Big Data Project - Milestone 2 Report

**4/2/2025**

**1. MapReduce Code**

**Code Explanation**

This MapReduce job is designed to count the number of ratings per movie from the MovieLens 1M dataset.  
- Mapper Class: Parses each line of the input file ratings.dat, extracts the movie ID (the second field), and outputs it as a key with the value 1.  
- Reducer Class: Aggregates the counts for each movie ID by summing up the values.  
- Driver Class: Configures the Hadoop job by specifying the mapper, reducer, input/output paths, and the key-value types.

**Full Java Code**

// Mapper Class  
public static class RatingMapper extends Mapper<LongWritable, Text, Text, IntWritable> {  
 public void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException {  
 String[] fields = value.toString().split("::");  
 if (fields.length > 1) {  
 context.write(new Text(fields[1]), new IntWritable(1));  
 }  
 }  
}  
  
// Reducer Class  
public static class RatingReducer extends Reducer<Text, IntWritable, Text, IntWritable> {  
 public void reduce(Text key, Iterable<IntWritable> values, Context context) throws IOException, InterruptedException {  
 int sum = 0;  
 for (IntWritable val : values) {  
 sum += val.get();  
 }  
 context.write(key, new IntWritable(sum));  
 }  
}  
  
// Driver Class  
public static void main(String[] args) throws Exception {  
 Configuration conf = new Configuration();  
 Job job = Job.getInstance(conf, "ratings count");  
 job.setJarByClass(RatingDriver.class);  
 job.setMapperClass(RatingMapper.class);  
 job.setReducerClass(RatingReducer.class);  
 job.setOutputKeyClass(Text.class);  
 job.setOutputValueClass(IntWritable.class);  
 FileInputFormat.addInputPath(job, new Path(args[0]));  
 FileOutputFormat.setOutputPath(job, new Path(args[1]));  
 System.exit(job.waitForCompletion(true) ? 0 : 1);  
}

**Compilation & Execution Commands**

# Compile  
javac -classpath `hadoop classpath` -d . RatingDriver.java  
jar -cvf ratings.jar \*.class  
  
# Run  
hadoop jar ratings.jar RatingDriver /input /output

**2. Code Execution & Output Interpretation**

Screenshot 1: Running Code and Result

Job was successfully submitted and tracked on YARN. The map and reduce phases completed successfully with no errors.

Screenshot 2: Process Completed Without Errors

Final counters were generated showing performance statistics, shuffle information, and memory usage confirming job completion.

Screenshot 3: Output Command

hdfs dfs -cat /output/part-r-00000  
1193 1  
661 1  
914 1

Output Interpretation

The job output displays the movie IDs and the number of ratings each received. For example, movie ID 1193 received 1 rating.

**3. Challenges & Troubleshooting**

Challenges Faced

- ratings.dat was uploaded as a folder, causing errors.  
- Hadoop job failed when /output directory already existed.  
- Connection refused issues when the cluster was inactive or master node down.

Troubleshooting Steps

- Used file ratings.dat to confirm it was a directory; fixed by uploading the actual file.  
- Cleared previous output directory using: hdfs dfs -rm -r /output  
- Restarted cluster and services when master node connection failed.

Performance Observations

- Execution time varied depending on data size and instance type.  
- Memory consumption and CPU time were moderate and appropriate for the dataset size.

**4. Summary & Key Learnings**

Project Reflection

- Gained hands-on experience running Hadoop jobs on AWS.  
- Learned how to write and debug MapReduce code using Java.  
- Understood how to interpret HDFS errors and job execution reports.

Real-World Application

This pipeline could be used by streaming services to count views/ratings, by retailers to count purchases per product, or any situation where item-frequency tracking is needed.

Future Improvements

- Add a combiner to reduce network load.  
- Extend logic to compute average ratings or filter top-N movies.  
- Optimize job by increasing parallelism.