BDE-LABS

Steps to execute map reduce program

1. Open virtual box
2. Create a text file with some data and name it as file1.txt
3. Copy the text file from local file system to HDFS

   Process:

   1. Open the terminal
   2. Give the command startCDH.sh(to start clouera distributed hadoop)
   3. Type the command

      hadoop fs –put file from local file system path file on to HDFS path

example:

hadoop fs –put file1.txt dc/a.txt

dc is the folder on HDFS

How to create the folder???

hadoop fs –mkdir dc

4. Open eclipse, copy the code by creating 3 java files(runner,mapper,reducer)
5. Right Click on folder goto->build path->configure build path->click on liberary-> add external jar->hadoop core – 1.2.1 After completion of the program , make a jar file ow to make a jar file???

Go to the folder where u have written ur code right click on the folder symbol
Folder→right click-> go to export-→go to java→select jar→give some name and click on ok

6. Copy the jar file on to the eclipse workspace →project folder→bin

4. Go to the folder bin copy the path.
5. Open the terminal
   Give the command
   cd copied path
6. To run the program type the command
   hadoop jar created jar name Mainclassname HDFS filename output directory name

   Example:
   hadoop jar sum.jar sum_runner dc/file1.txt out

   The map reduce program starts executing …….

1. Practice basic linux commands like file creation, modification, copying, deletion, etc,.

<u>Create:</u>

1. Make a text file on Linux:

```
$ cat > filename.txt
```

Add data and press CTRL+D to save the filename.txt when using cat on Linux.

2. Simply type any one of the following command:

```
$ > data.txt
```

OR

```
$ touch test.txt
```

Verify that empty files are created with the help of ls command:

```
$ ls -l data.txt test.txt
```

<u>Modify:</u>

To edit the text file, execute the below command to open with Vi editor:

```
$ vi filename.txt
```

- Press the ESC key for normal mode.
- Press i Key for insert mode.
- Press :q! keys to exit from the editor without saving a file.
- Press :wq! Keys to save the updated file and exit from the editor.
- Press :w test.txt to save the file as test.txt

<u>Copy:</u>

To copy a file in Linux, just use the cp command followed by the name of the source file and then the new file. For example:

```
$ cp SampleText.txt SampleText_2.txt
```

<u>Delete:</u>

To delete a single file, use the rm or unlink command followed by the file name:

```
$ unlink filename
```

```
$ rm filename
```

To delete multiple files at once, use the rm command followed by the file names separated by space.

```
$ rm filename1 filename2 filename3
```

2. Implement a Hadoop program to count words in a given file using Map Reduce – Word Count.

Writing space of the Problem:(Students use)

WC_Mapper.java

Input:

car, Bar, car, rear, bear, car,

Bar, car, car, Bar, bear, rear

Output:

```
car: 5

Bar: 3

rear: 2

bear: 2
```

```
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);
        }
    }

}
```

WC_Reducer.java

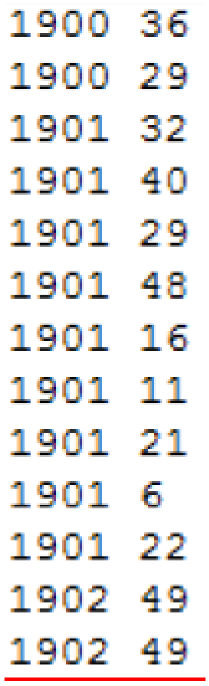
```
import java.io.IOException;
    import java.util.Iterator;
    import org.apache.hadoop.io.IntWritable;
    import org.apache.hadoop.io.Text;
    import org.apache.hadoop.mapred.MapReduceBase;
    import org.apache.hadoop.mapred.OutputCollector;
    import org.apache.hadoop.mapred.Reducer;
    import 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));
    }
    }
```

WC_Runner .java

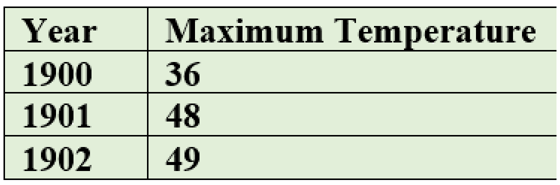
```
import java.io.IOException;
    import org.apache.hadoop.fs.Path;
    import org.apache.hadoop.io.IntWritable;
    import org.apache.hadoop.io.Text;
    import org.apache.hadoop.mapred.FileInputFormat;
    import org.apache.hadoop.mapred.FileOutputFormat;
    import org.apache.hadoop.mapred.JobClient;
    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);
       }
}
```

