

INTRODUCTION

This report intends to provide the summary of steps taken to achieve the tasks of the Programming for Big Data - Hadoop/Map-Reduce Assignment. This assignment involves the use of Hadoop Map-Reduce tasks using Java programming, in Eclipse Luna IDE.

OVERVIEW OF THE DATA USED:

This project uses the Aggregated page view statistics for Wikimedia projects, collected by the Wikimedia Foundation, a nonprofit charitable organization dedicated to encouraging the growth, development and distribution of free, multilingual, educational content, and to providing the full content of these wiki-based projects to the public free of charge.

Each lines of each files contain the following information:

- First column is the language and the project name. Projects without a period are Wikipedia projects.
- The second column is the title of the page retrieved.
- Third column is the number of requests of that particular page in that particular hour.
- Fourth column is the size of the page.

Required tasks for this assignment:

1. To identify the popularity of Wikipedia projects, to retrieve the most popular sites with the most number of requests for that site. This should be done for three to six hour periods as the input.
2. From the output of the first task, the average page count per language should be identified. This is done by the use of chained MapReduce jobs, (i.e) The output of the first MapReduce job will serve as the input file for the second MapReduce job.

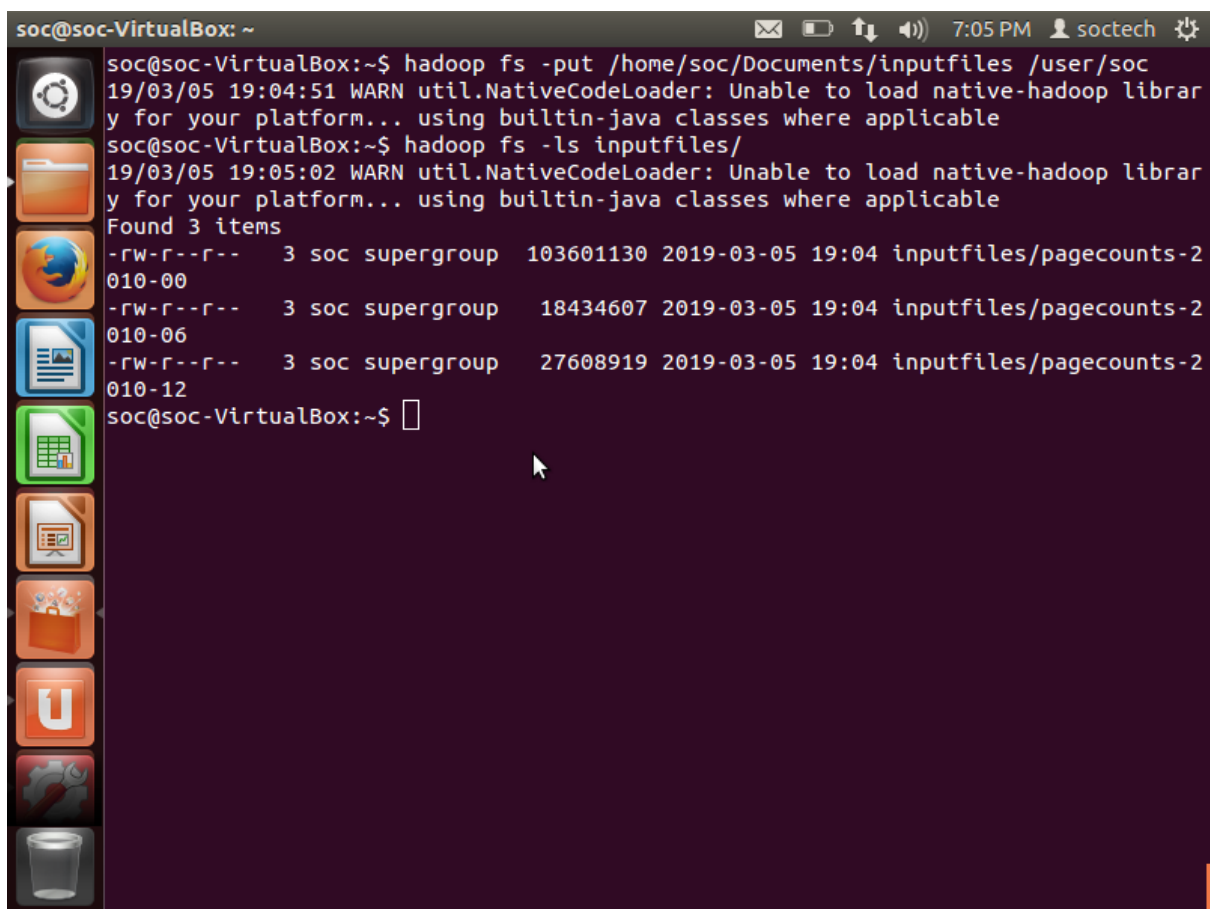
Explanation of tasks undertaken:

Loading of datasets into HDFS:

Three files from the year 2010 and month January were downloaded for the purpose of this assignment.

