**BIG DATA WITH HADOOP TRAINING ASSIGNMENTS**

**NAME: SAGNIK MUKHERJEE**

**UNIVERSITY: KIIT DEEMED UNIVERSITY**

**YEAR: 1ST**

**ASSIGNMENT #1**

Q1. Use “bible.txt” and write a java MapReduce code After running for a little while, the output should be as follows:  
and 12846  
i 8854  
god 4114  
The output has two columns: the first column has a lower-case version of a title-cased word that appears in the ASCII bible and the second column has a count of the number of times that word appears in the bible. The output is trimmed to only display the top 20 results sorted by descending word count.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

JAVA MAP-REDUCE CODE-

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.lib.input.FileInputFormat;

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

import org.apache.hadoop.mapreduce.Job;

import org.apache.hadoop.mapreduce.Mapper;

import org.apache.hadoop.mapreduce.Reducer;

public class bib {

static int x=0;

public static void main(String []args

)throws Exception{

Configuration conf=new Configuration();

Job job=Job.getInstance(conf,"wordcount");

job.setJarByClass(bib.class);

job.setMapperClass(MyMapper.class);

job.setReducerClass(MyReducer.class);

job.setOutputKeyClass(Text.class);

job.setOutputValueClass(IntWritable.class);

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

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

boolean bol=job.waitForCompletion(true);

System.exit(bol?0:1);

}

public static class MyMapper extends 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, Context context

)throws IOException, InterruptedException{

StringTokenizer itr=new StringTokenizer(value.toString());

while(itr.hasMoreTokens()){

word.set(itr.nextToken());

if(Character.isUpperCase(word.toString().charAt(0)))

{word=new Text(word.toString().toLowerCase());

context.write(word,one);}

}

}

}

public static class MyReducer extends Reducer<Text, IntWritable, Text, IntWritable>{

private IntWritable result=new IntWritable();

public void reduce(Text key, Iterable<IntWritable> values,Context context

)throws IOException,InterruptedException{

int sum=0;

for(IntWritable val : values)

{

sum+=val.get();

}

result.set(sum);

context.write(key,result);

}

}

}

TERMINAL COMMANDS-

1. Hadoop jar example.jar bible.txt /user/cloudera/output (Map-Reduce Begins!) <-
2. Hdfs dfs -cat /user/cloudera/output/part-r-00000 | sort -n -k2 -r | head -n20 <-

TERMINAL OUTPUT-

and 12738

i 8708

lord 4600

god 2282

the 1858

for 1670

lord, 1647

but 1542

then 1373

israel 1098

o 1065

god, 1007

jesus 777

israel, 766

he 753

david 694

lord. 689

now 622

so 621

thou 573

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Q2. Use “usdecaration.txt” and get the canonical "Word Count" example working. Add a partitioner code to segregate words and store into separate files:  
part-r-00000 => words of length 2 & 3  
part-r-00001 => words of length 4 & 5  
part-r-00002 => words of length 6 & 7  
part-r-00003 = > all other words  
Note: Omit words of count 1

JAVA MAP-REDUCE CODE-

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.lib.input.FileInputFormat;

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

import org.apache.hadoop.mapreduce.Job;

import org.apache.hadoop.mapreduce.Mapper;

import org.apache.hadoop.mapreduce.Reducer;

public class WC {

public static void main(String []args

)throws Exception{

Configuration conf=new Configuration();

Job job=Job.getInstance(conf,"wordcount");

job.setJarByClass(WC.class);

job.setPartitionerClass(MyPartition.class);

job.setNumReduceTasks(5);

job.setMapperClass(MyMapper.class);

job.setReducerClass(MyReducer.class);

job.setOutputKeyClass(Text.class);

job.setOutputValueClass(IntWritable.class);

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

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

boolean bol=job.waitForCompletion(true);

System.exit(bol?0:1);

}

public static class MyMapper extends 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, Context context

)throws IOException, InterruptedException{

StringTokenizer itr=new StringTokenizer(value.toString());

while(itr.hasMoreTokens()){

word.set(itr.nextToken());

if(word.toString().length()!=1)

context.write(word,one);

}

}

}

public static class MyReducer extends Reducer<Text, IntWritable,

Text, IntWritable>{

private IntWritable result=new IntWritable();

public void reduce(Text key, Iterable<IntWritable> values,Context context

)throws IOException,InterruptedException{

int sum=0;

for(IntWritable val : values)

{

sum+=val.get();

}

result.set(sum);

context.write(key,result);

}

}

}

JAVA PARTITIONER CODE-

**import** org.apache.hadoop.io.IntWritable;

**import** org.apache.hadoop.io.Text;

**import** org.apache.hadoop.mapreduce.Partitioner;

**public** **class** MyPartition **extends** Partitioner<Text, IntWritable>{

**public** **int** getPartition(Text key, IntWritable value, **int** setNumRed){

String s=key.toString().trim();

