**LAB NO.: 4** **Date:**

# MAP REDUCE PROGRAMMING

**Objectives:**

1. To understand the basics of hadoop.
2. To learn basics of Map reduce programming.

## 1. Introduction to Hadoop

Apache Hadoop is an open source software platform for distributed storage and distributed processing of very large data sets on computer clusters built from commodity hardware. Hadoop services provide for data storage, data processing, data access, data governance, security, and operations.

*Benefits*

Some of the reasons organizations use Hadoop is its' ability to store, manage and analyze vast amounts of structured and unstructured data quickly, reliably, flexibly and at low-cost.

- **Scalability and Performance** – distributed processing of data local to each node in a cluster enables Hadoop to store, manage, process and analyze data at petabyte scale.
- **Reliability** – large computing clusters are prone to failure of individual nodes in the cluster. Hadoop is fundamentally resilient – when a node fails processing is re-directed to the remaining nodes in the cluster and data is automatically re-replicated in preparation for future node failures.
- **Flexibility** – unlike traditional relational database management systems, you don't have to created structured schemas before storing data. You can store data in any format, including semi-structured or unstructured formats, and then parse and apply schema to the data when read.
- **Low Cost** – unlike proprietary software, Hadoop is open source and runs on low-cost commodity hardware.

High level architecture of the Hadoop as shown in the Figure 1 High level architecture of Hadoop. There are two primary components at the core of Apache Hadoop 1.x: the Hadoop Distributed File System (HDFS) and the MapReduce parallel processing framework.

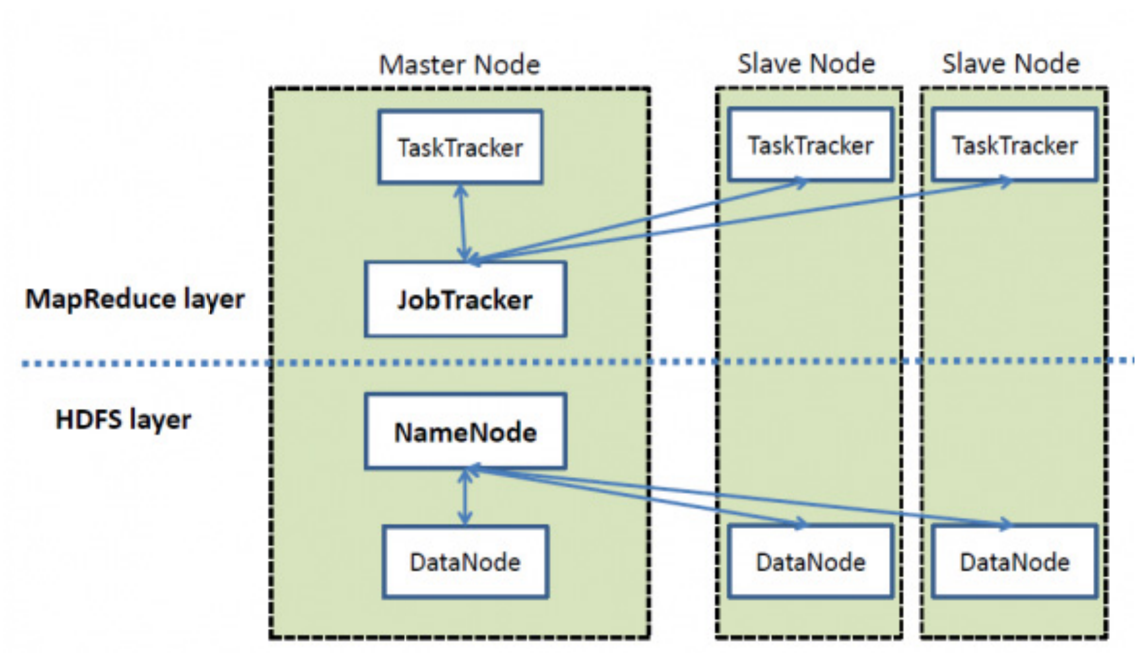

**Figure 1 High level architecture of Hadoop**

**Login to hduser : su –hduser**
**Start cluster: start-all.sh**

**HDFS Commands**

1. **Create a directory in HDFS at given path**
   hadoop fs -mkdir <paths>

2. **List the contents of a directory.**
   hadoop fs -ls <args>

3. **Copy a file from/To Local file system to HDFS**
   hadoop fs -copyFromLocal <localsrc> URI
   hadoop fs -copyToLocal [-ignorecrc] [-crc] URI <localdst>