The syntax used to load a dataset into the HDFS:

```
hadoop fs -put <SourcePath> <DestinationPath>
```

A terminal window titled 'soc@soc-VirtualBox: ~' with a dark purple background. The terminal shows the execution of Hadoop commands. The first command is 'hadoop fs -put /home/soc/Documents/inputfiles /user/soc', which results in a warning message: '19/03/05 19:04:51 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable'. The second command is 'hadoop fs -ls inputfiles/', which results in another warning message: '19/03/05 19:05:02 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable'. Below the warning, it says 'Found 3 items' and lists three files with their permissions, owner, group, size, and path. The files are 'inputfiles/pagecounts-2010-00', 'inputfiles/pagecounts-2010-06', and 'inputfiles/pagecounts-2010-12'. The terminal window has a sidebar on the left with various application icons and a top bar with system status icons and the time '7:05 PM'.

```
soc@soc-VirtualBox:~$ hadoop fs -put /home/soc/Documents/inputfiles /user/soc
19/03/05 19:04:51 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
soc@soc-VirtualBox:~$ hadoop fs -ls inputfiles/
19/03/05 19:05:02 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
Found 3 items
-rw-r--r--  3 soc supergroup  103601130 2019-03-05 19:04 inputfiles/pagecounts-2010-00
-rw-r--r--  3 soc supergroup  18434607 2019-03-05 19:04 inputfiles/pagecounts-2010-06
-rw-r--r--  3 soc supergroup  27608919 2019-03-05 19:04 inputfiles/pagecounts-2010-12
soc@soc-VirtualBox:~$
```

Thus the folder inputfiles has three files. This folder will be used as the input for the MapReduce jobs.

Design and structure of the map-reduce processes:

The main driver class – WordCount.java has totally two MapReduce jobs running sequentially. The first MapReduce job will retrieve the most popular sites accessed, in a descending order. The output of the first job, will be stored as a separate file, in the output directory mentioned in the command line.

The second MapReduce job's task is to count the overall pagecounts of each Wikipedia project's language and divide it by the number of occurrences of the language, (i.e) finding the average pagecounts of every language. As this process is sequential, the output generated by first mapreduce job serves as the input to the second mapreduce job.

Explanation of task 1:

First Mapper:

- The task of first mapper is to identify only the Wikipedia projects, out of every other project, present in the file.
- This is done by first splitting every line of the file and then splitting the entire line into separate columns.
- To identify Wikipedia projects, a condition is used to check for the presence of period symbol in the first column. Only if the period symbol is not present, further steps will be taken. Or else, another line will be checked again.
- Then the third column containing the pagecount is passed as the key and the entire line is passed as the value to the Reducer.

Thus, the task of first Mapper job is identify Wikipedia projects, find the page count and write the (key, value) pairs to the Reducer. The mapper automatically performs the sorting. The user can make changes and sort the files in the preferred format.

First Reducer:

- The First Reducer then handles all the unique keys from the map's output values and performs a count of the total number of times the same site were accessed.
- The pagecount, project and site information are written back to the driver.
- The driver then stores the output from the first MapReduce job, as a new separate in the output directory mentioned by the user.

Explanation of task 2:

Second Mapper:

- The second mapper takes the output from the first MapReduce job, as its input file.
- The job of this mapper is to split the lines from previous output and identify the language & page count of every Wikipedia site.
- The mapper then writes the language as the key and pagecount as the value to the input of second reducer.

Second Reducer:

- The second reducer then continuously counts the pagecount (value) of every unique language (key), inside an iterator.
- In this reducer, an additional variable called as the counter is set to incrementally add by 1, every time the iterator runs.
- Finally, after the iterator is finished running, the average is calculated by dividing the total sum of pagecounts by the counter.
- Finally, the language is passed as key and the average is passed as the value from the second reducer.

