GraphX: Unifying Data-Parallel and Graph-Parallel Analytics

Presented by Daniel Crankshaw

Joint work with Reynold Xin, Joseph Gonzalez, Ankur Dave, Michael Franklin, and Ion Stoica

Spark Meetup 3/25/2014
Graphs are Central to Analytics

Raw Wikipedia

Text Table

Term-Doc Graph

Hyperlinks

PageRank

Top 20 Pages

Title PR

XML

Title Body

Topic Model (LDA)

Word Topics

Word Topic

Discussion Table

Editor Graph

Community Detection

User Community

User Com.

Community Topic

Topic Com.
PageRank: Identifying Leaders

\[ R[i] \]

Update ranks in parallel

Iterate until convergence

Rank of user \( i \)

Weighted sum of neighbors’ ranks
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

MACHINE LEARNING

SOCIAL NETWORK ANALYSIS

GRAPH ALGORITHMS
Graph-Parallel Systems

Expose *specialized APIs* to simplify graph programming.

Exploit graph structure to achieve *orders-of-magnitude performance gains* over more general data-parallel systems.
PageRank on the Live-Journal Graph

Spark is \textit{4x faster} than Hadoop
GraphLab is \textit{16x faster} than Spark
are Central to Analytics

Raw Wikipedia

Text Table

Hyperlinks

PageRank

Top 20 Pages

XML

Hyperlinks

PageRank

Top 20 Pages

Discussion Table

Editor Graph

Community Detection

User Community

Word Topics

Term-Doc Graph

Topic Model (LDA)

User Community

Term-Doc Graph

Topic Model (LDA)

Community Topic

Editor Graph

Community Detection

User Community

Community Topic

Discussion Table

Editor Graph

Community Detection

User Community

Community Topic

Discussion Table

Editor Graph

Community Detection

User Community

Community Topic
Separate Systems to Support Each View

Table View

Graph View

Table

Row

Row

Row

Row

Result

Dependency Graph

hadoop

Spark

Pregel

GraphLab

Apache Giraph
Spark

Fast and expressive cluster computing system interoperable with Apache Hadoop

Improves efficiency through:
» In-memory computing primitives
» General computation graphs

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

Up to 100× faster
(2-10× on disk)

Often 5× less code
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
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:

- map
- filter
- groupBy
- sort
- union
- join
- leftOuterJoin
- rightOuterJoin
- reduce
- count
- fold
- reduceByKey
- groupByKey
- cogroup
- cross
- zip
- sample
- take
- first
- partitionBy
- mapWith
- pipe
- save
- ...
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:

\[
\text{SELECT } t.\text{dstId}, \text{reduceUDF( mapUDF(t) ) AS sum}
\text{FROM triplets AS t GROUPBY t.\text{dstId}}
\]
Map Reduce Triplets

*Map-Reduce for each vertex*

\[
\text{mapF}(A \rightarrow B) \rightarrow A_1
\]

\[
\text{mapF}(A \rightarrow C) \rightarrow A_2
\]

\[
\text{reduceF}(A_1, A_2) \rightarrow A
\]
We express the Pregel and GraphLab abstractions using the GraphX operators in less than 50 lines of code!
Example: Oldest Follower

What is the age of the oldest follower for each user?

val oldestFollowerAge = graph
  .mrTriplets(
    e => (e.dst.id, e.src.age), //Map
    (a, b) => max(a, b) //Reduce
  )
  .vertices
We express the Pregel and GraphLab abstractions using the GraphX operators in less than 50 lines of code!

By composing these operators we can construct entire graph-analytics pipelines.
DIY Demo this Afternoon

2. Introduction to the GraphX API

To get started you first need to import GraphX. Start the Spark-Shell (by running the following on the root node):

```
/root/spark/bin/spark shell
```

and paste the following in your Spark shell:

```
Scala
1 import org.apache.spark.graphx._
2 import org.apache.spark.rdd.ANDF
```

2.1. The Property Graph

The property graph is a directed multigraph (a directed graph with potentially multiple parallel edges sharing the same source and destination vertex) with properties attached to each vertex and edge. Each vertex is keyed by a unique 64-bit long identifier (VertexId). Similarly, edges have corresponding source and destination vertex identifiers. The properties are stored as Scala/Java objects with each edge and vertex in the graph.

