**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 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);

}

}

**Input files**

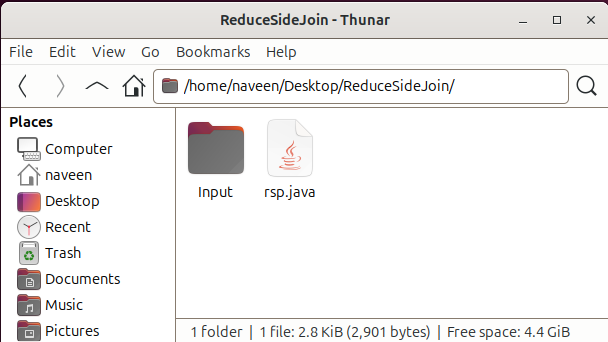
* File 1: employees.csv
* File 2: salaries.csv

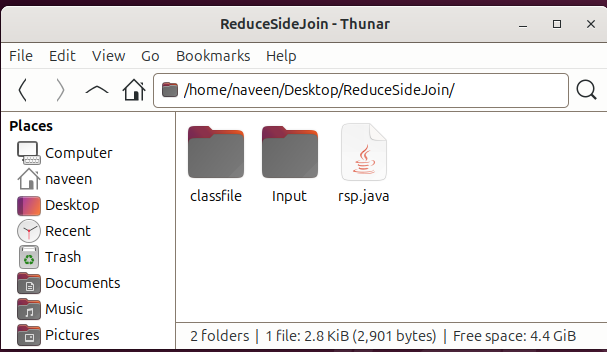
**Procedure**

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.



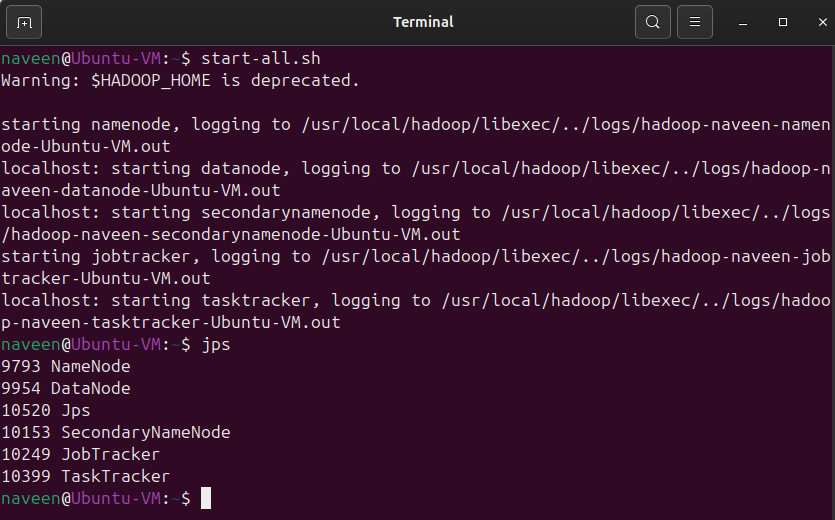


1. Input datsets in Input folder

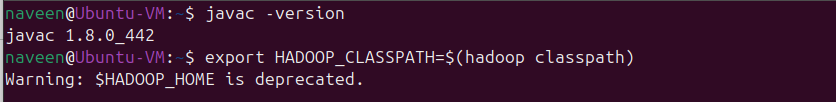
A screenshot of a computer

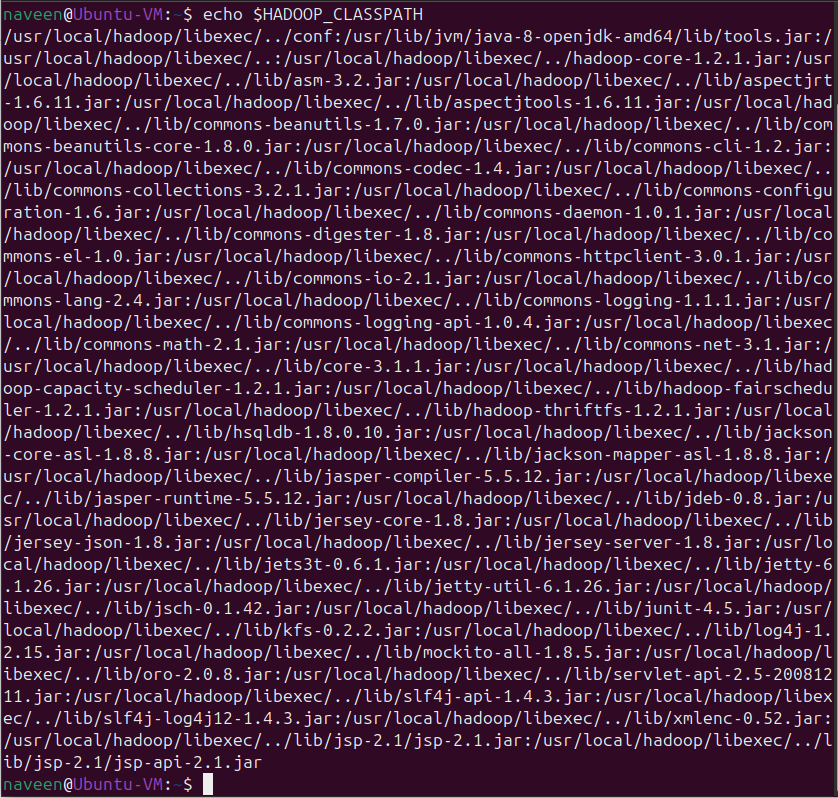
AI-generated content may be incorrect.