Alternative design decisions considered:

- Initially planned to make use of a Combiner between the Mapper and Reducers, but the time taken and space constraint, made me to use just the Mapper and Reducers. Also, while performing the sorting after Mapper and using that as an input into the Combiner threw some errors, so couldn't make use of Combiners.
- Considered the use of performing TreeMap/Hashmaps/ToolRunner for sorting, but finding an easier alternative and very less need for lines of codes, I went ahead with using the code
`"job.setSortComparatorClass(LongWritable.DecreasingComparator.class)"`
to perform decrease order sorting.

Java Code:

WordCount.java:

```
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

public class WordCount {
    public static void main(String[] args) throws Exception {
        Configuration conf = new Configuration();
        Path out = new Path(args[1]);
        if (args.length != 2) {
            System.err.println("Input file or output file missing!");
            System.exit(-1);
        }
        System.out.println("In WordCount!");

        // This is the first MapReduce job in a series of chained MapReduce jobs
        Job job = Job.getInstance(conf, "WikiPageCount");
        job.setJarByClass(WordCount.class);
        job.setMapperClass(WikiSiteMapper.class);

        // Setting the map's key output as LongWritable, which is the pagecount in this case
        job.setMapOutputKeyClass(LongWritable.class);

        // Setting the map's value output as Text, which is the entire line in this case
        job.setMapOutputValueClass(Text.class);

        /*
```

```
* As MapReduce automatically sorts the output of mapper class based on
* the key, I passed the pagecount as key, from the WikiSiteMapper
* class, so that we can automatically see the most popular websites by
* using the setSortComparatorClass and setting the DecreasingComparator
* to sort it in decreasing order
*/
```

```
job.setSortComparatorClass(LongWritable.DecreasingComparator.class);
job.setReducerClass(WikiSiteReducer.class);
job.setOutputKeyClass(LongWritable.class);
job.setOutputValueClass(Text.class);
```

```
// MapReduce will pick the first argument as the input path
FileInputFormat.addInputPath(job, new Path(args[0]));
```

```
/*
* MapReduce will pick the second argument from command line, as the
* output directory and create the first mapreduce job's output file as pagecount
*/
FileOutputFormat.setOutputPath(job, new Path(out, "pagecount"));
```

```
if (!job.waitForCompletion(true)) {
    System.exit(1);
}
```

```
/*
* In order to create chained MapReduce jobs, creating a second job for
* the second task, to find the average counts of each language
*/
Job jobAvg = Job.getInstance(conf, "Average Count");
jobAvg.setJarByClass(WordCount.class);
jobAvg.setMapperClass(AverageMapper.class);
```

```

        jobAvg.setReducerClass(AverageReducer.class);
        jobAvg.setOutputKeyClass(Text.class);
        jobAvg.setOutputValueClass(LongWritable.class);

        /*
         * MapReduce will now pick the output from the first job, as the input
         * to the second job now
         */
        FileInputFormat.addInputPath(jobAvg, new Path(out, "pagecount"));

        /*
         * MapReduce will now pick the second argument from command line, as the
         * output directory and create the second mapreduce job's output file as
         * average in the same output directory as the first job.
         */
        FileOutputFormat.setOutputPath(jobAvg, new Path(out, "average"));
        if (!jobAvg.waitForCompletion(true)) {
            System.exit(1);
        }
    }
}

```

WikiSiteMapper.java:

```

import java.io.IOException;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;

public class WikiSiteMapper extends Mapper<LongWritable, Text, LongWritable, Text> {

```

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

```
    /*
     * All the strings are converted to lower case first, so that all the
     * cases can be treated the same.
     */
    String s = value.toString().toLowerCase();
    int views = 0;
    String[] split = null;
    String firstCol = null;
    System.out.println("In WikiSiteMapper!");

    // Splitting the file by each line, using \n escape sequence
    for (String line : s.split("\\n")) {
        if (line.length() > 0) {
            // Splitting the line further by the whitespaces
            split = line.split(" ");

            // The first column has the language.
            firstCol = split[0];

            /*
             * As Wikipedia projects don't contain a period and a
             * following character, an if-condition is used to check if
             * the first column split has period or not. Only if period
             * symbol is not present, it will go inside the if-loop, else it
             * will break and return to the for-loop again.
             */
            if (!firstCol.contains(".")) {
                // The third column split has the page view counts
                views = Integer.parseInt(split[2]);
            }
        }
    }
}
```



```

        /*
        * As we don't need the fourth column, which is
        * the size of the wikipedia page, replacing the
        * third column and the whitespace before that
        * with an empty character.
        */
        line = line.replace(" " + split[3], "");

        /*
        * The page count and the entire line are written to
        * the main driver, so that the WikiSiteReducer
        * class can process the mapper's output, reduce
        * and count the total number of page hits per site.
        */
        context.write(new LongWritable.views), new Text(line));
    }
}
}
}
}
}
}
}

