University of CINCINNATI

BIG DATA ANALYTICS

CS 7070

ASSIGNMENT -1

Submitted on:

02/08/2019

Submitted by

Siva Sai Krishna Paladugu

Problem Statement:

1. Design and execute a MapReduce program to produce the frequencies of all the words in the book collection, retaining only those words whose frequencies are greater than 5.

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.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 WordCountFinal {

/*
 * Creating Custom Mapper Class
 */
public static class Mapper1 extends Mapper<Object, Text, Text,IntWritable>
{
      private final static IntWritable one = new IntWritable(1);
      private Text word = new Text();
      /*
       * Creating Custom map method with
       * Input - Object, Text
       * Output - Text, Intwritable (key, value)
       */
      public void map(Object key, Text value, Context context) throws
IOException, InterruptedException
      {

            /*
             * Preprocessing data
             * Entire line from document is converted to string and made
to lowercase.
             * Also taking only ALPHA values and space .i.e., only
alphabets and space.
             */
            String line = value.toString().toLowerCase();
            line = line.replaceAll("[^A-Za-z ]", "");

            StringTokenizer itr = new StringTokenizer(line);

            //checking for next word whether exist or not and writing that
      word as key to context and values as 1.
            while (itr.hasMoreTokens()) {
                  word.set(itr.nextToken());
                  context.write(word, one);
```

```
            }
      }
}
/*
 * Creating Custom Reducer Class
 */
public static class Reducer1 extends Reducer<Text, IntWritable, Text,
IntWritable>
{
      private IntWritable result = new IntWritable();
      /*
       * Creating Custom reduce method with
       * Input - Text, list of IntWritables (key , list<val>)
       * Output - Text, Intwritable (key, value)
       */
      public void reduce(Text key, Iterable<IntWritable> values, Context
context) throws IOException, InterruptedException
      {
            /*
             * Taking temporary variable "sum" and looping over values and
adding to it to get total count for wach key
             * Also considering only those key,value pairs in which value
is greater than 5
             */
            int sum = 0;
            for (IntWritable val: values) {
                  sum += val.get();
            }
            if (sum > 5)
            {
                  result.set(sum);
                  context.write(key, result);
            }
      }
}