4. **Move file from source to destination**
   hadoop fs -mv <src><dest>

5. **Remove a file or directory in HDFS.**
   Remove files specified as argument. Deletes directory only when it is empty
   hadoop fs -rm <arg>

   ***Recursive version of delete.***
   hadoop fs -rmr <arg>

6. **Display last few lines of a file.**
   hadoop fs -tail <path[filename]>
7. **Print the contents of the file on the terminal**
   hadoop fs -cat <path[filename]>

## 2. Introduction to Map Reduce

MapReduce is a framework designed for writing programs that process large volume of structured and unstructured data in parallel fashion across a cluster, in a reliable and fault-tolerant manner. MapReduce concept is simple to understand who are familiar with distributed processing framework.

MapReduce works with Key-Value pair. In the context of Hadoop, keys are associated with values. This data in MapReduce is stored in such a way that the values can be sorted and rearranged (Shuffle and sort wrt to MapReduce) across a set of keys. All data emitted in the flow of a MapReduce program is in the form of pairs. Some important features of key/value data will become apparent are:

1. Keys must be unique.
2. Each value must be associated with a key
3. A key can have no values also.

Key-value data is the foundation of MapReduce paradigm which means much of the data is in key-value nature or we can represent it in such a way. In short we can say that data model to be applied for designing MapReduce program is Key-Value pair.

It uses Divide and Conquer technique to process large amount of data. It divides input task into smaller and manageable sub-tasks (They should be executable independently) to execute them in-parallel.

MapReduce Algorithm uses the following three main steps:

1. Map Function
2. Shuffle Function
3. Reduce Function

Map Function is the first step in MapReduce Algorithm. It takes input tasks and divides them into smaller sub-tasks. Then perform required computation on each sub-task in parallel.

This step performs the following two sub-steps:

1. Splitting
2. Mapping
   - Splitting step takes input Data Set from Source and divide into smaller Sub-Data Sets.
   - Mapping step takes those smaller Sub-Data Sets and perform required action or computation on each Sub-Data Set.

The output of this Map Function is a set of key and value pairs as <Key, Value> as shown in the Figure 2 Key value pair generation.

