3.S3_HDFS_Parquet

October 11, 2018

0.1 Moving Data Around

- When you need to process a lot of data, a big part of the execution time of your program is devoted to moving the data between storage units.

- This notebook is **NOT** intended to be run on your personal computer. It is intended to show you the main steps needed when processing a large file on a multi-computer cluster. The main emphasis here is on the **Wall time** required for different operations.

0.2 Some terminology

0.2.1 Data Serialization.

- Data in memory is usually stored in **data structures** that allow for fast manipulation. This often means that the amount of memory needed is significantly larger than the amount that would be needed to store the same data on disk.
- We say that the data on disk is **serial** and the data stored in data structures is **deserialized**

0.2.2 AWS-EMR

We demonstrate the movement of data on "Amazon Web Services" (AWS) "Elastic Map Reduce" (EMR).

Recall the slide about data organization in the video "a short history of affordable massive computing" In the next figure we add to that slide the way it fits within AWS-EMR.

Three file systems:

- **S3:** long term persistent memory.
- **Head Node:** standard Unix file system.
- **HDFS:** distributed file system on the workers.

0.3 Reading a CSV file from S3

We start with a CSV file on S3, which we move through the head node to HDFS and than parse into a spark RDD.

0.3.1 Moving a file from S3 to the head node

```
In [1]: <div class="mark">
        %cd /mnt/workspace/edX-Micro-Master-in-Data-Science/big-data-analytics-using-spark/note
        !ls</div><i class="fa fa-lightbulb-o "></i>
```

```
/mnt/workspace/edX-Micro-Master-in-Data-Science/big-data-analytics-using-spark/notebooks/Data
kmeans_data.txt  OldData  people.json  Weather
```

```
In [2]: #create directory to hold data one node
        !mkdir Weather
        %cd Weather/
```

```
mkdir: cannot create directory Weather: File exists
/mnt/workspace/edX-Micro-Master-in-Data-Science/big-data-analytics-using-spark/notebooks/Data/W
```

```
In [4]: #list files on S3
        !aws s3 ls s3://dse-weather/ALL.csv.gz
        #compressed file is about 1.5GB
```

```
2016-02-09 04:09:49 1511481989 ALL.csv.gz
```

```
In [5]: %%time
        #copy file from S3
        !aws s3 cp s3://dse-weather/ALL.csv.gz ./ALL.csv.gz
```

```
download: s3://dse-weather/ALL.csv.gz to ./ALL.csv.gz
CPU times: user 956 ms, sys: 280 ms, total: 1.24 s
Wall time: 11.9 s
```

```
In [6]: %%time
        #unompress file
        !rm ALL.csv
        !gunzip ALL.csv.gz
```

```
CPU times: user 720 ms, sys: 260 ms, total: 980 ms
Wall time: 54.5 s
```

```
In [7]: !ls -l ALL.csv
        # About 7.7 GB
```

```
-rw-rw-r-- 1 hadoop hadoop 7668890105 Feb  9  2016 ALL.csv
```

```
In [8]: !head -2 ALL.csv
```

```
station,year,measurement,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,2
ASN00054128,DAPR,1969,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,
```

0.4 Distribute file into HDFS

copy file from the head-node file system to HDFS

```
In [2]: %%time
        !hadoop fs -mkdir /weather
```

```
/bin/sh: hadoop: command not found
CPU times: user 2.72 ms, sys: 7.76 ms, total: 10.5 ms
Wall time: 125 ms
```

```
In [12]: %%time
         #create a data directory on hdfs
         !hadoop fs -copyFromLocal ALL.csv hdfs:///weather/weather.csv
```

```
copyFromLocal: `/weather/weather.csv': File exists
CPU times: user 36 ms, sys: 12 ms, total: 48 ms
Wall time: 2.11 s
```

```
In [1]: !hadoop fs -ls /weather
```

```
/bin/sh: hadoop: command not found
```

0.4.1 Read csv file into an RDD

```
In [ ]: %cd /mnt/workspace/edX-Micro-Master-in-Data-Science/big-data-analytics-using-spark/note
        !ls lib
```

```
In [15]: %pwd
         !ls -l lib/numpy_pack.py
```

```
-rw-rw-r-- 1 hadoop hadoop 1085 Mar  1 20:32 lib/numpy_pack.py
```

```
In [16]: %%time
         %pwd
         from pyspark import SparkContext
         sc = SparkContext(pyFiles=['/mnt/workspace/edX-Micro-Master-in-Data-Science/big-data-a
```

```
CPU times: user 172 ms, sys: 16 ms, total: 188 ms
Wall time: 16.6 s
```

```
In [17]: %%time
         RDD=sc.textFile('/weather/weather.csv')
```

```
CPU times: user 0 ns, sys: 4 ms, total: 4 ms
Wall time: 367 ms
```

```
In [18]: fs_file="/mnt/workspace/edX-Micro-Master-in-Data-Science/big-data-analytics-using-spa
         !ls -l $fs_file

-rw-rw-r-- 1 hadoop hadoop 7668890105 Feb  9  2016 /mnt/workspace/edX-Micro-Master-in-Data-Scie
```

```
In [19]: %%time
         with open(fs_file,'r') as f:
             text=f.readlines()
         print(len(text))

9358395
CPU times: user 2.21 s, sys: 5.12 s, total: 7.33 s
Wall time: 7.32 s
```