```

WikiSiteReducer.java:

```

import java.io.IOException;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Reducer;

public class WikiSiteReducer extends
    Reducer<LongWritable, Text, LongWritable, Text> {
    public void reduce(Iterable<LongWritable> values, Text key, Context context)
        throws IOException, InterruptedException {

        int viewCount = 0;
    }
}

```

```

System.out.println("In Wiki Site Reducer!");

/*
 * The total view counts per wikipedia site are counted, that were
 * generated by the Mapper
 */
for (LongWritable views : values) {
    viewCount += views.get();
}

// The view count and the entire line are written to the main class.
context.write(new LongWritable(viewCount), key);
}
}

```

AverageMapper.java:

```

import java.io.IOException;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;

public class AverageMapper extends
    Mapper<LongWritable, Text, Text, LongWritable> {
    public void map(LongWritable key, Text value, Context context)
        throws IOException, InterruptedException {
        String s = value.toString().toLowerCase();
        String[] split = null;
        String language = null;
        long views = 0;

        // Splitting the output from first job by lines, using \n escape sequence.
        for (String word : s.split("\n")) {
            if (word.length() > 0) {
                /*

```

```

        * Splitting the lines further by whitespaces. Using \\s+
        * helps to remove more than one whitespace between
        * two columns
        */
        split = word.split("\\s+");

        // Assigning the third column, which is the page count
        views = Long.parseLong(split[3]);

        // Assigning the second column, which is the language
        language = split[1];

        /*
        * Writing language and page count as the (key, value)
        * pair back to the main class.
        */
        context.write(new Text(language), new LongWritable(views));
    }
}
}
}
}

```

AverageReducer.java:

```

import java.io.IOException;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Reducer;

public class AverageReducer extends
    Reducer<Text, LongWritable, Text, LongWritable> {
    public void reduce(Text key, Iterable<LongWritable> values, Context context)
        throws IOException, InterruptedException {
        long viewcount = 0;
    }
}

```

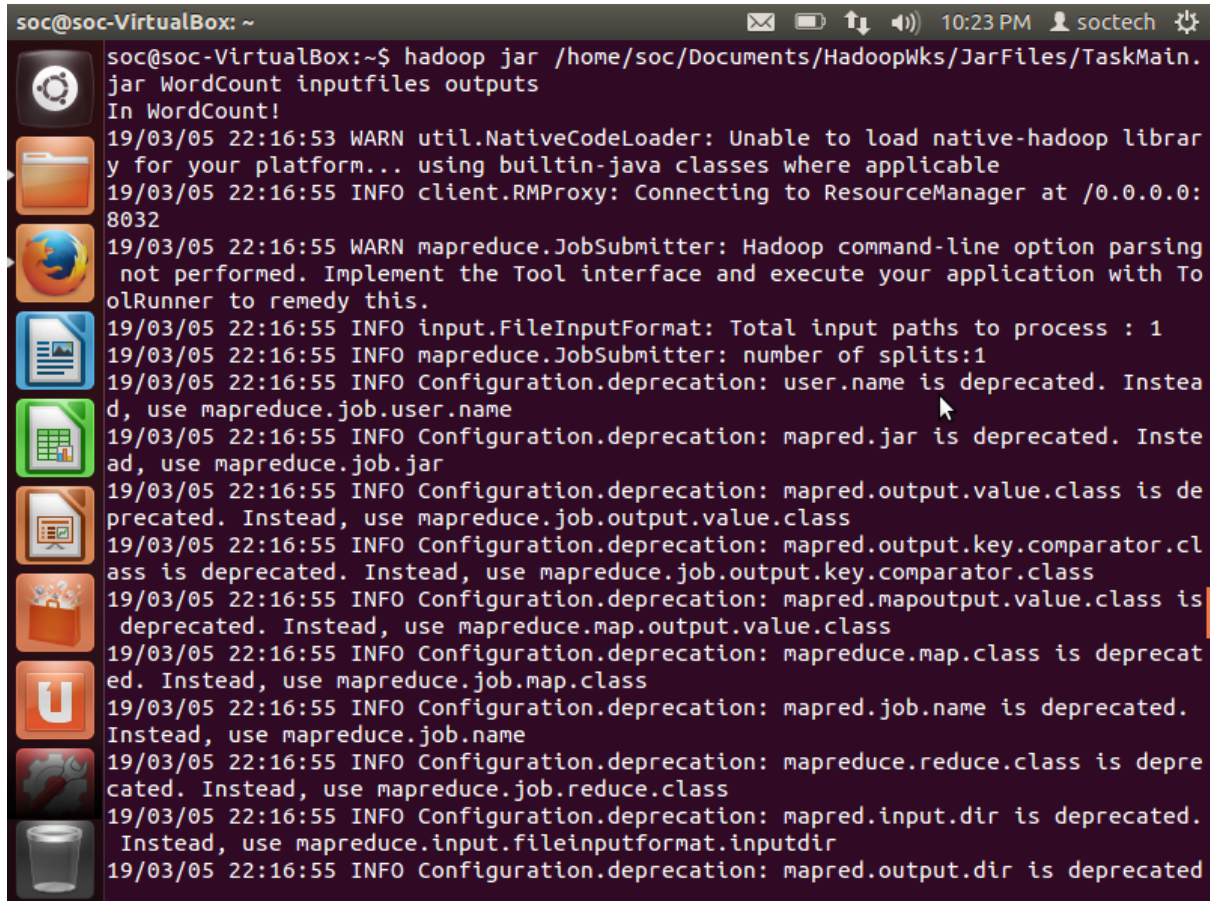
```
long counter = 0;
long avg = 0;
System.out.println("In Average Reducer now!");
for (LongWritable value : values) {
    // The page count is incrementally added in the variable viewcount
    viewcount += value.get();

    //A variable counter is used to incrementally count by 1,
    // every time this iterator runs.
    counter++;
}
//Average is calculated by dividing the sum of viewcounts by counter
avg = viewcount / counter;
context.write(key, new LongWritable(viewcount));
}
}
```

