Case Study 1 – Movie Ratings data

The dataset provide for this case study has multiple csv files. But we will be using the following 2 files.

1. Movies.csv
2. Ratings.csv

Both these files are loaded on the hdfs using the below commands:

hadoop fs –put movies.csv /user/hadoop/

hadoop fs –put ratings.csv /user/hadoop/

Steps followed for performing a mapreduce job on the above data for the first 2 problems:

Solution Folder structure:

- In the eclipse project, there are 3 packages one for each Problem statement.
- The driver class for all the three is the same.
- There are 2 mappers one for each dataset movies and ratings. The mappers' code differs slightly for each problem statement.
- The reducer is where the join is performed on both the datasets and the resultant data is written on to HDFS in a part file.

We will be using reduce side join to join the datasets.

Mappers:

The mapper prepares the join operation by taking each input record from each of the data sets that is movies and rating dataset in our case and extracting the foreign key which is a movieid from each record.

The foreign key is written as the output key in MoviesDataMapper and RatingDataMapper and movie name is written as the value from MoviesDataMapper and Rating from RatingDataMapper. This output value is flagged by some unique identifier for the data set, such as M for Movies data and R for Rating data.

Reducer:

The reducer performs the desired join operation by collecting the values of each input group into temporary lists. For example, all records flagged with M are stored in

the movies list and all records flagged with R are stored in the rating list. These lists are then iterated over and the records from both sets are joined together.

Driver Class:

As we have two different dataset with different representations we need to parse the two input dataset differently. These cases are handled elegantly by using the MultipleInputs class, which allows you to specify the InputFormat and Mapper to use on a per-path basis. Below is the driver class.

```
package com.acadgild.movieratings.task1;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.lib.input.MultipleInputs;
import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

public class CaseStudyIUseCasesDriver {

        @SuppressWarnings("deprecation")
        public static void main(String[] args) throws Exception {
   if (args.length != 3) {
    System.err.println("Usage: CaseStudyIUseCase2Driver <input path1> <input path2> <output path>");
    System.exit(-1);
   }
        //Job Related Configurations
        Configuration conf = new Configuration();
        Job job = new Job(conf, "CaseStudyIUseCase2Driver");
        job.setJarByClass(CaseStudyIUseCasesDriver.class);
```

```
        job.setNumReduceTasks(1);


        //Since there are multiple input, there is a slightly different way of specifying input path, input format and mapper
        MultipleInputs.addInputPath(job, new Path(args[0]),TextInputFormat.class, CaseStudyIUseCasesMoviesMapper.class);
        MultipleInputs.addInputPath(job, new Path(args[1]),TextInputFormat.class, CaseStudyIUseCasesRatingsMapper.class);


        //Set the reducer
        job.setReducerClass(CaseStudyIUseCasesReducer.class);


   //set the out path
        Path outputPath = new Path(args[2]);
        FileOutputFormat.setOutputPath(job, outputPath);
        outputPath.getFileSystem(conf).delete(outputPath, true);


   //set up the output key and value classes
   job.setOutputKeyClass(Text.class);
   job.setOutputValueClass(Text.class);


   //execute the job
   System.exit(job.waitForCompletion(true) ? 0 : 1);
  }
}
```

Problem Statement 1:

What are the movie titles that the user has rated?

In this case, there are two mapper classes for the first job, one for movies i,e CaseStudyIUseCasesDriver and one for rating i,e CaseStudyIUseCasesDriver. In both, we extract the movie id to use it as the output key. We output the movie name and the rating value prepended in respective mappers with a character ‘M’ for a movies data or

'R' for a rating data so we know which data set the record came from during the reduce phase.

CaseStudyIUseCasesMoviesMapper

```
package com.acadgild.movieratings.task1;

import java.io.IOException;

import org.apache.hadoop.io.LongWritable;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Mapper;


public class CaseStudyIUseCasesMoviesMapper extends

                    Mapper<LongWritable, Text, Text, Text> {

public void map(LongWritable key, Text value, Context context)

                            throws IOException, InterruptedException {

          try {

      if (key.get() == 0 && value.toString().contains("movieId")){

         return;

      } else {

          String record = value.toString();

                            String[] parts = record.split(",");

                            System.out.println(parts[0]+" "+"movies\t" + parts[1]);

                            context.write(new Text(parts[0]), new Text("M" + parts[1]));

      }

   } catch (Exception e) {

      e.printStackTrace();

   }

}

}
```

CaseStudyIUseCasesRatingsMapper

```
package com.acadgild.movieratings.task1;
```

```
import java.io.IOException;

import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;

public class CaseStudyIUseCasesRatingsMapper extends
                        Mapper<LongWritable, Text, Text, Text> {

    public void map(LongWritable key, Text value, Context context)
                                throws IOException, InterruptedException {

                        try {
            if (key.get() == 0 && value.toString().contains("userId")){
               return;
            } else {
                                String record = value.toString();
                                String[] parts = record.split(",");
                                System.out.println(parts[1]+" "+"ratings\t" + parts[2]);
                                context.write(new Text(parts[1]), new Text("R" + parts[2]));
            }
          } catch (Exception e) {
            e.printStackTrace();
          }

    }
}
```

In these mappers, we are bringing the movieid, title of the name form movies mapper and movie id and the corresponding rating from the ratings mapper.

