



# DIC - HW2 - MAP REDUCE

## Contents

[About My Environment](#)

[Basic Word Count](#)

[Initial Examination](#)

[Basic Word Count Extended](#)

[Glimpse of stop words file](#)

[Initial Examination](#)

[Analysis \[Questions 1 - 6\]](#)

[References](#)

For this homework assignment, we will delve into Hadoop, giving you the opportunity to gain hands-on experience in writing and executing MapReduce code. Additionally, we will conduct subsequent analysis. In broad terms, the tasks involve the installation and configuration of Hadoop and MapReduce, the creation of a simple word-count program, and responding to inquiries related to data flow within a MapReduce application.

## About My Environment

- **Hadoop Version:** Hadoop 3.2.1
- **Source Code Repository:** <https://gitbox.apache.org/repos/asf/hadoop.git>
  - Commit Reference: b3cbbb467e22ea829b3808f4b7b01d07e0bf3842

- **Compilation Environment:**
  - Compiled with Protoc Version: 2.5.0
  - Compilation Command Location:  
`/usr/local/hadoop/share/hadoop/common/hadoop-common-3.2.1.jar`
  - Environment: Windows Subsystem for Linux (WSL) - Ubuntu 20.04
- **Java Runtime Environment:**
  - OpenJDK Version: 1.8.0\_382
  - OpenJDK Runtime Environment: build 1.8.0\_382-8u382-ga-1~22.04.1-b05
  - Java Virtual Machine: 64-Bit Server VM, build 25.382-b05, mixed mode

```
kiyo@MININT-7B4IC2U:/mnt/c/users/tarun/desktop/dic/hw2$ hadoop version
Hadoop 3.2.1
Source code repository https://gitbox.apache.org/repos/asf/hadoop.git -r b3cbbb467e22ea829b3808f4b7b01d07e0bf3842
Compiled by rohitsharmaks on 2019-09-10T15:56Z
Compiled with protoc 2.5.0
From source with checksum 776eaf9eee9c0ffc370bcbe1888737
This command was run using /usr/local/hadoop/share/hadoop/common/hadoop-common-3.2.1.jar
```

## Basic Word Count

Before we proceed to count the words, as I mentioned in above I've placed my Hadoop setup in the WSL, which means i need to transfer the files of the books I downloaded from the Project Gutenberg to Hadoop. The command goes like `hadoop fs -put <input-file-path> <destination-path>`

The below figure shows the five books in the Hadoop placed in the `mydata` directory in text format that was downloaded from the project Gutenberg each is more than 100kb which satisfies the requirements of the task.

## Browse Directory

/mydata									Go!			
Show	25	entries							Search:			
	Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name				
<input type="checkbox"/>	-rw-r--r--	kiyo	supergroup	1.15 MB	Nov 02 19:14	2	128 MB	pg71885.txt				
<input type="checkbox"/>	-rw-r--r--	kiyo	supergroup	5.04 MB	Nov 02 19:14	2	128 MB	pg71897.txt				
<input type="checkbox"/>	-rw-r--r--	kiyo	supergroup	528.51 KB	Nov 02 19:15	2	128 MB	pg71913.txt				
<input type="checkbox"/>	-rw-r--r--	kiyo	supergroup	256.19 KB	Nov 02 19:15	2	128 MB	pg71927.txt				
<input type="checkbox"/>	-rw-r--r--	kiyo	supergroup	839.09 KB	Nov 02 19:15	2	128 MB	pg71940.txt				

Showing 1 to 5 of 5 entries

Previous 1 Next

Now let's focus on counting the words with the most frequency from the whole book, though there's a demo program that's given MapReduce tutorial documentation that program fails to remove the punctuation so it's modified to satisfy the following cases, that list should not include any punctuation (other than punctuation that is actually part of the word), and should be case insensitive (Dog and dog would be counted as the same word). Let's have a look at the below code for better understanding.

Firstly we import the required libraries inorder to perform the task, these are important for the Hadoop and Map reduce as they give access to the Hadoop libraries and classes that are essential to implement the Map reduce tasks.

```
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 static class TokenizerMapper
    extends Mapper<Object, Text, Text, IntWritable>{
```

Next define the `TokenizerMapper` class, which extends the `Mapper` class provided by Hadoop.