3. process a dataset with multiple temperatures for a year



```
1900 36
1900 29
1901 32
1901 40
1901 29
1901 48
1901 16
1901 11
1901 21
1901 6
1901 22
1902 49
1902 49
```

Output:



Year	Maximum Temperature
1900	36
1901	48
1902	49

TempMap.java

```
import java.io.IOException;

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 TempMap extends MapReduceBase implements
Mapper<LongWritable,Text,Text,IntWritable>{

        public void map(LongWritable key,Text value,OutputCollector<Text,IntWritable>
output,Reporter reporter) throws IOException {

                String record=value.toString();

                String[] parts=record.split(",");

                output.collect(new Text(parts[0]),new IntWritable(Integer.parseInt(parts[1])));

        }

}
```

TempReduce.java

```
package Tempdemo;

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 TempReduce 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 maxValue=0;

                while(values.hasNext()) {

                        maxValue=Math.max(maxValue,values.next().get());


                }

                output.collect(key,new IntWritable(maxValue));


        }

}
```

TempMR2.java

```
package Tempdemo;
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 TempMR2 {
	public static void main(String[] args) throws Exception {
		JobConf job=new JobConf(TempMR2.class);
		job.setMapOutputKeyClass(Text.class);
		job.setMapOutputValueClass(IntWritable.class);
		job.setOutputKeyClass(Text.class);
		job.setOutputValueClass(IntWritable.class);
		job.setMapperClass(TempMap.class);
		job.setReducerClass(TempReduce.class);
		job.setInputFormat(TextInputFormat.class);
		job.setOutputFormat(TextOutputFormat.class);
		FileInputFormat.addInputPath(job,new Path(args[0]));
		FileOutputFormat.setOutputPath(job,new Path(args[1]));
		JobClient.runJob(job);
			}
}
```

4. Implement Partitioner usage in Hadoop Map Reduce program.

You need to calculate the size of each word and count the number of words of that size in the text file using Partitioner.

Writing space of the Problem:(Students use)

Intellipaat_emp.java

```
package Intellipaat_emp;

import java.io.*;

import org.apache.hadoop.io.*;

import org.apache.hadoop.mapreduce.*;

import org.apache.hadoop.conf.*;

import org.apache.hadoop.fs.*;

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

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

import org.apache.hadoop.util.*;

public class Intellipaat_emp extends Configured implements Tool

{

 //Map class

 public static class MapClass extends Mapper<LongWritable,Text,Text,Text>

 {

 public void map(LongWritable key, Text value, Context context)

 {

 try{

 String[] str=value.toString().split("\t", -3);

 String gender=str[3];

 context.write(new Text(gender),new Text(value));

 }

 catch(Exception e)

 {

 System.out.println(e.getMessage());

 }
```

```
}
}
//Reducer class
public static class ReduceClass extends Reducer<Text,Text,Text,IntWritable>
{
public int max=-1;
public void reduce(Text key, Iterable <Text> values, Context context) throws IOException,
InterruptedException
{
max=-1;
for (Text val:values)
{
String[] str=val.toString().split("\t",-3);
if(Integer.parseInt(str[4])>max)
max=Integer.parseInt(str[4]);
}
context.write(new Text(key),new IntWritable(max));
}
}
//Intellipaat_emp class
public static class CaderPartitioner extends
Partitioner <Text,Text>
{
public int getPartition(Text key,Text value,int numReduceTasks)
{
String[] str=value.toString().split("\t");
int age=Integer.parseInt(str[2]);
if(numReduceTasks==0)
{
return 0;
}
```

```
 if(age<=20)
 {
 return 0;
 }
 else if(age>20 && age<=30)
 {
 return 1%numReduceTasks;
 }
 else
 {
 return 2%numReduceTasks;
 }
 }
 }
public int run(String[] arg) throws Exception
 {
 Configuration conf=getConf();
 Job job=new Job(conf, "topsal");
 job.setJarByClass(Intellipaat_emp.class);
 FileInputFormat.setInputPaths(job, new Path(arg[0]));
 FileOutputFormat.setOutputPath(job,new Path(arg[1]));
 job.setMapperClass(MapClass.class);
 job.setMapOutputKeyClass(Text.class);
 job.setMapOutputValueClass(Text.class);
 //set partitioner statement
 job.setPartitionerClass(CaderPartitioner.class);
 job.setReducerClass(ReduceClass.class);
 job.setNumReduceTasks(3);
 job.setInputFormatClass(TextInputFormat.class);
 job.setOutputFormatClass(TextOutputFormat.class);
 job.setOutputKeyClass(Text.class);
 job.setOutputValueClass(Text.class);
 System.exit(job.waitForCompletion(true)? 0 : 1);
 return 0;
 }
 public static void main(String ar[]) throws Exception
 {
 int res=ToolRunner.run(new Configuration(),new Intellipaat_emp(),ar);
System.exit(0);
 }
}
```

