Report On SPARK STREAMING

Submitted by- Ketan Walia

Step 1: Input data

- Data is being generated automatically (a few times a minute) in the following directory /project/sensor*
- This stream of data provides data for 1000x1000 items (exhaustively)

==========

COMMAND FOR GETTING THE INPUT DATA FROM PROJECT FOLDER HDFS AND ONLINE PROCESSING:

spark-submit --master spark://ip-172-31-72-124:7077 hdfsnew_wc.py hdfs://ip-172-31-72-124:9000/project

==========

Step 2: Online processing

Task:

• Compute wind speed variability in the last two minutes (sliding) window – Use increases of 20 seconds for your window – You will need to take, for each coordinate (x,y) the MAX and MIN values in that window

• Save the result of algorithm for each window in HDFS – Store the variability of wind speed (i.e., MAX – MIN) for each coordinate, for each window in HDFS.

Solution:

For convenience, we have written Python code which takes the streaming input data from HDFS for a window of 90 seconds with sliding interval of 30 seconds each.

NOTE: As discussed with Professor since we are getting same wind speed value for both i.e maximum value and minimum value for each coordinate in a window of 90 seconds we are just considering the maximum value of wind speed for each coordinate in code.

The code captures the maximum value in the window for each coordinate and stores the generated file into HDFS in a folder named- getfile. The file is stored in- JSON format.

Please find below the code i.e hdfsnew_wc.py:

```
# Licensed to the Apache Software Foundation (ASF) under one or more
# contributor license agreements.  See the NOTICE file distributed with
# this work for additional information regarding copyright ownership.
# The ASF licenses this file to You under the Apache License, Version 2.0
# (the "License"); you may not use this file except in compliance with
# the License.  You may obtain a copy of the License at
#
#    http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.
#

"""
 Counts words in new text files created in the given directory
 Usage: hdfs_wordcount.py <directory>
   <directory> is the directory that Spark Streaming will use to find and read new text files.

 To run this on your local machine on directory `localdir`, run this example
    $ bin/spark-submit examples/src/main/python/streaming/hdfs_wordcount.py localdir

 Then create a text file in `localdir` and the words in the file will get counted.
"""



from __future__ import print_function
import sys
from pyspark import SparkContext
from pyspark.streaming import StreamingContext
from pyspark.sql import Row, SparkSession

def getSparkSessionInstance(sparkConf):
    if ('sparkSessionSingletonInstance' not in globals()):
        globals()['sparkSessionSingletonInstance'] = SparkSession\
            .builder\
            .config(conf=sparkConf)\
            .getOrCreate()
    return globals()['sparkSessionSingletonInstance']

if __name__ == "__main__":
    if len(sys.argv) != 2:
        print("Usage: hdfs_wordcount.py <directory>", file=sys.stderr)
        exit(-1)
```

```
sc = SparkContext(appName="PythonStreamingHDFSWordCount")
ssc = StreamingContext(sc, 30)
ssc.checkpoint("hdfs:///user/kw475/check")
lines = ssc.textFileStream(sys.argv[1])
words = lines.map(lambda line: line.split(","))\
            .map(lambda a: ((int(a[1]),int(a[2])),int(a[3])))\
            .reduceByKeyAndWindow(lambda a,b:[a]+[b],90,30)


# words.pprint()
def process(time, rdd):
    print("========= %s =========" % str(time))

    try:
        # Get the singleton instance of SparkSession
        spark = getSparkSessionInstance(rdd.context.getConf())

        # Convert RDD[String] to RDD[Row] to DataFrame
        rowRdd = rdd.map(lambda w: Row(word=w[0], val=w[1]))
             print(rowRdd.count())

             #row_df = rdd.toDF()
             # row.df_show()

             #print(rowRdd.collect())
        wordsDataFrame = spark.createDataFrame(rowRdd)

        # Creates a temporary view using the DataFrame.
        wordsDataFrame.createOrReplaceTempView("words")

        # Do word count on table using SQL and print it
        wordCountsDataFrame = \
           spark.sql("SELECT word, MAX(val)-Min(val) as var from words group by word")

        wordCountsDataFrame.show()
            # wordCountsDataFrame.toPandas().to_csv('hdfs://user/kw475/')
        wordCountsDataFrame.write.json("hdfs:///user/kw475/getfile")

    except Exception, e:
        print('=====================ERROR=========================')
        print (str(e))
        pass


words.foreachRDD(process)
ssc.start()
ssc.awaitTermination()
```

