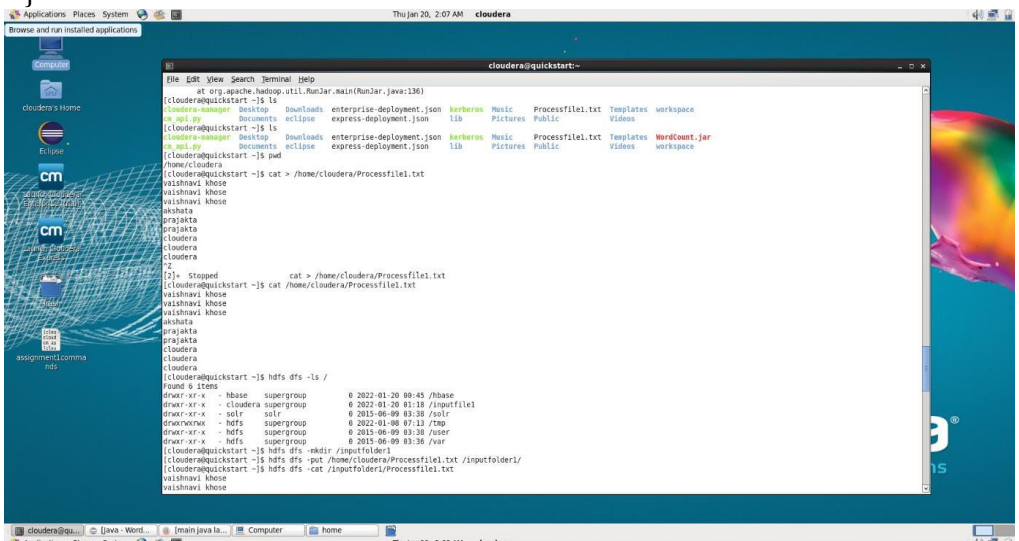
Assignment 2 – Output

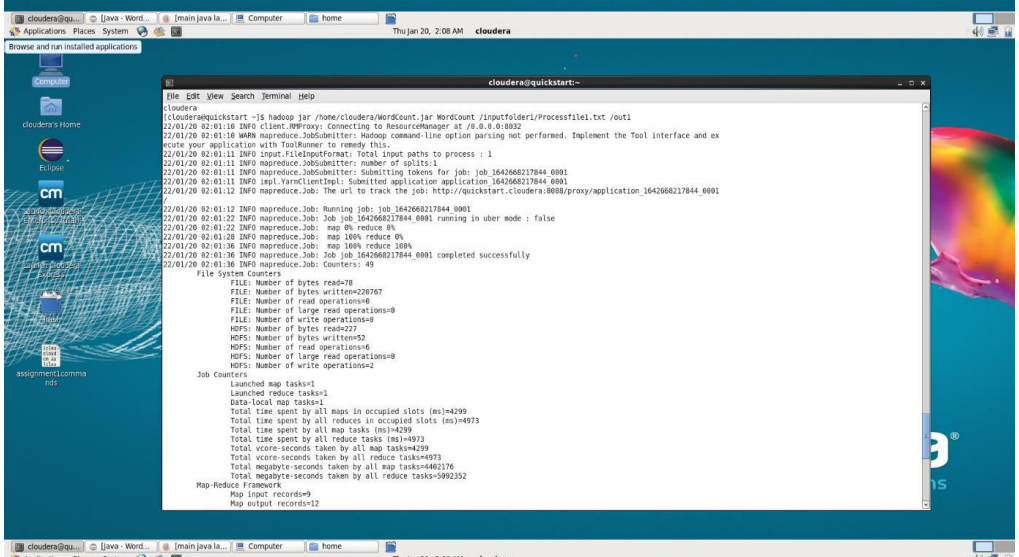
Code: Word Count

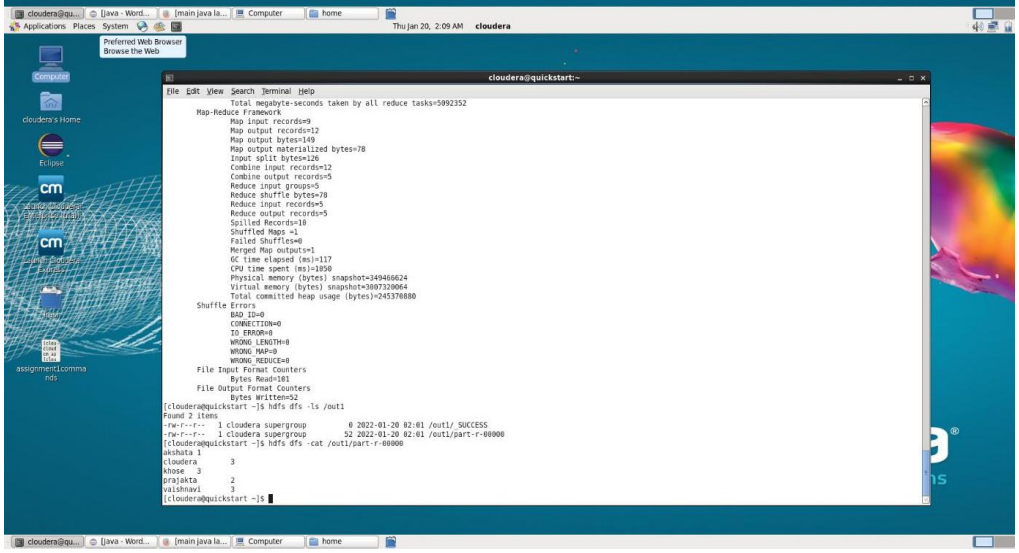
```
import java.io.IOException;
import java.util.StringTokenizer;

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

```
public static class IntSumReducer
    extends
  Reducer<Text,IntWritable,Text,IntWritable> { private
  IntWritable result = new IntWritable();

  public void reduce(Text key, Iterable<IntWritable>
                values, Context context
                ) throws IOException,
    InterruptedException { int sum = 0;
    for (IntWritable val :
      values) { sum +=
      val.get();
    }
    result.set(sum);
    context.write(key,
    result);
  }
}

public static void main(String[] args) throws
  Exception { Configuration conf = new
  Configuration();
  Job job = Job.getInstance(conf, "word count");
  job.setJarByClass(WordCount.class);
  job.setMapperClass(TokenizerMapper.class);
  job.setCombinerClass(IntSumReducer.class);
  job.setReducerClass(IntSumReducer.class);
  job.setOutputKeyClass(Text.class);
  job.setOutputValueClass(IntWritable.class);
  FileInputFormat.addInputPath(job, new
  Path(args[0])); FileOutputFormat.setOutputPath(job,
  new Path(args[1]));
```

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







Code: IP COUNT

```
package ass2;
import java.io.*;
import java.io.IOException;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
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;
import org.apache.hadoop.util.GenericOptionsParser;
public class logfile {
public static void main(String[] args) throws Exception {
Configuration c = new Configuration();
String[] files = new GenericOptionsParser(c, args).getRemainingArgs();
Path input = new Path(files[0]);
Path output = new Path(files[1]);
Job j = new Job(c, "maxdurationip");
j.setJarByClass(logfile.class);
j.setMapperClass(MapForMaxDurationIP.class);
