Large Scale Data Processing
WINTER SEMESTER 2017-18

INSTRUCTOR: N. MAHESWARI

CHALLA DINESH REDDY
15BCE1274

IAB PROGRAMS

Contents:-

- **Lab1 Introduction and basic commands in Hadoop.**
- **Lab2 basic wordcount Mapreduce program**
- **Lab3 part1 Find maximum occurring word**
- **Lab3 part2 total word count**
- **Lab4 part1 Combiner**
- **Lab4 part2 Check for certain range of value and output it.(salary>1lakh)**
- **Lab5 custom partitioning**
- **Lab6 Working with different i/o formats.**
- **Lab7 part1 Find top k records based on a value(Tree Map)**
- **Lab7 part2 User defined Counters**
- **Lab8 Side Data**
- **Lab9 part 1 Distributed Cache**
- **Lab9 part 2 Reduce side join Customer details and transaction details.**
- **Lab 10 Secondary sorting**

Lab 1:- basic hadoop commands.

/////hadoop commands

Start-all.sh:- starts all the demons

Jps:- shows the jobs or daemons running along with the ports.

hadoop dfs -ls <path> :- lists the files in hdfs current

hadoop dfs -put <local> <dst> :- puts the data file from local machine to hdfs

hadoop dfs -cat :- shows the file in terminal.

hadoop dfs -get :- get files from hdfs to local machine

hadoop dfs -rm :- removes files in hdfs

hadoop dfs -fsck :- Runs a HDFS file system checking utility.

hadoop jar :- run Mapreduce Programming

hadoop distcp <src> <dst> :- copy from hdfs to hdfs

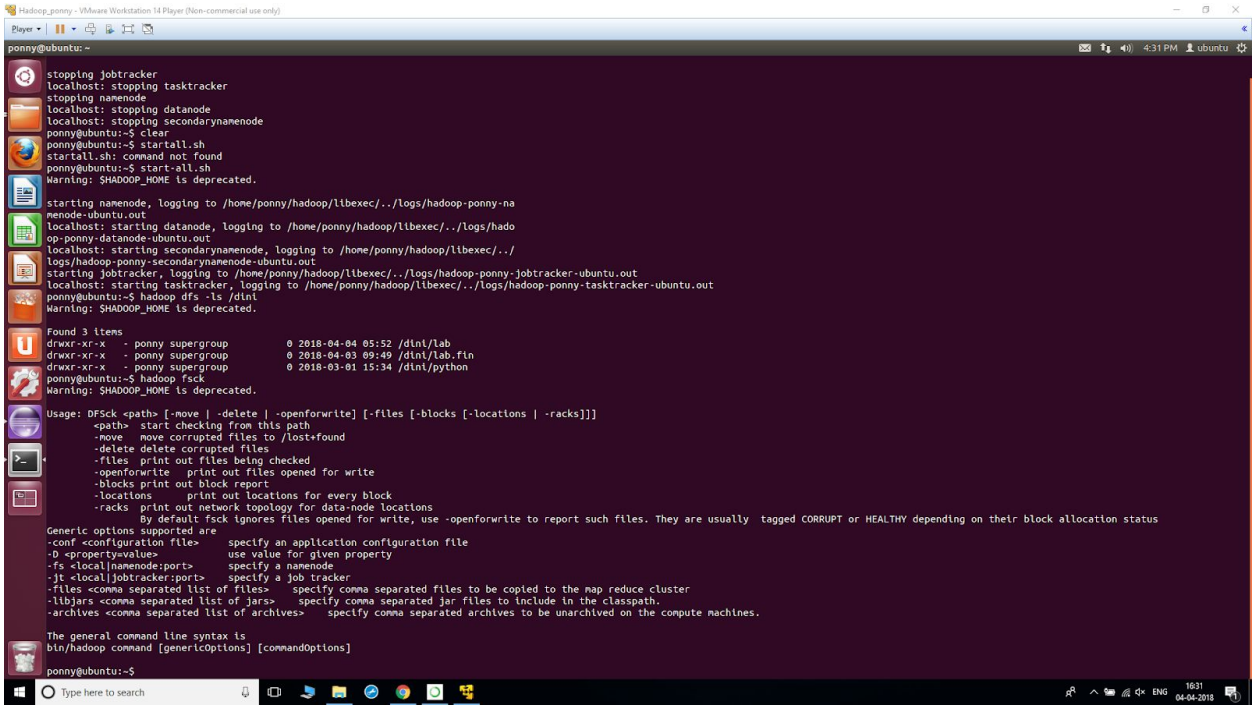
///web ui localhost:<port>

default ports:-

Namenode	50070
Datanodes	50075
Secondarynamenode	50090
Backup/Checkpoint node	50105
Jobracker	50030

Tasktrackers 50060

Output:-



Lab2:- First Map Reduce Program. Word count.

Algorithm:

Map Function – It takes a set of data and converts it into another set of data, where individual elements are broken down into tuples (Key-Value pair).

Example – (Map function in Word Count)

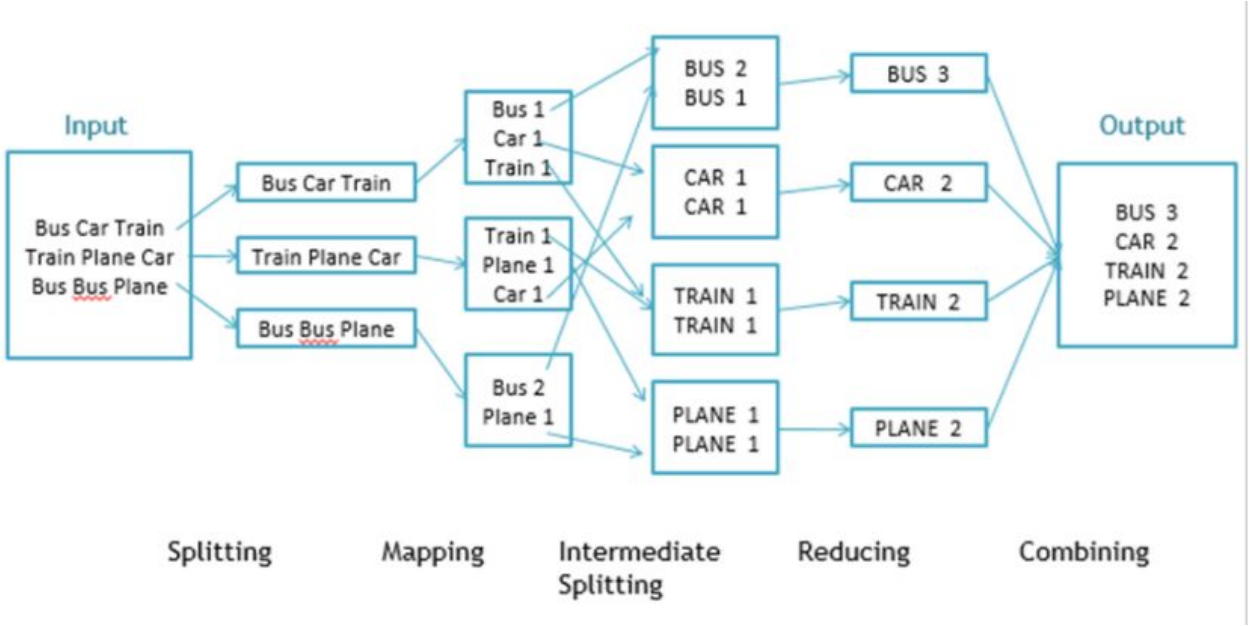
Input	Set of data	Bus, Car, bus, car, train, car, bus, car, train, bus, TRAIN,BUS, buS, caR, CAR, car, BUS, TRAIN
Output	Convert into another set of data (Key,Value)	(Bus,1), (Car,1), (bus,1), (car,1), (train,1), (car,1), (bus,1), (car,1), (train,1), (bus,1), (TRAIN,1),(BUS,1), (buS,1), (caR,1), (CAR,1), (car,1), (BUS,1), (TRAIN,1)

Reduce Function:-Takes the output from Map as an input and combines those data tuples into a smaller set of tuples.

Example – (Reduce function in Word Count)

Input **(output of Map function)**	Set of Tuples	(Bus,1), (Car,1), (bus,1), (car,1), (train,1), (car,1), (bus,1), (car,1), (train,1), (bus,1), (TRAIN,1),(BUS,1), (buS,1), (caR,1), (CAR,1), (car,1), (BUS,1), (TRAIN,1)
Output	Converts into smaller set of tuples	(BUS,7), (CAR,7), (TRAIN,4)

Overall workflow of Mapreduce program:-



Java code:-

```
//basic wordcount

import java.io.IOException;

import java.util.StringTokenizer;



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;
```

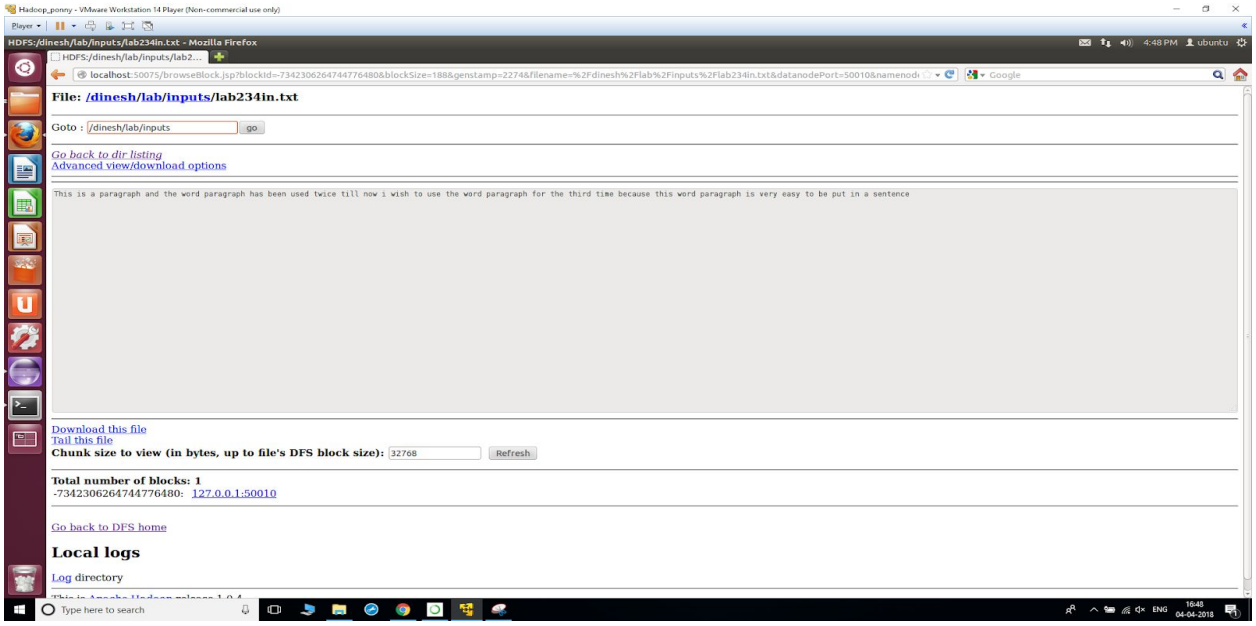
```
public class WordCount{

	public static class Map extends Mapper<Object, Text, Text, IntWritable>{

		private final static IntWritable one = new IntWritable(1);

		private Text word = new Text();


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

			StringTokenizer st = new StringTokenizer(value.toString());

			while(st.hasMoreTokens()){

				word.set(st.nextToken());

				context.write(word,one);

			}


		}


	}


	public static class Reduce 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));
                }
        }

        public static void main(String[] args) throws Exception{
                Configuration conf = new Configuration();
                Job job = new Job(conf,"wordcount");
                job.setJarByClass(WordCount.class);
                job.setMapperClass(Map.class);
                job.setOutputKeyClass(Text.class);
                job.setOutputValueClass(IntWritable.class);
                job.setReducerClass(Reduce.class);
                //job.setInputFormatClass(TextInputFormat.class);
                //job.setOutputFormatClass(TextOutputFormat.class);
                FileInputFormat.addInputPath(job, new Path(args[0]));
                FileOutputFormat.setOutputPath(job, new Path(args[1]));
                job.waitForCompletion(true);
        }
}
```

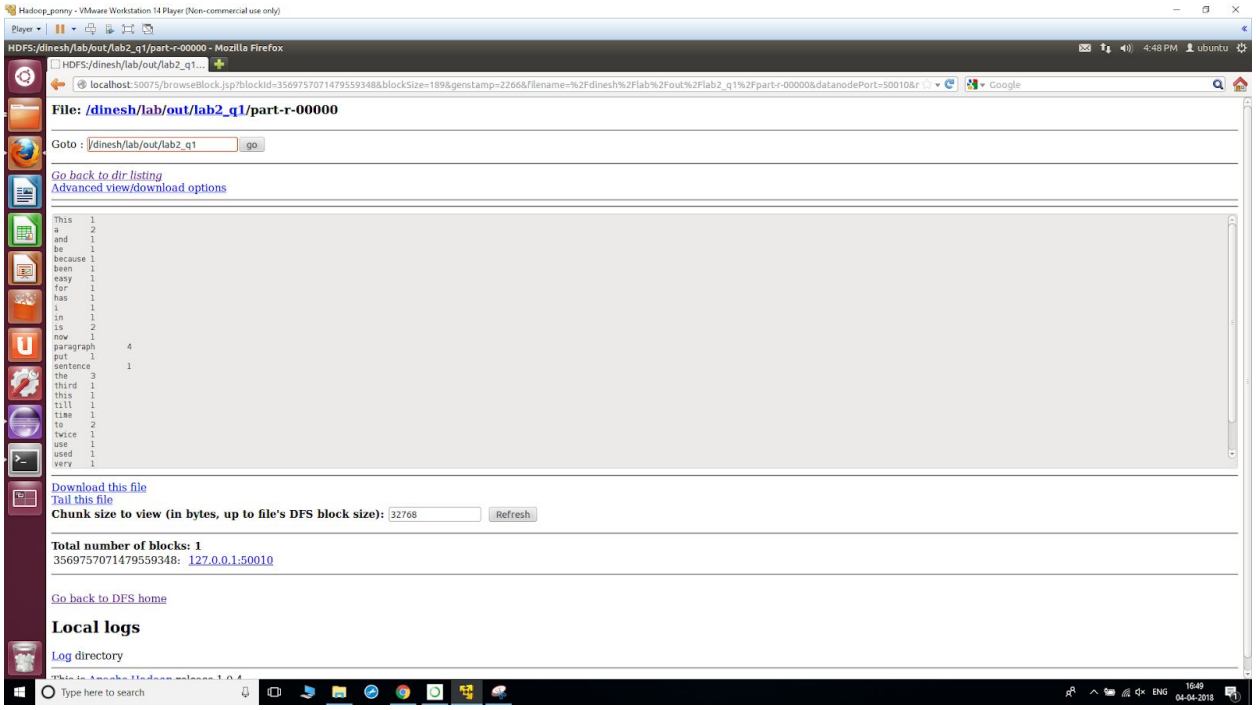
Execution:-

hadoop jar < jar file path <class name> <input path in hdfs> <output path in hdfs>

Input:-



Output:-



Lab3:-

q1)Find the maximum occurring word and write the no of time it occured.

Algorithm:-

In reducer have a seperate variable max which is initialised to 0 and it keeps on checking for all words, replace it the count greater than previous and finally use cleanup to write the output to hdfs.

