.UndirectedTriangleCount()
file = '/home/graphlab/python-demo/1M.tsv'
(seconds, triangles) = gl.execute(input_file = file)
print "Finding %d undirected triangles in '%s' took %f seconds." % (triangles, file, seconds)

Finding 329024 undirected triangles in '/home/graphlab/python-demo/1M.tsv' took 2.439280 seconds.
```
- ML algorithms as vertex programs
- Asynchronous execution and consistency models

- Natural graphs change the nature of computation
- Vertex cuts and gather/apply/scatter model

- Usability is key
- Access neighborhood through parallelizable iterators and latency hiding
Usability for Whom???

GL2
PowerGraph

GL3
WarpGraph

...
Machine Learning

PHASE 3

USABILITY

In a realm all its own... Cadillac
Exciting Time to Work in ML

With Big Data, I’ll take over the world!!!

We met because of Big Data

Why won’t Big Data read my mind???

Unique opportunities to change the world!! 😊

But, every deployed system is an one-off solution, and requires PhDs to make work... 😞
And, Usability for ML is not just “Engineering” – Must Be Easy to Iterate through Models to Solve Task

But, when ML doesn’t work, need a PhD to understand why...

One kind of ML usability:
Fast and easy iterations over huge datasets

Interpretable feature engineering? No parameters to tune, please...

Why was this prediction made? How can I give valuable feedback?
GraphLab 2.2 available now: graphlab.com