Matrix Multiplication Using Hadoop MapReduce

Introduction

This document outlines the implementation of a Hadoop MapReduce application for matrix multiplication. The program processes matrices stored in distributed files and computes their product using Hadoop's Mapper and Reducer functionalities.

Program Workflow

Input Data Format

Each matrix is stored in a text file, with entries formatted as follows:

> ***MatrixName, RowIndex, ColIndex, Value***

Example Input
Matrix A and B:

> ***A,0,0,25***
> ***A,0,1,9***
> ***B,0,0,44***
> ***B,1,0,13***

Output Data Format

The output matrix C = A × B is stored with the following format:

> ***RowIndex, ColIndex Value***

Example Output:

> ***0,0 841***
> ***1,0 1493***

MapReduce Process

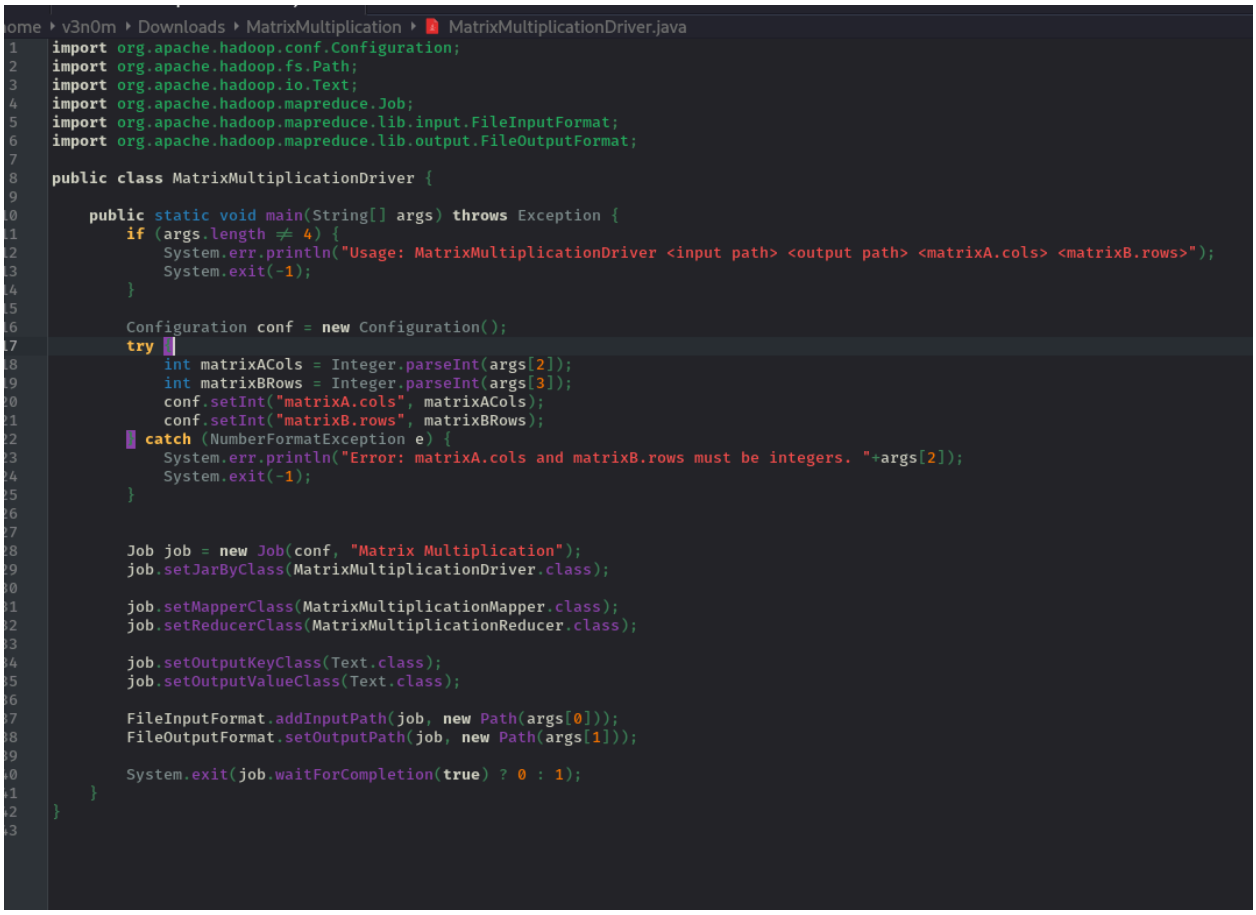
1. Mapper: Emits intermediate key-value pairs for each potential result cell in matrix C.

2. Reducer: Aggregates the contributions to each cell and computes the final value.

Code Description

Driver Code

The driver initializes the Hadoop job and configures the input/output paths and parameters like matrix dimensions.



The `MatrixMultiplicationDriver` class is the driver code for a Hadoop MapReduce application that performs matrix multiplication. It sets up and manages the MapReduce job's execution, including its configuration, input/output paths, and Mapper/Reducer assignments.

Key Points:

- **Input Arguments**:
  - `<input path>`: Path to the directory or file containing the input data.
  - `<output path>`: Directory where the output will be saved.
  - `<matrixA.cols>`: Number of columns in Matrix A.
  - `<matrixB.rows>`: Number of rows in Matrix B.
- **Error Handling**:

    - The program ensures `matrixA.cols` and `matrixB.rows` are integers and exits with an error message if they are not valid.
- **Role of Configuration**:
    - The `Configuration` object is used to pass custom parameters (`matrixA.cols` and `matrixB.rows`) to the Mapper and Reducer classes, enabling flexible handling of matrix dimensions.

This driver orchestrates the MapReduce job for matrix multiplication by ensuring all required configurations, paths, and operations are correctly set up.

Reducer Code

The reducer calculates the final value of each cell in the product matrix.

ome ▸ v3n0m ▸ Downloads ▸ MatrixMultiplication ▸ MatrixMultiplicationReducer.java

```java
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Reducer;

import java.io.IOException;
import java.util.HashMap;

public class MatrixMultiplicationReducer extends Reducer<Text, Text, Text, Text> {

    @Override
    protected void reduce(Text key, Iterable<Text> values, Context context) throws IOException, InterruptedException {
        HashMap<Integer, Double> matrixA = new HashMap<>();
        HashMap<Integer, Double> matrixB = new HashMap<>();

        for (Text value : values) {
            String[] tokens = value.toString().split(",");
            String matrixName = tokens[0];
            int index = Integer.parseInt(tokens[1]);
            double val = Double.parseDouble(tokens[2]);

            if (matrixName.equals("A")) {
                matrixA.put(index, val);
            } else if (matrixName.equals("B")) {
                matrixB.put(index, val);
            }
        }

        // Compute the dot product for C[i][k]
        double sum = 0.0;
        for (int j : matrixA.keySet()) {
            if (matrixB.containsKey(j)) {
                sum += matrixA.get(j) * matrixB.get(j);
            }
        }

        // Emit the result
        context.write(key, new Text(String.valueOf(sum)));
    }
}
```

Key Points:

1. **Intermediate Data Handling**:
    - The Reducer processes data grouped by the cell identifiers of Matrix C (e.g., `C[i][k]`).

- All relevant values from Matrix A and Matrix B for a specific `C[i][k]` are passed as input.

2. **Computational Logic**:
   - The dot product of a row from Matrix A and a column from Matrix B is computed.
   - Only matching indices (`j`) are used in the computation.
3. **Efficiency**:
   - The use of `HashMap` ensures quick lookups for matching indices, improving computational efficiency

Mapper Code

The mapper generates key-value pairs for intermediate computation.

ome ▸ v3n0m ▸ Downloads ▸ MatrixMultiplication ▸ MatrixMultiplicationMapper.java

```java
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;

import java.io.IOException;

public class MatrixMultiplicationMapper extends Mapper<Object, Text, Text, Text> {

    @Override
    protected void map(Object key, Text value, Context context) throws IOException, InterruptedException {
        String[] tokens = value.toString().split(",");
        String matrixName = tokens[0];
        int row = Integer.parseInt(tokens[1]);
        int col = Integer.parseInt(tokens[2]);
        double val = Double.parseDouble(tokens[3]);

        int matrixASize = context.getConfiguration().getInt("matrixA.cols", 0);
        int matrixBSize = context.getConfiguration().getInt("matrixB.rows", 0);

        if (matrixName.equals("A")) {
            // Emit (i, k) as key and "A,j,value" as value
            for (int k = 0; k < matrixASize; k++) {
                context.write(new Text(row + "," + k), new Text("A," + col + "," + val));
            }
        } else if (matrixName.equals("B")) {
            // Emit (i, k) as key and "B,j,value" as value
            for (int i = 0; i < matrixBSize; i++) {
                context.write(new Text(i + "," + col), new Text("B," + row + "," + val));
            }
        }
    }
}
```

Key Points:

1. **Key Design**:
   - The output key uniquely identifies a cell C[i][k] in the resulting matrix.
   - This ensures all relevant values for a specific C[i][k] are grouped together in the Reducer.
2. **Data Encoding**:
   - The output value encodes both the matrix name (`A` or `B`) and the relevant information (index and value).
   - This allows the Reducer to reconstruct the necessary values for the dot product computation.
3. **Efficiency**:

- The Mapper precomputes potential key-value pairs, distributing the computational workload across nodes.

Execution Steps

1. Compile the Program

Navigate to the source directory and compile the code:

javac -classpath $(hadoop classpath) -d classes MatrixMultiplication*.javaj

jar -cvf MatrixMultiplication.jar -C classes/ .

2. Prepare Input Files

Upload the matrix files to HDFS:

hadoop fs -mkdir /input

hadoop fs -put matrixA1.txt /input/

hadoop fs -put matrixA2.txt /input/

```
┌──(v3n0m㉿V3N0M)-[~/Downloads/MatrixMultiplication]
└─$ hadoop fs -mkdir /input
Picked up _JAVA_OPTIONS: -Dawt.useSystemAAFontSettings=on -Dswing.aatext=true
mkdir: `/input': Input/output error

┌──(v3n0m㉿V3N0M)-[~/Downloads/MatrixMultiplication]
└─$ hadoop fs -mkdir ./input
Picked up _JAVA_OPTIONS: -Dawt.useSystemAAFontSettings=on -Dswing.aatext=true
mkdir: `input': File exists

┌──(v3n0m㉿V3N0M)-[~/Downloads/MatrixMultiplication]
└─$ hadoop fs -put matrixA1.txt ./input/
Picked up _JAVA_OPTIONS: -Dawt.useSystemAAFontSettings=on -Dswing.aatext=true

┌──(v3n0m㉿V3N0M)-[~/Downloads/MatrixMultiplication]
└─$ hadoop fs -put matrixA2.txt ./input/
Picked up _JAVA_OPTIONS: -Dawt.useSystemAAFontSettings=on -Dswing.aatext=true
```

3. Run the Program

Execute the Hadoop job:

