CLOUD COMPUTING APPLICATIONS

Graphs: OLAP - GraphFrames

Prof. Reza Farivar

Spark GraphFrames

- Spark 1.3 onwards move away from RDD and more towards DataFrames
- More user friendly, simple
  - GraphX uses lower-level RDD-based API (vs. DataFrames)
- Simplified API
  - Python interface
  - DataFrames
    - Can benefit from DataFrame optimizations
- Support motif finding for structural pattern searches
- Easier to do optimization under the hood

Comparison of Spark GraphX and GraphFrames

	GraphFrames	GraphX
Core APIs	Scala, Python (Java?)	Scala only
Programming Abstraction	DataFrames	RDDs
Use Cases	Algorithms, Queries, Motif Finding	Algorithms
VertexIds	Any type (in Catalyst)	Long
Vertex/edge attributes	Any number of DataFrame columns	Any type (VD,ED)
Return Types	GraphFrames/DataFrames	Graph [VD,ED] or RDD[Long, VD]

GraphX compatibility

- Easy to convert between GraphX and graphFrame

- GraphFrame → GraphX

```
val g: GraphFrame = ….
val gx: Graph [Row, Row] = g.toGraphX
```

- GraphX → GraphFrame

```
val g2: GraphFrame = GraphFrame.fromGraphX(gx)
```

Graph Construction

- To construct a graphFrame, you need two DataFrames
  - Vertices
    - 1 vertex per row
    - id: column with unique id
  - Edges
    - 1 edge per row
    - src, dst columns using ids from vertices.id
  - **val** g **= GraphFrame**(v, e)
- Any Spark method to save and load DataFrames can be used
  - vertices = sqlContext.read.parquet(…)
  - vertices.write.parquet(…)

id	City	State
"JFK"	"New York"	NY
"SEA"	"Seattle"	WA

src	dst	delay	tripID
"JFK"	"SEA"	45	1058923
"DFW"	"SFO"	-7	4100224

Graph Algorithms

- Find important vertices
  - PageRank *
    - g.pageRank(resetProbability=0.15, tol=0.01)
    - g.pageRank(resetProbability=0.15, maxIter=10)
- Find paths between sets of vertices
  - Breadth-first search (BFS) ¤
    - paths = g.bfs("name = 'Esther'", "age < 32")
  - Shortest path *
    - g.shortestPaths(landmarks=["a", "d"])

* *Mostly wrappers around Spark GraphX algorithms*

¤ *Some algorithms implemented using DataFrames*

Graph Algorithms

- Find groups of vertices (components, communities)
  - Connected components *
  - Strongly connected components *
  - Label Propagation Algorithm (LPA) *
- Other
  - Triangle Counting ¤
  - SVDPlusPlus *
- Pregel

* *Mostly wrappers around Spark GraphX algorithms*

¤ *Some algorithms implemented using DataFrames*

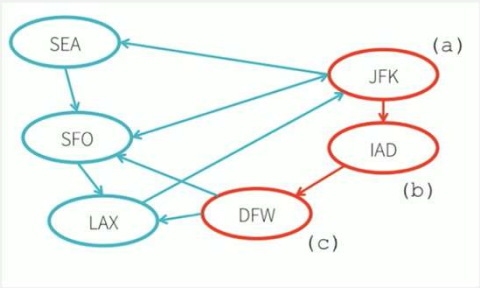
Simple Queries

- SQL queries on vertices & edges
- E.g. what trips are most likely to have significant delays?
  - Display(tripGraph.edges.groupBy("src", "dst").avg("delay").sort(desc("avg(delay)")))
- Graph Queries
  - Vertex degrees
    - Number of edges per vertex (incoming, outgoing, total)

Motif finding

- Search for structural patterns within a graph

```
motifs = g.find("(a)-[e1]->(b);
                 (b)-[e2]->(c);
                 !(c)-[]->(a)")
motifs.filter(“e1.delay > 20 and b.id = ‘SFO’”)
```



- Graphframes find() syntax

```
motifs = g.find("(a)-[e]->(b); (b)-[e2]->(a)")
motifs.filter("b.age > 30").show()
```

- filter

```
g1 = g.filterVertices("age > 30").filterEdges("relationship = 'friend’”)
```

Implementing Algorithms

- Method 1: DataFrame & GraphFrame operations
  - Motif finding
    - Series of DataFrame joins
  - Use other pre-packaged algorithms
- Method 2: Message passing
  - Send messages between vertices, and aggregate messages for each vertex
- Method 3: Pregel

GraphFrames AggregateMessages

- Send messages between vertices, and aggregate messages for each vertex
- aggregateMessages()
  - Similar to GraphX
  - Specify messages and aggregation using DataFrame expressions
- Joins: Join message aggregates with the original graph.
  - GraphFrames rely on DataFrame joins

```
from pyspark.sql.functions import sum as sqlsum
from graphframes.lib import AggregateMessages as AM

# For each user, sum the ages of the adjacent users.
msgToSrc = AM.dst["age"]
msgToDst = AM.src["age"]
agg = g.aggregateMessages( sqlsum(AM.msg).alias("summedAges"),
sendToSrc=msgToSrc, sendToDst=msgToDst)
agg.show()
```

Pregel in GraphFrames

- When a run starts, it expands the vertices DataFrame using column expressions defined by [[withVertexColumn]].
- Example code for PageRank

```
val edges = ...

val vertices = GraphFrame.fromEdges(edges).outDegrees.cache()

val numVertices = vertices.count()

val graph = GraphFrame(vertices, edges)

val alpha = 0.15

val ranks = graph.pregel.withVertexColumn(
                    "rank", lit(1.0 / numVertices),
                     coalesce(Pregel.msg, lit(0.0)) * (1.0 - alpha) + alpha / numVertices
 )

 .sendMsgToDst(Pregel.src("rank ") / Pregel.src("outDegree"))

 .aggMsgs(sum(Pregel.msg))

 .run()}
```

Creates a Column of literal value.

Returns the first column that is not null