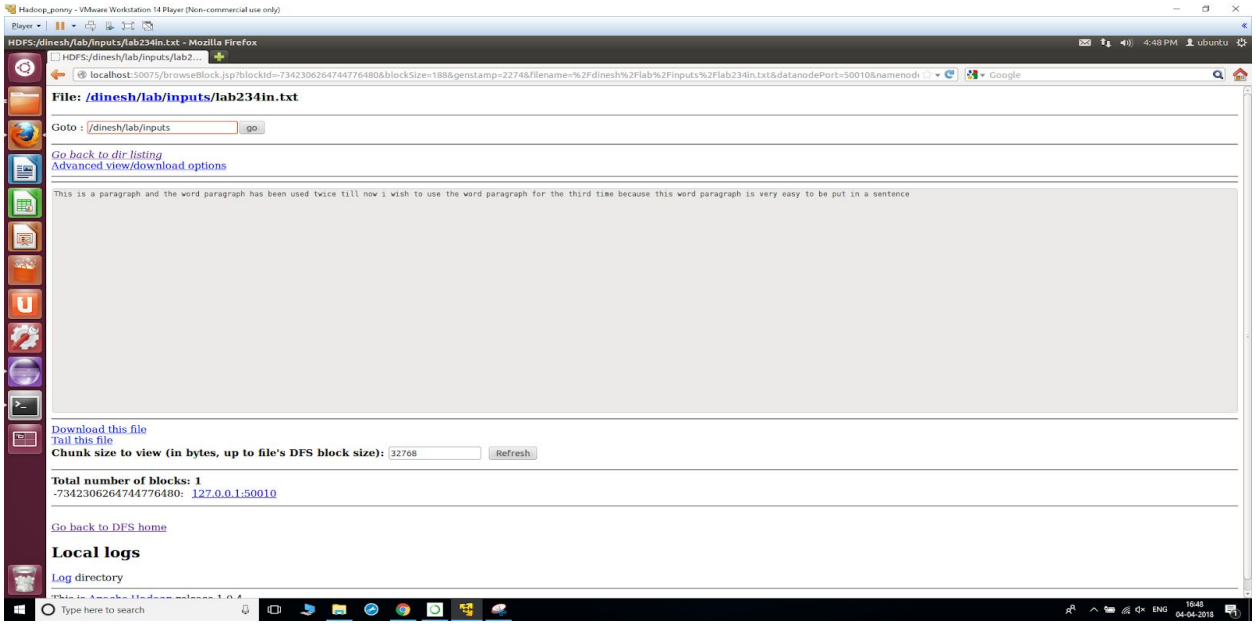
Code:-

```
import java.io.IOException;

import java.util.StringTokenizer;



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;



public class MaxCount{

	public static class Map extends Mapper<Object, Text, Text, IntWritable>{

		private final static IntWritable one = new IntWritable(1);

		private Text word = new Text();



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

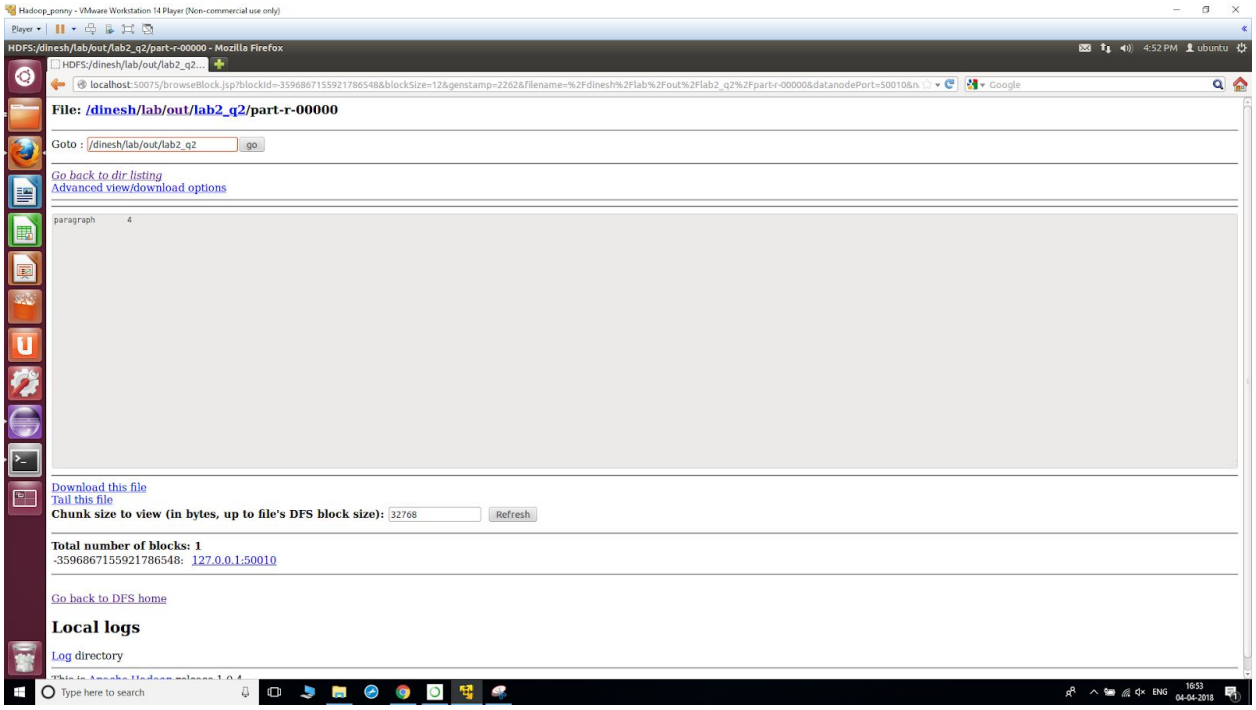
```
                StringTokenizer st = new StringTokenizer(value.toString());
                while(st.hasMoreTokens()){
                        word.set(st.nextToken());
                        context.write(word,one);
                }

        }

}

public static class Reduce extends Reducer<Text, IntWritable, Text, IntWritable>{
        int maxSum = 0;
        Text maxOccuredKey = new Text();
        public void reduce(Text key, Iterable<IntWritable> values, Context context) throws IOException,InterruptedException{
                int sum = 0;
                for(IntWritable val: values){
                        sum += val.get();
                }
                if(sum > maxSum){
                        maxSum = sum;
                        maxOccuredKey.set(key);
                }
        }
```

```
        @Override
        protected void cleanup(Context context) throws IOException, InterruptedException{
          context.write(maxOccuredKey, new IntWritable(maxSum));
        }
    }

    public static void main(String[] args) throws Exception{
        Configuration conf = new Configuration();
        Job job = new Job(conf,"maxcount");
        job.setJarByClass(MaxCount.class);
        job.setMapperClass(Map.class);
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(IntWritable.class);
        job.setReducerClass(Reduce.class);
        FileInputFormat.addInputPath(job, new Path(args[0]));
        FileOutputFormat.setOutputPath(job, new Path(args[1]));
        job.waitForCompletion(true);
    }
}
```

Input:- Same as that of lab2.



Output:-



q2) Find the total number of words. Total word count.

Algorithm:- In reducer, add up the individual counts which are in the value of key value pair.

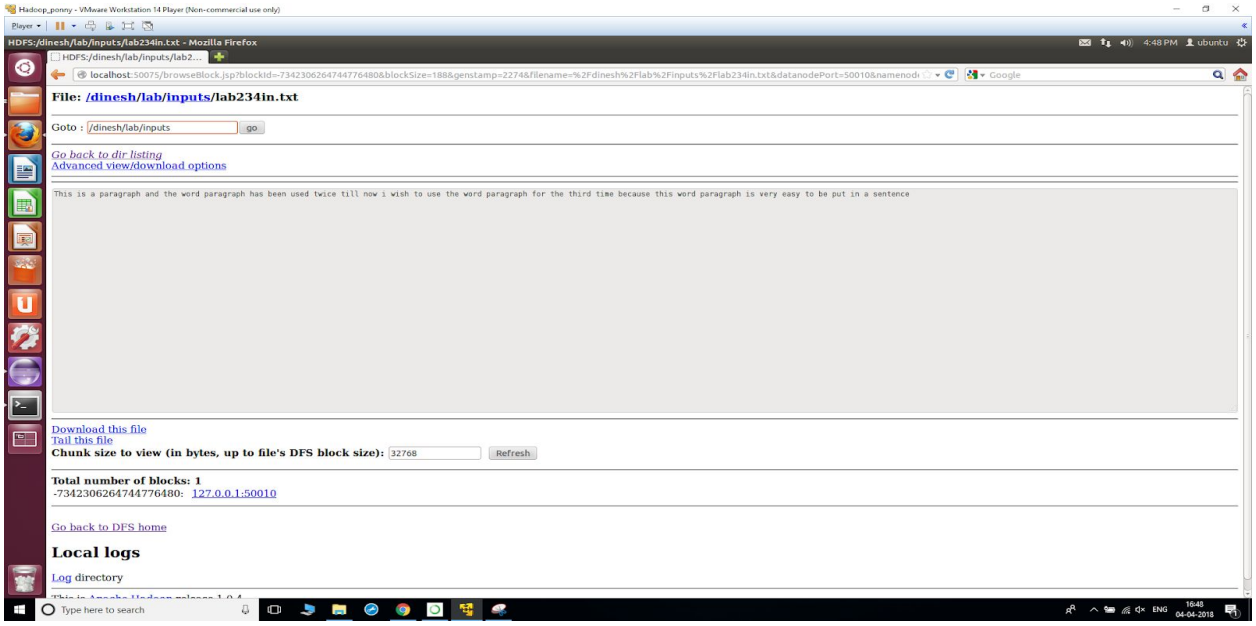
Code:-

```
import java.io.IOException;

import java.util.StringTokenizer;


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;


public class totalCount{

        public static class Map extends Mapper<Object, Text, Text, IntWritable>{

                private final static IntWritable one = new IntWritable(1);

                private Text word = new Text();


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

                        StringTokenizer st = new StringTokenizer(value.toString());

                        while(st.hasMoreTokens()){
```

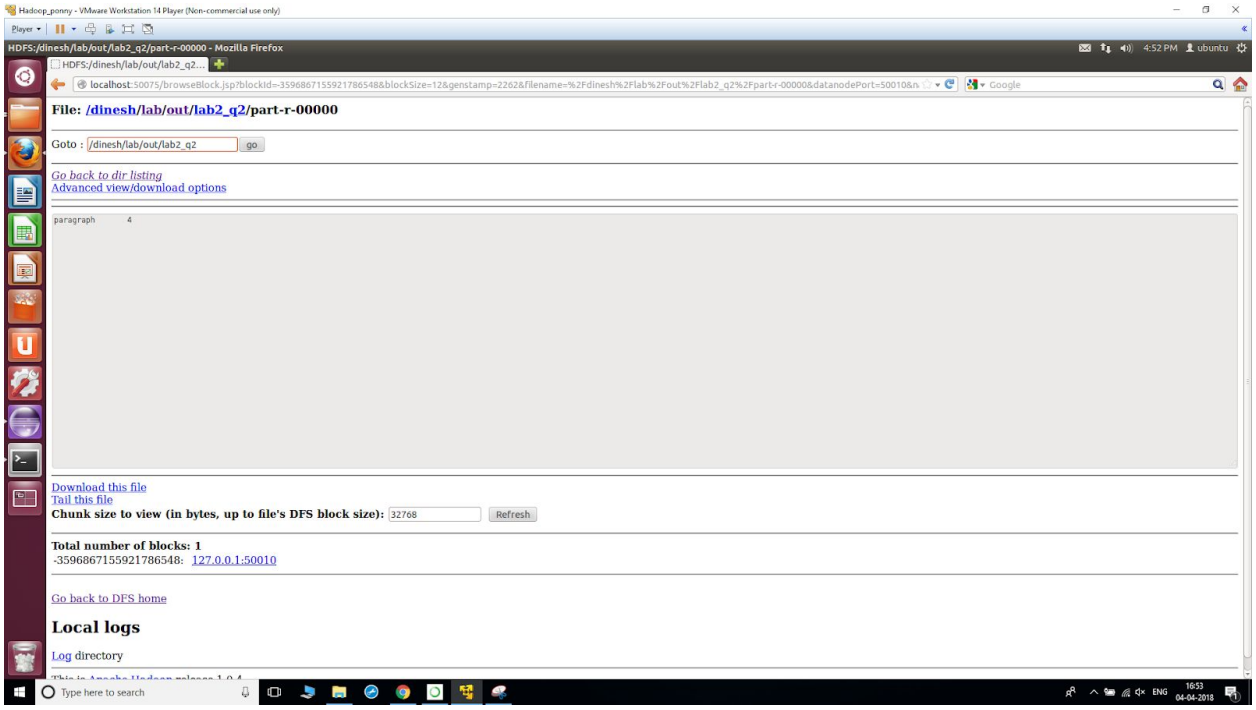
```
                    word.set(st.nextToken());
                    context.write(word,one);
            }

        }

}

public static class Reduce extends Reducer<Text, IntWritable, Text, IntWritable>{
        int maxSum = 0;
        Text totalKey = new Text();
        public void reduce(Text key, Iterable<IntWritable> values, Context context) throws IOException,InterruptedException{
            int sum = 0;
            for(IntWritable val: values){
                sum += val.get();
            }
            totalSum += sum;
        }
        @Override
        protected void cleanup(Context context) throws IOException, InterruptedException{
          context.write(totalKey, new IntWritable(totalSum));
        }
}
```