Throughout the first half of this tutorial we will use the following toy property graph. While this is hardly big data, it provides an opportunity to learn about the graph data model and the GraphX API. In this example we have a small social network with users and their ages modeled as vertices and likes modeled as directed edges.

We begin by creating the property graph from arrays of vertices and edges. Later we will demonstrate how to load real data. Paste the following code into the spark shell.

```
Scala
1 val vertexArray = Array(
2   (1L, "Alice", 29),
3   (2L, "Bob", 27),
4   (3L, "Dennis", 65),
5   (4L, "David", 42),
6   (5L, "Ed", 55),
7   (6L, "Finn", 28))
```
Demo
GraphX System Design
Distributed Graphs as Tables (RDDs)

Property Graph

2D Vertex Cut Heuristic

Part. 1

Part. 2

Vertex Table (RDD)

Routing Table (RDD)

Edge Table (RDD)
Caching for Iterative mrTriplets

Vertex Table (RDD)

Edge Table (RDD)

Mirror Cache

A
B
C
D

Mirror Cache

A
B
C
D

A
E
F

A
F
D

Incremental Updates for Iterative mrTriplets
Aggregation for Iterative mrTriplets
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**

- **Scan All Edges**
- **Index of “Active” Edges**

**Runtime (Seconds)**

**Iteration**

0  2  4  6  8  10  12  14  16
Runtime Join Elimination

Identify and bypass joins for unused triplet fields

\[
\text{sendMsg}(i \rightarrow j, R[i], R[j], E[i,j]):
// Compute single message
\text{return msg}(R[i]/E[i,j])
\]

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
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 **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.

- **Spark**: 1492 seconds
- **Giraph + Spark**: 605 seconds
- **GraphX**: 342 seconds
- **GraphLab + Spark**: 375 seconds

**Total Runtime (in Seconds)**

**Performance Parity!**
The GraphX Stack
(Lines of Code)
Status

In alpha as part of latest Spark release

Seeking collaborators and feedback
GraphX is the new (alpha) Spark API for graphs (e.g., Web-Graphs and Social Networks) and graph-parallel computation (e.g., PageRank and Collaborative Filtering). At a high-level, GraphX extends the Spark RDD abstraction 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 mapReduceTriplets) as well as an optimized variant of the Pregel API. In addition, GraphX includes a
Active Research

Static Graphs ➔ Streaming Graphs
  » Apply GraphX unified approach to time evolving data
  » Model and analyze relationships over time

Graph Analytics + Serving
  » Allow external systems to interact with GraphX
  » Unify distributed graph databases with relational database technology
Conclusions

Composable domain specific views and operators

Single system that efficiently spans the pipeline

Graphs through the lens of database systems
  » Graph-Parallel Pattern ➔ Triplet joins in relational alg.
  » Graph Systems ➔ Distributed join optimizations

Dan Crankshaw
crankshaw@eecs.berkeley.edu
http://tinyurl.com/ampgraphx
Thanks!

http://tinyurl.com/ampgraphx

crankshaw@eecs.berkeley.edu

ejgonzal@eecs.berkeley.edu
ankanurd@eecs.berkeley.edu
rxin@eecs.berkeley.edu
Graph Property 1
Real-World Graphs

Power-Law Degree Distribution

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

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

Edges >> Vertices

AltaVista WebGraph 1.4B Vertices, 6.6B Edges

Facebook

Number of Vertices

Ratio of Edges to Vertices

Degree

Year

2008  2009  2010  2011  2012

200  180  160  140  120

100  80  60  40  20

0
Graph Property 2

Active Vertices

PageRank on Web Graph

51% updated only once!
Graphs are Essential to Data Mining and Machine Learning

Identify influential people and information

Find communities

Understand people’s shared interests

Model complex data dependencies
Recommending Products
Recommending Products

Low-Rank Matrix Factorization:

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

Conditional Random Field
Belief Propagation
Finding Communities

Count triangles passing through each vertex:

Measures “cohesiveness” of local community

Fewer Triangles
Weaker Community

More Triangles
Stronger Community
Example Graph Analytics Pipeline

Preprocessing

Raw Data

ETL

Initial Graph

Slice

Subgraph

Compute

PageRank

Post Proc.

Analyse

Top Users

Repeat