CSE 4/587 Spring 2023

Name: Pavana Lakshmi Venugopal

UBIT: pavanala

Analysis

Answer the following questions based on your WordCount application. **For questions 1-4, include a screenshot of your application output that shows where your answer comes from in your report**.

Note: WordCount.java is a basic Hadoop application for word count.

WordCount_StopWords.java is to include stop words. The same code can be run for both Q1 and Q2

1. What are the 25 most common words and the number of occurrences of each when you do not remove stopwords?

Source Code Screenshots

Code 1 - Just basic word count app

Mapper

```
// Mapper
// takes the text, tokenize the words
public static class TokenizerMapper
    extends Mapper<Object, Text, Text, IntWritable>{

  private final static IntWritable one = new IntWritable(1);
  private Text word = new Text();

  public void map(Object key, Text value, Context context
                  ) throws IOException, InterruptedException {
    // tokenize the lowercased text string
    String line = value.toString().toLowerCase(); // convert line to lowercase and to string variable
    StringTokenizer itr = new StringTokenizer(line, "!\"#$%&()*+,-./:;<=>?@[]^_`{|}~ "); // remove possible punctuations
    while (itr.hasMoreTokens()) {
      word.set(itr.nextToken());
      context.write(word, one);
    }
  }
}
```

Reducer

```java
// Reducer
// counts the number of times the word is in data and writes the word and sum as key value pair
public static class IntSumReducer
    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);
  }
}
```

Main method

```java
  public static void main(String[] args) throws Exception {
    Configuration conf = new Configuration();
    Job job = Job.getInstance(conf, "word count");
    job.setJar("WordCount.jar");
    job.setJarByClass(WordCount.class);
    job.setMapperClass(TokenizerMapper.class);
    job.setCombinerClass(IntSumReducer.class);
    job.setReducerClass(IntSumReducer.class);
    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(IntWritable.class);

    FileInputFormat.addInputPath(job, new Path(args[0]));
    FileOutputFormat.setOutputPath(job, new Path(args[1]));
    job.waitForCompletion(true);
    System.exit(job.waitForCompletion(true) ? 0 : 1);
  }
```

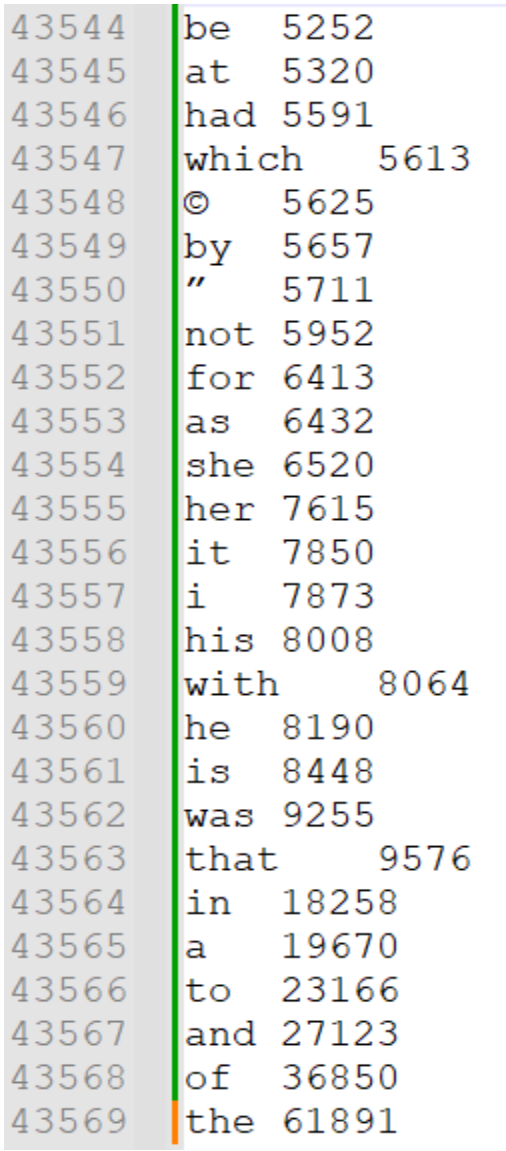
Input files

```
 705.4 K 2023-04-03 17:49 /input_dir/book1.txt
 730.8 K 2023-04-03 17:49 /input_dir/book10.txt
 970.9 K 2023-04-03 17:49 /input_dir/book2.txt
 464.0 K 2023-04-03 17:49 /input_dir/book3.txt
 405.8 K 2023-04-03 17:49 /input_dir/book4.txt
 953.9 K 2023-04-03 17:49 /input_dir/book5.txt
 438.1 K 2023-04-03 17:49 /input_dir/book6.txt
 754.1 K 2023-04-03 17:49 /input_dir/book7.txt
 165.3 K 2023-04-03 17:49 /input_dir/book8.txt
 297.9 K 2023-04-03 17:49 /input_dir/book9.txt
```

