Matrix Multiplication Using Hadoop MapReduce

Student Name: Yashwin Bangalore Subramani

Student ID: 1002248361

Instructor: Noor Zaman

Date of Submission: February 6th 2025

1. Datasets

Dataset Used:

- **Matrix A**: twotone.txt
- **Matrix B**: twotone-2.txt

These matrices were chosen from the **SuiteSparse Matrix Collection**, which includes large, sparse matrices that are commonly used for benchmarking scientific computing algorithms.

2. Changes Made to Improve Parallelism

The following optimizations were made to enhance parallelism and overall performance in the Hadoop MapReduce framework for matrix multiplication:

- **Incorporating HashMap for Local Aggregation in Mappers**
  - Implemented a HashMap in the mappers to locally aggregate intermediate key-value pairs.
  - Reduced the number of intermediate outputs emitted by the mappers, thereby decreasing the amount of data shuffled between the map and reduce phases. This optimization minimizes the overhead caused by network transfer and improves overall efficiency.
  - **Implementation Details**:
    - Local aggregation was performed within the map() method using a HashMap.
    - Emissions were deferred until the cleanup() method to output the aggregated results.
- **Increasing Input Splits for Parallel Processing**
  - **Parameter**: mapreduce.input.fileinputformat.split.maxsize
  - **Change Made**: Set to 64MB (16777216 bytes) to allow multiple mappers to process the input data concurrently.

    ```
    conf.set("mapreduce.input.fileinputformat.split.maxsize", "16777216");
    ```

- **Adjusting the Number of Reducers**
  - **For Job 2**: Set to 4 to ensure that the aggregation of results was distributed evenly across reducers.

    ```
    job2.setNumReduceTasks(4);
    ```

3. Performance Improvement

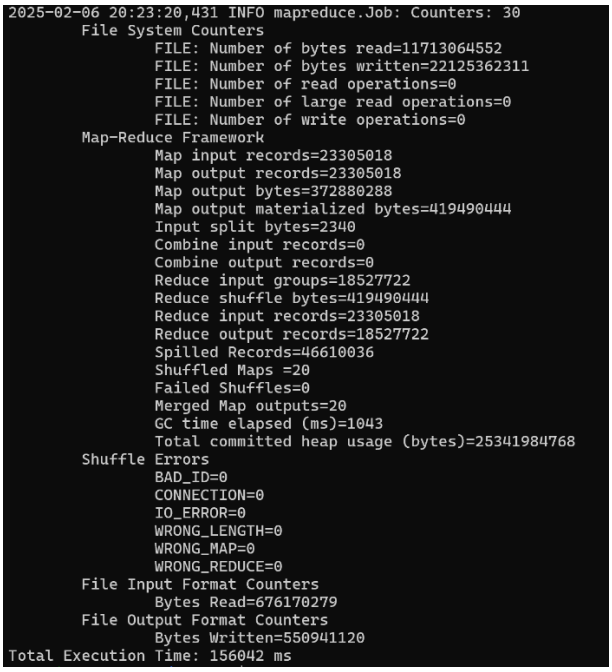
- **Old Execution Time**: 156,042 ms
- **New Execution Time**: 78,349 ms
- **Time Improvement**: The optimizations resulted in a significant reduction in execution time by approximately **77,693 ms (49.8%)**.

- **Contributors to Improvement**:
    - Local aggregation using HashMap reduced the volume of intermediate data shuffled.
    - Smaller input splits and an increased number of reducers improved parallelism and load balancing.

These changes helped significantly improve the run time by almost 78s for the large input dataset that was used.

Output screenshots:

Without optimization:



With optimization: