EMERGING SYSTEMS FOR LARGE-SCALE MACHINE LEARNING

Joseph E. Gonzalez
Postdoc, UC Berkeley AMPLab
Co-founder, GraphLab Inc.
Ph.D. 2012, CMU
jegonzal@eecs.berkeley.edu

Slides (draft): http://tinyurl.com/icml14-sysml

ICML’14 Tutorial
EVERY MINUTE OF THE DAY

- Pinterest: Users pin 3,472 images.
- Vine: Users share 8,333 videos.
- Skype: Users connect for 23,300 hours.
- Yelp: Users post 26,380 reviews.
- Apple: Users download 48,000 apps.
- Pandora: Users listen to 61,141 hours of music.
- Amazon: Makes $83,000 in online sales.
- Instagram: Users post 216,000 new photos.
- Twitter: Users tweet 277,000 times.
- Tinder: Users swipe 416,667 times.
- WhatsApp: Users share 347,222 photos.
- Facebook: Users share 2,460,000 pieces of content.
- Google: Receives over 4,000,000 search queries.
- Email: Users send 204,000,000 messages.
- YouTube: Users upload 72 hours of new video.

EVERY MINUTE OF THE DAY

- Pinterest users pin 3,472 images.
- Vine users upload 8,333 videos.
- Skype users connect for 23,300 hours.
- Yelp users post 26,380 reviews.
- Apple users download 48,000 apps.
- Pandora users listen to 61,141 hours of music.
- Amazon makes $83,000 in online sales.
- Instagram users post 216,000 new photos.
- Facebook users share 2,460,000 pieces of content.
- Tinder users swipe 416,667 times.
- WhatsApp users share 347,222 photos.
- Twitter users tweet 277,000 times.

Email users send 204,000,000 messages.

Google receives over 4,000,000 search queries.

EVERY MINUTE OF THE DAY

- Pinterest users pin 3,472 images.
- Vine users share 8,333 videos.
- Skype users connect for 23,300 hours.
- Yelp users post 26,380 reviews.
- Apple users download 48,000 apps.
- Pandora users listen to 61,141 hours of music.
- Email users send 204,000,000 messages.
- Google receives over 4,000,000 search queries.
- Facebook users share 2,460,000 pieces of content.
- Tinder users swipe 416,667 times.
- WhatsApp users share 347,222 photos.
- Twitter users tweet 277,000 times.
- Instagram users post $83,000 in online sales.
- Amazon users make 216,000 new photos.

Every Minute of the Day:

- **YouTube Users Upload**: 72 hours of new video.
- **Email Users Send**: 204,000,000 messages.
- **Google Receives Over**: 4,000,000 search queries.
- **Facebook Users Share**: 2,460,000 pieces of content.
- **Skype Users Connect For**: 23,300 hours.
- **Yelp Users Post**: 26,380 reviews.
- **Twitter Users Tweet**: 277,000 times.
- **WhatsApp Users Share**: 347,222 photos.
- **Tinder Users Swipe**: 416,667 times.
- **Pinterest Users Pin**: 3,472 images.
- **Vine Users Share**: 8,333 videos.
- **Yelp Users Post**: 26,380 reviews.
- **Apple Users Download**: 48,000 apps.
- **Pandora Users Listen To**: 61,141 hours of music.
- **Amazon Makes**: $83,000 in online sales.
- **Instagram Users Post**: 216,000 new photos.

My story …

Machine Learning  Learning Systems
As a young graduate student
As a young graduate student I worked on parallel algorithms for inference in graphical models:

I designed and implemented parallel learning algorithms on top of low level primitives …
Advantages of the Low-Level Approach

Extract *maximum performance* from hardware

Enable exploration of more complex algorithms

- Fine grained locking
- Atomic data-structures
- Distributed coordination protocols

*My implementation is better than your implementation.*
Limitations of the Low-Level Approach

Repeatedly address the same system challenges

Algorithm conflates learning and system logic

Difficult to debug and extend

Typically does not address issues at scale: hardware failure, stragglers, …

Months of tuning and engineering for one problem.
Design Complexity

Model

- Training
- Accuracy
- Machine Learning

Parallelism

- Large-Scale Systems
- Locality
- Scheduling
- Network
- Stragglers
- Fault Tolerance
Design Complexity

Model

Interaction

Machine Learning

Training

Accuracy

Parallelism

Network

Locality

Stragglers

Scheduling

Large-Scale Systems

Fault Tolerance
Learning systems combine the complexities of machine learning with system design.
Application
Idea

Serial
Prototype

Parallel
Prototype

Debug
Optimize
Evaluate

ML
Paper

Interesting
Machine Learning
Research

Distributed
Prototype

Abstraction
Interesting
Systems Research

Interesting
Research

Model +
Algorithm
Black Box  Learning Systems
Black Box Learning Algorithm

Common Patterns

System Abstraction (API)

