**Big Data Frameworks CSE3120**

Lab – 6 Reduce Side Join Experiment

**Name:** Naveen Nidadavolu

**Roll No:** 22MIA1049

**Reduce Side Join**

**Aim:** To implement **Reduce-Side Join** in Hadoop using MapReduce, where two datasets (**employees and salaries**) are joined based on a common key (**EmployeeID**) to produce a final output containing **employee names and their corresponding salaries**.

**Algorithm:**

**Mapper Phase**

1. Read each record from the input files (**employees.csv** and **salaries.csv**).
2. Extract **EmployeeID** as the key.
3. Tag the record:
   * If from **employees.csv**, tag as "A,EmployeeName".
   * If from **salaries.csv**, tag as "B,Salary".
4. Emit (**EmployeeID, TaggedRecord**).

**Shuffle & Sort Phase (Handled by Hadoop)**

1. Hadoop groups all records by **EmployeeID** before sending them to the Reducer.

**Reducer Phase**

1. Initialize **employeeName = null** and **salary = null**.
2. Loop through grouped values:
   * If tagged **"A"**, extract **EmployeeName**.
   * If tagged **"B"**, extract **Salary**.
3. If **both EmployeeName and Salary exist**, emit (**EmployeeID, EmployeeName, Salary**).

**Store Output in HDFS**

1. Write the final **joined result** to the HDFS output directory.

**Program**

import java.io.IOException;

import java.util.Iterator;

import org.apache.hadoop.fs.Path;

import org.apache.hadoop.conf.Configuration;

import org.apache.hadoop.io.\*;

import org.apache.hadoop.mapred.\*;

public class ReduceSideJoin {

// \*\*Mapper Class\*\*

public static class JoinMapper extends MapReduceBase

implements Mapper<LongWritable, Text, Text, Text> {

private Text joinKey = new Text();

private Text joinValue = new Text();

public void map(LongWritable key, Text value, OutputCollector<Text, Text> output, Reporter reporter)

throws IOException {

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

if (fields.length == 2) {

String userId = fields[0];

String data;

// Tagging data based on input source

if (reporter.getInputSplit().toString().contains("users.txt")) {

data = "U:" + fields[1]; // User data

} else {

data = "T:" + fields[1]; // Transaction data

}

joinKey.set(userId);

joinValue.set(data);

output.collect(joinKey, joinValue);

}

}

}

// \*\*Reducer Class (Fixed)\*\*

public static class JoinReducer extends MapReduceBase

implements Reducer<Text, Text, Text, Text> {

private Text result = new Text();

public void reduce(Text key, Iterator<Text> values, OutputCollector<Text, Text> output, Reporter reporter)

throws IOException {

String userName = null;

StringBuilder transactions = new StringBuilder();

while (values.hasNext()) {

String value = values.next().toString();

if (value.startsWith("U:")) {

userName = value.substring(2); // Extract UserName

} else if (value.startsWith("T:")) {

if (transactions.length() > 0) transactions.append(", ");

transactions.append(value.substring(2)); // Extract Amount

}

}

// \*\*Fix: Allow transactions to be printed even if userName is missing\*\*

if (userName == null) {

userName = "Unknown"; // If userName is missing, assign a default value

}

if (transactions.length() > 0) { // Only output if there are transactions

result.set(userName + "\t" + transactions.toString());

output.collect(key, result);

}

}

}

// \*\*Main Class\*\*

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

JobConf conf = new JobConf(ReduceSideJoin.class);

conf.setJobName("Reduce-Side Join");

conf.setMapperClass(JoinMapper.class);

conf.setReducerClass(JoinReducer.class);

conf.setOutputKeyClass(Text.class);

conf.setOutputValueClass(Text.class);

FileInputFormat.addInputPath(conf, new Path(args[0])); // Users file

FileInputFormat.addInputPath(conf, new Path(args[1])); // Transactions file

FileOutputFormat.setOutputPath(conf, new Path(args[2]));

conf.setNumReduceTasks(1); // Reducer is required

JobClient.runJob(conf);

}

}

**Input files**

* File 1 (users.txt): ID Name
* File 2 (trans.txt): ID Price

**Procedure**

1. Create a folder ReduceSideJoin. Inside the folder create the Input folder and place the input text files and also create an empty folder classfile.