      /*
       * Creating Main method to run the job with custom configuration
       * System exits once job completes.
       */
      public static void main(String[] args) throws Exception {
            Configuration conf = new Configuration();
            Job job = Job.getInstance(conf, "Word Count");
            job.setJarByClass(WordCountFinal.class);
            job.setMapperClass(Mapper1.class);
            job.setCombinerClass(Reducer1.class);
            job.setReducerClass(Reducer1.class);
            job.setOutputKeyClass(Text.class);
            job.setOutputValueClass(IntWritable.class);
            FileInputFormat.addInputPath(job, new Path(args[0]));
            FileOutputFormat.setOutputPath(job, new Path(args[1]));
            System.exit(job.waitForCompletion(true) ? 0 : 1);
      }
}
```

Commands Executed:

- mkdir run
- cd run
- export HADOOP_CLASSPATH=$(hadoop classpath)
- mkdir WCclasses
- hadoop com.sun.tools.javac.Main WordCountFinal.java -d WCclasses/
- jar -cvf WordCountFinal.jar -C WCclasses/ .
- hadoop jar WordCountFinal.jar WordCountFinal /user/paladusn/sskAssign1/input /user/paladusn/sskAssign1/WCFinaloutput.
- Used head -50 OPFILENAME > TMPFILE1.TXT to get top 50 words and saved it to file.
- Used tail -50 OPFILENAME > TMPFILE2.TXT to get last 50 words and saved it to file.

Results:

Top 50 words and their frequencies(freq>5):

a	34606
aaron	30
abandon	8
abandond	6
abandoned	17
abbey	9
abhorred	12
abhorrence	12
abilities	23
ability	41
able	368
aboard	49
abode	25
abodes	27
abolished	7
abominable	6
abounding	7
about	2460
above	414
abraham	46
abroad	37
absence	60
absent	32
absolute	73
absolutely	79
absolution	7
absolved	6
absorbed	7
absurd	45

absurdities	9
absurdity	12
abuse	15
acamas	8
accents	12
accept	57
acceptable	6
acceptation	6
accepted	48
accepteth	7
access	224
accident	43
accidents	28
acclaim	6
accompanied	21
accompany	6
accomplished	28
accord	6
according	161
accordingly	31
account	258

Last 50 words and their frequencies (freq>5):

xx	6
yaller	6
yard	44
yards	59
ye	638
yea	14
year	227
years	603
yee	24
yell	7
yelled	6
yelling	9
yellow	83
yer	10
yes	726
yesterday	91
yet	1862
yield	104
yielded	8
yielding	9
yields	23

yit	9
yo	14
yojo	14
yoke	16
yon	81
yonder	61
yore	10
york	6
you	13701
youd	67
youll	98
young	790
younger	48
youngest	48
your	3413
youre	125
yours	91
yourself	302
yourselves	17
yous	17
youth	214
youthful	46
youths	20
youve	58
yuther	7
zachary	6
zeal	9
zealand	7
zion	9

2. Design and execute a MapReduce program to produce frequencies of all 2-gram word-pairs in the book collection, retaining only those 2-grams whose frequencies are greater than 5.

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.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 BigramFinal {

   public static int flag=0; //flag for indication of the end word of line
   public static String tmpWord = "";
   /*
    * Creating Custom Mapper Class
    */
   public static class Mapper2 extends Mapper<Object, Text, Text,
IntWritable>
{
      private final static IntWritable one = new IntWritable(1);
      private Text word = new Text();
      /*
       * Creating Custom map method with
       * Input - Object, Text
       * Output - Text, Intwritable (key, value)
       */
      public void map(Object key, Text value, Context context) throws
IOException, InterruptedException {

            /*
             * Preprocessing data
             * Entire line from document is converted to string and made
to lowercase.
             * Also taking only ALPHA values and space .i.e., only
alphabets and space.
             */
            String line = value.toString().toLowerCase();
            line = line.replaceAll("[^A-Za-z ]", "");
            StringTokenizer itr = new StringTokenizer(line);
            //Taking two tmp variables to store consecutive words
            String word1 = "";
            String word2 = "";
            //Check for words exists and setting the first word also
considering flag to check whether to set from new line or previous line
            if(itr.hasMoreTokens())
```

```
            {
              if(flag==1)
              {
                  flag = 0;
                  word1 = tmpWord;
              }
              else
              {
                  word1 = itr.nextToken();
              }
             }
             //checking for next words and setting the consecutive word and
changing word1 variable to new word2. Setting key as two consecutive words
with space in between them.
             while (itr.hasMoreTokens())
             {
               word2 = itr.nextToken();
               word.set(word1 + ' ' + word2);
               //System.out.println(first+' '+second);
               context.write(word, one);
               word1=word2;
      //storing last word in new variable to set to word1 for next line
               if(!itr.hasMoreTokens())
               {
                   tmpWord = word2;
                   flag = 1;
               }
              }
            }
           }
/*
 * Creating Custom Reducer Class
 */
      public static class Reducer2 extends Reducer<Text, IntWritable,
Text, IntWritable> {


         private IntWritable result = new IntWritable();
      /*
       * Creating Custom reduce method with (here key is like "word1
word2")
       * Input - Text, list of IntWritables (key , list<val>)
       * Output - Text, Intwritable (key, value)
       */
         public void reduce(Text key, Iterable<IntWritable> values,
Context context) throws IOException, InterruptedException
       {
            /*
             * Taking temporary variable "sum" and looping over values and
adding to it to get total count for wach key
             * Also considering only those key,value pairs in which value
is greater than 5
             */
```

```
            int sum = 0;

            for (IntWritable val : values)
            {
                  sum += val.get();
            }
            if (sum > 5)
            {
                  result.set(sum);
                  context.write(key, result);
            }
       }
      }
      /*
       * Creating Main method to run the job with custom configuration
       * System exits once job completes.
       */
      public static void main(String[] args) throws Exception {
            Configuration conf = new Configuration();
            Job job = Job.getInstance(conf, "BiGram");
            job.setJarByClass(BigramFinal.class);
            job.setMapperClass(Mapper2.class);
            job.setCombinerClass(Reducer2.class);
            job.setReducerClass(Reducer2.class);
            job.setOutputKeyClass(Text.class);
            job.setOutputValueClass(IntWritable.class);
            FileInputFormat.addInputPath(job, new Path(args[0]));
            FileOutputFormat.setOutputPath(job, new Path(args[1]));
            System.exit(job.waitForCompletion(true) ? 0 : 1);
      }
}
```

Commands Executed:

- mkdir run
- cd run
- export HADOOP_CLASSPATH=$(hadoop classpath)
- mkdir Bgclasses
- hadoop com.sun.tools.javac.Main BigramFinal.java -d Bgclasses/
- jar -cvf Bg.jar -C Bgclasses/ .
- hadoop jar Bg.jar BigramFinal /user/paladusn/sskAssign1/input /user/paladusn/sskAssign1/Bgoutput.
- Used head -50 OPFILENAME > TMPFILE1.TXT to get top 50 words and saved it to file.
- Used tail -50 OPFILENAME > TMPFILE2.TXT to get last 50 words and saved it to file.

