Experiment 6: Hadoop Reduce Side Join

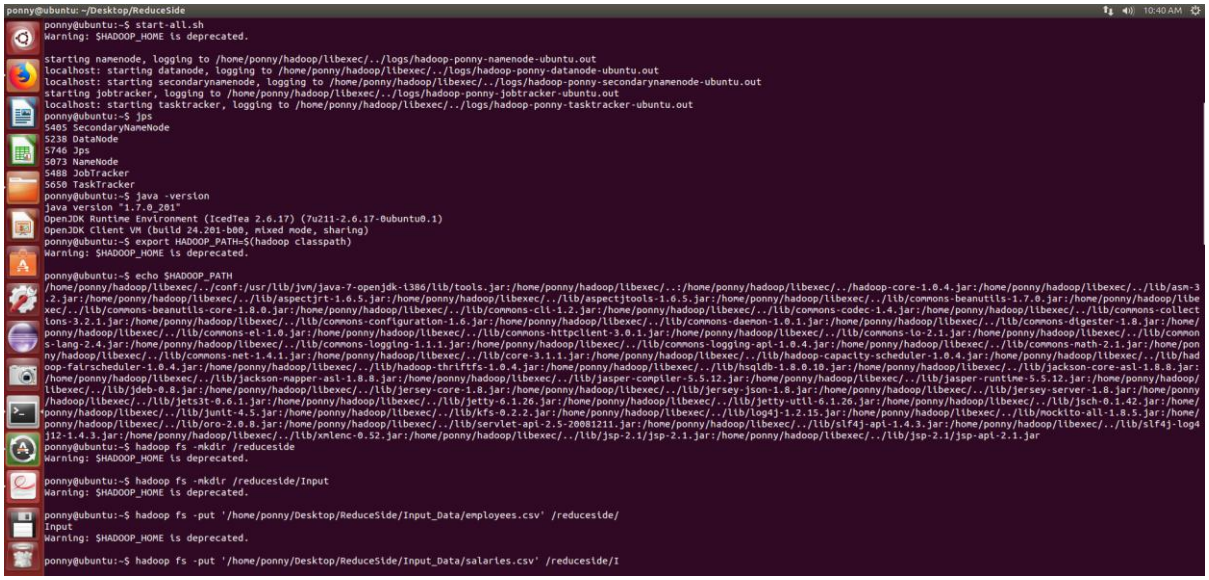
Namansh Singh Maurya
22MIA1034

Aim:

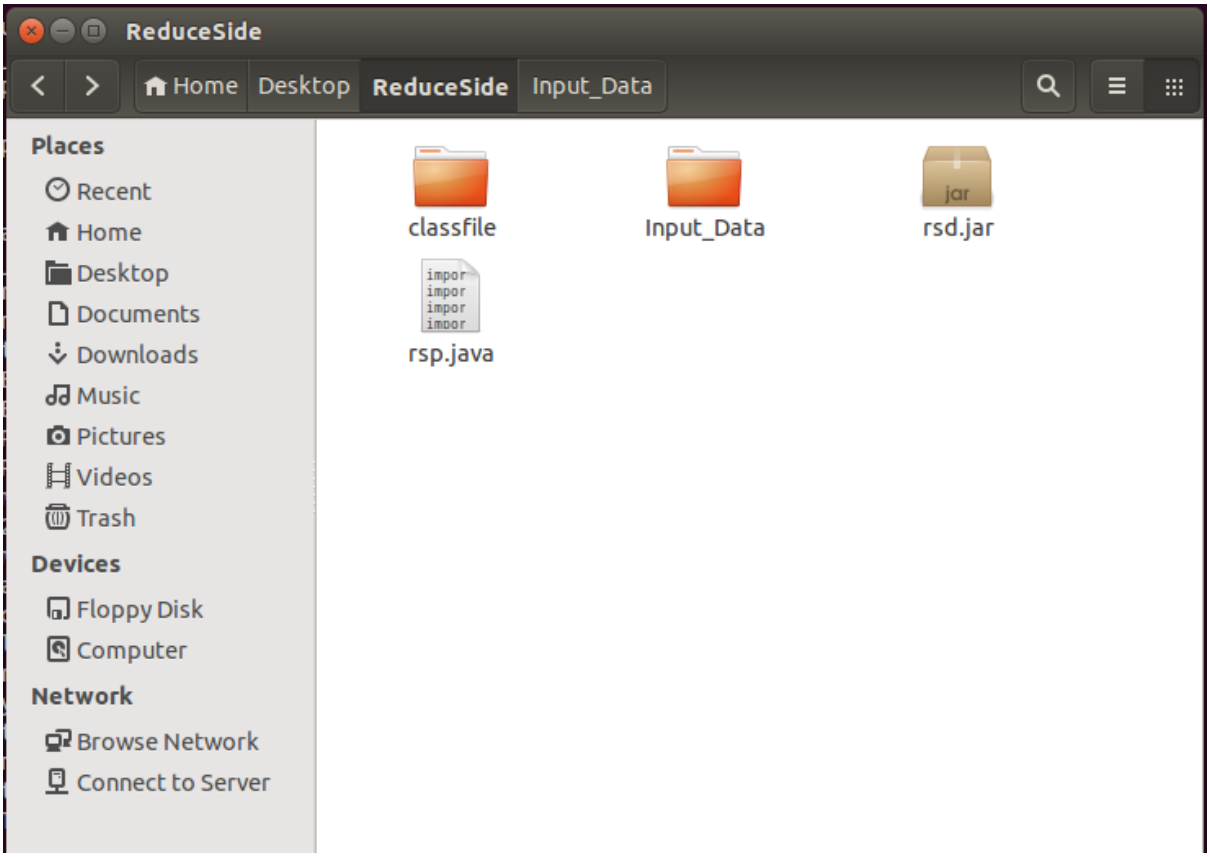
Implement the **Reduce Side Join** using Hadoop MapReduce.

