coalesce( ) Transformation

In [1]:
```
from pyspark.sql import SparkSession
import pyspark

spark = SparkSession \
        .builder \
        .master("local[*]") \
        .appName("coalesce Transformation") \
        .enableHiveSupport() \
        .getOrCreate()
```

```
22/10/14 12:05:42 WARN Utils: Your hostname, Vaishalis-MacBook-Pro.local resolves to a loopback address: 127.0.
0.1; using 192.168.0.105 instead (on interface en0)
22/10/14 12:05:42 WARN Utils: Set SPARK_LOCAL_IP if you need to bind to another address
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
22/10/14 12:05:43 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builti
n-java classes where applicable
22/10/14 12:05:44 WARN Utils: Service 'SparkUI' could not bind on port 4040. Attempting port 4041.
22/10/14 12:05:44 WARN Utils: Service 'SparkUI' could not bind on port 4041. Attempting port 4042.
```

In [2]:
```
def debug_a_partition(iterator):
    print("==begin-partition=")
    for x in iterator:
        print(x)
    #end-for
    print("==end-partition=")


names_list =["ABC - 1", "DEF - 1","GHI-1", "ABC - 2", "DEF - 2","GHI -2", \
             "ABC - 3", "DEF - 3","GHI -3","ABC - 4", "DEF - 4","GHI -4"]

names_rdd = spark.sparkContext.parallelize(names_list,4)

print("From local[4] =",names_rdd.getNumPartitions())
print("Repartition elements : ", names_rdd.foreachPartition(debug_a_partition))
```

```
From local[4] = 4
[Stage 0:>                                                          (0 + 4) / 4]
Repartition elements :  None
==begin-partition===begin-partition=

==begin-partition=
==begin-partition=ABC - 2
DEF - 2ABC - 3
ABC - 4
DEF - 3

DEF - 4GHI -3

GHI -4GHI -2

==end-partition=

==end-partition===end-partition=

ABC - 1
DEF - 1
GHI-1
==end-partition=
```

In [3]:
```
#reparition shuffles the data completely
repartition_rdd = names_rdd.repartition(3)
print("coalesce elements : ", repartition_rdd.foreachPartition(debug_a_partition))
```

```
[Stage 1:>                                                          (0 + 4) / 4]
coalesce elements :  None
==begin-partition=
==end-partition=
==begin-partition=
ABC - 2
DEF - 2
GHI -2
ABC - 4
DEF - 4
GHI -4
==end-partition=
==begin-partition=
ABC - 1
DEF - 1
GHI-1
ABC - 3
DEF - 3
GHI -3
==end-partition=
```

From the output above, we can see the reshuffling of the entire data from 4 partitions in names_rdd to 3 partitions in repartition_rdd. also, partition 1 is empty.

In [4]:
```
#coalesce combines the partition close to it
coalesce_rdd = names_rdd.coalesce(3)
print("Repartition size : ", coalesce_rdd.foreachPartition(debug_a_partition))
```

```
==begin-partition=
==begin-partition=ABC - 1

DEF - 1
GHI-1
==begin-partition===end-partition=ABC - 2

DEF - 2
GHI -2

==end-partition=
ABC - 3
DEF - 3
GHI -3
ABC - 4
DEF - 4
GHI -4
==end-partition=
Repartition size :  None
```

From the output above, we can see partitions 3 and 4 of names_rdd combined to form partition 3 in coalesce_rdd rather than shuffle around the entire data.

In [5]:
```
#Reading data from a file on the local machine
data_file_path = "/Users/vaishaliyasala/Desktop/Github/Spark/Exercise_Dependencies/SalesJan2009.csv"

df = spark.read.csv(data_file_path, header = True )

df1 = df.select(df["Name"],df["Country"]).repartition(4)

print("Count in the original data=", df1.count())

#Filter the names only from country United States
filtered_rdd = df1.rdd.filter(lambda x: (x[1] == "United States"))


filtered_rdd.toDF(["Name", "Country"]).show(5)


print("Filtered data Count =", filtered_rdd.count())
print("Number of Partitions =", filtered_rdd.getNumPartitions())
```

```
Count in the original data= 998
+---------+-------------+
|     Name|      Country|
+---------+-------------+
|   Abikay|United States|
|Christian|United States|
|  Alicja |United States|
|  Debora |United States|
| Sandrine|United States|
+---------+-------------+
only showing top 5 rows

Filtered data Count = 463
Number of Partitions = 4
```

In [6]:
```
#As the filtered data count increased significantly, we can reduce the number of partitions from 4 to 2
#It doesn't change the resultant RDD as seen from the outputs of both before and after coalesce transformation
#Only data changes in each partition


names_coalesce_rdd = filtered_rdd.coalesce(2)

names_coalesce_rdd.toDF(["Name", "Country"]).show(5)

print("Number of Partitions =", names_coalesce_rdd.getNumPartitions())
```

```
+---------+-------------+
|     Name|      Country|
+---------+-------------+
|   Abikay|United States|
|Christian|United States|
|  Alicja |United States|
|  Debora |United States|
| Sandrine|United States|
+---------+-------------+
only showing top 5 rows

Number of Partitions = 2
```