GraphX: Graph Processing in a Distributed Dataflow Framework

Joseph Gonzalez
Postdoc, UC-Berkeley AMPLab
Co-founder, GraphLab Inc.

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

OSDI 2014
Modern Analytics

Raw Wikipedia

Link Table

Hyperlinks

PageRank

Top 20 Pages

Top Communities

Discussion Table

Editor Graph

Community Detection

User Community

Table: Title | Link

Table: Title | PR

Table: Com. | PR..
Graphs

Raw Wikipedia

Link Table

Hyperlinks

PageRank

Top 20 Pages

XML

Editor Graph

Community Detection

Top Communities

Discussion Table

User Community

User | Com.
---|---

<table>
<thead>
<tr>
<th>Title</th>
<th>Link</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Title</th>
<th>PR</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Com.</th>
<th>PR..</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>User</th>
<th>Disc.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Separate Systems

Tables

Graphs
Separate Systems

Dataflow Systems

Graphs
Separate Systems

Dataflow Systems

Graph Systems

Table

Row

Row

Row

Row

Result

Dependency Graph

hadoop

Spark

Pregel

GraphLab

Apache Giraph
Difficult to 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 Unifies Computation on Tables and Graphs

Enabling a single system to easily and efficiently support the entire pipeline.
Separate Systems

Dataflow Systems

Graph Systems

Table

Row

Row

Row

Row

Result

Dependency Graph

hadoop

Spark

Pregel

GraphLab

APACHE GIRAPH
PageRank on the Live-Journal Graph

Hadoop is 60x slower than GraphLab
Spark is 16x slower than GraphLab
Key Question

How can we *naturally express* and *efficiently execute* graph computation in a general purpose dataflow framework?

Distill the lessons learned from specialized graph systems
Key Question

How can we *naturally express* and *efficiently execute* graph computation in a general purpose dataflow framework?

Representation    Optimizations
Raw Wikipedia XML

Link Table

Hyperlinks

PageRank

Top 20 Pages

Top Communities

Discussion Table

Editor Graph

Community Detection

User Community

Table

Table

Table
Example Computation: PageRank

Express computation \textit{locally}:

\[ R[i] = 0.15 + \sum_{j \in \text{InLinks}(i)} \frac{R[j]}{\text{OutLinks}(j)} \]

\textbf{Iterate} until convergence
“Think like a Vertex.”

- Malewicz et al., SIGMOD’10
Graph-Parallel Pattern

Gonzalez et al. [OSDI’12]

Gather information from neighboring vertices
Graph-Parallel Pattern

Gonzalez et al. [OSDI’12]

Apply an update the vertex property
Graph-Parallel Pattern

Gonzalez et al. [OSDI’12]

Scatter information to neighboring vertices
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

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

Semi-supervised ML
  » Graph SSL
  » CoEM

MACHINE LEARNING

NETWORK ANALYSIS

PageRank
  » Personalized PageRank
  » Shortest Path
  » Graph Coloring

Graph Analytics
  » PageRank
  » Semi-supervised ML
  » Community Detection
Specialized Computational Pattern → Specialized Graph Optimizations
Graph System Optimizations

Specialized Data-Structures

Vertex-Cuts Partitioning

Remote Caching / Mirroring

Message Combiners

Active Set Tracking
Representation

Distributed Graphs

Horizontally Partitioned Tables

Vertex Programs

Join

Dataflow Operators

Optimizations

Advances in Graph Processing Systems

Distributed Join Optimization

Materialized View Maintenance
Property Graph Data Model

Property Graph

Vertex Property:
- User Profile
- Current PageRank Value

Edge Property:
- Weights
- Timestamps
Encoding Property Graphs as Tables

Property Graph

Vertextail

Part. 1

Part. 2

Vertex Table (RDD)

Routing Table (RDD)

Edge Table (RDD)
Separate Properties and Structure

Reuse structural information across multiple graphs

Input Graph

Transform Vertex Properties

Transformed Graph
Table Operators

Table 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
- ...