1. Start the Hadoop services and basic operations

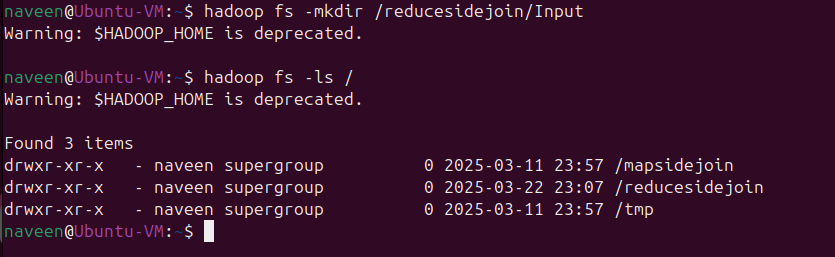


1. Store the Hadoop class path in a HADOOP\_PATH variable

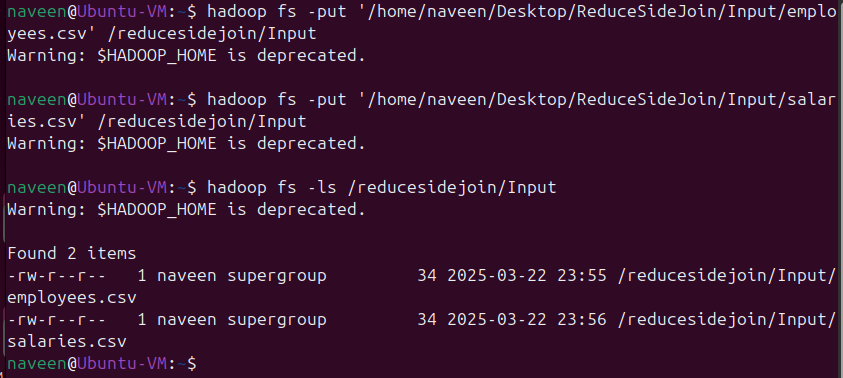


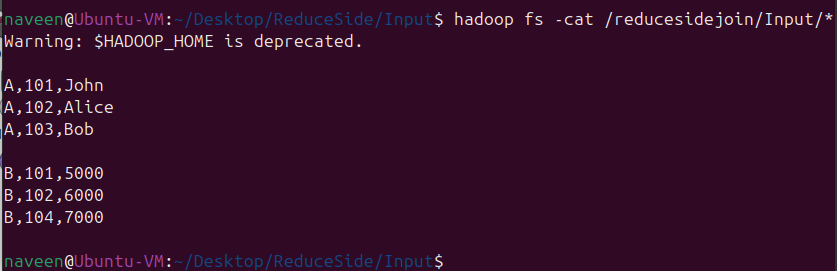


1. Create directory reducesidejoin/Input to store input text files in Hadoop.

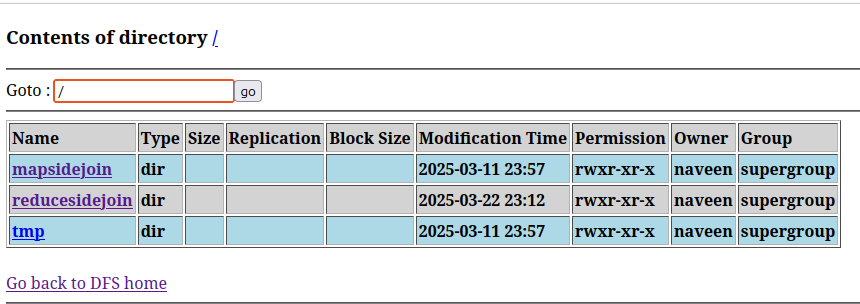


1. Place the input text files in the created directory.





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.

