Movie Dataset Format

Assume the movie dataset is in the following format (CSV):

python-repl

Copy code

```
userId,movieId,rating,timestamp
1,31,2.5,1260759144
1,1029,3,1260759179
1,1061,3,1260759182
2,31,4,1260759205
2,1029,4.5,1260759230
...
```

In this dataset:

- userId: The ID of the user.
- movieId: The ID of the movie.
- rating: The rating given by the user.
- timestamp: The timestamp of the rating.

Steps to Build the Distributed Movie Recommendation System

Step 1: Prepare the Input Data

Create a sample dataset (movie_data.csv) and upload it to HDFS.

bash

Copy code

```
echo -e
"userId,movieId,rating,timestamp\n1,31,2.5,1260759144\n1,1029,3,1260759179\n1,1061,3,12
60759182\n2,31,4,1260759205\n2,1029,4.5,1260759230\n" > movie_data.csv

hadoop fs -mkdir -p /movie/input

hadoop fs -put movie_data.csv /movie/input/
```

Step 2: Create the Java Files for MapReduce

We will create a MapReduce program to process the movie dataset and recommend the highest-rated movies.

1. MovieMapper.java

The mapper will process each rating and emit the movieId along with its rating.

java

Copy code

```java
import java.io.IOException;

import org.apache.hadoop.io.FloatWritable;

import org.apache.hadoop.io.LongWritable;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapred.MapReduceBase;

import org.apache.hadoop.mapred.Mapper;

import org.apache.hadoop.mapred.OutputCollector;

import org.apache.hadoop.mapred.Reporter;


public class MovieMapper extends MapReduceBase implements Mapper<LongWritable, Text, Text, FloatWritable> {


    public void map(LongWritable key, Text value, OutputCollector<Text, FloatWritable> output, Reporter reporter) throws IOException {

        String line = value.toString();


        // Skip empty lines or the header row

        if (line.trim().isEmpty() || line.startsWith("userId")) {

            return;
        }

        String[] fields = line.split(",");
        if (fields.length < 3) {
            return; // Skip malformed lines
        }

        try {
            String movieId = fields[1];
            float rating = Float.parseFloat(fields[2]);
            output.collect(new Text(movieId), new FloatWritable(rating));
        } catch (NumberFormatException e) {
            // Handle lines where rating is not a valid float
            return;
        }
    }
}
```

2. MovieReducer.java

The reducer will aggregate the ratings for each movie and calculate the average rating.

java

Copy code

```
import java.io.IOException;

import java.util.Iterator;

import org.apache.hadoop.io.FloatWritable;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapred.MapReduceBase;
import org.apache.hadoop.mapred.OutputCollector;
import org.apache.hadoop.mapred.Reducer;
import org.apache.hadoop.mapred.Reporter;

public class MovieReducer extends MapReduceBase implements Reducer<Text, FloatWritable,
Text, FloatWritable> {

  public void reduce(Text key, Iterator<FloatWritable> values, OutputCollector<Text,
FloatWritable> output, Reporter reporter) throws IOException {
    float sum = 0;
    int count = 0;

    while (values.hasNext()) {
      sum += values.next().get();
      count++;
    }

    // Calculate the average rating for the movie
    float averageRating = sum / count;
    output.collect(key, new FloatWritable(averageRating));
  }
}
```

3. MovieDriver.java

The driver class sets up and runs the MapReduce job.

java

Copy code

```
import java.io.IOException;
import org.apache.hadoop.conf.Configured;

import org.apache.hadoop.fs.Path;

import org.apache.hadoop.io.FloatWritable;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapred.FileInputFormat;

import org.apache.hadoop.mapred.FileOutputFormat;

import org.apache.hadoop.mapred.JobClient;

import org.apache.hadoop.mapred.JobConf;

import org.apache.hadoop.util.Tool;

import org.apache.hadoop.util.ToolRunner;


public class MovieDriver extends Configured implements Tool {


  public int run(String[] args) throws IOException {

    if (args.length < 2) {

      System.out.println("Please provide input and output paths");

      return -1;

    }


    JobConf conf = new JobConf(MovieDriver.class);

    FileInputFormat.setInputPaths(conf, new Path(args[0]));

    FileOutputFormat.setOutputPath(conf, new Path(args[1]));

    conf.setMapperClass(MovieMapper.class);

    conf.setReducerClass(MovieReducer.class);

    conf.setMapOutputKeyClass(Text.class);

    conf.setMapOutputValueClass(FloatWritable.class);
```

```
        conf.setOutputKeyClass(Text.class);
        conf.setOutputValueClass(FloatWritable.class);


        JobClient.runJob(conf);
        return 0;
    }


    public static void main(String[] args) throws Exception {
        int exitCode = ToolRunner.run(new MovieDriver(), args);
        System.out.println(exitCode);
    }
}
```

Step 3: Compile the Java Files

Use Hadoop's classpath to compile the Java files:

bash

Copy code

```
javac -classpath `hadoop classpath` -d . MovieMapper.java MovieReducer.java MovieDriver.java
```

Step 4: Create a JAR File

After compilation, package the Java classes into a JAR file.

bash

Copy code

```
jar cf movie_recommendation.jar *.class
```

Step 5: Run the MapReduce Job

Run the MapReduce job with the following command:

bash

Copy code

```
hadoop jar movie_recommendation.jar MovieDriver /movie/input /movie/output
```

Step 6: View the Output

After the job finishes, view the output to see the average ratings for each movie:

bash

Copy code

```
hadoop fs -cat /movie/output/part-00000
```

Expected Output:

yaml

Copy code

```
31    3.25

1029  3.75

1061  3.00
```

This output shows the average ratings for each movie. The movie with the highest average rating can be recommended.

Remove previous output (if exists):

bash

Copy code

```
hadoop fs -rm -r /movie/output
```