**Motivation**

- Graph analytics involves viewing the same data as both graphs and tables.

- We currently need separate systems to support each view: both specialized graph systems and general-purpose data-parallel systems.

- Graph systems provide specialized graph-parallel APIs and exploit the graph structure to reduce communication.

- But separate systems increase complexity, lead to unnecessary data movement, and hinder data structure reuse.

- GraphX embeds graph computation in Spark and implements a variety of join optimizations, enabling a much wider range of computation.

**Graphs as Tables**

Horizontally partitioned vertex and edge tables with indexing and join site information.

- Extends the Spark RDDs.
- Immutable, partitioned set of vertices and edges.
- Equivalent to GraphLab representation.
- Vertex-cut partitioning & vertex data movement pattern.

**The GraphX API**

```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)]
    // Computation
    def mTriplets(mapf: Edge[(V, E)] => List[(Id, T)],
                   reducef: (Vertex, T) => T): Graph[T, E]
    // Convenience Functions
    def mTriplets(mapf: Edge[(V, E)] => List[(Id, T)])
    def join(tbl: Table[(Id, T)]): Graph[(V, T)]
    def join(tbl: Table[((Id, V), (Id, V), E)]): Graph[(V, T), E]
    def reverse: Graph[V, E]
    def subgraph(pred: (Id, V) => Boolean,
                 pred: (Edge[(V, E)]) => Boolean): Graph[V, E]
  }
}
```

The `mTriplets` operator joins vertices and edges:

```sql```
SELECT src.Id, dst.Id, src.attr, e.attr, dst.attr
FROM edges AS e  JOIN vertices AS v, vertices AS dst
ON e.srcId = src.Id AND e.dstId = dst.Id
```

**Optimizations**

- Incremental vertex replication: Iterative graph algorithms have active vertex sets that shrink as the algorithm converges. We speed up `mTriplets` by shipping only changed vertex attributes each iteration.

- Structural index reuse: Vertex and edge indices enable fast aggregations and joins. We exploit immutability by reusing vertex/edge indices across operations, including filters, speeding up PageRank by 41% (27 s to 16 s per iteration).

- Sequential scan vs index scan: Though the map phase in `mTriplets` logically uses a sequential scan over edges, this is inefficient for small active vertex sets. When the active vertex fraction is less than 0.8, we instead scan the clustered index on vertices and filter by activeness.

**Evaluation**

- Comparable performance to fastest specialized systems.
- Gains efficient fault tolerance from Spark.

**Example: PageRank**

```scala```
// Load graph:
val graph = Graph(verticies, edges)
val g = new Graph(verticies, edges)
// Restrict to subgraph:
val g2 = g.subgraph().subgraph
// Run PageRank:
val pg = Pregel.run(g2, parameters, 10)
// Get highest-ranking page:
val ranking = pg.vertices.top
```

GraphX: Unified Data-Parallel and Graph-Parallel Analytics

Joseph E. Gonzalez, Reynold S. Xin, Ankur Dave, Daniel Crankshaw, Michael J. Franklin, Ion Stoica