**Big Data Analytics Mini Project**

**Friend recommendation based on movie data**

**Group Members :**

Janvi Jatakia (20)

Rohit Naik (33)

Kaushik Pai (37)

**Abstract :**

This mini project aims to collect, display and analyse Cinema data. It focuses on using the

IMDB (International Movie DataBase) website to gather information on past statistical data . This data is then analysed and ran using Hadoop Eco-System.

We will be applying Hadoop map-reduce to derive some statistics from IMDB movie data and also to make recommendations of users with similar liking for movies as friends.

**Objective:**

* To develop essential technical skills such as researching websites, recording data clearly, simple analysis and processing of data.
* To gain experience of handling large amounts of data and presenting it clearly and appropriately.
* To think about and be able to identify outliers in data, how to deal with outliers so they do not skew average results and to think of possible explanations for them.

**Introduction :**

Big data analytics is the process of examining large data sets containing a variety of data types -- i.e., big data -- to uncover hidden patterns, unknown correlations, market trends, customer preferences and other useful business information. The analytical findings can lead to more effective marketing, new revenue opportunities, better customer service, improved operational efficiency, competitive advantages over rival organizations and other business benefits.

In this project we are querying the database loaded in HDFS and retrieveing the records based on the given inputs. We have formed 3 basic questions using which we query the hadoop database. Once the result of the query is given, we get the answer for the required question. Also questions from users can be taken.

Also, we have created a movie recommendation system, wherein the user will enter the movies that he likes and based on the dataset, the MapReduce code will recommend users who likes similar movies as friends.

**Description :**

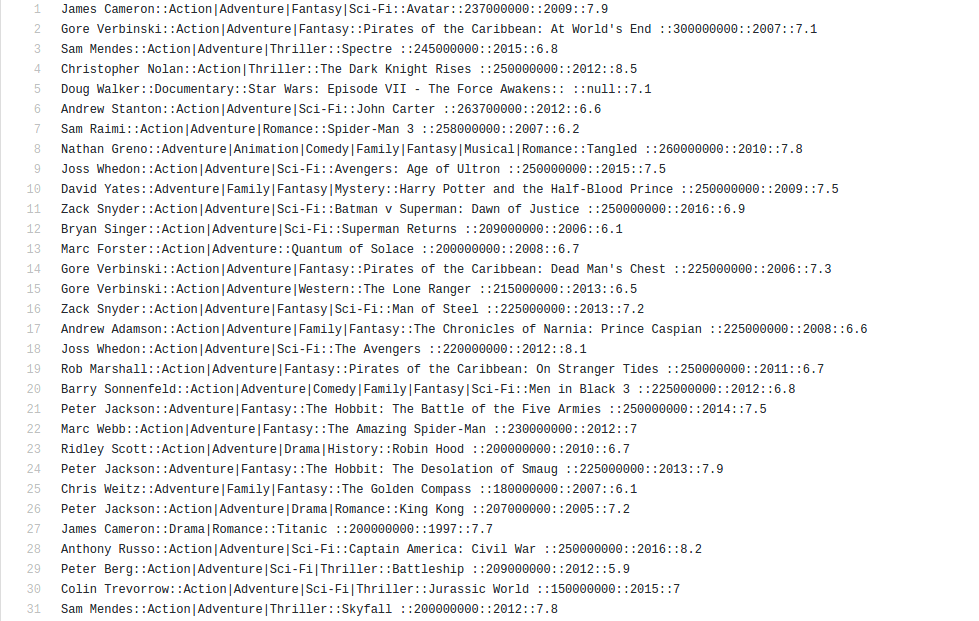
The 2 datasets used here are : impfields.txt, ip.txt.