A screenshot of a computer

AI-generated content may be incorrect.

1. Input text files in Input folder

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer program

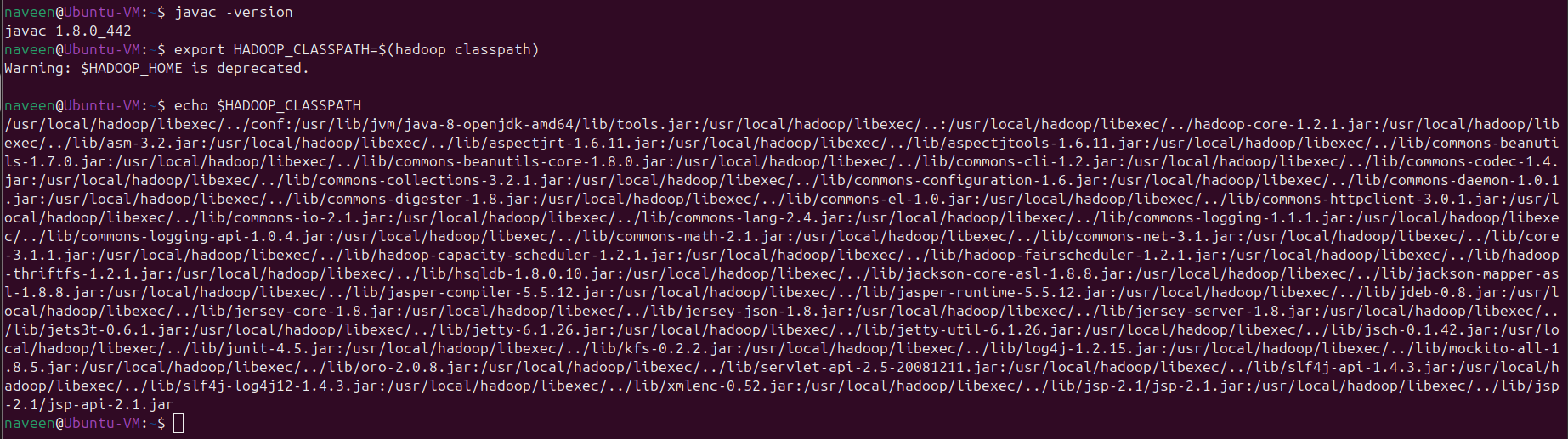
AI-generated content may be incorrect.

1. Start the Hadoop services and check the version

A screenshot of a computer

AI-generated content may be incorrect.

1. Store the Hadoop class path in a variable



1. Create directory reducesidejoin/Input to store input text files in Hadoop.

A screenshot of a computer program

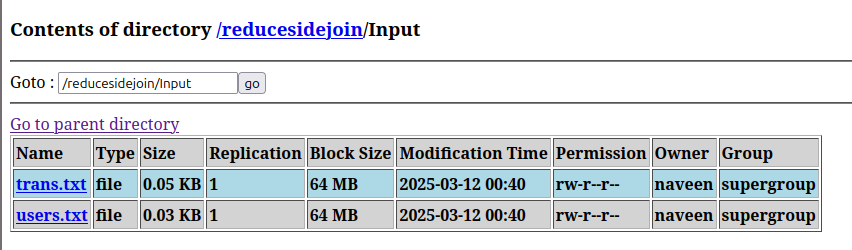
AI-generated content may be incorrect.

1. Place the input text files in the created directory.

A computer screen with white text

AI-generated content may be incorrect.