Output file for Q1 -

```
> hadoop fs -ls -h /output_init

rgroup          0 2023-04-03 21:24 /output_init/_SUCCESS
rgroup    469.2 K 2023-04-03 21:24 /output_init/part-r-00000
```



Screenshot of the 25 most common words (NOTE - The highest number of occurrences are at the bottom)

2. What are the 25 most common words and the number of occurrences of each when you do remove stopwords?

Q2 Code -
Code to parse stopWords file

```
private Configuration conf;
private BufferedReader fis;
private Set<String> patternsToSkip = new HashSet<String>();

static enum CountersEnum { INPUT_WORDS }

// setup method to check for stopwords file and making a call to parseStopWordFile method
public void setup(Context context) throws IOException, InterruptedException {

  conf = context.getConfiguration();
  if (conf.getBoolean("wordcount.skip.patterns", false)) {
    URI[] patternsURIs = Job.getInstance(conf).getCacheFiles();
    for (URI patternsURI : patternsURIs) {
      Path patternsPath = new Path(patternsURI.getPath());
      String patternsFileName = patternsPath.getName().toString();
      parseStopWordFile(patternsFileName);
    }
  }
}



// parseStopWordFile method to store all stopwords in file and storing in patternsToSkip variable
private void parseStopWordFile(String fileName) {
  try {
    fis = new BufferedReader(new FileReader(fileName));
    String pattern = null;
    while ((pattern = fis.readLine()) != null) {
      patternsToSkip.add(pattern.toLowerCase());
    }
  } catch (IOException ioe) {
    System.err.println("Caught exception while parsing the cached file '"
        + StringUtils.stringifyException(ioe));
  }
}
```

Updated map method to remove stopwords

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

  String line = value.toString().toLowerCase();       // converts the line to lowercase and saved as a string variable
  StringTokenizer itr = new StringTokenizer(line, "!\"#$%&()*+,-./:;<=>?@[]^_`{|}~ ");    // remove possible punctuations & tokenize the line into words
  while (itr.hasMoreTokens()) {                       // check if not empty
    String token = itr.nextToken();                   // store token in a variable
    if (!patternsToSkip.contains(token)) {            // check if token is not in patternsToSkip list
      word.set(token);
      context.write(word, one);
    }
  }
 }
}
```

No change in reducer logic

```
// Reducer
// counts the number of times the word is in data and writes the word and sum as key value pair
public static class IntSumReducer
    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);
  }
}
```

Main method -

```java
public static void main(String[] args) throws Exception {
  Configuration conf = new Configuration();

  // Check for optional parameter. The program has to work even if the stopwords file has not been passed
  GenericOptionsParser optionParser = new GenericOptionsParser(conf, args);
  String[] remainingArgs = optionParser.getRemainingArgs();
  if ((remainingArgs.length != 2) && (remainingArgs.length != 4)) {
    System.err.println("Usage: wordcount <in> <out> [-skip skipPatternFile]");
    System.exit(2);
  }

  Job job = Job.getInstance(conf, "word count");
   job.setJar("WordCount_StopWords.jar");
  job.setJarByClass(WordCount_StopWords.class);
  job.setMapperClass(TokenizerMapper.class);
  job.setCombinerClass(IntSumReducer.class);
  job.setReducerClass(IntSumReducer.class);
  job.setOutputKeyClass(Text.class);
  job.setOutputValueClass(IntWritable.class);

  // verify the right parameter '-skip' is passed. Based on the parameter
  // set config boolean paramter
  List<String> otherArgs = new ArrayList<String>();
  for (int i=0; i < remainingArgs.length; ++i) {
    if ("-skip".equals(remainingArgs[i])) {
      job.addCacheFile(new Path(remainingArgs[++i]).toUri());
      job.getConfiguration().setBoolean("wordcount.skip.patterns", true);
    } else {
      otherArgs.add(remainingArgs[i]);
    }
  }


  FileInputFormat.addInputPath(job, new Path(otherArgs.get(0)));
  FileOutputFormat.setOutputPath(job, new Path(otherArgs.get(1)));
  job.waitForCompletion(true);
  System.exit(job.waitForCompletion(true) ? 0 : 1);
}
```

```
43417   must    985
43418   man 1008
43419   great   1027
43420   first   1035
43421   like    1047
43422   much    1068
43423   water   1070
43424   time    1120
43425   may 1256
43426   upon    1269
43427   two 1271
43428   thomas  1370
43429   see 1378
43430   mrs 1459
43431   mr  1605
43432   mutoscope   1641
43433   could   1696
43434   said    1814
43435   edison  1828
43436   american    1927
43437   biograph    1949
43438   would   2354
43439   one 2745
43440   co  3347
43441   ©   5625
43442   ”   5711
```

Screenshot of the 25 most common words when stopwords are removed (NOTE - The highest number of occurrences are at the bottom)

Command for Q2 -

```
PS D:\Hadoop\hadoop-3.2.1> hadoop jar WordCount_StopWords.jar WordCount_StopWords /input_dir/ /output_sw -skip /stopwords/stopwords.txt
```

Output for Q2 -

```
> hadoop fs -ls -h /output_sw

rgroup          0 2023-04-03 21:32 /output_sw/_SUCCESS
rgroup    468.1 K 2023-04-03 21:32 /output_sw/part-r-00000
```

Command to sort the contents of file -

```
sort -k 2n q1Output.txt > q1Output-sorted.txt
sort -k 2n q2Output.txt > q2Output-sorted.txt
```

3. Based on the output of your application, how does removing stop words affect the total amount of bytes output by your mappers? Name one concrete way that this would affect the performance of your application.

Solution:

By getting rid of these words, we can make our text more focused on the essential information by eliminating the low-level information. The model we train for our job shows no adverse effects from the removal of such words. Stop phrases should be eliminated because they increase the dataset size, which increases training time due to the larger number of tokens required. By removing the stop words the length of the sentences are clearly reduced. Example 12 billion bytes, or about 11 GB, would be needed to store a stop word that appears in 1 billion papers. Using mappers if we remove stop words, we would be removing all these bytes in turn saving memory space or storage. Typically, removing stop words from a text lowers the overall number of bytes produced by the mappers in a MapReduce job. Stop words, which include common words like “the”, “a”, “an”, “so”, “what” etc., are typically eliminated from text data because they don't have a lot of significance on their own and aren't likely to add much to the text's analysis or processing. When these stop words get removed from the text, the resulting data contains fewer words, which causes the mappers to process and output fewer bytes. If there are more stop words then the resultant output by the mappers are reduced significantly. On the other hand, if there are very few stop words then the bytes might not be that less compared to the Input.

Without removing stop words.

```
> hadoop fs -ls -h /output_init

rgroup          0 2023-04-03 21:24 /output_init/_SUCCESS
rgroup    469.2 K 2023-04-03 21:24 /output_init/part-r-00000
```

With removing stop words.

```
> hadoop fs -ls -h /output_sw

rgroup          0 2023-04-03 21:32 /output_sw/_SUCCESS
rgroup    468.1 K 2023-04-03 21:32 /output_sw/part-r-00000
```

In my case with the screenshots above, the number of bytes is definitely reduced. From 469.2 Kbytes it is reduced to 468.1 Kbytes. Although it looks quite negligible, still it is a good result considering we have less Input data (less than 10 MB). When it comes to real world scenarios data would be huge and there would be a significant change in bytes after removing stop words.

Reducing the amount of data in a text processing application is one specific way that eliminating stop words might impact how well it performs. Performance is drastically increased when stop words have been removed, since processing time is also reduced. Even with the variety of optimizations that are available to speed up the process, say more than a billion entries takes a fair amount of time. Without a question, reducing the size of the data set can improve performance. It requires time to train models, and the training time should be reduced if there are fewer tokens to be trained. Therefore, it might improve classification accuracy. As the emphasis is moved to the more significant and informative words in the text, eliminating stop words can also result in more accurate results in some natural language processing tasks like sentiment analysis or text classification.

4. Based on the output of your application, what is the size of your keyspace with and without removing stopwords? How does this correspond to the number of stopwords you have chosen to remove?

Solution:

Depending on the type of data being processed and the keys generated by the mapper function, a key space is usually defined as the collection of distinct keys produced by a MapReduce job.

The key space can be reduced by eliminating stop words· Eliminating stop words would shrink the key space if the keys produced by the mapper function are single words because there will be fewer distinct keys.

The size of the key space can be impacted by the amount of stop words chosen for removal in a few different ways:

· The key space size could increase if not enough stop words are eliminated from the dataset. In this case, removing more stop words might result in a reduced key space.

· If too many stop words are eliminated, it's possible that some crucial information will be lost. In this case, fewer stop words being eliminated might result in a bigger key space.

With stopwords

```
43544 be  5252
43545 at  5320
43546 had 5591
43547 which   5613
43548 ©   5625
43549 by  5657
43550 ”   5711
43551 not 5952
43552 for 6413
43553 as  6432
43554 she 6520
43555 her 7615
43556 it  7850
43557 i   7873
43558 his 8008
43559 with    8064
43560 he  8190
43561 is  8448
43562 was 9255
43563 that    9576
43564 in  18258
43565 a   19670
43566 to  23166
43567 and 27123
43568 of  36850
43569 the 61891
```

Without stopwords(more than 100)

```
43417   must    985
43418   man 1008
43419   great   1027
43420   first   1035
43421   like    1047
43422   much    1068
43423   water   1070
43424   time    1120
43425   may 1256
43426   upon    1269
43427   two 1271
43428   thomas  1370
43429   see 1378
43430   mrs 1459
43431   mr  1605
43432   mutoscope   1641
43433   could   1696
43434   said    1814
43435   edison  1828
43436   american    1927
43437   biograph    1949
43438   would   2354
43439   one 2745
43440   co  3347
43441   ©   5625
43442   ”   5711
```

If we limit the stop words to 100 then the output changes as seen below.

```
43444  like    1047
43445  much    1068
43446  before  1070
43447  water   1070
43448  time    1120
43449  after   1149
43450  being   1184
43451  may 1256
43452  upon    1269
43453  two 1271
43454  thomas  1370
43455  see 1378
43456  mrs 1459
43457  into    1594
43458  mr  1605
43459  mutoscope   1641
43460  could   1696
43461  said    1814
43462  edison  1828
43463  american    1927
43464  biograph    1949
43465  would   2354
43466  one 2745
43467  co  3347
43468  ©   5625
43469  ”   5711
```

5. Let's now assume you were going to run your application on the entirety of Project Gutenberg. For this question, assume that there are **100TB** of input data, the data is spread over **10 sites**, and each site has **20 mappers**. Assume you ignore all but the **25 most common words** that you listed in question 2. Furthermore, assume that your combiners have been run optimally, so that each combiner will output at most 1 key-value pair per key.
    a. How much data will each mapper have to parse?
    b. What is the size of your keyspace?
    c. What is the maximum number of key-value pairs that could be communicated during the barrier between mapping and reducing?
    d. Assume you are running one reducer per site. On average, how many key-value pairs will each reducer have to handle?

Solution:

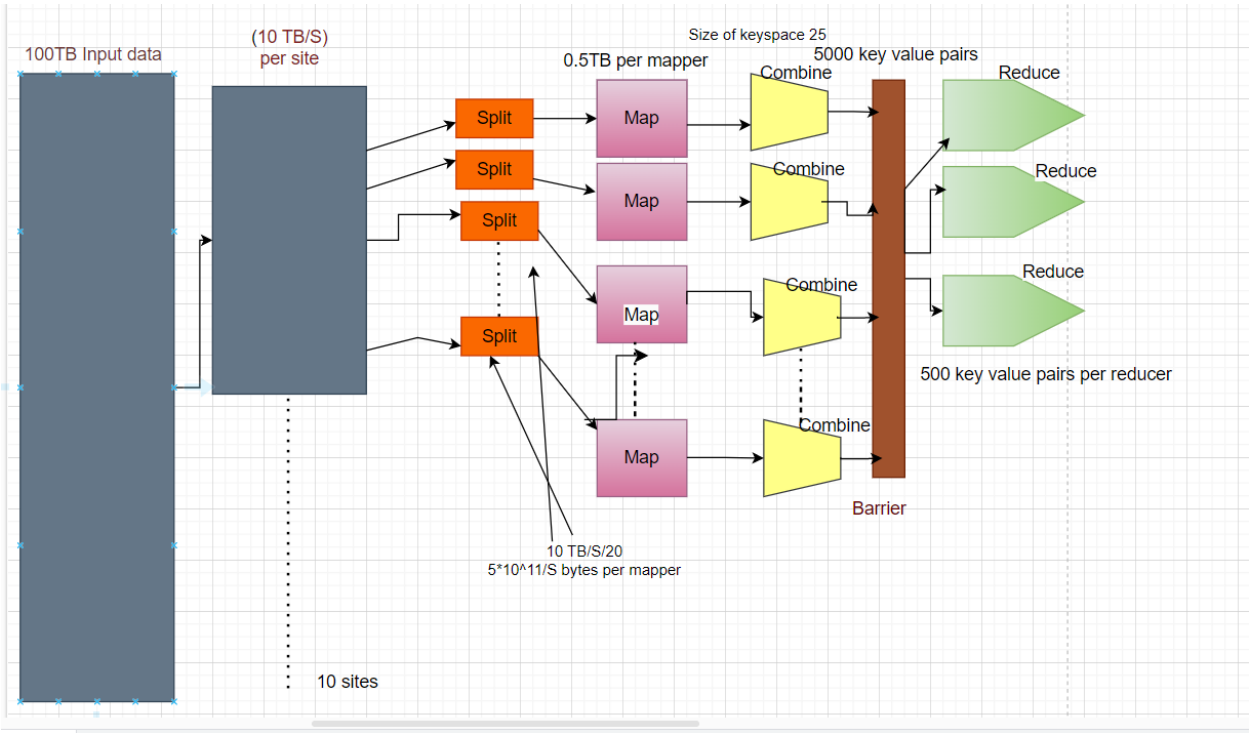
a)
Given that there are 10 sites and each site has 20 mappers, a total of 200 mappers will be working to handle the 100TB of Project Gutenberg input data. We can anticipate a substantial reduction in the amount of data that needs to be processed if we eliminate all words except the top 25.Each of the ten sites would have 10TB of input data if the size of the input data were spread equally among them.Accordingly, each mapper would have to handle roughly 500GB of input data (100TB / 200 mappers).The quantity of data that every mapper will process is contingent upon how the 25 most frequently occurring words are dispersed throughout the input data.

b)
The size of the keyspace will be 25 presuming that we are only taking into account the 25 most frequent words and that each combiner will yield no more than 1 key-value pair per key.

c)
Let's presume for the purposes of calculation that each mapper produces 25 distinct keys. (i.e., one for each of the 25 most common words).With a total of 200 mappers (20 mappers per site at 10 sites), the end result would be 5,000 distinct keys (200 mappers * 25 keys).The keyspace would also be 5,000 key-value pairs in size if each combiner only outputs one key-value combination for each key.It's essential to remember that this is only an approximate estimate based on some simplifying assumptions, and the actual size of the keyspace may vary depending on the precise distribution of the input data's 25 most prevalent words.

d)
Each reducer will have to deal with 1/10th of the total amount of key-value pairs if there are 10 reducers (one for each site) and the key-value pairs are distributed evenly among them.In a prior response, we predicted that, using only the 25 most frequent words, the mappers would produce about 5,000 key-value pairs. Each reducer would typically have to manage 500 key-value pairs (5,000 key-value pairs total / 10 reducers).

6. Draw the data flow diagram for question 5. The diagram should be similar to the diagram shown in lecture. On your diagram, label the specific quantities you got for 5a,b,c, and d.



References -

- For Stop Words - https://gist.github.com/sebleier/554280
- Word count basics - https://hadoop.apache.org/docs/stable/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html
- Hadoop Installation - https://www.joe0.com/2017/02/02/how-to-install-a-hadoop-single-node-cluster-on-windows-10/