**Impfields.txt Description-**

Movie information is in the file "impfields.txt" and is in the following

format:

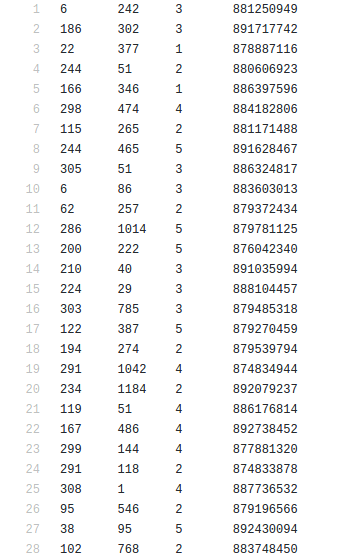
Director Name::Genres::Movie Name::Budget::Year::Rating



**ip.txt Description-**

User information is in the file "ip.txt" and is in the following format:

UserID::MovieID::User Rating::timestamp



**Questions**

1. Give the number of movies by a director. (Use impfields.txt file.)
2. Give the number of action movies in a year. (Use impfields.txt file.)
3. Give the average rating of movies by a director.

The input about the movies which the user likes should be taken fom command line. Use the ip.txt file to find similarity with this input.

**Implementation**

1. Apply HADOOP MapReduce to derive some statistics from IMDB Movie Data.
2. Copy Data into Hadoop cluster and use it as input data.
3. One can also use the 'put' or 'copyFromLocal' HDFS shell command to copy those files into HDFS Directory.
4. There are 2 data files : impfields.txt, ip.txt.

**Steps for Execution**

1. All JAR files have same name as class name.
2. Need to include extra jars :
   1. hadoop-common-2.6.0
   2. hadoop-mapreduce-client-core-2.6.0
3. Transfer all required files from local system to VM.
4. Run the following commands :
   1. For Question 1 :

hdfs dfs -mkdir input123

hdfs dfs -put impfields.txt input123

hadoop jar Question1.jar Question1 input123 outputq1

* 1. For Question 2 :

hadoop jar Question2.jar Question2 input123 outputq2

* 1. For Question 3 :

hadoop jar Question3.jar Question3 input123 outputq3

**To run the jobs use the following command :**

hadoop jar name\_of\_jar\_file Classname<input dir><output dir>[<extra input parameter>(may be optional due to question e.g. genre input)]

**Source Codes**

**Question1.java**

|  |
| --- |
| import java.io.IOException; |

|  |
| --- |
| import java.util.\*; |

|  |
| --- |
| import org.apache.hadoop.mapred.\*; |

|  |
| --- |
| import org.apache.hadoop.fs.Path; |

|  |
| --- |
| import org.apache.hadoop.io.IntWritable; |

|  |
| --- |
| import org.apache.hadoop.io.LongWritable; |

|  |
| --- |
| import org.apache.hadoop.io.Text; |

|  |
| --- |
| public class Question1 { |

|  |
| --- |
| public static class Map extends MapReduceBase implements Mapper<LongWritable, Text, Text, IntWritable> { |

|  |
| --- |
| private final static IntWritable wr = new IntWritable(1); |

|  |
| --- |
| private Text area = new Text(); |

|  |
| --- |
| private String val = null; |

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

|  |
| --- |
| String line = value.toString(); |

|  |
| --- |
| String[] myarray = line.split("::"); |

|  |
| --- |
| val = myarray[0]; |

|  |
| --- |
| area.set(val); |

|  |
| --- |
| output.collect(area, wr); |

|  |
| --- |
| } |

|  |
| --- |
| } |

|  |
| --- |
| public static class Reduce extends MapReduceBase implements Reducer<Text, IntWritable, Text, IntWritable> { |

|  |
| --- |
| public void reduce(Text key, Iterator<IntWritable> values, OutputCollector<Text, IntWritable> output, Reporter reporter) throws IOException { |

