GraphX: Unifying Table and Graph Analytics

Presented by Joseph Gonzalez

Joint work with Reynold Xin, Daniel Crankshaw, Ankur Dave, Michael Franklin, and Ion Stoica

IPDPS 2014
Graphs are Central to Analytics

Raw Wikipedia

Text Table

Hyperlinks

Term-Doc Graph

Topic Model (LDA)

Word Topics

Discussion Table

Editor Graph

Community Detection

User Community

Community Topic

User-Com

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

Rank of user \(i\)

Weighted sum of neighbors’ ranks
**Recommending Products**

Low-Rank Matrix Factorization:

Iterate:

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

**SOCIAL NETWORK ANALYSIS**
- Community Detection
  - CoEM
  - Triangle Counting
  - K-core Decomposition
  - K-Truss

**GRAPH ALGORITHMS**
- Graph Analytics
  - PageRank
  - Personalized PageRank
  - 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.

```python
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

```
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
Exposé spécialisées APIs pour simplifier le programmation sur graphes.

Exploiter la structure du graphique pour réaliser des gains d'ordre de magnitude par rapport à des systèmes plus généraux de programmation parallèle de données.
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

[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
PageRank
Graphs

- Hyperlinks
- PageRank
- Term-Doc Graph
- Topic Model (LDA)
- Editor Graph
- Community Detection
Separate Systems to Support Each View

Table View

Graph View

Table

Row

Row

Row

Row

Row

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.

Each view has its own operators that exploit the semantics of the view to achieve efficient execution.
Graphs $\rightarrow$ Relational Algebra

1. Encode graphs as distributed tables

2. Express graph computation in relational algebra

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

Vertex Table

Routing Table

Edge Table
Table Operators

Table operators are 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>left Outer Join</td>
<td>cross</td>
<td>save</td>
</tr>
<tr>
<td>right Outer Join</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 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
```

The *mrTriplets* operator sums adjacent triplets:

```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
  .mTriplets(
    e => // Map
      if(e.src.age < e.dst.age) {
        (e.srcId, 1)
      }
      else { Empty }
  )
  .vertices
  .reduce((a, b) => a + b))
```
We express *enhanced* Pregel and GraphLab abstractions using the GraphX operators in less than 50 lines of code!
**Enhanced Pregel in GraphX**

preGelPR(i, messageList):

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

---

Malewicz et al. [PODC’09, SIGMOD’10]
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)
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

Communication (MB)

Factor of 2 reduction in communication
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))
The GraphX Stack (Lines of Code)

- PageRank (5)
- Connected Comp. (10)
- Shortest Path (10)
- SVD (40)
- ALS (40)
- K-core (51)
- Triangle Count (45)
- LDA (120)

- Pregel (28) + GraphLab (50)

- GraphX (3575)

- Spark
Performance Comparisons

Live-Journal: 69 Million Edges

- **Mahout/Hadoop**: 1340 seconds
- **Naïve Spark**: 354 seconds

GraphX is roughly **3x slower** than GraphLab.
GraphX scales to larger graphs

Twitter Graph: 1.5 Billion Edges

<table>
<thead>
<tr>
<th>Library</th>
<th>Runtime (in seconds, PageRank for 10 iterations)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Giraph</td>
<td>749</td>
</tr>
<tr>
<td>GraphX</td>
<td>451</td>
</tr>
<tr>
<td>GraphLab</td>
<td>203</td>
</tr>
</tbody>
</table>

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.
Status

Part of Apache Spark

In production at several large technology companies
GraphX: Unified 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
A Case for Algebra in Graphs

A standard algebra is essential for graph systems:

- e.g.: SQL \(\rightarrow\) proliferation of relational system

By embedding graphs in *relational algebra*:

- Integration with tables and preprocessing
- Leverage advances in relational systems
- Graph opt. recast to relational systems
Thanks!

http://amplab.cs.berkeley.edu/projects/gra
phx/

ankurd@eecs.berkeley.edu
crankshaw@eecs.berkeley.edu
rxin@eecs.berkeley.edu
jegonzal@eecs.berkeley.edu