From Graphs to Tables: The Design of Scalable Systems for Graph Analytics

Joseph E. Gonzalez
Post-doc, UC Berkeley AMPLab
jegonzal@eecs.berkeley.edu
Co-founder, GraphLab Inc.
joseph@graphlab.com

WWW’14 Workshop on Big Graph Mining

*These slides are best viewed in PowerPoint with animation.
Graphs are Central to Analytics

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

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

Update ranks in parallel
Iterate until convergence
Recommending Products

Users  Ratings  Items

[Images of users, ratings, and items]


**Recommending Products**

Low-Rank Matrix Factorization:

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

Conditional Random Field
Belief Propagation
Mean Field Algorithm

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

Count triangles passing through each vertex:

Measures “cohesiveness” of local community

Fewer Triangles
Weaker Community

More Triangles
Stronger Community
The Graph-Parallel Pattern

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.
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

Compute

Communicate

Barrier
Graph-Parallel Systems

Exploit graph structure to achieve orders-of-magnitude performance gains over more general data-parallel systems.
Real-World Graphs

Edges >> Vertices

Power-Law Degree Distribution

Facebook

Ratio of Edges to Vertices

Year

Number of Vertices

Degree

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

Top 1% of vertices are adjacent to 50% of the edges!

AltaVista WebGraph 1.4B Vertices, 6.6B Edges
Challenges of High-Degree Vertices

- Sequentially process edges
- Touches a large fraction of graph

Provably Difficult to Partition
GraphLab
(PowerGraph, OSDI’12)

Program This

Run on This

Split High-Degree vertices

New Abstraction $\rightarrow$ Equivalence on Split Vertices
GAS Decomposition

Gather

Apply

Scatter
Minimizing Communication in PowerGraph

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

Total communication upper bound:

\[ O\left(\#\text{vertices}\sqrt{\#\text{machines}}\right) \]
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 nbrsOf(i) to be recomputed
```

Trigger computation only when necessary.
PageRank on the Live-Journal Graph

GraphLab is *60x 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

[WWW'11]

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
# Tables

- Raw Wikipedia
- XML
- Hyperlinks
- PageRank
- Word Topics
- User Community
- Community Topic
- Top 20 Pages

## Illustration

- Text Table
- Term-Doc Graph
- Editor Graph
- Community Detection
- Discussion Table
- Topic Model (LDA)
Graphs

Hyperlinks

PageRank

Top 20 Pages

Title

PR

Word Topics

User Topic

Community Topic

Editor Graph

Community Detection

User Community

Discussion Table

User Disc.
Separate Systems to Support Each View

Table View

Graph View

![Diagram showing the separation of systems for Table View and Graph View. The Table View uses Hadoop and Spark for processing rows of data, resulting in a Table. The Graph View uses Pregel, GraphLab, and Giraph for processing a Dependency Graph.](image-url)
Separate systems for each view can be 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.
Solution: The GraphX Unified Approach

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
Tables and Graphs are composable views of the same physical data.

Each view has its own operators that exploit the semantics of the view to achieve efficient execution.
View a Graph as a Table

**Property Graph**

**Vertex Property Table**

<table>
<thead>
<tr>
<th>Id</th>
<th>Property (V)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rxin</td>
<td>(Stu., Berk.)</td>
</tr>
<tr>
<td>Jegonzal</td>
<td>(PstDoc, Berk.)</td>
</tr>
<tr>
<td>Franklin</td>
<td>(Prof., Berk)</td>
</tr>
<tr>
<td>Istoica</td>
<td>(Prof., Berk)</td>
</tr>
</tbody>
</table>

**Edge Property Table**

<table>
<thead>
<tr>
<th>SrcId</th>
<th>DstId</th>
<th>Property (E)</th>
</tr>
</thead>
<tbody>
<tr>
<td>rxin</td>
<td>jegonzal</td>
<td>Friend</td>
</tr>
<tr>
<td>franklin</td>
<td>rxin</td>
<td>Advisor</td>
</tr>
<tr>
<td>istoica</td>
<td>franklin</td>
<td>Coworker</td>
</tr>
<tr>
<td>franklin</td>
<td>jegonzal</td>
<td>PI</td>
</tr>
</tbody>
</table>
# Table Operators

Table (RDD) operators are inherited from Spark:

<table>
<thead>
<tr>
<th>map</th>
<th>reduce</th>
<th>sample</th>
</tr>
</thead>
<tbody>
<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]
The *triplets* operator joins vertices and edges:

```sql
SELECT t.dstId,
reduce( map(t) ) AS sum
FROM triplets AS t
GROUPBY 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
```
We express *enhanced* Pregel and GraphLab abstractions using the GraphX operators in less than 50 lines of code!
Enhanced to Pregel in GraphX

```python
pregelPR(i, messageSum):
    // Receive all the messages
    total = 0
    foreach(msg in messageList):
        total = total + msg