Outputs:

The command used to run the map reduce program is:

`hadoop jar location/jar-name main-class-name input-file output-file`



```
soc@soc-VirtualBox: ~  
soc@soc-VirtualBox:~$ hadoop jar /home/soc/Documents/HadoopWks/JarFiles/TaskMain.jar WordCount inputfiles outputs  
In WordCount!  
19/03/05 22:16:53 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable  
19/03/05 22:16:55 INFO client.RMProxy: Connecting to ResourceManager at /0.0.0.0:8032  
19/03/05 22:16:55 WARN mapreduce.JobSubmitter: Hadoop command-line option parsing not performed. Implement the Tool interface and execute your application with ToolRunner to remedy this.  
19/03/05 22:16:55 INFO input.FileInputFormat: Total input paths to process : 1  
19/03/05 22:16:55 INFO mapreduce.JobSubmitter: number of splits:1  
19/03/05 22:16:55 INFO Configuration.deprecation: user.name is deprecated. Instead, use mapreduce.job.user.name  
19/03/05 22:16:55 INFO Configuration.deprecation: mapred.jar is deprecated. Instead, use mapreduce.job.jar  
19/03/05 22:16:55 INFO Configuration.deprecation: mapred.output.value.class is deprecated. Instead, use mapreduce.job.output.value.class  
19/03/05 22:16:55 INFO Configuration.deprecation: mapred.output.key.comparator.class is deprecated. Instead, use mapreduce.job.output.key.comparator.class  
19/03/05 22:16:55 INFO Configuration.deprecation: mapred.mapoutput.value.class is deprecated. Instead, use mapreduce.map.output.value.class  
19/03/05 22:16:55 INFO Configuration.deprecation: mapreduce.map.class is deprecated. Instead, use mapreduce.job.map.class  
19/03/05 22:16:55 INFO Configuration.deprecation: mapred.job.name is deprecated. Instead, use mapreduce.job.name  
19/03/05 22:16:55 INFO Configuration.deprecation: mapreduce.reduce.class is deprecated. Instead, use mapreduce.job.reduce.class  
19/03/05 22:16:55 INFO Configuration.deprecation: mapred.input.dir is deprecated. Instead, use mapreduce.input.fileinputformat.inputdir  
19/03/05 22:16:55 INFO Configuration.deprecation: mapred.output.dir is deprecated
```

