PySpark Optimization Techniques

1. Catalyst Optimizer
2. Tungsten Engine
3. Partitioning & Data Layout
4. Join Optimization
5. Caching & Persistence
6. File Format & Storage Optimizations
7. Execution Engine Tuning
8. Code-Level Optimizations

1. Catalyst Optimizer

Technique: Catalyst Query Optimization

- **Why it matters:**Automatically rewrites logical queries into efficient physical plans using rules like predicate pushdown, constant folding, and projection pruning.
- **Use Case:**ETL pipeline filtering large user datasets by status and date range — Catalyst reorders and optimizes the filters.

- **Code Example:**

```
 df = spark.read.parquet("/data/users")  df = df.filter(col("status") ==
"active").select("user_id", "email")
```

- **Pro Tip:**Avoid using UDFs too early — they're black-boxes for Catalyst and block many optimizations.

2. Tungsten

Technique: Tungsten In-Memory Optimization

- **Why it matters:**Optimizes memory layout using off-heap storage and bytecode generation to reduce JVM overhead.
- **Use Case:**Running ML models or iterative processing on large datasets, where JVM GC and object creation cost is high.
- **Code Example:**Tungsten is internal; you benefit automatically with DataFrame APIs.
- **Pro Tip:**Stick to high-level APIs (`DataFrame`, `Dataset`). Avoid `RDD` unless necessary.

3. Partitioning & Data Layout

Technique: Partition Pruning

- **Why it matters:**Reduces data scanned by skipping partitions not needed for query.
- **Use Case:**Monthly-partitioned sales data — analyzing only `month=06`.
- **Code Example:**

```
df = spark.read.parquet("/data/sales")  df_filtered = df.filter("month = 6")
```

- **Pro Tip:**Make sure your filter column matches partition column name. Use `spark.sql.sources.partitionOverwriteMode=dynamic` when overwriting.

Technique: Column Pruning

- **Why it matters:**Only reads required columns, reducing I/O.
- **Use Case:**Processing audit logs with 100+ columns but reporting on 3.
- **Code Example:**

```
df.select("user_id", "event_time", "action").show()
```

- **Pro Tip:**Avoid using `select("*")` in production code.

Technique: Repartitioning & Coalescing

- **Why it matters:**Controls the number of partitions — avoids shuffles or small file problems.
- **Use Case:**Before writing large files to HDFS or S3.
- **Code Example:**

```
df = df.repartition(10)  # increase and shuffle  df = df.coalesce(1)      # reduce
partitions (no shuffle)
```

- **Pro Tip:**Use `coalesce()` for write performance; use `repartition()` before joins to optimize shuffle.

4. Join Optimization

Technique: Broadcast Join

- **Why it matters:**Avoids shuffle when one table is small (fits in memory) — replicates it to all executors.
- **Use Case:**Joining a 1 crore transaction table with a small reference table (states list).
- **Code Example:**

```
from pyspark.sql.functions import broadcast  df = large_df.join(broadcast(small_df),
"state_id")
```

- **Pro Tip:**Works best for tables < 10MB. Set threshold via: `spark.sql.autoBroadcastJoinThreshold`

Technique: Skew Join Handling

- **Why it matters:**Some keys (like `India`) may appear too frequently, causing skewed joins and slow tasks.
- **Use Case:**Customer table has 80% records from Maharashtra.
- **Code Example (AQE):**

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

- **Pro Tip:**If AQE doesn't help, manually add "salting":

```python
from pyspark.sql.functions import concat, col, rand  skewed_df =
df.withColumn("salted_key", concat(col("key"), (rand() * 10).cast("int")))
```

5. Caching & Persistence

Technique: Cache / Persist

- **Why it matters:**Prevents recomputation of expensive intermediate results used multiple times.
- **Use Case:**ML pipelines where features are reused across models.
- **Code Example:**

```python
df.cache()  # or df.persist(StorageLevel.MEMORY_AND_DISK)
```

- **Pro Tip:**Always `unpersist()` after use to release memory.

6. File Format & Storage Optimizations

Technique: Columnar Formats (Parquet / ORC)

- **Why it matters:**Supports predicate pushdown, compression, and column pruning.
- **Use Case:**Storing customer transactions for analytics.
- **Code Example:**

```python
df.write.mode("overwrite").parquet("/output/path")
```

- **Pro Tip:**Avoid CSV for large-scale data — it's row-based and not optimized for Spark.

Technique: Optimal File Sizing

- **Why it matters:**Too many small files slow down job planning and reading.
- **Use Case:**Streaming data saved with 1000 tiny files per batch.
- **Code Example:**

```python
df.coalesce(10).write.mode("overwrite").parquet("/data/cleaned")
```

- **Pro Tip:**Aim for file sizes between 128MB and 1GB per file.

Technique: Bucketing

- **Why it matters:**Helps Spark do hash joins more efficiently.
- **Use Case:**Joining historical tables on `customer_id`.
- **Code Example:**

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

- **Pro Tip:**Works best when both tables are bucketed on the same column and number of buckets.

7. Execution Engine Tuning

Technique: Adaptive Query Execution (AQE)

- **Why it matters:**Spark adjusts physical plan at runtime (e.g., for skewed joins).
- **Use Case:**Queries with unknown cardinality and potential skew.
- **Code Example:**

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

- **Pro Tip:**Enabled by default in Spark 3.x+. Combine with skew join and dynamic partition coalescing.

Technique: Cost-Based Optimization (CBO)

- **Why it matters:**Helps Spark reorder joins more effectively using statistics.
- **Use Case:**Star schema with fact and many dimension tables.
- **Code Example:**

```
ANALYZE TABLE customers COMPUTE STATISTICS
```

- **Pro Tip:**Enable:

```
spark.conf.set("spark.sql.cbo.enabled", True)
```

8. Code-Level Optimizations

Technique: Avoid UDFs Where Possible

- **Why it matters:**UDFs block Catalyst and Tungsten optimizations.
- **Use Case:**Data cleaning or parsing logic.
- **Bad:**

```
from pyspark.sql.functions import udf  udf_upper = udf(lambda x: x.upper())
df.withColumn("name_upper", udf_upper("name"))
```

- **Good:**

```
df.withColumn("name_upper", upper(col("name")))
```

- **Pro Tip:**Use Spark SQL functions (`F.*`) — they're optimized internally.

Summary Cheat Sheet

Category	Optimization Technique	Tip
Catalyst	Logical/Physical Plan Optimization	Avoid early UDFs
Tungsten	In-memory layout, CPU tuning	Use DataFrame API
Partitioning	Partition Pruning, Column Pruning	Filter on partition column
Joins	Broadcast Join, Skew Handling	Enable AQE
Storage	Parquet, Bucketing, Coalesce	Avoid small files
Execution	AQE, CBO, Stats Collection	Combine for best results
Coding Practices	Avoid UDFs, Use Built-in Functions	More Catalyst wins

Seekho Bigdata

Data is the New Oil

Let's build your Data Engineering journey together!

Call us directly at: 9989454737

https://seekhobigdata.com/