1. Check the files by going to HDFS NameNode Web UI using the port 50070.

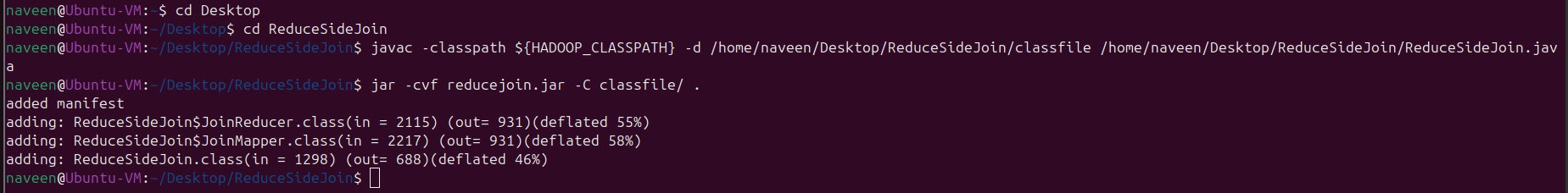


A screenshot of a computer

AI-generated content may be incorrect.A screenshot of a computer

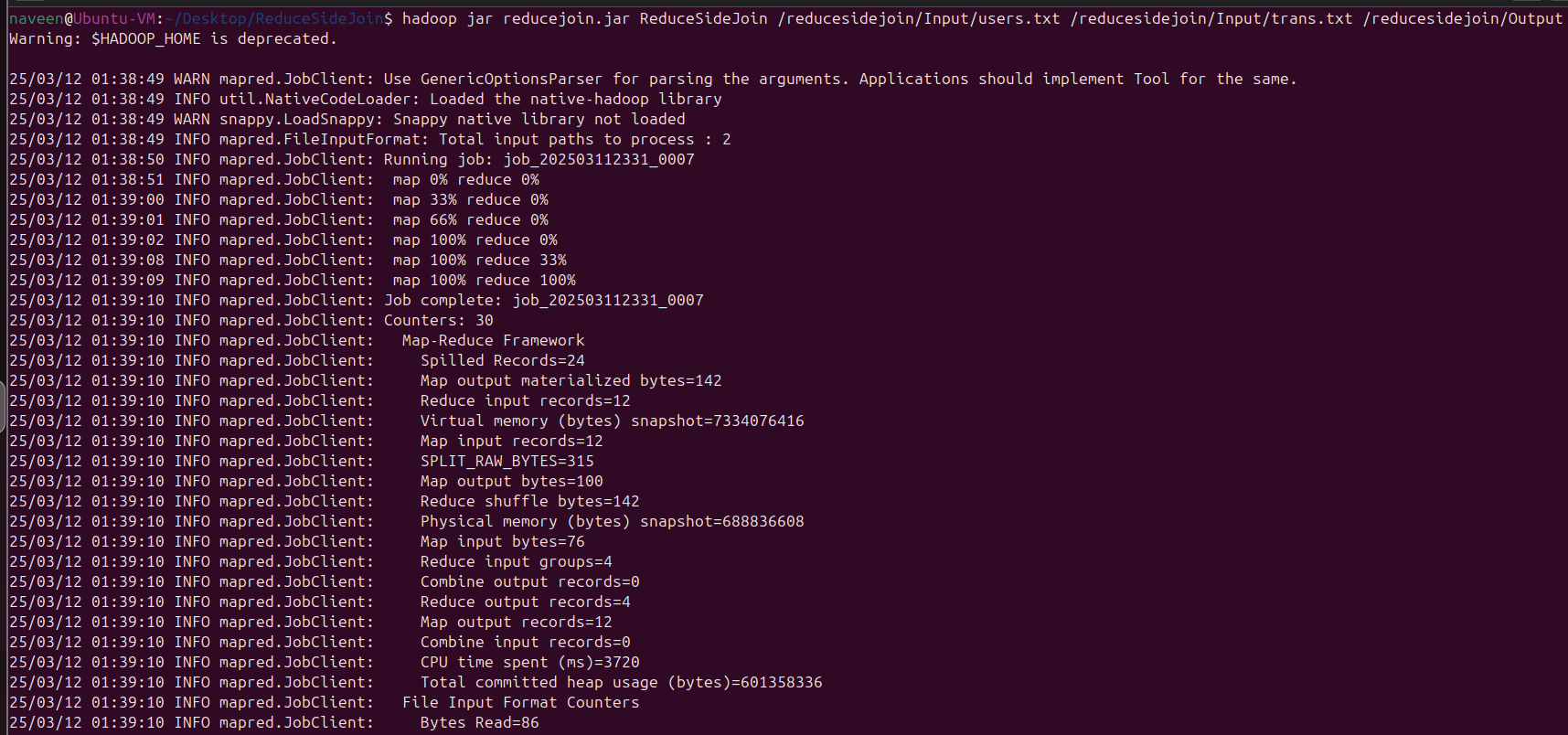
AI-generated content may be incorrect.