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

In the `map` method within the `TokenizerMapper` class, which is executed for each input key-value pair.

```
StringTokenizer itr = new StringTokenizer(value.toString());
while (itr.hasMoreTokens()) {
    String token = itr.nextToken().replaceAll("[^a-zA-Z]", "").toLowerCase();
    if (!token.isEmpty()) {
        word.set(token);
        context.write(word, one);
    }
}
```

This is the crucial part of the code where the modification happened to filter out the duplicates and characters like ‘ / ‘ , “ , etc by utilizing regex `replaceAll("[^a-zA-Z]", "")` to remove any non-alphabetical characters (punctuation) from the word by doing this we’re only extracting the alphabetx from the texts, and now Convert the word to lowercase using `.toLowerCase()` . This will help to satisfy the requirements to not include duplicates and punctuation in the counts.

```
public static class IntSumReducer
    extends Reducer<Text, IntWritable, Text, IntWritable> {

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

Here the `reduce` method within the `IntSumReducer` class, which is executed for each unique word which contains the values of the counts for each words.

```
int sum = 0;
for (IntWritable val : values) {
    sum += val.get();
}
```

```
result.set(sum);
context.write(key, result);
```

Here in the reduce method the code iterates through the values, which are the counts of the words emitted by mappers, within the `reduce` method. It calculates the total word count by adding the separate counts. It returns the ' result' as a `IntWritable` with the overall count. Finally, it uses `context.write (key, result)` to print the word (key) with its overall count.

```
public static void main(String[] args) throws Exception {
    Configuration conf = new Configuration();
    Job job = Job.getInstance(conf, "word count");
    job.setJarByClass(WordCount.class);
    job.setMapperClass(TokenizerMapper.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]));
    System.exit(job.waitForCompletion(true) ? 0 : 1);
}
```

- The `main` method is the entry point of the program. This calls for the Map Reducer to start by setting up Hadoop `Configuration` and sets various properties.
- It then sets the various classes like mapper, reducer, output key and value classes.
- After that it specifies the input and output paths. then it waits for the job to complete and exits the program.

Now lets have a look at the whole 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 WordCount {

    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 {
            StringTokenizer itr = new StringTokenizer(value.toString());
            while (itr.hasMoreTokens()) {
                String token = itr.nextToken().replaceAll("[^a-zA-Z]", "").toLowerCase(); // Remove punctuation and make it lowercase
                if (!token.isEmpty()) {
                    word.set(token);
                    context.write(word, one);
                }
            }
        }
    }

    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);
        }
    }

    public static void main(String[] args) throws Exception {
        Configuration conf = new Configuration();
        Job job = Job.getInstance(conf, "word count");
        job.setJarByClass(WordCount.class);
        job.setMapperClass(TokenizerMapper.class);
        job.setReducerClass(IntSumReducer.class);
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(IntWritable.class);
        TextInputFormat.addInputPath(job, new Path(args[0]));
        FileOutputFormat.setOutputPath(job, new Path(args[1]));
        System.exit(job.waitForCompletion(true) ? 0 : 1);
    }
}

