| **Ex. No. 3** | **Matrix Multiplication Using RDDs in Spark** |
| --- | --- |
| **Date of Exercise** | **02/08/2025** |

**Aim**

To compute the product of two matrices A (m × n) and B (n × p) using Apache

Spark RDDs.

**Procedure**

**1. Data Preparation**

Define two small matrices A and B (e.g., A of shape 2×3, B of shape 3×2) using

MatrixEntry.

**2. Spark Logic**

Create two RDDs: entriesA and entriesB, each as RDD[MatrixEntry].

**3. Map:**

o aKeyed = entriesA.map(e =&gt; (e.j, (e.i, e.value)))

o bKeyed = entriesB.map(e =&gt; (e.i, (e.j, e.value)))

**4 Join** on key j (common index).

**5. Map t**o ((i, k), product) and **use reduceByKey** to sum contributions.

**6.** Construct **CoordinateMatrix** from summed entries and collect the results.

**Program**

# Complete Matrix Multiplication using PySpark and NumPy

# Imports and Spark Configuration

from pyspark import SparkContext, SparkConf

from pyspark.sql import SparkSession

from pyspark.mllib.linalg.distributed import MatrixEntry, CoordinateMatrix

# Initialize Spark Session

spark = SparkSession.builder \

.appName("RDD MatrixMultiply") \

.master("local") \

.getOrCreate()

sc = spark.sparkContext

print("Spark initialized successfully!")

# Define Matrix A as RDD

# Matrix A (2x3): [1, 0, 2]

# [0, 3, -1]

entries\_a = sc.parallelize([

MatrixEntry(0, 0, 1.0), MatrixEntry(0, 2, 2.0),

MatrixEntry(1, 1, 3.0), MatrixEntry(1, 2, -1.0)

])

print("\nMatrix A entries:")

for entry in entries\_a.collect():

print(f"({entry.i},{entry.j}) = {entry.value}")

# Define Matrix B as RDD

# Matrix B (3x2): [1, 2]

# [3, 4]

# [5, 6]

entries\_b = sc.parallelize([

MatrixEntry(0, 0, 1.0), MatrixEntry(1, 0, 3.0), MatrixEntry(2, 0, 5.0),

MatrixEntry(0, 1, 2.0), MatrixEntry(1, 1, 4.0), MatrixEntry(2, 1, 6.0)

])

print("\nMatrix B entries:")

for entry in entries\_b.collect():

print(f"({entry.i},{entry.j}) = {entry.value}")

# Matrix Multiplication Logic

# Step 1: Key matrices for join operation

a\_keyed = entries\_a.map(lambda e: (e.j, (e.i, e.value)))

b\_keyed = entries\_b.map(lambda e: (e.i, (e.j, e.value)))

print("\nMatrix A keyed by column:")

for item in a\_keyed.collect():

print(item)

print("\nMatrix B keyed by row:")

for item in b\_keyed.collect():

print(item)

# Step 2: Perform join and multiplication

product = (a\_keyed.join(b\_keyed)

.map(lambda x: ((x[1][0][0], x[1][1][0]), x[1][0][1] \* x[1][1][1]))

.reduceByKey(lambda a, b: a + b)

.map(lambda x: MatrixEntry(x[0][0], x[0][1], x[1])))

print("\nIntermediate products:")

for item in a\_keyed.join(b\_keyed).collect():

print(item)

print("\nAfter multiplication and grouping:")

for entry in product.collect():

print(f"({entry.i},{entry.j}) = {entry.value}")

# Final Result

result = CoordinateMatrix(product)

print("\nFinal Matrix Multiplication Result (A × B):")

print("=" \* 50)

sorted\_entries = sorted(result.entries.collect(), key=lambda e: (e.i, e.j))

for entry in sorted\_entries:

print(f"({entry.i},{entry.j}) = {entry.value}")

# Stop Spark Session

spark.stop()

print("\nSpark session stopped successfully!")

import numpy as np

# Define matrices as NumPy arrays

matrix\_a = np.array([

[1.0, 0.0, 2.0],

[0.0, 3.0, -1.0]

])

matrix\_b = np.array([

[1.0, 2.0],

[3.0, 4.0],

[5.0, 6.0]

])

# Perform matrix multiplication

numpy\_result = np.dot(matrix\_a, matrix\_b)

print("\n" + "="\*50)

print("NumPy Matrix Multiplication Result (for verification):")

print("Matrix A (2x3):")

print(matrix\_a)

print("\nMatrix B (3x2):")

print(matrix\_b)

print("\nResult A × B (2x2):")

print(numpy\_result)

# Convert to same format as Spark result for comparison

print("\nNumPy result in coordinate format:")

for i in range(numpy\_result.shape[0]):

for j in range(numpy\_result.shape[1]):

if numpy\_result[i,j] != 0: # Only show non-zero entries

print(f"({i},{j}) = {numpy\_result[i,j]}")

print("\n" + "="\*50)

print("COMPARISON COMPLETE")

**Output**

(0,0) = 11.0

(0,1) = 14.0

(1,0) = 4.0

**(1,1) = 6.0**

**Result**

The resulting 2×2 product matrix is correctly computed using distributed operations.