1. Parallelism
2. Data Locality
3. Network
4. Scheduling
5. Fault-tolerance
6. Stragglers
Managing Complexity Through Abstraction

Learning Algorithm Common Patterns

Abstraction (API)

System
1. Parallelism
2. Data Locality
3. Network
4. Scheduling
5. Fault Tolerance
6. Stragglers

Identify common patterns

Define a narrow interface

Exploit limited abstraction to address system design challenges
The GraphLab project allowed us to:

- Separate algorithm and system design
- Optimize system for many applications at once
- Accelerate research in large-scale ML
Outline of the Tutorial

1. Distributed Aggregation: Map-Reduce
   Data Parallel
2. Iterative Machine Learning: Spark
3. Large Shared Models: Parameter Server
4. Graphical Computation: GraphLab to GraphX
What is not covered

Linear Algebra Patterns: BLAS/ScaLAPACK

• core of high-performance computing
• communication avoiding & randomized algorithms
• Joel Tropp Tutorial (right now)

GPU Accelerated Systems

• converging to BLAS patterns

Probabilistic Programming

• See tutorial 5
Elephant in the Room
Map-Reduce
Aggregation Queries

Common Pattern

\[ \tilde{E}_D [f(X)] = \frac{1}{n} \sum_{i=1}^{n} f(x_i) \]

Abstraction: Map, Reduce

System

Hadoop
Learning from Aggregation Statistics

- Chu et al., Map-Reduce for Machine Learning on Multicore. NIPS’06.
Learning from Aggregation Statistics

Query Function:

\[ f : \mathcal{X} \rightarrow \mathbb{R}^d \]

System Executes:

\[ \tilde{E}_D [f(X)] = \frac{1}{n} \sum_{i=1}^{n} f(x_i) \]

on data \( D = \{x_1, \ldots, x_n\} \)
Example Statistics

Sufficient Statistics (e.g., $E[X]$, $E[X^2]$): 

$$\frac{1}{n} \sum_{i=1}^{n} x_i$$

Empirical loss: 

$$\frac{1}{n} \sum_{i=1}^{n} l(y, h(x))$$

Gradient of the loss: 

$$\frac{1}{n} \sum_{i=1}^{n} \nabla_w l(y, h_w(x)) \bigg|_{w=w(t)}$$
Map-Reduce Abstraction

[Dean & Ghemawat, OSDI’04]

```java
Map(docRecord) {
  for (word in docRecord) {
    emit (word, 1)
  }
}

Reduce(word, counts) {
  emit (word, SUM(counts))
}
```

Example: Word-Count

Map: Idempotent
Reduce: Commutative and Associative

[Dean & Ghemawat, OSDI’04]
Map-Reduce System

[Dean & Ghemawat, OSDI'04]
Map-Reduce System

[Dean & Ghemawat, OSDI'04]
Map-Reduce Fault-Recovery

[Dean & Ghemawat, OSDI'04]
Map-Reduce Fault-Recovery

[Dean & Ghemawat, OSDI'04]
Distributed File Systems

[Ghemawat et al., SOSP'03]
Distributed File Systems

[Ghemawat et al., SOSP'03]
What functionality can we remove?

Learning algorithm cannot directly access data.

- Restrict computation to:

  \[ \mathbb{E}_D [f(X)] = \frac{1}{n} \sum_{i=1}^{n} f(x_i) \]

System controls interaction with data:

- Distribute computation and data access
- Fault tolerance & straggler mitigation
Example:

Least Squares Regression
Least-Squares Regression with Aggregation Statistics

Objective:

\[ \hat{\theta}_{\text{MLE}} = \arg\min_{\theta \in \mathbb{R}^d} \sum_{i=1}^{n} (y_i - \theta^T x_i)^2 \]

Solution (Normal Equations):

\[ \hat{\theta}_{\text{MLE}} = (X^T X)^{-1} (X^T Y) \]
Deriving the Aggregation Stats.

\[ \hat{\theta}_{\text{MLE}} = \left( X^T X \right)^{-1} X^T Y \]

Aggregation Statistics:

\[ \hat{\theta}_{\text{MLE}} = \left( \sum_{i=1}^{n} x_i x_i^T \right)^{-1} \left( \sum_{i=1}^{n} x_i y_i \right) \]

Map( (x,y) record ) {
    emit ("xx", x * Trans(x))
    emit ("xy", x * y)
}

Reduce(key, mats) {
    emit (key, SUM(mats))
}

\[ O(nd^2) \]

#mappers
Deriving the Aggregation Stats.

\[ \hat{\theta}_{\text{MLE}} = \left( X^T X \right)^{-1} X^T Y \]

