Breaking Down Jobs, Tasks, and Stages in Spark Execution

Document by – Siddhartha Subudhi

Visit my LinkedIn profile

Scenario

Suppose you are working with a retail sales dataset stored in a data lake, and your daily task is to load this data, clean it, transform it, and then write the output to a data warehouse for reporting.

Dataset Example

You have a dataset with sales data of around **100 million rows** from multiple regions, with columns like sale_id, region, product_id, amount, and timestamp.

Step 1: Load Data from a Data Lake

```
# Importing necessary libraries
from pyspark.sql import SparkSession

# Creating Spark session
spark = SparkSession.builder \
    .appName("RetailSalesETL") \
    .getOrCreate()

# Reading data from the data lake (assume the format is parquet)
sales_data = spark.read.parquet("/data/sales_data/2023/")
```

Step 2: Data Cleaning and Filtering

- Remove any rows with null values in important columns like sale_id and amount.
- Filter out data for specific regions (e.g., removing region = 'Test').

```
# Data cleaning
cleaned_data = sales_data.filter((sales_data.sale_id.isNotNull()) &
(sales_data.amount.isNotNull()))

# Filter by region
filtered_data = cleaned_data.filter(sales_data.region != 'Test')
```

Step 3: Data Transformation (Aggregation)

- Calculate total sales per region and product for reporting.

```
from pyspark.sql.functions import col, sum

# Aggregation: Total sales per region and product
aggregated_data = filtered_data.groupBy("region",
"product_id").agg(sum("amount").alias("total_sales"))
```

Step 4: Write the Transformed Data to the Data Warehouse

```
# Writing the output to a data warehouse
aggregated_data.write.mode("overwrite").parquet("/datawarehouse/aggregated_sales/")
```

Spark Concepts: Jobs, Stages, and Tasks

1. Job

A **Job** in Spark is triggered by an **action** like write, count, or collect. Every time you call an action, Spark breaks down the computation into one or more jobs depending on the complexity of the transformations. In our scenario, writing the output triggers a job.

- **Example from above**: The .write() call at the end will trigger a **job** that includes all transformations (cleaning, filtering, and aggregation).

2. Stage

A **Stage** in Spark is a set of parallel tasks, where each task works on a partition of the data. Spark divides a job into multiple stages based on transformations and data shuffles.

- **Stages are created when**: A shuffle operation (like groupBy) occurs, or a wide transformation like join, reduceByKey, or groupByKey happens.
- In our example, the groupBy("region", "product_id") will result in a shuffle, meaning Spark will need to redistribute data, triggering a new stage.

- **Example**: If there are two stages, one might be for reading and filtering the data, and another for the aggregation and writing process.

3. Task

A **Task** is the smallest unit of execution, and it operates on a single data partition. The number of tasks equals the number of data partitions Spark has assigned.

- **Example**: If your data is split into 500 partitions, Spark will create 500 tasks to process this data in parallel during each stage.

Thumb Rules for Calculating Jobs, Stages, and Tasks

1. Jobs

- One job is triggered for each action. Multiple actions like write, count, or collect will create multiple jobs.
- **Rule**: Count the number of actions in your code to estimate the number of jobs.

2. Stages

- Each shuffle operation (e.g., groupBy, join) will create a new stage.
- Wide transformations like groupByKey, join, reduceByKey involve data shuffling and thus new stages.
- **Rule**: Estimate the number of stages by counting the wide transformations or shuffle points.

3. Tasks

- The number of tasks depends on the number of partitions.
- **Rule**: Number of tasks = Number of partitions at the time of the stage execution. You can control the number of partitions with .repartition() or .coalesce().