|  |
| --- |
| int sum = 0; |

|  |
| --- |
| while (values.hasNext()) { |

|  |
| --- |
| sum += values.next().get(); |

|  |
| --- |
| } |

|  |
| --- |
| output.collect(key, new IntWritable(sum)); |

|  |
| --- |
| } |

|  |
| --- |
| } |

|  |
| --- |
| public static void main(String[] args) throws Exception { |

|  |
| --- |
| JobConf conf = new JobConf(Question1.class); |

|  |
| --- |
| conf.setJobName("regionCount"); |

|  |
| --- |
| conf.setOutputKeyClass(Text.class); |

|  |
| --- |
| conf.setOutputValueClass(IntWritable.class); |

|  |
| --- |
| conf.setMapperClass(Map.class); |

|  |
| --- |
| conf.setReducerClass(Reduce.class); |

|  |
| --- |
| conf.setInputFormat(TextInputFormat.class); |

|  |
| --- |
| conf.setOutputFormat(TextOutputFormat.class); |

|  |
| --- |
| FileInputFormat.setInputPaths(conf, new Path("/home/rohit/workspace/Bda/input123/")); |

FileOutputFormat.setOutputPath(conf, new Path("/home/rohit/workspace/Bda/outputq1/"));

|  |
| --- |
| JobClient.runJob(conf); |

|  |
| --- |
| } |

}

**Question2.java**

|  |
| --- |
| import java.io.IOException; |

|  |
| --- |
| import java.util.\*; |

|  |
| --- |
| import org.apache.hadoop.mapred.\*; |

|  |
| --- |
| import org.apache.hadoop.fs.Path; |

|  |
| --- |
| import org.apache.hadoop.io.IntWritable; |

|  |
| --- |
| import org.apache.hadoop.io.LongWritable; |

|  |
| --- |
| import org.apache.hadoop.io.Text; |

|  |
| --- |
| public class Question2 { |

|  |
| --- |
| public static class Map extends MapReduceBase implements Mapper<LongWritable, Text, Text, IntWritable> { |

|  |
| --- |
| private final static IntWritable wr = new IntWritable(1); |

|  |
| --- |
| private Text area = new Text(); |

|  |
| --- |
| private String val = null; |

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

|  |
| --- |
| String line = value.toString(); |

|  |
| --- |
| String[] myarray = line.split("::"); |

|  |
| --- |
| if(myarray[1].contains("Action")) |

|  |
| --- |
| { |

|  |
| --- |
| val = myarray[4]; |

|  |
| --- |
| area.set(val); |

|  |
| --- |
| output.collect(area, wr); |

|  |
| --- |
| } |

|  |
| --- |
| } |

|  |
| --- |
| } |

|  |
| --- |
| public static class Reduce extends MapReduceBase implements Reducer<Text, IntWritable, Text, IntWritable> { |

|  |
| --- |
| public void reduce(Text key, Iterator<IntWritable> values, OutputCollector<Text, IntWritable> output, Reporter reporter) throws IOException { |

|  |
| --- |
| int sum = 0; |

|  |
| --- |
| while (values.hasNext()) { |

|  |
| --- |
| sum += values.next().get(); |

|  |
| --- |
| } |

|  |
| --- |
| output.collect(key, new IntWritable(sum)); |

|  |
| --- |
| } |

|  |
| --- |
| } |

|  |
| --- |
| public static void main(String[] args) throws Exception { |

|  |
| --- |
| JobConf conf = new JobConf(Question2.class); |

|  |
| --- |
| conf.setJobName("regionCount"); |

|  |
| --- |
| conf.setOutputKeyClass(Text.class); |

|  |
| --- |
| conf.setOutputValueClass(IntWritable.class); |

|  |
| --- |
| conf.setMapperClass(Map.class); |

|  |
| --- |
| conf.setReducerClass(Reduce.class); |

|  |
| --- |
| conf.setInputFormat(TextInputFormat.class); |

|  |
| --- |
| conf.setOutputFormat(TextOutputFormat.class); |

|  |
| --- |
| FileInputFormat.setInputPaths(conf, new Path("/home/rohit/workspace/Bda/input123/")); |

|  |
| --- |
| FileOutputFormat.setOutputPath(conf, new Path("/home/rohit/workspace/Bda/outputq2/")); |