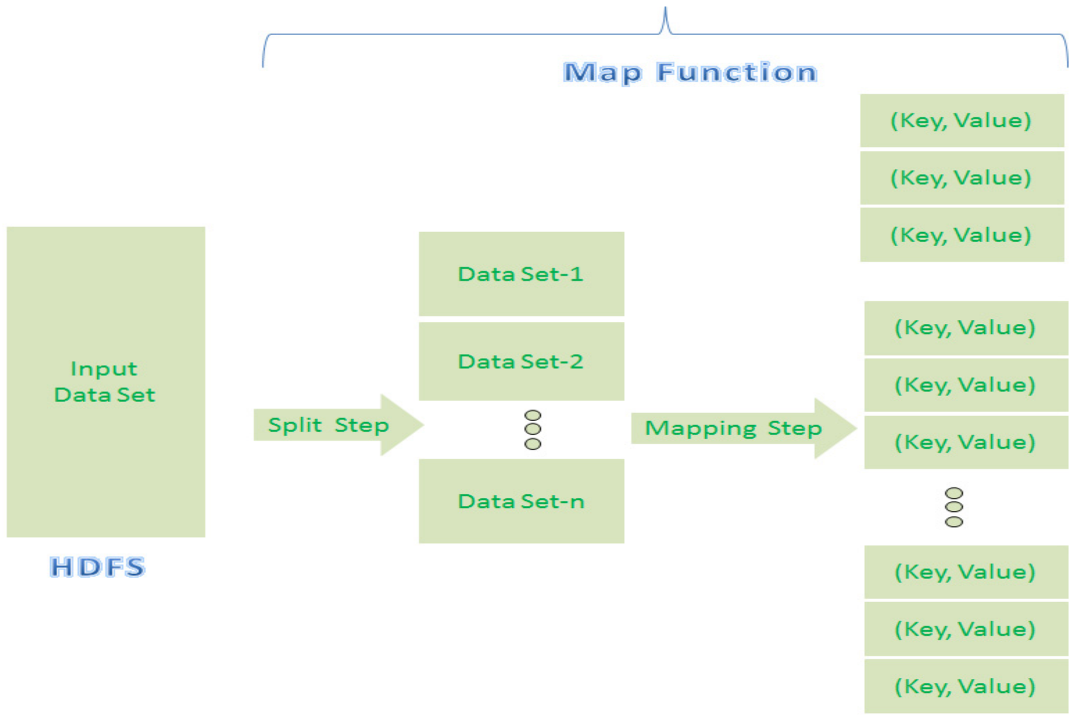

**Figure 2 Key value pair generation**

**Shuffle Function**

It is the second step in MapReduce Algorithm. Shuffle Function is also known as "Combine Function".

It performs the following two sub-steps:

1. Merging
2. Sorting

It takes a list of outputs coming from "Map Function" and perform these two sub-steps on each and every key-value pair.

- Merging step combines all key-value pairs which have same keys (that is grouping key-value pairs by comparing "Key"). This step returns <Key, List<Value>>.
- Sorting step takes input from Merging step and sort all key-value pairs by using Keys. This step also returns <Key, List<Value>> output but with sorted key-value pairs.

Finally, Shuffle Function returns a list of <Key, List<Value>> sorted pairs to next step. Generation key, list<value> is shown in Figure 3 Shuffle function.

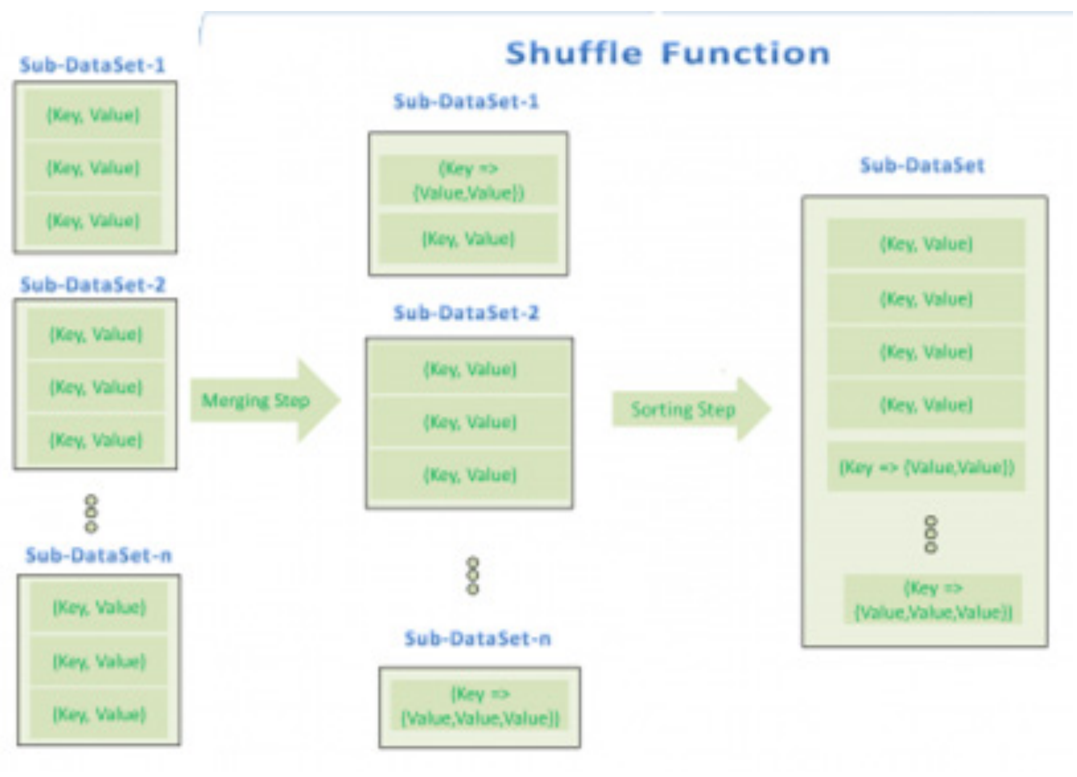

**Figure 3 Shuffle function**

**MapReduce Second Step Output:**

Shuffle Function Output = List of <Key, List<Value>> Pairs

**Reduce Function**

It is the final step in MapReduce Algorithm. It performs only one step : Reduce step.

