Project Title: Music Genre Data Analysis

DS644006 Introduction to Big Data
Group Members:
Pavan Rahul Konathala(**UCID:**pk756 ,mail: pk756@njit.edu)
Ashot Kirakosyan(**UCID:** ak2995, mail: ak2995@njit.edu)

Dataset: Discogs Datasets(2025)

Dataset Description

This file contains information about master releases from the Discogs Data Dump as of January 1, 2025. Master releases represent a grouping of all versions of a specific release (e.g., different formats, editions, or reissues) under a single entry, providing a consolidated view of a musical work.

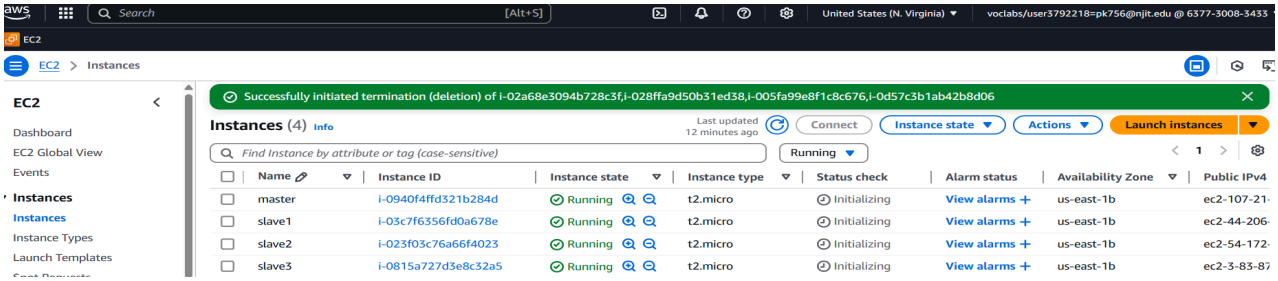
Columns:
• Master ID: Unique identifier for each master release.
• Main Release ID: ID of the primary release in the master entry.
• Title: The title of the master release.
• Artists: The primary artists associated with the release.
• Genres: The musical genres of the master release.
• Styles: Specific styles or sub-genres associated with the release.
• Year: The original release year of the master release.
• Tracklist: A list of tracks included in the master release.
• Data Quality: Indicates the quality or completeness of the data entry.

Size & Format: 608.46 MB & .csv

Reason for Selection: This dataset is very useful for conducting trend analysis across musical genres, styles, and release years . One of the main reasons why this dataset is useful for map-reduce is it is not real-time data and the dataset is very large 56330 non-null entries and we could do different kinds of analysis like occurrence of each genre , yearly music trends or top artists by release count.