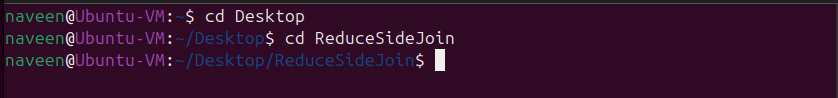
A screenshot of a computer

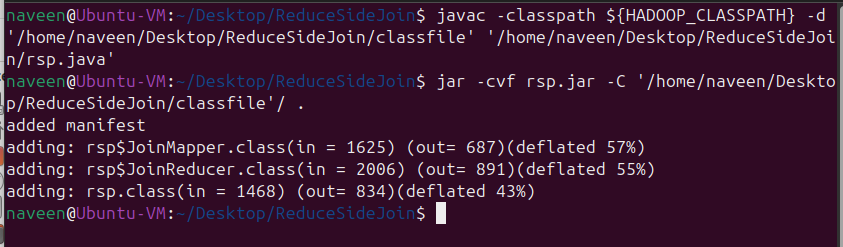
AI-generated content may be incorrect.

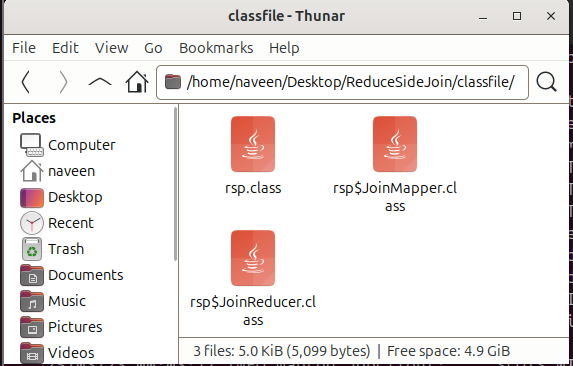
A screenshot of a computer

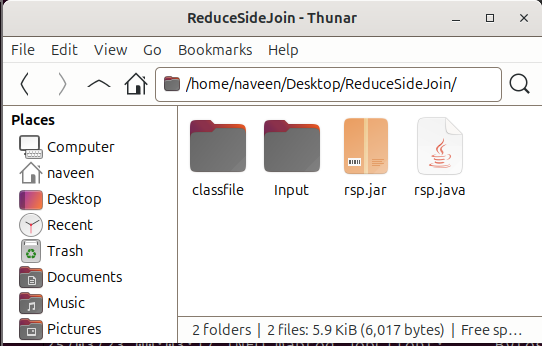
AI-generated content may be incorrect.

