

## EXPERIMENT 3

**AIM:** Write a map reduce program to count words in a text file.

### THEORY:

MapReduce is a processing technique and a program model for distributed computing based on java. The MapReduce algorithm contains two important tasks, namely Map and Reduce. Map takes a set of data and converts it into another set of data, where individual elements are broken down into tuples (key/value pairs). Secondly, reduce task, which takes the output from a map as an input and combines those data tuples into a smaller set of tuples. As the sequence of the name MapReduce implies, the reduce task is always performed after the map job.

The major advantage of MapReduce is that it is easy to scale data processing over multiple computing nodes. Under the MapReduce model, the data processing primitives are called mappers and reducers. Decomposing a data processing application into mappers and reducers is sometimes nontrivial. But, once we write an application in the MapReduce form, scaling the application to run over hundreds, thousands, or even tens of thousands of machines in a cluster is merely a configuration change. This simple scalability is what has attracted many programmers to use the MapReduce model.

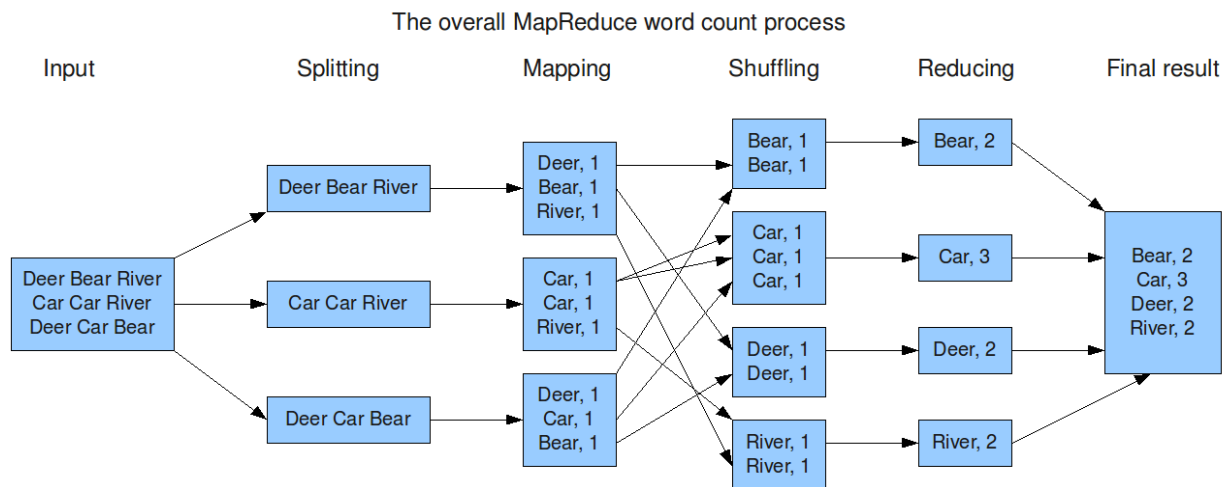
### Usage:

- It can be used in various applications like document clustering, distributed sorting, and web link-graph reversal.
- It can be used for distributed pattern-based searching.
- We can also use MapReduce in machine learning.
- It was used by Google to regenerate Google's index of the World Wide Web.
- It can be used in multiple computing environments such as multi-cluster, multi-core, and mobile environments.

**Stages:** MapReduce works in 2 stages:

1. Map stage – The map or mapper's job is to process the input data. Generally, the input data is in the form of a file or directory and is stored in the Hadoop file system (HDFS). The input file is passed to the mapper function line by line. The mapper processes the data and creates several small chunks of data.

2. Reduce stage – This stage is the combination of the Shuffle stage and the Reduce stage. The Reducer's job is to process the data that comes from the mapper. After processing, it produces a new set of output, which will be stored in the HDFS.



