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
Tables

Raw Wikipedia

Link Table

Hyperlinks

PageRank

Top 20 Pages

Editor Graph

Community Detection

Top Communities

Discussion Table

User Community

User | Disc.
--- | ---
| |

User | Com.
--- | ---
| |
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

Pregel

GraphLab

Apache Giraph
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
Example Computation: PageRank

Express computation \textit{locally}:

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

- \text{Rank of Page } i
- \text{Random Reset Prob.}
- \text{Weighted sum of neighbors’ ranks}

\textit{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
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

Vertext Cut

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:

<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 (Scala)
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
Triplet 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 \((A, B)\) \(\rightarrow\) \((B, \text{Src. or Dst.})\)

MapFunction \((A, C)\) \(\rightarrow\) \((C, \text{Src. or Dst.})\)

MapFunction \((B, C)\) \(\rightarrow\) \((C, \text{Src. or Dst.})\)

MapFunction \((C, D)\) \(\rightarrow\) \((D, \text{Src. or Dst.})\)

Reduce \((B, \text{Src. or Dst.})\) \(\rightarrow\) \((\text{Src. or Dst.}, \text{Message Combiners})\)

Reduce \((C, \text{Src. or Dst.})\) \(\rightarrow\) \((\text{Src. or Dst.}, \text{Message Combiners})\)

Reduce \((D, \text{Src. or Dst.})\) \(\rightarrow\) \((\text{Src. or Dst.}, \text{Message Combiners})\)
Using these basic GraphX operators we implemented Pregel and GraphLab in under 50 lines of code!
The GraphX Stack
(Lines of Code)

PageRank (20)
Connected Comp. (20)
K-core (60)
Triangle Count (50)
...
LDA (220)
SVD++ (110)

GraphX (2,500)

Spark (30,000)

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

Distributed Graphs → Horizontally Partitioned Tables → Vertex Programs → Join

Optimizations

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

Routing Table (RDD)
- A 1 2
- B 1
- C 1
- D 1 2
- E 2
- F 2

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

Edge Table (RDD)
- A -> B
- A -> C
- B -> C
- C -> D
- A -> E
- A -> F
- E -> D
- E -> F

Never Shuffle Edges!
Caching for Iterative mrTriplets

Vertex Table (RDD)

Edge Table (RDD)

Reusable Hash Index

Reusable Hash Index
Incremental Updates for Triplets View

Vertex Table (RDD)

Edge Table (RDD)

Change

Change

Mirror Cache

Scan
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 Vertices

Connected Components on Twitter Graph

- Without Active Tracking
- Active Vertex Tracking

Runtime (Seconds) vs. Iteration
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.

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

Integrated Frameworks

GraphX

Parameter Server

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

GraphX

Linear Scaling

Runtime vs. EC2-Nodes
Apache Spark Dataflow Platform

Resilient Distributed Datasets (RDD):
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)
PageRank Benchmark

Twitter Graph (42M Vertices, 1.5B Edges)

UK-Graph (106M Vertices, 3.7B Edges)

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

![Bar Chart]

- **No Failure**: 500 seconds
- **Lineage**: 700 seconds
- **Restart**: 900 seconds
Graph-Processing Systems

GraphLab  CombBLAS  Pregel
Ligra  ComblLAS  X-Stream
GraphChi  GPS  Kineograph

Representation

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

```python
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
```

Low, Gonzalez, et al. [UAI’10]
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

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

sendMessage(i -> j, R[i], R[j], E[i, j]):
    // Compute single message
    msg(R[i]/E[i, j])
    return msg(R[i]/E[i, j]) to vertex

Require Message Combiners

Remove Message Computation from the Vertex Program
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

Resilient Distributed Datasets (RDD):

Zaharia et al., NSDI'12

Lineage:

Persist in Memory

.code()`