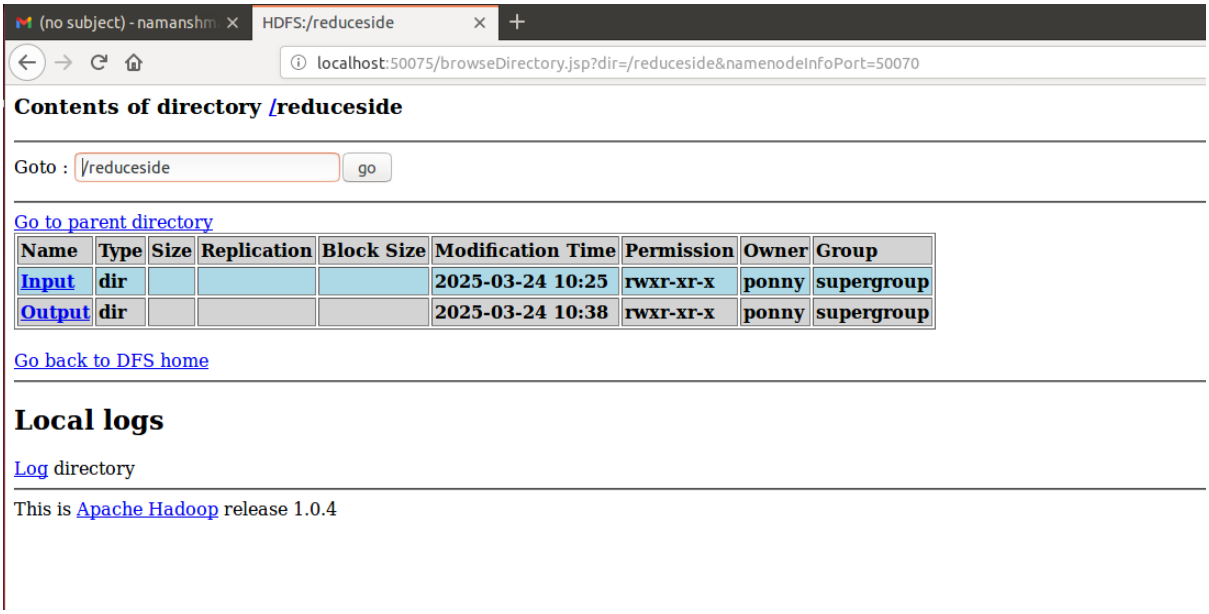
Algorithm/Procedure:

1. **Firstly, we will start our Hadoop system and check if it is working correctly or not,**
   with that, we will export the environment variable of Hadoop and echo it back to the terminal.