## CODE:

### mapper.java

```

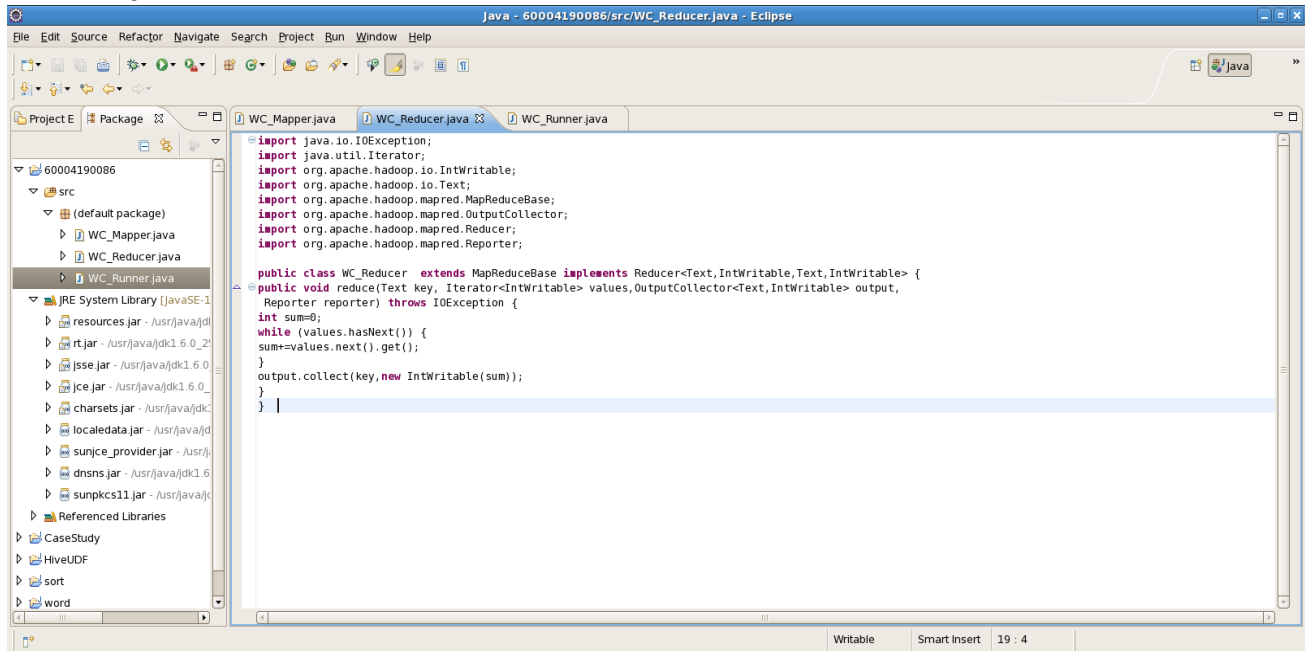
Java - 60004190086/src/WC_Mapper.java - Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help

WC_Mapper.java WC_Reducer.java WC_Runner.java

import java.io.IOException;
import java.util.StringTokenizer;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapred.MapReduceBase;
import org.apache.hadoop.mapred.Mapper;
import org.apache.hadoop.mapred.OutputCollector;
import org.apache.hadoop.mapred.Reporter;

public class WC_Mapper extends MapReduceBase implements Mapper<LongWritable,Text,Text,IntWritable>{
    private final static IntWritable one = new IntWritable(1);
    private Text word = new Text();
    public void map(LongWritable key, Text value, OutputCollector<Text,IntWritable> output,
        Reporter reporter) throws IOException{
        String line = value.toString();
        StringTokenizer tokenizer = new StringTokenizer(line);
        while (tokenizer.hasMoreTokens()){
            word.set(tokenizer.nextToken());
            output.collect(word, one);
        }
    }
}
  