Aggregation Statistics:

\[ \hat{\theta}_{\text{MLE}} = \left( \sum_{i=1}^{n} x_i x_i^T \right)^{-1} \left( \sum_{i=1}^{n} x_i y_i \right) \]

Solve linear system on the master:

\[ \hat{\theta}_{\text{MLE}} = \begin{pmatrix} d \\ d \end{pmatrix}^{-1} \begin{pmatrix} d \\ 1 \end{pmatrix} = d \quad \text{Inversion doesn't depend on } n \]

\[ O(nd^2) \quad \frac{\text{#mappers}}{O(d^3)} \]
Apache Mahout

Open-Source Library of Algorithms on Hadoop

ALS Matrix Fact.  Naïve Bayes
SVD  PCA
Random Forests  Spectral Clustering
LDA  Canopy Clustering
K-Means  Logistic Regression?
Limitations?
Map-Reduce

hadoop
Why not Logistic Regression?
Logistic Regression

Iterative batch gradient descent.
Logistic Regression in Map-Reduce

Gradient descent:

\[ f_w(x, y) = \nabla \log L(y, h_w(x)) \]

Learning Algorithm

Update Model:

\[ w \leftarrow w - \eta_t g \]

Query: \( f_w \)

\[ g = \frac{1}{n} \sum_{i=1}^{n} f_w(x_i, y_i) \]

System

Data

Hadoop
Map-Reduce is not optimized for iteration and multi-stage computation.

Logistic Regression in Map-Reduce

Gradient descent:

\[ f_w(x, y) = \log L(y, h_w(x)) \]

Update Model:

\[ w \leftarrow w - \eta_t g \]

Query:

\[ g = \frac{1}{n} \sum_{i=1}^{n} f_w(x_i, y_i) \]
Iteration in Map-Reduce

Initial Model

Training Data

Map

Reduce

Learned Model

$w^{(0)}$

$w^{(1)}$

$w^{(2)}$

$w^{(3)}$

$w^{(4)}$

$w^{(5)}$

$h^{(0)}$

$h^{(1)}$

$h^{(2)}$

$h^{(3)}$
Cost of Iteration in Map-Reduce

- Initial Model: $w^{(0)}$
- Read 1
- Map
- Reduce
- Learned Model: $w^{(1)}$
- Read 2
- Repeatedly load same data
- Read 3
- $w^{(2)}$
- $w^{(3)}$
Cost of Iteration in Map-Reduce

Redundantly save output between stages
Iteration and Multi-stage computation

In-Memory Dataflow System


Dataflow View

Training Data (HDFS)
Memory Opt. Dataflow

Cached Load

Training Data (HDFS)

Map → Reduce

Map → Reduce

Map → Reduce

10-100× faster than network and disk
Memory Opt. Dataflow View

Training Data (HDFS)

Map → Reduce
Map → Reduce
Map → Reduce

Efficiently move data between stages
In-Memory Data-Flow Systems

Common Pattern: Multi-Stage Aggregation

Abstraction: Dataflow Ops. on Immutable datasets

System Spark
What is Spark?

Fault-tolerant distributed dataflow framework

Improves efficiency through:
  » In-memory computing primitives
  » Pipelined computation

Improves usability through:
  » Rich APIs in Scala, Java, Python
  » Interactive shell

Up to 100× faster (2-10× on disk)

2-5× less code
Spark Programming Abstraction

Write programs in terms of transformations on distributed datasets

Resilient Distributed Datasets (RDDs)

» Distributed collections of objects that can stored in memory or on disk
» Built via parallel transformations (map, filter, …)
» Automatically rebuilt on failure
Example: Log Mining

Load error messages from a log into memory, then interactivity search for various patterns.

```python
lines = spark.textFile("hdfs://log")
errors = lines.filter(x => x.startsWith("ERROR"))
msgs = errors.map(x => x.split(\'\t\')(2)).cache()

msgs.filter(x => x.contains("foo")).count
msgs.filter(x => x.contains("bar")).count

Result: scaled to 1 TB data in 5-7 sec (vs 170 sec for on-disk data)
```
Fault Tolerance

RDDS track *lineage* info to rebuild lost data

```python
file.map(lambda rec: (rec.type, 1))
  .reduceByKey(lambda x, y: x + y)
  .filter(lambda (type, count): count > 10)
```
Fault Tolerance

RDDs track lineage info to rebuild lost data

```python
file.map(lambda rec: (rec.type, 1)).reduceByKey(lambda x, y: x + y).filter(lambda (type, count): count > 10)
```
Abstraction: *Dataflow Operators*

- map
- filter
- groupBy
- sort
- union
- join
- leftOuterJoin
- rightOuterJoin

- reduce
- count
- fold
- reduceByKey
- groupByKey
- cogroup
- cross
- zip

- sample
- take
- first
- partitionBy
- mapWith
- pipe
- save
- ...

