Step 1: Import and Initialize Spark

python

CopyEdit

```
from pyspark.sql import SparkSession

# Initialize Spark session
spark = SparkSession.builder.appName("MatrixMultiplication").getOrCreate()
sc = spark.sparkContext
```

- **What it does:**
  - Imports Spark.
  - Starts a new Spark session with the app name "MatrixMultiplication".
  - sc is the SparkContext used to create RDDs (Resilient Distributed Datasets).

◆ **Step 2: Define Matrices**

python

CopyEdit

```
matrix_A = [
   (0, 0, 4), (0, 1, 6), (0, 2, 8),
   (1, 0, 5), (1, 1, 5), (1, 2, 4)
]

matrix_B = [
   (0, 0, 7), (0, 1, 8),
   (1, 0, 9), (1, 1, 10),
   (2, 0, 11), (2, 1, 12)
]
```

- Each tuple is of the form (row, column, value).
- **Matrix A** is 2×3 (2 rows, 3 columns)
- **Matrix B** is 3×2 (3 rows, 2 columns)

You are multiplying a 2×3 matrix with a 3×2 matrix, so the result will be **2×2**.

◆ Step 3: Convert to RDDs

python

CopyEdit

```
rdd_A = sc.parallelize(matrix_A)

rdd_B = sc.parallelize(matrix_B)
```

- Converts both matrices into RDDs so Spark can process them in parallel.

◆ Step 4: Map Phase (Key by Shared Dimension)

python

CopyEdit

```
mapped_A = rdd_A.map(lambda x: (x[1], (x[0], x[2])))  # A: key = column index

mapped_B = rdd_B.map(lambda x: (x[0], (x[1], x[2])))  # B: key = row index
```

- We **key A by its column** and **B by its row**, so that we can **join on common indices** for multiplication.

◆ Step 5: Join A and B on Shared Index

python

CopyEdit

```
joined = mapped_A.join(mapped_B)
```

- Joins elements from A and B that have the same column index of A and row index of B.
- Resulting format:
  (shared_index, ((row_A, val_A), (col_B, val_B)))

◆ Step 6: Multiply Corresponding Values

python

CopyEdit

```
partial_products = joined.map(lambda x: ((x[1][0][0], x[1][1][0]), x[1][0][1] * x[1][1][1]))
```

- For each matching pair, multiply the values and emit a partial product for position (row_A, col_B).

◆ Step 7: Reduce (Sum Partial Products)

python

CopyEdit

```
result = partial_products.reduceByKey(lambda x, y: x + y)
```

- Sums all the partial products for the same output cell in the resulting matrix.

◆ Step 8: Collect and Display Output

python

CopyEdit

```
output = result.collect()
for ((row, col), value) in sorted(output):
    print(f"({row}, {col}) -> {value}")
```

- Collects the results from the distributed environment and prints them.
- Output format: (row_index, column_index) -> value

What is an RDD?

RDD stands for **Resilient Distributed Dataset**.

It's the **core data structure** of Apache Spark. Think of it like a **fault-tolerant, distributed list** that can be processed in parallel across a cluster of computers.

Here's what that means:

✅ 1. Resilient

- It can recover from failures automatically.
- Spark keeps track of how the data was derived (its **lineage**), so if a node fails, it can **recompute** the lost data.

✅ 2. Distributed

- Data is **split across multiple nodes** (computers) in a cluster.
- So operations (like map, reduce, filter, etc.) can be run in **parallel**, making it super fast for big data.

✅ 3. Dataset

- It's basically a **collection of records**, similar to a list or array in Python.
- Each record can be any type of object: number, string, tuple, etc.

◆ What is sc.parallelize() doing?

python

CopyEdit

```
rdd_A = sc.parallelize(matrix_A)
```

- sc is your **SparkContext**, and parallelize() is a method that:
  - Takes a regular Python collection (like a list)
  - Breaks it into **partitions** (chunks)
  - Distributes these chunks across multiple machines (or threads, if local)
  - Returns an **RDD object** that you can now process in parallel