CaseStudyIUseCasesReducer

```
package com.acadgild.movieratings.task1;

import java.io.IOException;
import java.util.ArrayList;

import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Reducer;


public class CaseStudyIUseCasesReducer extends
                Reducer<Text, Text, Text, Text> {
        private ArrayList<Text> listMovies = new ArrayList<Text>();
        private ArrayList<Text> listRating = new ArrayList<Text>();

        public void reduce(Text key, Iterable<Text> values, Context context)
                        throws IOException, InterruptedException {
                listMovies.clear();
                listRating.clear();

                for(Text text:values) {
                        if(text.charAt(0)=='M') {
                                listMovies.add(new Text(text.toString().substring(1)));
                        }else if(text.charAt(0)=='R') {
                                listRating.add(new Text(text.toString().substring(1)));
                        }
                }

                executeJoinLogic(context);
```

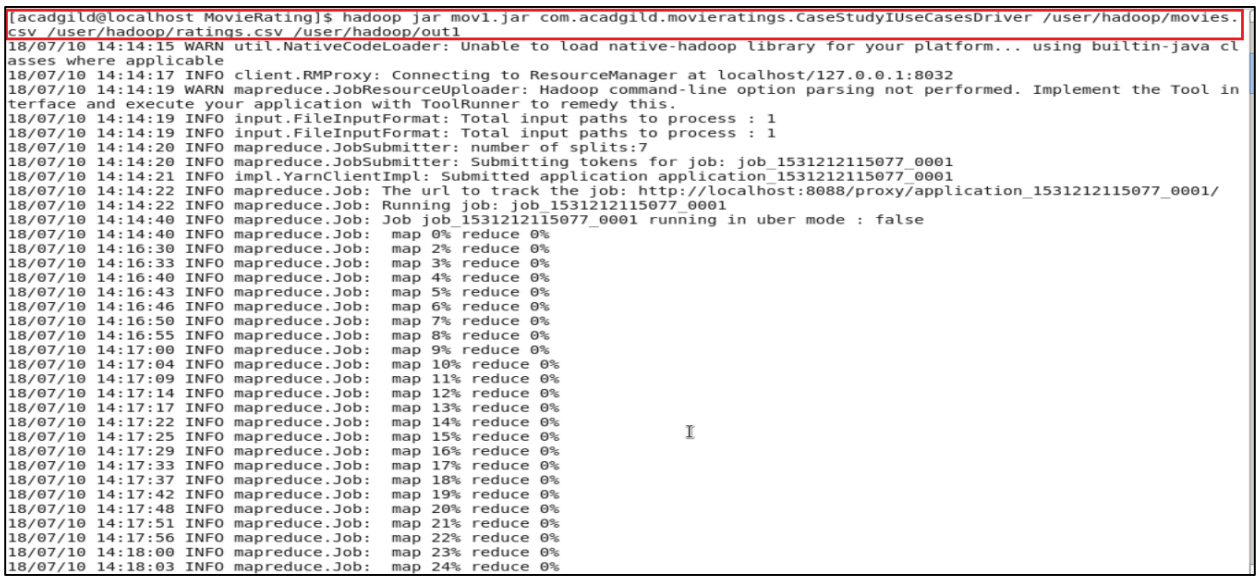
```
        }
        private void executeJoinLogic(Context context) throws IOException,InterruptedException{
            if(!listMovies.isEmpty()&&!listRating.isEmpty()) {
                for(Text moviesData:listMovies) {
                    context.write(moviesData,new Text(" has been rated by user"));
                }
            }
        }
}
```

In the reducer, we join both the datasets. Intially based on the first character of the value coming from mapper, we put the records into movies list or ratings list respectively.

Then we display the titles of the movies that are present In the ratings table, thus specified that the movie is rated by user.

Executing the jar:

hadoop jar mov1.jar com.acadgild.movieratings.task1.CaseStudyIUseCasesDriver /user/hadoop/movies.csv /user/hadoop/ratings.csv /user/hadoop/out1