```

```
}
```

## Initial Examination

First compile it `javac -classpath hadoop classpath -d`

`/mnt/c/users/tarun/desktop/dic/hw2/basic_count/ WordCount.java` the create a .jar file `jar cf wc.jar WordCount*.class`, finally save the output to a folder `hadoop jar 'wc.jar' WordCount /mydata /basic_output`

```
kiyo@MININT-7B4IC2U:/mnt/c/users/tarun/desktop/dic/hw2$ javac -classpath `hadoop classpath` -d /mnt/c/users/tarun/desktop/dic/hw2/basic_count/WordCount.java
kiyo@MININT-7B4IC2U:/mnt/c/users/tarun/desktop/dic/hw2$ cd basic_count/
kiyo@MININT-7B4IC2U:/mnt/c/users/tarun/desktop/dic/hw2/basic_count$ jar cf wc.jar WordCount*.class
kiyo@MININT-7B4IC2U:/mnt/c/users/tarun/desktop/dic/hw2/basic_count$ hadoop jar 'uc.jar' WordCount /mydata /basic_output
JAR does not exist or is not a normal file: /mnt/c/users/tarun/desktop/dic/hw2/basic_count/uc.jar
kiyo@MININT-7B4IC2U:/mnt/c/users/tarun/desktop/dic/hw2/basic_count$ hadoop jar 'wc.jar' WordCount /mydata /basic_output
2023-11-04 20:16:14,637 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
2023-11-04 20:16:15,134 INFO client.RMProxy: Connecting to ResourceManager at localhost/127.0.0.1:8032
2023-11-04 20:16:15,519 WARN mapreduce.JobResourceUploader: Hadoop command-line option parsing not performed. Implement the Tool interface and execute your application with ToolRunner to remedy this.
2023-11-04 20:16:15,538 INFO mapreduce.JobResourceUploader: Disabling Erasure Coding for path: /tmp/hadoop-yarn/staging/kiyo/.staging/job_1698962573900_0018
2023-11-04 20:16:15,641 INFO sasl.SaslDataTransferClient: SASL encryption trust check: localHostTrusted = false, remoteHostTrusted = false
2023-11-04 20:16:15,759 INFO input.FileInputFormat: Total input files to process : 5
2023-11-04 20:16:15,790 INFO sasl.SaslDataTransferClient: SASL encryption trust check: localHostTrusted = false, remoteHostTrusted = false
2023-11-04 20:16:15,810 INFO sasl.SaslDataTransferClient: SASL encryption trust check: localHostTrusted = false, remoteHostTrusted = false
2023-11-04 20:16:15,824 INFO mapreduce.JobSubmitter: number of splits:5
2023-11-04 20:16:15,955 INFO sasl.SaslDataTransferClient: SASL encryption trust check: localHostTrusted = false, remoteHostTrusted = false
```

## Browse Directory

/output_basic_count										Go!			
Show 25 entries		Search:											
	Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name					
<input type="checkbox"/>	-rw-r--r--	kiyo	supergroup	0 B	Nov 03 16:14	2	128 MB	_SUCCESS					
<input type="checkbox"/>	-rw-r--r--	kiyo	supergroup	231 B	Nov 03 16:14	2	128 MB	part-r-00000					

Showing 1 to 2 of 2 entries

Previous 1 Next

Hadoop, 2022.

Now let's get the output files form the Hadoop to local `hadoop fs -copyToLocal`

`/basic_output/* /mnt/c/users/tarun/desktop/dic/hw2`

```
kiyo@MININT-7B4IC2U:/mnt/c/users/tarun/desktop/dic/hw2/count_w_stp$ hadoop fs -copyToLocal /basic_output/* /mnt/c/users/tarun/desktop/dic/hw2
2023-11-04 20:45:42,590 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
2023-11-04 20:45:43,296 INFO sasl.SaslDataTransferClient: SASL encryption trust check: localHostTrusted = false, remoteHostTrusted = false
```



Word	Count
a	24433
aah	2
aana	2
aaron	2
ab	5
abacists	2
abadiyah	2
abandon	32
abandoned	53
abandoning	5
abandonment	19
abomination	2

As you can see from the above picture these words are sorted based on the key values, so we need to sort them based on the count using a simple java program. as shown below.

```
File Edit Selection View Go Run ... ⏪ ⏩ ⏴ ⏵ HW2
CountWithoutStop.java 9+ SortWord.java 1 SortWord.java basic_c_out 1 WordCount.java 9+
basic_c_out > SortWord.java > SortWord > main(String[])
6 import java.util.List;
7 import java.util.ArrayList;
8 import java.util.Collections;
9 import java.util.Comparator;
10
11 public class SortWord {
12     Run|Debug
13     public static void main(String[] args) {
14         if (args.length != 1) {
15             System.exit(status:1);
16         }
17
18         String fileName = args[0];
19
20         try (BufferedReader br = new BufferedReader(new FileReader(fileName))) {
21             String line;
22             Map<String, Integer> wordCounts = new HashMap<>();
23
24             while ((line = br.readLine()) != null) {
25                 String[] wordCount = line.split(regex:"\\t");
26                 if (wordCount.length == 2) {
27                     String word = wordCount[0];
28                     int count = Integer.parseInt(wordCount[1].trim());
29                     wordCounts.put(word, count);
30                 }
31             }
32
33             List<Map.Entry<String, Integer>> sortedWordCounts = new ArrayList<>(wordCounts.entrySet());
34             sortedWordCounts.sort(Comparator.comparing(Map.Entry::getValue, Comparator.reverseOrder()));
35
36             List<Map.Entry<String, Integer>> top25 = sortedWordCounts.subList(fromIndex:0, Math.min(a:25, sortedWordCounts.size()));
37             System.out.println(x:"The top 25 words with highest counts");
38
39             for (Map.Entry<String, Integer> entry : top25) {
40                 String word = entry.getKey();
41                 int count = entry.getValue();
42
43                 System.out.println( word + " - " + count);
44             }
45         } catch (IOException e) {
46             e.printStackTrace();
47         }
48     }
49 }
```

