1. Implement Map-Reduce application to find sum of salaries of employees for each department

**Input file for Employee salaries**

Assume we have the following employee data in a text file (employee.txt):

emp1 dept1 5000

emp2 dept1 6000

emp3 dept2 4500

emp4 dept2 5500

emp5 dept3 7000

### MapReduce Components

1. **Mapper Class:** Reads the employee records and emits the department as the key and the salary as the value.
2. **Reducer Class:** Sums the salaries for each department.
3. **Driver Class:** Sets up and runs the job.

**Complete Code**

**package com.example.employeesalary;**

**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;**

**public class EmployeeSalary {**

**// Mapper Class**

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

**private Text department = new Text();**

**private IntWritable salary = new IntWritable();**

**@Override**

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

**String line = value.toString();**

**String[] fields = line.split("\\s+");**

**if (fields.length == 3) {**

**department.set(fields[1]);**

**salary.set(Integer.parseInt(fields[2]));**

**context.write(department, salary);**

**}**

**}**

**}**

**// Reducer Class**

**public static class SalaryReducer extends Reducer<Text, IntWritable, Text, IntWritable> {**

**@Override**

**protected 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));**

**}**

**}**

**// Driver Class**

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

**if (args.length != 2) {**

**System.err.println("Usage: EmployeeSalary <input path> <output path>");**

**System.exit(-1);**

**}**

**Configuration conf = new Configuration();**

**Job job = Job.getInstance(conf, "Employee Salary Sum");**

**job.setJarByClass(EmployeeSalary.class);**

**job.setMapperClass(SalaryMapper.class);**

**job.setReducerClass(SalaryReducer.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);**

**}**

**}**

**Explanation**

1. **Mapper Class:**
   * **SalaryMapper** reads each line of the input data.
   * It splits the line into fields using whitespace as the delimiter.
   * The department is set as the key and the salary as the value.
   * It emits the department and salary as key-value pairs.
2. **Reducer Class:**
   * **SalaryReducer** receives each department and the list of salaries associated with it.
   * It iterates through the salaries to compute the sum.
   * It emits the department and the sum of salaries.
3. **Driver Class:**
   * **EmployeeSalary** sets up the job configuration, specifying the mapper and reducer classes, and the input and output formats and paths.
   * The job is then submitted and monitored until completion.

**Running the MapReduce Job**

1. **Compile the Code:**
   * Save the code in a file, say EmployeeSalary.java.
   * Compile the Java file using the Hadoop libraries.

**javac -classpath `hadoop classpath` -d . EmployeeSalary.java**

**jar -cvf employeesalary.jar -C . .**

**Run the Job:**

* Use the Hadoop command to run the job. Provide input and output paths.

**hadoop jar employeesalary.jar com.example.employeesalary.EmployeeSalary /input/path /output/path**

**Output:**

* The output will be a file located at the specified output path, containing the sum of salaries for each department.

This example demonstrates how to use MapReduce to calculate the sum of salaries for each department in a set of employee records. The Mapper class reads the data and emits the department and salary. The Reducer class sums the salaries for each department and writes the result to the output. The Driver class sets up and runs the MapReduce job. This approach can be adapted to other types of data aggregation tasks using the MapReduce framework.