```
        public static void main(String[] args) throws Exception{
                Configuration conf = new Configuration();
                Job job = new Job(conf,"totalcount");
                job.setJarByClass(totalCount.class);
                job.setMapperClass(Map.class);
                job.setOutputKeyClass(Text.class);
                job.setOutputValueClass(IntWritable.class);
                job.setReducerClass(Reduce.class);
                FileInputFormat.addInputPath(job, new Path(args[0]));
                FileOutputFormat.setOutputPath(job, new Path(args[1]));
                job.waitForCompletion(true);
        }
}
```

Input:-same as lab2.



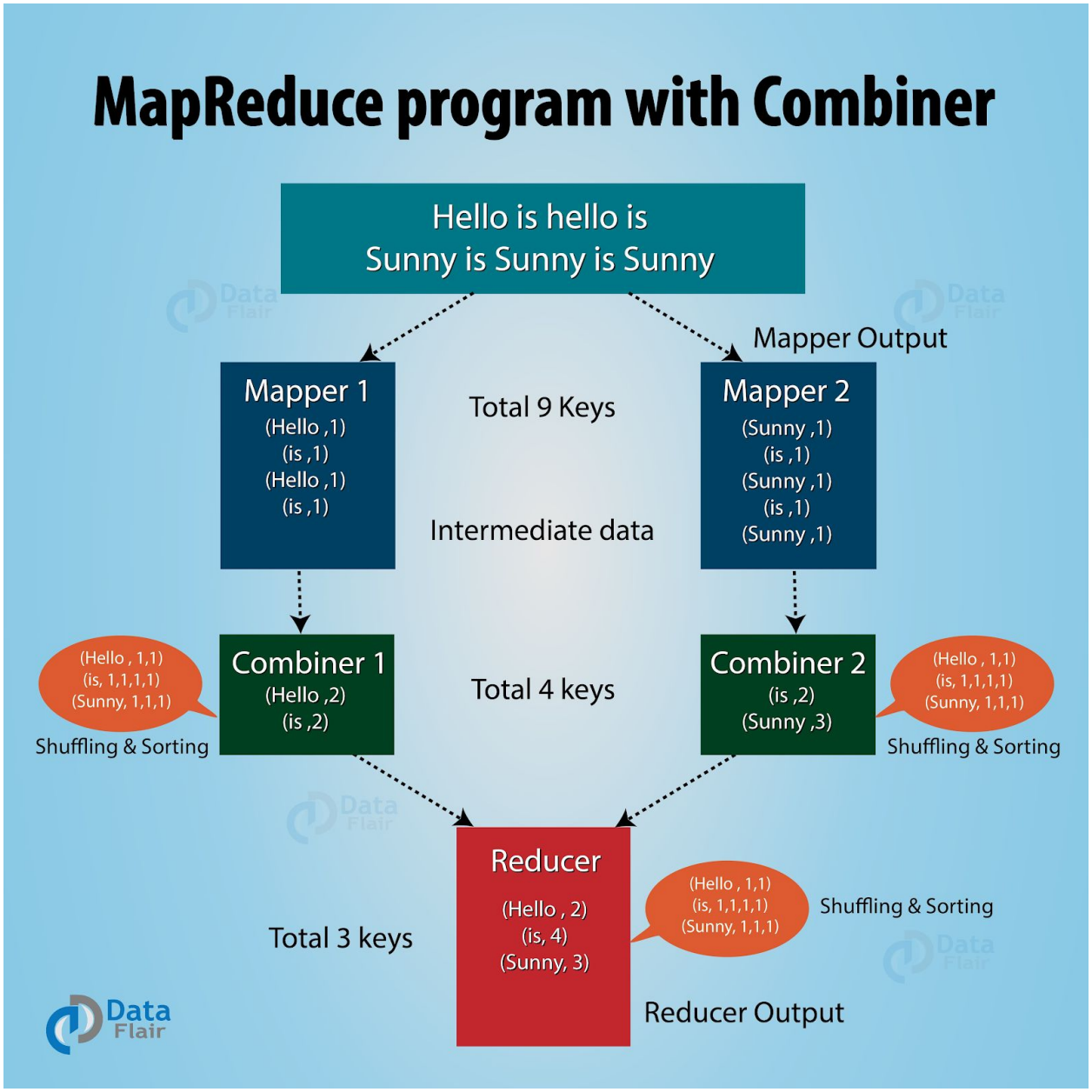
Output:-



lab4:-COMBINER

Algorithm:-

The combiner in MapReduce is also known as 'Mini-reducer'. The primary job of Combiner is to process the output data from the Mapper, before passing it to Reducer. It runs after the mapper and before the Reducer and its use is optional.



Code:-

**********lab4-combiner

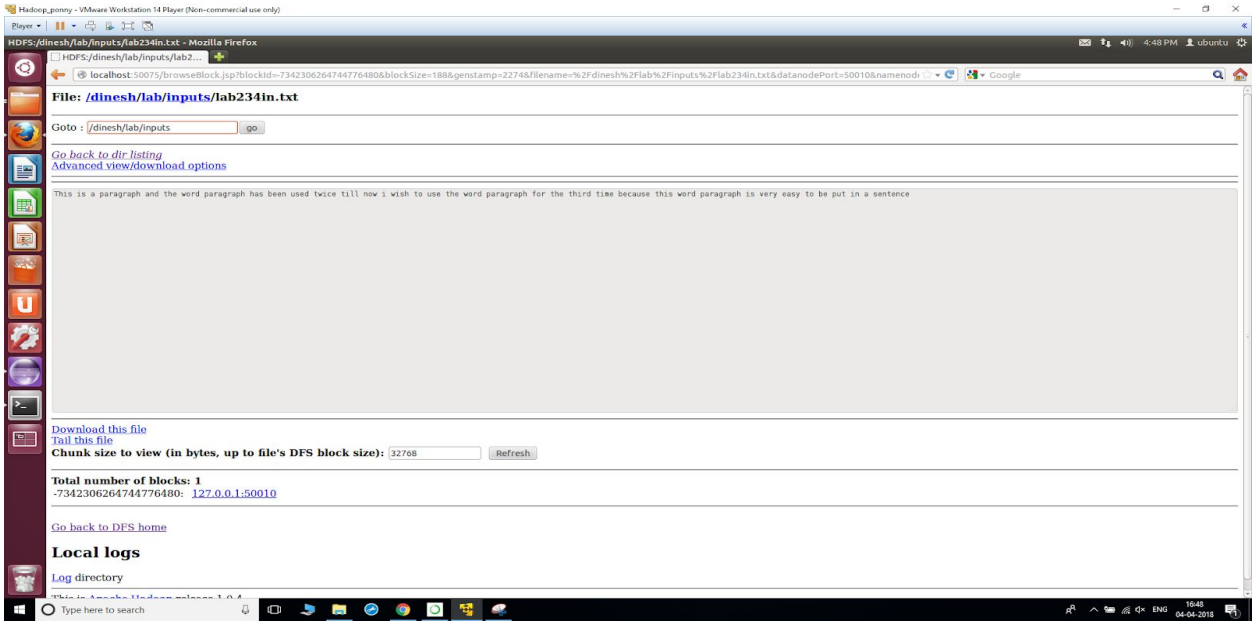
```
import java.io.IOException;

import java.util.StringTokenizer;


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;


public class WordCountCombiner{

	public static class Map extends Mapper<Object, Text, Text, IntWritable>{

		private final static IntWritable one = new IntWritable(1);

		private Text word = new Text();


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

			StringTokenizer st = new StringTokenizer(value.toString());

			while(st.hasMoreTokens()){

				word.set(st.nextToken());
```

```
                    context.write(word,one);
                }

            }

        }

        public static class Reduce 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));
            }
        }

        public static void main(String[] args) throws Exception{
            Configuration conf = new Configuration();
            Job job = new Job(conf,"wordcount");
            job.setJarByClass(WordCountCombiner.class);
            job.setMapperClass(Map.class);
            job.setOutputKeyClass(Text.class);
```

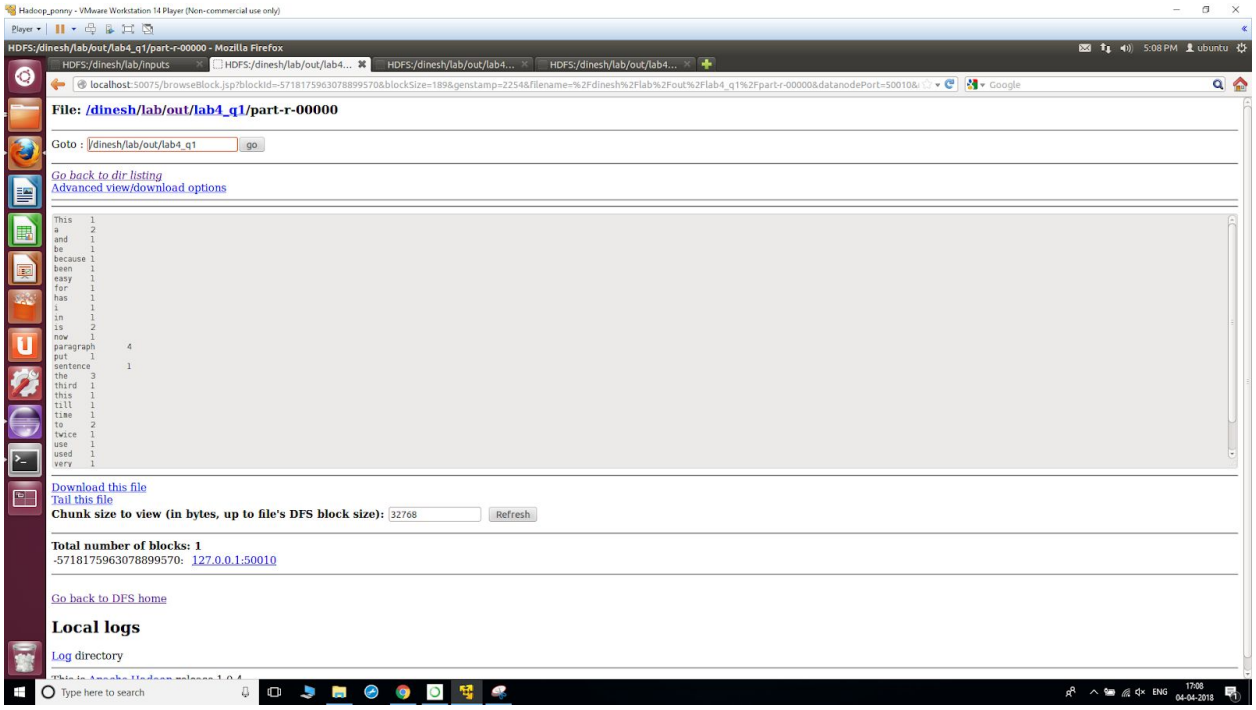
```
                job.setOuttyputValueClass(IntWritable.class);

                job.setReducerClass(Reduce.class);

                job.setCombinerClass(Reduce.class);

                //job.setInputFormatClass(TextInputFormat.class);

                //job.setOutputFormatClass(TextOutputFormat.class);

                FileInputFormat.addInputPath(job, new Path(args[0]));

                FileOutputFormat.setOutputPath(job, new Path(args[1]));

                job.waitForCompletion(true);

        }

}

/*

***

to execute:-

hadoop jar /home/ponny/*.jar WordCountCombiner.java /user/testWords.txt  /user/dini/outl4

*/
```

Input:-

Same as lab 2



Output:-



q2) Write output of only a certain range.

Check for salary greater than 1 lakh and write if name if salary greater than 1 lakh.

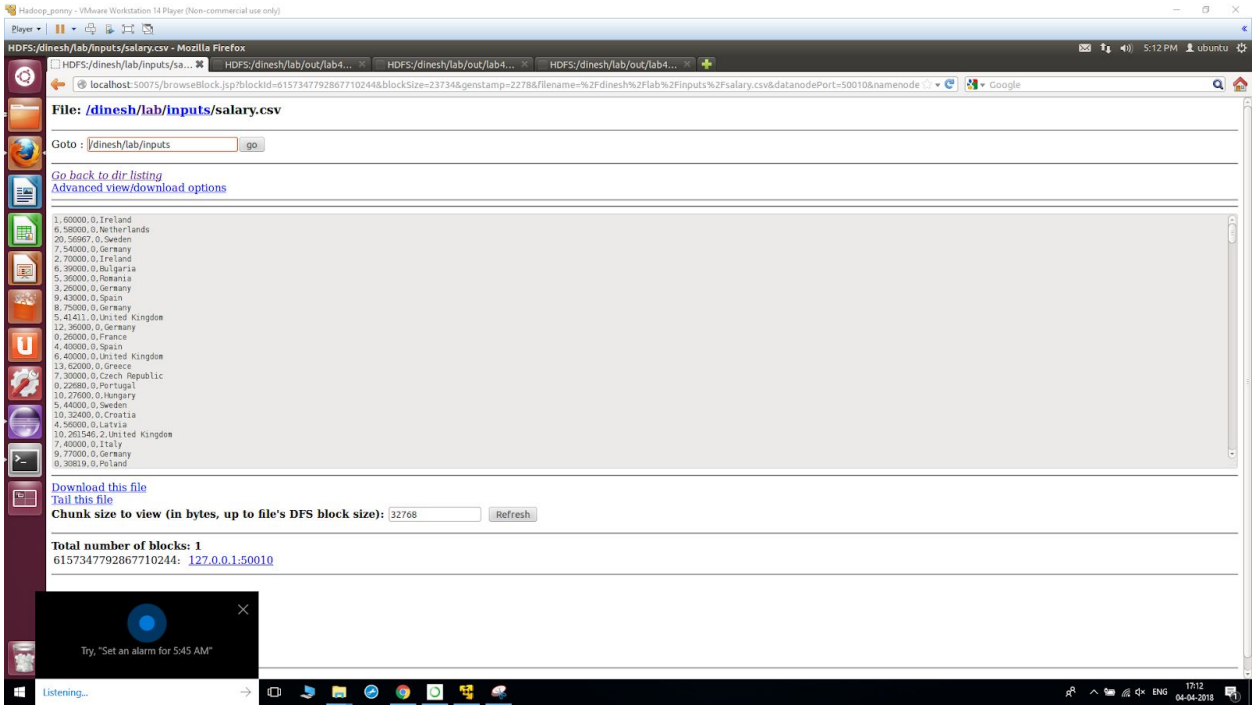
Code:-

```
//if salary is more than 1 lakh write the data to output file else ignore.

import java.io.IOException;

import org.apache.hadoop.conf.*;

import org.apache.hadoop.fs.Path;

import org.apache.hadoop.io.*;

import org.apache.hadoop.mapreduce.*;

import org.apache.hadoop.mapreduce.lib.output.*;

import org.apache.hadoop.mapreduce.lib.input.*;


public class Salary {


        public static void main(String[] args) throws IOException, ClassNotFoundException, InterruptedException {

                Configuration conf = new Configuration();

                Job job = new Job(conf, "wordcount");

                job.setJarByClass(Salary.class);

                job.setOutputKeyClass(Text.class);

                job.setOutputValueClass(IntWritable.class);

                job.setMapperClass(Map.class);

                //job.setReducerClass(Reduce.class);

                job.setInputFormatClass(TextInputFormat.class);

                job.setOutputFormatClass(TextOutputFormat.class);

                FileInputFormat.addInputPath(job, new Path(args[0]));
```

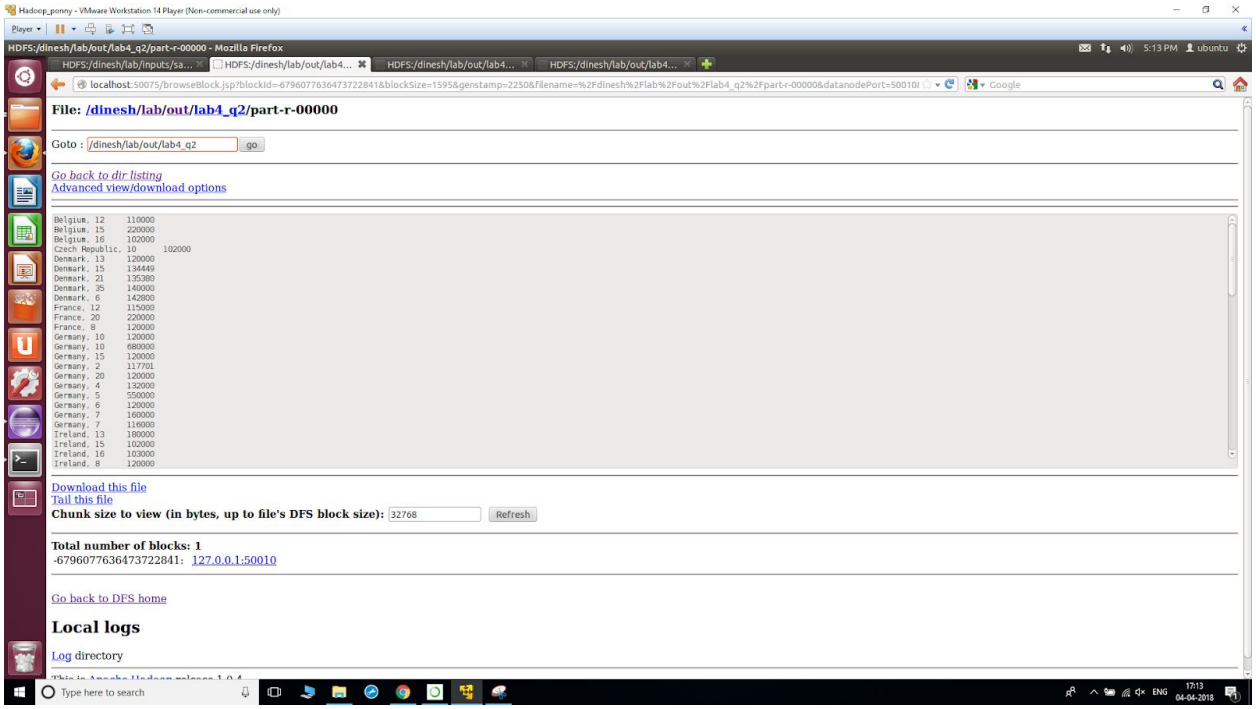
```
        FileOutputFormat.setOutputPath(job, new Path(args[1]));

        job.waitForCompletion(true);

    }

    public static class Map extends Mapper<LongWritable, Text, Text, IntWritable>{

        int count = 0;
        public Text word = new Text("Total Number of Employees: ");
        public void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException{

            String[] line = value.toString().split(",");
            int i = Integer.parseInt(line[1]);
            final IntWritable sal = new IntWritable(i);
            if(i > 100000) {
                context.write(new Text(line[3]+", "+line[0]),sal);
                //sal as value and rest two as paired key.
                count++;
            }

        }
```

```
        public void cleanup(Context context) throws IOException, InterruptedException {
            context.write(word, new IntWritable(count));
        }