This code takes the filename as input and print the top 25 words with most freequency as the output. Now let's have look into the output.

```
C:\Windows\System32\cmd.exe

C:\Users\Tarun\Desktop\DIC\HW2\basic_c_out>java SortWord part-r-00000
The top 25 words with highest counts
the - 105455
of - 65749
and - 35378
to - 33337
in - 28082
a - 24433
that - 13606
is - 12978
as - 10608
it - 10439
by - 10368
be - 10073
for - 9278
with - 8612
which - 8603
was - 8258
this - 7577
on - 7194
or - 6774
at - 6444
are - 5918
from - 5905
not - 5887
have - 5234
we - 5039

C:\Users\Tarun\Desktop\DIC\HW2\basic_c_out>
```

As you can see there are so many stop words, so in the next iteration we'll exclude them and see the difference.

## Basic Word Count Extended

Here we need to modify the previous code to ensure we don't count the stop words, for doing this first need to load the stop words, I have taken stop words text file from internet, and in the code I created a method to load the stop words from the file, this adds the stop words from a file by trimming the line spaces between the words.

```
private Set<String> stopWords = new HashSet<String>();

@Override
protected void setup(Context context) throws IOException, InterruptedException {
    // Load stop words from the stop words file
    Configuration conf = context.getConfiguration();
    String exclstoprds = conf.get("stopwords.file");
    if (exclstoprds != null && !exclstoprds.isEmpty()) {
        try (BufferedReader reader = new BufferedReader(new FileReader(exclstoprds))) {
            String line;
            while ((line = reader.readLine()) != null) {
                stopWords.add(line.trim());
            }
        }
    }
}
```

and in the main function define the path to the stop words file

```
Run | Debug
public static void main(String[] args) throws Exception {
    Configuration conf = new Configuration();
    // Pass the path to the stop words file as a configuration parameter
    conf.set("stopwords.file", "/mnt/c/users/tarun/desktop/dic/hw2/gist_stopwords.txt");
    Job job = Job.getInstance(conf, "word count");
```

Now let's look at the whole code

```

J CountWithoutStop.java > CountWithoutStop > main(String[])
220
221 public class CountWithoutStop {
222
223     public static class TokenizerMapper
224         extends Mapper<Object, Text, Text, IntWritable>{
225
226         private final static IntWritable one = new IntWritable(1);
227         private Text word = new Text();
228         private Set<String> stopWords = new HashSet<>();
229
230         @Override
231         protected void setup(Context context) throws IOException, InterruptedException {
232             // Load stop words from the stop words file
233             Configuration conf = context.getConfiguration();
234             String exclstopwrd = conf.get("stopwords.file");
235             if (exclstopwrd != null && !exclstopwrd.isEmpty()) {
236                 try (BufferedReader reader = new BufferedReader(new FileReader(exclstopwrd))) {
237                     String line;
238                     while ((line = reader.readLine()) != null) {
239                         stopWords.add(line.trim());
240                     }
241                 }
242             }
243         }
244
245         public void map(Object key, Text value, Context context
246                         ) throws IOException, InterruptedException {
247             StringTokenizer itr = new StringTokenizer(value.toString());
248             while (itr.hasMoreTokens()) {
249                 String token = itr.nextToken().replaceAll(regex:"[^a-zA-Z]",replacement:"").toLowerCase(); // Remove punctuation
250                 if (!token.isEmpty() && !stopWords.contains(token)) {
251                     word.set(token);
252                     context.write(word, one);
253                 }
254             }
255         }
256     }
257
258     public static class IntSumReducer
259         extends Reducer<Text, IntWritable, Text, IntWritable> {
260
261         public void reduce(Text key, Iterable<IntWritable> values,
262                           Context context) throws IOException, InterruptedException {
263             int sum = 0;
264             for (IntWritable val : values) {
265                 sum += val.get();
266             }
267             context.write(key, new IntWritable(sum));
268         }
269     }
270
271     Run | Debug
272     public static void main(String[] args) throws Exception {
273         Configuration conf = new Configuration();
274         // Pass the path to the stop words file as a configuration parameter
275         conf.set("stopwords.file", "/mnt/c/users/tarun/desktop/dic/hw2/gist_stopwords.txt");
276         Job job = Job.getInstance(conf, "word count");
277         job.setJarByClass(CountWithoutStop.class);
278         job.setMapperClass(TokenizerMapper.class);
279         job.setReducerClass(IntSumReducer.class);
280         job.setOutputKeyClass(Text.class);
281         job.setOutputValueClass(IntWritable.class);
282         FileInputFormat.addInputPath(job, new Path(args[0]));
283         FileOutputFormat.setOutputPath(job, new Path(args[1]));
284         System.exit(job.waitForCompletion(true) ? 0 : 1);
285     }
286 }

