Ex.No: 12	**Implementing Map Side Join in MapReduce**
Date: 3.11.25	

Aim:

To implement a **Map-Side Join** using Hadoop MapReduce for joining a large dataset with a small lookup dataset using the Distributed Cache mechanism

Procedure:

1. **Prepare input files:**
   customers.txt (Small file to be distributed)
   C101,John
   C102,Mary
   C103,Steve
   orders.txt (Large file)
   O1001,C101,500
   O1002,C102,1000
   O1003,C103,300
2. **Add the small file to HDFS:**

   hdfs dfs -mkdir /input

   hdfs dfs -put orders.txt /input/

   hdfs dfs -put customers.txt /input/
3. **Compilation and Execution:**
   Compile and Create JAR:

   javac -classpath $(hadoop classpath) -d . MapSideJoinMapper.java MapSideJoinDriver.java

   jar -cf mapjoin.jar *

   Run MapReduce Job:

   hadoop jar mapjoin.jar MapSideJoinDriver
4. **View Output:**

   hdfs dfs -cat /output_mapjoin/part-m-00000

Program:

MapSideJoinMapper.java :

```
import java.io.*;
import java.util.HashMap;
import org.apache.hadoop.io.*;
import org.apache.hadoop.fs.Path;
```

```
import org.apache.hadoop.mapreduce.*;
import org.apache.hadoop.conf.Configuration;

public class MapSideJoinMapper extends Mapper<LongWritable, Text, Text, Text> {

    private HashMap<String, String> customerMap = new HashMap<>();
    private Text outputKey = new Text();
    private Text outputValue = new Text();

    @Override
    protected void setup(Context context) throws IOException {
        Path[] cacheFiles = context.getLocalCacheFiles();
        BufferedReader reader = new BufferedReader(new
FileReader(cacheFiles[0].toString()));

        String line;
        while ((line = reader.readLine()) != null) {
            String[] parts = line.split(",");
            customerMap.put(parts[0], parts[1]); // C101 → John
        }
        reader.close();
    }

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

        String[] parts = value.toString().split(",");
        String orderId = parts[0];
        String custId = parts[1];
        String amount = parts[2];

        String custName = customerMap.get(custId);
```

```
        if (custName != null) {
            outputKey.set(orderId);
            outputValue.set(custName + "," + amount);
            context.write(outputKey, outputValue);
        }
    }
}
```

Driver Class:

```
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.lib.input.*;
import org.apache.hadoop.mapreduce.lib.output.*;

public class MapSideJoinDriver {
    public static void main(String[] args) throws Exception {
        Configuration conf = new Configuration();
        Job job = Job.getInstance(conf, "Map Side Join");

        job.setJarByClass(MapSideJoinDriver.class);
        job.setMapperClass(MapSideJoinMapper.class);
        job.setNumReduceTasks(0); // No reducer
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(Text.class);

        // Add customers.txt to distributed cache
        job.addCacheFile(new Path("/input/customers.txt").toUri());

        FileInputFormat.addInputPath(job, new Path("/input/orders.txt"));
        FileOutputFormat.setOutputPath(job, new Path("/output_mapjoin"));

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

```
    }
}
```

Output:

O1001 John,500

O1002 Mary,1000

O1003 Steve,300

Result:

Map-side join was successfully implemented using the Distributed Cache. The join was performed in the Mapper, improving performance by avoiding the shuffle and reduce phases.