Hadoop Cluster Setup Successful Screenshots

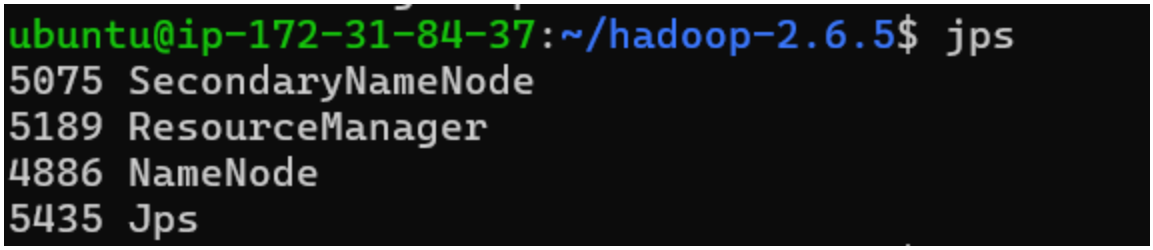
1)Instances



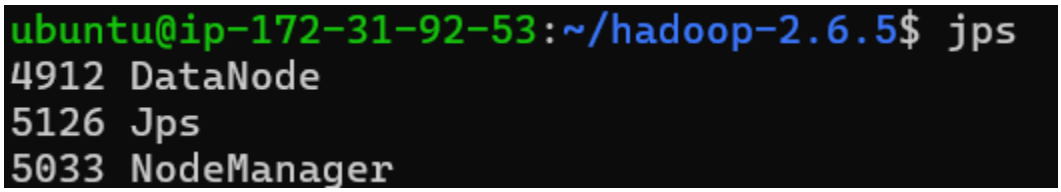
2) Accessing slave nodes from master node

```
ubuntu@ip-172-31-84-37:~$ ssh slave1
The authenticity of host 'slave1 (172.31.92.53)' can't be established.
ED25519 key fingerprint is SHA256:9qzM84g6AVInRb5kgo181FwrMvEimFWDAeP8MctMOf
E.
This key is not known by any other names
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added 'slave1' (ED25519) to the list of known hosts.
Welcome to Ubuntu 22.04.5 LTS (GNU/Linux 6.8.0-1024-aws x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/pro

 System information as of Fri Mar 21 21:39:25 UTC 2025

  System load:  0.0               Processes:             103
  Usage of /:   22.3% of 7.57GB   Users logged in:       1
  Memory usage: 20%               IPv4 address for eth0: 172.31.92.53
  Swap usage:   0%


Expanded Security Maintenance for Applications is not enabled.

0 updates can be applied immediately.

Enable ESM Apps to receive additional future security updates.
See https://ubuntu.com/esm or run: sudo pro status


The list of available updates is more than a week old.
To check for new updates run: sudo apt update
New release '24.04.2 LTS' available.
Run 'do-release-upgrade' to upgrade to it.


Last login: Fri Mar 21 21:30:29 2025 from 69.120.124.240
ubuntu@ip-172-31-92-53:~$
logout
Connection to slave1 closed.
```

3)Jps output
MasterNode:



Slave node:



---

4. MapReduce Code

Code Explanation:

Map class:
This class is responsible for reading input lines, processing them, and emitting key-value pairs. Where it checks if the genre belongs to the predefined list of valid genres in the dataset which are "Electronic", "Hip Hop", "Rock", "Pop", "Folk, World, & Country", "Funk / Soul", "Latin", "Reggae", "Jazz", "Stage & Screen","Non-Music", "Classical", "Blues", "Children's", "Brass & Military" this genres are extracted from python which was very helpful to implement the code in java.
If the genres are valid then it adds count for each genre.

Reduce class:
This class aggregates the key-value pairs emitted by the Mapper and It sums up all occurrences of each genre. So, the final output contains the genre name and its total count.

Driver Class:
It initiates the MapReduce job and its function is to assign the Mapper and reducer classes and specifies the output key-value data types.

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;

```
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

import java.io.IOException;
import java.util.Arrays;
import java.util.HashSet;
import java.util.Set;
import java.util.StringTokenizer;

public class GenreCount {

    /**
     * Mapper class that processes input lines, filters based on valid genres,
     * and emits key-value pairs (genre, 1) for counting.
     */
    public static class GenreMapper extends Mapper<Object, Text, Text, IntWritable> {
        private final static IntWritable one = new IntWritable(1);
        private Text genre = new Text();

        // Set of valid genres to filter the input
        private static final Set<String> VALID_GENRES = new HashSet<>(Arrays.asList(
                "Electronic", "Hip Hop", "Rock", "Pop", "Folk, World, & Country",
                "Funk / Soul", "Latin", "Reggae", "Jazz", "Stage & Screen",
                "Non-Music", "Classical", "Blues", "Children's", "Brass & Military"
        ));

        /**
         * The map function reads each line of input, filters it based on the valid genres,
         * and writes the genre with a count of 1 to the context.
         */
        public void map(Object key, Text value, Context context) throws IOException,
InterruptedException {
            StringTokenizer itr = new StringTokenizer(value.toString(), "\n");
            while (itr.hasMoreTokens()) {
                String genreText = itr.nextToken().trim();
                if (!genreText.isEmpty() && VALID_GENRES.contains(genreText)) {
                    genre.set(genreText);
                    context.write(genre, one);
                }
```

```
            }
        }
    }

    /**
     * Reducer class that sums up the counts for each genre.
     */
    public static class GenreReducer extends Reducer<Text, IntWritable, Text, IntWritable> {
        /**
         * The reduce function aggregates the counts for each genre and writes the final count.
         */
        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));
        }
    }

    /**
     * The main method sets up the Hadoop MapReduce job.
     */
    public static void main(String[] args) throws Exception {
        Configuration conf = new Configuration();
        Job job = Job.getInstance(conf, "genre count");
        job.setJarByClass(GenreCount.class);
        job.setMapperClass(GenreMapper.class);
        job.setCombinerClass(GenreReducer.class);
        job.setReducerClass(GenreReducer.class);
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(IntWritable.class);
        FileInputFormat.addInputPath(job, new Path(args[0])); // Input file path
        FileOutputFormat.setOutputPath(job, new Path(args[1])); // Output file path
        System.exit(job.waitForCompletion(true) ? 0 : 1);
    }
}
```