JobClient.runJob(conf);

|  |
| --- |
| } |

}

**Question3.java**

|  |
| --- |
| import java.io.IOException; |

|  |
| --- |
| import java.util.\*; |

|  |
| --- |
| import org.apache.hadoop.mapred.\*; |

|  |
| --- |
| import org.apache.hadoop.fs.Path; |

|  |
| --- |
| import org.apache.hadoop.io.FloatWritable; |

|  |
| --- |
| import org.apache.hadoop.io.LongWritable; |

|  |
| --- |
| import org.apache.hadoop.io.Text; |

|  |
| --- |
| public class Question3 { |

|  |
| --- |
| public static class Map extends MapReduceBase implements Mapper<LongWritable, Text, Text, FloatWritable> { |

|  |
| --- |
|  |

|  |
| --- |
| private Text area = new Text(); |

|  |
| --- |
| private String val = null; |

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

|  |
| --- |
| String line = value.toString(); |

|  |
| --- |
| String[] myarray = line.split("::"); |

|  |
| --- |
| val = myarray[0]; |

|  |
| --- |
| area.set(val); |

|  |
| --- |
| FloatWritable wr = new FloatWritable(Float.parseFloat(myarray[5])); |

|  |
| --- |
| //System.out.println(myarray[4]); |

|  |
| --- |
| //System.out.println(myarray[5]); |

|  |
| --- |
| output.collect(area, wr); |

|  |
| --- |
| } |

|  |
| --- |
| } |

|  |
| --- |
| public static class Reduce 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++; |

|  |
| --- |
| } |

|  |
| --- |
| output.collect(key, new FloatWritable(sum/count)); |

|  |
| --- |
| } |

|  |
| --- |
| } |

|  |
| --- |
|  |

|  |
| --- |
| // public static RunningJob myCustomRunJob(JobConf job) throws Exception { |

|  |
| --- |
| // JobClient jc = new JobClient(job); |

|  |
| --- |
| // RunningJob rj = jc.submitJob(job); |

|  |
| --- |
| // if (!jc.monitorAndPrintJob(job, rj)) { |

|  |
| --- |
| // throw new IOException("Job failed with info: " + rj.getFailureInfo()); |

|  |
| --- |
| // } |

|  |
| --- |
| // return rj; |

|  |
| --- |
| // } |

|  |
| --- |
|  |

|  |
| --- |
| public static void main(String[] args) throws Exception { |

|  |
| --- |
| JobConf conf = new JobConf(Question3.class); |

|  |
| --- |
| conf.setJobName("regionCount"); |

|  |
| --- |
| conf.setOutputKeyClass(Text.class); |

|  |
| --- |
| conf.setOutputValueClass(FloatWritable.class); |

|  |
| --- |
| conf.setMapperClass(Map.class); |

|  |
| --- |
| conf.setReducerClass(Reduce.class); |

|  |
| --- |
| conf.setInputFormat(TextInputFormat.class); |

|  |
| --- |
| conf.setOutputFormat(TextOutputFormat.class); |

|  |
| --- |
| FileInputFormat.setInputPaths(conf, new Path("/home/rohit/workspace/Bda/input123/")); |

|  |
| --- |
| FileOutputFormat.setOutputPath(conf, new Path("/home/rohit/workspace/Bda/outputq3/")); |

|  |
| --- |
| JobClient.runJob(conf); |

//myCustomRunJob(conf);

|  |
| --- |
| } |

}

**Reducer2.java**

|  |
| --- |
| package job2; |

|  |
| --- |
|  |

|  |
| --- |
| import java.io.IOException; |

|  |
| --- |
| import java.util.ArrayList; |

|  |
| --- |
| import java.util.Iterator; |

|  |
| --- |
|  |

|  |
| --- |
| 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; |

|  |
| --- |
|  |

|  |
| --- |
| import driver.Driver; |

|  |
| --- |
|  |

|  |
| --- |
| import java.util.HashSet; |

|  |
| --- |
| import java.util.Set; |

|  |
| --- |
|  |

|  |
| --- |
| public class Reducer2<T> extends MapReduceBase implements Reducer<Text, Text, Text, |