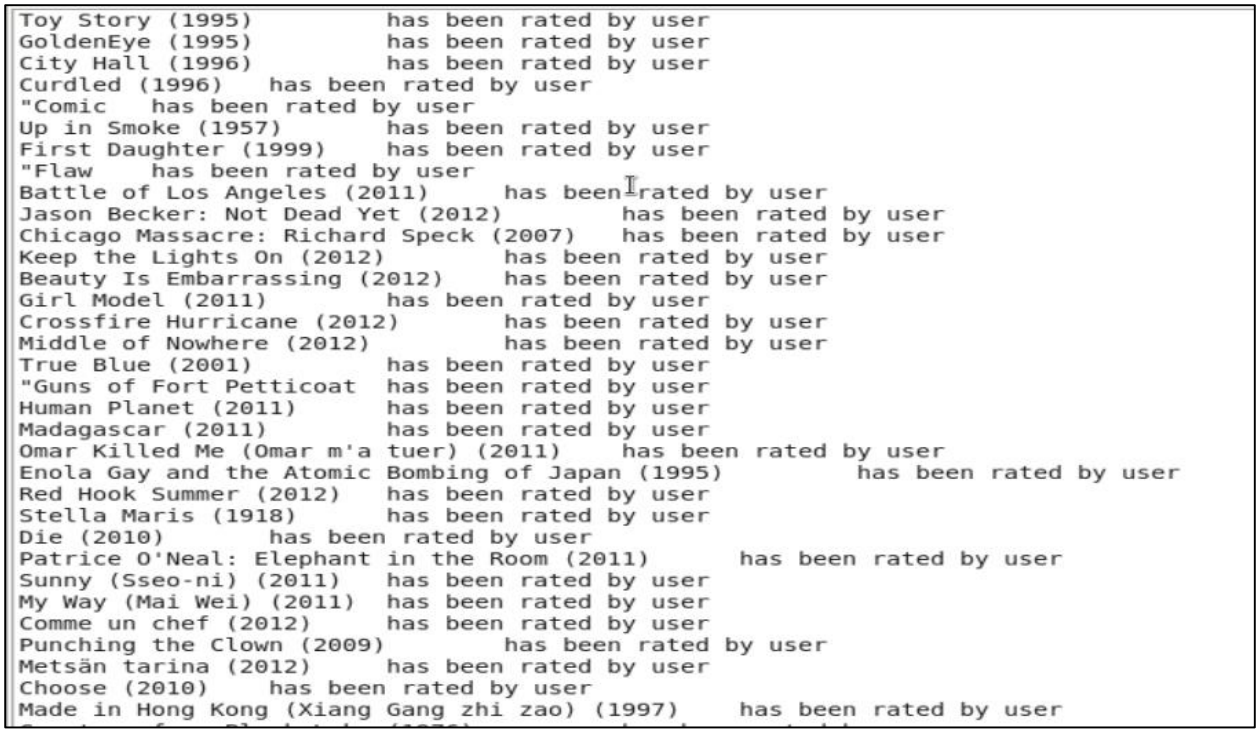
```
18/07/10 14:23:12 INFO mapreduce.Job:  map 98% reduce 14%
18/07/10 14:23:14 INFO mapreduce.Job:  map 99% reduce 14%
18/07/10 14:23:15 INFO mapreduce.Job:  map 100% reduce 14%
18/07/10 14:23:18 INFO mapreduce.Job:  map 100% reduce 24%
18/07/10 14:23:20 INFO mapreduce.Job:  map 100% reduce 33%
18/07/10 14:23:25 INFO mapreduce.Job:  map 100% reduce 67%
18/07/10 14:23:28 INFO mapreduce.Job:  map 100% reduce 68%
18/07/10 14:23:31 INFO mapreduce.Job:  map 100% reduce 72%
18/07/10 14:23:34 INFO mapreduce.Job:  map 100% reduce 75%
18/07/10 14:23:37 INFO mapreduce.Job:  map 100% reduce 79%
18/07/10 14:23:40 INFO mapreduce.Job:  map 100% reduce 82%
18/07/10 14:23:43 INFO mapreduce.Job:  map 100% reduce 86%
18/07/10 14:23:46 INFO mapreduce.Job:  map 100% reduce 90%
18/07/10 14:23:49 INFO mapreduce.Job:  map 100% reduce 93%
18/07/10 14:23:52 INFO mapreduce.Job:  map 100% reduce 96%
18/07/10 14:23:55 INFO mapreduce.Job:  map 100% reduce 100%
18/07/10 14:24:01 INFO mapreduce.Job: Job job_1531212115077_0001 completed successfully
18/07/10 14:24:05 INFO mapreduce.Job: Counters: 50
        File System Counters
                FILE: Number of bytes read=606802368
                FILE: Number of bytes written=920150115
                FILE: Number of read operations=0
                FILE: Number of large read operations=0
                FILE: Number of write operations=0
                HDFS: Number of bytes read=711856154
                HDFS: Number of bytes written=1240053
                HDFS: Number of read operations=24
                HDFS: Number of large read operations=0
                HDFS: Number of write operations=2
        Job Counters
                Killed map tasks=1
                Launched map tasks=8
                Launched reduce tasks=1
                Data-local map tasks=8
                Total time spent by all maps in occupied slots (ms)=2859859
                Total time spent by all reduces in occupied slots (ms)=249927
                Total time spent by all map tasks (ms)=2859859
                Total time spent by all reduce tasks (ms)=249927
                Total vcore-milliseconds taken by all map tasks=2859859
```

```
                Total time spent by all map tasks (ms)=2859859
                Total time spent by all reduce tasks (ms)=249927
                Total vcore-milliseconds taken by all map tasks=2859859
                Total vcore-milliseconds taken by all reduce tasks=249927
                Total megabyte-milliseconds taken by all map tasks=2928495616
                Total megabyte-milliseconds taken by all reduce tasks=255925248
        Map-Reduce Framework
                Map input records=26070134
                Map output records=26070132
                Map output bytes=260344746
                Map output materialized bytes=312485058
                Input split bytes=1937
                Combine input records=0
                Combine output records=0
                Reduce input groups=45843
                Reduce shuffle bytes=312485058
                Reduce input records=26070132
                Reduce output records=45115
                Spilled Records=76775523
                Shuffled Maps =7
                Failed Shuffles=0
                Merged Map outputs=7
                GC time elapsed (ms)=26456
                CPU time spent (ms)=370710
                Physical memory (bytes) snapshot=1365487616
                Virtual memory (bytes) snapshot=16449425408
                Total committed heap usage (bytes)=1018683392
        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=0
        File Output Format Counters
                Bytes Written=1240053
ou have new mail in /var/spool/mail/acadgild
```

