1. **Implement a distributed application on Hadoop framework to count word frequency with Map Reduce.**

**Description:**

In Hadoop, MapReduce is a computation that decomposes large manipulation jobs into individual tasks that can be executed in parallel across a cluster of servers. The results of tasks can be joined together to compute final results.

MapReduce consists of 2 steps:

* **Map Function –**It takes a set of data and converts it into another set of data, where individual elements are broken down into tuples (Key-Value pair).

**Example –**(Map function in Word Count)

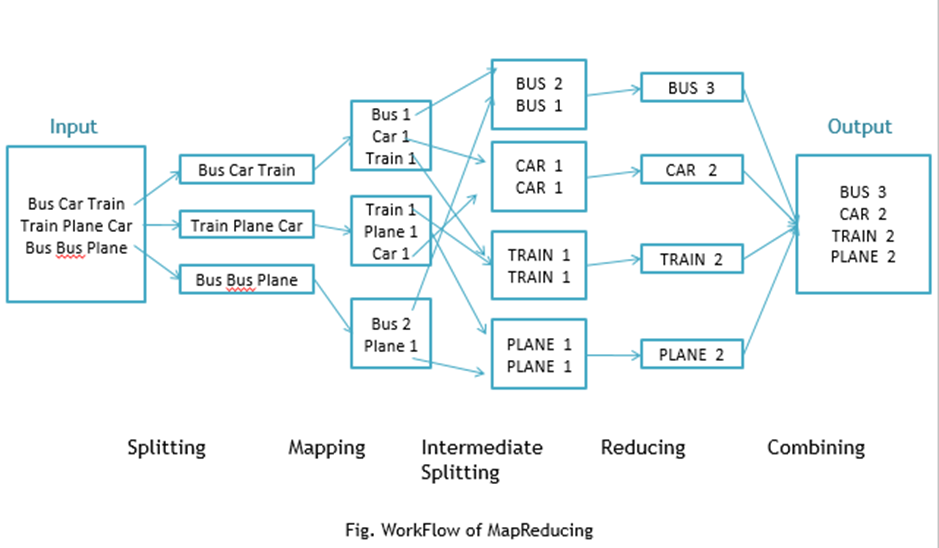
|  |  |  |
| --- | --- | --- |
| **Input** | Set of data | Bus, Car, bus,  car, train, car, bus, car, train, bus, TRAIN,BUS, bus, car, CAR, car, BUs, TrAin |
| **Output** | Convert into another set of data  (Key,Value) | (Bus,1), (Car,1), (bus,1), (car,1), (train,1),  (car,1), (bus,1), (car,1), (train,1), (bus,1),  (TRAIN,1),(BUS,1), (BUS,1), (car,1), (CAR,1),  (car,1), (BUs,1), (TrAin,1) |

* **Reduce Function –**Takes the output from Map as an input and combines those data tuples into a smaller set of tuples.

**Example –**(Reduce function in Word Count)

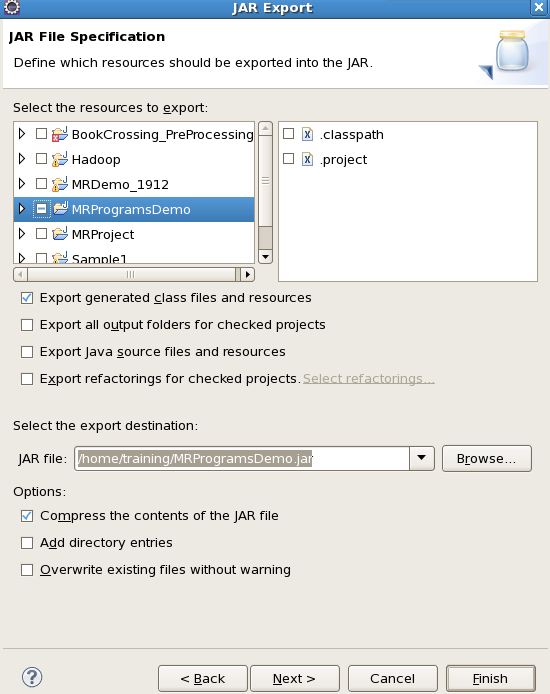
|  |  |  |
| --- | --- | --- |
| **Input**  **(output of Map function)** | Set of Tuples | (Bus,1), (Car,1), (bus,1), (car,1), (train,1),  (car,1), (bus,1), (car,1), (train,1), (bus,1),  (TRAIN,1),(BUS,1), (buS,1), (caR,1), (CAR,1),  (car,1), (BUS,1), (TRAIN,1) |
| **Output** | Converts into smaller set of tuples | (BUS,7),  (CAR,7),  (TRAIN,4) |