The first MapReduce job is running as shown below:

```
soc@soc-VirtualBox: ~  
mode : false  
19/03/05 22:17:07 INFO mapreduce.Job: map 0% reduce 0%  
19/03/05 22:17:20 INFO mapreduce.Job: map 16% reduce 0%  
19/03/05 22:17:23 INFO mapreduce.Job: map 24% reduce 0%  
19/03/05 22:17:26 INFO mapreduce.Job: map 33% reduce 0%  
19/03/05 22:17:29 INFO mapreduce.Job: map 40% reduce 0%  
19/03/05 22:17:32 INFO mapreduce.Job: map 45% reduce 0%  
19/03/05 22:17:35 INFO mapreduce.Job: map 52% reduce 0%  
19/03/05 22:17:38 INFO mapreduce.Job: map 62% reduce 0%  
19/03/05 22:17:41 INFO mapreduce.Job: map 70% reduce 0%  
19/03/05 22:17:43 INFO mapreduce.Job: map 100% reduce 0%  
19/03/05 22:18:00 INFO mapreduce.Job: map 100% reduce 100%  
19/03/05 22:18:01 INFO mapreduce.Job: Job job_1551823174284_0008 completed successfully  
19/03/05 22:18:01 INFO mapreduce.Job: Counters: 43  
    File System Counters  
        FILE: Number of bytes read=196366694  
        FILE: Number of bytes written=295475823  
        FILE: Number of read operations=0  
        FILE: Number of large read operations=0  
        FILE: Number of write operations=0  
        HDFS: Number of bytes read=103601255  
        HDFS: Number of bytes written=77647023  
        HDFS: Number of read operations=6  
        HDFS: Number of large read operations=0  
        HDFS: Number of write operations=2  
    Job Counters  
        Launched map tasks=1  
        Launched reduce tasks=1  
        Data-local map tasks=1  
        Total time spent by all maps in occupied slots (ms)=35158
```

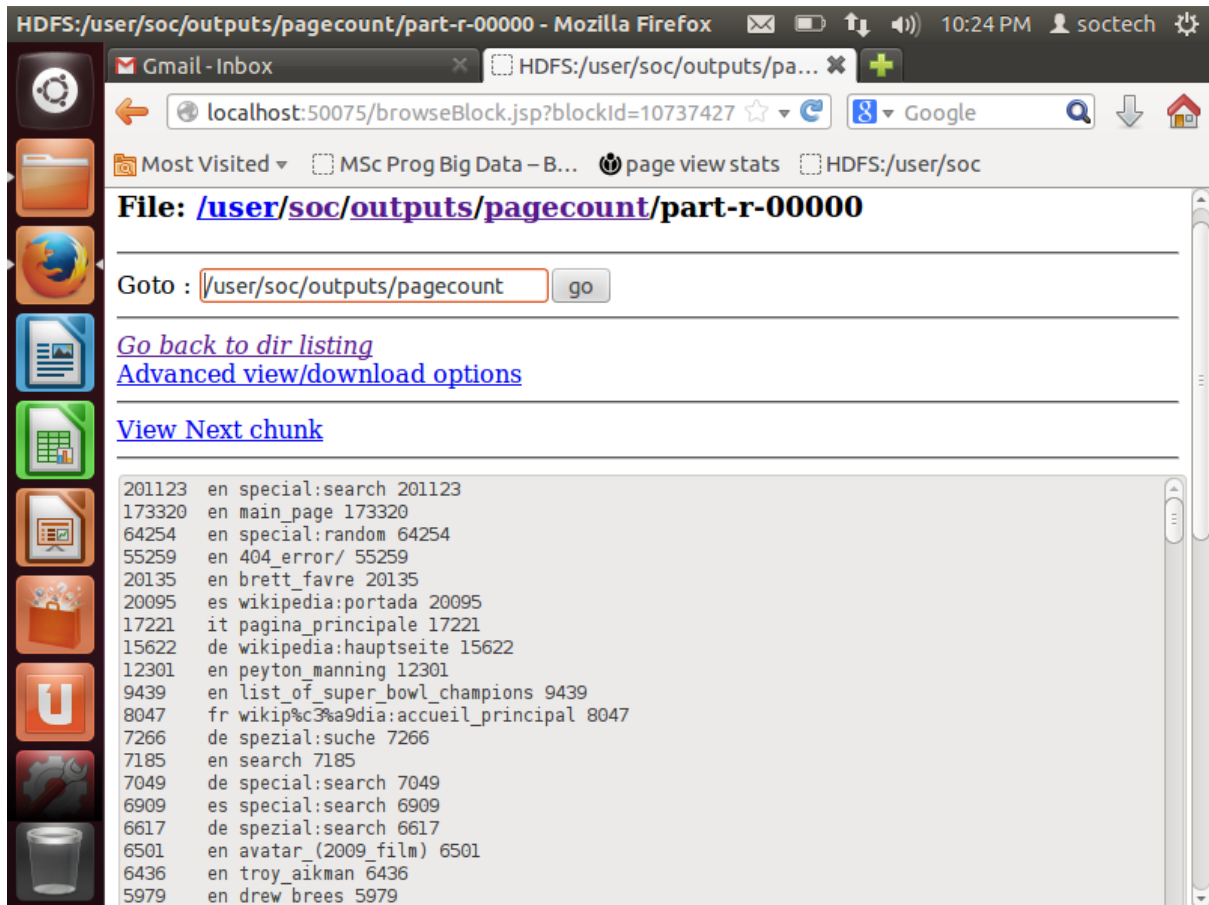
The second mapreduce job has started and generated the output successfully:

```
soc@soc-VirtualBox: ~  
File Input Format Counters  
  Bytes Read=103601130  
File Output Format Counters  
  Bytes Written=77647023  
19/03/05 22:18:01 INFO client.RMProxy: Connecting to ResourceManager at /0.0.0.0:8032  
19/03/05 22:18:01 WARN mapreduce.JobSubmitter: Hadoop command-line option parsing not performed. Implement the Tool interface and execute your application with ToolRunner to remedy this.  
19/03/05 22:18:01 INFO input.FileInputFormat: Total input paths to process : 1  
19/03/05 22:18:01 INFO mapreduce.JobSubmitter: number of splits:1  
19/03/05 22:18:01 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1551823174284_0009  
19/03/05 22:18:01 INFO impl.YarnClientImpl: Submitted application application_1551823174284_0009 to ResourceManager at /0.0.0.0:8032  
19/03/05 22:18:02 INFO mapreduce.Job: The url to track the job: http://soc-VirtualBox:8088/proxy/application_1551823174284_0009/  
19/03/05 22:18:02 INFO mapreduce.Job: Running job: job_1551823174284_0009  
19/03/05 22:18:19 INFO mapreduce.Job: Job job_1551823174284_0009 running in uber mode : false  
19/03/05 22:18:19 INFO mapreduce.Job:  map 0% reduce 0%  
19/03/05 22:18:33 INFO mapreduce.Job:  map 17% reduce 0%  
19/03/05 22:18:36 INFO mapreduce.Job:  map 32% reduce 0%  
19/03/05 22:18:39 INFO mapreduce.Job:  map 45% reduce 0%  
19/03/05 22:18:42 INFO mapreduce.Job:  map 61% reduce 0%  
19/03/05 22:18:45 INFO mapreduce.Job:  map 100% reduce 0%  
19/03/05 22:18:58 INFO mapreduce.Job:  map 100% reduce 100%  
19/03/05 22:18:58 INFO mapreduce.Job: Job job_1551823174284_0009 completed successfully  
19/03/05 22:18:58 INFO mapreduce.Job: Counters: 43  
File System Counters
```