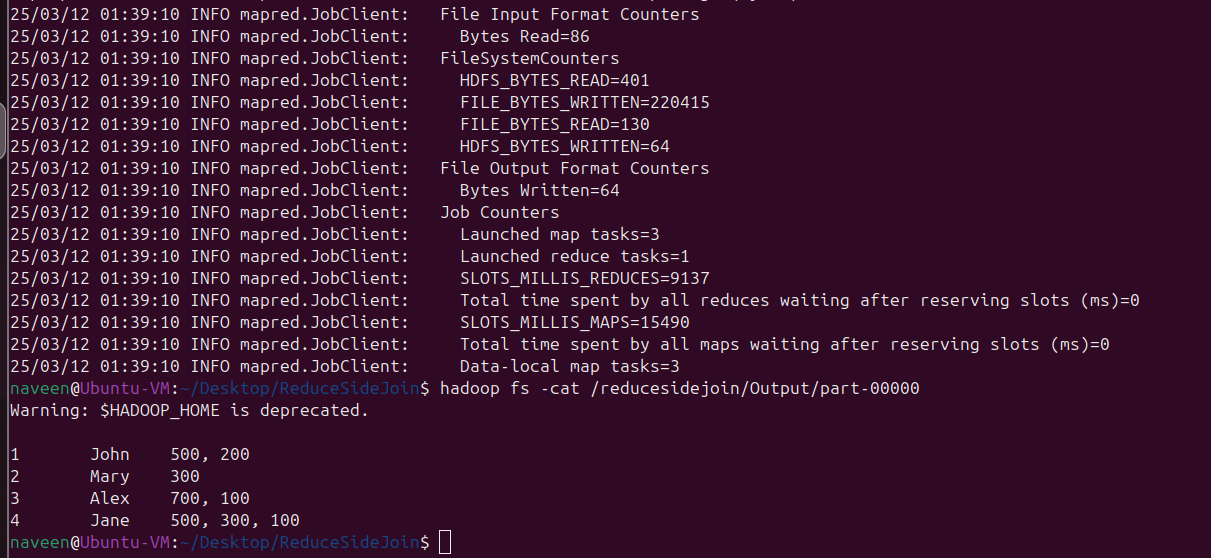
1. Compile the ReduceSideJoin.java and store it in the classfile and create reducejoin.jar

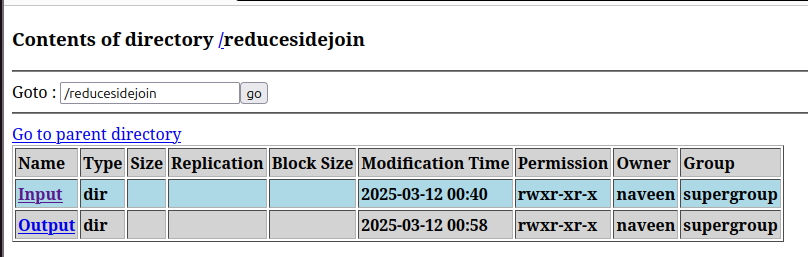
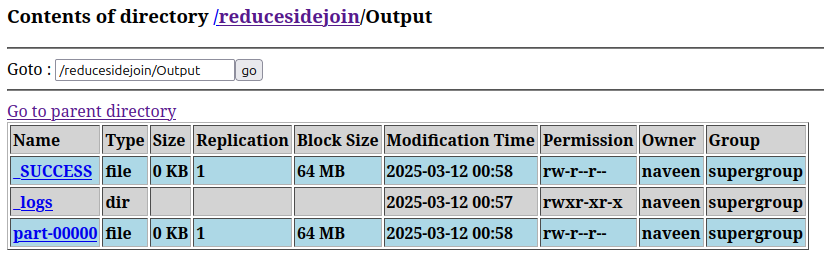
A screenshot of a computer

AI-generated content may be incorrect.

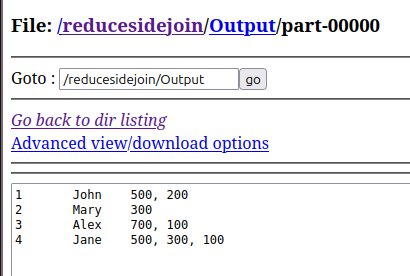
1. Run the Hadoop job and store the output







**Output:**



**Result:** Successfully performed a reduce side join operation using MapReduce, merging user details with their transactions based on ID.