P.S: The Python Code is also attached along with this Report

The above code gives the following output:

```
17/05/08 22:35:51 INFO storage.BlockManagerInfo: Removed broadcast_18_piece0 on 172.31.72.124:3
17/05/08 22:35:51 INFO storage.BlockManagerInfo: Removed broadcast_18_piece0 on 172.31.66.239:4
+----------+---+
|      word|var|
+----------+---+
| [505,173]|-91|
| [711,401]| 84|
| [929,433]|-97|
| [492,382]|  2|
| [190,458]|  2|
| [561,355]|-25|
| [431,231]| 18|
| [764,628]| 57|
|   [9,585]| 19|
| [187,787]|  8|
| [782,560]| 99|
| [374,730]| 60|
| [674,854]| 80|
|[114,1056]| 90|
|  [686,42]|-79|
| [523,357]| 36|
| [940,550]| 40|
| [652,350]| 34|
| [877,401]|-23|
|[853,1043]|-98|
+----------+---+
only showing top 20 rows

17/05/08 22:35:51 INFO scheduler.JobScheduler: Finished job streaming job 1494282930000 ms.0 fr
17/05/08 22:35:51 INFO scheduler.JobScheduler: Total delay: 21.360 s for time 1494282930000 ms
17/05/08 22:35:51 INFO dstream.FileInputDStream: Cleared 0 old files that were older than 14942
17/05/08 22:35:51 INFO storage.BlockManagerInfo: Removed broadcast_17_piece0 on 172.31.72.124:3
17/05/08 22:35:51 INFO scheduler.JobGenerator: Checkpointing graph for time 1494282930000 ms
```

Step 3: Batch processing

Batch processing is under demand, i.e., we will execute MapReduce code for further data analysis.

TASK:

• Process the data in the output file in HDFS from Step 2

– This file should contain that variability of wind speed in each point for a number of time windows

SOLUTION:

The output file generated from previous code in Step-2 stores the data in a folder named- “getfile”. The files are stored in JSON format and there are around 200 files stored in the folder.

As a part of batch processing we have written the codes which will output the following statistics:

a) Maximum Wind Speed Value across all the coordinates
b) Minimum Wind Speed Value across all the coordinates
c) Average Wind Speed Value across all the coordinates

d) Standard Deviation of Wind Speed Value across all the coordinates

A) Please find below the code for generating Maximum Wind Speed Value across all the coordinates i..e max.py

```
================================================================================

from mrjob.job import MRJob
from mrjob.step import MRStep
import json

class MRMax(MRJob):

#       def step(self):
#               return [
#                       MRStep(mapper=self.mapper1,
#                               reducer=self.reducer1),
#                       MRStep(reducer=self.reducer2)
#               ]

        def mapper(self, _, line):
                lineJSON = json.loads(line)
                yield 'value',lineJSON["var"]

        def reducer(self, key, value):
                yield "max", max(value)


if __name__ == '__main__':
        MRMax.run()
```

The code is run using the following command executing a MAPREDUCE Job on the data. The result is also shown below:

Command for Max:

python max.py -r hadoop hdfs:///user/kw475/getfile/part*

Output:

```
Streaming final output from hdfs:///user/tb540/tmp/mrjob/max.tb540.20170508.165403.895494
utput...
"max"   100
Removing HDFS temp directory hdfs:///user/tb540/tmp/mrjob/max.tb540.20170508.165403.89549
```

B) Please find below the code for generating Minimum Wind Speed Value across all the coordinates i..e min.py

==========

```
from mrjob.job import MRJob
from mrjob.step import MRStep
import json

class MRMin(MRJob):

#       def step(self):
#               return [
#                       MRStep(mapper=self.mapper1,
#                               reducer=self.reducer1),
#                       MRStep(reducer=self.reducer2)
#               ]

        def mapper(self, _, line):
                lineJSON = json.loads(line)
                yield 'value',lineJSON["var"]

        def reducer(self, key, value):
                yield "max", min(value)


if __name__ == '__main__':
        MRMin.run()
```

The code is run using the following command executing a MAPREDUCE Job on the data. The result is also shown below:

Command for Min:

python min.py -r hadoop hdfs:///user/kw475/getfile/part*

Output:

```
                WRONG_REDUCE=0
treaming final output from hdfs:///user/tb540/tmp/mrjob/min.tb540.20170
tput...
min"    -98
emoving HDFS temp directory hdfs:///user/tb540/tmp/mrjob/min.tb540.2017
```

C) Please find below the code for generating Average Wind Speed Value across all the coordinates i..e avg.py

==========

```
from mrjob.job import MRJob
```

```
from mrjob.step import MRStep
import json

class MRAvg(MRJob):

#       def step(self):
#               return [
#                       MRStep(mapper=self.mapper1,
#                               reducer=self.reducer1),
#                       MRStep(reducer=self.reducer2)
#               ]

        def mapper(self, _, line):
                lineJSON = json.loads(line)
                yield 'value',lineJSON["var"]

        def reducer(self, key, value):
                n = 0
                s = 0
                for i in value:
                        n += 1
                        s +=i
                avg = float(s)/float(n)
                yield "average", avg


if __name__ == '__main__':
        MRAvg.run()
```