```

## Glimpse of stop words file



A screenshot of a Windows Notepad window titled "gist\_stopwords - Notepad". The window contains a list of stop words, each on a new line:

```
a's
aside
ask
asking
associated
at
available
away
awfully
back
backward
backwards
he
```

## Initial Examination

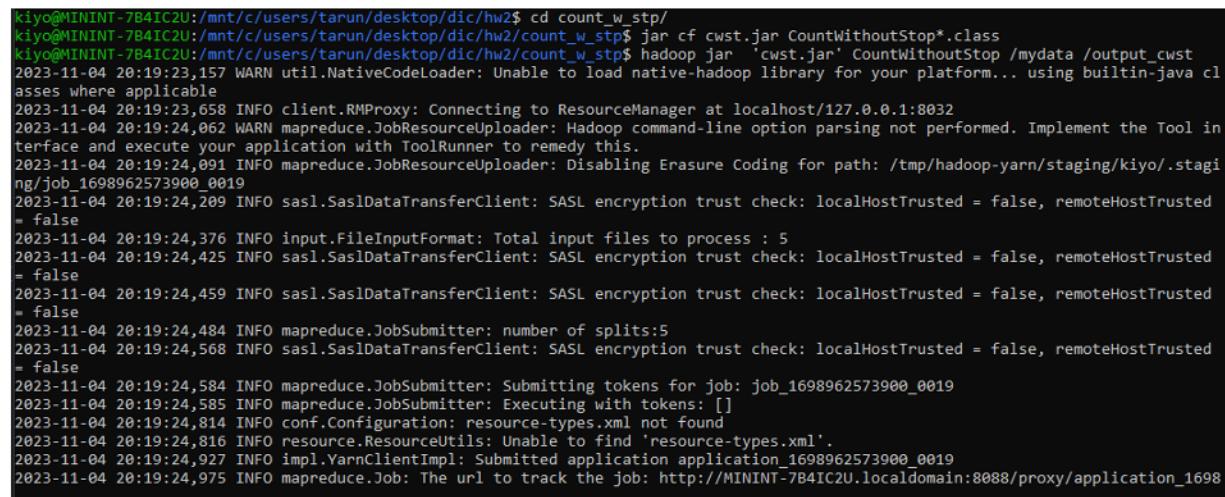
Now let's run the program. First Compile the Java code for MapReduce: `javac -classpath`

```
'hadoop classpath' -d /mnt/c/users/tarun/desktop/dic/hw2/count_w_stp/ CountWithoutStop.java
```

Then create a JAR File so .class files will be compiled files into a JAR `jar cf`

```
cwst.jar CountWithoutStop*.class Finally execute the jar file by running hadoop jar
```

```
'cwst.jar' CountWithoutStop /mydata /output_cwst
```