**Work Flow of the Program**



**Workflow of MapReduce consists of 5 steps:**

1. Splitting – The splitting parameter can be anything, e.g. splitting by space, comma, semicolon, or even by a new line (‘\n’).
2. Mapping – as explained above.
3. Intermediate splitting – the entire process in parallel on different clusters. In order to group them in “Reduce Phase” the similar KEY data should be on the same cluster.
4. Reduce – it is nothing but mostly group by phase.
5. Combining – The last phase where all the data (individual result set from each cluster) is combined together to form a result.
   * 1. Steps
6. Open Eclipse> File > New > Java Project >( Name it – MRProgramsDemo) > Finish.
7. Right Click > New > Package ( Name it - PackageDemo) > Finish.
8. Right Click on Package > New > Class (Name it - WordCount).
9. Add Following Reference Libraries:
   1. Right Click on Project > Build Path> Add External
      1. /usr/lib/hadoop-0.20/hadoop-core.jar
      2. Usr/lib/hadoop-0.20/lib/Commons-cli-1.2.jar



**Source Code:**

package PackageDemo;

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;

import org.apache.hadoop.util.GenericOptionsParser;

public class WordCount {

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

{

Configuration c=new Configuration();

String[] files=new GenericOptionsParser(c,args).getRemainingArgs();

Path input=new Path(files[0]);

Path output=new Path(files[1]);

Job j=new Job(c,"wordcount");

j.setJarByClass(WordCount.class);

j.setMapperClass(MapForWordCount.class);

j.setReducerClass(ReduceForWordCount.class);

j.setOutputKeyClass(Text.class);

j.setOutputValueClass(IntWritable.class);

FileInputFormat.addInputPath(j, input);

FileOutputFormat.setOutputPath(j, output);

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

}

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

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

{

String line = value.toString();

String[] words=line.split(",");

for(String word: words )

{

Text outputKey = new Text(word.toUpperCase().trim());

IntWritable outputValue = new IntWritable(1);

con.write(outputKey, outputValue);

}

}

}

public static class ReduceForWordCount extends Reducer<Text, IntWritable, Text, IntWritable>

{

public void reduce(Text word, Iterable<IntWritable> values, Context con) throws IOException, InterruptedException

{

int sum = 0;

for(IntWritable value : values)

{

sum += value.get();

}

con.write(word, new IntWritable(sum));

}

}

}

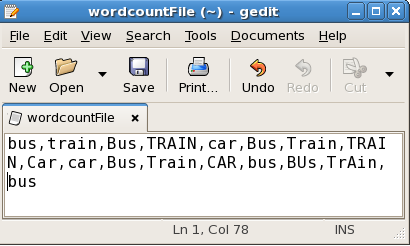
The above program consists of three classes:

* Driver class (Public, void, static, or main; this is the entry point).
* The Map class which **extends** the public class Mapper<KEYIN,VALUEIN,KEYOUT,VALUEOUT>  and implements the Map function.
* The Reduce class which extends the public class Reducer<KEYIN,VALUEIN,KEYOUT,VALUEOUT> and implements the Reduce function.

1. Make  a jar file

Right Click on Project> Export> Select export destination as **Jar File** > next> Finish.

1. Take a text file and move it into HDFS format:



move this into Hadoop directly, open the terminal and enter the following commands:

[training@localhost~]$hadoop fs -putwordcountFilewordCountFile

Run the jar file:

[training@localhost~]$hadoop jar MRProgramsDemo.jar PackageDemo.WordCountwordCountFile MRDir1

**Input / Output:**

[training@localhost~]$hadoop fs -ls MRDir1

Found 3 items

-rw-r--r-- 1 training supergroup 0 2016-02-23 03:36 /user/training/MRDir1/\_SUCCESS

drwxr-xr-x - training supergroup 0 2016-02-23 03:36 /user/training/MRDir1/\_logs

-rw-r--r-- 1 training supergroup 20 2016-02-23 03:36 /user/training/MRDir1/part-r-00000

[training@localhost~]$hadoop fs -cat MRDir1/part-r-00000

BUS 7

CAR 4

TRAIN 6