Output is generated in the folder.

```
[acadgild@localhost MovieRating]$ hadoop fs -ls /user/hadoop/out1
18/07/10 22:36:03 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java cl
asses where applicable
Found 2 items
-rw-r--r--   1 acadgild supergroup          0 2018-07-10 14:23 /user/hadoop/out1/_SUCCESS
-rw-r--r--   1 acadgild supergroup    1240053 2018-07-10 14:23 /user/hadoop/out1/part-r-00000
You have new mail in /var/spool/mail/acadgild
```

The contents of the file.



List of movies that have been reviewd is displayed.

Problem Statement 2:

How many times a movie has been rated by the user?

Similar to the case above, there are two mapper classes for the first job, one for movies i,e CaseStudyIUseCasesDriver and one for rating i,e CaseStudyIUseCasesDriver. In both, we extract the movie id to use it as the output key. We output the movie name and the rating value prepended in respective mappers with a character ‘M’ for a movies data or ‘R’ for a rating data so we know which data set the record came from during the reduce phase.

CaseStudyIUseCasesMoviesMapper

```
package com.acadgild.movieratings.task2;


import java.io.IOException;


import org.apache.hadoop.io.LongWritable;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Mapper;
```

```
public class CaseStudyIUseCasesMoviesMapper extends
                    Mapper<LongWritable, Text, Text, Text> {
public void map(LongWritable key, Text value, Context context)
                            throws IOException, InterruptedException {


          try {
      if (key.get() == 0 && value.toString().contains("movieId")){
         return;
      } else {
          String record = value.toString();
                            String[] parts = record.split(",");
                            System.out.println(parts[0]+" "+"movies\t" + parts[1]);
                            context.write(new Text(parts[0]), new Text("M" + parts[1]));
      }
   } catch (Exception e) {
      e.printStackTrace();
   }
}
}
```

CaseStudyIUseCasesRatingsMapper

```
package com.acadgild.movieratings.task2;


import java.io.IOException;


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

```
public class CaseStudyIUseCasesRatingsMapper extends
                        Mapper<LongWritable, Text, Text, Text> {

    public void map(LongWritable key, Text value, Context context)
                                throws IOException, InterruptedException {

                        try {
            if (key.get() == 0 && value.toString().contains("userId")){
              return;
            } else {
                                String record = value.toString();
                                String[] parts = record.split(",");
                                System.out.println(parts[1]+" "+"ratings\t" + parts[2]);
                                context.write(new Text(parts[1]), new Text("R" + parts[2]));
            }
          } catch (Exception e) {
            e.printStackTrace();
          }

    }
}
```

In these mappers, we are bringing the movieid, title of the name form movies mapper and movie id and the corresponding rating from the ratings mapper.

CaseStudyIUseCasesReducer

```
package com.acadgild.movieratings.task2;

import java.io.IOException;
import java.util.ArrayList;
```

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


public class CaseStudyIUseCasesReducer extends
                Reducer<Text, Text, Text, Text> {
        private ArrayList<Text> listMovies = new ArrayList<Text>();
        private ArrayList<Text> listRating = new ArrayList<Text>();

        public void reduce(Text key, Iterable<Text> values, Context context)
                        throws IOException, InterruptedException {
                listMovies.clear();
                listRating.clear();

                for(Text text:values) {
                        if(text.charAt(0)=='M') {
                                listMovies.add(new Text(text.toString().substring(1)));
                        }else if(text.charAt(0)=='R') {
                                listRating.add(new Text(text.toString().substring(1)));
                        }
                }

                executeJoinLogic(context);

        }
        private void executeJoinLogic(Context context) throws IOException,InterruptedException{
                if(!listMovies.isEmpty()&&!listRating.isEmpty()) {
                        for(Text moviesData:listMovies) {
```

context.write(moviesData,new Text(String.valueOf(listRating.size())));