Execution Commands:

vi GenreCount.java (for creating java file and pasting the above code)

javac -classpath `hadoop classpath` -d . GenreCount.java (compiling the java code)

jar -cvf GenreCount.jar *.class (creating a jar file)

hadoop jar GenreCount.jar GenreCount /hadoop/input /hadoop/output (executing the mapreduce job)

hdfs dfs -cat /hadoop/output/part-r-00000 (to view the output of the result)

---

5. Code Execution & Output Interpretation

Map reduce successful execution:

```
        Map-Reduce Framework
                Map input records=9123219
                Map output records=6275
                Map output bytes=91003
                Map output materialized bytes=918
                Input split bytes=480
                Combine input records=6275
                Combine output records=58
                Reduce input groups=13
                Reduce shuffle bytes=918
                Reduce input records=58
                Reduce output records=13
                Spilled Records=116
                Shuffled Maps =5
                Failed Shuffles=0
                Merged Map outputs=5
                GC time elapsed (ms)=359
                CPU time spent (ms)=0
                Physical memory (bytes) snapshot=0
                Virtual memory (bytes) snapshot=0
                Total committed heap usage (bytes)=1002717184
        Shuffle Errors
                BAD_ID=0
                CONNECTION=0
                IO_ERROR=0
                WRONG_LENGTH=0
                WRONG_MAP=0
                WRONG_REDUCE=0
        File Input Format Counters
                Bytes Read=608480097
        File Output Format Counters
                Bytes Written=162
```

Output:

```
ubuntu@ip-172-31-84-37:~/hadoop-2.6.5$ hdfs dfs -cat /hadoop/output/part-r-0
0000
Blues   20
Classical       2
Electronic      5488
Folk, World, & Country  52
Funk / Soul     79
Hip Hop 39
Jazz    55
Latin   12
Non-Music       1
Pop     175
Reggae  31
Rock    315
Stage & Screen  6
```

---

6. Challenges & Troubleshooting

1)Cannot see the datanode running on slaves whenever an instance is launched again after disconnecting

```
ubuntu@ip-172-31-92-53:~$ jps
1184 Jps
1069 NodeManager
```