j.setReducerClass(ReduceForMaxDurationIP.class);
j.setOutputKeyClass(Text.class);
j.setOutputValueClass(IntWritable.class);
FileInputFormat.addInputPath(j, input);
FileOutputFormat.setOutputPath(j, output);
System.exit(j.waitForCompletion(true) ? 0 : 1);
}
public static class MapForMaxDurationIP extends Mapper<LongWritable, Text, Text,
```

```
IntWritable> {
public void map(LongWritable key, Text value, Context con) throws IOException,
InterruptedException {
String line = value.toString();
String[] parts = line.split(",");
String ip = parts[2].trim();
int duration = Integer.parseInt(parts[4].trim());
Text outputKey = new Text(ip);
IntWritable outputValue = new IntWritable(duration);
con.write(outputKey, outputValue);
}
}
public static class ReduceForMaxDurationIP extends Reducer<Text, IntWritable, Text,
IntWritable> {
private Text maxIp = new Text();
private IntWritable maxDuration = new IntWritable(Integer.MIN_VALUE);
private Text minIp = new Text();
private IntWritable minDuration = new IntWritable(Integer.MAX_VALUE);
private Text aip = new Text();
private IntWritable td = new IntWritable(Integer.MIN_VALUE);
public void reduce(Text ip, Iterable<IntWritable> durations, Context con)
throws IOException, InterruptedException {
int totalDuration = 0;
// int count = 0;
for (IntWritable duration : durations) {
totalDuration += duration.get();
// count++;
}
aip.set(ip);
td.set(totalDuration);
con.write(aip, td);
// totalDuration = totalDuration/count;
```

```
if (totalDuration > maxDuration.get()) {
maxIp.set(ip);
maxDuration.set(totalDuration);
}
if (totalDuration < minDuration.get()) {
minIp.set(ip);
minDuration.set(totalDuration);
}
}
@Override
protected void cleanup(Context con) throws IOException, InterruptedException {
con.write(new Text("\nIp with Max Active Time : " + maxIp.toString()), maxDuration);
con.write(new Text("\nIp with Min Active Time : " + minIp.toString()), minDuration);
}
}
}
```