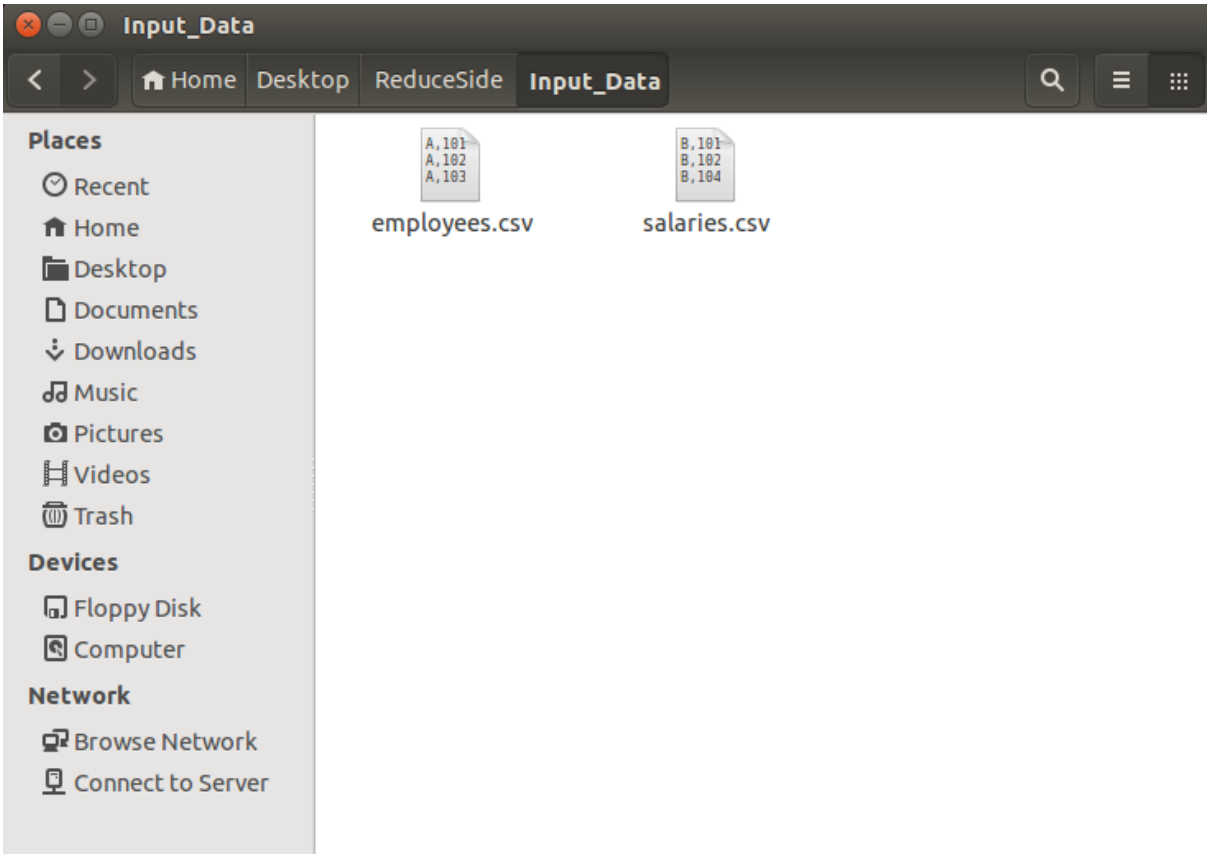


2. **Then we will create one folder named reducejoin**,
   inside that, we will make **input_data** and **outputs** named folders and **rsp.java** file.

