Lab Distributed Big Data Analytics

Worksheet-3: **Spark GraphX and Spark SQL operations**

Dr. Hajira Jabeen, **Gezim Sejdiu**, **Prof. Dr. Jens Lehmann**

May 9, 2017

In this lab we are going to perform basic Spark SQL and Spark GraphX operations (described on "Spark Fundamentals II"). Spark SQL provides the ability to write sql-like queries which can run on Spark. Their main abstraction is SchemaRDD which allows creating an RDD in which you can run SQL, HiveQL, and Scala.
GraphX is the new Spark API for graphs and graph-parallel computation. At a high-level, GraphX extends the Spark RDD abstraction by introducing the Resilient Distributed Property Graph: a directed multigraph with properties attached to each vertex and edge.
In this lab, you will use SQL and GraphX to find out the subject distribution over nt file. The purpose is to demonstrate how to use the Spark SQL and GraphX libraries on Spark.

IN CLASS

1. Spark SQL operations
    a. After a file (page_links_simple.nt.bz2) have been downloaded, unzipped, and uploaded on HDFS under /yourname folder you may need to create an RDD out of this file.
    b. First create a Scala class `Triple` containing information about a triple read from a file, which will be used as schema. Since the data is going to be type of .nt file which inside contains rows of triples in format `<subject> <predicate> <object>` we may need to transform this data into a different format of representation. Hint: Use `map` function.
    c. Create an RDD of a `Triple` object
    d. Use the filter transformation to return a new RDD with a subset of the triples on the file by checking if the first row contains "#", which on .nt file represent a comment.
    e. Run SQL statements using sql method provided by the SQLContext:
        i. Taking all triples which are relxated to 'Category:Events'
        ii. Taking all triples for predicate 'author'.
        iii. Taking all triples authored by 'Andre_Engels'
        iv. Count how many time the specific subject have been used on our dataset.

f. Since result is considered to be SchemaRDD, every RDD operations work out-of-the-box. By using map function collect and print out Subjects and their frequencies.

---Solution---

```
object sqlab {

  def main(args: Array[String]) = {
    val input = "src/main/resources/rdf.nt" // args(0)

    val spark = SparkSession.builder
      .master("local[*]")
                                                    .config("spark.serializer",
"org.apache.spark.serializer.KryoSerializer")
      .appName("SparkSQL example")
      .getOrCreate()

    import spark.implicits._

    val tripleDF = spark.sparkContext.textFile(input)
      .map(TripleUtils.parsTriples)
      .toDF()

    tripleDF.show()

    //tripleDF.collect().foreach(println(_))

    tripleDF.createOrReplaceTempView("triple")

    val    sqlText    =    "SELECT    *    from    triple    where    subject    =
'http://commons.dbpedia.org/resource/Category:Events'"
    val triplerelatedtoEvents = spark.sql(sqlText)

    triplerelatedtoEvents.collect().foreach(println(_))

     val subjectdistribution = spark.sql("select subject, count(*) from triple
group by subject")
    println("subjectdistribution:")
    subjectdistribution.collect().foreach(println(_))
  }
}
```

----

2. Spark GraphX operations
   a. After a file (page_links_simple.nt.bz2) have been downloaded, unzipped, and uploaded on HDFS under /yourname folder you may need to create an RDD out of this file.
   b. First create a Scala class `Triple` containing information about a triple read from a file. Since the data is going to be type of .nt file which inside contains rows of triples in format `<subject> <predicate> <object>` we may need to transform this data into a different format of representation. Hint: Use `map` function.
   c. Use the filter transformation to return a new RDD with a subset of the triples on the file by checking if the first row contains "#", which on .nt file represent a comment.
   d. Perform these operations in order to transform your data into GraphX
      i. Generate vertices by combining (Subject, Object) as VertexId and their value.x
      ii. Create Edges by using subject as a key to join within vertices and generate Edge into format (s_index, obj_index, predicate)
   e. Compute connected components for triples containing "author" as a predicate.
   f. Compute triangle count.
   g. List top 5 connected component by applying pagerank over them.

---Solution---

```
object graphxlab {
  def main(args: Array[String]) = {
    val input = "src/main/resources/rdf.nt" // args(0)

    val spark = SparkSession.builder
      .master("local[*]")
                                    .config("spark.serializer",
"org.apache.spark.serializer.KryoSerializer")
      .appName("GraphX example")
      .getOrCreate()

    val tripleRDD = spark.sparkContext.textFile(input)
      .map(TripleUtils.parsTriples)

     val  tutleSubjectObject  =  tripleRDD.map  {  x  =>  (x.subject,
x.`object`) }
```

```
    type VertexId = Long

val     indexVertexID     =     (tripleRDD.map(_.subject)     union
tripleRDD.map(_.`object`)).distinct().zipWithIndex()

     val  vertices:  RDD[(VertexId,  String)]  =  indexVertexID.map(f
=> (f._2, f._1))

 val tuples = tripleRDD.keyBy(_.subject).join(indexVertexID).map(
      {
          case  (k,  (tech.sda.arcana.spark.worksheet3.Triple(s,  p,
o), si)) => (o, (si, p))
      })

  val edges: RDD[Edge[String]] = tuples.join(indexVertexID).map({
      case (k, ((si, p), oi)) => Edge(si, oi, p)
    })

    val graph = Graph(vertices, edges)

    graph.vertices.collect().foreach(println(_))

    println("edges")
    graph.edges.collect().foreach(println(_))

         val   subrealsourse   =   graph.subgraph(t   =>   t.attr   ==
"http://commons.dbpedia.org/property/source")
    println("subrealsourse")
    subrealsourse.vertices.collect().foreach(println(_))

    val conncompo = subrealsourse.connectedComponents()

    val pageranl = graph.pageRank(0.0001)

val                      printoutrankedtriples                      =
pageranl.vertices.join(graph.vertices)
      .map({ case (k, (r, v)) => (k, r, v) })
      .sortBy(5 - _._2)

    println("printoutrankedtriples")
    printoutrankedtriples.take(5).foreach(println(_))
  }
}
```

AT HOME

1. Read and explore
    a. Spark SQL, DataFrames and Datasets Guide
    b. GraphX Programming Guide
2. RDF Class Distribution - using Spark SQL - count the usage of respective classes of a RDF dataset.
   Hint: Class fulfils the rule(`?predicate = rdf:type && ?object.isIRI()`).
    a. Read the nt file into an RDD of triples.
    b. Apply map function for separating triples into (Subject, Predicate, Object)
    c. Apply filter transformation for defining the respective classes.
    d. Count the frequencies of Object by using sql statement
    e. Return the top 100 classes used in the dataset.
3. Using GraphX To Analyze a Real Graph
    a. Count the number of vertices and edges in the graph
    b. How many resources are on your graph?
    c. What is the max in-degree of this graph?
    d. Which triple are related to 'Category:Events'
    e. Run Pagerank for 50 iterations.
    f. Compute similarity between two nodes - using Spark GraphX
        i. Apply different similarity measures
            1. Jaccard similarity
            2. Edit distance
4. Further readings
    a. Spark SQL: Relational Data Processing in Spark
    b. Shark: SQL and Rich Analytics at Scale
    c. GraphX: Graph Processing in a Distributed Dataflow Framework