    }

}
```

Input:-salary.csv


HDFS:/dinesh/lab/inputs/salary.csv - Mozilla Firefox
File: /dinesh/lab/inputs/salary.csv
Goto : /dinesh/lab/inputs
go
Go back to dir listing
Advanced view/download options
1,60000,0,Ireland
6,58000,0,Netherlands
20,56967,0,Sweden
7,54000,0,Germany
2,70000,0,Ireland
6,39000,0,Bulgaria
5,36000,0,Romania
3,26000,0,Germany
9,43000,0,Spain
8,75000,0,Germany
5,41411,0,United Kingdom
12,36000,0,Germany
0,26000,0,France
4,40000,0,Spain
6,40000,0,United Kingdom
13,62000,0,Greece
7,30000,0,Czech Republic
0,22680,0,Portugal
10,27600,0,Hungary
5,44000,0,Sweden
10,32400,0,Croatia
4,56000,0,Latvia
10,261546,2,United Kingdom
7,40000,0,Italy
9,77000,0,Germany
0,30819,0,Poland
Download this file
Tail this file
Chunk size to view (in bytes, up to file's DFS block size): 32768
Refresh
Total number of blocks: 1
6157347792867710244: 127.0.0.1:50010
Try, "Set an alarm for 5:45 AM"
Listening...

Output:-


HDFS:/dinesh/lab/out/lab4_q2/part-r-00000 - Mozilla Firefox
File: /dinesh/lab/out/lab4_q2/part-r-00000
Goto : /dinesh/lab/out/lab4_q2
go
Go back to dir listing
Advanced view/download options
Belgium, 12 110000
Belgium, 15 220000
Belgium, 16 102000
Czech Republic, 10 102000
Denmark, 13 120000
Denmark, 15 134449
Denmark, 21 135380
Denmark, 35 140000
Denmark, 6 142800
France, 12 115000
France, 20 220000
France, 8 120000
Germany, 10 120000
Germany, 10 680000
Germany, 15 120000
Germany, 2 117701
Germany, 20 120000
Germany, 4 132000
Germany, 5 550000
Germany, 6 120000
Germany, 7 160000
Germany, 7 116000
Ireland, 13 180000
Ireland, 15 102000
Ireland, 16 103000
Ireland, 8 120000
Download this file
Tail this file
Chunk size to view (in bytes, up to file's DFS block size): 32768
Refresh
Total number of blocks: 1
-6796077636473722841: 127.0.0.1:50010
Go back to DFS home
Local logs
Log directory
Type here to search

lab5:-custom partitioning

Output data in different files based on a criterion.

Ex:-

salary less than 30000 in 1 file

Salary less than 50000 in another

And rest in another file. Output.

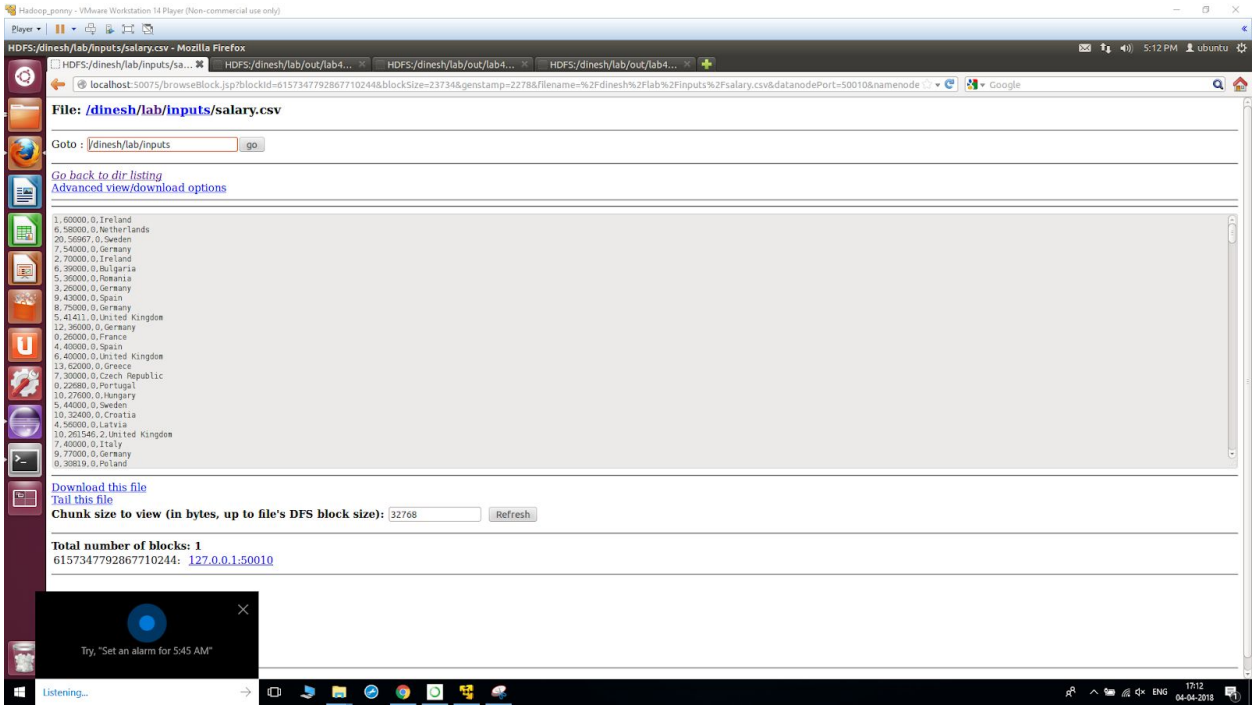
Code:-

```
import java.io.*;

import java.util.*;

import org.apache.hadoop.fs.Path;

import org.apache.hadoop.io.*;

import org.apache.hadoop.conf.*;

import org.apache.hadoop.mapreduce.*;

import org.apache.hadoop.mapreduce.Reducer.Context;

import org.apache.hadoop.mapreduce.lib.input.*;

import org.apache.hadoop.mapreduce.lib.output.*;

public class customP{

        public static class Map extends Mapper<LongWritable,Text,Text,IntWritable>{

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



                        String[] line=value.toString().split(",");

                        int i=Integer.parseInt(line[1]);

                        context.write(new Text(line[3]), new IntWritable(i));
```

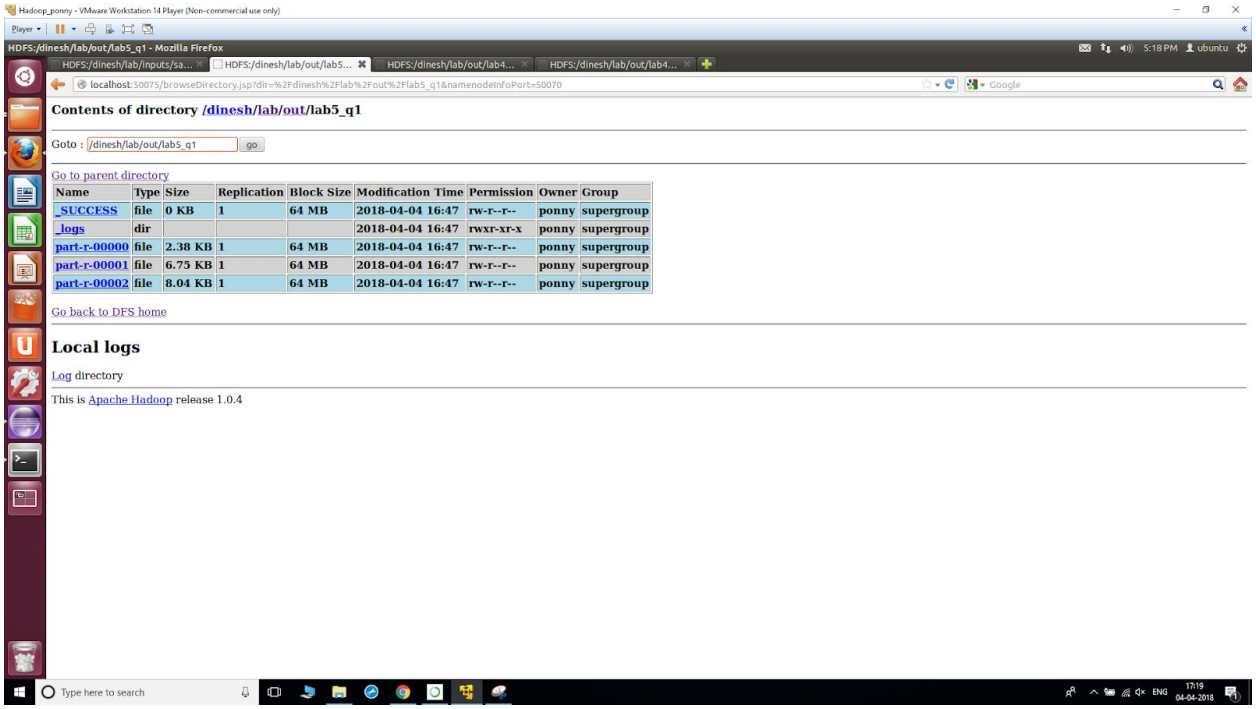
```
        }
    }
    public static class dpart extends Partitioner<Text,IntWritable>{
        public int getPartition(Text key,IntWritable value,int nr){
            if(value.get()<30000)
                return 0;
            if(value.get() < 50000)
                return 1;
            else
                return 2;
        }
    }
    public static class Reduce extends Reducer<Text,IntWritable,Text,IntWritable>{
        public void reduce(Text  key,IntWritable value,Context context)throws IOException,InterruptedException{


            context.write(key,value);
        }


    }
    public static void main(String[] args) throws Exception{
        Configuration conf=new Configuration();
        Job job=new Job(conf,"customP");
        job.setJarByClass(customP.class);
```

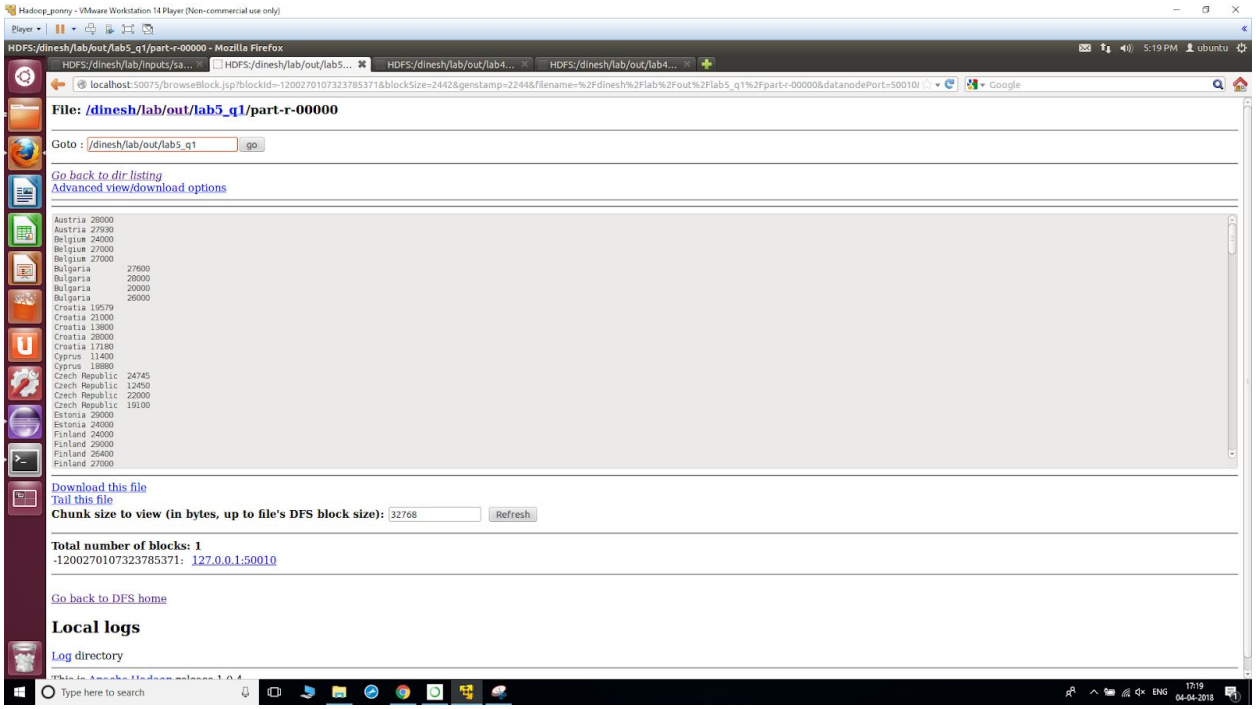
```
                job.setOutputKeyClass(Text.class);

                job.setOutputValueClass(IntWritable.class);

                job.setMapperClass(Map.class);

                job.setPartitionerClass(dpart.class);

                job.setNumReduceTasks(3);

                job.setInputFormatClass(TextInputFormat.class);

                job.setOutputFormatClass(TextOutputFormat.class);

                FileInputFormat.addInputPath(job,new Path(args[0]));

                FileOutputFormat.setOutputPath(job,new Path(args[1]));

                job.waitForCompletion(true);