**if**(s.length()==2 || s.length()==3)

**return** 0;

**else** **if** (s.length()==4 || s.length()==5)

**return** 1;

**else** **if** (s.length()==6 || s.length()==7)

**return** 2;

**else**

**return** 3;

}

}

TERMINAL COMMANDS-

1. hadoop jar part10.jar sample.txt /user/cloudera/output3 <-
2. hdfs dfs -cat /user/cloudera/output3/part-r-00000 <-

TERMINAL OUTPUTS(part-m-00000)-

(it 1

By 1

HPC 1

MPI 2

So 1

The 2

and 6

are 1

as 6

at 3

but 4

by 4

can 1

do 1

fed 1

for 5

has 2

if 1

in 9

is 15

it 8

its 1

map 1

no 1

not 2

of 7

on 2

one 1

or 1

own 1

run 1

so 1

the 33

to 16

via 1

you 1

TERMINAL OUTPUTS(part-m-00001)-

(HPC) 1

(This 1

(such 1

This 2

able 1

about 1

and, 1

area 1

been 1

care 1

case, 1

data 11

does 1

doing 1

don’t 1

easy 1

fast 1

flow 1

flow, 1

from 3

gives 2

goes 1

good 1

great 2

grid 1

have 3

heart 1

idle. 1

jobs, 1

know 1

known 1

like 1

links 1

make 1

makes 1

map, 1

maps 1

model 1

more 3

most 1

need 1

needs 1

nodes 2

not, 1

only 1

order 1

pairs 1

point 2

since 3

spare 1

still 1

such 2

sure 1

take 1

tasks 3

terms 1

than 1

that 5

their 1

them 2

they 1

think 1

this 4

tries 1

under 1

using 1

view, 1

well 2

when 1

which 4

while 1

with 2

work 1

works

1

Q3. Modify the Wordcount code and apply it on “bible.txt” to display the output in the following format:  
<word> [<count>] i.e. and [12846].

JAVA MAP-REDUCE CODE-

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.lib.input.FileInputFormat;

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

import org.apache.hadoop.mapreduce.Job;

import org.apache.hadoop.mapreduce.Mapper;

import org.apache.hadoop.mapreduce.Reducer;

public class WC{

public static void main(String []args

)throws Exception{

Configuration conf=new Configuration();

Job job=Job.getInstance(conf,"wordcount");

job.setJarByClass(WC.class);

job.setMapperClass(MyMapper.class);

job.setReducerClass(MyReducer.class);

job.setOutputKeyClass(Text.class);

job.setOutputValueClass(IntWritable.class);

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

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

boolean bol=job.waitForCompletion(true);

System.exit(bol?0:1);

}

public static class MyMapper extends 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, Context context

)throws IOException, InterruptedException{

StringTokenizer itr=new StringTokenizer(value.toString());

while(itr.hasMoreTokens()){

word.set(itr.nextToken());

context.write(word,one);

}

}

}

public static class MyReducer extends Reducer<Text, IntWritable, Text, Text>{

private IntWritable result=new IntWritable();

public void reduce(Text key, Iterable<IntWritable> values,Context context

)throws IOException,InterruptedException{

int sum=0;

for(IntWritable val : values)

{

sum+=val.get();

}

result.set(sum);

String str="["+sum+"]";

Text xm = new Text(str);

context.write(key,xm);

}

}

}

TERMINAL COMMANDS-

1. hadoop jar wc20.jar sample.txt /user/cloudera/output4 <-
2. hdfs dfs -cat /user/cloudera/output4/part-r-00000 <-

TERMINAL OUTPUTS(part-m-00000)-

(APIs) [1]

(HPC) [1]

(MPI). [1]

(SAN). [1]

(This [1]

(hundreds [1]

(it [1]

(such [1]

Broadly, [1]

By [1]

C [1]

Coordinating [1]

Distributed [1]

HPC [1]

Hadoop [5]

Hadoop. [1]

Interface [1]

MPI [2]

MapReduce [3]

MapReduce), [1]

Message [1]

Notice [1]

Passing [1]

Processing [1]

Recognizing [1]

So [1]

The [2]

This [2]

a [12]

able [1]

about [1]

access [3]

across [1]

again.) [1]

algorithms [1]

analyses [1]

analyses. [1]

and [6]

and, [1]

application [1]

approach [1]

architecture, [1]

are [1]

area [1]

around), [1]

arrangement [1]

as [6]

aspect [1]

at [3]

bandwidth [2]

because [3]

become [1]

becomes [1]

been [1]

bottleneck [1]

but [4]

by [4]

can [1]

care [1]

case, [1]

center [1]

challenge. [1]

checkpointing [1]

cluster [1]

co-locate [1]

communities [1]

computation [1]

computation. [1]

compute [2]

compute-intensive [1]

computing [2]

conserve [1]

constructs [1]

contrast, [1]

control [3]