Troubleshooting commands for staring datanodes in slaves

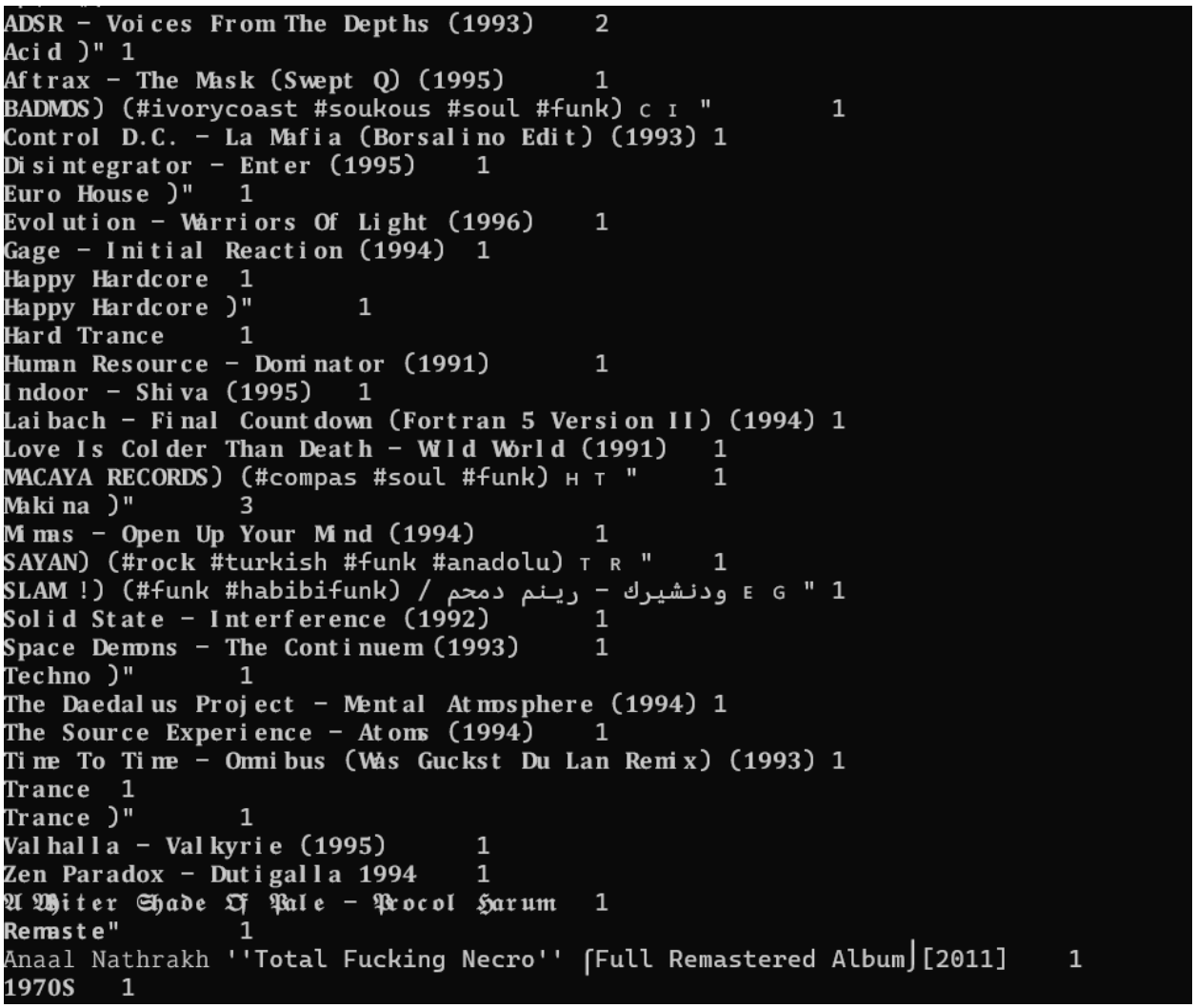
cd hadoop-2.6.5

rm -rf /home/ubuntu/hadoop-2.6.5/dfs/data/*

./sbin/hadoop-daemon.sh start datanode

```
ubuntu@ip-172-31-92-53:~/hadoop-2.6.5$ jps
1217 DataNode
1281 Jps
1069 NodeManager
```

2)We had trouble getting desired output due to several parsing errors and not cleaning the dataset like removing null values.



The above output was caused due to wrong parsing in Java so we had to clean the dataset in python and upload it in the master then run the program. But still we couldn't get the desired output as in python due to different symbols and other language words used for it like Arabic and Chinese characters.

So I have used python to find out all the genres in the dataset and so that it would be easy for my mapreduce code to parse through the dataset and find out the words.

Python code to find out genres:

```python
import pandas as pd

df = pd.read_csv('dataset.csv')

# Extract unique genres from the 'genres_genre' column
unique_genres = df['genres_genre'].unique()

# Print the unique genres
print("Unique Genres:")
for genre in unique_genres:
    print(genre)
```

```
Unique Genres:
Electronic
Hip Hop
Rock
Pop
Folk, World, & Country
Funk / Soul
Latin
Reggae
Jazz
Stage & Screen
Non-Music
Classical
Blues
Children's
Brass & Military
nan
```

Summary & Key Learnings

Project Reflection:

This project involved developing a Hadoop MapReduce program to process and count occurrences of specific music genres from a dataset. Hadoop MapReduce proved effective for large-scale text processing, with efficient genre counting achieved through structured data handling and performance optimization. Data accuracy was achieved by filtering genres and cleaning input with StringTokenizer. Performance was improved by using the Reducer as a Combiner.

Real-World Application:

This Mapreduce would help major streaming platforms like AppleMusic,YouTubeMusic,Spotify which deal with big data to analyze trends across different and recommend the music to the user based on his region,music taste, etc..

Future Improvements:
As we face a lot of problems processing the inputs, implementing more robust parsing techniques to handle various data formats may make the program more efficient.

We could not only use genre counts but we can try to find out which most released genre each year this would be a better approach to get deeper insights from the dataset.