}

}

}

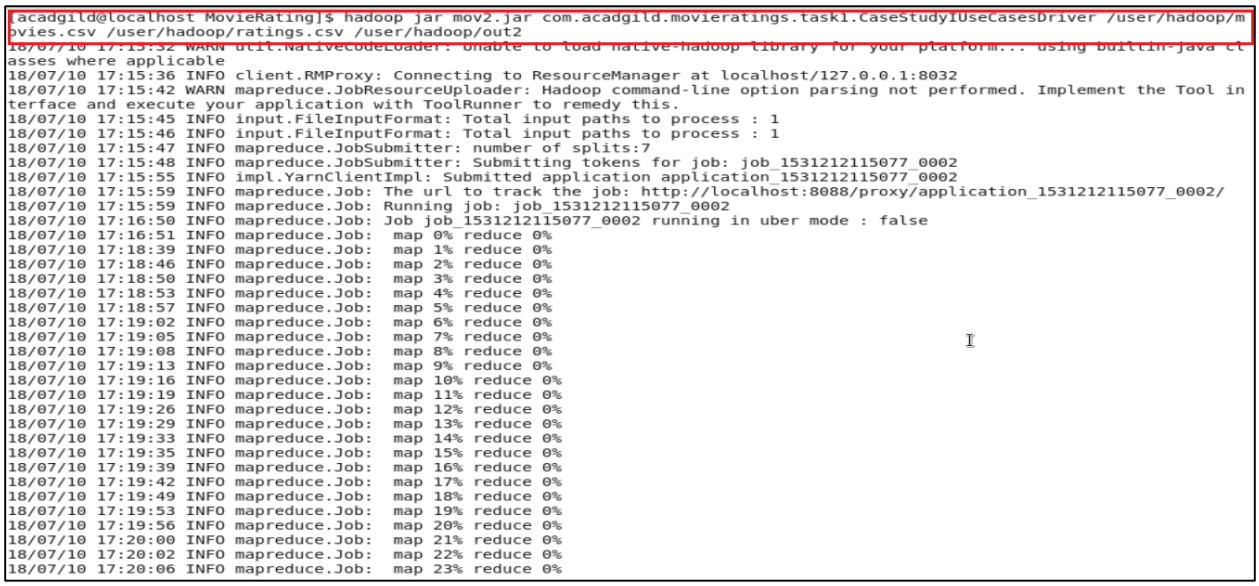
}

In the reducer here, the values from mapper are segregated into two lists movies and ratings similar to the above fashion.

Then the size of the ratings list for each movie is displayed along with the movie title, thus indicating how many times the movie appeared in the ratings table and thus how many times the movie was rated.

Executing the jar:

hadoop jar mov2.jar com.acadgild.movieratings.task2.CaseStudyIUseCasesDriver /user/hadoop/movies.csv /user/hadoop/ratings.csv /user/hadoop/out2



```
18/07/10 17:27:53 INFO mapreduce.Job:  map 100% reduce 14%
18/07/10 17:27:56 INFO mapreduce.Job:  map 100% reduce 24%
18/07/10 17:27:59 INFO mapreduce.Job:  map 100% reduce 29%
18/07/10 17:28:02 INFO mapreduce.Job:  map 100% reduce 33%
18/07/10 17:28:05 INFO mapreduce.Job:  map 100% reduce 67%
18/07/10 17:28:15 INFO mapreduce.Job:  map 100% reduce 70%
18/07/10 17:28:18 INFO mapreduce.Job:  map 100% reduce 73%
18/07/10 17:28:21 INFO mapreduce.Job:  map 100% reduce 77%
18/07/10 17:28:24 INFO mapreduce.Job:  map 100% reduce 80%
18/07/10 17:28:27 INFO mapreduce.Job:  map 100% reduce 83%
18/07/10 17:28:30 INFO mapreduce.Job:  map 100% reduce 87%
18/07/10 17:28:33 INFO mapreduce.Job:  map 100% reduce 90%
18/07/10 17:28:36 INFO mapreduce.Job:  map 100% reduce 94%
18/07/10 17:28:39 INFO mapreduce.Job:  map 100% reduce 97%
18/07/10 17:28:42 INFO mapreduce.Job:  map 100% reduce 100%
18/07/10 17:28:55 INFO mapreduce.Job: Job job_1531212115077_0002 completed successfully
18/07/10 17:29:04 INFO mapreduce.Job: Counters: 50
        File System Counters
                FILE: Number of bytes read=606802368
                FILE: Number of bytes written=920150259
                FILE: Number of read operations=0
                FILE: Number of large read operations=0
                FILE: Number of write operations=0
                HDFS: Number of bytes read=711856196
                HDFS: Number of bytes written=2196965
                HDFS: Number of read operations=24
                HDFS: Number of large read operations=0
                HDFS: Number of write operations=2
        Job Counters
                Killed map tasks=1
                Launched map tasks=8
                Launched reduce tasks=1
                Data-local map tasks=8
                Total time spent by all maps in occupied slots (ms)=3256711
                Total time spent by all reduces in occupied slots (ms)=147790
                Total time spent by all map tasks (ms)=3256711
                Total time spent by all reduce tasks (ms)=147790
                Total vcore-milliseconds taken by all map tasks=3256711
                Total vcore-milliseconds taken by all reduce tasks=147790
```

```
                Total time spent by all reduces in occupied slots (ms)=147790
                Total time spent by all map tasks (ms)=3256711
                Total time spent by all reduce tasks (ms)=147790
                Total vcore-milliseconds taken by all map tasks=3256711
                Total vcore-milliseconds taken by all reduce tasks=147790
                Total megabyte-milliseconds taken by all map tasks=3334872064
                Total megabyte-milliseconds taken by all reduce tasks=151336960
        Map-Reduce Framework
                Map input records=26070134
                Map output records=26070132
                Map output bytes=260344746
                Map output materialized bytes=312485058
                Input split bytes=1979
                Combine input records=0
                Combine output records=0
                Reduce input groups=45843
                Reduce shuffle bytes=312485058
                Reduce input records=26070132
                Reduce output records=45115
                Spilled Records=76775523
                Shuffled Maps =7
                Failed Shuffles=0
                Merged Map outputs=7
                GC time elapsed (ms)=29565
                CPU time spent (ms)=347600
                Physical memory (bytes) snapshot=1329532928
                Virtual memory (bytes) snapshot=16445984768
                Total committed heap usage (bytes)=1018683392
        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=0
        File Output Format Counters
                Bytes Written=2196965
```

