PySpark

Thursday, September 21, 2023 9:06 AM

Spark:
- Large scale data processing
- Driver-executor
- Spark splits work and gives to individual executors
- Custom manager: takes input from driver and allocates tasks to individual workers and returns access back to driver node.
- Languages supported in spark: Python, Scala, R, SQL

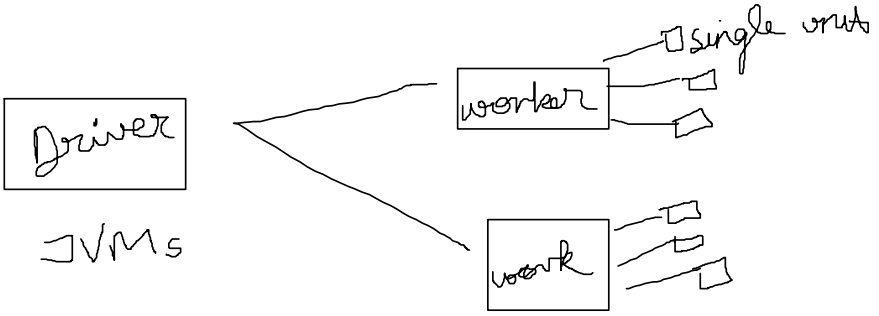
Lazy evaluation: until and unless you call for it, the transformation won't be executed
- Saves memory and time
- If particular record needed, transformation only applied on the record

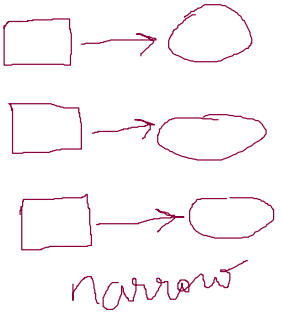
Job depends on number of actions called.
RDD Operations:
1. Transformation: data manipulation activities, narrow and wide transformation
2. Actions

Actions -> Jobs -> Stages -> Tasks

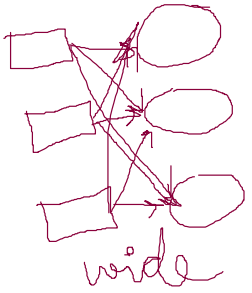
RDD: Resilient Distributed Dataset
Able to recover quickly
1. Rdd
2. Dataframe (we will use as it solves our needs as data engineers)
3. Dataset (combines rdd and df)

Caching: Cache memory
Catalyst and Tungsten: converts to RDD for machine to understand.





Select * from each
Where sal > 10000



Select max(salary), emp_loc
From emp_tbl
Group by emp_loc

```
In [1]: import findspark

In [2]: findspark.init()

In [3]: from pyspark.sql import SparkSession

In [4]: #initialise spark session

        spark = SparkSession.builder.appName("WordCount").getOrCreate()

        text_file = spark.sparkContext.textFile("/home/labuser/Desktop/testfile.txt")

        Setting default log level to "WARN".
        To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
        23/09/21 08:52:10 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-j
        ava classes where applicable

In [5]: #split lines into words and flatten them
        words = text_file.flatMap(lambda line: line.split(" "))

        #map each word to (word, 1) to prepare for counting

        word_counts = words.map(lambda word: (word, 1))

        #reduce by key to count the occurances of each word

        word_count = word_counts.reduceByKey(lambda a, b: a+b)

        #collect the results

        results = word_count.collect()
```

In [6]:
```
rdd = spark.sparkContext.parallelize([1,23, 4, 5])

rdd.collect()
```

Out[6]:
```
[1, 23, 4, 5]
```

In [7]:
```
wordlist=['this','is','a','sample','text','document','for','word','count','example','word','count']
rdd=spark.sparkContext.parallelize(wordlist)
wordcount=rdd.map(lambda word:(word,1)).groupByKey().mapValues(sum)



wordcount.collect()
```

Out[7]:
```
[('this', 1),
 ('sample', 1),
 ('text', 1),
 ('for', 1),
 ('word', 2),
 ('is', 1),
 ('a', 1),
 ('document', 1),
 ('count', 2),
 ('example', 1)]
```

In [10]:
```
purchaserdd = spark.sparkContext.textFile("/home/labuser/Downloads/Pandas_datasets/purchases.csv")
purchaserdd.collect()
```

Out[10]:
```
[',apples,oranges', 'June,3,0', 'Robert,2,3', 'Lily,0,7', 'David,1,2']
```

In [11]:
```
purchasedf = spark.read.csv("/home/labuser/Downloads/Pandas_datasets/purchases.csv")
purchasedf.show()
```

```
+------+------+-------+
|   _c0|   _c1|    _c2|
+------+------+-------+
|  null|apples|oranges|
|  June|     3|      0|
|Robert|     2|      3|
|  Lily|     0|      7|
| David|     1|      2|
+------+------+-------+
```

In [13]:
```
purchasedf_01 = spark.read.option("inferSchema", True).option("Header", True).csv("/home/labuser/Downloads/Pandas_d
purchasedf_01.show()
```

```
+------+------+-------+
|   _c0|apples|oranges|
+------+------+-------+
|  June|     3|      0|
|Robert|     2|      3|
|  Lily|     0|      7|
| David|     1|      2|
+------+------+-------+
```

```
23/09/21 09:03:42 WARN CSVHeaderChecker: CSV header does not conform to the schema.
 Header: , apples, oranges
 Schema: _c0, apples, oranges
Expected: _c0 but found:
CSV file: file:///home/labuser/Downloads/Pandas_datasets/purchases.csv
```