```

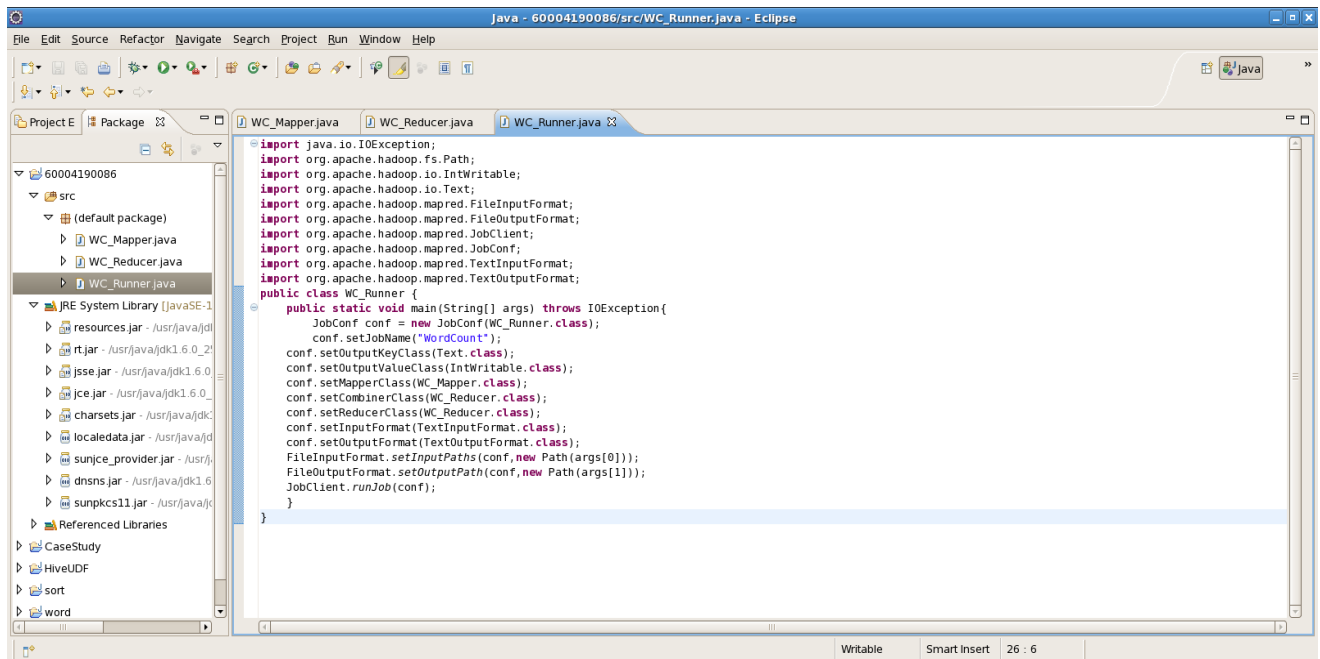
## reducer.java



```
Java - 60004190086/src/WC_Reducer.java - Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help
Project E Package
60004190086
src
  (default package)
    WC_Mapper.java
    WC_Reducer.java
    WC_Runner.java
JRE System Library [javaSE-1
resources.jar - /usr/java/jd
rt.jar - /usr/java/jdk1.6.0_2
jsse.jar - /usr/java/jdk1.6.0
jce.jar - /usr/java/jdk1.6.0
charsets.jar - /usr/java/jdk
localedata.jar - /usr/java/d
sunjce_provider.jar - /usr/f
dnsns.jar - /usr/java/jdk1.6
sunpkcs11.jar - /usr/java/jc
Referenced Libraries
CaseStudy
HiveUDF
sort
word
WC_Mapper.java WC_Reducer.java WC_Runner.java
import java.io.IOException;
import java.util.Iterator;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapred.MapReduceBase;
import org.apache.hadoop.mapred.OutputCollector;
import org.apache.hadoop.mapred.Reducer;
import org.apache.hadoop.mapred.Reporter;

public class WC_Reducer extends MapReduceBase implements Reducer<Text,IntWritable,Text,IntWritable> {
    public void reduce(Text key, Iterator<IntWritable> values, OutputCollector<Text,IntWritable> output,
        Reporter reporter) throws IOException {
        int sum=0;
        while (values.hasNext()) {
            sum+=values.next().get();
        }
        output.collect(key,new IntWritable(sum));
    }
}
```

## driver code



```
Java - 60004190086/src/WC_Runner.java - Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help
Project E Package
60004190086
src
  (default package)
    WC_Mapper.java
    WC_Reducer.java
    WC_Runner.java
JRE System Library [javaSE-1
resources.jar - /usr/java/jd
rt.jar - /usr/java/jdk1.6.0_2
jsse.jar - /usr/java/jdk1.6.0
jce.jar - /usr/java/jdk1.6.0
charsets.jar - /usr/java/jdk
localedata.jar - /usr/java/d
sunjce_provider.jar - /usr/f
dnsns.jar - /usr/java/jdk1.6
sunpkcs11.jar - /usr/java/jc
Referenced Libraries
CaseStudy
HiveUDF
sort
word
WC_Mapper.java WC_Reducer.java WC_Runner.java
import java.io.IOException;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapred.FileInputFormat;
import org.apache.hadoop.mapred.FileOutputFormat;
import org.apache.hadoop.mapred.JobClient;
import org.apache.hadoop.mapred.JobConf;
import org.apache.hadoop.mapred.TextInputFormat;
import org.apache.hadoop.mapred.TextOutputFormat;

public class WC_Runner {
    public static void main(String[] args) throws IOException {
        JobConf conf = new JobConf(WC_Runner.class);
        conf.setJobName("WordCount");
        conf.setOutputKeyClass(Text.class);
        conf.setOutputValueClass(IntWritable.class);
        conf.setMapperClass(WC_Mapper.class);
        conf.setCombinerClass(WC_Reducer.class);
        conf.setReducerClass(WC_Reducer.class);
        conf.setInputFormat(TextInputFormat.class);
        conf.setOutputFormat(TextOutputFormat.class);
        FileInputFormat.setInputPaths(conf,new Path(args[0]));
        FileOutputFormat.setOutputPath(conf,new Path(args[1]));
        JobClient.runJob(conf);
    }
}
```