        }


}

//results in 3 different output files.
```

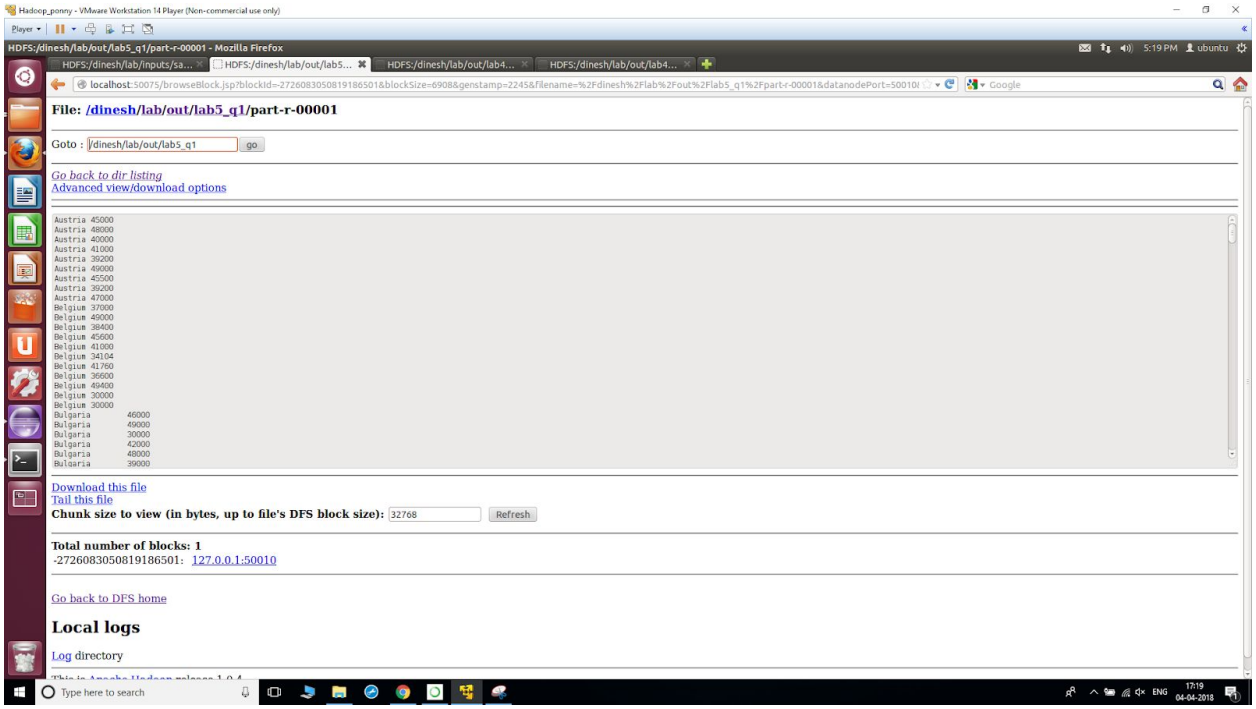
Input:-same salary.csv file.


HDFS:/dinesh/lab/inputs/salary.csv - Mozilla Firefox
File: /dinesh/lab/inputs/salary.csv
Goto : /dinesh/lab/inputs
go
Go back to dir listing
Advanced view/download options
1,60000,0,Ireland
6,58000,0,Netherlands
20,56967,0,Sweden
7,54000,0,Germany
2,70000,0,Ireland
6,39000,0,Bulgaria
5,36000,0,Romania
3,26000,0,Germany
9,43000,0,Spain
8,75000,0,Germany
5,41411,0,United Kingdom
12,36000,0,Germany
0,26000,0,France
4,40000,0,Spain
6,40000,0,United Kingdom
13,62000,0,Greece
7,30000,0,Czech Republic
0,22680,0,Portugal
10,27600,0,Hungary
5,44000,0,Sweden
10,32400,0,Croatia
4,56000,0,Latvia
10,261546,2,United Kingdom
7,40000,0,Italy
9,77000,0,Germany
0,30819,0,Poland
Download this file
Tail this file
Chunk size to view (in bytes, up to file's DFS block size): 32768
Refresh
Total number of blocks: 1
6157347792867710244: 127.0.0.1:50010
Try, "Set an alarm for 5:45 AM"
Listening...

Output:-


HDFS:/dinesh/lab/out/lab5_q1 - Mozilla Firefox
Contents of directory /dinesh/lab/out/lab5_q1
Goto : /dinesh/lab/out/lab5_q1
go
Go to parent directory
Name | Type | Size | Replication | Block Size | Modification Time | Permission | Owner | Group
_SUCCESS | file | 0 KB | 1 | 64 MB | 2018-04-04 16:47 | rw-r--r-- | ponny | supergroup
_logs | dir | | | | 2018-04-04 16:47 | rwxr-xr-x | ponny | supergroup
part-r-00000 | file | 2.38 KB | 1 | 64 MB | 2018-04-04 16:47 | rw-r--r-- | ponny | supergroup
part-r-00001 | file | 6.75 KB | 1 | 64 MB | 2018-04-04 16:47 | rw-r--r-- | ponny | supergroup
part-r-00002 | file | 8.04 KB | 1 | 64 MB | 2018-04-04 16:47 | rw-r--r-- | ponny | supergroup
Go back to DFS home
Local logs
Log directory
This is Apache Hadoop release 1.0.4


Hadoop_ponny - VMware Workstation 14 Player (Non-commercial use only)
HDFS:/dinesh/lab/out/lab5_q1/part-r-00000 - Mozilla Firefox
File: /dinesh/lab/out/lab5_q1/part-r-00000
Goto : /dinesh/lab/out/lab5_q1 go
Go back to dir listing
Advanced view/download options
Austria 28000
Austria 27930
Belgium 24000
Belgium 27000
Belgium 27000
Bulgaria 27600
Bulgaria 28000
Bulgaria 20000
Bulgaria 26000
Croatia 19579
Croatia 21000
Croatia 13800
Croatia 28000
Croatia 17180
Cyprus 11400
Cyprus 18880
Czech Republic 24745
Czech Republic 12450
Czech Republic 22000
Czech Republic 19100
Estonia 29000
Estonia 24000
Finland 24000
Finland 29000
Finland 26400
Finland 27000
Download this file
Tail this file
Chunk size to view (in bytes, up to file's DFS block size): 32768 Refresh
Total number of blocks: 1
-1200270107323785371: 127.0.0.1:50010
Go back to DFS home
Local logs
Log directory


Hadoop_ponny - VMware Workstation 14 Player (Non-commercial use only)
HDFS:/dinesh/lab/out/lab5_q1/part-r-00001 - Mozilla Firefox
File: /dinesh/lab/out/lab5_q1/part-r-00001
Goto : /dinesh/lab/out/lab5_q1 go
Go back to dir listing
Advanced view/download options
Austria 45000
Austria 48000
Austria 40000
Austria 41000
Austria 39200
Austria 49000
Austria 45500
Austria 39200
Austria 47000
Belgium 37000
Belgium 49000
Belgium 38400
Belgium 45600
Belgium 41000
Belgium 34104
Belgium 41760
Belgium 36600
Belgium 49400
Belgium 30000
Belgium 30000
Bulgaria 46000
Bulgaria 49000
Bulgaria 30000
Bulgaria 42000
Bulgaria 48000
Bulgaria 39000
Download this file
Tail this file
Chunk size to view (in bytes, up to file's DFS block size): 32768 Refresh
Total number of blocks: 1
-2726083050819186501: 127.0.0.1:50010
Go back to DFS home
Local logs
Log directory

q2) Specify the data that is to be assigned to each mapper by using splittble function.

Only the number of mappers will change, Output will be the same

Code:

results in 3 different output files.

**********splitable

```
import java.io.*;

import java.util.*;

import org.apache.hadoop.fs.Path;

import org.apache.hadoop.io.*;

import org.apache.hadoop.conf.*;

import org.apache.hadoop.mapreduce.*;

import org.apache.hadoop.mapreduce.Reducer.Context;

import org.apache.hadoop.mapreduce.lib.input.*;

import org.apache.hadoop.mapreduce.lib.output.*;

public class splitable{

	public static class Map extends Mapper<LongWritable,Text,Text,IntWritable>{

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

			String[] line=value.toString().split(",");

			int i=Integer.parseInt(line[1]);

			context.write(new Text(line[3]), new IntWritable(i));

		}

	}
```

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

                Configuration conf=new Configuration();

                //setting the split size before the job set as this is the configuration section.

                conf.set("mapred.max.split.size","10000");

                Job job=new Job(conf,"splitable");

                job.setJarByClass(splitable.class);

                job.setOutputKeyClass(Text.class);

                job.setOutputValueClass(IntWritable.class);

                job.setMapperClass(Map.class);

                job.setInputFormatClass(TextInputFormat.class);

                job.setOutputFormatClass(TextOutputFormat.class);

                FileInputFormat.addInputPath(job,new Path(args[0]));

                FileOutputFormat.setOutputPath(job,new Path(args[1]));

                job.waitForCompletion(true);


        }


}

//open job tracker or task tracker to see the way job is done job tracker:localhost:50030 and task tracker :50060
```

Code for no split, i.e. All data to be processed in a single mapper.

```
*********split false

import java.io.*;

import org.apache.hadoop.fs.Path;

import org.apache.hadoop.io.*;

import org.apache.hadoop.conf.*;

import org.apache.hadoop.mapreduce.*;

import org.apache.hadoop.mapreduce.lib.input.*;

import org.apache.hadoop.mapreduce.lib.output.*;

public class split{

        public static class Map extends Mapper<LongWritable,Text,Text,IntWritable>{

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



                        String[] line=value.toString().split(",");

                        int i=Integer.parseInt(line[1]);

                        context.write(new Text(line[3]), new IntWritable(i));

                }

        }

//all the input file will be done in a single map task

        public class splitfalse extends TextInputFormat{



                protected boolean isSplitable(JobContext context, Path filename) {

                  return false;
```

```
            }
        }
        public static void main(String[] args) throws Exception
        {
            Configuration conf=new Configuration();
            Job job=new Job(conf,"split");
            job.setInputFormatClass(splitfalse.class);
            job.setJarByClass(split.class);
            job.setOutputKeyClass(Text.class);
            job.setOutputValueClass(IntWritable.class);
            job.setMapperClass(Map.class);
            job.setInputFormatClass(TextInputFormat.class);
            job.setOutputFormatClass(TextOutputFormat.class);
            FileInputFormat.addInputPath(job,new Path(args[0]));
            FileOutputFormat.setOutputPath(job,new Path(args[1]));
            job.waitForCompletion(true);

        }

}
```

Lab6:-

Different input and output formats

1) **Key value input format**

Code:-

q1)working with keyval input pair

```
import java.io.IOException;

import org.apache.hadoop.conf.*;

import org.apache.hadoop.fs.Path;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.*;

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

import org.apache.hadoop.mapreduce.lib.output.*;

import org.apache.hadoop.mapreduce.lib.input.*;



public class keyval {

	public static void main(String[] args) throws IOException, ClassNotFoundException, InterruptedException {

		Configuration conf = new Configuration();

		//setting to keyval pair

		conf.set("mapreduce.input.keyvaluelinerecordreader.key.value.separator", ",");


		Job job = new Job(conf, "keyval");
```

```
        job.setJarByClass(keyval.class);
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(Text.class);
        job.setMapperClass(Map.class);

        //
        job.setInputFormatClass(KeyValueTextInputFormat.class);
        //
        job.setOutputFormatClass(TextOutputFormat.class);
        FileInputFormat.addInputPath(job, new Path(args[0]));
        FileOutputFormat.setOutputPath(job, new Path(args[1]));

        job.waitForCompletion(true);

}
//all the arguments are text.
public static class Map extends Mapper<Text, Text, Text, Text>{
            String c="ram";
                public void map(Text key, Text value, Context context) throws
                IOException, InterruptedException
                {
                String[] line=key.toString().split(",");
                if(c.equalsIgnoreCase(line[0]))
```

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

}
```

Input will be taken considered as pair of key and value.

2)Nline input format

No.of lines are set per each mapper. So based on number of lines of data input file, no of mappers are decided.

Code:-

```
import java.io.*;
import java.util.*;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.*;
import org.apache.hadoop.conf.*;
import org.apache.hadoop.mapreduce.*;
import org.apache.hadoop.mapreduce.Reducer.Context;
import org.apache.hadoop.mapreduce.lib.input.*;
import org.apache.hadoop.mapreduce.lib.output.*;
public class nline{
public static class Map extends Mapper<Object,Text,Text,Text>{
String c = "ram";
```

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

String line=key.toString();

if(c.equalsIgnoreCase(line))

context.write(key,value);

}

}

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

Configuration conf=new Configuration();

//set no of lines per each mapper.

conf.setInt(NLineInputFormat.LINES_PER_MAP, 3);


Job job = new Job(conf,"nline");

job.setJarByClass(nline.class);

job.setOutputKeyClass(Text.class);

job.setOutputValueClass(Text.class);

job.setMapperClass(Map.class);

job.setNumReduceTasks(0);//setting no of reducers to 0

job.setInputFormatClass(NLineInputFormat.class);

FileInputFormat.addInputPath(job,new Path(args[0]));

FileOutputFormat.setOutputPath(job,new Path(args[1]));

job.waitForCompletion(true);

}

}
```

3) Sequential file input and output.

A file that contains records or other elements that are stored in a chronological order based on account number or some other identifying data.

Code:-

```
*******output part

import java.io.*;

import java.util.*;

import org.apache.hadoop.fs.Path;

import org.apache.hadoop.io.*;

import org.apache.hadoop.conf.*;

import org.apache.hadoop.mapreduce.*;

import org.apache.hadoop.mapreduce.Reducer.Context;

import org.apache.hadoop.mapreduce.lib.input.*;

import org.apache.hadoop.mapreduce.lib.output.*;

public class output{

public static class Map extends Mapper<Object,Text,LongWritable,Text>{

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

context.write(key,value);

}

}

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

Configuration conf=new Configuration();
```

```
conf.setInt(NLineInputFormat.LINES_PER_MAP, 3);

Job job = new Job(conf,"output");

job.setJarByClass(output.class);

job.setOutputKeyClass(LongWritable.class);

job.setOutputValueClass(Text.class);

job.setMapperClass(Map.class);

job.setNumReduceTasks(0);

//setting output as seqential file

job.setOutputFormatClass(SequenceFileOutputFormat.class);

//

FileInputFormat.addInputPath(job,new Path(args[0]));

FileOutputFormat.setOutputPath(job,new Path(args[1]));

job.waitForCompletion(true);

}

}


******input part

import java.io.*;

import java.util.*;

import org.apache.hadoop.fs.Path;

import org.apache.hadoop.io.*;

import org.apache.hadoop.conf.*;

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

```
import org.apache.hadoop.mapreduce.Reducer.Context;

import org.apache.hadoop.mapreduce.lib.input.*;

import org.apache.hadoop.mapreduce.lib.output.*;

public class input{

public static class Map extends Mapper<LongWritable,Text,LongWritable,Text>{



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

context.write(key,value);

}

}

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

Configuration conf=new Configuration();

//conf.setInt(NLineInputFormat.LINES_PER_MAP, 3);

Job job = new Job(conf,"input");

job.setJarByClass(input.class);

job.setOutputKeyClass(LongWritable.class);

job.setOutputValueClass(Text.class);

job.setMapperClass(Map.class);

job.setNumReduceTasks(0);

//input as sequential data.

job.setInputFormatClass(SequenceFileInputFormat.class);

//

FileInputFormat.addInputPath(job,new Path(args[0]));
```

```
FileOutputFormat.setOutputPath(job,new Path(args[1]));

job.waitForCompletion(true);

}

}
```

Lab7 :- Finding the top k records using tree map.

Algorithm:- general method of sorting data and resulting the top k won't be a efficient method for big data as data is large and parallel computations are incorporated. So we use tree map which sorts the data as it gets entered.

Code:-

```
import java.io.IOException;

import java.util.TreeMap;



import org.apache.hadoop.conf.*;

import org.apache.hadoop.fs.Path;

import org.apache.hadoop.io.*;

import org.apache.hadoop.mapreduce.*;

import org.apache.hadoop.mapreduce.lib.output.*;

import org.apache.hadoop.mapreduce.lib.input.*;



public class topk {



        public static void main(String[] args) throws IOException, ClassNotFoundException, InterruptedException {

                Configuration conf = new Configuration();

                Job job = new Job(conf, "topk");
```

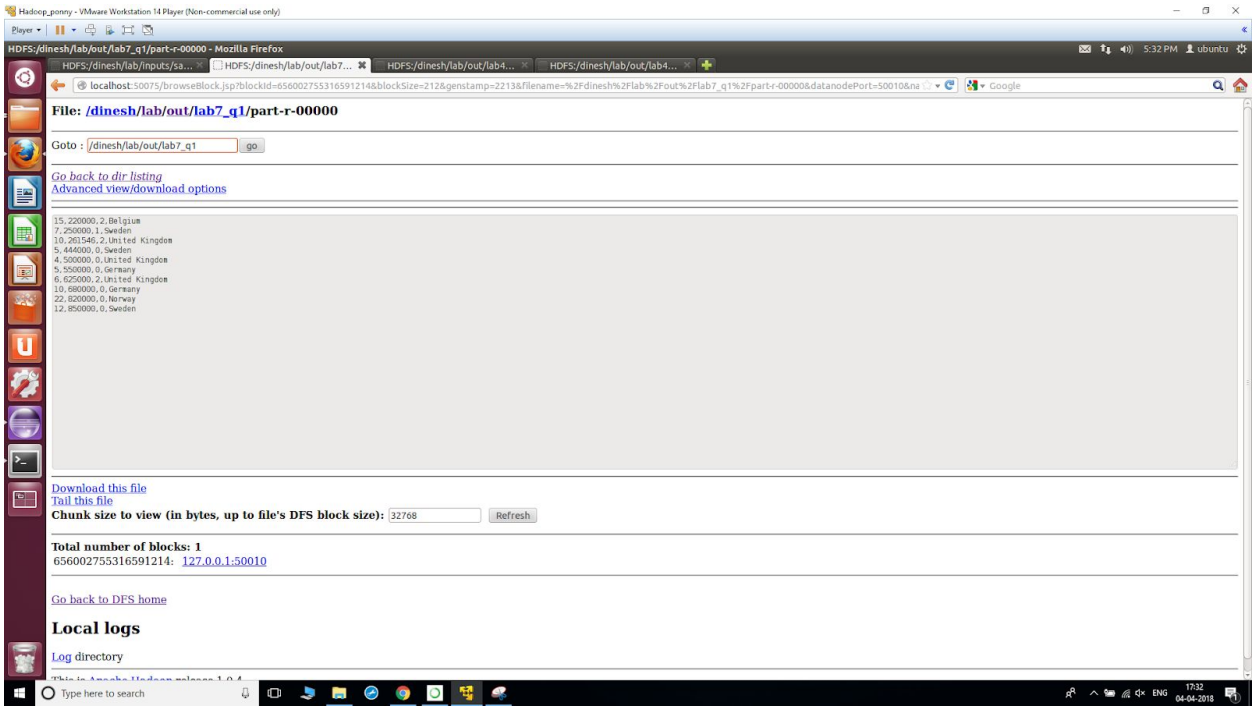
```
        job.setJarByClass(topk.class);
        //
        job.setOutputKeyClass(NullWritable.class);
        job.setOutputValueClass(Text.class);
        job.setNumReduceTasks(1);
        //
        /*job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(IntWritable.class);
        */
        job.setMapperClass(Map.class);
        job.setInputFormatClass(TextInputFormat.class);
        job.setOutputFormatClass(TextOutputFormat.class);
        FileInputFormat.addInputPath(job, new Path(args[0]));
        FileOutputFormat.setOutputPath(job, new Path(args[1]));

        job.waitForCompletion(true);