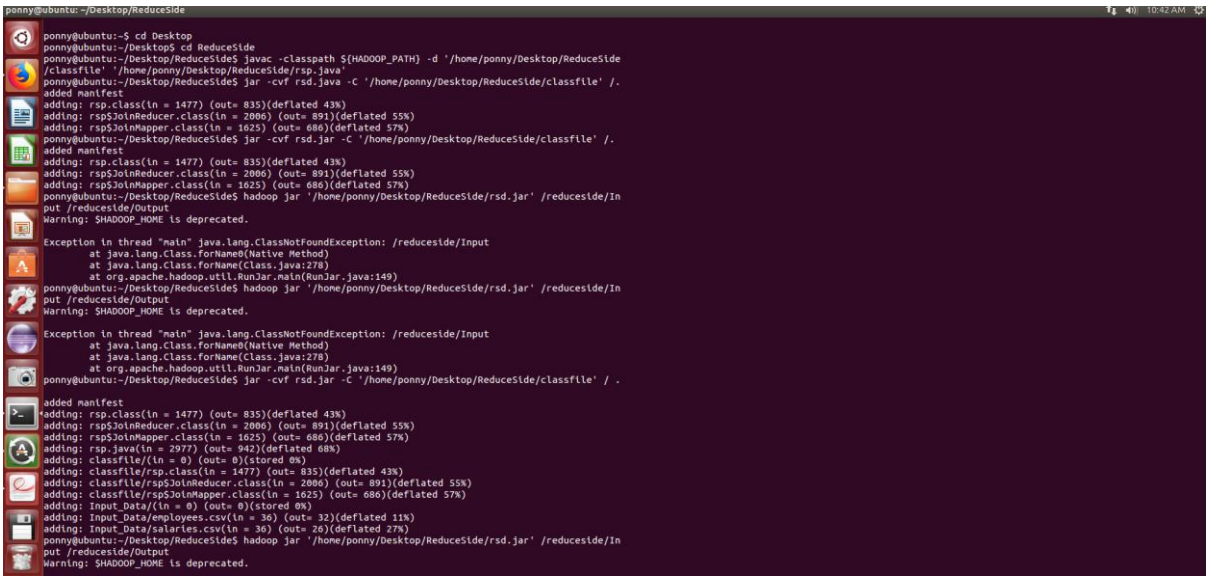


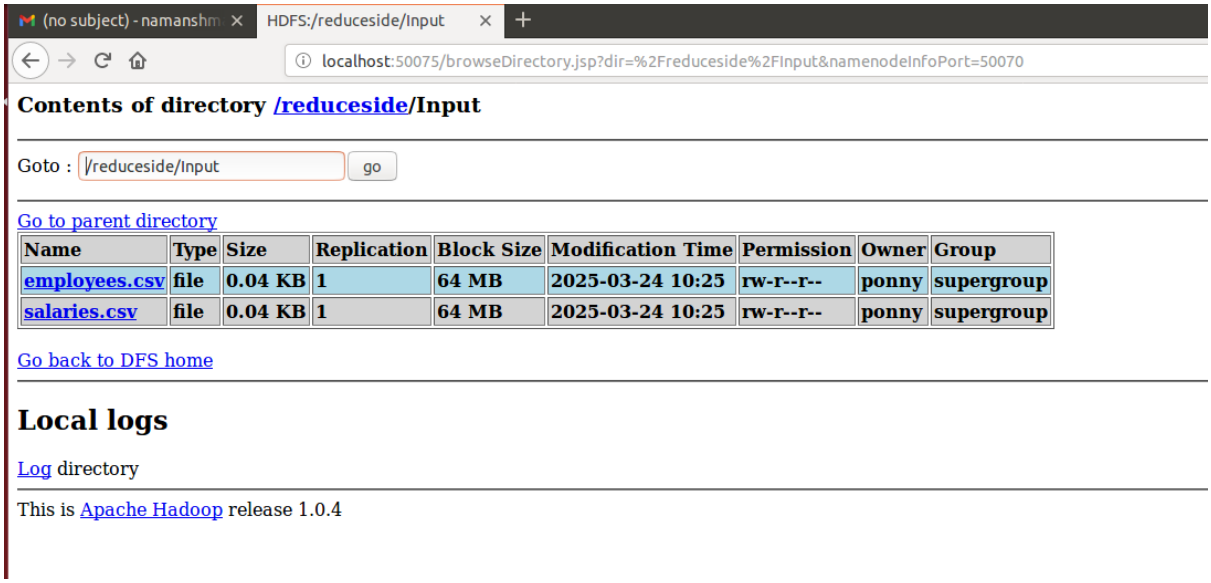


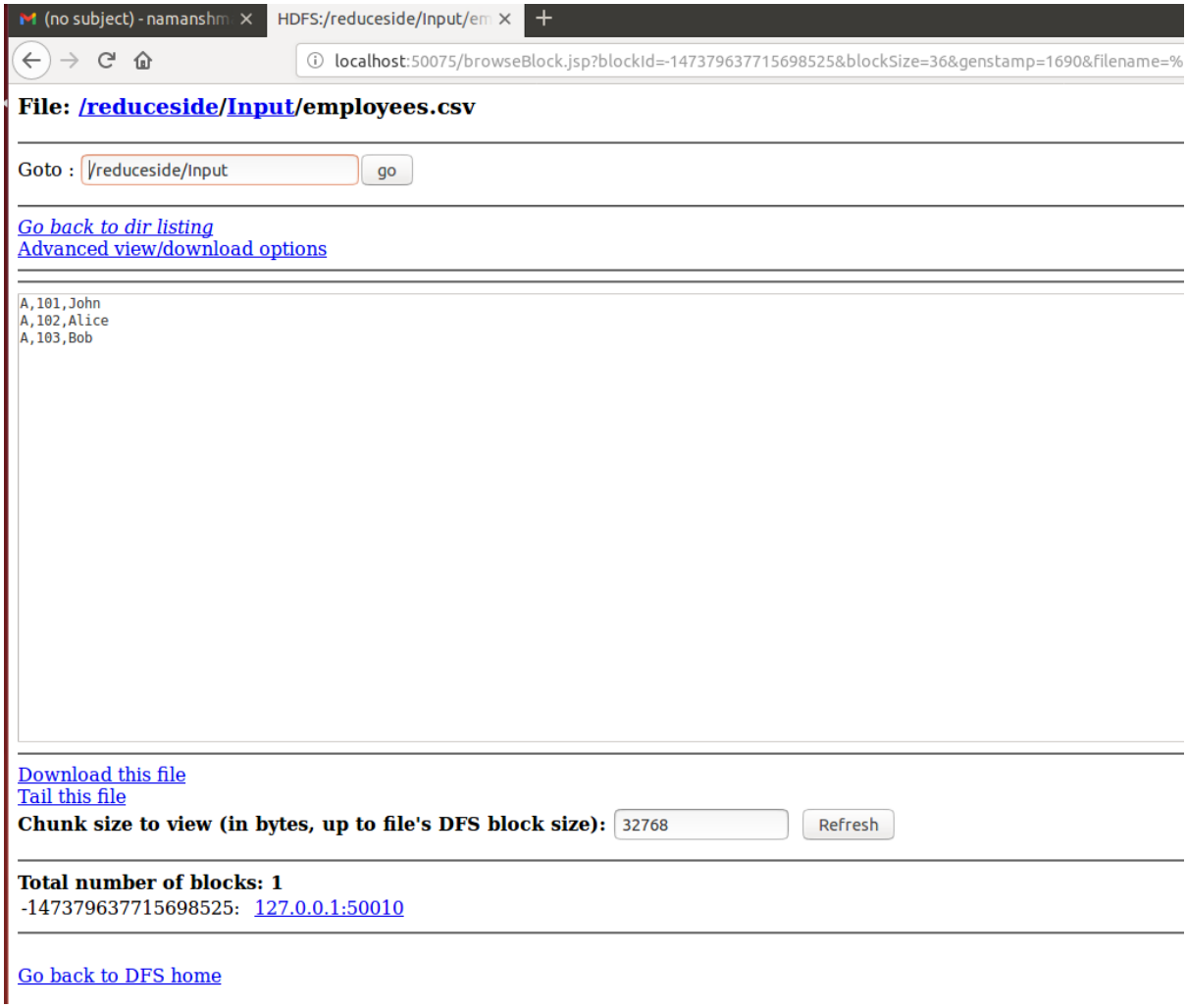
3. **Inside input_data, we will create two text files named employees.csv and salaries.txt.**