58
Batch Gradient Logistic Regression

```scala
val data = spark.textFile("hdfs://data")
  .map(readPoint).cache()

var w = Vector.random(D)

for (i <- 1 to ITERATIONS) {
  val gradient = data.map(p =>
    (1 / (1 + exp(-p.y*(w dot p.x))) - 1) * p.y * p.x
  ).reduce(_ + _)
  w -= gradient
}

println("Final w: " + w)
```

Load data in memory once

Initial parameter vector

Repeated Map-Reduce steps for gradient descent

Slide provided by M. Zaharia
Logistic Regression Performance

- **Hadoop**: 110 s / iteration
- **Spark**: first iteration 80 s
  - further iterations 1 s

29 GB dataset on 20 EC2 m1.xlarge machines (4 cores each)

Slide provided by M. Zaharia
MLlib: high quality library for ML algorithms
  » Included in Apache Spark
MLbase: make ML accessible to non-experts
  » Automatically pick best algorithm
  » Allow developers to easily add and test new algorithms

E. Sparks, A. Talwalkar, V. Smith, J. Kottalam, X. Pan, J. Gonzalez, Michael Franklin, Michael Jordan, Tim Kraska. ML: An API for Distributed Machine Learning. ICDM’13

Mahout Moves to Spark

On 25 April 2014 - Goodbye MapReduce

The Mahout community decided to move its codebase onto modern data processing systems that offer a richer programming model and more efficient execution than Hadoop MapReduce. Mahout will therefore reject new MapReduce algorithm implementations from now on.

We are building our future implementations on top of a DSL for linear algebraic operations which has been developed over the last months. Programs written in this DSL are automatically optimized and executed in parallel on Apache Spark.
Other Related Systems

Microsoft Dryad and Naiad:

Hyracks: http://hyracks.org

Stratosphere: http://stratosphere.eu

MADlib: http://madlib.net

Hadoop Tez: http://hortonworks.com/hadoop/tez/

See publication lists at each of the sites
Outline of the Tutorial

1. Distributed Aggregation: **Map-Reduce**
2. Iterative Machine Learning: **Spark**
3. Large Shared Models: **Parameter Server**
4. Graphical Computation: **GraphLab** to **GraphX**
Big Models and Online Algorithms

Parameter Servers

A. Smola and S. Narayanamurthy. An architecture for parallel topic models. VLDB’10

Small $\rightarrow$ Big Models

$\begin{align*}
\text{Sparse Big Data} & \quad \text{Big Model} \\
\mathbf{n} & \approx \mathbf{d} \approx \mathbf{n}
\end{align*}$
Examples

Spam prediction using bi-grams:
• Weight vector in $(\#\text{Words})^2$

Deep Learning:
• Billions of model parameters

Topic Modeling (LDA):
• Distribution over words for each topic
Common Pattern

Latent Var. Models

Global Variables

$\mu_j$

$j \in \{1, \ldots, K\}$

Local Variables

$x_i$

$z_i$

$i \in \{1, \ldots, N\}$
Challenge of Big Models

Example (Gradient Descent):

Broadcast and store a copy of the model each iteration
Challenge of Big Models
Challenge of Big Models

Sparse Changes to Model
Challenge of Big Models
Online Algorithms

Example: Stochastic Gradient Descent

\[ \text{Model} \leftarrow \text{Model} \oplus f(x_i, \text{Model}) \]

Sparse updates:

1. Comp. depends on a small part of model:

\[ \delta_i \leftarrow f(x_i, \text{Model}) \]

2. Sparse additive model update:
Bulk Synchronous Execution

Machine 1

W

W^1

W^2

W^3

Machine 2

W

W^1

W^2

W

Machine 3

W

W^1

W^2

W

Compute

Waste

Iteration

Compute

Communicate

Iteration

Waste

Iteration

Waste

Iteration

Waste

Barrier

Barrier
Asynchronous Execution

Enable more frequent coordination on parameter values
Asynchronous Execution

Parameter Server (Logical)

Machine 1

Machine 2

Machine 3
Parameter Server Abstraction

Key-Value API with two basic operations:

1. get(key) \rightarrow value

\[ \delta_i \leftarrow f(x_i, \text{Model}) \]

2. add(key, delta)

\[ \text{Model} \leftarrow \text{Model} \oplus \delta_i \]
Split Model Across Machines

Parameter Server (Logical)

W_1 W_2 W_3 W_4 W_5 W_6 W_7 W_8 W_9

Machine 1

Iteration Iteration Iteration Iteration

Machine 2

Iteration Iteration Iteration

Machine 3

Iteration Iteration Iteration Iteration Iteration Iteration
Split Model Across Machines

Parameter Server

$W_1 \quad W_2 \quad W_3$

Parameter Server

$W_4 \quad W_5 \quad W_6$

Parameter Server

$W_7 \quad W_8 \quad W_9$

Split Data Across Machines

$W_9$

Data

$W_6$

Data

$W_8$

Data

$W_7$

Data
Example: Topic Modeling with LDA

Word Dist. by Topic

\( \beta_t \)

t \in \{1, \ldots, T\}

Maintained by the Parameter Server

Local Variables Documents

Tokens

\( x_i \)

\( z_i \)

\( \theta_d \)

\( i \in \{1, \ldots, \text{Len}(d)\} \)

\( d \in \{1, \ldots, D\} \)

Maintained by the Workers Nodes
Gibbs Sampling for LDA

Title: *Oh, The Places You’ll Go!*

You have brains in your head.