    }

    public static class Map extends Mapper<LongWritable,
    Text,NullWritable,Text>
    {
```

```
private TreeMap<Integer, Text> salary = new       TreeMap<Integer, Text>();

public void map(LongWritable key, Text value, Context

context) throws IOException, InterruptedException

{

String line = value.toString();

String[] elements=line.split(",");

int

i= Integer.parseInt(elements[1]);

salary.put(i,new Text(value));

//specifying the k value  as 10.

if (salary.size() > 10) {

salary.remove(salary.firstKey());

}

}

protected void cleanup(Context context) throws

IOException, InterruptedException

{

for ( Text name : salary.values() ) {

context.write(NullWritable.get(), name);

}

}

}

public static class Reduce extends
```

```
Reducer<NullWritable, Text, NullWritable, Text>

{

public void reduce(NullWritable key, Iterable<Text>

values, Context context) throws IOException,

InterruptedException {

TreeMap<Integer, Text> salary = new TreeMap< Integer,Text>();

for (Text value : values) {

String line = value.toString();

String[] elements=line.split(",");

int          i= Integer.parseInt(elements[1]);

salary.put(i, new Text(value));

if (salary.size() > 10) {

salary.remove(salary.firstKey());

}

}

for (Text t : salary.values()) {

context.write(NullWritable.get(), t);

}

}

}

}
```

Input:-salary.csv same as before.

output:-



q2) User defined counters

code:

```
import java.io.IOException;

import java.util.TreeMap;


import org.apache.hadoop.conf.*;

import org.apache.hadoop.fs.Path;

import org.apache.hadoop.io.*;

import org.apache.hadoop.mapreduce.*;

import org.apache.hadoop.mapreduce.lib.output.*;
```

```
import org.apache.hadoop.mapreduce.lib.input.*;


public class count {


        public static void main(String[] args) throws IOException, ClassNotFoundException, InterruptedException {

            Configuration conf = new Configuration();

            Job job = new Job(conf, "topk");

            job.setJarByClass(count.class);

            //


            job.setNumReduceTasks(0);

            //

            job.setOutputKeyClass(Text.class);

            job.setOutputValueClass(IntWritable.class);


            job.setMapperClass(Map.class);

            job.setInputFormatClass(TextInputFormat.class);

            job.setOutputFormatClass(TextOutputFormat.class);

            FileInputFormat.addInputPath(job, new Path(args[0]));

            FileOutputFormat.setOutputPath(job, new Path(args[1]));


            job.waitForCompletion(true);

            Counters cn=job.getCounters();
```

```
        cn.findCounter(ct.cnt).getValue();

        cn.findCounter(ct.nt).getValue();




}




        public enum ct

        {

        cnt,nt

        };

        public static class Map extends Mapper<LongWritable,

        Text, Text, IntWritable>

        {

        public void map(LongWritable key, Text value, Context

        context) throws IOException, InterruptedException {

        String line = value.toString();

        String[] elements=line.split(",");

        int i= Integer.parseInt(elements[1]);//i takes the salary values

        float exp= Float.parseFloat(elements[0]);

        Text tt=new Text(elements[3]);    //tt takes the country names.

        if(exp==0.0)
```

```
{
        //incrementing the counter
context.getCounter(ct.cnt).increment(1);


context.write(tt,new IntWritable(i));
}
if(i > 50000)
{
context.getCounter(ct.nt).increment(1);
}
}
}


}
```

Input:- salary.csv

output:-

```
Warning: $HADOOP_HOME is deprecated.

18/04/04 17:36:06 WARN mapred.JobClient: Use GenericOptionsParser for parsing the arguments. Applications should implement Tool for the same.
18/04/04 17:36:06 INFO input.FileInputFormat: Total input paths to process : 1
18/04/04 17:36:06 INFO util.NativeCodeLoader: Loaded the native-hadoop library
18/04/04 17:36:06 WARN snappy.LoadSnappy: Snappy native library not loaded
18/04/04 17:36:06 INFO mapred.JobClient: Running job: job_201804041630_0003
18/04/04 17:36:07 INFO mapred.JobClient:  map 0% reduce 0%
18/04/04 17:36:23 INFO mapred.JobClient:  map 100% reduce 0%
18/04/04 17:36:28 INFO mapred.JobClient: Job complete: job_201804041630_0003
18/04/04 17:36:28 INFO mapred.JobClient: Counters: 21
18/04/04 17:36:28 INFO mapred.JobClient:   Job Counters
18/04/04 17:36:28 INFO mapred.JobClient:     SLOTS_MILLIS_MAPS=14572
18/04/04 17:36:28 INFO mapred.JobClient:     Total time spent by all reduces waiting after reserving slots (ms)=0
18/04/04 17:36:28 INFO mapred.JobClient:     Total time spent by all maps waiting after reserving slots (ms)=0
18/04/04 17:36:28 INFO mapred.JobClient:     Launched map tasks=1
18/04/04 17:36:28 INFO mapred.JobClient:     Data-local map tasks=1
18/04/04 17:36:28 INFO mapred.JobClient:     SLOTS_MILLIS_REDUCES=0
18/04/04 17:36:28 INFO mapred.JobClient:   lab7_q2$ct
18/04/04 17:36:28 INFO mapred.JobClient:     nt=476
18/04/04 17:36:28 INFO mapred.JobClient:     cnt=26
18/04/04 17:36:28 INFO mapred.JobClient:   File Output Format Counters
18/04/04 17:36:28 INFO mapred.JobClient:     Bytes Written=411
18/04/04 17:36:28 INFO mapred.JobClient:   FileSystemCounters
18/04/04 17:36:28 INFO mapred.JobClient:     HDFS_BYTES_READ=23851
18/04/04 17:36:28 INFO mapred.JobClient:     FILE_BYTES_WRITTEN=21100
18/04/04 17:36:28 INFO mapred.JobClient:     HDFS_BYTES_WRITTEN=411
18/04/04 17:36:28 INFO mapred.JobClient:   File Input Format Counters
18/04/04 17:36:28 INFO mapred.JobClient:     Bytes Read=23734
18/04/04 17:36:28 INFO mapred.JobClient:   Map-Reduce Framework
18/04/04 17:36:28 INFO mapred.JobClient:     Map input records=1147
18/04/04 17:36:28 INFO mapred.JobClient:     Physical memory (bytes) snapshot=41254912
18/04/04 17:36:28 INFO mapred.JobClient:     Spilled Records=0
18/04/04 17:36:28 INFO mapred.JobClient:     CPU time spent (ms)=90
18/04/04 17:36:28 INFO mapred.JobClient:     Total committed heap usage (bytes)=16252928
18/04/04 17:36:28 INFO mapred.JobClient:     Virtual memory (bytes) snapshot=382939136
18/04/04 17:36:28 INFO mapred.JobClient:     Map output records=26
18/04/04 17:36:28 INFO mapred.JobClient:     SPLIT_RAW_BYTES=117
```

Lab 8 :-

q1) Side data

Take data from console and use it in Mapreduce job.

Code:-

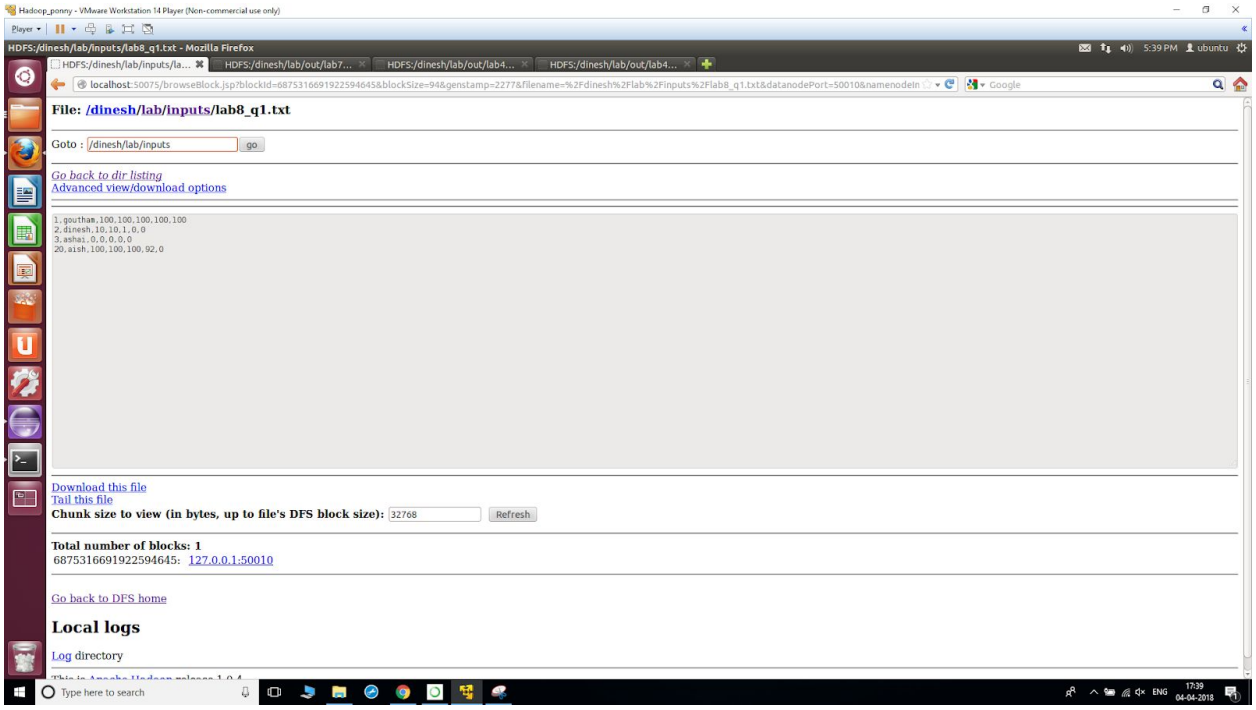
q1) // side data. taaking data from terminal as final argument.

```
import java.io.*;

import org.apache.hadoop.fs.Path;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.conf.*;

import org.apache.hadoop.mapreduce.*;

import org.apache.hadoop.mapreduce.lib.input.*;

import org.apache.hadoop.mapreduce.lib.output.*;
```

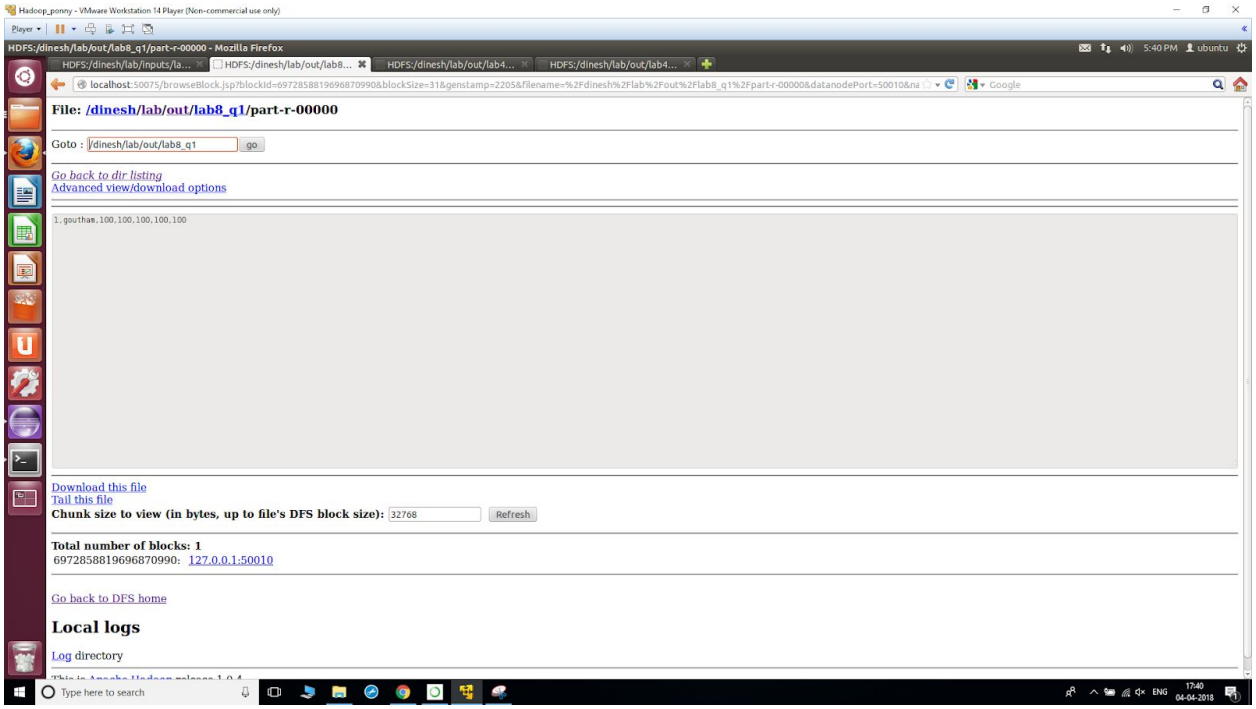
```
public class student{
        public static class Map extends Mapper‹Text,Text,Text,Text›
        {
                public void map(Text key, Text value, Context context) throws IOException, InterruptedException
                {
                        String name = context.getConfiguration().get("name");
                        String[] elements = key.toString().split(",");
                        if(name.equalsIgnoreCase(elements[1]))
                                context.write(key,value);
                }

        }
        public static void main(String[] args) throws Exception
        {
                Configuration conf=new Configuration();
                conf.set("name",args[2]);
                Job job=new Job(conf,"student");
                job.setInputFormatClass(KeyValueTextInputFormat.class);
                job.setJarByClass(student.class);
                job.setOutputKeyClass(Text.class);
                job.setOutputValueClass(Text.class);
                job.setMapperClass(Map.class);
                job.setOutputFormatClass(TextOutputFormat.class);
```

```
            FileInputFormat.addInputPath(job,new Path(args[0]));

            FileOutputFormat.setOutputPath(job,new Path(args[1]));

            job.waitForCompletion(true);

        }

}
```

Input:-



Output:-



q2) Reduce side join.

code:-q2) //basic reduce side join

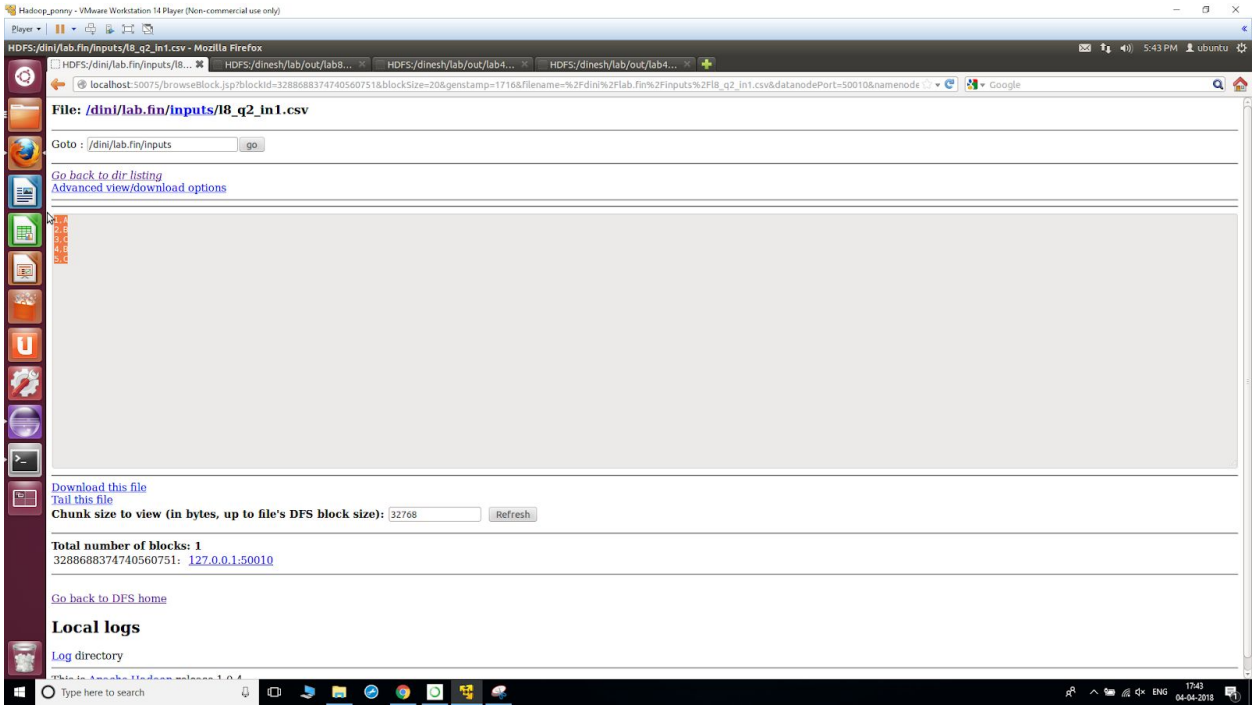
```
import java.io.IOException;

import org.apache.hadoop.conf.Configuration;

import org.apache.hadoop.fs.Path;