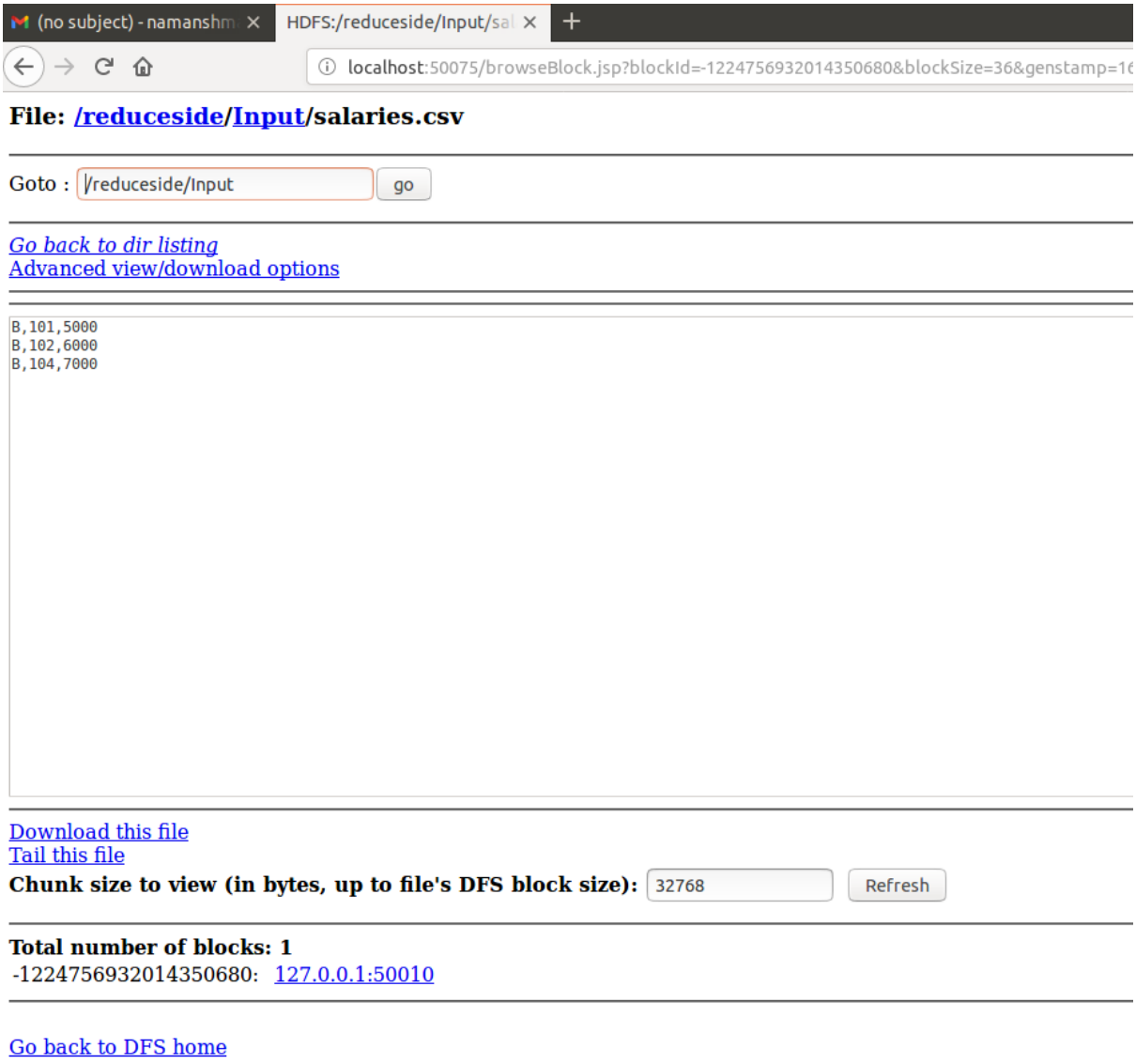


4. **After this, we will make the reducesidejoin directory in the Hadoop system**. We can view that using localhost, and we will put both the text files inside **reducesidejoin/Input** folder.
   Then we will enter the **ReduceSideJoin** directory for making required files inside **classfile**.




(no subject) - namanshm
HDFS:/reduceside/Input
localhost:50075/browseDirectory.jsp?dir=%2Freduceside%2FInput&namenodeInfoPort=50070
Contents of directory /reduceside/Input
Goto : /reduceside/Input
go
Go to parent directory
Name Type Size Replication Block Size Modification Time Permission Owner Group
employees.csv file 0.04 KB 1 64 MB 2025-03-24 10:25 rw-r--r-- ponny supergroup
salaries.csv file 0.04 KB 1 64 MB 2025-03-24 10:25 rw-r--r-- ponny supergroup
Go back to DFS home
Local logs
Log directory
This is Apache Hadoop release 1.0.4

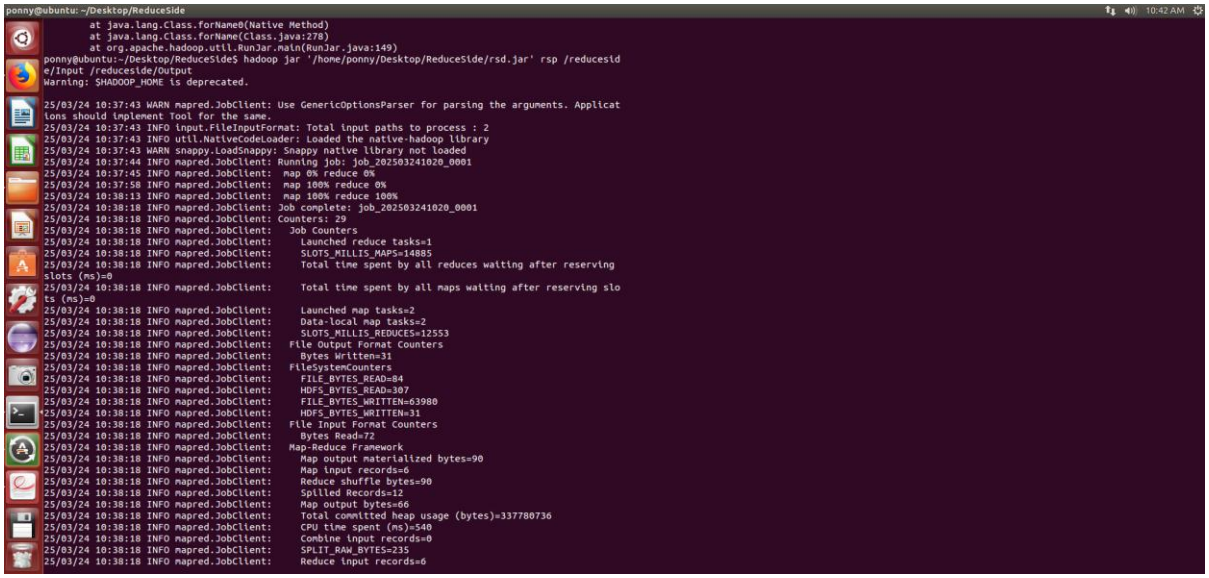

(no subject) - namanshm
HDFS:/reduceside/Input/em
localhost:50075/browseBlock.jsp?blockId=-1473796377156985258blockSize=36&genstamp=1690&filename=%
File: /reduceside/Input/employees.csv
Goto : /reduceside/Input
go
Go back to dir listing
Advanced view/download options
A,101,John
A,102,Alice
A,103,Bob
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
-147379637715698525: 127.0.0.1:50010
Go back to DFS home