1. Compile the reduce side join program rsp.java and store it in the classfile and create jar file rsp.jar

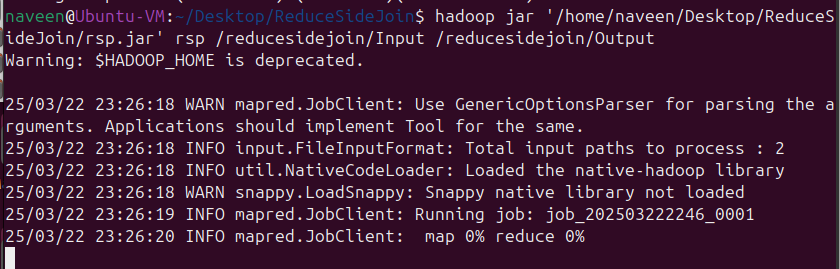


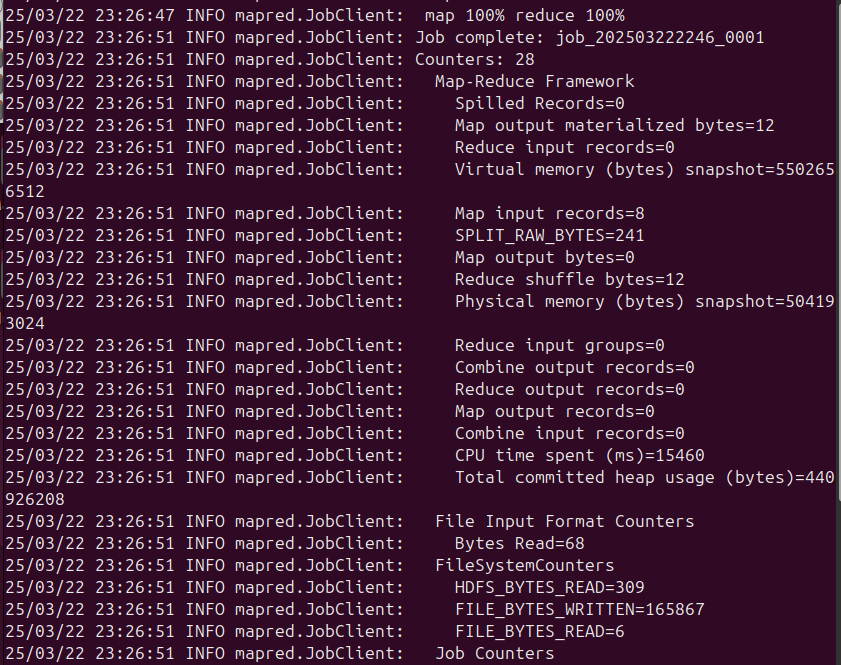


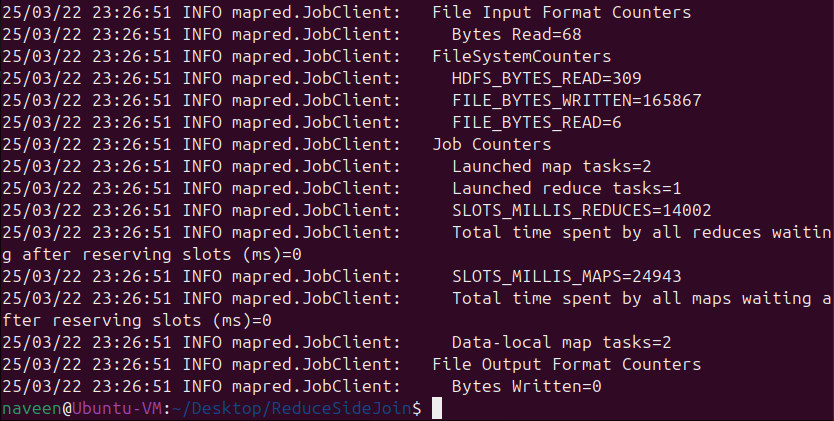




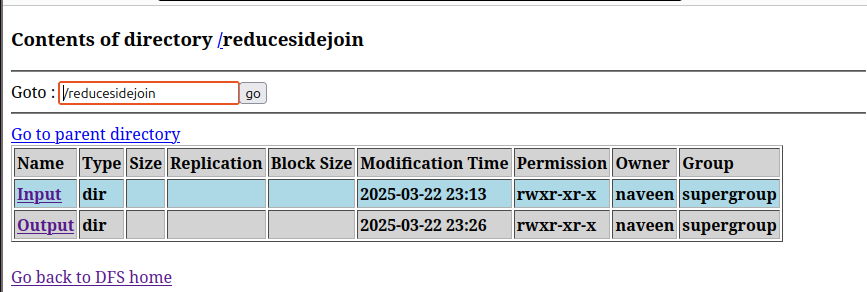
1. Run the Hadoop job and store the output

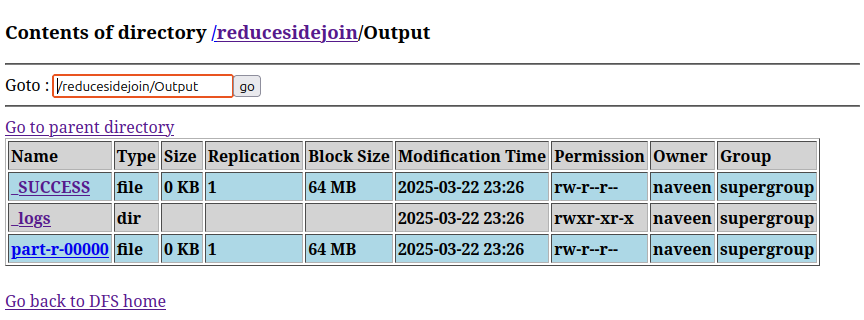






1. Check the Output files by going to HDFS NameNode Web UI using the port 50070.

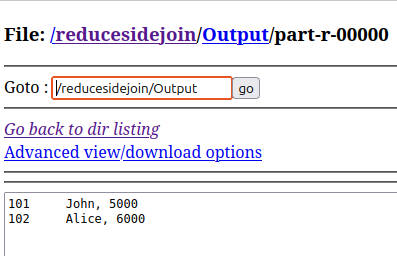




**Output:**

A black background with white text

AI-generated content may be incorrect.



**Result:**

Successfully performed a **Reduce-Side Join** operation using **MapReduce**, merging employee details with their salaries based on **EmployeeID**.