Seekho Bigdata
Data is the New Oil

Data Skew

Karthik Kondpak
9989454737

Day 7 — Spark Optimization Topic

7. Data Skew — Why One Partition Slows Your Entire Job

Data**skew** is one ofthe MOSTcommonreal-worldSpark performance problems.

It happens when **some partitions contain much more data than others**, causing:

- Long-running straggler tasks
- Executors waiting idle
- Slow stages
- Job delays (sometimes hours)

In short:

> **1 heavy partition = entire job becomes slow**

What Is Data Skew?

Dataskew=WhenSparkdistributes data unevenly across partitions.

Example:

If you group by

`state`, but **60% of data belongs to "Maharashtra"**,

then the partition responsible for “Maharashtra” becomes *very large*.

Why Skew Happens?

Because Spark partitions data **based on key values**.

Skew occurs when:

✓One key appears much more frequently

✓ Joins group large chunks under one key

✓ Bad partitioning strategy

✓ Real-world distributions are uneven

Impact of Skew

Impact	Explanation
Straggler tasks	One partition takes too long
Executors idle	Most finish early, one keeps running
High shuffle time	Heavy partition causes disk spill
OutOfMemory errors	Skew partition may not fit in memory
Slow job completion	Job finishes only when last partition finishes

Real Example of Skew

Dataset `/delta/order s`

order_id	state	amount
1001	Maharashtra	500
1002	Maharashtra	300
1003	Maharashtra	800
...	... Mi zo ra m	...
9999		150

Problem:

Maharashtra = 70% of India's e-commerce orders

→ Leads to skew during:

```
df.groupBy("state").sum("amount")
```

Spark UI Indicators of Skew

Check in**SQL→StageDetails**:

You'll see:

- One task taking **much longer** than others
- Very large input size for one partition
- Spill to disk
- High shuffle read/write time

Example:

Task	Duration	Input Size
0	5 sec	50 MB
1	6 sec	52 MB
2	48 sec	970 MB

How to Fix Data Skew

Thesearethe**exactsolutionsusedby data engineers in production**.

Salting Technique (Most Common Fix)

Add a random number to skewed keys → distribute load.

Before:

```
df.groupBy("customer_id").count()
```

If `customer_id = 101` is repeated 5 million times → skew.

After — Salt the key:

```
from pyspark.sql.functions import col, concat_ws,
rand


df2 = df.withColumn("customer_id_salted",
                    concat_ws("_",
col("customer_id"), (rand()*10).cast("int")))

df2.groupBy("customer_id_salted").count()
```

Later, remove the salt and aggregate again if needed.

Use Broadcast Join (Avoid Shuffle on Skew Key)

Instead of joining a huge table → broadcast the small one.

```
from pyspark.sql.functions import broadcast

df_large.join(broadcast(df_small), "customer_id")
```

No shuffle → no skew.

Skew Join Optimization (Spark Built-in)

Enable this when joining skewed data:

```
spark.conf.set("spark.sql.adaptive.skewJoin.enabled",
true)
```

Spark will:

- Detect skewed partitions
- Split them into smaller chunks
- Rebalance workload

Use `repartitionByRange` for Natural Balanced Splits

```
df.repartitionByRange(10, "amount")
```

Better distribution based on numeric ranges.

Use Bucketing for Repeated Joins

If you join same tables daily based on skewed keys:

```
df.write.bucketBy(50,
"customer_id").sortBy("customer_id").saveAsTable("orde
rs_bucketed")
```

Buckets → less shuffle → less skew.

Filter Early

If Maharashtra is skewed, filter unnecessary rows early:

```
df = df.filter("amount > 0")
```

Let's build your Data Engineering journey together!

Call us directly at: 9989454737

https://seekhobigdata.com/