    // Update the rank of this vertex
    R[i] = 0.15 + total

combineMsg(a, b):
    // Compute sum of two messages
    return a + b

sendMsg(i → j, R[i], R[j], E[i, j]):
    // Compute single message
    return msg(R[i] / E[i, j])
```

Require Message Combiners

Remove Message Computation from the Vertex Program

Malewicz et al. [PODC’09, SIGMOD’10]
Implementing 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)
We express the Pregel and GraphLab *like* abstractions using the GraphX operators in less than 50 lines of code!

By composing these operators we can construct entire graph-analytics pipelines.
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
topUsers.foreach(u => println(u.name + \t' + u.pr))
GraphX System Design
Distributed Graphs as Tables (RDDs)

Property Graph

2D Vertex Cut Heuristics

Distributed Graphs as Tables (RDDs)

Vertex Table (RDD)

Routing Table (RDD)

Edge Table (RDD)

1D Vertex Cut Heuristic
Caching for Iterative mrTriplets

Vertex Table (RDD)

Edge Table (RDD)

Mirror Cache

A
B
C
D

A
B
C
D

Mirror Cache

A
D
E
F

A
D
E
F
Incremental Updates for Iterative mrTriplets

**Vertex Table (RDD)**
- Change
  - A
  - B
  - C
  - D
  - E
  - F

**Edge Table (RDD)**
- Mirror Cache
  - A
  - B
  - C
  - D
  - E
  - F

**Mirror Cache**
- Scan
- A
- B
- C
- D
- E
- F
Aggregation for Iterative mrTriplets

Vertex Table (RDD)

Change

Change

Change

Change

Change

Change

Edge Table (RDD)

Local Aggregate

Local Aggregate

Scan
Reduction in Communication Due to Cached Updates

Connected Components on Twitter Graph

Most vertices are within 8 hops of all vertices in their comp.
Benefit of Indexing Active Edges

Connected Components on Twitter Graph

- Scan
- Indexed

Runtime (Seconds)

Iteration

Scan All Edges

Index of “Active” Edges
Join Elimination

Identify and bypass joins for unused triplet fields

sendMsg(i⇒j, R[i], \[R[j]\], E[i,j]):

// Compute single message
return msg(R[i]/E[i,j])

PageRank on Twitter

- Three Way Join
- Join Elimination

Factor of 2 reduction in communication
Additional Query Optimizations

Indexing and Bitmaps:
- To accelerate joins across graphs
- To efficiently construct sub-graphs

Substantial Index and Data Reuse:
- Reuse routing tables across graphs and sub-graphs
- Reuse edge adjacency information and indices
Performance Comparisons

GraphX is roughly 3x slower than GraphLab
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 \textbf{2x slower} than GraphLab

\(\text{» Scala + Java overhead: Lambdas, GC time, ...}\)

\(\text{» No shared memory parallelism: 2x increase in comm.}\)
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
Conclusion and Observations

Domain specific views: *Tables and Graphs*
- tables and graphs are first-class composable objects
- specialized operators which exploit view semantics

Single system that efficiently spans the pipeline
- minimize data movement and duplication
- eliminates need to learn and manage multiple systems

Graphs through the lens of database systems
- Graph-Parallel Pattern $\rightarrow$ Triplet joins in relational alg.
- Graph Systems $\rightarrow$ Distributed join optimizations
Open Source Project

Alpha release as part of Spark 0.9

GraphX Programming Guide – Spark 0.9.0.0 Documentation

GraphX is the new (alpha) Spark API for graphs and graph-parallel computation. At a high-level, GraphX extends the Spark RDD by introducing the Resilient Distributed Property Graph: a directed multigraph with properties attached to each vertex and edge. To support graph computation, GraphX exposes a set of fundamental operators (e.g., subgraph, joinVertices, and mapReduceVertices) as well as an optimized variant of the Pregel API. In addition, GraphX includes a growing collection of graph algorithms and builders to simplify graph analytics tasks.

Background on Graph-Parallel Computation

From social networks to language modeling, the growing scale and importance of graph data has driven the development of numerous new graph-parallel systems (e.g., Giraph and GraphLab). By restricting the types of computation that can be expressed and introducing new techniques to partition and distribute graphs, these systems can efficiently execute sophisticated graph algorithms orders of magnitude faster than more general data-parallel systems.
Active Research

Static Data ➔ Dynamic Data
» Apply GraphX unified approach to time evolving data
» Materialized view maintenance for graphs

Serving Graph Structured Data
» Allow external systems to interact with GraphX
» Unify distributed graph databases with relational database technology
Collaborators

**GraphLab:**
- Yucheng Low
- Haijie Gu
- Aapo Kyrola
- Danny Bickson
- Carlos Guestrin
- Alex Smola
- Guy Blelloch

**GraphX:**
- Reynold Xin
- Ankur Dave
- Daniel Crankshaw
- Michael Franklin
- Ion Stoica
Thanks!

http://tinyurl.com/ampgraphx

jegonzal@eecs.berkeley.edu