Seekho Bigdata
Data is the New Oil

File Size Optimization

Karthik Kondpak
9989454737

Day 15 — Spark Optimization Topic

🔥 File Size Optimization — How Small Files Kill Spark Performance

Small files are one of the **biggest silent killers** in Spark jobs.

They slow down queries, increase cost, and overload the cluster without you even noticing.
If your data lake has **thousands of small Parquet/Delta files**, Spark will struggle — even if the data size is small.

Why Small Files Hurt Performance

Small files create:

Too many tasks → Too much scheduling overhead

Spark creates **1 task per file** per partition.

Example:

10,000 Parquet files = 10,000 tasks.

Most tasks finish in milliseconds, but **scheduler overhead** becomes huge.

Large Shuffle and Skew

Small files → small partitions → imbalance in size.

Some partitions = 5 KB

Some partitions = 500 MB

This causes:

- Slow executors
- Long-running tasks
- Skew during shuffle

High Metadata Load on Hive Metastore / Glue Catalog

Each file entry = one metadata entry.

100K small files → Metadata queries become slow.

Slow reads & writes

Spark reads multiple tiny row groups → more I/O operations.

Delta Lake Transaction Log growth

Each write adds one entry.

Small files → Too many commits → Slow time travel & vacuum.

Ideal File Size for Spark

Format	Ideal File Size
Parquet	256 MB – 1 GB
ORC	256 MB – 512 MB
Delta	256 MB – 1 GB

Anything below **16 MB** is considered a "small file" problem.

Scenario :

You work at **Zomato India**.

A pipeline ingests **600 MB** of daily orders, but writes **3,000 Parquet files of 200 KB each**.

Later when analysts query this table:

```
select * from orders where city = 'Mumbai'
```

Spark must:

• Open 3,000 files

• Read metadata per file

• Create thousands of tasks

The query becomes 10× slower.

How Spark Creates Small Files

You get small files when:

- Writing without partitioning correctly
- Writing with too many partitions
- Upstream system produces micro-batches
- Using **foreachBatch** or **streaming jobs**

- Using **overwrite** in loops
- Using **repartition(N)** incorrectly

How to Fix Small Files

Repartition before writing

Best method.

```
df.repa rtitio n(50) .writ e.form at("del ta").s ave(" /path
)
```

Choose partition count using:
`total_data_size / 256MB`

Use `coalesce()` for reducing partitions

Faster than repartition (no shuffle).

```
df.coalesce(20).write.parquet("/path")
```

Use OPTIMIZE (Delta Lake Only)

```
OPTIMIZE orders
WHERE order_date >= '2024-11-01'
```

This compacts small files → Large healthy files.

Use Auto-optimize + Auto-compaction (Databricks)

```
spark.c onf.se t("sp ark.d atabri cks.del ta.aut oComp act.e
abled", "true")
spark.c onf.se t("sp ark.d atabri cks.del ta.opt imize Write
enabled", "true")
```

Use `maxRecordsPerFile` to avoid extremely large files

```
df.write.option("maxRecordsPerFile",
1000000).parquet("/path")
```

Compact files manually (Apache Spark)

```
(

    spark.read.format("parquet").load("/path")
         .repartition(20)
         .write.mode("overwrite")
         .parquet("/path_temp")
)
```

Then swap folders.

Before & After Example

Condition	Spark Tasks	Runtime
Before (3000 small files)	3000+ tasks	20 mins
After compaction (20 files)	20 tasks	2 mins

10× FASTER — same data, better file layout.

Seekho Bigdata

Data is the New Oil

Let's build your Data Engineering journey together!

Call us directly at: 9989454737

https://seekhobigdata.com/