0.4.2 Code for packing and unpacking byte arrays

```
In [20]: import numpy as np
         """Code for packing and unpacking a numpy array into a byte array.
            the array is flattened if it is not 1D.
            This is intended to be used as the interface for storing

            This code is intended to be used to store numpy array as fields in a dataframe and
            dataframes in a parquet file.
         """

Out[20]: 'Code for packing and unpacking a numpy array into a byte array.\n   the array is fla

In [21]: def packArray(a):
             """
             pack a numpy array into a bytearray that can be stored as a single
             field in a spark DataFrame

             :param a: a numpy ndarray
             :returns: a bytearray
             :rtype:

             """
             if type(a)!=np.ndarray:
                 raise Exception("input to packArray should be numpy.ndarray. It is instead "+
             return bytearray(a.tobytes())

In [22]: def unpackArray(x,data_type=np.float16):
             """
             unpack a bytearray into a numpy.ndarray

             :param x: a bytearray
             :param data_type: The dtype of the array. This is important because if determines
```

```
        :returns: a numpy array
        :rtype: a numpy ndarray of dtype data_type.

        """
        return np.frombuffer(x,dtype=data_type)
```

0.4.3 range values

Using code that was removed we find that the range of values is
`-1000.0, 97892.0`
which means that as ints we will need 32 but, but with float we can use just 16.

```
In [23]: #main parsing code

         import numpy as np
         def parse_weather(line):
             L=line.split(',')
             try:
                 assert len(L)==368
                 i=2
                 L[i]=int(L[i])
                 for i in range(3,368):
                     if L[i]!='':
                         L[i]=np.float16(L[i])
                     else:
                         L[i]=np.nan
             except:
                 #if error in parsing, return (1, input line)
                 return (1,line)
             Out=L[:3]
             Out.append(packArray(np.array(L[3:],dtype=np.float16)))
             # if parsing OK, return (0, parsed data)
             return (0,Out)
```

```
In [24]: #this cell demonstrates how to test the parse_weather function on an individual row.
         Debug=False
         if Debug:
             lines=RDD.take(10)
             GG=parse_weather(lines[-2])
             GG
```

```
In [25]: %%time
         Parsed=RDD.map(parse_weather).cache() # filter out bad rows which are mapped (1,line)
         DATA=Parsed.filter(lambda x:x[0]==0).map(lambda x:x[1])
         ERRORS=Parsed.filter(lambda x:x[0]==1).map(lambda x:x[1])
```

```
CPU times: user 8 ms, sys: 0 ns, total: 8 ms
Wall time: 26.3 ms
```

```
In [26]: print(DATA.toDebugString().decode())

(58) PythonRDD[3] at RDD at PythonRDD.scala:48 []
 |   PythonRDD[2] at RDD at PythonRDD.scala:48 []
 |   /weather/weather.csv MapPartitionsRDD[1] at textFile at NativeMethodAccessorImpl.java:0 [
 |   /weather/weather.csv HadoopRDD[0] at textFile at NativeMethodAccessorImpl.java:0 []
```

```
In [27]: %%time
         PRCP=DATA.filter(lambda row:row[1]=='PRCP')
         print('PRCP records:',PRCP.count())

('PRCP records:', 2521007)
CPU times: user 16 ms, sys: 12 ms, total: 28 ms
Wall time: 2min 2s
```

```
In [28]: %%time
         print('bad records:',ERRORS.count())
         #all lines: 9358395
         # only the first line (the header) is bad.
         # Good lines: 9358394

('bad records:', 1)
CPU times: user 4 ms, sys: 8 ms, total: 12 ms
Wall time: 2.3 s
```

```
In [29]: DATA.take(1)

Out[29]: [[u'ASN00054128',
           u'DAPR',
           1969,
           bytearray(b'\x00~\x00~\x00~\x00~\x00~\x00~\x00~\x00~\x00~\x00~\x00~\x00~\x00~\x00~\
```