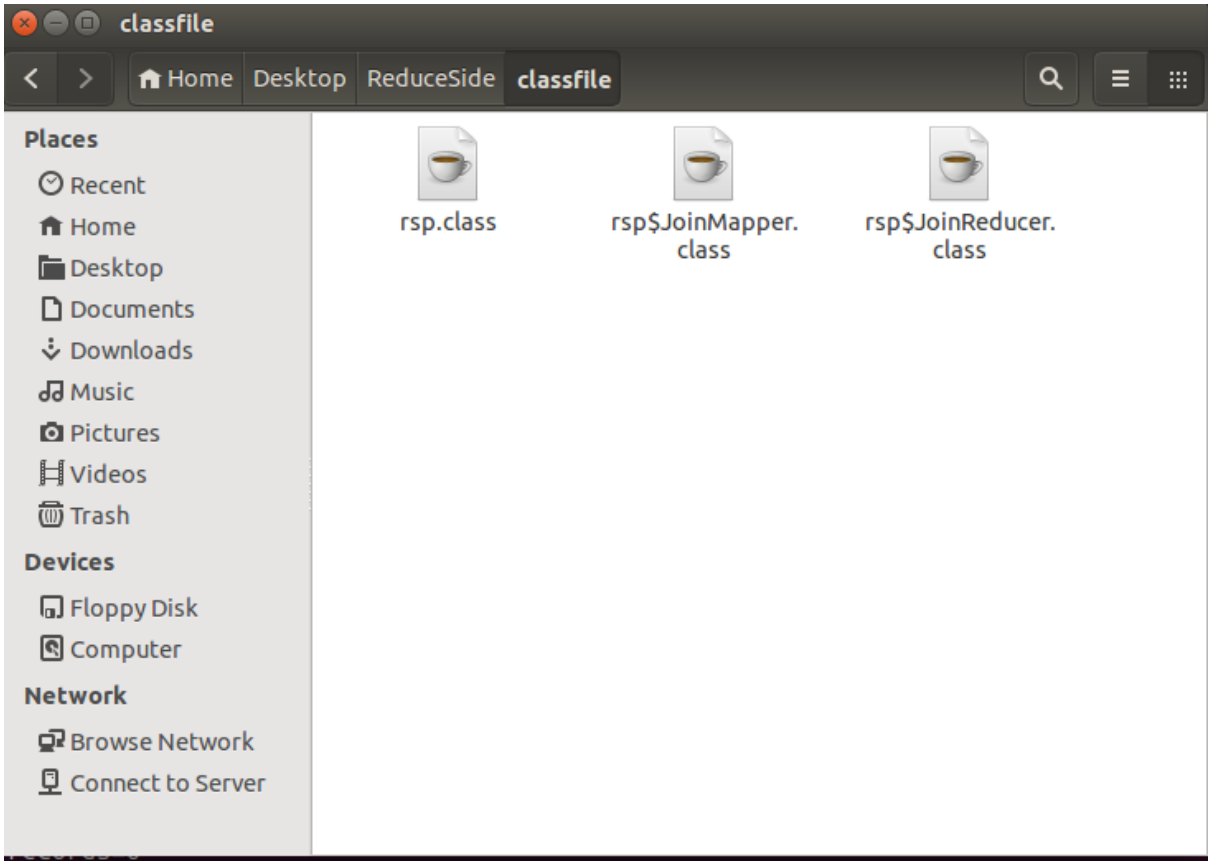


5. **Now using this command:**

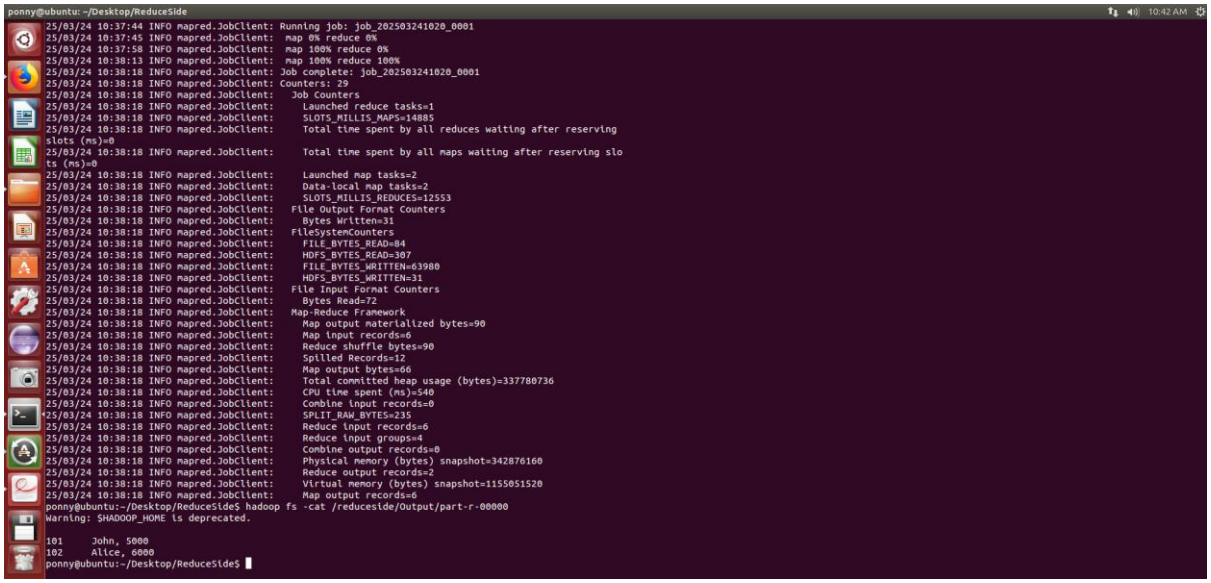
   javac -classpath ${HADOOP_CLASSPATH} -d /home/user/ReduceSideJoin/classfile /home/user/ReduceSideJoin/ReduceSideJoin.java

   we will get the following three files in the **classfile** directory.





