from pyspark.sql import SparkSession

import getpass

username = getpass.getuser()

spark = SparkSession. \

builder. \

config('spark.ui.port', '0'). \

config("spark.sql.warehouse.dir", f"/user/{username}/warehouse"). \

enableHiveSupport(). \

master('yarn'). \

getOrCreate()

orders\_rdd=spark.sparkContext.textFile(“/public/trendytech/retail\_db/orders/\*”)

mapped\_rdd = orders\_rdd.map(lambda x : (x.split(",")[3],1))

reduced\_rdd = mapped\_rdd.reduceByKey(lambda x,y : x+y)

reduced\_sorted = reduced\_rdd.sortBy(lambda x : x[1], False)

distinct\_customers = orders\_rdd.map(lambda x : (x.split(“,”)[2]).distinct()

distinct\_customers.count()

words =

("big","Data","Is","SUPER","Interesting","BIG","data","IS","A","Trending","techn

ology")

words\_rdd = spark.sparkContext.parallelize(words)

words\_normalized = words\_rdd.map(lambda x:x.lower())

words\_normalized.collect()

mapped\_words = words\_normalized.map(lambda x:(x,1))

aggregated\_result = mapped\_words.reduceByKey(lambda x,y:x+y)

aggregated\_result.collect()

spark.sparkContext.parallelize(words).map(lambda x:x.lower()).map(lambda x:(x,1)).reduceByKey(lambda x,y:x+y).collect()

result = spark. \

sparkContext. \

parallelize(words). \

map(lambda x:x.lower()). \

map(lambda x:(x,1)). \

reduceByKey(lambda x,y:x+y)

result.collect()

getNumPartitions()

spark.SparkContext.defaultMinPartitions

spark.SparkContext.defaultParallelism

**Narrow Transformation -** map, filter, flatmap

**Wide Transformation** - reduceByKey, groupByKey

Number of tasks = Number of partitions

No. of jobs = No.of actions executed

No. of stages = No.of wide transformations + 1

File - Orders Dataset of 3.5GB (3500 MB)

No.of blocks = 3500 MB /128 MB(default block size) = 28 Block = 28 Partitions

**joined\_rdd = customers.join(orders)**

**spark.sparkContext.broadcast(customers.collect())**

**Repartition Vs Coalesce**

Suppose there is a need to increase or decrease the number of partitions,

then **repartition** can be used as it can do both, i.e., increase / decrease the

no.of partitions in a RDD.

**Scenario 1 : When to increase the number of partitions**

- Say you have a 5GB file => 40 blocks ( = 40 partitions) running on a

100 node cluster.

- Since there are only 40 partitions, they will run on 40 nodes and the

remaining 60 nodes are idle.

- This leads to underutilization of resources. In such cases, the partitions

can be increased to achieve more parallelism.

**repartitioned\_rdd = base\_rdd.repartition(<increased partitions number>)**

**Scenario 2 : When to decrease the number of partitions**

- Say you have 1 TB file => 8000 blocks ( = 8000 partitions) running on a

100 node cluster.

- Each node will handle around 80 partitions.

- After applying transformations like - filter, the data would reduce

significantly ( say from 128MB[default] to 1MB )

- This will lead to a lot of sparse data and also would be tedious to

maintain large no.of partitions.

- It would be more efficient to have few no.of completely filled partitions

than having large no.of sparsely filled partitions.

How to decrease the no.of partitions?

**repartitioned\_rdd = base\_rdd.repartition(<decreased partitions number>)**

**Coalesce**

Coalesce can only decrease the number of partitions and cannot increase the

no.of partitions.

**When repartition can both increase / decrease the partitions, then what**

**is the need for coalesce?**

**Repartition** does a complete reshuffle of data while changing the no.of

partitions with the intent to have equal sized partitions.

**Repartition works well while increasing the number of partitions** but is

not efficient when decreasing the partitions.

**Coalesce** on the other hand tries to merge the partitions on the same node to

form a new partition that could be of unequal sizes but the shuffling is avoided.

Therefore, coalesce is **preferred when decreasing the no.of partitions as it**

**avoids shuffling of data.**