YOTABITES

Big Data Summit KC 2017

SPARK WORKSHOP

Yotabites Consulting **bigdata@yotabites.com**

Flat File Processing

1.	**Getting Started with Apache Spark**
1.1	Creating SparkContext ***sc***

1.2	Import necessary Packages
	`import sys`
	`import os`
	`import seaborn as sns`
	`import matplotlib.pyplot as plt`
	`import pandas as pd`
	`% matplotlib inline`
	`plt.rcParams["figure.figsize"] = [16,9]`
	`from pyspark import SparkContext`
	`from pyspark.streaming import StreamingContext`
	`import warnings`
	`warnings.filterwarnings("ignore")`
	`from operator import add`
	`import random`
	`from __future__ import print_function`
	`from IPython.display import Image,HTML,display`
	`from IPython.core.display import *`
	`display(HTML("<style>.container { width:75% !important; }</style>"))`

2.	**Creation of rdd and Loading a file from HDFS or S3**

2.1	Create a RDD
	`rdd = sc.textFile("/tmp/meetup/meetup.csv")`
	`rdd.count()`

2.2	Another way to create an rdd `rdd1 = sc.parallelize([1,2,3,4,5])`
2.3	Display all the elements in the rdd `ondriver = rdd.collect()`

2.4	Display desired number of elements
	`prntSTR=rdd.take(5)`
	`print(pd.DataFrame(prntSTR))`

3.	**Transformations on the RDD**

3.1	Extract all the headers
	`pairRDD=rdd.map(lambda x: (x,x.split(",")[19]))`
	`pairRDD.first()`

3.2	Filter out all the rows whose state is kansas
	`filterRDD=pairRDD.filter(lambda pair: pair[1]=="KS")`
	`filterRDD.count()`

3.3	A flatMap on rdd
	`flatMapRDD=rdd.map(lambda x: x.split(",")).flatMap(lambda x:x)`
	`flatMapRDD.first()`

3.4	Word count on the rdd
	`wordcountrdd = flatMapRDD\`
	`.map(lambda x: (x,1))\`
	`.reduceByKey(lambda x,y: x+y)`
	`wordcountrdd.take(5)`
	Note: To display a limited output use take(n), n is the number of desired outputs

3.5	RDD Lineage `wordcountrdd.toDebugString()`
3.6	Sort the output by Value `wordcountrdd.sortBy(lambda value: value[1], ascending=False).take(5)`

3.7	Estimating Pi using the Monte Carlo Method
	`import random`
	`def inside(p):`
	`  x, y = random.random(), random.random()`
	`  return x*x + y*y < 1`
	`NUM_SAMPLES=1000000`
	`count = sc.parallelize(range(0, NUM_SAMPLES)) \`
	`  .filter(inside).count()`
	`print("Pi is roughly %f" % (4.0 * count / NUM_SAMPLES))`

4.	**Let's perform some Actions on the rdd**
4.1	CountbyKey action on rdd `wordcountrdd.map(lambda line: (line,1)).countByKey()`
4.2	Return the first element in this RDD `wordcountrdd.first()`

4.3	TakeOrdered(n) - Display number of desired elements in sequential order `wordcountrdd.takeOrdered(5)`
4.3	Output a Python RDD of key-value pairs (of form RDD[(K, V)]) to any Hadoop file system `wordcountrdd.saveAsSequenceFile("/tmp/bdkc_pyspark/bdkcSampleSequenceOutput")`
4.4	Save this RDD as a text file, using string representations of elements `wordcountrdd.saveAsTextFile("/tmp/bdkc_pyspark/bdkcSampleTextOutput")`
5.	**DATAFRAMES** A DataFrame is a Dataset organized into named columns. It is conceptually equivalent to a table in a relational database or a dataframe in R/Python, but with richer optimizations under the hood.