You have feet in your shoes.

You can steer yourself any direction you choose.
Gibbs Sampling for LDA

Dictionary

Brains:
Choose:
Direction:
Feet:
Head:
Shoes:
Steer:

Document Model $\theta_d$

Title: Oh, The Places You’ll Go!

You have brains in your head.
You have feet in your shoes.
You can steer yourself any direction you choose.
Ex: Collapsed Gibbs Sampler for LDA

Partitioning the model and data

Parameter Server

\[ W_{1:10K} \]

Parameter Server

\[ W_{10k:20K} \]

Parameter Server

\[ W_{20k:30K} \]
Ex: Collapsed Gibbs Sampler for LDA

Get model parameters and compute update

Parameter Server

Parameter Server

Parameter Server

get("car")

get("cat")

get("tire")

get("mouse")

Ex: Collapsed Gibbs Sampler for LDA

Get model parameters and compute update

Parameter Server

Parameter Server

Parameter Server

get("car")

get("cat")

get("tire")

get("mouse")

Ex: Collapsed Gibbs Sampler for LDA

Get model parameters and compute update

Parameter Server

Parameter Server

Parameter Server

get("car")

get("cat")

get("tire")

get("mouse")

Ex: Collapsed Gibbs Sampler for LDA

Get model parameters and compute update

Parameter Server

Parameter Server

Parameter Server

get("car")

get("cat")

get("tire")

get("mouse")
Ex: Collapsed Gibbs Sampler for LDA

Send changes back to the parameter server
Ex: Collapsed Gibbs Sampler for LDA

Adding a caching layer to collect updates
**Ex: Collapsed Gibbs Sampler for LDA**

Inconsistent model replicas

Inconsistent Values

"Model Replicas"

[Dean et al., “Large Scale Distributed Deep Networks” NIPS'12]
Bounding Staleness

Ho et al. “More Effective Distributed ML via a Stale Synchronous Parallel Parameter Server.” NIPS’13

Slow-down fast workers

Force periodic cache synchronization

Machine 1

- Tick
- Tick
- Tick
- Tick

Staleness of 3

Barrier

Fast

Machine 2

- Tick

Slow

Machine 3

- Tick
- Tick
- Tick
Bounding Staleness

Ho et al. “More Effective Distributed ML via a Stale Synchronous Parallel Parameter Server.” NIPS’13

Slow-down fast workers

Force periodic cache synchronization

Currently the analysis only:

• applies to convex functions

• characterizes the average objective value

Opportunity for more research.
Fault Tolerance

M. Li et al. *Parameter Server for Distributed Machine Learning, Big Learning Workshop, NIPS’13*

Consistent Hashing:

```
get("cat")
```
Parameter Server
Implementations

ParameterServer.org: Alex Smola’s Lab

- C++, Apache License
- Code: https://github.com/mli/parameter_server

Petuum.org: Eric Xing’s Lab

- C++, BSD License
- Code: https://github.com/sailinglab/petuum
Applications of Parameter Servers

**Distributed Gibbs Sampling:** A. Ahmed et al., *Scalable inference in latent variable models.* WSDM '12

**Deep Learning:** Dean et al., *Large Scale Distributed Deep Networks.* NIPS’12

**Matrix Factorization:** Ho et al. *More Effective Distributed ML via a Stale Synchronous Parallel Parameter Server.* NIPS’13
Specialization for SGD

Vowpal Wabbit: http://hunch.net/~vw/

• Primary use is fast online linear optimization
• Distributed optimization tools

Bismark: X. Feng, A. Kumar, B. Recht, and C. Ré. Towards a unified architecture for in-RDBMS analytics. SIGMOD’12

• In database incremental gradient descent
Limitations of the Parameter Server

Does not address the data/worker management:

- Data-partitioning, recovery, stragglers
- Opportunity: *Dataflow Integration* (e.g., *Spark*)

Asynchronous model is complicated to debug

Does not capture static dependency structure between data and parameters
Static Data Dependencies