Output file is generated in the folder.

```
[acadgild@localhost MovieRating]$ hadoop fs -ls /user/hadoop/out2
18/07/10 22:36:19 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java cl
asses where applicable
Found 2 items
-rw-r--r--   1 acadgild supergroup          0 2018-07-10 17:28 /user/hadoop/out2/_SUCCESS
-rw-r--r--   1 acadgild supergroup    2196965 2018-07-10 17:28 /user/hadoop/out2/part-r-00000
```

Part file contents:

```
Toy Story (1995)        66008
GoldenEye (1995)        32534
City Hall (1996)        4436
Curdled (1996)  217
"Comic  1
Up in Smoke (1957)      3
First Daughter (1999)   3
"Flaw   14
Battle of Los Angeles (2011)    44
Jason Becker: Not Dead Yet (2012)       9
Chicago Massacre: Richard Speck (2007)  2
Keep the Lights On (2012)       25
Beauty Is Embarrassing (2012)   15
Girl Model (2011)       32
Crossfire Hurricane (2012)      18
Middle of Nowhere (2012)        11
True Blue (2001)        3
"Guns of Fort Petticoat 3
Human Planet (2011)     197
Madagascar (2011)       26
Omar Killed Me (Omar m'a tuer) (2011)   9
Enola Gay and the Atomic Bombing of Japan (1995)        1
Red Hook Summer (2012)  11
Stella Maris (1918)     2
Die (2010)      10
Patrice O'Neal: Elephant in the Room (2011)     22
Sunny (Sseo-ni) (2011)  26
My Way (Mai Wei) (2011) 30
Comme un chef (2012)    83
Punching the Clown (2009)       12
Metsän tarina (2012)    8
Choose (2010)   19
Made in Hong Kong (Xiang Gang zhi zao) (1997)   3
```

Number of times each movie is reviewd is displayed.

Problem Statement 3:

In question 2 above, what is the average rating given for a movie?

In the mappers here, the movie id and movie title are pulled out and the movie title is concatenated with a string to specify that it has come from Movies dataset. Similar is the case with ratings. It is concatenated with a string to specify it comes from ratings dataset.

CaseStudyIUseCasesMoviesMapper

```
package com.acadgild.movieratings.task3;


import java.io.IOException;

import org.apache.hadoop.io.LongWritable;

import org.apache.hadoop.io.Text;
```

```
import org.apache.hadoop.mapreduce.Mapper;


public class CaseStudyIUseCasesMoviesMapper extends Mapper<LongWritable, Text, Text, Text> {

	public void map(LongWritable key, Text value, Context context) throws IOException,
InterruptedException {

		try {

			if (key.get() == 0 && value.toString().contains("movieId")) {

				return;

			} else {

				String record = value.toString();

				String[] parts = record.split(",");

				context.write(new Text(parts[0]), new Text("movies\t" + parts[1]));

			}

		} catch (Exception e) {

			e.printStackTrace();

		}

	}

}
```

CaseStudyIUseCasesRatingsMapper

```
package com.acadgild.movieratings.task3;


import java.io.IOException;

import org.apache.hadoop.io.LongWritable;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Mapper;


public class CaseStudyIUseCasesRatingsMapper extends Mapper<LongWritable, Text, Text, Text> {

	public void map(LongWritable key, Text value, Context context) throws IOException,
InterruptedException {
```

```
            try {
                if (key.get() == 0 && value.toString().contains("userId")) {
                    return;
                } else {
                    String record = value.toString();
                    String[] parts = record.split(",");
                    context.write(new Text(parts[1]), new Text("ratings\t" + parts[2]));
                }
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
}
```

In the reducer here, we will loop through the values present in the list of values in the reducer.
Then, we will split the list of values and check whether the value is of movies type or ratings type.
And then we will update the amount value to calculate the total of rated value of that movie.
On the other hand, if the value is of movies type, store it in a string variable, so that we will assign the titles as key in output key-value pair.
Next to get the average rating of a movie, we divide the total of rated value of a movie with no of time of that movie rated