29
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 (Scala)

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]
}

capture the Gather-Scatter pattern from specialized graph-processing systems
Triplets Join Vertices and Edges

The *triplets* operator joins vertices and edges:

**Vertices**

- A
- B
- C
- D

**Triplets**

- A
- B
- C
- D

**Edges**

- A
- B
- C
- D
Map-Reduce Triplets

Map-Reduce triplets collects information about the neighborhood of each vertex:

MapFunction( \(A\rightarrow B\) ) \rightarrow (B, \text{message})
MapFunction( \(A\rightarrow C\) ) \rightarrow (C, \text{message})
MapFunction( \(B\rightarrow C\) ) \rightarrow (C, \text{message})
MapFunction( \(C\rightarrow D\) ) \rightarrow (D, \text{message})

Reducing:
(B, \text{message}) + (C, \text{message}) \rightarrow (C, \text{message} \oplus \text{message})

Src. or Dst.
Using these basic GraphX operators we implemented Pregel and GraphLab in under 50 lines of code!
The GraphX Stack
(Lines of Code)

Some algorithms are more naturally expressed using the GraphX primitive operators
Representation

- Distributed Graphs
- Horizontally Partitioned Tables
- Vertex Programs
- Join
- Dataflow Operators

Optimizations

- Advances in Graph Processing Systems
- Distributed Join Optimization
- Materialized View Maintenance
Join Site Selection using Routing Tables

Routing Table (RDD)

Vertex Table (RDD)

Edge Table (RDD)

Never Shuffle Edges!
Caching for Iterative mrTriplets

Vertex Table (RDD)

Edge Table (RDD)

Mirror Cache

Reuseable Hash Index

Scan

Reuseable Hash Index
Incremental Updates for Triplets View
Aggregation for Iterative mrTriplets

- **Vertex Table (RDD)**
  - Change
  - Change
  - Change
  - Change

- **Edge Table (RDD)**
  - **Mirror Cache**
    - Change
    - Change
    - Change
    - Change

- **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 Vertices

Connected Components on Twitter Graph

- Without Active Tracking
- Active Vertex Tracking

Runtime (Seconds)

Iteration

0 2 4 6 8 10 12 14 16
Join Elimination

Identify and bypass joins for unused triplet fields

- Java bytecode inspection

PageRank on Twitter

Communication (MB) vs. Iteration

Without Join Elimination

Join Elimination

Factor of 2 reduction in communication

Better
Additional Optimizations

Indexing and Bitmaps:

» To accelerate joins across graphs
» To efficiently construct sub-graphs

Lineage based fault-tolerance

» Exploits Spark lineage to recover in parallel
» Eliminates need for costly check-points

Substantial Index and Data Reuse:

» Reuse routing tables across graphs and sub-graphs
» Reuse edge adjacency information and indices
System Comparison

Goal:

Demonstrate that GraphX achieves performance parity with specialized graph-processing systems.

Setup:

16 node EC2 Cluster (m2.4xLarge) + 1 GigE

Compare against GraphLab/PowerGraph (C++), Giraph (Java), & Spark (Java/Scala)
GraphX performs comparably to state-of-the-art graph processing systems.
GraphX performs comparably to state-of-the-art graph processing systems.
Graphs are just one stage....

What about a pipeline?
A Small Pipeline in GraphX

Raw Wikipedia → Hyperlinks → PageRank → Top 20 Pages

Spark Preprocess → Compute → Spark Post.

Total Runtime (in Seconds)

Timed end-to-end GraphX is the fastest
Adoption and Impact

GraphX is now part of Apache Spark

• Part of Cloudera Hadoop Distribution

In production at Alibaba Taobao

• Order of magnitude gains over Spark

Inspired GraphLab Inc. SFrame technology

• Unifies Tables & Graphs on Disk
GraphX → Unified Tables and Graphs

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

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

Enabling users to easily and efficiently express the entire analytics pipeline
What did we Learn?

