① scala> sc.defaultParallelism → check parallelism level

res: Int = 8

↓

∴ we have 8 partitions

val originalLogsRdd = sc.parallelize(myList)

② To check the no. of partition we have -

rdd.getNumPartitions

↓ rdd name

scala> originalLogsRdd.getNumPartitions

res: Int = 8.

> val input = sc.textFile("/Users/smaheshwa129/Desktop/dataset/customer-orders.csv")

↳ KB - file size

↓↓

no. of partitions

(local → 32 MB blocks size

⇒ 1 partition)

> input.getNumPartitions

res: Int = 2

↓

why we got 2?

Property

> sc.defaultMinPartitions

res: Int = 2

↓

determines the minimum no. of partitions rdd has; when we load from file

NumPartitions ≠ 1

↓

can never be 1, always start with 2.

even if we load file from HDFS of size = 100 MB

→ no. of partitions = 2 ✓ (not 1 as per 128 MB)

Diff b/w repartition and coalesce -

Suppose we have 500MB file in HDFS, and you have spark cluster of 20 machines.

If file size = 500MB, we have 4 blocks in hdfs (based on default block size = 128 MB)

⇒ 4 blocks in hdfs = 4 partitions in your rdd.

↳ out of 20 only 4 m/c are used to store partition ⇒ so we can do repartition

① val input = sc.textFile("_____")

input.repartition(10)

② input.getNumPartitions
= 2

↓

capture in new rdd as its again a transformation

repartition → increase/dec. no. of partitions, inc. parallelism.

③ val newRdd = input.repartition(10)

check:

④ newRdd.getNumPartitions ✓

we can also decrease no. of partition

val rdd2 = newRdd.repartition(1)

rdd2.getNumPartitions

res: Int = 1

coalesce - approach: minimize shuffling; & perform/ combine partitions on each m/c so that no shuffling is involved.

repartition is a wide transformation because shuffling is involved.

coalesce - used for same purpose but it can only decrease no. of partitions. It cannot increase no. of partitions.

```
newRdd.getNumPartitions
res: Int = 10

val rdd3 = newRdd.coalesce(4)

rdd3.getNumPartitions
res: Int = 4 ✓

val rdd4 = rdd3.coalesce(6)
```

↳ It won't give error but also not change no. of partitions.

```
rdd4.getNumPartitions
res: Int = 4
```

Coalesce is also a transformation.

⇒ If we want to dec no. of partitions, use coalesce or repartition
↳ preferred*

⇒ If we want to inc. no. of partitions: use repartition

✓ To dec no. of partitions; coalesce is preferred as it will try to minimize the shuffling.

node: N1 - p1, p2, p3, p4

N2 - p5, p6, p7, p8

N3 - p9, p10, p11, p12

N4 - p13, p14, p15, p16

sdd1 has 16 partitions
sdd1.repartition(8)

Repartition has an intention to have final partitions of exactly equal size and for this it has to go through complete shuffling / full shuffling

✓ sdd1.coalesce(8) → more performant

N1 - p1, p2, p3, p4 → p1, p2
N2 - p5, p6, p7, p8 → p3, p4
N3 - p9, p10, p11, p12 → p5, p6
N4 - p13, p14, p15, p16 → p7, p8

Coalesce has the intention to minimize the shuffling & combines existing partition on each machine to avoid a full shuffle.
(aggregate partitions internally to give big partitions within same m/c ⇒ combine local partitions to big partn ⇒ This is not always the case but whenever possible it will try to do, no gaurantee is there, chances are there)