Results:

Top 50 words and their frequencies (freq>5):

a bad	21
a ball	10
a beautiful	36
a bed	11

a been	18
a being	7
a ben	7
a better	20
a big	46
a bird	11
a bit	61
a black	22
a blame	6
a boat	24
a body	89
a book	13
a boon	7
a bound	9
a boy	29
a brave	6
a brazen	7
a broad	8
a brother	18
a bull	6
a cab	14
a calm	7
a candle	10
a cannibal	6
a canoe	7
a case	9
a cat	21
a cause	7
a certain	101
a chair	13
a chance	38
a change	8
a charming	6
a chaw	9
a chief	17
a child	42
a christian	39
a church	9
a circumstance	7
a circus	6
a civill	20
a clergyman	6
a clever	8
a cloak	6
a cloud	13

a coach	6

Last 50 words and their frequencies (freq>5):

you wont	33
you would	196
you wouldnt	14
you you	61
youd a	6
youd see	6
youll be	7
young and	7
young jerry	26
young ladies	25
young lady	85
young man	85
young mccarthy	6
young men	20
young woman	9
younger sisters	8
youngest son	6
your ancient	6
your brother	10
your case	9
your daughter	7
your diary	6
your eyes	8
your father	43
your friend	6
your hair	6
your hand	16
your head	15
your husband	17
your ladyship	7
your life	13
your majesty	34
your master	6
your mother	11
your name	16
your opinion	7
your own	88
your pardon	6
your room	7
your sister	22
your sisters	9

your son	11
your time	6
your uncle	19
your uncles	6
your wife	6
youre a	9
yourself in	6
youth and	12
youth of	9

3. Design and execute a MapReduce program to produce the top 100 most frequent words in the book collection. You may need two rounds of Map and Reduce processors.

Code:

```
import java.io.IOException;
import java.nio.ByteBuffer;
import java.util.StringTokenizer;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.conf.Configured;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.io.WritableComparator;
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 Program3 extends Configured
{
/*
 * Creating Custom Mapper Class for mapreduce phase 1
 */
      public static class Mapper3 extends Mapper<Object, Text, Text,
IntWritable> {
            private final static IntWritable one = new IntWritable(1);
            private Text word = new Text();
      /*
       * Creating Custom map method with
       * Input - Object, Text
       * Output - Text, Intwritable (key, value)
       */
            public void map(Object key, Text value, Context context)
throws IOException, InterruptedException {

            /*
             * Preprocessing data
             * Entire line from document is converted to string and made
to lowercase.
```

```
            * Also taking only ALPHA values and space .i.e., only
alphabets and space.
            */

                  String line = value.toString().toLowerCase();
                  line = line.replaceAll("[^A-Za-z ]", "");
                  StringTokenizer itr = new StringTokenizer(line);

            //checking for next word whether exist or not and writing that
      word as key to context and values as 1.
                  while (itr.hasMoreTokens()) {
                        word.set(itr.nextToken());
                        context.write(word, one);
                  }
            }
      }

/*
 * Creating Custom Reducer Class for mapreduce phase1
 */
      public static class Reducer3 extends Reducer<Text, IntWritable,
Text, IntWritable> {
            private IntWritable result = new IntWritable();
      /*
       * Creating Custom reduce method with
       * Input - Text, list of IntWritables (key , list<val>)
       * Output - Text, Intwritable (key, value)
       */
            public void reduce(Text key, Iterable<IntWritable> values,
Context context) throws IOException, InterruptedException {
            /*
             * Taking temporary variable "sum" and looping over values and
adding to it to get total count for wach key
             * considering all words independent of freq
             */

                  int sum = 0;
                  for (IntWritable val : values) {
                        sum += val.get();
                  }
                        result.set(sum);
                        context.write(key, result);
                  }
            }
      }

/*
 * Creating Custom Mapper Class for mapreduce phase 2 for swapping
 */

      public static class Mapper4 extends Mapper<Object, Text,
IntWritable, Text> {
            private Text word = new Text();
            private final static IntWritable tmpInt = new IntWritable();
```

```
	/*
	 * Creating Custom map method with
	 * Input - Object, Text
	 * Output - IntWritable, Text (key, value)
	 */

		@Override
		public void map(Object key, Text value, Context context)
throws IOException, InterruptedException {
			StringTokenizer itr = new
StringTokenizer(value.toString());

			//Logic for swapping
			String first = "";
			String second = "";
			if (itr.hasMoreTokens()) {
				first = itr.nextToken();
			}
			word.set(first);
			while (itr.hasMoreTokens()) {
				second = itr.nextToken();
			}
			int i = Integer.parseInt(second);

			//MULTIPLYING KEY WITH -1 TO GET DECREASING ORDER FINALLY
AFTER REDUCER
			i=-1*i;
			tmpInt.set(i);
			context.write(tmpInt, word);
		}
	}

/*
 * Creating Custom Reducer Class for mapreduce phase2
 */
	public static class Reducer4 extends Reducer<IntWritable, Text,
IntWritable, Text> {
		Text word = new Text();
	/*
	 * Creating Custom reduce method with
	 * Input - IntWritables, List of <Text> (key , list<val>)
	 * Output - IntWritable, Text (key, value)
	 */

		@Override
		public void reduce(IntWritable key, Iterable<Text> values,
Context context) throws IOException, InterruptedException {

		//setting each word and key as separate jey value pairs.
			for (Text val : values) {
				word.set(val);
				context.write(key, word);
			}
```

```
            }
      }

      /*
       * Creating Main method to run TWO JOBS with custom configuration
       * System exits once jobS completes.
       */
      public static void main(String[] args) throws Exception {
            Configuration conf = new Configuration();
            Job job = Job.getInstance(conf, "Word Count-3");
            job.setJarByClass(Program3.class);
            job.setMapperClass(Mapper3.class);
            job.setCombinerClass(Reducer3.class);
            job.setReducerClass(Reducer3.class);
            job.setOutputKeyClass(Text.class);
            job.setOutputValueClass(IntWritable.class);
            FileInputFormat.setInputPaths(job, new Path(args[0]));
            FileOutputFormat.setOutputPath(job, new Path(args[1] +
"/mpred1_op"));

            //Setting job2 after first job completes. Also giving path of
output for job1 as input to job2
            if (job.waitForCompletion(true)) {
                  Job job2 = Job.getInstance(conf, "Frequency");
                  job2.setJarByClass(Program3.class);
                  job2.setMapperClass(Mapper4.class);
                  job2.setCombinerClass(Reducer4.class);
                  job2.setReducerClass(Reducer4.class);
                  job2.setOutputKeyClass(IntWritable.class);
                  job2.setOutputValueClass(Text.class);
                  FileInputFormat.setInputPaths(job2, new Path(args[1] +
"/mpred1_op"));
                  FileOutputFormat.setOutputPath(job2, new Path(args[1] +
"/mpred2_op"));
                  System.exit(job2.waitForCompletion(true) ? 0 : 1);
            }
      }
}
```