0.5 Transform RDD into a Spark DataFrame

```
In [30]: import os
         import sys

         from pyspark import SparkContext
         from pyspark.sql import SQLContext
         from pyspark.sql.types import Row, StructField, StructType, StringType, IntegerType, 

         # Just like using Spark requires having a SparkContext, using SQL requires an SQLCont
         sqlContext = SQLContext(sc)
         sqlContext

Out[30]: <pyspark.sql.context.SQLContext at 0x7fd7e1266650>
```

```
In [31]: ### Defining the Schema explicitly
         # The advantage of creating a DataFrame using a pre-defined schema allows the content

         # In this case we create the dataframe from an RDD of tuples (rather than Rows) and p
         # Schema with two fields - person_name and person_age
         schema = StructType([StructField("Station",     StringType(), True),
                              StructField("Measurement", StringType(), True),
                              StructField("Year",        IntegerType(),True),
                              StructField("Values",      BinaryType(),True)
                             ])
         schema

Out[31]: StructType(List(StructField(Station,StringType,true),StructField(Measurement,StringTy

In [32]: %%time
         # Create a DataFrame by applying the schema to the RDD and print the schema
         ALL_DataFrame = sqlContext.createDataFrame(DATA, schema)
         ALL_DataFrame.printSchema()

root
 |-- Station: string (nullable = true)
 |-- Measurement: string (nullable = true)
 |-- Year: integer (nullable = true)
 |-- Values: binary (nullable = true)

CPU times: user 460 ms, sys: 0 ns, total: 460 ms
Wall time: 1.29 s
```

0.5.1 Write out data frame into Parquet directory

```
In [33]: %%time
         !hadoop fs -rm -r /weather/weather.parquet

18/03/01 21:55:39 INFO fs.TrashPolicyDefault: Namenode trash configuration: Deletion interval =
Deleted /weather/weather.parquet
CPU times: user 52 ms, sys: 76 ms, total: 128 ms
Wall time: 2.2 s


In [34]: %%time
         outfilename="hdfs:///weather/weather.parquet"
         ALL_DataFrame.write.save(outfilename)

CPU times: user 4 ms, sys: 4 ms, total: 8 ms
Wall time: 17.1 s


In [35]: !hadoop fs -du /weather/
```

```
7668890105  /weather/weather.csv
2302800781  /weather/weather.parquet
```

0.5.2 Copy parquet directory to head node and then to S3

```
In [36]: %cd /mnt/workspace/Data/
         !rm -rf weather.parquet/
         !ls -lrt
```

```
/mnt/workspace/Data
total 0
drwxrwxr-x 2 hadoop hadoop 199 Mar  1 20:07 stations.parquet
```

```
In [37]: %%time
         !hadoop fs -copyToLocal /weather/weather.parquet weather.parquet
```

```
CPU times: user 84 ms, sys: 140 ms, total: 224 ms
Wall time: 6.46 s
```

```
In [38]: !du .
```

```
1612        ./stations.parquet
2248960        ./weather.parquet
2250572        .
```

```
In [39]: %%time
         #rm parquet directory from s3
         !aws s3 rm --recursive --quiet s3://dse-weather/weather.parquet
```

```
CPU times: user 16 ms, sys: 80 ms, total: 96 ms
Wall time: 1.14 s
```

```
In [40]: %%time
         # Copy parquet directory from headnode to s3
         !aws s3 cp --recursive --quiet ./weather.parquet s3://dse-weather/weather.parquet
```

```
CPU times: user 344 ms, sys: 208 ms, total: 552 ms
Wall time: 18.1 s
```

0.6 Loading and using a parquet file

```
In [41]: !ls
```

```
stations.parquet  weather.parquet
```

```
In [42]: !rm -rf weather.parquet/

In [43]: %%time
         !aws s3 cp --recursive --quiet s3://dse-weather/weather.parquet ./weather.parquet

CPU times: user 328 ms, sys: 224 ms, total: 552 ms
Wall time: 17.1 s


In [44]: %%time
         !hadoop fs -copyFromLocal  weather.parquet /weather/weather.parquet

CPU times: user 176 ms, sys: 144 ms, total: 320 ms
Wall time: 9.53 s


In [45]: %%time
         parquet_name='/weather/weather'
         query="""SELECT station,measurement,year
         FROM parquet.`%s.parquet`
         WHERE measurement=\"PRCP\" """%parquet_name
         print(query)
         df2 = sqlContext.sql(query)
         print 'number of rows=',df2.count()
         df2.show(5)

SELECT station,measurement,year
FROM parquet.`/weather/weather.parquet`
WHERE measurement="PRCP"
number of rows= 2521007
+-----------+-----------+----+
|    station|measurement|year|
+-----------+-----------+----+
|IN011020501|       PRCP|1939|
|IN011020501|       PRCP|1940|
|IN011020501|       PRCP|1941|
|IN011020501|       PRCP|1942|
|IN011020501|       PRCP|1943|
+-----------+-----------+----+
only showing top 5 rows

CPU times: user 12 ms, sys: 0 ns, total: 12 ms
Wall time: 2.59 s
```