==========

The code is run using the following command executing a MAPREDUCE Job on the data. The result is also shown below:

Command for Average:

python avg.py -r hadoop hdfs:///user/kw475/getfile/part*

Output:

```
                WRONG_REDUCE=0
Streaming final output from hdfs:///user/tb540/tmp/mrjob/avg.tb540.20170508.1648
37.547449/output...
"average"       0.9813836363636363
Removing HDFS temp directory hdfs:///user/tb540/tmp/mrjob/avg.tb540.20170508.164
```

D) Please find below the code for generating Standard Deviation of Wind Speed Value across all the coordinates i.e std.py

```
from mrjob.job import MRJob
from mrjob.step import MRStep
import json
```

```
class MRStd(MRJob):

        def mapper(self, _, line):
                lineJSON = json.loads(line)
                yield 'value',lineJSON["var"]

        def reducer(self, key, value):
                n = 0
                s = 0
                s2 = 0
                for i in value:
                        n += 1
                        s +=i
                        s2 += i**2
                std = (s2/n - (s/n) **2) ** 0.5
                yield "std", std


if __name__ == '__main__':
        MRStd.run()
```

The code is run using the following command executing a MAPREDUCE Job on the data. The result is also shown below:

Command for Standard deviation:

python std.py -r hadoop hdfs:///user/kw475/getfile/part*

Output:

```
Streaming final output from hdfs:///user/tb540/tmp/mrjob/std.tb540.20170508.165124.07320
utput...
"std"   57.367238037053866
Removing HDFS temp directory hdfs:///user/tb540/tmp/mrjob/std.tb540.20170508.165124.0732
```

• Generate a heat map with the average wind speed variability for each coordinate (x,y) – Only a single heat map is expected

To generate the heatmap we combined all the 200 files into a single file using the below code. The code is – combine.py

Please find below the code for combining all the files i.e combine.py

```
from mrjob.job import MRJob
from mrjob.step import MRStep
import json

class Combine(MRJob):
```

```
#       def step(self):
#               return [
#                       MRStep(mapper=self.mapper1,
#                               reducer=self.reducer1),
#                       MRStep(reducer=self.reducer2)
#               ]

        def mapper(self, _, line):
                lineJSON = json.loads(line)
                cord = (lineJSON["word"]["_1"], lineJSON["word"]["_2"])
                yield cord ,lineJSON["var"]

        def reducer(self, key, value):
                yield key, max(value)


if __name__ == '__main__':
        Combine.run()
```

Combine.py : Combine all the files in /getfiles into one single .txt file

Command for Combine:

python combine.py -r hadoop hdfs:///user/kw475/getfile/part* > output.txt

Output Snapshot:

```
[366, 219]      14
[366, 21]       73
[366, 220]      97
[366, 221]      -69
[366, 222]      63
[366, 223]      -57
[366, 224]      24
[366, 225]      -53
[366, 226]      86
[366, 227]      31
[366, 228]      100
[366, 229]      -8
[366, 22]       51
[366, 230]      -88
[366, 231]      -12
[366, 232]      -79
[366, 233]      31
[366, 234]      -26
[366, 235]      -3
[366, 236]      -48
[366, 237]      -66
[366, 238]      -24
[366, 239]      85
[366, 23]       49
[366, 240]      -95
[366, 241]      -12
[366, 242]      87
[366, 243]      77
[366, 244]      29
[366, 245]      64
[366, 246]      -46
[366, 247]      -52
[366, 248]      -64
[366, 249]      58
[366, 24]       -72
[366, 250]      -2
[366, 251]      -8
[366, 252]      -8
[366, 253]      7
[366, 254]      88
[366, 255]      -89
[366, 256]      22
[366, 257]      -45
[366, 258]      -76
[366, 259]      -23
[366, 25]       -33
```

Generating HEATMAP:

The file- output.txt is transferred into our Windows machine using WinSCP to generate heatmap

Please find below the code for generating heatmap for the above file i.e project_heatmap.py

```
import numpy as np
import matplotlib.pyplot as plt

f = open('output.txt', 'r')
x = f.read().splitlines()
f.close()
```

```
array1 = np.zeros((1000, 1100))

for line in x:
    line = line.replace("[","").replace(",","").replace("]","")
    line = line.split()
    x = int(line[0]) - 1
    y = int(line[1]) - 1
    val = int(line[2])
    array1[x,y] = val

print(array1)

plt.imshow(array1.astype(float), cmap='hot', interpolation='nearest')
plt.show()
```

The above code gives the following HeatMap: