Big Data Platform Assignment 1

Samou Timothée

May 14, 2017

Abstract

This assignment is a basic application of Hadoop framework in JAVA based on several text corpus. We are going to implement simple applications, TF-IDF and Page Rank.
Github : https://github.com/timothee001/HM1_BDP_SAMOU

Contents

1 What helped me for this homework

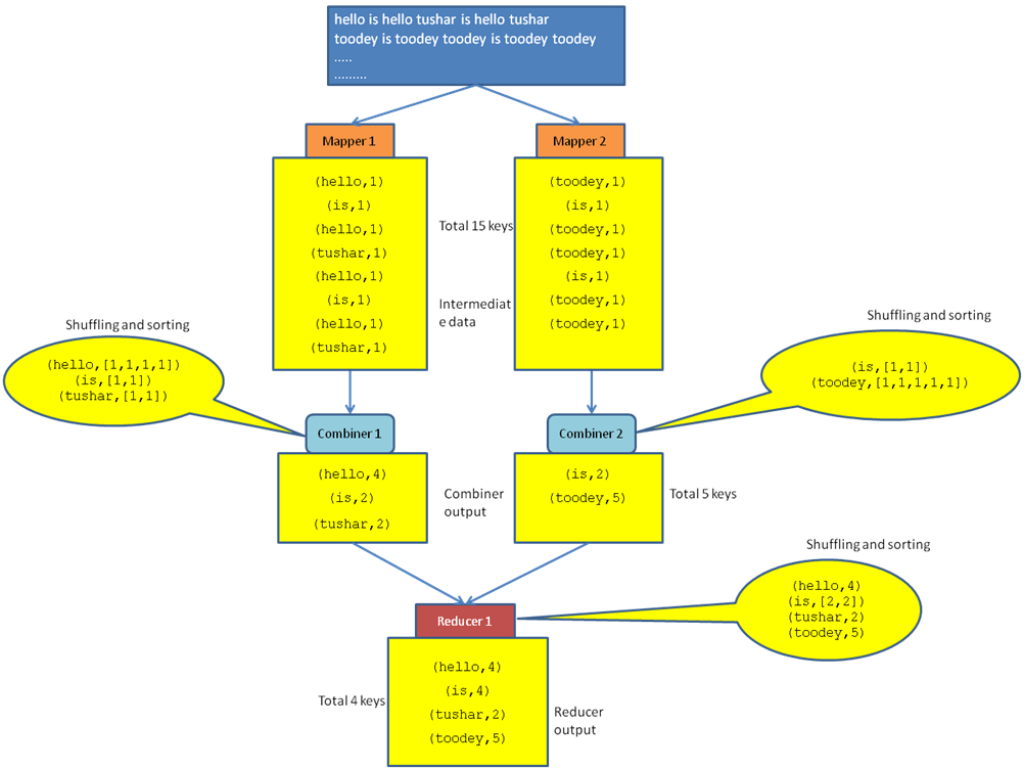


Figure 1: How works Hadoop

In order to do every part of this assignment, I used this comprehensive figure. My methodology was to start from the expected result and building the Hadoop code by back-propagating to the input.

2 Displaying the content of a CSV file

This part is a very simple job where we print target result at every line, we split the string by the ";" and we get the corresponding fields we need.

Listing 1: Question 2.7

```
   package twoseven;

import java.io.IOException;
import java.util.Arrays;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.conf.Configured;
import org.apache.hadoop.fs.FileSystem;
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;
import org.apache.hadoop.util.Tool;
import org.apache.hadoop.util.ToolRunner;

public class Arbre extends Configured implements Tool {
   public static void main(String[] args) throws Exception {
      System.out.println(Arrays.toString(args));
      int res = ToolRunner.run(new Configuration(), new Arbre(), args);

      System.exit(res);
   }

   @Override
   public int run(String[] args) throws Exception {
      System.out.println(Arrays.toString(args));
      Configuration configuration = this.getConf();

      Job job = new Job(configuration, "WordCount");
      job.setNumReduceTasks(2);
      job.setJarByClass(Arbre.class);
      job.setOutputKeyClass(Text.class);
      job.setOutputValueClass(IntWritable.class);

      job.setMapperClass(Map.class);
      job.setReducerClass(Reduce.class);

      job.setInputFormatClass(TextInputFormat.class);
      job.setOutputFormatClass(TextOutputFormat.class);

      FileInputFormat.addInputPath(job, new Path("input"));
      FileOutputFormat.setOutputPath(job, new Path("outputArbre"));
      FileSystem hdfs = FileSystem.get(getConf());
     if (hdfs.exists(new Path("outputArbre")))
         hdfs.delete(new Path("outputArbre"), true);
      job.waitForCompletion(true);

      return 0;
   }

   public static class Map extends Mapper<LongWritable, Text, Text, IntWritable> {
      private final static IntWritable ONE = new IntWritable(1);
      private Text word = new Text();

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

        String[] tree= value.toString().split(";");
        if(key.get()>0&&!tree[5].isEmpty())
           System.out.println("Nom : " + tree[2] + " Age :
               "+(2017-Integer.parseInt(tree[5]))+ " Hauteur : "+tree[6]);




      }
   }

   public static class Reduce extends Reducer<Text, IntWritable, Text, IntWritable> {
      @Override
      public void reduce(Text key, Iterable<IntWritable> values, Context context)
             throws IOException, InterruptedException {
```

```
    }
  }
}
```

3 Displaying the content of a compact file

It is also a simple job with no high complexity

Listing 2: Question 2.8

```
      package twoeight;

import java.io.IOException;
import java.util.Arrays;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.conf.Configured;
import org.apache.hadoop.fs.FileSystem;
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;
import org.apache.hadoop.util.Tool;
import org.apache.hadoop.util.ToolRunner;

public class Compact extends Configured implements Tool {
   public static void main(String[] args) throws Exception {
      System.out.println(Arrays.toString(args));
      int res = ToolRunner.run(new Configuration(), new Compact(), args);

      System.exit(res);
   }

   @Override
   public int run(String[] args) throws Exception {
      System.out.println(Arrays.toString(args));
      Configuration configuration = this.getConf();

      Job job = new Job(configuration, "Compact");
      job.setNumReduceTasks(2);
      job.setJarByClass(Compact.class);
      job.setOutputKeyClass(Text.class);
      job.setOutputValueClass(IntWritable.class);

      job.setMapperClass(Map.class);
      job.setReducerClass(Reduce.class);

      job.setInputFormatClass(TextInputFormat.class);
      job.setOutputFormatClass(TextOutputFormat.class);

      FileInputFormat.addInputPath(job, new Path("input2"));
      FileOutputFormat.setOutputPath(job, new Path("outputCompact"));
      FileSystem hdfs = FileSystem.get(getConf());
     if (hdfs.exists(new Path("outputCompact")))
```

```
        hdfs.delete(new Path("outputCompact"), true);
      job.waitForCompletion(true);

      return 0;
   }

   public static class Map extends Mapper<LongWritable, Text, Text, IntWritable> {
      private final static IntWritable ONE = new IntWritable(1);
      private Text word = new Text();

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

        String line= value.toString();
        String stationName="";
        String FIPS = "";
         String altitude = "";

        if(line.length()>41){
           stationName = line.substring(13, 42).trim();
        }

        if(line.length()>44){
           FIPS = line.substring(43, 45).trim();
        }

        if(line.length()>80){
           altitude = line.substring(74, 81).trim();
        }



        if(key.get()>21)
           System.out.println("Station Name : " + stationName + " FIPS : "+FIPS + "
                altitude : "+ altitude);




      }
   }

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

      }
   }
}
```

4 TF-IDF

We are going to implements tree parts in order to do the TF-IDF as seen on the following figure: The wordcount, the wordcount per doc and the docCount per word.

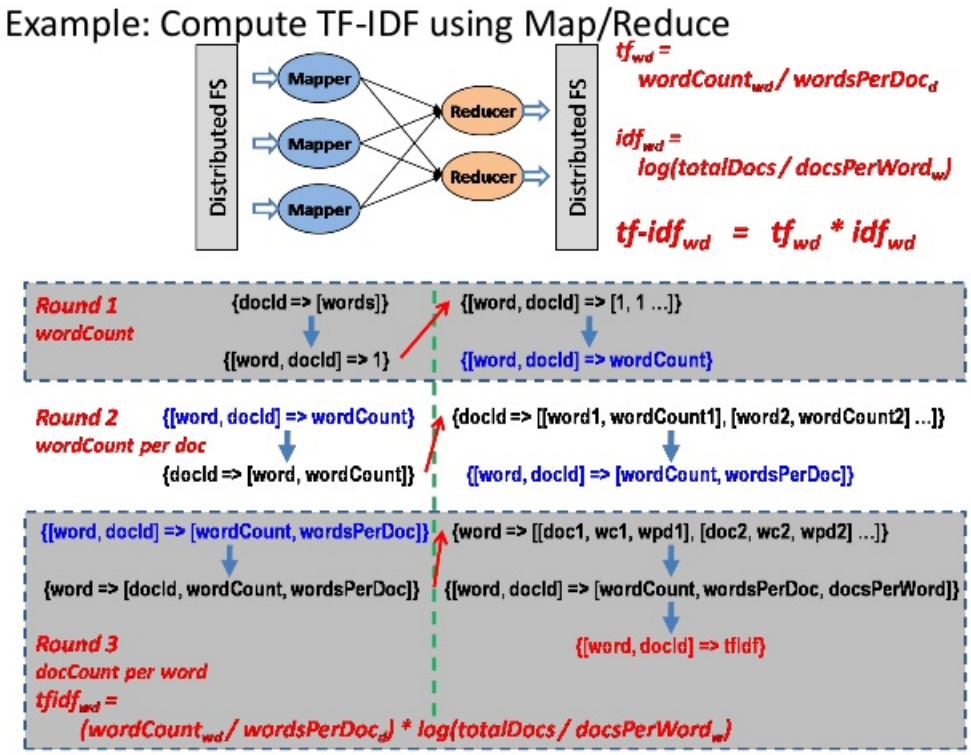


Figure 2: TF-IDF process

```
defoe-robinson-103.txt  Documents  enterprise-deployment.json  lib                                  parcels       Templates
[cloudera@quickstart ~]$ wget http://www.textfiles.com/etext/FICTION/callwild
--2017-05-14 08:19:40--  http://www.textfiles.com/etext/FICTION/callwild
Resolving www.textfiles.com... 208.86.224.90
Connecting to www.textfiles.com|208.86.224.90|:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 175998 (172K)
Saving to: “callwild”

100%[======================================================================================>] 175,998      349K/s   in 0.5s

2017-05-14 08:19:41 (349 KB/s) - “callwild” saved [175998/175998]

[cloudera@quickstart ~]$ hdfs dfs -copyFromLocal callwild.txt inputtfidf
copyFromLocal: `callwild.txt': No such file or directory
[cloudera@quickstart ~]$ hdfs dfs -copyFromLocal callwild inputtfidf
[cloudera@quickstart ~]$ cd Desktop/
[cloudera@quickstart Desktop]$ ls
Eclipse.desktop     HadoopSAMOUHM2-master.zip  hm1_bdp.zip          HM.tar.gz         soc-Epinions1.txt
Enterprise.desktop  HM                         HM_BDP_Final.tar.gz  Kerberos.desktop
Express.desktop     hm1bdp.jar                 HM_final_2.tar.gz    Parcels.desktop
[cloudera@quickstart Desktop]$ hadoop jar hm1bdp.jar TFIDF.WordCount
[]
[]
17/05/14 08:21:45 INFO client.RMProxy: Connecting to ResourceManager at /0.0.0.0:8032
17/05/14 08:21:47 INFO input.FileInputFormat: Total input paths to process : 2
17/05/14 08:21:48 INFO mapreduce.JobSubmitter: number of splits:2
17/05/14 08:21:48 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1494774628926_0001
17/05/14 08:21:49 INFO impl.YarnClientImpl: Submitted application application_1494774628926_0001
17/05/14 08:21:49 INFO mapreduce.Job: The url to track the job: http://quickstart.cloudera:8088/proxy/application_149477462
8926_0001/
17/05/14 08:21:49 INFO mapreduce.Job: Running job: job_1494774628926_0001
17/05/14 08:22:08 INFO mapreduce.Job: Job job_1494774628926_0001 running in uber mode : false
17/05/14 08:22:08 INFO mapreduce.Job:  map 0% reduce 0%
```

Figure 3: Launching the Job in HDFS

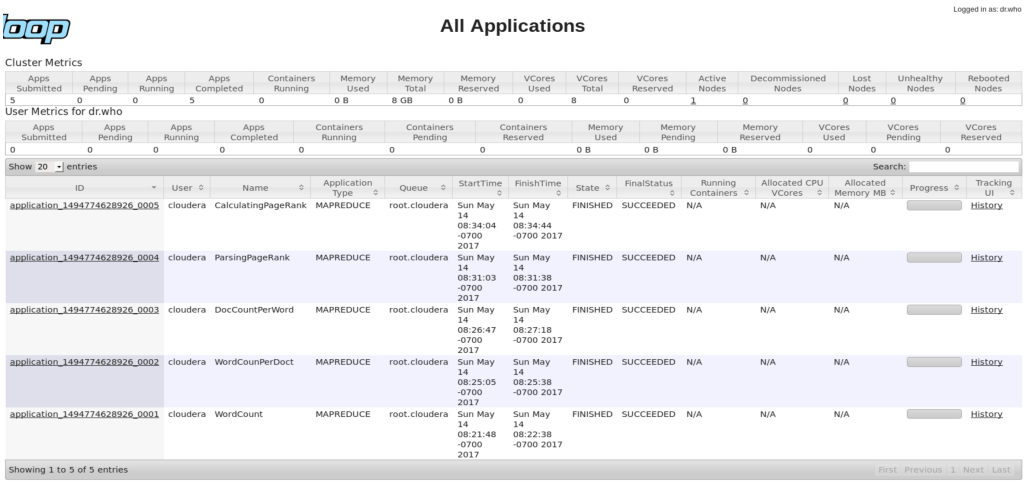


Figure 4: All jobs in resource manager in HDFS

4.1 Word Count

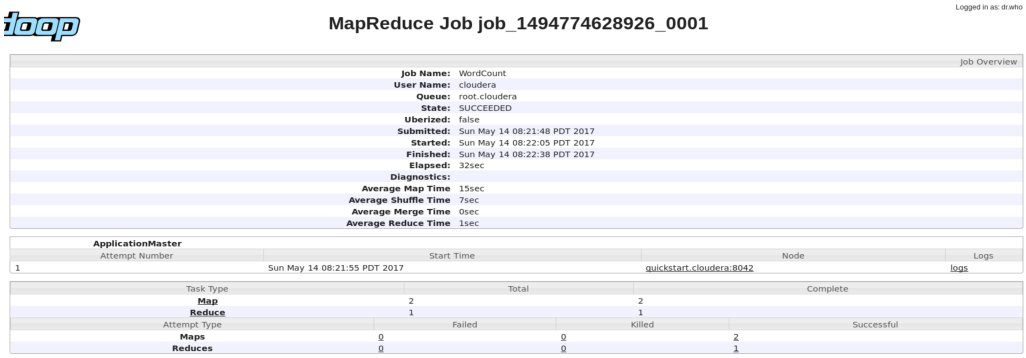


Figure 5: WordCount Job

Listing 3: Wordcount code

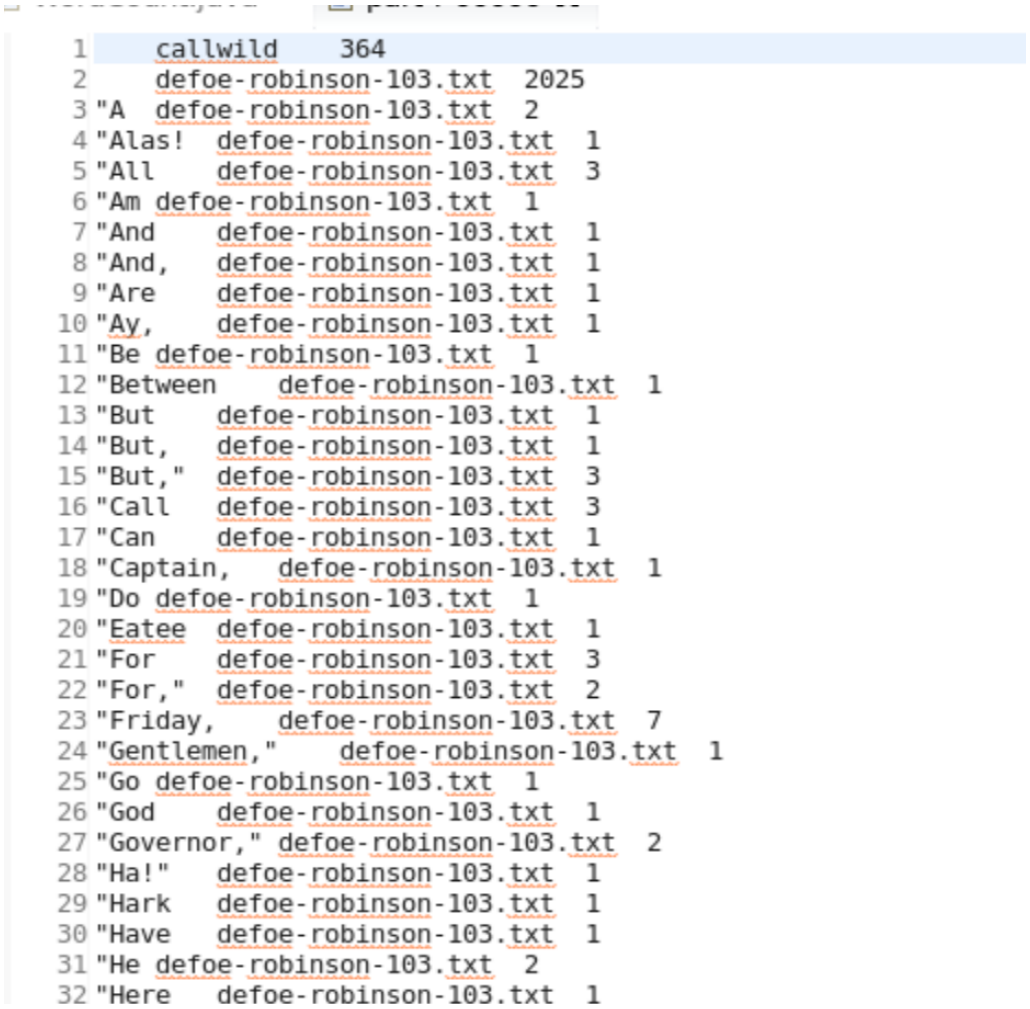
```
public static class Map extends Mapper<LongWritable, Text, Text, IntWritable> {
    private final static IntWritable ONE = new IntWritable(1);


    @Override
    public void map(LongWritable key, Text value, Context context)
           throws IOException, InterruptedException {
     String filename = ((FileSplit)
         context.getInputSplit()).getPath().getName().toString();
       for (String token: value.toString().split("\\s+")) {

          context.write(new Text(token+"\t"+filename), ONE);
       }
    }
  }

  public static class Reduce extends Reducer<Text, IntWritable, Text, IntWritable> {
    @Override
    public void reduce(Text key, Iterable<IntWritable> values, Context context)
           throws IOException, InterruptedException {
       int sum = 0;
       for (IntWritable val : values) {
          sum += val.get();
```

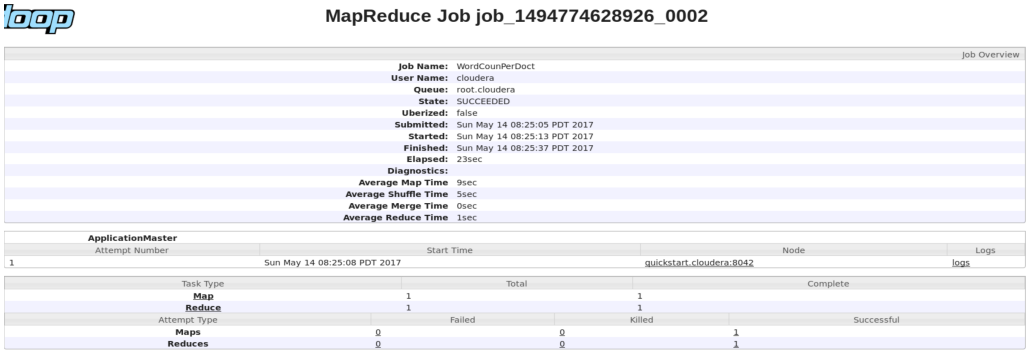
```
        }
        System.out.println(key+" : "+sum);
        context.write(key, new IntWritable(sum));
    }
}
```



```
callwild    364
defoe-robinson-103.txt  2025
"A  defoe-robinson-103.txt  2
"Alas!  defoe-robinson-103.txt  1
"All    defoe-robinson-103.txt  3
"Am defoe-robinson-103.txt  1
"And    defoe-robinson-103.txt  1
"And,   defoe-robinson-103.txt  1
"Are    defoe-robinson-103.txt  1
"Ay,    defoe-robinson-103.txt  1
"Be defoe-robinson-103.txt  1
"Between    defoe-robinson-103.txt  1
"But    defoe-robinson-103.txt  1
"But,   defoe-robinson-103.txt  1
"But,"  defoe-robinson-103.txt  3
"Call   defoe-robinson-103.txt  3
"Can    defoe-robinson-103.txt  1
"Captain,   defoe-robinson-103.txt  1
"Do defoe-robinson-103.txt  1
"Eatee  defoe-robinson-103.txt  1
"For    defoe-robinson-103.txt  3
"For,"  defoe-robinson-103.txt  2
"Friday,    defoe-robinson-103.txt  7
"Gentlemen,"    defoe-robinson-103.txt  1
"Go defoe-robinson-103.txt  1
"God    defoe-robinson-103.txt  1
"Governor," defoe-robinson-103.txt  2
"Ha!"   defoe-robinson-103.txt  1
"Hark   defoe-robinson-103.txt  1
"Have   defoe-robinson-103.txt  1
"He defoe-robinson-103.txt  2
"Here   defoe-robinson-103.txt  1
```

Figure 6: WordCount result

4.2 Word Count per doc



MapReduce Job job_1494774628926_0002

Job Overview

Job Name:	WordCounPerDoct
User Name:	cloudera
Queue:	root.cloudera
State:	SUCCEEDED
Uberized:	false
Submitted:	Sun May 14 08:25:05 PDT 2017
Started:	Sun May 14 08:25:13 PDT 2017
Finished:	Sun May 14 08:25:37 PDT 2017
Elapsed:	23sec
Diagnostics:	
Average Map Time	9sec
Average Shuffle Time	5sec
Average Merge Time	0sec
Average Reduce Time	1sec

ApplicationMaster

Attempt Number	Start Time	Node	Logs
1	Sun May 14 08:25:08 PDT 2017	quickstart.cloudera:8042	logs

Task Type	Total	Complete
Map	1	1
Reduce	1	1

Attempt Type	Failed	Killed	Successful
Maps	0	0	1
Reduces	0	0	1

Figure 7: WordCount Per Doc Job

Listing 4: Wordcount per doc code

```
public static class Map extends Mapper<LongWritable, Text, Text, Text> {
     private final static IntWritable ONE = new IntWritable(1);


     @Override
     public void map(LongWritable key, Text value, Context context)
            throws IOException, InterruptedException {
       String[] words = value.toString().split("\t");
       /* System.out.println(words[0]);
        System.out.println(words[1]);
        System.out.println(words[2]);
*/

        context.write(new Text(words[1]), new Text(words[0]+"\t"+words[2]));
     }
  }

  public static class Reduce extends Reducer<Text, Text, Text, Text> {
     @Override
     public void reduce(Text key, Iterable<Text> values, Context context)
            throws IOException, InterruptedException {

       HashMap<String, String> hm = new HashMap<String,String>();
       int countWordsPerDoc = 0;

       for (Text val : values) {
          String[] words = val.toString().split("\t");

          //System.out.println(words[0]);
          int wordCount = Integer.parseInt(words[1]);
          hm.put(words[0]+"\t"+key.toString(), ""+wordCount);

          countWordsPerDoc +=wordCount;

           }


       Set set2 = hm.entrySet();
         Iterator iterator2 = set2.iterator();
         while(iterator2.hasNext()) {
             Entry mentry2 = (Entry)iterator2.next();
            System.out.println(mentry2.getKey() +" : " +
                mentry2.getValue()+"\t"+countWordsPerDoc);
             context.write(new Text(mentry2.getKey().toString()), new
                 Text(mentry2.getValue()+"\t"+countWordsPerDoc));

          }
     }
  }
```

```
1  geepole,     callwild    1   32142
2  bone,    callwild    1   32142
3  Dawson   callwild    3   32142
4  stirrings    callwild    1   32142
5  toil.    callwild    2   32142
6  unaided,     callwild    1   32142
7  bony     callwild    1   32142
8  that);   callwild    1   32142
9  moved.   callwild    1   32142
10 could    callwild    78  32142
11 business.    callwild    1   32142
12 half     callwild    34  32142
13 danger   callwild    2   32142
14 river,   callwild    2   32142
15 wolf     callwild    17  32142
16 thin,    callwild    1   32142
17 Alaska.'     callwild    1   32142
18 overhead,    callwild    1   32142
19 mining,  callwild    1   32142
20 demon.   callwild    1   32142
21 moments,     callwild    2   32142
22 hands    callwild    9   32142
23 ran.     callwild    1   32142
24 adventures   callwild    1   32142
25 progress     callwild    2   32142
26 team,    callwild    6   32142
27 contraption callwild     1   32142
28 sun.     callwild    1   32142
29 amongst  callwild    1   32142
30 Sunland  callwild    1   32142
31 faintly  callwild    1   32142
```

Figure 8: Word count per doc Job

4.3 DocCount per word

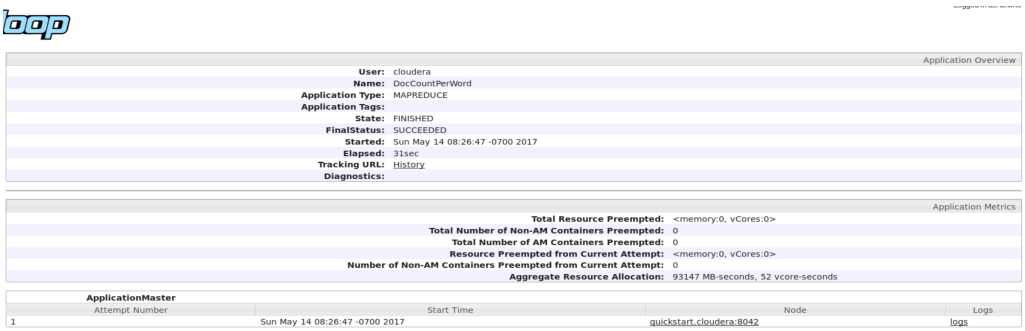


Figure 9: DocCount Per Woc Job

Listing 5: DocCount per word code

```
public static class Map extends Mapper<LongWritable, Text, Text, Text> {
     private final static IntWritable ONE = new IntWritable(1);


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

       String[] words = value.toString().split("\t");

       context.write(new Text(words[0]), new Text(words[1]+"\t"+words[2]+"\t"+words[3]));
     }
   }

   public static class Reduce extends Reducer<Text, Text, Text, Text> {
     TreeSet<Pair> topWordsPair = new TreeSet<>();
     @Override
     public void reduce(Text key, Iterable<Text> values, Context context)
            throws IOException, InterruptedException {

        //System.out.println(key);
        ArrayList<String> words = new ArrayList<String>();
       for (Text val : values) {
        words.add(val.toString());
         // System.out.println(val.toString());
       }

       for(int i=0;i<words.size();i++){

          String [] tmp = words.get(i).split("\t");

          context.write(new Text(key+"\t"+tmp[0]), new
              Text(tmp[1]+"\t"+tmp[2]+"\t"+words.size()));

          System.out.println("tmp[1] : " + tmp[1]+ " tmp[2] : " + tmp[2] + " wordsize : "
              +words.size());

          double tfidf =
              (Double.parseDouble(tmp[1])/Double.parseDouble(tmp[2]))*Math.log(2.0/words.size());

          topWordsPair.add(new Pair(tfidf, key.toString(), tmp[0]));

             if (topWordsPair.size() > 20) {
                topWordsPair.pollFirst();

                //we delete the first pair with the lowest frequency
             }

       }


     }

     protected void cleanup(Context ctxt) throws IOException,InterruptedException {
        //we call this fonction once at the end
        while (!topWordsPair.isEmpty()) {
            Pair pair = topWordsPair.pollLast();
            System.out.println("["+pair.getWord()+"\t"+ pair.getDoc()+"] : " +
                pair.frequency);
        }
     }
```

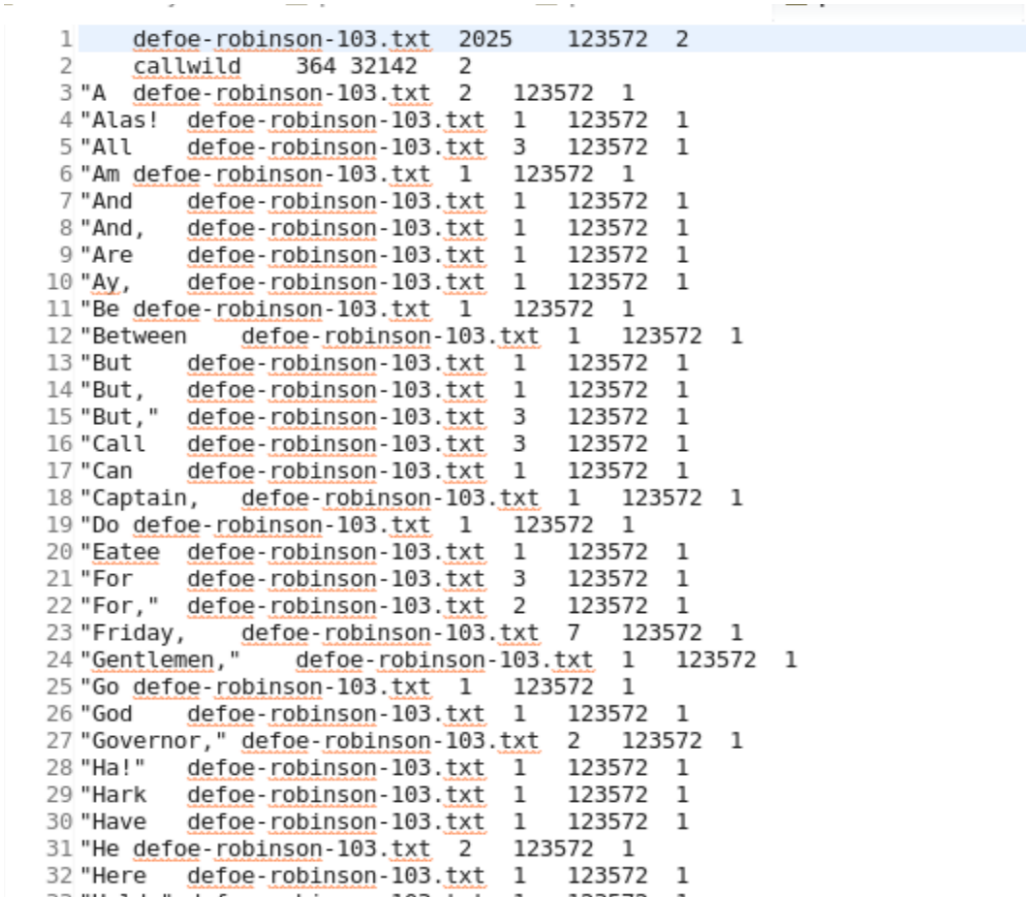


Figure 10: DocCount per word

4.4 Job result

As you saw in the code, we added a real time treeset object with ranked pair. We added a Pair class that implements comparable interface. Then we restrict the size of the list and only keep pairs with highest score.

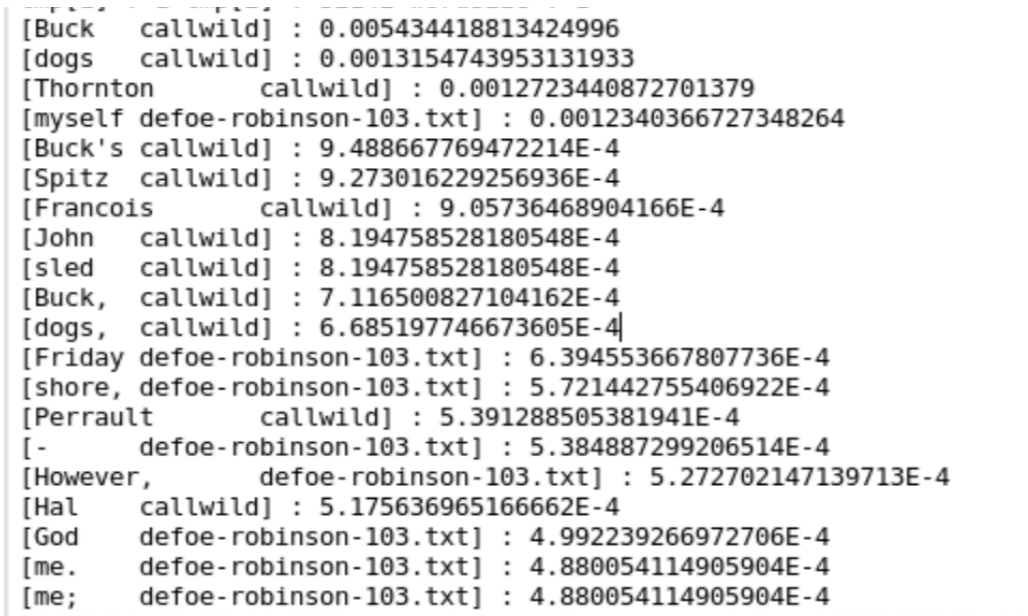


Figure 11: TFIDF with highest score

5 Page Rank

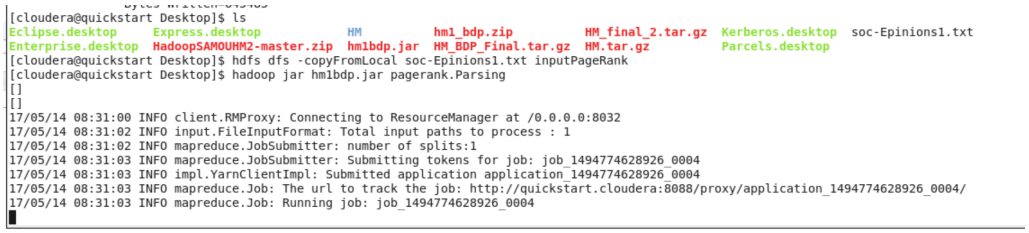


Figure 12: PageRank Job

5.1 Parsing

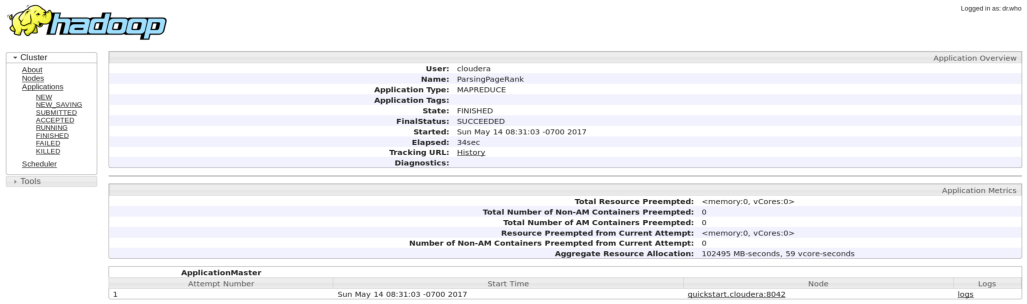


Figure 13: Parsing Job

Listing 6: Parsing code

```
public static class Map extends Mapper<LongWritable, Text, Text, Text> {
     private final static IntWritable ONE = new IntWritable(1);



     @Override
     public void map(LongWritable key, Text value, Context context)
             throws IOException, InterruptedException {
       String [] fromTo = value.toString().split("\t");

      // System.out.println("Map parser " + fromTo[0] + " : " + fromTo[1]);
         context.write(new Text(fromTo[0]), new Text(fromTo[1]));

     }

  }

  public static class Reduce extends Reducer<Text, Text, Text, Text> {

       TreeSet<String> froms = new TreeSet<String>();
       TreeSet<String> tos = new TreeSet<String>();


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

       String LinksToPages = "";
```

```
    froms.add(key.toString());


    for(Text link : values){

       LinksToPages+=link.toString()+",";
       tos.add(link.toString());
    }
    if(LinksToPages.length()>0)
       LinksToPages = LinksToPages.substring(0, LinksToPages.length()-1);

   // System.out.println("Map reducer " + key + " : " + "1.0\t"+LinksToPages);
    context.write(key, new Text("1.0\t"+LinksToPages));

  }

  protected void cleanup(Context context) throws IOException,InterruptedException {
     //we call this fonction once at the end
    System.out.println("froms : "+froms);
    System.out.println("tos : "+tos);
    //context.write(new Text("1"), new Text());


    froms.removeAll(tos);
    System.out.println("tos after sub : "+tos);

    for(String s: froms){
     context.write(new Text(s), new Text("1.0\tnull"));
     System.out.println("Map reducer " + s + " : " + "1.0\tnull");
    }


}
```

```
100 1.0 447,546,550,551,553,555,582,588,597,611,449,2,4,5,6,10,12,19,21,22,26,27,28,30,38,43,48,50,51,55,61,75,76,
1000    1.0 682,639,634,295,195,22,90,135,663,16675,16674,16673,12453,10700,10498,9750,7714,6878,5659,5417,3687,34
10000   1.0 128,64,9646,7590,5514,4040,3984,1719,1619,1304,1179,737,394,141
10001   1.0 7445,24954,395,639,972,2042,3637
10002   1.0 13960,47619,663,143,135,18,735,737,849,852,854,860,909,918,1177,1268,1274,1394,1401,1430,1433,1471,171
10003   1.0 3723,3717,2025,1914,1741,395
10004   1.0 395
10005   1.0 401,395
10006   1.0 395
10007   1.0 655,395
10008   1.0 1561,395
10009   1.0 395
1001    1.0 155,168,195,340,375,492,590,645,846,903,1979,1761,1647,1293,1206,1169,975,8647,2792,2042,19,22,34,74,7
10010   1.0 395,2637
10011   1.0 1008,965,849,581,395,7696,5818,4030,2370,2338,1024,3582
10012   1.0 849,395
10013   1.0 1440,1179,738,395,9504,6144,1719,1516,125,295
10014   1.0 883,3179,3255,3678,4700,5100,22551,395
10015   1.0 2557,1715,395
10016   1.0 395,746
10017   1.0 639,663,849,1712,395
10018   1.0 395
10019   1.0 395
1002    1.0 2323,2222,2215,2065,2030,2008,1975,1753,1574,1274,1206,1177,1011,1006,997,988,976,965,862,849,565,439,
10020   1.0 395,3189
10021   1.0 737,395,11876,5952,2694,1401,1304,1145,1004,849
10022   1.0 18292,395,656,71388,763,1315,1396,1895
10023   1.0 395,737,1740
10024   1.0 3650,395
10025   1.0 8598,395
10026   1.0 395
```

Figure 14: Result of the parsing

5.2 Calculating

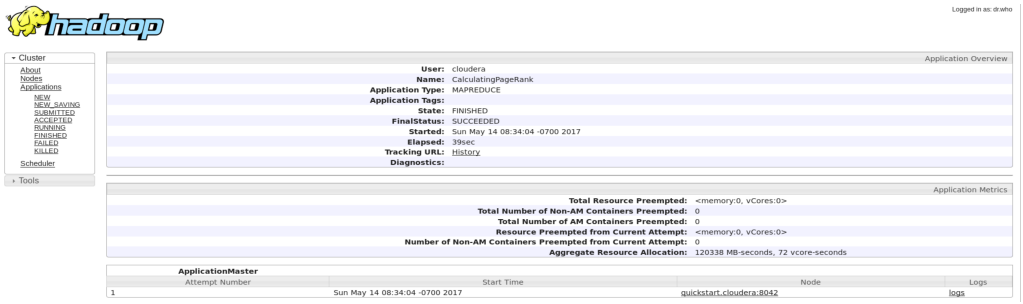


Figure 15: Calculating Job

Listing 7: Calculating code

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

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


      System.out.println(value);

      String [] res = value.toString().split("\t");

      String [] vals = res[2].split(",");

      hist.put(res[0], res[2]);


      for(String s: vals){
         context.write(new Text(s), new Text(res[0]+"\t"+res[1]+"\t"+vals.length));
      }

    }

    protected void cleanup(Context context) throws IOException,InterruptedException {

      //context.write(new Text("C"), new Text());
   }
 }

 public static class Reduce extends Reducer<Text, Text, Text, Text> {
    TreeSet<PageAndRank> topPages = new TreeSet<>();
    @Override
    public void reduce(Text key, Iterable<Text> values, Context context)
            throws IOException, InterruptedException {

      double pageRankScore=(1.0-dumpingFactor);

      System.out.println("key : "+key);

      if(!key.toString().equals("null")){


      for(Text val : values){
        // System.out.println("val : "+val);
         String [] res =val.toString().split("\t");
```

```
        pageRankScore+=dumpingFactor*(Double.parseDouble(res[1])/Double.parseDouble(res[2]));

      }
      System.out.println(key + " : "+pageRankScore);
      context.write(key, new Text(""+pageRankScore+"\t"+hist.get(key.toString())));
      topPages.add(new PageAndRank(pageRankScore,key.toString()));
       if (topPages.size() > 10) {
            topPages.pollFirst();

             //we delete the first pair with the lowest frequency
        }

      }else{
       //  System.out.println("Not null");
        for(Text val : values){

           String [] res =val.toString().split("\t");

           //System.out.println(res[0] + " : "+pageRankScore);
           context.write(new Text(res[0]), new
               Text(""+pageRankScore+"\t"+hist.get(res[0])));
           topPages.add(new PageAndRank(pageRankScore,res[0]));
            if (topPages.size() > 10) {
                topPages.pollFirst();
                   //we delete the first pair with the lowest frequency
               }

        }

      }

    }


    protected void cleanup(Context ctxt) throws IOException,InterruptedException {
       //we call this fonction once at the end
      System.out.println("results pagerank : ");
        while (!topPages.isEmpty()) {
         PageAndRank pr = topPages.pollLast();
           System.out.println(pr.getPage() + " : "+pr.rank);
        }
    }
```

```
100 11.042045254979799  447,546,550,551,553,555,582,588,597,611,449,2,4,5,6,10,12,19,21,22,
1000    1.4597987358285724  682,639,634,295,195,22,90,135,663,16675,16674,16673,12453,10706
10000   1.4159971940397307  128,64,9646,7590,5514,4040,3984,1719,1619,1304,1179,737,394,141
10001   0.35365217061773313 7445,24954,395,639,972,2042,3637
10002   3.1644109686684554  13960,47619,663,143,135,18,735,737,849,852,854,860,909,918,1177
10003   0.938534186209047   3723,3717,2025,1914,1741,395
1001    0.4835745925892303  155,168,195,340,375,492,590,645,846,903,1979,1761,1647,1293,126
10010   1.8541262135922332  395,2637
10011   0.23386287625418062 1008,965,849,581,395,7696,5818,4030,2370,2338,1024,3582
10013   0.1774193548387097  1440,1179,738,395,9504,6144,1719,1516,125,295
10014   0.17207240948813987 883,3179,3255,3678,4700,5100,22551,395
10016   0.15352697095435686 395,746
10017   0.16693611704700478 639,663,849,1712,395
1002    6.35201736778816    2323,2222,2215,2065,2030,2008,1975,1753,1574,1274,1206,1177,101
10021   0.15458748750371512 737,395,11876,5952,2694,1401,1304,1145,1004,849
10022   0.5787289839678995  18292,395,656,71388,763,1315,1396,1895
10023   1.0 395,737,1740
10024   0.3388888888888889  3650,395
10028   1.0 47626,4030,395
10029   3.272709481765055   1401,1621,1712,1914,3090,3523,3655,4028,4421,5304,5933,5952,117
1003    5.333418079549417   49,64,79,83,90,5673,5386,4497,4416,4411,4003,3901,3352,101,2897
10030   2.444677527940115   10555,10567,64,395,19324,18647,11519,1398,3174,737,26048,20411,
10031   0.16808510638297874 997,725,5386,395,3550,1712
10032   1.0666983711805968  395,32279
10035   0.16787235359996555 432,395
10036   2.1012797032027803  21812,7843,47636,32932
```

Figure 16: File after one iteration Job

5.3 Launching the calculating job several times and Results

The PageRunk use several iterations, for each run we use the rank scores from the previous iterations and we run until stabilization. We did 4 runs here to observe a beginning of stable results.

First run :

18 : 312.93555203381777
4415 : 144.14703769619285
737 : 133.42269871965257
790 : 116.03602201059891
1753 : 114.63997163346549
143 : 114.12885559942849
1719 : 113.59237762549027
136 : 99.75254092814379
751 : 99.36994537940271
118 : 86.05181430397894

Second run :

18 : 314.10571690616104
737 : 155.36773600810724
790 : 147.47316281421982
143 : 135.76601999083752
1719 : 134.68609840717656
136 : 125.72835084417396
118 : 111.7858345638847
4415 : 92.60938757913095
1179 : 85.93071484849997
1619 : 85.66330405624858

Third run :

18 : 237.42478262384276
737 : 145.9147551376583
1719 : 116.32879723765485
790 : 113.88729083809942
118 : 111.23600582512132
136 : 110.36572322036692
143 : 107.2685431556065
40 : 85.58886513039347
1619 : 79.46937972188049

1179 : 73.80954746465616

Fourth run :
18 : 217.01547759254927
737 : 143.20814950102775
1719 : 107.0248749367538
118 : 103.8714352824032
790 : 100.44854080160394
136 : 99.0541771243672
143 : 98.17351320732719
40 : 86.97856965456168
1619 : 75.61105294758272
128 : 69.72161866173403

References

[1] Tushar Sarde. Hadoop interview questions and answers - what is combiner in mapreduce framework ? *http://toodey.com/*, 2005.

[2] Jimmy Lin and Chris Dyer. *Data-Intensive Text Processing with MapReduce*. 2010.

[3] Bill Bejeck. Calculating a co-occurrence matrix with hadoop. *http://codingjunkie.net/cooccurrence/*, 2012.