Agenda

• Analytics space background
• Motivation
• Goal
• Approach
• Optimizations
• Results
• Flaws/Limitations
• Questions
Real life Analytics Pipeline

Raw data → Link Table → Page Rank → Desired results

Eg. Google Knowledge graph: 570M Vertices, 18B Edges (as in Mid 2017)
Real life Analytics Pipeline

Raw data → Link Table → Page Rank → Desired results

Tables
Real life Analytics Pipeline

Raw data → Link Table → Page Rank → Desired results

Graphs
Systems landscape
Motivation

• Currently separate systems exist to compute on these data representation.
• Ability to combine data sources.
• Enhance dataflow frameworks to leverage inherent positives.
Current drawbacks of dataflow frameworks

• Implementing iterative algorithms -> requires multiple stages of complex joins.
• Do not cover common patterns in graph algorithms -> Room for optimization.
• Unlike Spark, no fine grained control of data partitioning.
Current drawbacks of specialized systems

• Lacking ability for combining graphs with unstructured or tabular data
• Systems favoring snapshot recovery rather than fault tolerance like in Spark
What can we leverage?

- Immutability of RDD’s
- Reusing indices across graph and collection views over iterations.
- Increase in performance
Goal

• General purpose distributed frameworks for graph computations
• Comparable performances to specialized graph processing systems
Approach

• Unifies Tabular view and Graph view
• Imbibe the best of specialized systems
• Graph representation on dataflow frameworks
• Optimizations
• Develop GraphX API on top of Spark
Graph approach: Page Rank example

- Eg. Page Rank algorithm
- Graph parallel abstraction
- Define a vertex program
- Terminate when vertex programs vote to halt

```scala
def PageRank(v: Id, msgs: List[Double]) {
  // Compute the message sum
  var msgSum = 0
  for (m <- msgs) { msgSum += m }
  // Update the PageRank
  PR(v) = 0.15 + 0.85 * msgSum
  // Broadcast messages with new PR
  for (j <- OutNbrs(v)) {
    msg = PR(v) / NumLinks(v)
    send_msg(to=j, msg)
  }
  // Check for termination
  if (converged(PR(v))) voteToHalt(v)
}
```

Figure: PageRank in Pregel
Approach

- GAS (Gather Apply Scatter)

```python
# Example code

def Gather(a: Double, b: Double) = a + b

def Apply(v, msgSum) {
    PR(v) = 0.15 + 0.85 * msgSum
    if (converged(PR(v))) voteToHalt(v)
}

def Scatter(v, j) = PR(v) / NumLinks(v)
```

How to apply this in dataflow frameworks?

- Map, group-by, join dataflow operators
Representing Property graphs as Tables

Never transfer edges
GraphX API

class Graph[V, E] {
  // Constructor
  def Graph(v: Collection[(Id, V)],
            e: Collection[(Id, Id, E)])

  // Collection views
  def vertices: Collection[(Id, V)]
  def edges: Collection[(Id, Id, E)]
  def triplets: Collection[Triplet]

  // Graph-parallel computation
  def mrTriples(f: (Triplet) => M,
                 sum: (M, M) => M): Collection[(Id, M)]

  // Convenience functions
  def mapV(f: (Id, V) => V): Graph[V, E]
  def mapE(f: (Id, Id, E) => E): Graph[V, E]
  def leftJoinV(v: Collection[(Id, V)],
                f: (Id, V, V) => V): Graph[V, E]
  def leftJoinE(e: Collection[(Id, Id, E)],
                f: (Id, Id, E, E) => E): Graph[V, E]
  def subgraph(vPred: (Id, V) => Boolean,
               ePred: (Triplet) => Boolean): Graph[V, E]
  def reverse: Graph[V, E]
}
Using the dataflow operators

Vertices

A
B
C
D

Triplets

A
B
C

Edges

A
B
C

Logical representation Join of vertices table on edges table
Using the dataflow operators on vertex program

Userdefined

MapFunction(\(A\rightarrow B\)) $\rightarrow$ (B, \(\square\))

MapFunction(\(A\rightarrow C\)) $\rightarrow$ (C, \(\square\))

MapFunction(\(B\rightarrow C\)) $\rightarrow$ (C, \(\square\))

MapFunction(\(C\rightarrow D\)) $\rightarrow$ (D, \(\square\))

Src. or Dst.

Reduce

(B, \(\square\))

(C, \(\square\) $\oplus$ \(\square\))

(D, \(\square\))

Message Combiners
Optimizations

- Specialized Data Structure
- Vertex-cut Partitioning
- Remote caching
- Message Combiners
- Active Set Tracking
Implementing Optimizations

• Reusable Hash index
• Sequential scan or clustered scan based on active set (Dynamic)
• Incremental updates
• Automatic Join elimination

Additional optimizations:
• Memory based shuffle
• Batching and columnar structure
• Variable Integer encoding
Results

(a) Conn. Comp. Twitter
(b) PageRank Twitter
(c) Conn. Comp. uk-2007-05
(d) PageRank uk-2007-05
Results

Scaling for PageRank on Twitter dataset

Effect of partitioning on communication
Current Flaws

• Is not optimized for dynamic graphs.
• Requires incremental updates to routing table.
• Is not designed for streaming applications.
• Asynchronous graph computation not available. This is where Naiad will outperform.
Questions