It takes list of <Key, List<Value>> sorted pairs from Shuffle Function and perform reduce operation as shown in Figure 4 Reduce function

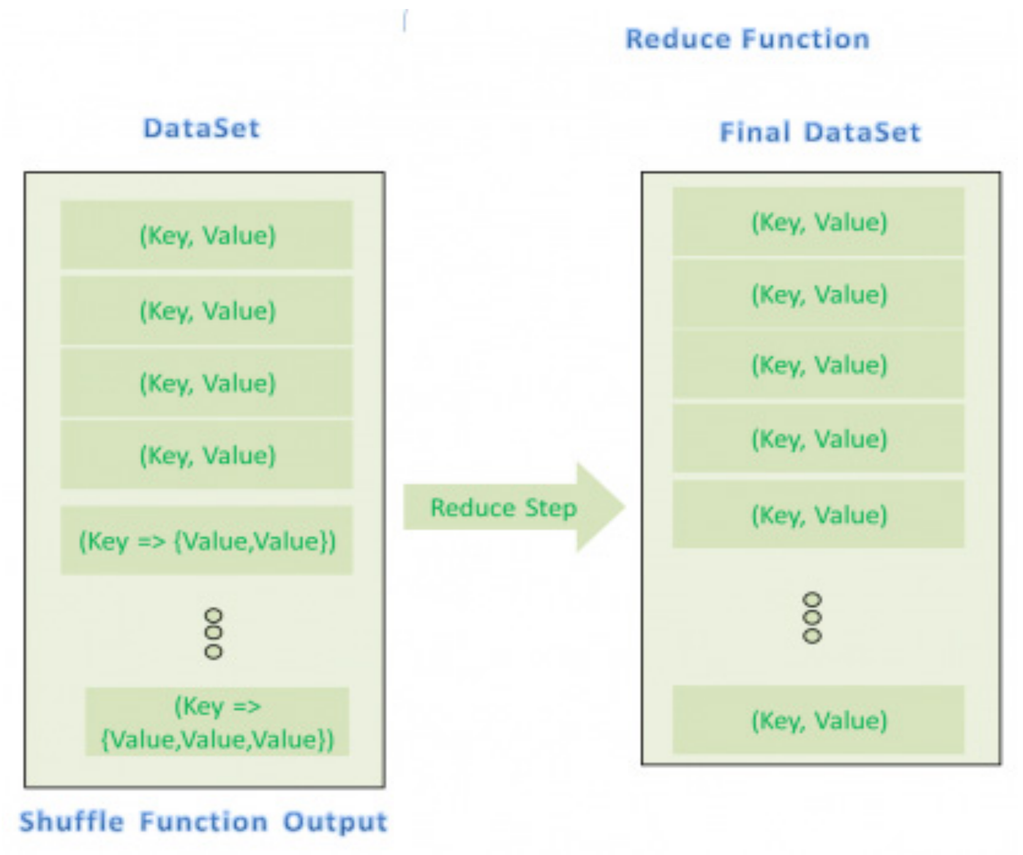

**Figure 4 Reduce function**

**MapReduce Final Step Output:**

Reduce Function Output = List of <Key, Value> Pairs

Final step output looks like first step output. However final step <Key, Value> pairs are different than first step <Key, Value> pairs. Final step <Key, Value> pairs are computed and sorted pairs.

**Solved Exercise:**

- Count the number of occurrences of each word available in a Data Set.

Workflow of MapReduce for word count consists of 5 steps which is shown in Figure 5.

