Repartition vs Coalesce

Karthik Kondpak
9989454737

Day 4 — Spark Optimization Topic

4. Repartition vs Coalesce

(One ofthe mostmisunderstood topicsinPySparkinterviews)

Most Spark performance issues come from incorrect partitioning.

At 20–40 LPA interviews, this is asked in **almost every round**.

1. Repartition

Creates **N new partitions** using a **full shuffle**.

Key Points

- Increases OR decreases partitions
- Uses **shuffle** → expensive
- Redistributes data evenly
- Randomizes data across executors

When to Use

✓Before **joins** (to balance skew)

✓ Before **large window functions**

✓ Before **writing large outputs** (e.g., S3/Delta)

✔When your DataFrame has **too few partitions**

Example

```
df2 = df.repartition(200)
```

This triggers a **full shuffle**.

2. Coalesce

Reduces partitions **without shuffle**.

Key Points

- Only decreases number of partitions
- No shuffle
- Very cheap
- But may create **unbalanced partitions**

When to Use

✔When writing **small output files**

✔ After heavy filtering (less data left)

✔ When reading Delta/Parquet leads to many small partitions

Example

```
df2 = df.coalesce(10)
```

No shuffle → faster.

🔥 Repartition vs Coalesce — Real Difference

Feature	repartition()	coalesce()
Shuffle	Yes (expensive)	No (cheap)
Can increase	Yes Yes Good	No
Can decrease	(even) Big ops,	Yes
Data balance	joins	Poor (uneven)
Use cases		Final writes

Scenario — Zomato Orders

Dataset: zomato_orders (1.2B records)

Partitions after load: **1500 partitions** (too many)

You want to write results to Delta in 100 partitions.

Wrong (huge shuffle):

```
df2 = df.repartition(100)
```

✔ Correct:

```
df2 = df.coalesce(100)
```

Reason:

Filtering + partition pruning already reduced dataset massively, so **shuffling again is wasteful**.

When repartition() is BETTER

Example:Joininglargetables

You join:

- `2B-row transactions`
- `150M-row customers`

If their partition count is different → **skew**.

Fix:

```
tx = transactions.repartition(400, "customer_id")
cust = customers.repartition(400, "customer_id")
```

✓Balanced shuffle

✓Faster join

✓No skew on customer_id

When coalesce() is BETTER

Case:

After filtering:

```
small_df = big_df.filter("state = 'Maharashtra'")
```

Now only 3% data is left.

Coalesce is best:

```
optimized = small_df.coalesce(20)
```

✓No shuffle

✓Faster

✓Optimal number of partitions for writing

Let's build your Data Engineering journey together!

Call us directly at: 9989454737

https://seekhobigdata.com/