|  |
| --- |
| Text>{ |

|  |
| --- |
| private final Set<Integer> intersect = new HashSet<>(); |

|  |
| --- |
| private final Set<Integer> union = new HashSet<>(); |

|  |
| --- |
|  |

|  |
| --- |
| Set<Integer> set1=new HashSet<Integer>(); |

|  |
| --- |
| String user=Driver.arguments; |

|  |
| --- |
| String[] user\_movies=user.split(","); |

|  |
| --- |
| int firstTime=1; |

|  |
| --- |
|  |

|  |
| --- |
| public double compute(Set<Integer> set1, Set<Integer> set2) |

|  |
| --- |
| { |

|  |
| --- |
| intersect.clear(); |

|  |
| --- |
| intersect.addAll(set1); |

|  |
| --- |
| intersect.retainAll(set2); |

|  |
| --- |
| union.clear(); |

|  |
| --- |
| union.addAll(set1); |

|  |
| --- |
| union.addAll(set2); |

|  |
| --- |
| System.out.println("Set1 and Set2 --> "+set1.toString()+set2.toString()); |

|  |
| --- |
| System.out.println("Union and intersection --> "+union.toString() + intersect.toString()); |

|  |
| --- |
| return (double)intersect.size()/(double)union.size(); |

|  |
| --- |
| } |

|  |
| --- |
|  |

|  |
| --- |
| public void reduce(Text key, Iterator<Text> values, |

|  |
| --- |
| OutputCollector<Text, Text> output, Reporter reporter) throws IOException { |