Words

Apple → Cat → Dog → The

Documents
Outline of the Tutorial

1. Distributed Aggregation: Map-Reduce

2. Iterative Machine Learning: Spark

3. Large Shared Models: Parameter Server

4. Graphical Computation: GraphLab to GraphX
Graph-Structured Big Models

R. Xin, J. Gonzalez, M. Franklin, I. Stoica., *GraphX: A Resilient Distributed Graph System on Spark.* SIGMOD GRADES'13

J. Gonzalez, Y. Low, H. Gu, D. Bickson, C. Guestrin. *PowerGraph: Distributed Graph-Parallel Computation on Natural Graphs.* OSDI'12
Graphs are Central to Analytics

- Raw Wikipedia
- XML
- Table
- Text
- Title Body
- Hyperlinks
- Term-Doc Graph
- PageRank
- Top 20 Pages
  - Title
  - PR
- Topic Model (LDA)
- Word Topics
  - Word
  - Topic
- Editor Graph
- Community Detection
- User Community
- Community Topic
- User Com.
- Topic Com.
- Discussion Table
- User Disc.
PageRank: Identifying Leaders

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

Update ranks in parallel
Iterate until convergence
Mean Field Algorithm

\[ \phi_2(x_2) \]

\[ \phi_1(x_1) \]

\[ \phi_3(x_3) \]

\[ b_i(x_i) \propto \phi_i(x_i) \exp \left( \sum_{j \in N_i} f(x_i, b_j) \right) \]

\[ f(x_i, b_j) = \sum_{x_j} b_j(x_j) \log \phi_{i,j}(x_i, x_j) \]
Recommending Products

Low-Rank Matrix Factorization:

\[ f[i] = \arg \min_{w \in \mathbb{R}^d} \sum_{j \in \text{Nbrs}(i)} (r_{ij} - w^T f[j])^2 + \lambda \|w\|_2^2 \]

Iterate:
The Graph-Parallel Pattern

Model / Alg. State

Computation depends only on the neighbors
Many Graph-Parallel Algorithms

- Collaborative Filtering
  - Alternating Least Squares
  - Stochastic Gradient Descent
  - Tensor Factorization

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

- Semi-supervised ML
  - Graph SSL
  - CoEM

- Community Detection
  - Triangle Counting
  - K-core Decomposition
  - K-Truss

- Graph Analytics
  - PageRank
  - Personalized PageRank
  - Shortest Path
  - Graph Coloring

- Classification
  - Neural Networks
Graph-Parallel Systems

Expose *specialized APIs* to simplify graph programming.
“Think like a Vertex.”
- Pregel [SIGMOD’10]
The Pregel (Push) 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]) to vertex j
```

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

Vertex Programs directly access adjacent vertices and edges

