Average-Chaining-Sorting

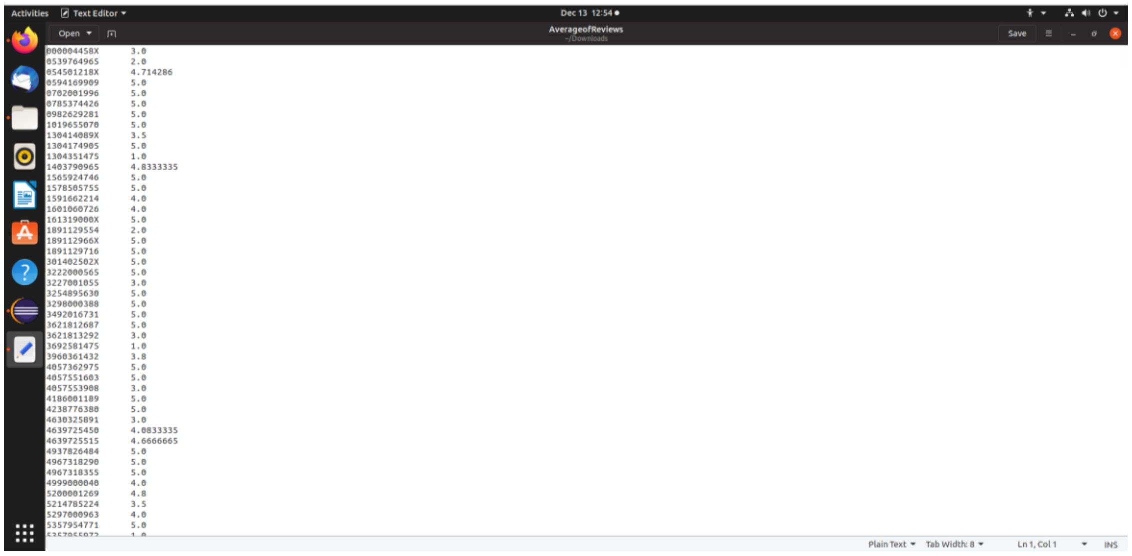
(Numerical Summarization Pattern & Job Chaining)

This MR job uses the chaining where the output of one MapReduce job goes to the other before finally outputting it to the user. There are 2 mappers and 2 reducers. The first map reduce will give average of ratings

OUTPUT:

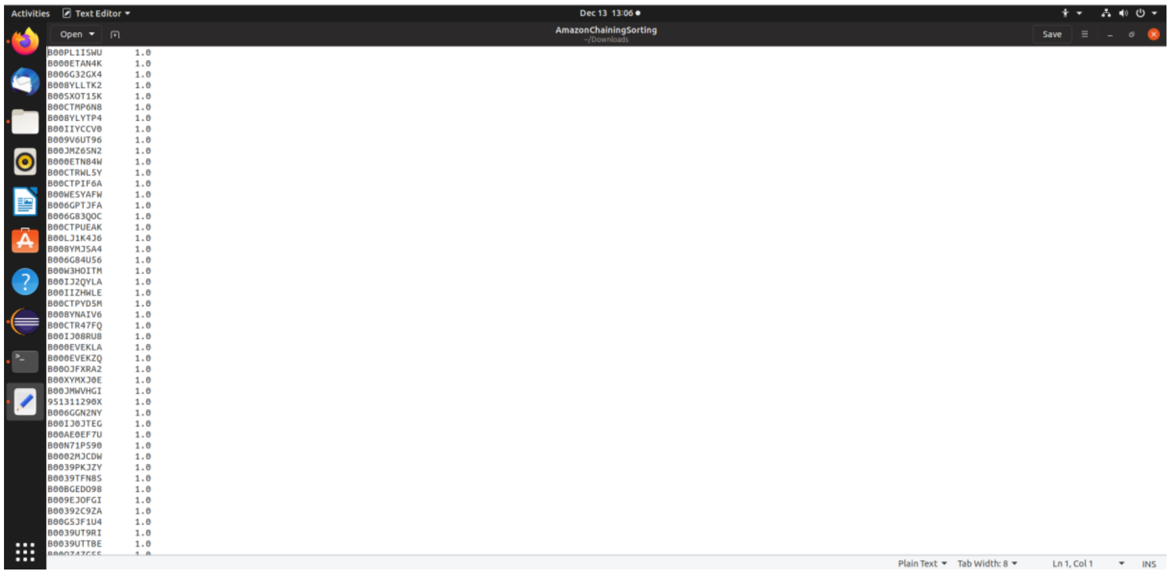
```
./hadoop jar /home/sayali/Desktop/AverageSorting.jar sayali.AmazonReviews1.AverageSorting
/AmazonReviews/AmazonReviews.tsv  /AverageReviews /SortedReviews
```

OUTPUT of First Map reduce :



```
000004458X      3.0
0539764965      2.0
054501218X      4.714286
0594169909      5.0
0702001996      5.0
0785374426      5.0
0982629281      5.0
1019655070      5.0
130414089X      3.5
1304174905      5.0
1304351475      1.0
1403790965      4.8333335
1565924746      5.0
1578505755      5.0
1591662214      4.0
1601060726      4.0
161319000X      5.0
1891129554      2.0
189112966X      5.0
1891129716      5.0
301402502X      5.0
3222000565      5.0
3227001055      3.0
3254895630      5.0
3298000388      5.0
3492016731      5.0
3621812687      5.0
3621813292      3.0
3692581475      1.0
3960361432      3.8
4057362975      5.0
4057551603      5.0
4057553908      3.0
4186001189      5.0
4238776380      5.0
4630325891      3.0
4639725450      4.0833335
4639725515      4.6666665
4937826484      5.0
4967318290      5.0
4967318355      5.0
4999000040      4.0
5200001269      4.8
5214785224      3.5
5297000963      4.0
5357954771      5.0
```

