CS 348 - PSO Week 12

Hadoop and Spark!

Overview

You need to go through Project 3 Step 1 until Basic commands for HDFS. Ignore Dataset section because we will be using a different dataset for PSO.

We will be using Hadoop for file handling and Spark for processing tasks! Learn about Hadoop and Spark with their documentation as well as `.ppt` files provided.

Environment

Refer Project 3 Step 1 document.

Basic commands for HDFS

Refer Project 3 Step 1 document.

Dataset

We are assuming that you went through Step 1 document until Basic commands for HDFS and you are in `jacobi00` now.

For PSO, we will be using data from American Statistical Association:

http://stat-computing.org/dataexpo/2009/the-data.html.

CAUTION: There are several datasets listed on the webpage. Be sure you get the `2006.csv` dataset.

Download the data:

```
$ wget http://stat-computing.org/dataexpo/2009/2006.csv.bz2
```

Unzip the folder:

```
$ bzip2 -d 2006.csv.bz2
```

View the contents of the extracted file (first few entries since dataset is large):

```
$ head 2006.csv
```

Let us get rid of the header:

```
$ tail -n +2 2006.csv > 2006_noheader.csv
```

View the contents of the extracted file (first few entries since dataset is large):

```
$ wc -l 2006_noheader.csv
```

To list the files in long format i.e. with an index number, owner name, group name, size, and permissions.

```
$ ls -lh 2006_noheader.csv
```

Now let us make new directories and put the files in HDFS (NOTE: `$USER` means your Purdue Career Account Username.),

```
$ hdfs dfs -mkdir /user/$USER/rita
$ hdfs dfs -mkdir /user/$USER/rita/psoinput
$ hdfs dfs -put ./2006_noheader.csv /user/$USER/rita/psoinput
$ hdfs dfs -ls /user/$USER/rita/psoinput
$ hdfs dfs -cat /user/$USER/rita/psoinput/2006_noheader.csv | less
```

Let us see how the file is distributed as blocks and stored in several locations.

```
$ hdfs fsck /user/$USER/rita/psoinput/2006_noheader.csv -files -blocks -locations
```

This will display the distributions of the file and their status along with other properties. The data has been fragmented and moved to worker nodes. You can also see number of HDFS blocks.

For Spark, the master's address is specified in the `--master` parameter, in `YARN` mode the ResourceManager's address is picked up from the Hadoop configuration. Thus, the `--master` parameter is `yarn`. Let us start the Spark in `cluster` mode.

```
$ pyspark --master yarn
```

Let us load our data into Spark for processing. Note that the folder specified is retriving data from HDFS.

```
>>> text = sc.textFile("/user/$USER/rita/psoinput/2006_noheader.csv")
```

Lets get some basic information.

```
>>> text.count()
>>> text.getNumPartitions()
>>> text.first()
```

Spark creates logs for each operation. It is better to supress them and save space. So we can just get warnings instead of saving logs.

```
>>> sc.setLogLevel("WARN")
```

Finally, some `pyspark` queries.

1. Find all flights that left in January.

```
>>> withMonthCol = text.map(lambda line: [line.split(",")
    [1], line])

# Returns the count of the result
>>> withMonthCol.filter(lambda line: line[0] == "1").count
    ()

# Store the result in a variable
>>> selected = withMonthCol.filter(lambda line: line[0] ==
    "1").map(lambda line: line[1])

# You can get the type of the variable.
# It is RDD. Refer Spark documentation to learn more.
>>> type(selected)

# Since result is a large set, we can just see some 10
    items.
>>> selected.take(10)
```

2. Find unique departure airports.

```
>>> depAirports = text.map(lambda line: line.split(",")
    [16])

# Let us count the number of unique airports.
>>> depAirports.distinct().count()

>>> distDepAirports = depAirports.distinct().collect()

# Since result is a large set, let us get just first
    element.
>>> distDepAirports[0]
>>> distDepAirports[0].encode('ascii', 'ignore')
```

A `Pyspark` SQL query:

1. Which airports had flights that flew to TUL on January 20?

```
>>> from pyspark.sql import *
>>> from pyspark.sql.types import *

>>> splitLines= text.map(lambda l: l.split(",")).map(lambda p:
    (p[0], p[1], p[2], p[3], p[4], p[5], p[6], p[7], p[8], p
    [9], p[10], p[11], p[12], p[13], p[14], p[15], p[16], p
    [17], p[18], p[19], p[20], p[21], p[22], p[23], p[24], p
    [25], p[26], p[27], p[28].strip()))

>>> schemaString = "Year Month DayofMonth DayOfWeek DepTime
    CRSDepTime ArrTime CRSArrTime UniqueCarrier FlightNum
    TailNum ActualElapsedTime CRSElapsedTime AirTime ArrDelay
    DepDelay Origin Dest Distance TaxiIn TaxiOut Cancelled
    CancellationCode Diverted CarrierDelay WeatherDelay
    NASDelay SecurityDelay LateAircraftDelay"

>>> fields = [StructField(field_name, StringType(), True) for
    field_name in schemaString.split()]

>>> schema = StructType(fields)

>>> schemaDF = sqlContext.createDataFrame(splitLines, schema)

>>> schemaDF.registerTempTable("flights")

>>> query="select distinct flights.origin from flights where
    flights.month = 1 and flights.dayofmonth = 20 and flights.
    dest = 'TUL' order by flights.origin"

>>> results = sqlContext.sql(query)

>>> results.first()

>>> collected = results.collect()

>>> len(collected)

>>> print collected
```

2. Exercise: Which airports had flights that flew to JFK on June 15?