```python
GraphLab_PageRank(i):
    // Compute sum over neighbors
    total = 0
    foreach (j in neighbors(i)):
        total = total + R[j] * w_{ji}
    // Update the PageRank
    R[i] = 0.15 + total
```

Data movement is managed by the system and not the user.
Iterative Bulk Synchronous Execution
Graph-Parallel Systems

Exploit graph structure to achieve orders-of-magnitude performance gains over more general data-parallel systems.
Shrinking Working Sets

PageRank on Web Graph

51% of vertices run only once!
The GraphLab (Pull) Abstraction

Vertex Programs directly **access** adjacent vertices and edges

---

**GraphLab_PageRank(i)**

// Compute sum over neighbors
total = 0
foreach (j 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
  signal nbrs0f(i) to be recomputed

---

**Trigger computation only when necessary.**
Synthetic Noisy Image

Splash

Factor Graph

Vertex Updates

Algorithm identifies and focuses on hidden sequential structure

Many Updates

Few Updates
Real-World Graphs

Edges >> Vertices

Power-Law Degree Distribution

Facebook

 AltaVista WebGraph 1.4B Vertices, 6.6B Edges

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

Top 1% of vertices are adjacent to 50% of the edges!
Challenges of High-Degree Vertices

Sequentially process edges

Touches a large fraction of graph

Provably Difficult to Partition
Program This

Split High-Degree vertices

New Abstraction $\rightarrow$ Equivalence on Split Vertices

Run on This

Machine 1

Machine 2
GAS Decomposition

Gather

Apply

Scatter
Minimizing Communication in PowerGraph

Communication is linear in the number of machines each vertex spans.

Total communication upper bound:

\[ O\left(\#\text{vertices} \sqrt{\#\text{machines}}\right) \]
2D Partitioning

$\nu_i$ only has neighbors on 7 machines

16 Machines
PageRank on the Live-Journal Graph

Spark is 4x faster than Hadoop
GraphLab is 16x faster than Spark
Triangle Counting on Twitter

40M Users, 1.4 Billion Links

Counted: 34.8 Billion Triangles

Hadoop

1536 Machines
423 Minutes

GraphLab

64 Machines
15 Seconds

1000 x Faster

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

Raw Wikipedia

Text Table

XML

Discussion Table

Editor Graph

Hyperlinks

Term-Doc Graph

PageRank

Topic Model (LDA)

Community Detection

Top 20 Pages

Word Topics

User Community

Topic Model (LDA)

Community Table

123
Separate Systems to Support Each View

Table View

Graph View

Table

Row

Row

Row

Row

Result

Dependency Graph
Having separate systems for each view is difficult to use and inefficient.
Difficult to Program and Use

Users must **Learn, Deploy, and Manage** multiple systems

Leads to brittle and often complex interfaces
Inefficient

Extensive data movement and duplication across the network and file system

Limited reuse internal data-structures across stages
GraphX Solution: Tables and Graphs are views of the same physical data.

Table View

GraphX Unified Representation

Graph View

Each view has its own operators that exploit the semantics of the view to achieve efficient execution.
Graphs → Dataflow

1. Encode graphs as distributed tables (RDDs)
2. Express graph computation in relational ops.
3. Recast graph systems optimizations as:
   1. Distributed join optimization
   2. Incremental materialized maintenance

Integrate Graph and Table data processing systems.
Achieve performance parity with specialized systems.
Distributed Graphs as Distributed Tables

Property Graph

Game Graph

2D Vertex Cut Heuristic

Part. 1

Part. 2

Vertex Table

Routing Table

Edge Table
# Spark Dataflow Operators

Inherited from Spark:

<table>
<thead>
<tr>
<th>Operator</th>
<th>Operator</th>
<th>Operator</th>
</tr>
</thead>
<tbody>
<tr>
<td>map</td>
<td>reduce</td>
<td>sample</td>
</tr>
<tr>
<td>filter</td>
<td>count</td>
<td>take</td>
</tr>
<tr>
<td>groupBy</td>
<td>fold</td>
<td>first</td>
</tr>
<tr>
<td>sort</td>
<td>reduceByKey</td>
<td>partitionBy</td>
</tr>
<tr>
<td>union</td>
<td>groupByKey</td>
<td>mapWith</td>
</tr>
<tr>
<td>join</td>
<td>cogroup</td>
<td>pipe</td>
</tr>
<tr>
<td>leftOuterJoin</td>
<td>cross</td>
<td>save</td>
</tr>
<tr>
<td>rightOuterJoin</td>
<td>zip</td>
<td>...</td>
</tr>
</tbody>
</table>
class Graph[ V, E ] {
    def Graph(vertices: Table[ (Id, V) ],
              edges: Table[ (Id, Id, E) ]) {
        // Table Views ----------------------------
        def vertices: Table[ (Id, V) ]
        def edges: Table[ (Id, Id, E) ]
        def triplets: Table[ ((Id, V), (Id, V), E) ]
        // Transformations ------------------------
        def reverse: Graph[ V, E ]
        def subgraph(pV: (Id, V) => Boolean,
                      pE: Edge[ V, E ] => Boolean): Graph[ V, E ]
        def mapV(m: (Id, V) => T): Graph[ T, E ]
        def mapE(m: Edge[ V, E ] => T): Graph[ V, T ]
        // Joins ---------------------------------
        def joinV(tbl: Table[ (Id, T) ]): Graph[ (V, T), E ]
        def joinE(tbl: Table[ (Id, Id, T) ]): Graph[ V, (E, T) ]
        // Computation ---------------------------
        def mrTriplets(mapF: (Edge[ V, E ]) => List[(Id, T)],
                       reduceF: (T, T) => T): Graph[ T, E ]
    }
}

Graph Operators
Triplets Join Vertices and Edges

The *triplets* operator joins vertices and edges:

```sql
SELECT t.dstId,
       reduce(map(t)) AS sum
FROM triplets AS t
GROUP BY t.dstId
```

The *mrTriplets* operator sums adjacent triplets.

```sql
SELECT s.Id, d.Id, s.P, e.P, d.P
FROM edges AS e
JOIN vertices AS s, vertices AS d ON
e.srcId = s.Id AND e.dstId = d.Id
```

```sql
SELECT t.dstId, reduce(map(t)) AS sum
FROM triplets AS t
GROUP BY t.dstId
```
Example: Oldest Follower

Calculate the number of older followers for each user?