hadoop jar MatrixMultiplication.jar MatrixMultiplicationDriver /input /output matrixA.cols matrixB.rows

```
└─$ hadoop jar MatrixMultiplication.jar MatrixMultiplicationDriver ./input ./output 4 5
Picked up _JAVA_OPTIONS: -Dawt.useSystemAAFontSettings=on -Dswing.aatext=true
2024-11-29 21:46:09,607 INFO impl.MetricsConfig: Loaded properties from hadoop-metrics2.properties
2024-11-29 21:46:09,675 INFO impl.MetricsSystemImpl: Scheduled Metric snapshot period at 10 second(s).
2024-11-29 21:46:09,675 INFO impl.MetricsSystemImpl: JobTracker metrics system started
2024-11-29 21:46:09,749 WARN mapreduce.JobResourceUploader: Hadoop command-line option parsing not performed. Implement the Tool interface and execute your application with ToolRunne
2024-11-29 21:46:09,817 INFO input.FileInputFormat: Total input files to process : 3
2024-11-29 21:46:09,956 INFO mapreduce.JobSubmitter: number of splits:3
2024-11-29 21:46:10,262 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_local1614633083_0001
2024-11-29 21:46:10,262 INFO mapreduce.JobSubmitter: Executing with tokens: []
2024-11-29 21:46:10,480 INFO mapreduce.Job: The url to track the job: http://localhost:8080/
2024-11-29 21:46:10,487 INFO mapreduce.Job: Running job: job_local1614633083_0001
2024-11-29 21:46:10,490 INFO mapred.LocalJobRunner: OutputCommitter set in config null
2024-11-29 21:46:10,560 INFO output.PathOutputCommitterFactory: No output committer factory defined, defaulting to FileOutputCommitterFactory
2024-11-29 21:46:10,561 INFO output.FileOutputCommitter: File Output Committer Algorithm version is 2
2024-11-29 21:46:10,561 INFO output.FileOutputCommitter: FileOutputCommitter skip cleanup _temporary folders under output directory:false, ignore cleanup failures: false
2024-11-29 21:46:10,561 INFO mapred.LocalJobRunner: OutputCommitter is org.apache.hadoop.mapreduce.lib.output.FileOutputCommitter
2024-11-29 21:46:10,597 INFO mapred.LocalJobRunner: Waiting for map tasks
2024-11-29 21:46:10,597 INFO mapred.LocalJobRunner: Starting task: attempt_local1614633083_0001_m_000000_0
2024-11-29 21:46:10,664 INFO output.PathOutputCommitterFactory: No output committer factory defined, defaulting to FileOutputCommitterFactory
2024-11-29 21:46:10,664 INFO output.FileOutputCommitter: File Output Committer Algorithm version is 2
2024-11-29 21:46:10,664 INFO output.FileOutputCommitter: FileOutputCommitter skip cleanup _temporary folders under output directory:false, ignore cleanup failures: false
2024-11-29 21:46:10,713 INFO mapred.Task:  Using ResourceCalculatorProcessTree : [ ]
2024-11-29 21:46:10,718 INFO mapred.MapTask: Processing split: file:/home/v3n0m/Downloads/MatrixMultiplication/input/matrix.txt:0+59
2024-11-29 21:46:10,746 INFO mapred.MapTask: (EQUATOR) 0 kvi 26214396(104857584)
2024-11-29 21:46:10,746 INFO mapred.MapTask: mapreduce.task.io.sort.mb: 100
2024-11-29 21:46:10,746 INFO mapred.MapTask: soft limit at 83886080
2024-11-29 21:46:10,746 INFO mapred.MapTask: bufstart = 0; bufvoid = 104857600
2024-11-29 21:46:10,746 INFO mapred.MapTask: kvstart = 26214396; length = 6553600
2024-11-29 21:46:10,749 INFO mapred.MapTask: Map output collector class = org.apache.hadoop.mapred.MapTask$MapOutputBuffer
2024-11-29 21:46:10,798 INFO mapred.LocalJobRunner:
2024-11-29 21:46:10,799 INFO mapred.MapTask: Starting flush of map output
2024-11-29 21:46:10,799 INFO mapred.MapTask: Spilling map output
2024-11-29 21:46:10,799 INFO mapred.MapTask: bufstart = 0; bufend = 334; bufvoid = 104857600
2024-11-29 21:46:10,799 INFO mapred.MapTask: kvstart = 26214396(104857584); kvend = 26214296(104857184); length = 101/6553600
2024-11-29 21:46:10,816 INFO mapred.MapTask: Finished spill 0
2024-11-29 21:46:10,825 INFO mapred.Task: Task:attempt_local1614633083_0001_m_000000_0 is done. And is in the process of committing
2024-11-29 21:46:10,828 INFO mapred.LocalJobRunner: map
2024-11-29 21:46:10,828 INFO mapred.Task: Task 'attempt_local1614633083_0001_m_000000_0' done.
2024-11-29 21:46:10,847 INFO mapred.Task: Final Counters for attempt_local1614633083_0001_m_000000_0: Counters: 17
        File System Counters
                FILE: Number of bytes read=4815
                FILE: Number of bytes written=719127
                FILE: Number of read operations=0
                FILE: Number of large read operations=0
```

4. View Results

Check the output in HDFS:

hadoop fs -cat /output/part-r-00000