|  |
| --- |
|  |

|  |
| --- |
| Set<Integer> set2=new HashSet<Integer>(); |

|  |
| --- |
| if(firstTime==1) |

|  |
| --- |
| { |

|  |
| --- |
| for (int i=0;i<user\_movies.length;i++) |

|  |
| --- |
| { |

|  |
| --- |
| set1.add(Integer.parseInt(user\_movies[i])); |

|  |
| --- |
| } |

|  |
| --- |
| firstTime=0; |

|  |
| --- |
| } |

|  |
| --- |
| while(values.hasNext()) |

|  |
| --- |
| { |

|  |
| --- |
| String[] reducer=values.next().toString().split(","); |

|  |
| --- |
| for(int i=0;i<reducer.length;i++) |

|  |
| --- |
| System.out.println("REDUCER --> "+reducer[i]); |

|  |
| --- |
| for (int i=0;i<reducer.length;i++) |

|  |
| --- |
| { |

|  |
| --- |
| set2.add(Integer.parseInt(reducer[i])); |

|  |
| --- |
| } |

|  |
| --- |
| } |

|  |
| --- |
|  |

|  |
| --- |
| double similarity=compute(set1, set2); |

|  |
| --- |
|  |

|  |
| --- |
| output.collect(key, new Text(Double.toString(similarity))); |

|  |
| --- |
| // ArrayList<Double> itemI = new ArrayList<Double>(); |

|  |
| --- |
| // ArrayList<Double> itemJ = new ArrayList<Double>(); |

|  |
| --- |
| // |

|  |
| --- |
| // while (values.hasNext()){ |

|  |
| --- |
| // String[] iJ = values.next().toString().split(","); |

|  |
| --- |
| // itemI.add(Double.parseDouble(iJ[0])); |

|  |
| --- |
| // itemJ.add(Double.parseDouble(iJ[1])); |

|  |
| --- |
| // } |

|  |
| --- |
| // |

|  |
| --- |
| // double iSum = 0; |

|  |
| --- |
| // double jSum = 0; |

|  |
| --- |
| // |

|  |
| --- |
| // for(int i=0; i<itemI.size(); i++){ |

|  |
| --- |
| // iSum= iSum+itemI.get(i); |

|  |
| --- |
| // jSum = jSum+itemJ.get(i); |

|  |
| --- |
| // } |

|  |
| --- |
| // |

|  |
| --- |
| // /\* |

|  |
| --- |
| // \* Mean of all users rating for a movie pair (i, j) |

|  |
| --- |
| // \*/ |

|  |
| --- |
| // double iMean = iSum/itemI.size(); |

|  |
| --- |
| // double jMean = jSum/itemJ.size(); |

|  |
| --- |
| // //System.out.println("I mean:"+iMean+","+"J mean"+jMean); |

|  |
| --- |
| // double numerator = 0.0; |

|  |
| --- |
| // double denominator = 0.0; |

|  |
| --- |
| // double similarity = 0.0; |

|  |
| --- |
| // /\* |

|  |
| --- |
| // \* Calculating numerator of Cosine similarity formula |

|  |
| --- |
| // \*/ |

|  |
| --- |
| // for (int i=0; i<itemI.size(); i++){ |

|  |
| --- |
| // numerator= numerator+(itemI.get(i) \* itemJ.get(i)); |

|  |
| --- |
| // } |

|  |
| --- |
| // |

|  |
| --- |
| // /\* |

|  |
| --- |
| // \* Calculating denominator of Cosine similarity formula |

|  |
| --- |
| // \*/ |

|  |
| --- |
| // double denom1 = 0.0; |

|  |
| --- |
| // double denom2 = 0.0; |

|  |
| --- |
| // for (int i=0; i<itemI.size(); i++){ |

|  |
| --- |
| // double part1 = (itemI.get(i))\*(itemI.get(i)); |

|  |
| --- |
| // double part2 = (itemJ.get(i))\*(itemJ.get(i)); |

|  |
| --- |
| // denom1 = denom1 + part1; |

|  |
| --- |
| // denom2 = denom2 + part2; |

|  |
| --- |
| // } |

|  |
| --- |
| // denominator = Math.sqrt(denom1) \* Math.sqrt(denom2); |

|  |
| --- |
| // |

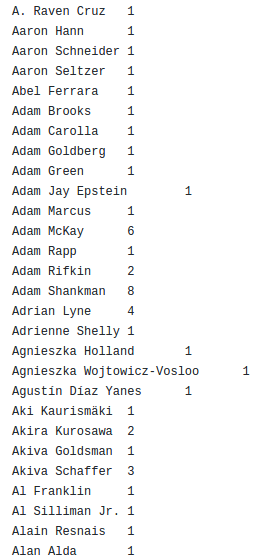
|  |
| --- |
| // similarity = numerator/denominator; |

|  |
| --- |
| } |

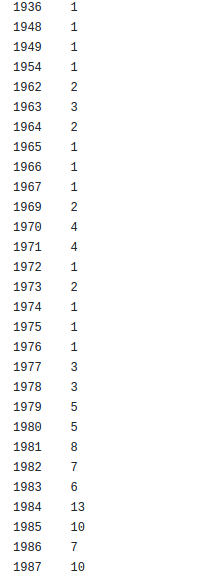
|  |
| --- |
| } |

**Results-**

**Question 1**

****

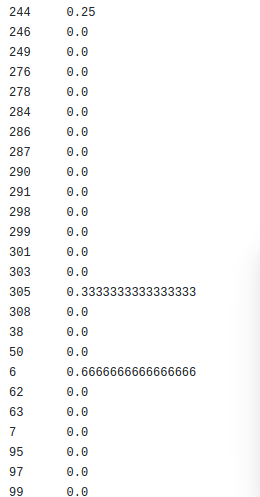
**Question2**

****

**Question3**

****

**Friend recommendation**

****

**Conclusion**

Hence with this project we have understood the need of Big Data and also understood the basic HDFS architecture and its working. We have also learnt the use of mapreduce for querying the database and obtaining the result. Here we have loaded our data set into hadoop database and queried it to get desired output using mapreduce code.