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.MultipleInputs;
```

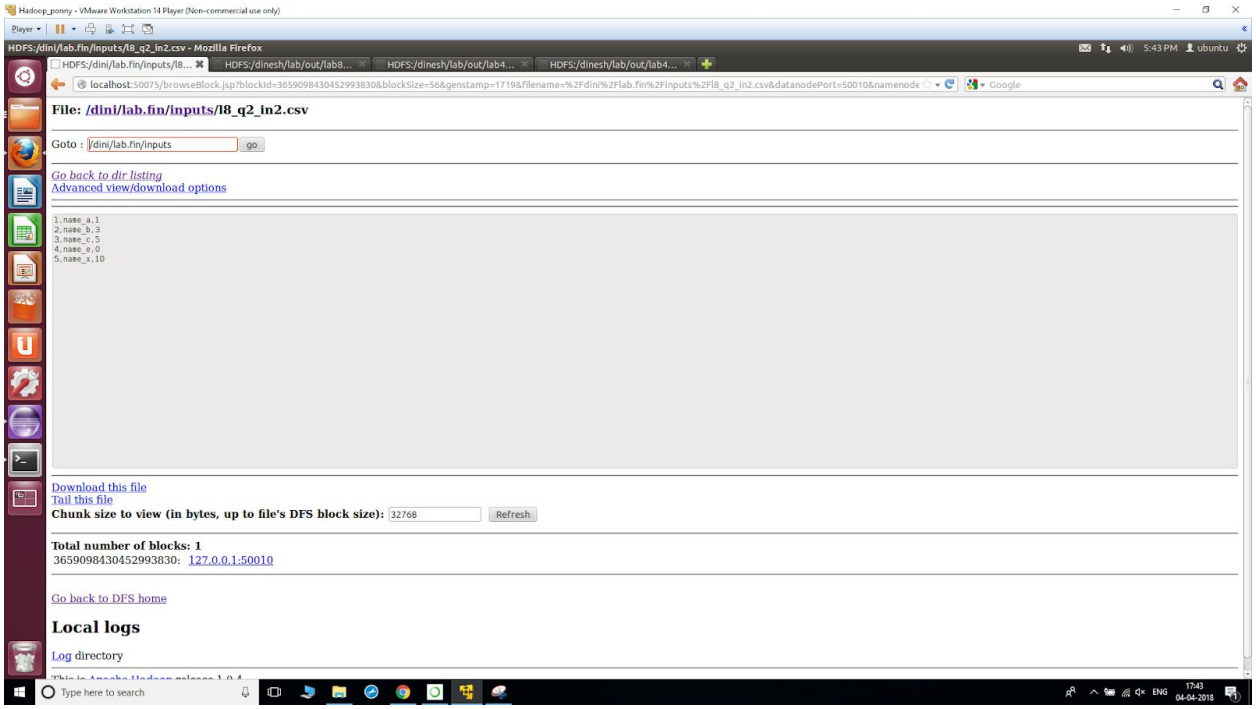
```
import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;

import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;


public class multi {

	public static class Map1 extends Mapper<LongWritable, Text, Text, Text>{

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

		{

			String[] line=value.toString().split(",");

			context.write(new Text(line[0]),new Text(line[1] + " " + line[2]));

		}

	}

	public static class Map2 extends Mapper<LongWritable, Text, Text, Text>{

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

		{

			String[] line=value.toString().split(",");

			context.write(new Text(line[0]),new Text(line[1]));

		}

	}

	public static class Reduce extends Reducer<Text, Text, Text, Text>{

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

```
                String line = null;
                for(Text val:values)
                {
                        line = val.toString();
                }
                context.write(key, new Text(line));
        }
    }
    public static void main(String[] args) throws IOException, ClassNotFoundException,
InterruptedException {
        Configuration conf = new Configuration();
        Job job = new Job(conf, "multi");
        job.setJarByClass(multi.class);
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(Text.class);
        job.setMapperClass(Map1.class);
        job.setMapperClass(Map2.class);
        job.setInputFormatClass(TextInputFormat.class);
        job.setOutputFormatClass(TextOutputFormat.class);
        MultipleInputs.addInputPath(job,new Path(args[0]), TextInputFormat.class,
Map1.class);
        MultipleInputs.addInputPath(job,new Path(args[1]), TextInputFormat.class,
Map2.class);
        FileOutputFormat.setOutputPath(job, new Path(args[2]));
```

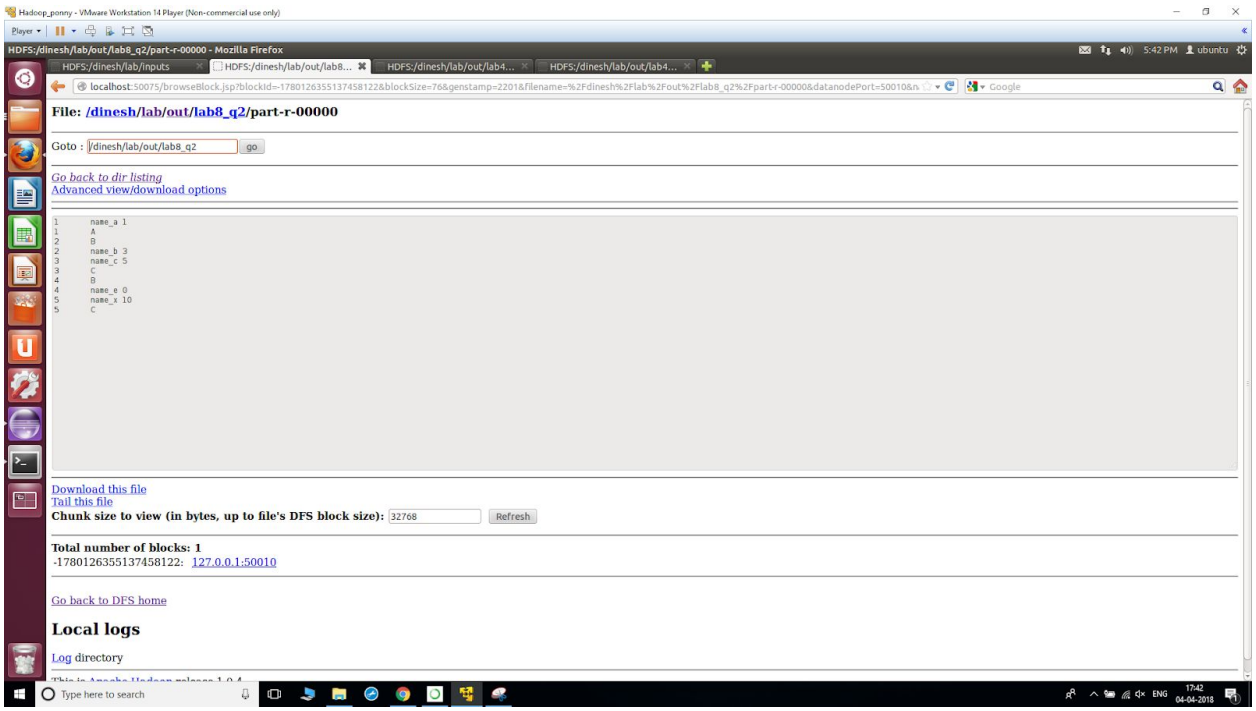
```
                job.waitForCompletion(true);

        }

}
```

Input:- 2 files





Output:- writing the data after joining 2 files.



l9)

q1) distributed cache.

One of the 2 files is cached for faster access. The file that is more frequently used is cached.

Code:-

```
import java.io.*;

import java.util.*;

import org.apache.hadoop.conf.Configuration;

import org.apache.hadoop.fs.Path;

import org.apache.hadoop.io.*;

import org.apache.hadoop.mapreduce.*;

import org.apache.hadoop.filecache.*;

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;



public class emp{

        public static class Map extends Mapper<LongWritable,Text,Text,Text> {

                Path[] cfile=new Path[0];

                ArrayList<Text> empl=new ArrayList<Text>();

                public void setup(Context context)

                {
```

```
        Configuration conf=context.getConfiguration();
        try
        {
            cfile = DistributedCache.getLocalCacheFiles(conf);
            BufferedReader reader=new BufferedReader(new FileReader(cfile[0].toString()));
            String line;
            while ((line=reader.readLine())!=null)
            {
                Text tt=new Text(line);
                empl.add(tt);
            }
        }
        catch(IOException e)
        {
            e.printStackTrace();
        }
    }
    public void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException {
        String[] elements = value.toString().split(",");
        for(Text e:empl)
        {
            String[] line1 = e.toString().split(",");
            if(elements[0].equals(line1[0]))
```

```
                    {
                        context.write(new Text(elements[0]),new Text(elements[1]+",",+elements[2]+",",+line1[1]));
                    }
                }
            }
        }

        public static void main(String[] args) throws Exception{
            Configuration conf = new Configuration();
            Job job = new Job(conf,"emp");
            job.setJarByClass(emp.class);
            DistributedCache.addCacheFile(new Path(args[0]).toUri(),job.getConfiguration());
            job.setMapperClass(Map.class);
            job.setOutputKeyClass(Text.class);
            job.setOutputValueClass(Text.class);
            //job.setInputFormatClass(TextInputFormat.class);
            //job.setOutputFormatClass(TextOutputFormat.class);
            FileInputFormat.addInputPath(job, new Path(args[1]));
            FileOutputFormat.setOutputPath(job, new Path(args[2]));
            job.waitForCompletion(true);
        }
}
```

Input and output will be same as normal join but the execution will be faster.

q2) reduce side join customer and transaction details.

Find total amount of transaction of a customer.

code:-

```
import java.io.IOException;

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.Mapper;

import org.apache.hadoop.mapreduce.Reducer;

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 ReduceJoin {

public static class CustsMapper extends Mapper <Object, Text, Text, Text>

{

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

throws IOException, InterruptedException

{

String record = value.toString();

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

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

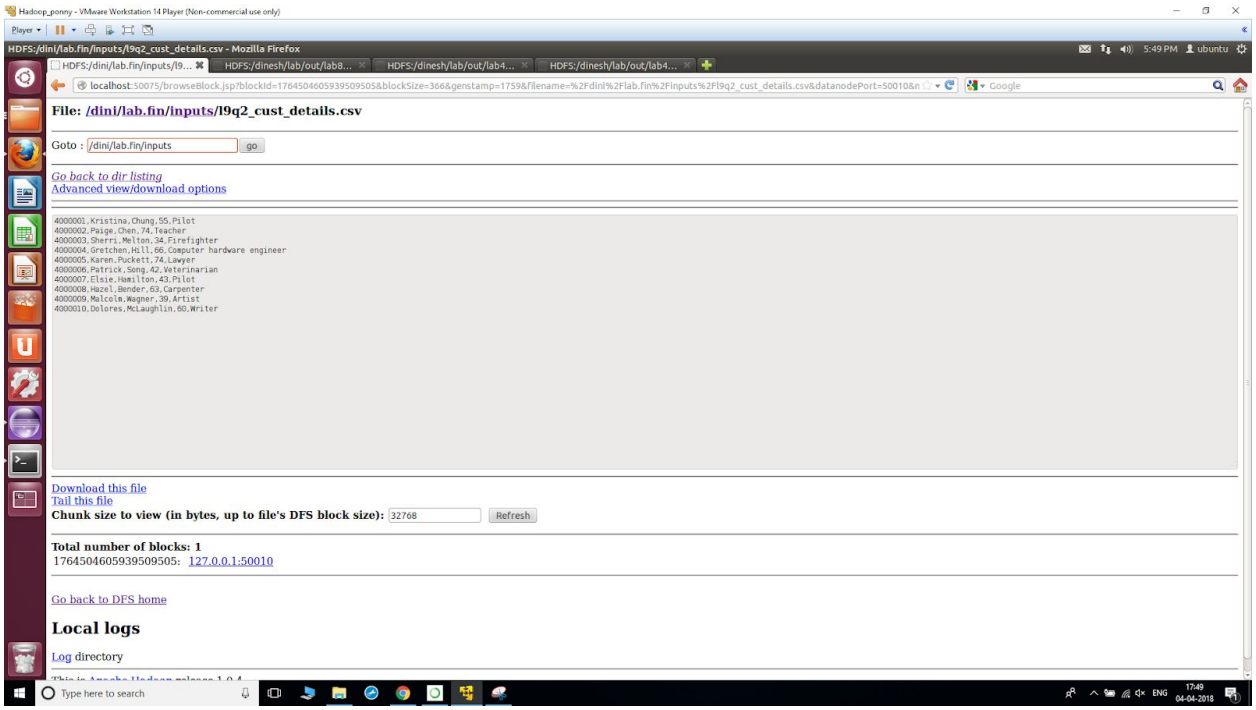
```
}
}

public static class TxnsMapper extends Mapper ‹Object, Text, Text, Text›
{
public void map(Object key, Text value, Context context)
throws IOException, InterruptedException
{
String record = value.toString();
String[] parts = record.split(",");
context.write(new Text(parts[2]), new Text("tnxn\t" + parts[3]));
}
}

public static class ReduceJoinReducer extends Reducer ‹Text, Text, Text, Text›
{
public void reduce(Text key, Iterable‹Text› values, Context context)
throws IOException, InterruptedException
{
String name = "";
double total = 0.0;
int count = 0;
for (Text t : values)
```

```
{
String parts[] = t.toString().split("\t");
if (parts[0].equals("tnxn"))
{
count++;
total += Float.parseFloat(parts[1]);
}
else if (parts[0].equals("cust"))
{
name = parts[1];
}
}
String str = String.format("%d\t%f", count, total);
context.write(new Text(name), new Text(str));
}
}

public static void main(String[] args) throws Exception {
Configuration conf = new Configuration();
Job job = new Job(conf, "Reduce-side join");
job.setJarByClass(ReduceJoin.class);
job.setReducerClass(ReduceJoinReducer.class);
job.setOutputKeyClass(Text.class);
```

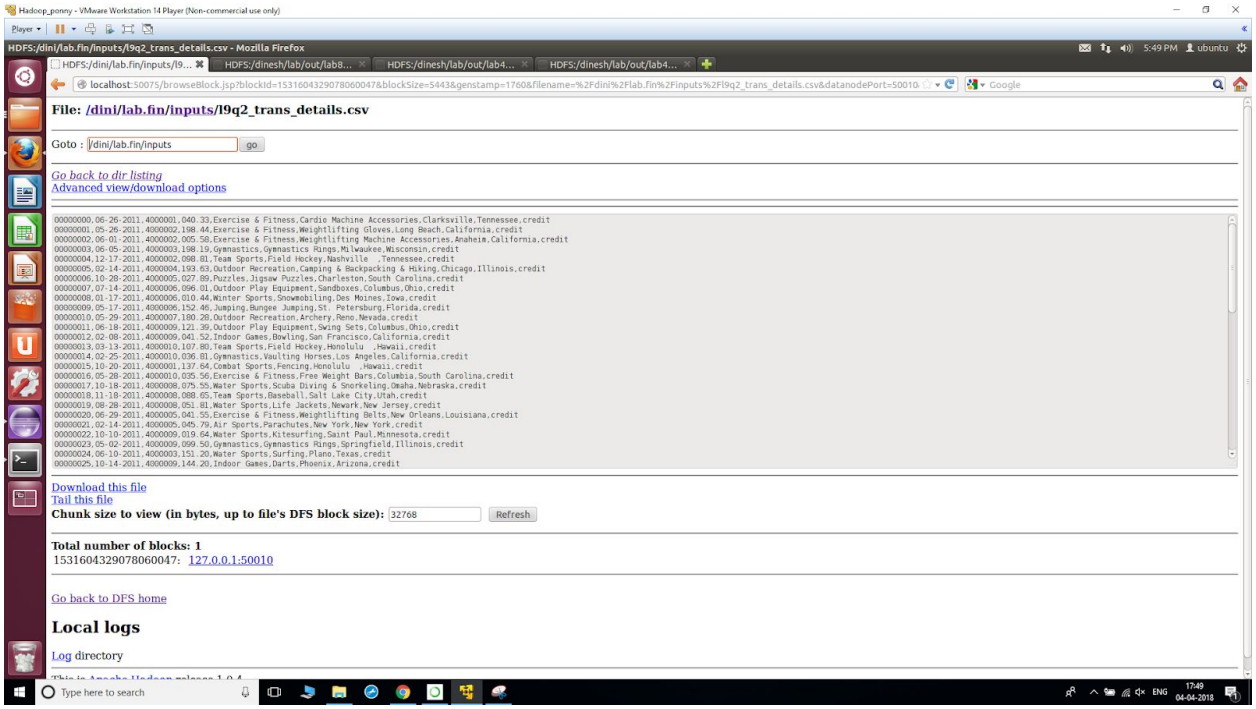
```
job.setOutputValueClass(Text.class);


