

Managing and Modeling Big Data [2021-2022]

Apache Spark

What is Apache Spark?

Apache Spark is an open-source, distributed processing system used for big data workloads. It utilizes in-memory caching and optimized query execution for fast queries against data of any size. Simply put, Spark is a **fast and general engine for large-scale data processing**.


The **fast** part means that it's faster than previous approaches to work with Big Data like classical MapReduce. The secret for being faster is that Spark runs on memory (RAM), and that makes the processing much faster than on disk drives.

The **general** part means that it can be used for multiple things like running distributed SQL, creating data pipelines, ingesting data into a database, running Machine Learning algorithms, working with graphs or data streams, and much more.

Apache Spark RDD

RDD (Resilient Distributed Dataset) is the fundamental data structure of Apache Spark which is an immutable collection of objects which computes on the different node of the cluster. Each and every dataset in **Spark RDD** is logically partitioned across many servers so that they can be computed on different nodes of the cluster. Apache Spark RDD supports two types of Operations Transformations and Actions

RDD Transformation

- **map(func)**
The map function iterates over every line in RDD and split into new RDD. Using map() transformation we take in any function, and that function is applied to every element of RDD. For example, in RDD {1, 2, 3, 4, 5} if we apply "rdd.map(x=>x+2)" we will get the result as (3, 4, 5, 6, 7).
- **flatMap()** 
With the help of flatMap() function, to each input element, we have many elements in an output RDD. The most simple use of flatMap() is to split each input string into words.
- **filter(func)**
Spark RDD filter() function returns a new RDD, containing only the elements that meet a predicate. It is a narrow operation because it does not shuffle data from one partition to many partitions.

- **reduceByKey(func, [numTasks])**
When we use reduceByKey on a dataset (K, V), the pairs on the same machine with the same key are combined, before the data is shuffled.
- **sortByKey()**
We apply sortByKey() function on a dataset of (K, V) pairs, the data is sorted according to the key K in another RDD.

RDD Actions

- **count()**
Action count() returns the number of elements in RDD.
- **take(n)**
The action take(n) returns n number of elements from RDD. It tries to cut the number of partition it accesses, so it represents a biased collection. We cannot presume the order of the elements.
Returns an array with the first n elements of the dataset.
- **countByValue()**
The countByValue() returns, many times each element occur in RDD.
For example, RDD has values {1, 2, 2, 3, 4, 5, 5, 6}
in this RDD “rdd.countByValue()”
will give the result {(1,1), (2,2), (3,1), (4,1), (5,2), (6,1)}
- **saveAsTextFile(path)**
Writes the elements of the dataset as a text file (or set of text files) in a given directory in the local filesystem, HDFS or any other Hadoop-supported file system. Spark calls toString on each element to convert it to a line of text

Running Big Data Spark on Cloudera VM Platform

Example: WordCount (Using pyspark API)

- 1) Start Spark Service

Open FireFox → Cloudera Manager → Find the Spark service → right click and start

- 2) Use 'big.txt' file as input file

--copy the file from local file system to hdfs (Note: file in Cloudera folder)

```
hadoop fs -copyFromLocal big.txt
```

- 3) Create the following a python file in the Cloudera folder (*WordCount.py*)

```
import sys

from pyspark import SparkContext, SparkConf

if __name__ == "__main__":

    # create Spark context with Spark configuration

    conf = SparkConf().setMaster("local").setAppName("Word Count")

    sc = SparkContext(conf=conf)

    # read in text file and split each document into words

    words = sc.textFile("big.txt").flatMap(lambda line: line.split(" "))

    # count the occurrence of each word

    wordCounts = words.map(lambda word: (word, 1)).reduceByKey(lambda a,b:a +b)

    wordCounts.saveAsTextFile("wordcount")
```

4) Executing the Code in terminal

- Open the terminal at the same directory of the code file and write the following command

```
spark-submit WordCount.py
```

-- After executing the code, Copy the output directory from hdfs to the local file System:

```
hadoop fs -copyToLocal wordcount
```

Example 2: Minimum Temperature

Use the temperature data set to calculate the temperature in Fahrenheit and then determine the minimum temperature for each station. This dataset contains the weather information of a city reported in 1800 like StationID, TMax, TMin, Precipitation with the temperature.

Example to convert from Celsius to Fahrenheit: $(30^{\circ}\text{C} \times 9/5) + 32 = 86^{\circ}\text{F}$

- 1) Copy the dataset csv file to hdfs

```
hadoop fs -copyFromLocal 1800.csv
```

- 2) Create **minTemperature.py** file

```
from pyspark import SparkConf, SparkContext

conf = SparkConf().setMaster("local").setAppName("MinTemperatures")
sc = SparkContext(conf = conf)

def parseLine(line):
    fields = line.split(',')
    stationID = fields[0]
    entryType = fields[2]
    temperature = float(fields[3]) * 0.1 * (9.0 / 5.0) + 32.0
    return (stationID, entryType, temperature)

lines = sc.textFile("1800.csv")
parsedLines = lines.map(parseLine)

minTemps = parsedLines.filter(lambda x: "TMIN" in x[1])
stationTemps = minTemps.map(lambda x: (x[0], x[2]))
minTemps = stationTemps.reduceByKey(lambda x, y: min(x,y))
minTemps.saveAsTextFile('minTemperatures')
```

5) Executing the Code in terminal

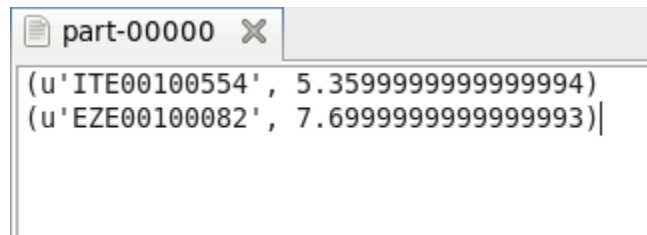
- Open the terminal at the same directory of the code file and write the following command

```
spark-submit minTemperature.py
```

-- After executing the code, Copy the output directory from hdfs to the local file System:

```
hadoop fs -copyToLocal minTemperetures
```

Output



```
(u'ITE00100554', 5.3599999999999994)
(u'EZE00100082', 7.6999999999999993)|
```

Notes:

- Start Hadoop Commands:
 - sudo service hadoop-hdfs-namenode start
 - sudo service hadoop-hdfs-datanode start
 - sudo service hadoop-hdfs-secondarynamenode start
 - sudo service hadoop-yarn-resourcemanager start
 - sudo service hadoop-yarn-nodemanager start
 - sudo service hadoop-mapreduce-historyserver start
- If you cannot create directory or add files in hdfs NameNode because it is in safe mode, run the below command:
 - sudo -u hdfs hdfs dfsadmin -safemode leave

Sources

- https://www.tutorialspoint.com/apache_spark/apache_spark_tutorial.pdf
- <https://data-flair.training/blogs/spark-rdd-operations-transformations-actions/>