CaseStudyIUseCasesReducer

```
package com.acadgild.movieratings.task3;


import java.io.IOException;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Reducer;


public class CaseStudyIUseCasesReducer extends Reducer<Text, Text, Text, Text> {
```

```
	public void reduce(Text key, Iterable<Text> values, Context context) throws IOException,
InterruptedException {
		String titles = "";
		double total = 0.0;
		int count = 0;
		System.out.println("Text Key =>" + key.toString());
		for (Text t : values) {
			String parts[] = t.toString().split("\t");
			System.out.println("Text values =>" + t.toString());
			if (parts[0].equals("ratings")) {
				count++;
				String rating = parts[1].trim();
				System.out.println("Rating is =>" + rating);
				total += Double.parseDouble(rating);
			} else if (parts[0].equals("movies")) {
				titles = parts[1];
			}
		}
		double average = total / count;
		String str = String.format("%f", average);
		context.write(new Text(titles), new Text(str));
	}
}
```

Executing the jar:

hadoop jar mov3.jar com.acadgild.movieratings.task3.CaseStudyIUseCasesDriver /user/hadoop/movies.csv /user/hadoop/ratings.csv /user/hadoop/out3

```
You have new mail in /var/spool/mail/acadgild
[acadgild@localhost MovieRating]$ hadoop jar mov3.jar com.acadgild.movieratings.task3.CaseStudyIUseCasesDriver /user/hadoop/movies.csv /user/hadoop/ratings.csv /user/hadoop/out3
18/07/10 21:48:47 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
18/07/10 21:48:51 INFO client.RMProxy: Connecting to ResourceManager at localhost/127.0.0.1:8032
18/07/10 21:48:55 WARN mapreduce.JobResourceUploader: Hadoop command-line option parsing not performed. Implement the Tool interface and execute your application with ToolRunner to remedy this.
18/07/10 21:48:59 INFO input.FileInputFormat: Total input paths to process : 1
18/07/10 21:48:59 INFO input.FileInputFormat: Total input paths to process : 1
18/07/10 21:49:01 INFO mapreduce.JobSubmitter: number of splits:7
18/07/10 21:49:03 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1531212115077_0003
18/07/10 21:49:08 INFO impl.YarnClientImpl: Submitted application application_1531212115077_0003
18/07/10 21:49:09 INFO mapreduce.Job: The url to track the job: http://localhost:8088/proxy/application_1531212115077_0003/
18/07/10 21:49:09 INFO mapreduce.Job: Running job: job_1531212115077_0003
18/07/10 21:50:08 INFO mapreduce.Job: Job job_1531212115077_0003 running in uber mode : false
18/07/10 21:50:08 INFO mapreduce.Job:  map 0% reduce 0%
18/07/10 21:51:34 INFO mapreduce.Job:  map 2% reduce 0%
18/07/10 21:51:35 INFO mapreduce.Job:  map 6% reduce 0%
18/07/10 21:51:39 INFO mapreduce.Job:  map 10% reduce 0%
18/07/10 21:51:42 INFO mapreduce.Job:  map 11% reduce 0%
18/07/10 21:51:43 INFO mapreduce.Job:  map 14% reduce 0%
18/07/10 21:51:46 INFO mapreduce.Job:  map 18% reduce 0%
18/07/10 21:51:50 INFO mapreduce.Job:  map 21% reduce 0%
18/07/10 21:51:51 INFO mapreduce.Job:  map 22% reduce 0%
18/07/10 21:51:52 INFO mapreduce.Job:  map 23% reduce 0%
18/07/10 21:51:53 INFO mapreduce.Job:  map 25% reduce 0%
18/07/10 21:51:56 INFO mapreduce.Job:  map 26% reduce 0%
18/07/10 21:52:01 INFO mapreduce.Job:  map 29% reduce 0%
18/07/10 21:52:05 INFO mapreduce.Job:  map 31% reduce 0%
18/07/10 21:52:06 INFO mapreduce.Job:  map 32% reduce 0%
18/07/10 21:52:10 INFO mapreduce.Job:  map 37% reduce 0%
18/07/10 21:52:31 INFO mapreduce.Job:  map 43% reduce 0%
18/07/10 21:52:35 INFO mapreduce.Job:  map 44% reduce 0%
18/07/10 21:52:36 INFO mapreduce.Job:  map 45% reduce 0%
18/07/10 21:53:15 INFO mapreduce.Job:  map 46% reduce 0%
18/07/10 21:53:17 INFO mapreduce.Job:  map 48% reduce 0%
18/07/10 21:53:19 INFO mapreduce.Job:  map 49% reduce 0%
18/07/10 21:53:24 INFO mapreduce.Job:  map 54% reduce 0%
```

```
18/07/10 22:01:02 INFO mapreduce.Job:  map 100% reduce 88%
18/07/10 22:01:11 INFO mapreduce.Job:  map 100% reduce 89%
18/07/10 22:01:20 INFO mapreduce.Job:  map 100% reduce 90%
18/07/10 22:01:29 INFO mapreduce.Job:  map 100% reduce 91%
18/07/10 22:01:39 INFO mapreduce.Job:  map 100% reduce 92%
18/07/10 22:01:48 INFO mapreduce.Job:  map 100% reduce 93%
18/07/10 22:01:57 INFO mapreduce.Job:  map 100% reduce 94%
18/07/10 22:02:06 INFO mapreduce.Job:  map 100% reduce 95%
18/07/10 22:02:15 INFO mapreduce.Job:  map 100% reduce 96%
18/07/10 22:02:24 INFO mapreduce.Job:  map 100% reduce 97%
18/07/10 22:02:33 INFO mapreduce.Job:  map 100% reduce 98%
18/07/10 22:02:42 INFO mapreduce.Job:  map 100% reduce 99%
18/07/10 22:02:51 INFO mapreduce.Job:  map 100% reduce 100%
18/07/10 22:03:03 INFO mapreduce.Job: Job job_1531212115077_0003 completed successfully
18/07/10 22:03:08 INFO mapreduce.Job: Counters: 50
        File System Counters
                FILE: Number of bytes read=961694264
                FILE: Number of bytes written=1457487206
                FILE: Number of read operations=0
                FILE: Number of large read operations=0
                FILE: Number of write operations=0
                HDFS: Number of bytes read=711856196
                HDFS: Number of bytes written=1669001
                HDFS: Number of read operations=24
                HDFS: Number of large read operations=0
                HDFS: Number of write operations=2
        Job Counters
                Killed map tasks=1
                Launched map tasks=7
                Launched reduce tasks=1
                Data-local map tasks=8
                Total time spent by all maps in occupied slots (ms)=1697414
                Total time spent by all reduces in occupied slots (ms)=538561
                Total time spent by all map tasks (ms)=1697414
                Total time spent by all reduce tasks (ms)=538561
                Total vcore-milliseconds taken by all map tasks=1697414
                Total vcore-milliseconds taken by all reduce tasks=538561
                Total megabyte-milliseconds taken by all map tasks=1738151936
                Total megabyte-milliseconds taken by all reduce tasks=551486464
```

```
                Total time spent by all map tasks (ms)=1697414
                Total time spent by all reduce tasks (ms)=538561
                Total vcore-milliseconds taken by all map tasks=1697414
                Total vcore-milliseconds taken by all reduce tasks=538561
                Total megabyte-milliseconds taken by all map tasks=1738151936
                Total megabyte-milliseconds taken by all reduce tasks=551486464
        Map-Reduce Framework
                Map input records=26070134
                Map output records=26070132
                Map output bytes=442789828
                Map output materialized bytes=494930141
                Input split bytes=1979
                Combine input records=0
                Combine output records=0
                Reduce input groups=45843
                Reduce shuffle bytes=494930141
                Reduce input records=26070132
                Reduce output records=45843
                Spilled Records=76775523
                Shuffled Maps =7
                Failed Shuffles=0
                Merged Map outputs=7
                GC time elapsed (ms)=20864
                CPU time spent (ms)=459510
                Physical memory (bytes) snapshot=1478062080
                Virtual memory (bytes) snapshot=16448622592
                Total committed heap usage (bytes)=1032732672
        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=0
        File Output Format Counters
                Bytes Written=1669001
```