5.1	Load a CSV file and returns the result as a DataFrame
	`df = spark.read.csv("path")`
	
	Load a JSON file and returns the result as a DataFrame
	`jsondf=spark.read.format('json').load('/tmp/bdkc_pyspark/satori_data/')`

5.2	Print the schema in a tree format `df.printSchema()`
5.3	Displays the content of the DataFrame `df.show()`

5.4	Filter and select operations on Dataframe
	`df.select(["group.group_state","event.event_name"])\`
	`  .filter(df.group.group_state == "KS")\`
	`  .show()`

5.5	Number of meetups in each city `df.groupby("group.group_city").count().orderBy("count",ascending=False).show()`

5.6	Number of members whose response was yes in each state
	`df.filter(df.response == 'yes')\`
	`  .groupby(["group.group_state",'response'])\`
	`  .count()\`
	`  .show()`

5.7	Most Number of responses by each state
	`dataframe_df.filter(dataframe_df.group.group_state != 'null')\`
	`.groupby(dataframe_df['group.group_state'],dataframe_df['response'])\`
	`  .count().orderBy("count",ascending=False)\`
	`  .show()`

5.8	Use Sql to find query the dataframe
	`df.createOrReplaceTempView("df")`
	`spark.sql("select event_time, count(*) as count \`
	`  from df \`
	`  group by event_time \`
	`  order by count desc")\`
	`  .show()`

RDBMS

1.	Loading data from a JDBC source.

1.1	Read data from Mysql.
	`jdbc_tblemployees = spark.read \`
	`.format("jdbc") \`
	`.option("url", "jdbc:mysql://172.31.89.20:3306/employee") \`
	`.option("dbtable", "tblemployees") \`
	`.option("user", "spark") \`
	`.option("password", "spark") \`
	`.option("driver","com.mysql.jdbc.Driver") \`
	`.load()`
	
	`jdbc_db.printSchema()`

1.2	Read another table tblpayemployeeparamdetails from dbo database
	`jdbc_employeepay = spark.read \`
	`.format("jdbc") \`
	`.option("url", "jdbc:mysql://172.31.89.20:3306/employee") \`
	`.option("dbtable", "tblpayemployeeparamdetails") \`
	`.option("user", "spark") \`
	`.option("password", "spark") \`
	`.option("driver","com.mysql.jdbc.Driver") \`
	`.load()`
	
	`jdbc_user.printSchema()`

2.	**Joining the above dataframes with inner and leftsemi joins**

2.1	Join the above two tables
	`joined_df = jdbc_tblemployees.join(jdbc_employeepay,jdbc_tblemployees['EmployeeNumber'] == jdbc_employeepay['EmployeeNumber'],"inner").drop(jdbc_tblemployees['EmployeeNumber'])`
	
	`joined_df.printSchema()`

2.2	leftsemi join the above two tables
	`outer_join = jdbc_employeepay.join(jdbc_tblemployees, jdbc_tblemployees['EmployeeNumber'] == jdbc_employeepay['EmployeeNumber'],'leftsemi')\ .drop(jdbc_employeepay['Employeenumber'])`
	
	`outer_join.printSchema()`

3.	**Querying on Joined Dataframes**

3.1	Which department is earning the most `joined_df.groupBy("DepartmentCode").agg({'Amount':"mean"}).orderBy("avg(Amount)",ascending=False).show()`
3.2	Find Average Salaries of Employees `joined_df.groupby("EmployeeNumber").agg({'Amount':"mean"}).orderBy("avg(Amount)",ascending=False).show()`

3.3	Find Average Salaries of Employees in SQL
	`joined_df.createOrReplaceTempView("table")`
	`spark.sql("select EmployeeNumber, avg(Amount) as avg from table group by\ EmployeeNumber,Amount order by avg desc ").show()`

3.4	Another SQL query `spark.sql("select LocationType, Sex, count(*) as count from table group by \  LocationType,Sex order by count desc").show()`

SPARK STREAMING

1.	**Spark Streaming**
1.1	Spark StreamingContext `ssc = StreamingContext(sc, 1)`

1.2	WordCount Spark Streaming
	
	`lines = ssc.textFileStream("/tmp/bdkc_pyspark/test.txt")`
	`counts = lines.flatMap(lambda line: line.split(" "))\`
	`.map(lambda x: (x, 1))\`
	`.reduceByKey(lambda a, b: a+b)`
	`counts.pprint()`
	`counts.saveAsTextFiles("/tmp/bdkc_pyspark/output.txt")`
	`ssc.start()`
	`ssc.awaitTermination(10)`
	`ssc.stop()`