copying [1]

data [11]

dependence [1]

detects [1]

difficult [1]

distribute [1]

distributed [1]

do [1]

does [1]

doesn’t [1]

doing [1]

don’t [1]

easy [1]

environment [1]

explicitly [3]

exposed [1]

failed [3]

failed—and [1]

failure, [1]

failure—when [1]

fast [1]

feature, [1]

fed [1]

filesystem, [1]

flow [1]

flow, [1]

for [5]

frameworks [1]

from [3]

gigabytes, [1]

gives [2]

goes [1]

good [1]

gracefully [1]

great [2]

grid [1]

handle [1]

handling [1]

hardest [1]

has [2]

have [3]

having [1]

healthy. [1]

heart [1]

high-CPU [1]

high-performance [1]

higher [1]

higher-level [1]

hosted [1]

idle. [1]

if [1]

implementation [1]

implicit. [1]

in [9]

interfaces [1]

is [15]

it [8]

its [1]

jobs, [1]

key-value [1]

know [1]

known [1]

large-scale [2]

larger [1]

lengths [1]

level: [1]

like [1]

links [1]

local.6 [1]

locality, [1]

low-level [1]

machines [1]

machines, [1]

make [1]

makes [1]

making [1]

manage [1]

map [1]

map, [1]

mappers [1]

maps [1]

matter. [1]

meaning [1]

mechanics [1]

model [1]

modeling [1]

more [3]

most [1]

necessary [1]

need [1]

needs [1]

network [5]

no [1]

nodes [2]

nodes, [1]

not [2]

not, [1]

of [7]

on [2]

one [1]

only [1]

operates [1]

or [1]

order [1]

other. [1]

output [1]

outputs [1]

overall [1]

oversimplification, [1]

own [1]

pairs [1]

partial [1]

performance. [1]

point [2]

precious [1]

preclude [1]

predominantly [1]

problem [1]

process [1]

processes [1]

processing [3]

program [1]

programmer [3]

programmers, [1]

programmer’s [1]

programs [1]

progress [1]

really [1]

reason [1]

recovery, [1]

reducer [1]

reducers, [1]

regenerate [1]

relevant [1]

remains [1]

remote [1]

replacements [1]

requires [1]

rerunning [2]

reschedules [1]

resource [1]

retrieve [1]

routines [1]

run [1]

running [1]

saturate [1]

shared [1]

shared-nothing [1]

shine), [1]

since [3]

slight [1]

so [1]

sockets, [1]

spare [1]

starts [1]

still [1]

storage [1]

such [2]

sure [1]

system; [1]

take [1]

tasks [3]

terms [1]

than [1]

that [5]

the [33]

their [1]

them [2]

they [1]

think [1]

thinks [1]

this [4]

to [16]

topology. [1]

tries [1]

under [1]

using [1]

via [1]

view, [1]

volumes [1]

well [2]

when [1]

whether [1]

which [4]

while [1]

with [2]

work [1]

works [1]

write. [1]

years, [1]

you [1]

**ASSIGNMENT #2 (SQOOP)**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Q3. Process “xml\_data.txt” using Hive or Pig then export the xml contents to MYSQL database using sqoop. Create the table schema to store all the contents assume types as per data content.

MYSQL COMMANDS-

1. mysql -u root -p <-
2. mysql> show databases; <-
3. mysql> use detail\_db; <-
4. mysql> create table book (id varchar(10),author char(10),title varchar(20),genre char(10),price float,publish\_date varchar(10),description varchar(50));
5. mysql> quit;

HIVE COMMANDS-

1. hive> show tables; <-
2. hive> add jar hivexmlserde-1.0.0.0.jar; <-
3. hive> use aspire; <-
4. hive> CREATE EXTERNAL TABLE xml\_data

> (id string,

> author string,

> title string,

> genre string,

> price float,

> publish\_date string,

> description string)

> ROW FORMAT SERDE 'com.ibm.spss.hive.serde2.xml.XmlSerDe'

> with serdeproperties(

> "column.xpath.id"="/book/id/text()",

> "column.xpath.author"="/book/author/text()",

> "column.xpath.title"="/book/title/text()",

> "column.xpath.genre"="/book/genre/text()",

> "column.xpath.price"="/book/price/text()",

> "column.xpath.publish\_date"="/book/publish\_date/text()",

> "column.xpath.description"="/book/description/text()")

> stored as inputformat 'com.ibm.spss.hive.serde2.xml.XmlInputFormat'

> outputformat 'org.apache.hadoop.hive.ql.io.IgnoreKeyTextOutputFormat'

> tblproperties("xmlinput.start"="<book>","xmlinput.end"="</book>"); <-

1. hive> LOAD DATA LOCAL INPATH 'xml\_data.txt' INTO TABLE xml\_data; <-
2. hive> select \* from xml\_data; <- (OPTIONAL)

SQOOP COMMANDS-