Specialized Systems

Integrated Frameworks

Graph Systems

GraphX
Future Work

Specialized Systems

- Graph Systems

Integrated Frameworks

- GraphX

Parameter Server

?
Future Work

Specialized Systems
- Graph Systems
- Parameter Server

Integrated Frameworks
- GraphX
- Asynchrony
  - Non-deterministic
  - Shared-State
Thank You

http://amplab.cs.berkeley.edu/projects/graphx/

jegonzal@eecs.berkeley.edu

Reynold Xin
Ankur Dave
Daniel Crankshaw
Michael Franklin
Ion Stoica
Related Work

Specialized Graph-Processing Systems:
- GraphLab [UAI’10], Pregel [SIGMOD’10], Signal-Collect [ISWC’10], Combinatorial BLAS [IJHPCA’11],
- GraphChi [OSDI’12], PowerGraph [OSDI’12],
- Ligra [PPoPP’13], X-Stream [SOSP’13]

Alternative to Dataflow framework:
- Naiad [SOSP’13]: GraphLINQ
- Hyracks: Pregelix [VLDB’15]

Distributed Join Optimization:
- Multicast Join [Afrati et al., EDBT’10]
- Semi-Join in MapReduce [Blanas et al., SIGMOD’10]
Edge Files Have Locality

GraphLab rebalances the edge-files on-load.

GraphX preserves the on-disk layout through Spark.

→ Better Vertex-Cut
Scalability

Twitter Graph (42M Vertices, 1.5B Edges)

Scales slightly better than PowerGraph/GraphLab

Runtime

EC2-Nodes

GraphX

Linear Scaling
Apache Spark Dataflow Platform

Resilient Distributed Datasets (RDD):

HDFS → Load → RDD

Map → RDD

Reduce → RDD
Apache Spark Dataflow Platform

Resilient Distributed Datasets (RDD):

Optimized for iterative access to data.
GraphX performs comparably to state-of-the-art graph processing systems.
Shared Memory Advantage

Spark Shared Nothing Model

GraphLab Shared Memory

Shared
De-serialized
In-Memory
Graph
Shared Memory Advantage

Spark Shared Nothing Model

GraphLab No SHM.

Twitter Graph (42M Vertices, 1.5B Edges)

Runtime (Seconds)

0  50  100  150  200  250  300  350  400  450  500

GraphLab | GraphLab NoSHM | GraphX
GraphX performs comparably to state-of-the-art graph processing systems.
GraphX performs comparably to state-of-the-art graph processing systems.
Fault-Tolerance

Leverage Spark Fault-Tolerance Mechanism

![Graph showing runtime in seconds for No Failure, Lineage, and Restart conditions. The bars are significantly higher for Restart compared to No Failure and Lineage.](image)
Graph-Processing Systems

Ligra
GraphChi
CombBLAS
GPS
X-Stream
Kineograph

Expose *specialized API* to simplify graph programming.
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
The Vertex-Program Abstraction

GraphLab_PageRank(i)

// Compute sum over neighbors
total = 0
foreach (j in neighbors(i)):
    total += R[j] * w_{ji}

// Update the PageRank
R[i] = 0.15 + total
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
```
Enhanced Pregel in GraphX

\[
\text{pregelPR}(i, \text{messageList}) : \\
\text{// Receive all the messages} \\
\text{total} = 0 \\
\text{foreach (msg in messageList)} : \\
\text{total} = \text{total} + \text{msg} \\
\text{// Update the rank of this vertex} \\
\text{R[i]} = 0.15 + \text{total} \\
\text{combineMsg(a, b)} : \\
\text{// Compute sum of two messages} \\
\text{sendMsg(i \rightarrow j, \text{R[i]}, \text{R[j]}, E[i,j]) :} \\
\text{// Compute single message} \\
\text{Send msg(R[i]/E[i,j]) to vertex}
\]
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 articles = sc.textFile("hdfs://wiki.xml").map(xmlParser)
val links = articles.flatMap(article => article.outLinks)

// Build the graph from tables
val graph = new Graph(articles, links)

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

// Extract and print the top 20 articles
val topArticles = articles.join(pr).top(20).collect
for ((article, pageRank) <- topArticles) {
  println(article.title + \t + pageRank)
}
Apache Spark Dataflow Platform

Zaharia et al., NSDI’12

Resilient Distributed Datasets (RDD):