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 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 combine together to form a Result

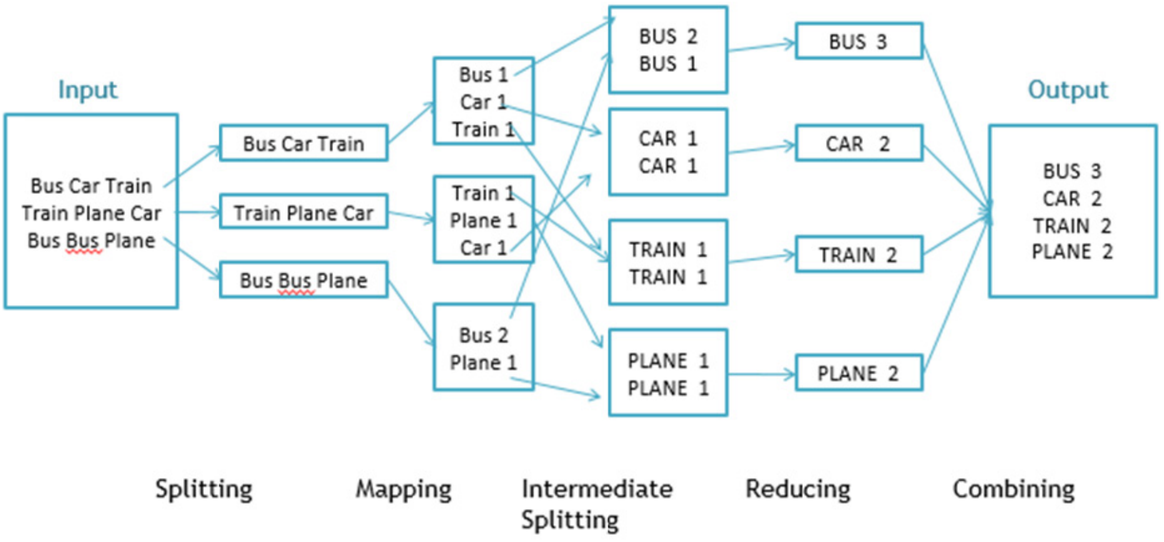

**Figure 5 Work Flow Diagram for Word Count**

**Steps**

**Step 1.** Open Eclipse> File > New > Java Project >( Name it – MRProgramsDemo) > Finish

**Step 2.** Right Click > New > Package ( Name it - PackageDemo) > Finish

**Step 3**. Right Click on Package > New > Class (Name it - WordCount)

**Step 4.** Add Following Reference Libraries –

Right Click on Project > Build Path> Add External Libraries

- */usr/lib/hadoop-0.20/***hadoop-core.jar**
- *Usr/lib/hadoop-0.20/lib/***Commons-cli-1.2.jar**

**Step 5. Type following Program :**

```
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));
}
}
}
```

*Explanation*

The program consist of 3 classes:

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

**Step 6. Make Jar File**

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

✓ Working on Hadoop Cluster:

```
admin@ubuntu$ su – hduser
hduser@ubuntu:/usr/local/hadoop$start-all.sh
```

hduser@ubuntu:/usr/local/hadoop$jps :

All the nodes should be up and running

```
2287 TaskTracker
2149 JobTracker
1938 DataNode
2085 SecondaryNameNode
2349 Jps
1788 NameNode
```

**Step 7: Copy file and move it in HDFS**

```
bin/hadoop dfs -copyFromLocal /path/input_data input_directory_in_hdfs
```

**Step 8 : Run Jar file**

*(hadoop jar jarfilename.jar packageName.ClassName PathToInputTextFile PathToOutputDirectry)*

```
hadoop jar MRProgramsDemo.jar PackageDemo.WordCount wordCountFile MRDir1
```

**Step 9: Open Result**

```
hadoop fs -ls MRDir1

hadoop fs -cat MRDir1/part-r-00000
```

**or**

**localhost:50070 -> Utilities->Browse file system**

**Lab Exercises**

1. Modify the word count program to display the document_id word count. Consider each sentence as one document.
2. Consider Movie Lens data, the objective is to find the movies falling more than two genres. The solution is a single Map-Reduce job.
3. Given a set of webserver logs for a site, the objective here is to find the hourly web traffic to particular site. The solution is a single Map-Reduce job. The mapper reads the logline and uses a regular expression to extract the date format from the log, parses it into a Date object using SimpleDateFormat, and extracts the hour from it. The mapper emits the (HOUR, 1) pair to the reducer, which sorts them.