**Name: Ajinkya Kumbhar**

**Roll no: 36**

**Class: BE(AI&DS)**

**Subject: Computer Laboratory IV**

**ASSIGNMENT No: 03**

**Title:**

Implement Matrix Multiplication using Map-Reduce

**Introduction:**

Matrix multiplication is a fundamental operation in linear algebra, widely used in various scientific and engineering applications. The process involves multiplying two matrices to produce a third matrix. However, when dealing with large matrices, traditional sequential algorithms might become inefficient. MapReduce offers a parallel processing paradigm well-suited for handling large-scale data processing tasks, including matrix multiplication. By leveraging MapReduce, we can distribute the workload across multiple nodes in a cluster, thereby speeding up the computation process.

**Objective:**

The objective of this task is to implement matrix multiplication using the MapReduce programming model. By breaking down the matrix multiplication operation into smaller tasks and distributing them across multiple processing nodes, we aim to achieve efficient parallel processing and expedite the computation of the resulting matrix

**Equipment/Requirements:**

Personal computer/laptop with internet connectivity

Access to CloudEra or Google Cloud Platform, Databricks, Snowflake, and Amazon Web Services (AWS) accounts

Basic knowledge of big data concepts

**Procedure:**

1. Input Data Distribution:

The input matrices are divided into smaller chunks and distributed across the nodes in the MapReduce cluster.

Map Phase:

Each mapper task receives a portion of the input matrices.

For each element in the input matrices, the mapper emits key-value pairs where the key represents the resulting matrix cell's coordinates (row, column), and the value contains the partial product of the corresponding elements from the input matrices.

The mapper tasks operate independently and in parallel.

Shuffle and Sort:

The framework shuffles and sorts the intermediate key-value pairs generated by the mapper tasks based on their keys.

Reduce Phase:

Each reducer task receives a group of intermediate key-value pairs sharing the same key (resulting matrix cell coordinates).

For each resulting matrix cell, the reducer computes the final value by summing up the partial products from all the mapper outputs associated with that cell.

The reducer tasks operate independently and in parallel.

Output Consolidation:

The reducer tasks produce the final output, which represents the resulting matrix.

Output Presentation:

The resulting matrix is presented as the output of the MapReduce job.

**Conclusion:**

In conclusion, employing the MapReduce paradigm for matrix multiplication offers scalability and efficiency benefits, enabling the processing of large matrices across distributed computing environments. By decomposing the computation into parallel tasks and leveraging distributed resources, MapReduce significantly reduces computation time. Despite its advantages, challenges such as communication overhead and load balancing must be addressed to optimize performance. Nonetheless, ongoing advancements in distributed computing frameworks continue to enhance MapReduce's capability for handling complex matrix computations effectively at scale.