Objective

The goal of this section is to work with GraphX. GraphX is Apache Spark's API for parallel computation on graphs and graph-like datasets.

Prerequisites

Follow directions from the Spark section to deploy an Apache Spark cluster on Amazon EMR. Launch Jupyter as usual, except modify the last line of `launchJupyter.sh` to read:

```
pyspark --packages graphframes:graphframes:0.6.0-spark2.3-s_2.11
```

Datasets

Source Data:

1. OpenFlights: Airport, airline and route data
2. United States Department of Transportation: Bureau of Transportation Statistics (TranStats)
   *Note, the data used here was extracted from the US DOT:BTS between 7/1/2017 and 9/30/2017**

Ingest & Prepare Data

Load and transform the flights and airports data from S3. This dataset uses International Air Transport Association's (IATA) location identifiers, a unique 3-letter code to identify airports.

```
# Obtain departure Delays data
delay = spark.read.csv("s3n://csed516/Flights/flights.csv", header="true",inferSchema="true")
delay.registerTempTable("departureDelays")
delay.cache()

# Obtain airports dataset
airports = spark.read.csv("s3n://csed516/Flights/airports.csv", header="true", inferSchema="true")
airports.registerTempTable("airports")


# create trip IATA codes table
tripIATA = sqlContext.sql("select distinct IATA from(select distinct ORIGIN as iata from departureDelays
union all select distinct DEST as iata from departureDelays) a")

tripIATA.registerTempTable("tripIATA")


#merge airport data with tripIATA data
airports = sqlContext.sql("select f.IATA_CODE as IATA, f.City, f.State, f.Country from airports f
join tripIATA t on t.IATA = f.IATA_CODE")
airports.registerTempTable("airports")
airports.cache()


# Build `departureDelays_geo` DataFrame
# Obtain key attributes such as Date of flight, delays, distance,
# and airport information (Origin, Destination)

departureDelays_geo = sqlContext.sql("select cast(f.FL_DATE as int) as tripid,
cast(concat(concat(concat(concat(concat(concat('2017-',
```

```
concat(concat(substr(cast(f.FL_DATE as string), 1, 2), '-')),
substr(cast(f.FL_DATE as string), 3, 2)), ' '), substr(cast(f.FL_DATE as string), 5, 2)), ':'),
substr(cast(f.FL_DATE as string), 7, 2)), ':00') as timestamp) as `localdate`,
cast(f.DEP_DELAY as int) as delay, f.ORIGIN as src, f.DEST as dst, o.city as city_src,
d.city as city_dst, o.state as state_src, d.state as state_dst
from departuredelays f join airports o on o.IATA = f.ORIGIN join airports d on d.IATA = f.DEST")

departureDelays_geo.registerTempTable("departureDelays_geo")

# Cache and Count
departureDelays_geo.cache()
departureDelays_geo.count()
```

Build Graph

Now we build the graph. We need to have two dataframes: one for vertices and one for edges. To build a graph from these two dataframes, we need to rename some columns to what the GraphX API expects:

1. Rename the IATA airport code to `id` in the vertices RDD
2. Rename the source and destination airports to `src` and `dst` for the edges RDD

```
from pyspark.sql.functions import *
from graphframes import *

tripVertices = airports.withColumnRenamed("IATA", "id").distinct()
tripEdges = departureDelays_geo.select("tripid", "delay", "src", "dst", "city_dst", "state_dst")

# Cache Vertices and Edges
tripEdges.cache()
tripVertices.cache()

#lets look at the vertices and edges
tripVertices.show()
tripEdges.show()

#build a graph!
tripGraph = GraphFrame(tripVertices, tripEdges)

print tripGraph

#subset columns to build the smaller datastructure.
tripEdgesPrime = departureDelays_geo.select("tripid", "delay", "src", "dst")
tripGraphPrime = GraphFrame(tripVertices, tripEdgesPrime)
```

Simple Queries

Having constructed the graph, we can now run some queries on it:

```
#number of vertices and edges?
print "Airports: %d" % tripGraph.vertices.count()
print "Trips: %d" % tripGraph.edges.count()


#Finding the longest Delay
longestDelay = tripGraph.edges.groupBy().max("delay")
longestDelay.show()

print "On-time / Early Flights: %d" % tripGraph.edges.filter("delay <= 0").count()
print "Delayed Flights: %d" % tripGraph.edges.filter("delay > 0").count()
```

```
# Degrees
tripGraph.degrees.sort(desc("degree")).show()


tripGraph.edges.filter("src = 'SFO' and delay > 0").groupBy("src", "dst").avg("delay").
   sort(desc("avg(delay)")).show()


# States with the longest cumulative delays (with individual delays > 100 minutes) (origin: Seattle)
tripGraph.edges.filter("src = 'SEA' and delay > 100").show()
```

References:

1. https://graphframes.github.io/graphframes/docs/_site/quick-start.html
2. https://spark.apache.org/graphx/