Input:

```
6001    aaaaa   45      Male    50000
6002    bbbbb   40      Female  50000
6003    ccccc   34      Male    30000
6004    ddddd   30      Male    30000
6005    eeeee   20      Male    40000
6006    fffff   25      Female  35000
6007    ggggg   20      Female  15000
6008    hhhhh   19      Female  15000
6009    iiiii   22      Male    22000
6010    jjjjj   24      Male    25000
6011    kkkk    25      Male    25000
6012    hhhh    28      Male    20000
6013    tttt    18      Female  8000
```

Output:

Output in Part-r-00000

```
Female  15000
Male    40000
```

Output in Part-r-00001

```
Female  35000
Male    30000
```

Output in Part-r-00002

```
Female  50000
Male    50000
```

5. Calculate the average salary in the department.

AverageSalary.java

```
import java.io.IOException;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.FloatWritable;
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;

public class AverageSalary {
public static class AvgMapper extends Mapper<Object, Text, Text, FloatWritable> {
        private Text dept_id=new Text();
        private FloatWritable salary=new FloatWritable();
 public void map(Object key, Text value, Context context) throws IOException, InterruptedException {
  String values[] = value.toString().split("\t");
  dept_id.set(values[0]);
  salary.set(Float.parseFloat(values[1]));
  context.write(dept_id,salary);
 }
}

public static class avgReducer extends Reducer<Text, FloatWritable, Text, FloatWritable> {
        private FloatWritable result=new FloatWritable();
 public void reduce(Text key, Iterable<FloatWritable> values, Context context) throws IOException, InterruptedException {
         float sum=0;
```

```
            float count=0;
   for (FloatWritable val : values) {
   sum+= val.get();
     count++;
  }
   result.set(sum/count);
   context.write(key,result);
 }
}

public static void main(String[] args) throws Exception {
 Configuration conf = new Configuration();
  Job job=new Job(conf,"averagesal");
  job.setJarByClass(AverageSalary.class);
  job.setMapperClass(AvgMapper.class);
  job.setCombinerClass(avgReducer.class);
  job.setReducerClass(avgReducer.class);
  job.setOutputKeyClass(Text.class);
  job.setOutputValueClass(FloatWritable.class);
  Path p=new Path(args[0]);
  Path p1=new Path(args[1]);
  FileInputFormat.addInputPath(job, p);
  FileOutputFormat.setOutputPath(job, p1);
  job.waitForCompletion(true);
 }
}
```

Input:

```
5154    15.00
5155    16.00
5154    17.00
```

Output:

```
5154    16.0
5155    16.0
```

6. Implement matrix multiplication with Hadoop Map Reduce

Map.java

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

        import org.apache.hadoop.io.LongWritable;

        import org.apache.hadoop.io.Text;

        import org.apache.hadoop.mapreduce.Mapper;

        import java.io.IOException;

        public class Map extends org.apache.hadoop.mapreduce.Mapper<LongWritable, Text,
Text, Text> {

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

                        Configuration conf=context.getConfiguration();

                        int m = Integer.parseInt(conf.get("m"));