Commands Executed:

- mkdir run
- cd run
- export HADOOP_CLASSPATH=$(hadoop classpath)
- mkdir Freqclasses
- hadoop com.sun.tools.javac.Main Program3.java -d Freqclasses/
- jar -cvf Freq.jar -C Freqclasses/ .
- hadoop jar Freq.jar Program3 /user/paladusn/sskAssign1/input /user/paladusn/sskAssign1/Freqoutput.
- Used head -100 OPFILENAME > TMPFILE1.TXT to get top 100 words and saved it to file.

Results:

Top 100 words and their frequencies: (Keys are frequencies. Ignore "-" which kept for sorting. Values are words)

-108996	the
-64491	and
-53779	of
-49146	to
-34606	a
-30063	in
-26635	i
-23867	that
-21249	it
-21223	he
-19292	his
-18563	was
-15232	is
-14974	with
-14128	for
-14021	as
-13701	you
-12950	but
-11992	not
-10868	be
-10237	had
-9955	by
-9884	on
-9570	at
-9560	all
-9150	him
-8815	my
-8665	her
-8590	this
-8451	they
-8141	so
-8113	have
-8089	from
-7911	or
-7302	which
-7146	she
-7044	me
-6272	there
-6116	no
-5982	when
-5815	one

-5814	said
-5794	are
-5713	their
-5640	we
-5444	if
-5418	were
-5300	then
-5256	them
-4988	what
-4427	out
-4222	an
-4162	would
-4014	up
-3909	will
-3863	now
-3786	been
-3736	any
-3688	do
-3609	who
-3579	more
-3565	man
-3564	some
-3477	could
-3413	your
-3324	into
-3146	other
-3008	time
-2938	such
-2901	very
-2736	see
-2732	may
-2725	upon
-2635	can
-2620	like
-2602	down
-2595	before
-2560	shall
-2559	our
-2519	than
-2460	about
-2448	little
-2427	well
-2425	its
-2390	must
-2375	did

-2360	has
-2269	know
-2257	only
-2244	over
-2217	mr
-2191	where
-2162	these
-2154	how
-2140	men
-2131	again
-2124	should
-2122	come
-2117	good
-2113	great