NAME:- DUBEY KARAN SANJEEV
CLASS:- B.E - 4
ROLL NO:- 04
BATCH:- A

EXPERIMENT NO. 4

AIM:-Write a program to implement word count using MapReduce.

THEORY:

Mapper maps input key/value pairs to a set of intermediate key/value pairs. Maps are the individual tasks that transform input records into intermediate records. The transformed intermediate records do not need to be of the same type as the input records. A given input pair may map to zero or many output pairs. The Hadoop Map Reduce framework spawns one map task for each InputSplit generated by the InputFormat for the job. Overall, Mapper implementations are passed the Job for the job via the Job.setMapperClass(Class) method. The framework then calls map(WritableComparable, Writable, Context) for each key/value pair in the InputSplit for that task. Applications can then override the cleanup(Context) method to perform any required cleanup.Output pairs do not need to be of the same types as input pairs. A given input pair may map to zero or many output pairs. Output pairs are collected with calls to context.write(WritableComparable, Writable).

Applications can use the Counter to report its statistics. All intermediate values associated with a given output key are subsequently grouped by the framework, and passed to the Reducer(s) to determine the final output. Users can control the grouping by specifying a Comparator via Job.setGroupingComparatorClass(Class).

The Mapper outputs are sorted and then partitioned per Reducer. The total number of partitions is the same as the number of reduce tasks for the job. Users can control which keys (and hence records) go to which Reducer by implementing a custom Partitioner.

Users can optionally specify a combiner, via Job.setCombinerClass(Class), to perform local aggregation of the intermediate outputs, which helps to cut down the amount of data transferred from the Mapper to the Reducer.

Reducer reduces a set of intermediate values which share a key to a smaller set of values.The number of reduces for the job is set by the user via Job.setNumReduceTasks(int). Overall,Reducer implementations are passed the Job for the job via the Job.setReducerClass(Class) method and can override it to initialize themselves. The framework then calls reduce(WritableComparable, Iterable<Writable>, Context) method for each <key, (list of values)> pair in the grouped inputs. Applications can then override the cleanup(Context) method to perform any required cleanup.

Reducer has 3 primary phases: shuffle, sort and reduce.

Shuffle
Reducer reduces a set of intermediate values which share a key to a smaller set of values.The number of Input to the Reducer is the sorted output of the mappers. In this phase the framework fetches the relevant

partition of the output of all the mappers, via HTTP.

Sort

The framework groups Reducer inputs by keys (since different mappers may have output the same key) in this stage.

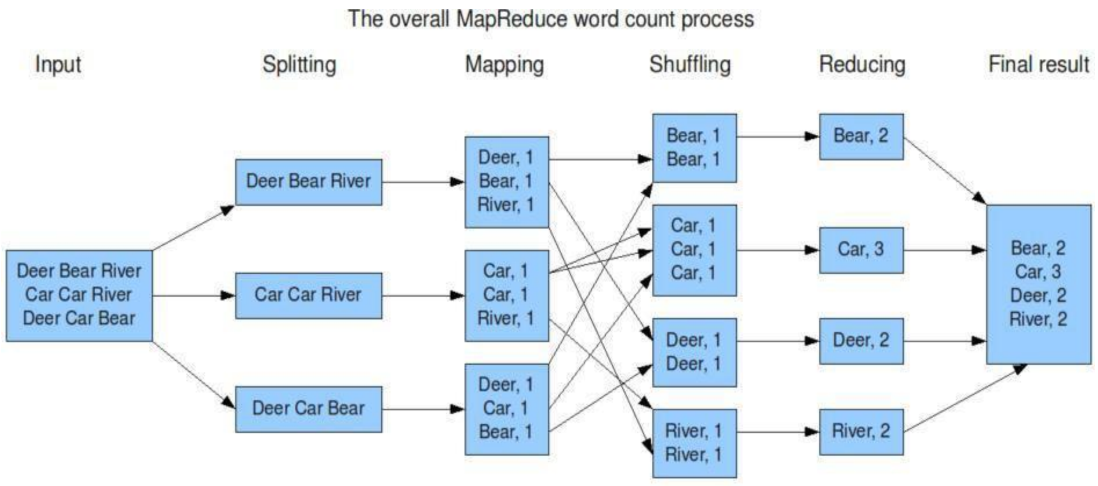
The shuffle and sort phases occur simultaneously; while map-outputs are being fetched they are merged.

Reduce

In this phase the reduce(WritableComparable, Iterable<Writable>, Context) method is called for each <key, (list of values)> pair in the grouped inputs.

The output of the reduce task is typically written to the FileSystem via Context.write(WritableComparable, Writable).

Applications can use the Counter to report its statistics. The output of the Reducer is *not sorted*.



CONCLUSION:

In this experiment we have successfully studied and understood the flow of word count program and implemented it using map reduce functions.

Program formation/ Execution/ ethical practices (06)	**Timely Submission and Documentation (02)**	**Viva Answer (02)**	**Experiment Marks (10)**	**Teacher Signature with date**

```
package wc;


import java.io.IOException;
import java.util.Iterator;
import java.util.StringTokenizer;

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.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.MapReduceBase;
import org.apache.hadoop.mapred.Mapper;
import org.apache.hadoop.mapred.OutputCollector;
import org.apache.hadoop.mapred.Reducer;
import org.apache.hadoop.mapred.Reporter;
import org.apache.hadoop.mapred.TextInputFormat;
import org.apache.hadoop.mapred.TextOutputFormat;

public class wordcount1
{

public static class Map 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);
}
}
}

public static class Reduce 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));
}
}

public static void main(String[] args) throws Exception
{
JobConf conf = newJobConf(wordcount1.class); conf.setJobName("wordcount");

conf.setOutputKeyClass(Text.class); conf.setOutputValueClass(IntWritable.class);

conf.setMapperClass(Map.class);
conf.setCombinerClass(Reduce.class);
conf.setReducerClass(Reduce.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);
}
}
```

Input file: test.txt in HDFS

```
This is first lab
This is first program
```

Output file: part-00000 in HDFS

```
first,2
is,2
lab,1
program,1
This,2
```