                        int p = Integer.parseInt(conf.get("p"));

                        String line = value.toString();

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

                        Text outputKey = new Text();

                        Text outputValue =  new Text();

                        if(indiciesAndValue[0].equals("M")) {

                                for(int k=0; k<p; k++) {

                                        outputKey.set(indiciesAndValue[1] + "," +k);

                                        outputValue.set(indiciesAndValue[0]    +    ","    +
indiciesAndValue[2] + "," + indiciesAndValue[3]);

                                        context.write(outputKey, outputValue);


                                }

                        } else {

                                for(int i=0; i<m; i++){

                                        outputKey.set(i + "," + indiciesAndValue[2]);

                                                outputValue.set("N,"  +  indiciesAndValue[1]  +
                "," + indiciesAndValue[3]);
```

```
                                        context.write(outputKey, outputValue);
                                }
                        }

                }

        }
```

Reduce.java

```
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Reducer;
import java.io.IOException;
import java.util.HashMap;
```

```
public class Reduce extends org.apache.hadoop.mapreduce.Reducer<Text, Text, Text, Text>
{
        public void reduce(Text key, Iterable<Text> values, Context context) throws
IOException, InterruptedException {
                String[] value;
                HashMap<Integer,Float> hashA = new HashMap<Integer, Float>();
                HashMap<Integer,Float> hashB = new HashMap<Integer, Float>();
                for(Text val : values) {
                        value = val.toString().split(",");
                        if(value[0].equals("M")) {
                        hashA.put(Integer.parseInt(value[1]), Float.parseFloat(value[2]));
                        }else {
                                hashB.put(Integer.parseInt(value[1]),
Float.parseFloat(value[2]));
                        }
                }
                int n = Integer.parseInt(context.getConfiguration().get("n"));
                float result = 0.0f;
                float m_ij;
                float n_jk;
                for(int j=0; j<n; j++) {
                        m_ij = hashA.containsKey(j) ? hashA.get(j) : 0.0f;
                        n_jk = hashB.containsKey(j) ? hashB.get(j) : 0.0f;
                        result += m_ij * n_jk;
                        }
                if(result != 0.0f) {
                        context.write(null, new Text(key.toString() + "," +
Float.toString(result)));
                }
        }
}
```

}

MatrixMultiply.java

```
import org.apache.hadoop.conf.*;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.*;
import org.apache.hadoop.mapreduce.*;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;

public class MatrixMultiply {

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

                if(args.length != 2) {

                        System.err.println("Usage: MatrixMultiply <in_dir> <out_dir>");

                        System.exit(2);

                }

                Configuration conf = new Configuration();

                conf.set("m","1000");

                conf.set("n","100");

                conf.set("p","1000");

                Job job = new Job(conf, "MatrixMultiply");

                job.setJarByClass(MatrixMultiply.class);

                job.setOutputKeyClass(Text.class);

                job.setOutputValueClass(Text.class);

                job.setMapperClass(Map.class);

                job.setReducerClass(Reduce.class);

                job.setInputFormatClass(TextInputFormat.class);

                job.setOutputFormatClass(TextOutputFormat.class);

                FileInputFormat.addInputPath(job, new Path(args[0]));

                FileOutputFormat.setOutputPath(job, new Path(args[1]));

                job.waitForCompletion(true);

                        }

}
```

Input:

```
M,1,1,1

M,1,2,2

M,2,1,3

M,2,2,4

N,1,1,5

N,1,2,6

N,2,1,7

N,2,2,8
```

Output:

```
1,1,19.0

1,2,22.0

2,1,43.0

2,2,50.0
```

Patent Reduce Program

Problem Definition: Apply your MapReduce programming knowledge and write a MapReduce program to process a dataset with patent records. You need to calculate the number of sub-patents associated with each patent.

Writing space of the Problem:(Students use)

Patent.java

```
import java.io.IOException;

import java.util.StringTokenizer;

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.Mapper.Context;

import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;

import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;

import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;

public class Patent {

        public static class Map extends Mapper<LongWritable, Text, Text, Text> {

                Text k= new Text();