## Text file

```
training@localhost: ~/Desktop/exp3
File Edit View Terminal Tabs Help
[training@localhost exp3]$ hadoop fs -cat /mapreduce
This is a good example to learn mapreduce in the Hadoop ecosystem in
BDI. HDFS in Hadoop is the file system used by Hadoop and is built
similar to GFS.
[training@localhost exp3]$
```

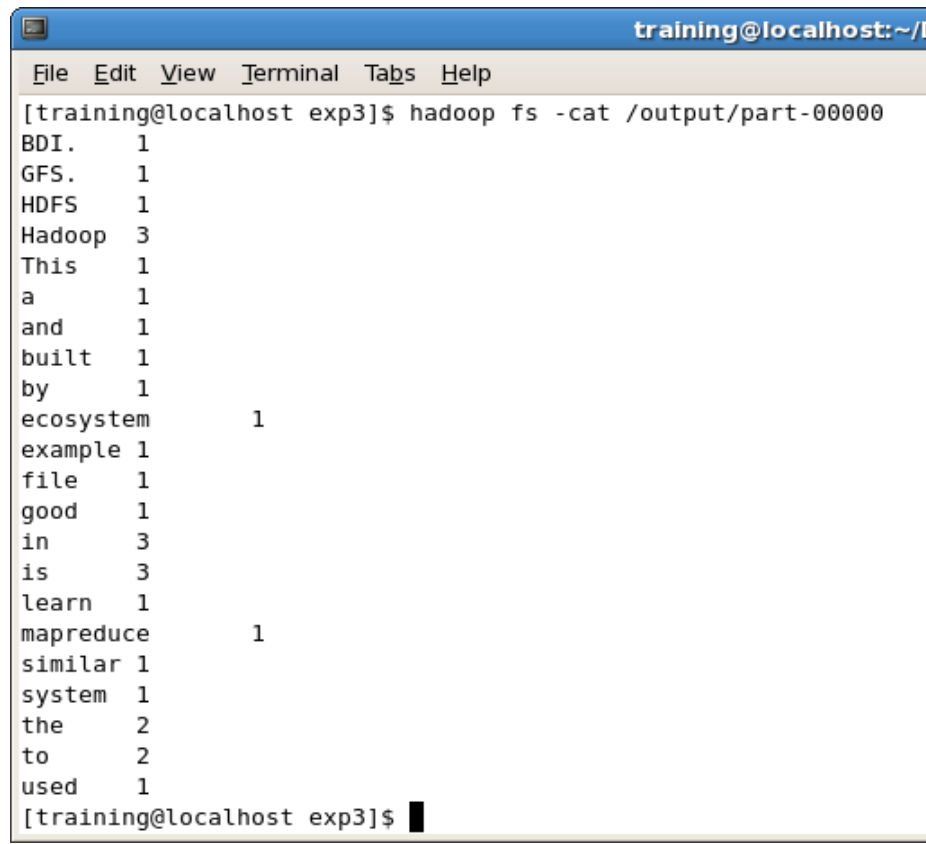
## Execute mapReduce

```
training@localhost: ~/Desktop/exp3
File Edit View Terminal Tabs Help
[training@localhost exp3]$ hadoop fs -ls mapreduce/
Found 2 items
-rw-r--r-- 1 training supergroup 3804 2022-04-29 02:12 /user/training/mapreduce/countword.jar
-rw-r--r-- 1 training supergroup 154 2022-04-29 03:19 /user/training/mapreduce/data.txt
```