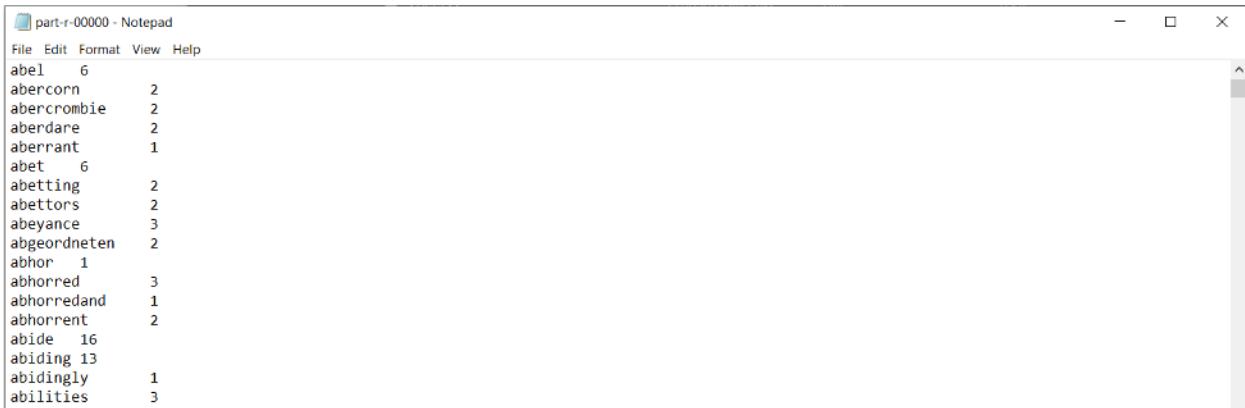


A terminal window showing the execution of a Hadoop job. The command run is:

```
kiyo@MININT-7B4IC2U:/mnt/c/users/tarun/desktop/dic/hw2$ cd count_w_stp/
kiyo@MININT-7B4IC2U:/mnt/c/users/tarun/desktop/dic/hw2/count_w_stp$ jar cf cwst.jar CountWithoutStop*.class
kiyo@MININT-7B4IC2U:/mnt/c/users/tarun/desktop/dic/hw2/count_w_stp$ hadoop jar 'cwst.jar' CountWithoutStop /mydata /output_cwst
2023-11-04 20:19:23,157 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
2023-11-04 20:19:23,658 INFO client.RMProxy: Connecting to ResourceManager at localhost/127.0.0.1:8032
2023-11-04 20:19:24,062 WARN mapreduce.JobResourceUploader: Hadoop command-line option parsing not performed. Implement the Tool interface and execute your application with ToolRunner to remedy this.
2023-11-04 20:19:24,091 INFO mapreduce.JobResourceUploader: Disabling Erasure Coding for path: /tmp/hadoop-yarn/staging/kiyo/.staging/job_1698962573900_0019
2023-11-04 20:19:24,289 INFO sasl.SaslDataTransferClient: SASL encryption trust check: localHostTrusted = false, remoteHostTrusted = false
2023-11-04 20:19:24,376 INFO input.FileInputFormat: Total input files to process : 5
2023-11-04 20:19:24,425 INFO sasl.SaslDataTransferClient: SASL encryption trust check: localHostTrusted = false, remoteHostTrusted = false
2023-11-04 20:19:24,459 INFO sasl.SaslDataTransferClient: SASL encryption trust check: localHostTrusted = false, remoteHostTrusted = false
2023-11-04 20:19:24,484 INFO mapreduce.JobSubmitter: number of splits:5
2023-11-04 20:19:24,568 INFO sasl.SaslDataTransferClient: SASL encryption trust check: localHostTrusted = false, remoteHostTrusted = false
2023-11-04 20:19:24,584 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1698962573900_0019
2023-11-04 20:19:24,585 INFO mapreduce.JobSubmitter: Executing with tokens: []
2023-11-04 20:19:24,814 INFO conf.Configuration: resource-types.xml not found
2023-11-04 20:19:24,816 INFO resource.ResourceUtils: Unable to find 'resource-types.xml'.
2023-11-04 20:19:24,927 INFO impl.YarnClientImpl: Submitted application application_1698962573900_0019
2023-11-04 20:19:24,975 INFO mapreduce.Job: The url to track the job: http://MININT-7B4IC2U.localdomain:8088/proxy/application_1698962573900_0019
```

Now let's get the output files from the Hadoop to local `hadoop fs -copyToLocal`

```
/output_cwst/*
```

