Spark, GraphX, and Scala

By Mark Horeni
History

• Started as a project at Berkley in 2012
• Main Design Goals
  • Parallelism
  • Fault Tolerance
• Donated to Apache
• GraphX also started at Berkley, then again donated to Apache
• Created in response to linearity of MapReduce
Spark Architecture

Driver Program

SparkContext

Cluster Manager

Worker Node

Executor

Cache

Task

Task

Executor

Cache

Task

Task
Scala

• Based off of Java
• Compiles into Java Bytecode
• “Aimed to address criticisms of Java”
• Meant to be more functional
  • Type inferencing, immutability, pattern matching
  • Algebraic Data Types and anonymous types
  • Operator overloading, optional parameters
RDDS and Property Graphs

• Resilient Distributed Dataset
• Multisets (Basically Databases)
• Fault Tolerant
• Read only (Not useful for online Data)
• Directed Multigraph with User Defined Objects
• Have basic operations like map, filter, and reduce
Graph as an RDD

**Property Graph**

- Node 3: Advisor
  - nxin
  - stu.
- Node 5: franklin, prof.
- Node 7: Collab.
  - jgonzal, pst.doc.
- Node 2: Colleague
  - istoica prof.

**Vertex Table**

<table>
<thead>
<tr>
<th>Id</th>
<th>Property (V)</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>(nxin, student)</td>
</tr>
<tr>
<td>7</td>
<td>(jgonzal, postdoc)</td>
</tr>
<tr>
<td>5</td>
<td>(franklin, professor)</td>
</tr>
<tr>
<td>2</td>
<td>(istoica, professor)</td>
</tr>
</tbody>
</table>

**Edge Table**

<table>
<thead>
<tr>
<th>SrcId</th>
<th>DstId</th>
<th>Property (E)</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>7</td>
<td>Collaborator</td>
</tr>
<tr>
<td>5</td>
<td>3</td>
<td>Advisor</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
<td>Colleague</td>
</tr>
<tr>
<td>5</td>
<td>7</td>
<td>PI</td>
</tr>
</tbody>
</table>
val userGraph: Graph[((String, String), String)]

// Assume the SparkContext has already been constructed
val sc: SparkContext
// Create an RDD for the vertices
val users: RDD[((VertexId, (String, String)))] =
  sc.parallelize(Array((3L, ("rxin", "student")), (7L, ("jgonzal", "postdoc")),
                     (5L, ("franklin", "prof")), (2L, ("istoica", "prof"))))

// Create an RDD for edges
val relationships: RDD[Edge[String]] =
  sc.parallelize(Array(Edge(3L, 7L, "collab"),
                     Edge(5L, 3L, "advisor"),
                     Edge(2L, 5L, "colleague"), Edge(5L, 7L, "pi")))

// Define a default user in case there are relationship with missing user
val defaultUser = ("John Doe", "Missing")
// Build the initial Graph
val graph = Graph(users, relationships, defaultUser)
Simple Operations

```scala
val graph: Graph[(String, String), String] // Constructed from above

// Count all users which are postdocs
graph.vertices.filter { case (id, (name, pos)) => pos == "postdoc" }.count

// Count all the edges where src > dst
graph.edges.filter(e => e.srcId > e.dstId).count
```

[https://spark.apache.org/docs/latest/graphx-programming-guide.html#summary-list-of-operators](https://spark.apache.org/docs/latest/graphx-programming-guide.html#summary-list-of-operators)
How it Parallelizes

Edge Cut

Vertex Cut
Built in Graph Algorithms

- Label Propagation
- (Strongly) Connected Components
- Triangle Counting
- PageRank
- SVD++
Implementation of BFS

def bfs(VD, ED)(graph: Graph[VD, ED], src: VertexId, dst: VertexId): Seq[VertexId] = {
  if (src == dst) return List(src)

  // The attribute of each vertex is (dist from src, id of vertex with dist-1)
  val g: Graph[Int, VertexId], ED) =
  graph.mapVertices((id, _) => (if (id == src) 0 else Int.MaxValue, 0)).cache()

  // Traverse forward from src
  val dstAttr = (Int.MaxValue, 0)
  while (dstAttr._2 == Int.MaxValue) {
    val msgs = g.aggregateMessages((Int, VertexId))(c => if (c.srcAttr._1 == dstAttr._1) {
      c.sendToOut((c.srcAttr._1 + 1, c.srcId))
    },
    (a, b) => if (a._1 < b._1) a else b).cache()
    if (msgs.count == 0) return List.empty

    g = g.ops.joinVertices(msgs) {
      (id, oldAttr, newAttr) =>
        if (newAttr._1 < oldAttr._1) newAttr else oldAttr
    }.cache()

    dstAttr = g.vertices.filter(_._1 == dst).first()._2
  }

  // Traverse backward from dst and collect the path
  var path: List[VertexId] = dstAttr._2 :: dst :: Nil
  while (path.head != src) {
    path = g.vertices.filter(_._1 == path.head).first()._2 :: path
  }

  path
}
Future

- New API in Spark 2, the Dataset
  - Based off RDDs
- GraphFrames
  - WORKS WITH PYTHON!