## Store output

```
training@localhost: ~/Desktop/exp3
File Edit View Terminal Tabs Help
[training@localhost exp3]$ hadoop jar ~/Desktop/countword.jar /user/training/mapreduce/data.txt /output
22/04/29 03:20:22 WARN mapred.JobClient: Use GenericOptionsParser for parsing the arguments. Applications should implement
22/04/29 03:20:22 WARN snappy.LoadSnappy: Snappy native library is available
22/04/29 03:20:22 INFO util.NativeCodeLoader: Loaded the native-hadoop library
22/04/29 03:20:22 INFO snappy.LoadSnappy: Snappy native library loaded
22/04/29 03:20:22 INFO mapred.FileInputFormat: Total input paths to process : 1
22/04/29 03:20:24 INFO mapred.JobClient: Running job: job_202204262107_0008
22/04/29 03:20:25 INFO mapred.JobClient: map 0% reduce 0%
22/04/29 03:21:12 INFO mapred.JobClient: map 100% reduce 0%
22/04/29 03:21:46 INFO mapred.JobClient: map 100% reduce 100%
22/04/29 03:21:47 INFO mapred.JobClient: Job complete: job_202204262107_0008
22/04/29 03:21:47 INFO mapred.JobClient: Counters: 23
22/04/29 03:21:47 INFO mapred.JobClient: Job Counters
22/04/29 03:21:47 INFO mapred.JobClient: Launched reduce tasks=1
22/04/29 03:21:47 INFO mapred.JobClient: SLOTS_MILLIS_MAPS=70777
22/04/29 03:21:47 INFO mapred.JobClient: Total time spent by all reduces waiting after reserving slots (ms)=0
22/04/29 03:21:47 INFO mapred.JobClient: Total time spent by all maps waiting after reserving slots (ms)=0
22/04/29 03:21:47 INFO mapred.JobClient: Launched map tasks=2
22/04/29 03:21:47 INFO mapred.JobClient: Data-local map tasks=2
22/04/29 03:21:47 INFO mapred.JobClient: SLOTS_MILLIS_REDUCES=32845
22/04/29 03:21:47 INFO mapred.JobClient: FileSystemCounters
22/04/29 03:21:47 INFO mapred.JobClient: FILE_BYTES_READ=266
22/04/29 03:21:47 INFO mapred.JobClient: HDFS_BYTES_READ=436
22/04/29 03:21:47 INFO mapred.JobClient: FILE_BYTES_WRITTEN=165237
22/04/29 03:21:47 INFO mapred.JobClient: HDFS_BYTES_WRITTEN=163
22/04/29 03:21:47 INFO mapred.JobClient: Map-Reduce Framework
22/04/29 03:21:47 INFO mapred.JobClient: Reduce input groups=22
22/04/29 03:21:47 INFO mapred.JobClient: Combine output records=23
22/04/29 03:21:47 INFO mapred.JobClient: Map input records=3
22/04/29 03:21:47 INFO mapred.JobClient: Reduce shuffle bytes=232
22/04/29 03:21:47 INFO mapred.JobClient: Reduce output records=22
22/04/29 03:21:47 INFO mapred.JobClient: Spilled Records=46
22/04/29 03:21:47 INFO mapred.JobClient: Map output bytes=272
22/04/29 03:21:47 INFO mapred.JobClient: Map input bytes=154
22/04/29 03:21:47 INFO mapred.JobClient: Combine input records=30
22/04/29 03:21:47 INFO mapred.JobClient: Map output records=30
22/04/29 03:21:47 INFO mapred.JobClient: SPLIT_RAW_BYTES=204
22/04/29 03:21:47 INFO mapred.JobClient: Reduce input records=23
```

## OUTPUT:



```
training@localhost:~/D
File Edit View Terminal Tabs Help
[training@localhost exp3]$ hadoop fs -cat /output/part-00000
BDI.      1
GFS.      1
HDFS      1
Hadoop    3
This      1
a         1
and       1
built     1
by        1
ecosystem      1
example 1
file      1
good      1
in        3
is        3
learn     1
mapreduce      1
similar 1
system    1
the       2
to        2
used      1
[training@localhost exp3]$
```

## CONCLUSION:

In this experiment, I have implemented MapReduce program in Hadoop for getting the word count of the data in the text file. I first coded the mapper and the reducer stages and then handled both the functions through the driver function. The mapper converts the given data into key-value pair and then the reducer shuffles and re-arranges the data from the mapper and reduces it according to the frequency of the words. Finally, we get the word count of the given data which is dumped into the output folder.