OUTPUT of Second Map reduce:



```
B00PL1ISWU      1.0
B000ETAN4K      1.0
B006G32GX4      1.0
B008YLLTK2      1.0
B00SXOT15K      1.0
B00CTMP6N8      1.0
B008YLYTP4      1.0
B00IIYCCV0      1.0
B009V6UT96      1.0
B00JMZ6SN2      1.0
B000ETN84W      1.0
B00CTRWL5Y      1.0
B00CTPIF6A      1.0
B00WESYAFW      1.0
B006GPTJFA      1.0
B006G83QOC      1.0
B00CTPUEAK      1.0
B00LJ1K4J6      1.0
B008YMJSA4      1.0
B006G84U56      1.0
B00W3HOITM      1.0
B00IJ2QYLA      1.0
B00IIZHWLE      1.0
B00CTPYD5M      1.0
B008YNAIV6      1.0
B00CTR47FQ      1.0
B00IJ08RU8      1.0
B000EVEKLA      1.0
B000EVEKZQ      1.0
B000JFXRA2      1.0
B00XYMXJ0E      1.0
B00JMWVHGI      1.0
951311290X      1.0
B006GGN2NY      1.0
B00IJ0JTEG      1.0
B00AE0EF7U      1.0
B00N71P590      1.0
B0002MJCDW      1.0
B0039PKJZY      1.0
B0039TFN8S      1.0
B00BGEDO98      1.0
B009EJOFGI      1.0
B00392C9ZA      1.0
B00G5JF1U4      1.0
B0039UT9RI      1.0
B0039UTTBE      1.0
```

Map reduce code

```java
package sayali.AmazonReviews1;
import java.io.IOException;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.FloatWritable;
import org.apache.hadoop.io.LongWritable;
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;
```

Mapper 1:

```java
public class AverageSortingMapper1 extends Mapper<LongWritable, Text, Text, FloatWritable> {
	private Text text = new Text();
	private FloatWritable score = new FloatWritable();

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

		if (key.get() == 0) {
			return;
		}
		else {
			String[] line = value.toString().split("\\t");
			String productId = line[3].trim();
			float ratingVal = Float.valueOf(line[7].trim());
```

```
                    text.set(productId);
                    score.set(ratingVal);

                    context.write(text, score);
                }
            }
}
```

Reducer 1:

```
public class AverageSortingReducer1 extends Reducer<Text, FloatWritable, Text, FloatWritable> {
        private FloatWritable result = new FloatWritable();

        @Override
        protected void reduce(Text key, Iterable<FloatWritable> values, Context context)
                        throws IOException, InterruptedException {

                float sum = 0;
                int count = 0;
                for (FloatWritable val : values) {
                        sum += val.get();
                        count = count + 1; }
                float average = sum / count;
                result.set(average);
                context.write(key, result);
        }
}
```

Mapper 2:

```
public class AverageSortingMapper2 extends Mapper<LongWritable, Text, FloatWritable, Text> {

        public void map(LongWritable key, Text value, Context context) {
                String[] row = value.toString().split("\\t");
                Text Id = new Text(row[0]);
                float Ratings = Float.valueOf(row[1].trim());
      try {
                        FloatWritable count = new FloatWritable(Ratings);
                        context.write(count, Id);
                }
                        catch (Exception e) {
                }
        }
}
```

Reducer 2:

```
public class AverageSortingReducer2 extends Reducer<FloatWritable, Text, Text, FloatWritable> {

        public void reduce(FloatWritable key, Iterable<Text> value, Context context)
                        throws IOException, InterruptedException {
```

```
                    for (Text val : value) {

                              context.write(val, key);
                    }
          }
}
```

Main Class:

```
public class AverageSorting {
          public static void main(String[] args) throws IOException, InterruptedException,
ClassNotFoundException {
                    Configuration conf1 = new Configuration();
                    Job job1 = Job.getInstance(conf1, "Amazon Average");
                    job1.setJarByClass(AverageSorting.class);
                    job1.setMapperClass(AverageSortingMapper1.class);
                    job1.setMapOutputKeyClass(Text.class);
                    job1.setMapOutputValueClass(FloatWritable.class);

                    job1.setReducerClass(AverageSortingReducer1.class);
                    job1.setOutputKeyClass(Text.class);
                    job1.setOutputValueClass(FloatWritable.class);

                    FileInputFormat.addInputPath(job1, new Path(args[0]));
                    FileOutputFormat.setOutputPath(job1, new Path(args[1]));
                    boolean complete = job1.waitForCompletion(true);

                    Configuration conf2 = new Configuration();
                    Job job2 = Job.getInstance(conf2, "Chaining Sorting");

                    if (complete) {
                              job2.setJarByClass(AverageSorting.class);
                              job2.setMapperClass(AverageSortingMapper2.class);
                              job2.setMapOutputKeyClass(FloatWritable.class);
                              job2.setMapOutputValueClass(Text.class);

                              job2.setReducerClass(AverageSortingReducer2.class);
                              job2.setOutputKeyClass(Text.class);
                              job2.setOutputValueClass(FloatWritable.class);

                              FileInputFormat.addInputPath(job2, new Path(args[1]));
                              FileOutputFormat.setOutputPath(job2, new Path(args[2]));

                              System.exit(job2.waitForCompletion(true) ? 0 : 1);
                    }
          }
}
```