MultipleInputs.addInputPath(job, new Path(args[0]),TextInputFormat.class, CustsMapper.class);

MultipleInputs.addInputPath(job, new Path(args[1]),TextInputFormat.class, TxnsMapper.class);

Path outputPath = new Path(args[2]);


FileOutputFormat.setOutputPath(job, outputPath);

outputPath.getFileSystem(conf).delete(outputPath);

System.exit(job.waitForCompletion(true) ? 0 : 1);

}

}
```

Input:-

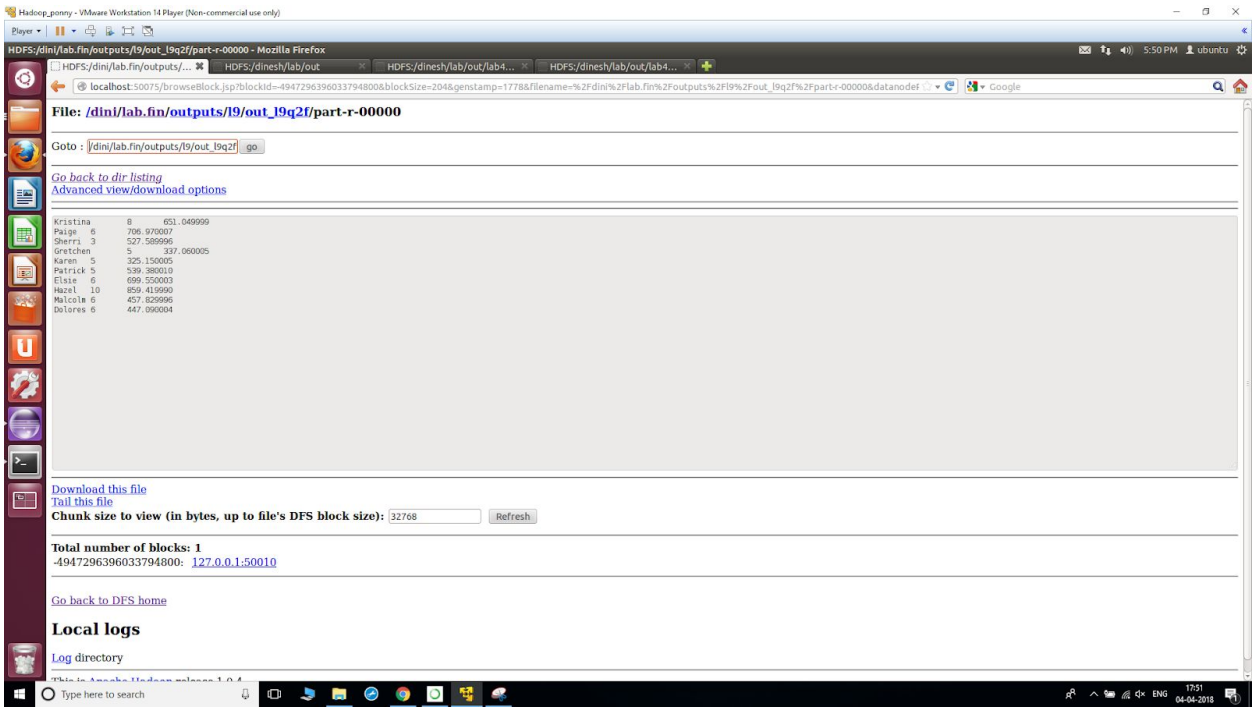
Cust_details



transaction_details



Output:-



Lab10:- Secondary sorting

Hadoop automatically sorts by key. If we want to sort by value also, or a part of value , that process is called secondary sorting. With a slight manipulation to the format of the key object, secondary sorting gives us the ability to take the value into account during the sort phase by creating a composite key by adding a part of, or the entire value to the natural key to achieve your sorting objectives.

Input contains country name and state name our aim to sort by country and inside the country sort by state.

Code:-

```
//secondary sorting


import java.io.BufferedReader;

import java.io.DataInput;

import java.io.DataOutput;

import java.io.FileReader;

import java.io.IOException;

import java.nio.file.FileSystem;

import java.util.*;

import org.apache.hadoop.filecache.DistributedCache;

import org.apache.hadoop.fs.Path;

import org.apache.hadoop.conf.*;

import org.apache.hadoop.io.*;

import org.apache.hadoop.mapreduce.*;

import org.apache.hadoop.mapreduce.Reducer.Context;

import org.apache.hadoop.mapreduce.lib.output.*;
```

```
import org.apache.hadoop.mapreduce.lib.input.*;

public class lab10 {

    public static class CompositeKeyWritable implements Writable, WritableComparable<CompositeKeyWritable> {

        private String deptNo;

        private String emp;

        public CompositeKeyWritable() {

        }

        public CompositeKeyWritable(String deptNo, String emp) {

            this.deptNo = deptNo;

            this.emp = emp;

        }

        public String toString() {

            return (new StringBuilder().append(deptNo).append("\t").append(emp)).toString();

        }

        public void readFields(DataInput dataInput) throws IOException {

            deptNo = WritableUtils.readString(dataInput);

            emp = WritableUtils.readString(dataInput);



        }

        public void write(DataOutput dataOutput) throws IOException {

            WritableUtils.writeString(dataOutput,deptNo);

            WritableUtils.writeString(dataOutput,emp);
```

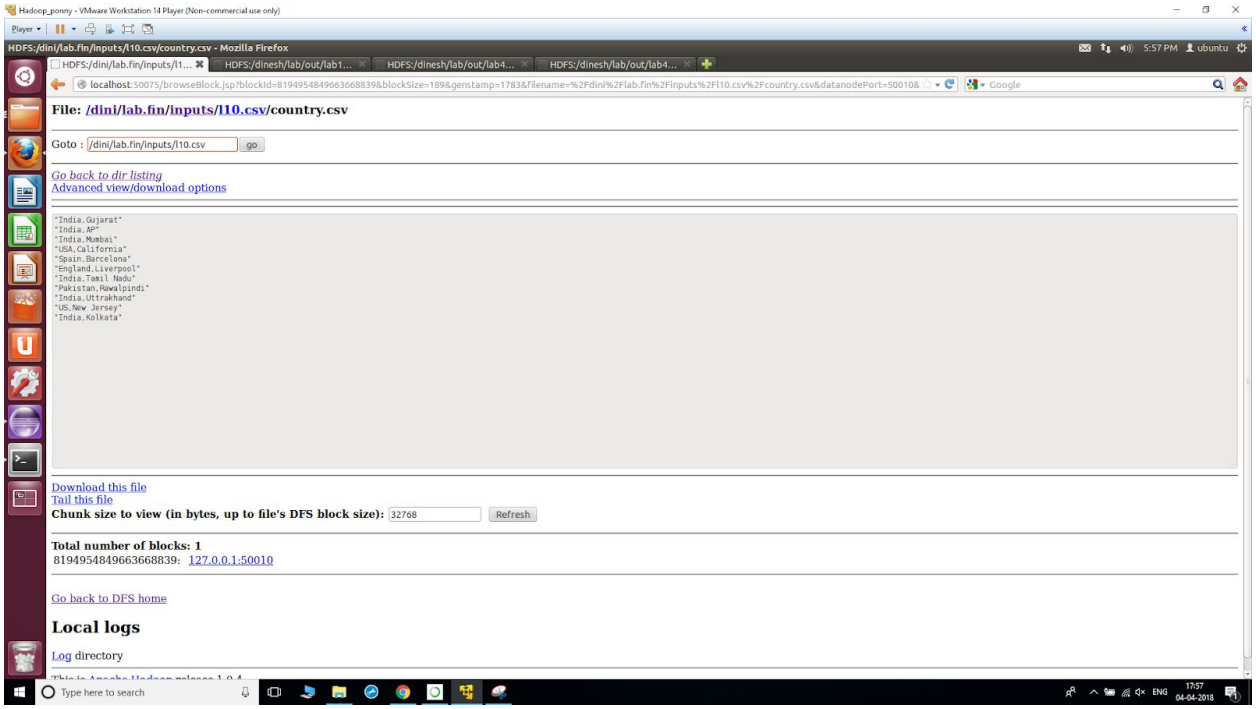
```
    }
    public int compareTo(CompositeKeyWritable objKeyPair) {
        int result = deptNo.compareTo(objKeyPair.deptNo);
        if (0 == result) {
            result = emp.compareTo(objKeyPair.emp);
        }
        return result;
    }
}
public static class mapper1 extends Mapper<LongWritable, Text,
CompositeKeyWritable,NullWritable> {
    public void map(LongWritable key, Text value, Context context) throws
    IOException, InterruptedException {
        if (value.toString().length() > 0) {
            String arrEmpAttributes[] = value.toString().split(",");
            context.write(new
                CompositeKeyWritable(arrEmpAttributes[0].toString(),(arrEmpAttributes[1].toString())),NullWritable.get());
        }
    }
}
public static class SecondarySortBasicPartitioner extends Partitioner<CompositeKeyWritable, NullWritable> {
    public int getPartition(CompositeKeyWritable key,NullWritable
        value,int numReduceTasks) {
```

```
        return (key.deptNo.hashCode() % numReduceTasks);
    }
}
public static class SecondarySortBasicCompKeySortComparator extends WritableComparator {
    protected SecondarySortBasicCompKeySortComparator()
    {
        super(CompositeKeyWritable.class, true);
    }
    public int compare(WritableComparable w1,WritableComparable w2) {
        CompositeKeyWritable key1 = (CompositeKeyWritable) w1;
        CompositeKeyWritable key2 = (CompositeKeyWritable) w2;
        int cmpResult = key1.deptNo.compareTo(key2.deptNo);
        if (cmpResult == 0)
        {
            return -key1.emp.compareTo(key2.emp);
        }
        return cmpResult;
    }
}
public static class SecondarySortBasicGroupingComparator extends WritableComparator {
    protected SecondarySortBasicGroupingComparator() {
        super(CompositeKeyWritable.class, true);
    }
```

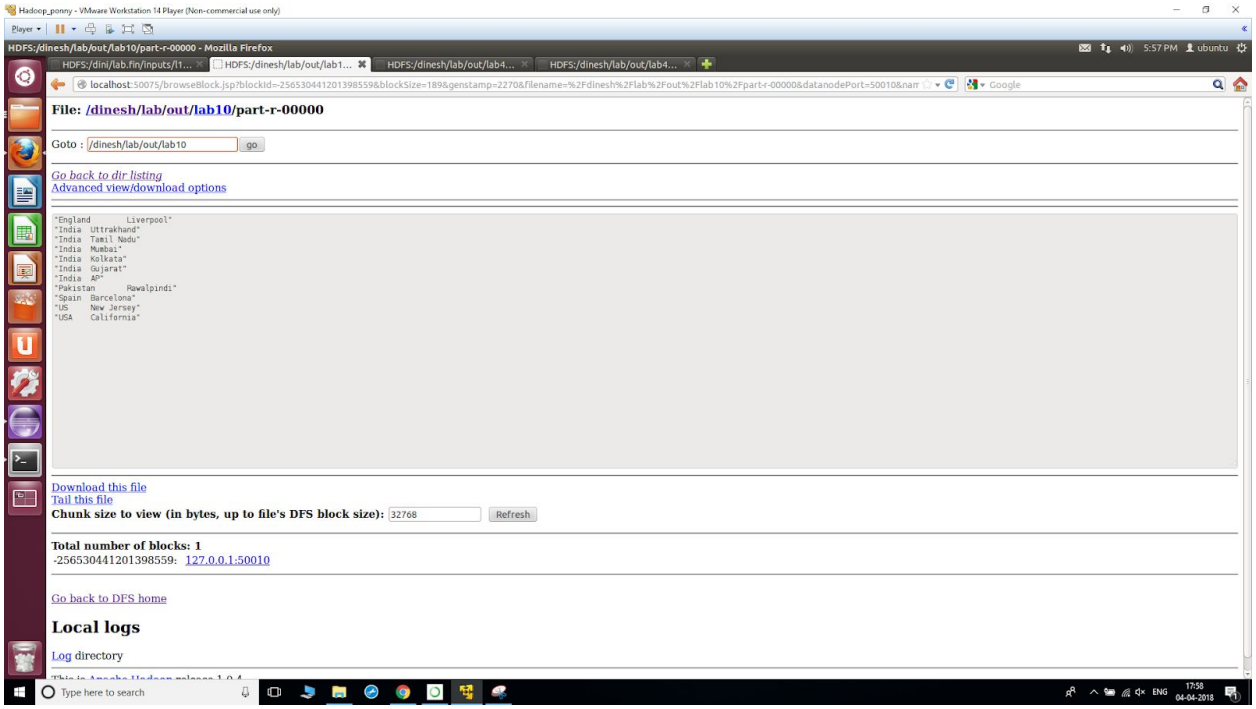
```
    public int compare(WritableComparable w1,WritableComparable w2) {
        CompositeKeyWritable key1 =  (CompositeKeyWritable) w1;
        CompositeKeyWritable key2 = (CompositeKeyWritable) w2;
        return key1.deptNo.compareTo(key2.deptNo);
    }
}
public static class SecondarySortBasicReducer extends Reducer<CompositeKeyWritable, NullWritable, CompositeKeyWritable,
NullWritable> {
    public void reduce(CompositeKeyWritable key, Iterable<NullWritable>
        values, Context context) throws IOException, InterruptedException
    {
        for(NullWritable val:values)
        {
            context.write(key,NullWritable.get());
        }
    }
}

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

    Configuration conf=new Configuration();
    Job job=new Job(conf,"country");
    job.setJarByClass(lab10.class);
```

```
        job.setOutputKeyClass(Text.class);

        job.setOutputValueClass(Text.class);

        job.setMapperClass(mapper1.class);

        job.setMapOutputKeyClass(CompositeKeyWritable.class);

        job.setMapOutputValueClass(NullWritable.class);

        job.setPartitionerClass(SecondarySortBasicPartitioner.class);

        job.setSortComparatorClass(SecondarySortBasicCompKeySortComparator.class);

        job.setGroupingComparatorClass(SecondarySortBasicGroupingComparator.class);

        job.setReducerClass(SecondarySortBasicReducer.class);

        job.setInputFormatClass(TextInputFormat.class);

        job.setOutputFormatClass(TextOutputFormat.class);

        FileInputFormat.addInputPath(job, new Path(args[0]));

        FileOutputFormat.setOutputPath(job, new Path(args[1]));

        job.waitForCompletion(true);

    }

}
```

Input:-



Output:-



reference:-https://dzone.com/articles/mapreduce-algorithms-%E2%80%93