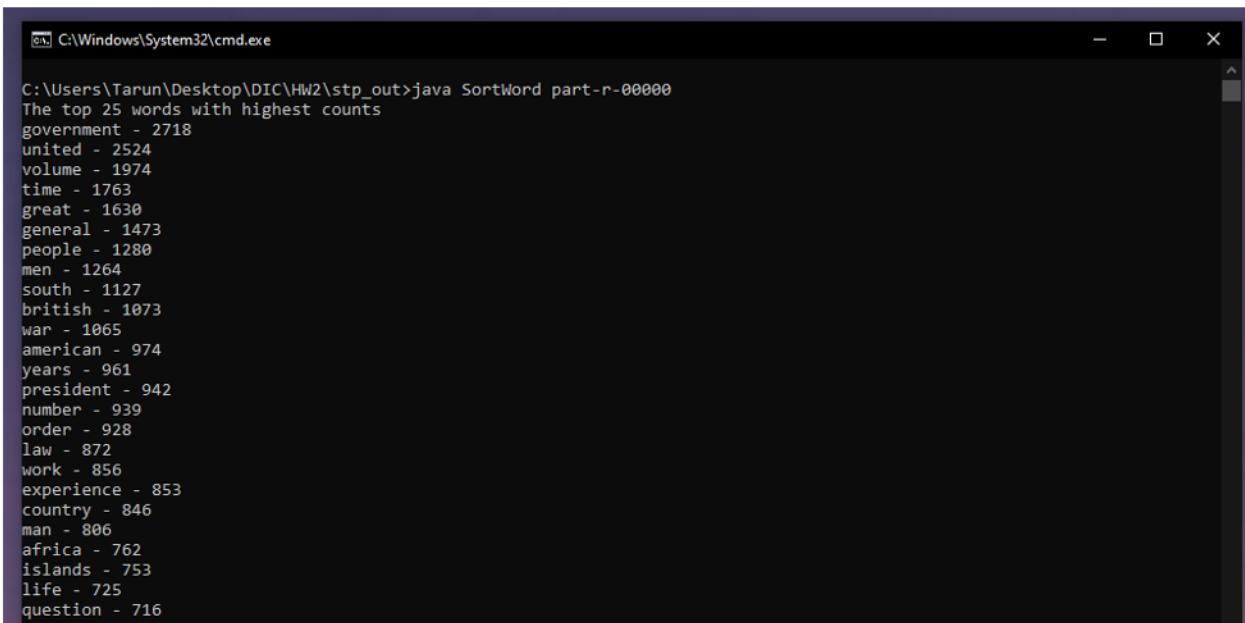


part-r-00000 - Notepad

File Edit Format View Help

Word	Count
abel	6
abercorn	2
abercrombie	2
aberdare	2
aberrant	1
abet	6
abetting	2
abettors	2
abeyance	3
abgeordneten	2
abhor	1
abhorred	3
abhorredand	1
abhorrent	2
abide	16
abiding	13
abidingly	1
abilities	3

As these are sorted by the key, let's sort this based on their count by using simple sort program in java, shown in previous section.



C:\Windows\System32\cmd.exe

```
C:\Users\Tarun\Desktop\DIGI\HW2\stp_out>java SortWord part-r-00000
The top 25 words with highest counts
government - 2718
united - 2524
volume - 1974
time - 1763
great - 1630
general - 1473
people - 1280
men - 1264
south - 1127
british - 1073
war - 1065
american - 974
years - 961
president - 942
number - 939
order - 928
law - 872
work - 856
experience - 853
country - 846
man - 806
africa - 762
islands - 753
life - 725
question - 716
```

## Analysis [Questions 1 - 6]

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

Answer:

Here are 25 most common words and the number of occurrences of each when you do not remove stopwords:

```
C:\Windows\System32\cmd.exe
C:\Users\Tarun\Desktop\DIC\HW2\basic_c_out>java SortWord part-r-00000
The top 25 words with highest counts
the - 105455
of - 65749
and - 35378
to - 33337
in - 28082
a - 24433
that - 13606
is - 12978
as - 10608
it - 10439
by - 10368
be - 10073
for - 9278
with - 8612
which - 8603
was - 8258
this - 7577
on - 7194
or - 6774
at - 6444
are - 5918
from - 5905
not - 5887
have - 5234
we - 5039
C:\Users\Tarun\Desktop\DIC\HW2\basic_c_out>
```