                Text v= new Text();
```

```
                    public void map(LongWritable key, Text value, Context context) throws
IOException, InterruptedException {
                              String line = value.toString();
                              StringTokenizer tokenizer = new StringTokenizer(line," ");
                    while (tokenizer.hasMoreTokens()) {
                              String jiten= tokenizer.nextToken();
                              k.set(jiten);
                              String jiten1= tokenizer.nextToken();
                              v.set(jiten1);
                              context.write(k,v);
                    }
                    }
          }
          public static class Reduce extends Reducer<Text, Text, Text, IntWritable> {
                    public void reduce(Text key, Iterable<Text> values, Context context) throws
IOException, InterruptedException {
          int sum = 0;
          for(Text x : values)
          {
                    sum++;
          }
          context.write(key, new IntWritable(sum));
                    }
          }
          public static void main(String[] args) throws Exception {
                    Configuration conf = new Configuration();
                    Job job = new Job(conf, "patent");
                    job.setJarByClass(Patent.class);
                    job.setMapperClass(Map.class);
                    job.setReducerClass(Reduce.class);
                    job.setMapOutputKeyClass(Text.class);
```

```
		job.setMapOutputValueClass(Text.class);
		job.setOutputKeyClass(Text.class);
		job.setOutputValueClass(IntWritable.class);
		job.setOutputKeyClass(Text.class);
		job.setOutputValueClass(Text.class);
		job.setInputFormatClass(TextInputFormat.class);
		job.setOutputFormatClass(TextOutputFormat.class);
		Path outputPath = new Path(args[1]);
		FileInputFormat.addInputPath(job, new Path(args[0]));
		FileOutputFormat.setOutputPath(job, new Path(args[1]));
		outputPath.getFileSystem(conf).delete(outputPath);
		System.exit(job.waitForCompletion(true) ? 0 : 1);
	}
}
```

Input:

```
1 1.111
1 1.11
1 1.220
1 1.3
1 1.169
1 1.189
1 1.19
1 1.236
1 1.67
1 1.160
1 1.205
1 1.68
1 1.92
2 2.179
2 2.206
```

```
2 2.59
2 2.111
2 2.163
2 2.12
2 2.93
2 2.75
2 2.20
2 2.29
3 3.233
3 3.171
3 3.197
3 3.40
```

Output:

```
1       13
2       10
3       4
```

{

Word Size Word Count

Exp 9: Implement a Hadoop program to count words and size in a given file using Map Reduce – WordCountSize.

Problem Definition: Apply the MapReduce programming knowledge and write a MapReduce program to calculate wordcount size in a given file.

Writing space of the Problem:(Students use)

WordSizeWordCount.java

```
import java.io.IOException;

import java.util.StringTokenizer;

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.input.TextInputFormat;

import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;

public class WordSizeWordCount

{

        public static class Map extends Mapper<LongWritable, Text, IntWritable, Text>

        {

                private static IntWritable count ;

                private Text word = new Text();

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

                 {

                        String line = value.toString();
                        StringTokenizer tokenizer = new StringTokenizer(line);
                        while (tokenizer.hasMoreTokens())
                        {
                                String thisH = tokenizer.nextToken();
                                count= new IntWritable(thisH.length());
                                word.set(thisH);
                                context.write(count,word);
                        }
                }
}
        public static class Reduce extends Reducer<IntWritable, Text, IntWritable,
IntWritable>
        {
                public void reduce(IntWritable key, Iterable<Text> values, Context context)
throws IOException, InterruptedException
                {
                        int sum = 0;
                        for(Text x : values)
                        {
                                sum++;
                        }
                        context.write(key, new IntWritable(sum));
                }
        }
        public static void main(String[] args) throws Exception
        {
                Configuration conf = new Configuration();
                Job job= new Job(conf, "Wordsize");
                job.setJarByClass(WordSizeWordCount.class);
                job.setMapperClass(Map.class);
                job.setReducerClass(Reduce.class);
```