Output file is generated.

```
[acadgild@localhost MovieRating]$ hadoop fs -ls /user/hadoop/out3
18/07/10 22:36:28 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java cl
asses where applicable
Found 2 items
-rw-r--r--   1 acadgild supergroup          0 2018-07-10 22:27 /user/hadoop/out3/_SUCCESS
-rw-r--r--   1 acadgild supergroup    1541697 2018-07-10 22:27 /user/hadoop/out3/part-r-00000
```

Contents of output file

```
Toy Story (1995)        3.888157
GoldenEye (1995)        3.431841
City Hall (1996)        3.232304
Curdled (1996)  3.099078
"Comic  4.000000
Up in Smoke (1957)      3.666667
First Daughter (1999)   3.333333
"Flaw   3.714286
Battle of Los Angeles (2011)    2.522727
Jason Becker: Not Dead Yet (2012)       3.444444
Chicago Massacre: Richard Speck (2007)  2.500000
Keep the Lights On (2012)       3.100000
Beauty Is Embarrassing (2012)   3.600000
Girl Model (2011)       3.281250
Crossfire Hurricane (2012)      3.388889
Middle of Nowhere (2012)        3.454545
True Blue (2001)        3.000000
"Guns of Fort Petticoat 3.333333
Human Planet (2011)     4.271574
Madagascar (2011)       3.769231
Omar Killed Me (Omar m'a tuer) (2011)   3.166667
Enola Gay and the Atomic Bombing of Japan (1995)        3.500000
Red Hook Summer (2012)  2.045455
Stella Maris (1918)     3.750000
Die (2010)      2.550000
Patrice O'Neal: Elephant in the Room (2011)     3.204545
Sunny (Sseo-ni) (2011)  3.576923
My Way (Mai Wei) (2011) 3.716667
Comme un chef (2012)    3.626506
Punching the Clown (2009)       3.541667
Metsän tarina (2012)    2.875000
Choose (2010)   2.815789
Made in Hong Kong (Xiang Gang zhi zao) (1997)   2.333333
```

Plain Text

Average rating of each movie is displayed.