6. **Now we will compile the Java file, then we will get the JAR file**,
and we will execute the code and get the required output inside the **output** folder on Hadoop.



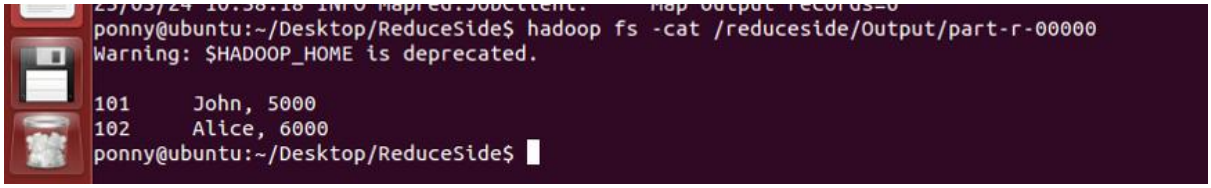
Program:

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

/**
 * Reduce Side Join Example for Employee and Salary Data
 */
public class rsp {

    // Mapper Class
    public static class JoinMapper extends Mapper<Object, Text, Text, Text> {
        public void map(Object key, Text value, Context context) throws
IOException, InterruptedException {
            String[] fields = value.toString().split(",");

            // Ensure valid record
            if (fields.length >= 3) {
                String recordType = fields[0].trim(); // "A" for Employee, "B"
for Salary
                String joinKey = fields[1].trim();    // Employee ID (Join
Key)
                String details = fields[2].trim();    // Employee Name or
Salary
```

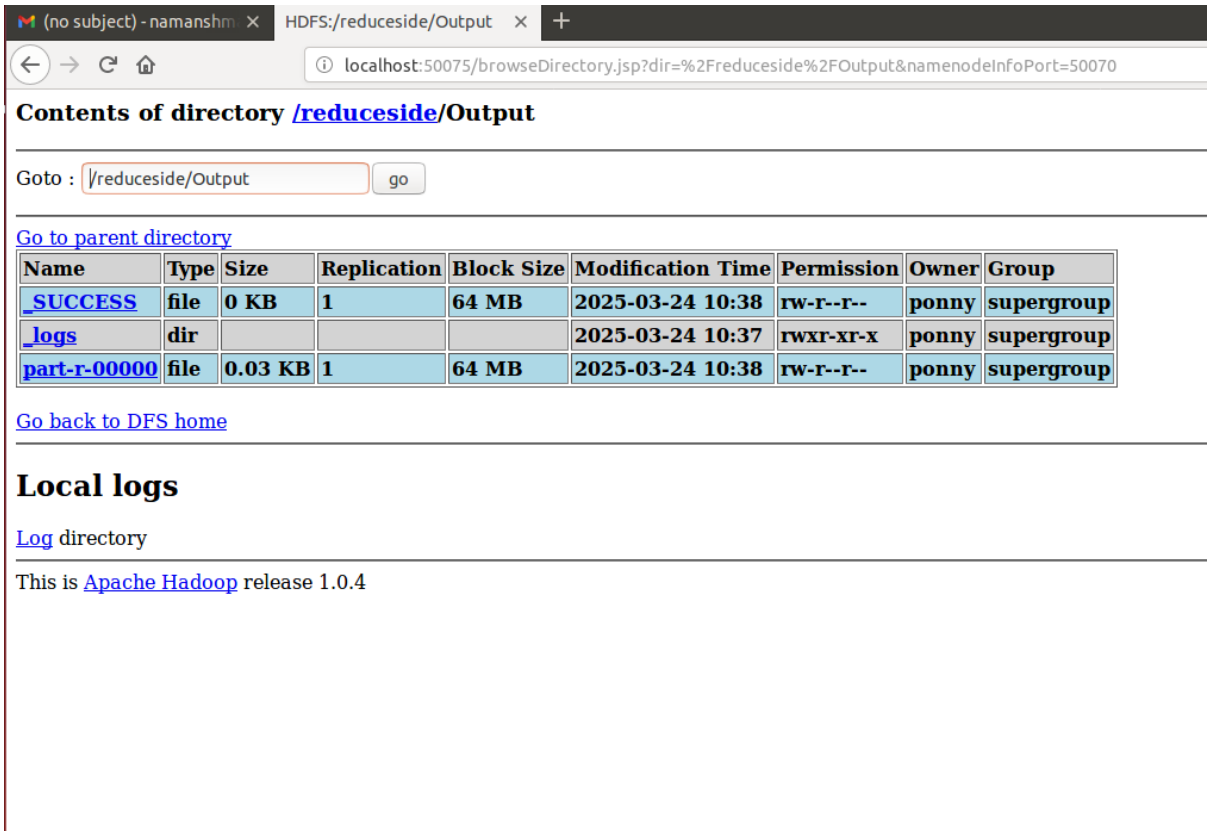
```
                context.write(new Text(joinKey), new Text(recordType + "," +
details));
            }
        }
    }