```
		job.setMapOutputKeyClass(IntWritable.class);
		job.setMapOutputValueClass(Text.class);
		job.setOutputKeyClass(IntWritable.class);
		job.setOutputValueClass(IntWritable.class);
		job.setInputFormatClass(TextInputFormat.class);
		job.setOutputFormatClass(TextOutputFormat.class);
		Path outputPath = new Path(args[1]);
		FileInputFormat.addInputPath(job, new Path(args[0]));
		FileOutputFormat.setOutputPath(job, new Path(args[1]));
		outputPath.getFileSystem(conf).delete(outputPath);
		System.exit(job.waitForCompletion(true) ? 0 : 1);
	}
}
```

Input:

```
welcome to AIDS
welcome
```

Output:

```
2	1
4	1
7	2
```

Big data is a field that treats ways to analyze, systematically extract information from, or otherwise deal with data sets that are too large or complex to be dealt with by traditional data processing application software.

1. **Volume**
2. **Variety(structured**(table)**, unstructured**(audio,video)**, semi-structured**(XML, JSON, CSV)**)**
3. **Velocity**
4. **Value**
5. **Veracity** (Uncertainty, Inconsistency)

Applications:

Smarter Healthcare, Multi-channel Sales, Homeland Security, Telecom, Traffic Control, Trading Analysis, Manufacturing, Search Engine.

Industry Examples:

Web Analytics, Marketing, Fraud detection, Credit Risk Management, Healthcare(Health Records, Prescription, Appointment, Drug Detection, Disease Prediction), Advertising

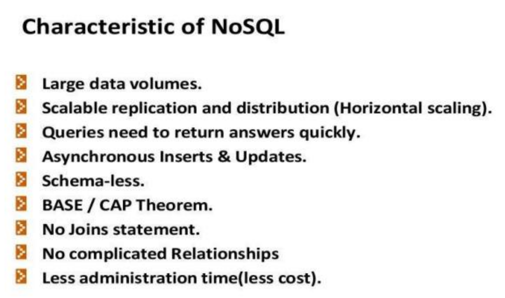
Big-Data Technologies:

Column-oriented Databases, NOSQL, Map Reduce, Hadoop, Hive.

Hadoop: Open Source framework, Reliable and fault tolerance, Horizontal Scalability, native data format and used for to perform a variety of analysis and transformations on the given data.

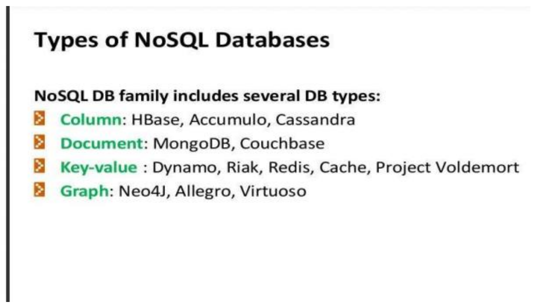
It Consists:

1. **Hadoop common** (Libraries and utilities required)
2. **Hadoop Distributed File System(HDFS)** (Server->Name node ->Data-Node)
3. **Hadoop YARN** (Resource manager for scalability, compatibility etc)
4. **Hadoop Map Reduce**



Characteristic of NoSQL

- Large data volumes.
- Scalable replication and distribution (Horizontal scaling).
- Queries need to return answers quickly.
- Asynchronous Inserts & Updates.
- Schema-less.
- BASE / CAP Theorem.
- No Joins statement.
- No complicated Relationships
- Less administration time(less cost).



Types of NoSQL Databases

NoSQL DB family includes several DB types:

- **Column**: HBase, Accumulo, Cassandra
- **Document**: MongoDB, Couchbase
- **Key-value** : Dynamo, Riak, Redis, Cache, Project Voldemort
- **Graph**: Neo4J, Allegro, Virtuoso

Distribution Models:

1. **Replication** (takes same data and copies it over multiple nodes)
   a) Master-slave
   b) Peer-Peer

   Advantage is: Availability
2. **Sharding** (put different data on different nodes) (Scalability)

Version Stamps:

Version stamps help you detect concurrency conflicts. When you read data, then update it, you can check the version stamp to ensure nobody updated the data between your read and write

Version stamps can be implemented using

1. counters,
2. GUIDs(Guaranteed Unique ID),
3. content hashes,
4. timestamps, or
5. a combination of these.

Composition of Map Reducer:

1. **Input**
2. **Splitting**
3. **Mapping**
4. **Shuffling**
5. **Reducing**
6. **Result**