Ran successfully:

```
soc@soc-VirtualBox: ~  
Map output bytes=28427277  
Map output materialized bytes=33590451  
Input split bytes=126  
Combine input records=0  
Combine output records=0  
Reduce input groups=173  
Reduce shuffle bytes=33590451  
Reduce input records=2581584  
Reduce output records=173  
Spilled Records=5163168  
Shuffled Maps =1  
Failed Shuffles=0  
Merged Map outputs=1  
GC time elapsed (ms)=444  
CPU time spent (ms)=20750  
Physical memory (bytes) snapshot=366120960  
Virtual memory (bytes) snapshot=2258427904  
Total committed heap usage (bytes)=222101504  
Shuffle Errors  
BAD_ID=0  
CONNECTION=0  
IO_ERROR=0  
WRONG_LENGTH=0  
WRONG_MAP=0  
WRONG_REDUCE=0  
File Input Format Counters  
Bytes Read=77647023  
File Output Format Counters  
Bytes Written=961  
soc@soc-VirtualBox:~$
```


Output of First task:



HDFS:/user/soc/outputs/pagecount/part-r-00000 - Mozilla Firefox 10:24 PM soctech

Gmail - Inbox HDFS:/user/soc/outputs/pa... +

localhost:50075/browseBlock.jsp?blockId=10737427 Google

Most Visited MSc Prog Big Data - B... page view stats HDFS:/user/soc

File: /user/soc/outputs/pagecount/part-r-00000

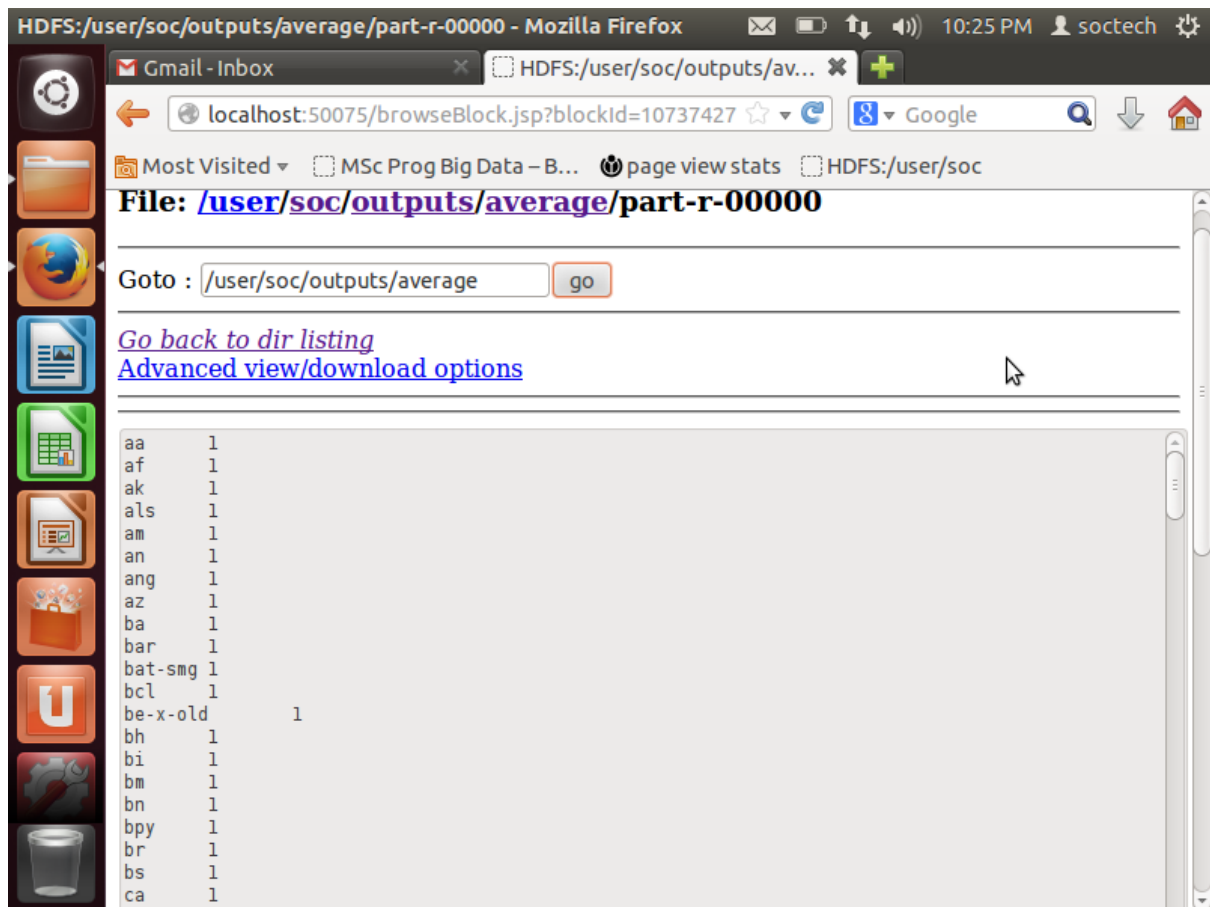
Goto : go

[Go back to dir listing](#)
[Advanced view/download options](#)

[View Next chunk](#)

201123	en	special:search	201123
173320	en	main_page	173320
64254	en	special:random	64254
55259	en	404_error/	55259
20135	en	brett_favre	20135
20095	es	wikipedia:portada	20095
17221	it	pagina_principale	17221
15622	de	wikipedia:hauptseite	15622
12301	en	peyton_manning	12301
9439	en	list_of_super_bowl_champions	9439
8047	fr	wikip%a9dia:accueil_principal	8047
7266	de	spezial:suche	7266
7185	en	search	7185
7049	de	special:search	7049
6909	es	special:search	6909
6617	de	spezial:search	6617
6501	en	avatar_(2009_film)	6501
6436	en	troy_aikman	6436
5979	en	drew_brees	5979

Output of second task:



As we can see, both the output files are in the same output directory “outputs”, as a result of chained mapreduce jobs.

The full output files are attached along with this assignment submission.