    // Reducer Class
    public static class JoinReducer extends Reducer<Text, Text, Text, Text> {
        public void reduce(Text key, Iterable<Text> values, Context context)
throws IOException, InterruptedException {
            String employeeName = null;
            String salary = null;

            // Iterate over values
            for (Text val : values) {
                String[] tokens = val.toString().split(",");
                if (tokens.length == 2) {
                    if (tokens[0].equals("A")) {
                        employeeName = tokens[1];  // Employee Name
                    } else if (tokens[0].equals("B")) {
                        salary = tokens[1];  // Employee Salary
                    }
                }
            }

            // Output only if both values exist
            if (employeeName != null && salary != null) {
                context.write(key, new Text(employeeName + ", " + salary));
            }
        }
    }

    // Driver Method
    public static void main(String[] args) throws Exception {
        if (args.length < 2) {
            System.err.println("Usage: ReduceSideJoin <input path> <output
path>");
            System.exit(-1);
        }

        Configuration conf = new Configuration();
        Job job = new Job(conf);
        job.setJarByClass(rsp.class);
        job.setMapperClass(JoinMapper.class);
        job.setReducerClass(JoinReducer.class);
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(Text.class);
```

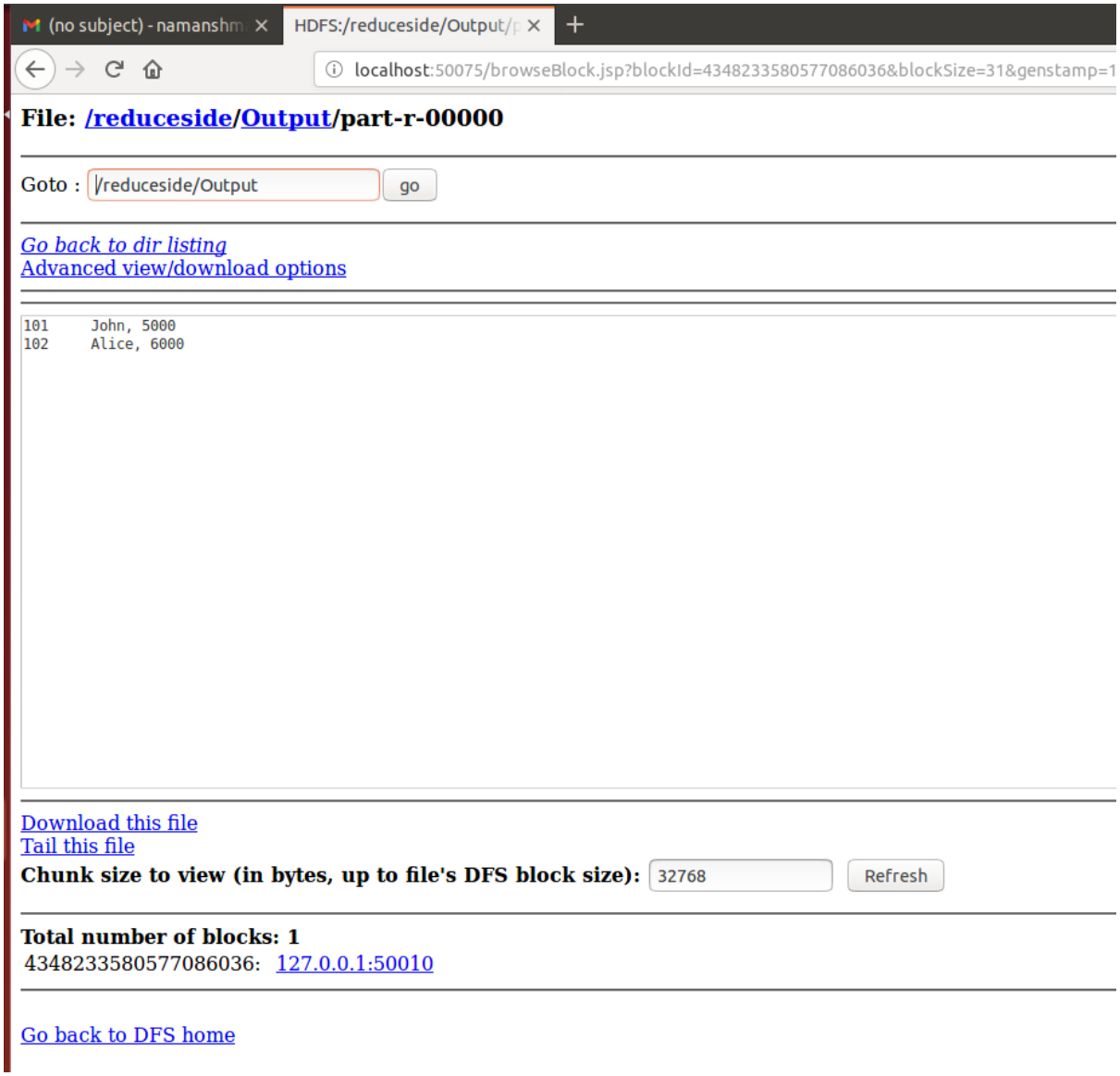
```
        FileInputFormat.addInputPath(job, new Path(args[0]));
        FileOutputFormat.setOutputPath(job, new Path(args[1]));

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

Output in the terminal:



Output in the Localhost:





Result:

The **Reduce Side Join** in Hadoop MapReduce was successfully implemented.
The program merged **customer details with their respective transaction amounts** using the distributed computing framework, enabling efficient large-scale data processing.