```scala
val olderFollowerAge = graph
  .mrTriplets(
    e => // Map
      if(e.src.age < e.dst.age) {
        (e.srcId, 1)
      } else { Empty }
    ,
    (a,b) => a + b // Reduce
  )
  .vertices
```
Caching for Iterative mrTriplets

Vertex Table (RDD)

A
B
C
D
E
F

Edge Table (RDD)

Mirror Cache

A
B
C
D

Mirror Cache

A
D
E
F

A
B
C
D

A
B
C
D

A
E
F

A
F
D
E
Incremental Updates for Iterative mrTriplets
Aggregation for Iterative mrTriplets

Vertex Table (RDD)

<table>
<thead>
<tr>
<th>Change</th>
<th>A</th>
<th>Change</th>
<th>B</th>
<th>Change</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Edge Table (RDD)

<table>
<thead>
<tr>
<th>Mirror Cache</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Local Aggregate

Local Aggregate

Local Aggregate

Scan

Change

Change
PageRank in GraphX

// Load and initialize the graph
val graph = GraphBuilder.text("hdfs://web.txt")
val prGraph = graph.joinVertices(graph.outDegrees)

// Implement and Run PageRank
val pageRank =
  prGraph.pregel(initialMessage = 0.0, iter = 10)(
    (oldV, msgSum) => 0.15 + 0.85 * msgSum,
    triplet => triplet.src.pr / triplet.src.deg,
    (msgA, msgB) => msgA + msgB)
Example Analytics Pipeline

// Load raw data tables
val verts = sc.textFile("hdfs://users.txt").map(parserV)
val edges = sc.textFile("hdfs://follow.txt").map(parserE)

// Build the graph from tables and restrict to recent links
val graph = new Graph(verts, edges)
val recent = graph.subgraph(edge => edge.date > LAST_MONTH)

// Run PageRank Algorithm
val pr = graph.PageRank(tol = 1.0e-5)

// Extract and print the top 25 users
val topUsers = verts.join(pr).top(25).collect

// Print the top users
val topUsers.foreach(u => println(u.name + \’\t\’ + u.pr))
GraphX scales to larger graphs

Twitter Graph: 1.5 Billion Edges

- Giraph: 749 seconds
- GraphX: 451 seconds
- GraphLab: 203 seconds

Runtime (in seconds, PageRank for 10 iterations)

GraphX is roughly **2x slower** than GraphLab

- Scala + Java overhead: Lambdas, GC time, …
- No shared memory parallelism: **2x increase** in comm.
PageRank is just one stage....

What about a pipeline?
A Small Pipeline in GraphX

Raw Wikipedia → Hyperlinks → PageRank → Top 20 Pages

- Spark Preprocess
- Compute
- Spark Post.

Timed end-to-end GraphX is faster than GraphLab
GraphX: Unified Graph Analytics

**New API**
Blurs the distinction between Tables and Graphs

**New System**
Combines Data-Parallel Graph-Parallel Systems

Enabling users to easily and efficiently express the entire graph analytics pipeline
Current Limitations of GraphX

No support for asynchronous computation

• Favor determinism over speed

Not optimized for out-of-core processing

GraphLab Create (GraphLab + GraphX):

• Supports asynchrony and out-of-core processing

• Currently not distributed
Outline of the Tutorial

- Data Parallel
- Model Parallel
- Graph Parallel
Outline of the Tutorial

- Data Parallel
- GraphX & GraphLab Create
- Graph Parallel
- Model Parallel
Future Directions
Themes in Learning Systems

Optimize for common patterns: aggregation, iteration, large-models, and graphs

- Others?

Leverage hardware trends: in-memory comp and elastic compute on commodity hardware

- RDMA, SSDs?

Tradeoff accuracy and runtime with sampling and asynchrony
Research Opportunities

Database Systems Research

Concurrent State Mgmt.

Sampling and Approx. Query Processing

Query Optimization

Machine Learning Research
Concurrency Control

Coordination Free (Parameter Server):

Provably fast and correct under key assumptions.

Optimistic Concurrency Control:

Provably correct and fast under key assumptions.


Database Systems Improve Efficiency
Exploit sampling for fast, approximate answers with error bars:

\[
\text{SELECT } \text{avg(sessionTime)} \\
\text{FROM Table} \\
\text{WHERE city='San Francisco'} \\
\text{WITHIN 2 SECONDS}
\]

\[
\text{SELECT } \text{avg(sessionTime)} \\
\text{FROM Table} \\
\text{WHERE city='San Francisco'} \\
\text{ERROR 0.1 CONFIDENCE 95.0%}
\]

Queries with Time Bounds

Queries with Error Bounds

Can we do the same for learning?

Agarwal et al., BlinkDB: Queries with Bounded Errors and Bounded Response Times on Very Large Data. ACM EuroSys 2013,
Insight: A Declarative Approach to ML

SQL → Result → MLQL → Model

Thank You

Questions?

Joseph E. Gonzalez
jegonzal@eecs.berkeley.edu

Slides (with animations) available at
http://eecs.berkeley.edu/~jegonzal
How Systems Researchers Build Systems

Define the Problem
  » Identify constraints and abstract the problem

Propose Solution: *Simple Idea*
  » Don’t try to solve everything

Implement the System
  » Reuse existing systems wherever possible

Evaluation
  » Support the design decisions
  » What are the tradeoffs and limitations?