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

**Answer:**

Here are 25 most common words and the number of occurrences of each when we remove stopwords:

```
C:\Windows\System32\cmd.exe
C:\Users\Tarun\Desktop\Dic\HW2\stp_out>java SortWord part-r-00000
The top 25 words with highest counts
government - 2718
united - 2524
volume - 1974
time - 1763
great - 1630
general - 1473
people - 1280
men - 1264
south - 1127
british - 1073
war - 1065
american - 974
years - 961
president - 942
number - 939
order - 928
law - 872
work - 856
experience - 853
country - 846
man - 806
africa - 762
islands - 753
life - 725
question - 716
C:\Users\Tarun\Desktop\Dic\HW2\stp_out>
```

**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.**

- Without stop words:
  - Total map output bytes: 6,209,159 bytes
  - File Output Bytes Written=433133 bytes
- Data with stop words:
  - Total map output bytes: 12,194,134 bytes
  - File Output Bytes Written=425910

```

Select kyo@MININT-7B4IC2U:/mnt/c/users/tarun/desktop/dic/hw2/count_w_stp
Data-local map tasks=5
Total time spent by all maps in occupied slots (ms)=25279
Total time spent by all reduces in occupied slots (ms)=2682
Total time spent by all map tasks (ms)=25279
Total time spent by all reduce tasks (ms)=2682
Total vcore-milliseconds taken by all map tasks=25279
Total vcore-milliseconds taken by all reduce tasks=2682
Total megabyte-milliseconds taken by all map tasks=25885696
Total megabyte-milliseconds taken by all reduce tasks=2746368
Map-Reduce Framework
  Map input records=174482
  Map output records=507357
  Map output bytes=6209159
  Map output materialized bytes=7223903
  Input split bytes=525
  Combine input records=0
  Combine output records=0
  Reduce input groups=36981
  Reduce shuffle bytes=7223903
  Reduce input records=507357
  Reduce output records=36981
  Spilled Records=1014714
  Shuffled Maps =5
  Failed Shuffles=0
  Merged Map outputs=5
  GC time elapsed (ms)=762
  CPU time spent (ms)=12110
  Physical memory (bytes) snapshot=2097188864
  Virtual memory (bytes) snapshot=5335248728064
  Total committed heap usage (bytes)=1871183872
  Peak Map Physical memory (bytes)=435761152
  Peak Map Virtual memory (bytes)=1370805088064
  Peak Reduce Physical memory (bytes)=243027968
  Peak Reduce Virtual memory (bytes)=551231320064
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=8154751
File Output Format Counters
  Bytes Written=425910

Select kyo@MININT-7B4IC2U:/mnt/c/users/tarun/desktop/dic/hw2/count_w_no_stp
Killed map tasks=1
Launched map tasks=5
Launched reduce tasks=1
Data-local map tasks=5
Total time spent by all maps in occupied slots (ms)=43392
Total time spent by all reduces in occupied slots (ms)=3724
Total time spent by all map tasks (ms)=43392
Total time spent by all reduce tasks (ms)=3724
Total vcore-milliseconds taken by all map tasks=43392
Total vcore-milliseconds taken by all reduce tasks=3724
Total megabyte-milliseconds taken by all map tasks=44433408
Total megabyte-milliseconds taken by all reduce tasks=3813376
Map-Reduce Framework
  Map input records=174482
  Map output records=12194134
  Map output bytes=12194134
  Map output materialized bytes=14667614
  Input split bytes=525
  Combine input records=0
  Combine output records=0
  Reduce input groups=37698
  Reduce shuffle bytes=14667614
  Reduce input records=1236725
  Reduce output records=37698
  Spilled Records=2473450
  Shuffled Maps =5
  Failed Shuffles=0
  Merged Map outputs=5
  GC time elapsed (ms)=1073
  CPU time spent (ms)=14598
  Physical memory (bytes) snapshot=2067402752
  Virtual memory (bytes) snapshot=5915289108488
  Total committed heap usage (bytes)=1956642816
  Peak Map Physical memory (bytes)=472326144
  Peak Map Virtual memory (bytes)=1418684137472
  Peak Reduce Physical memory (bytes)=192131072
  Peak Reduce Virtual memory (bytes)=4624405668992
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=8154751
File Output Format Counters

```

Without Stop Words

With Stop Words

The removal of stopwords can bring significant performance increase it reduces the data that mappers need to transmit to the reducers, as we can see the mappers have 6,209,159 bytes to send, compared to 12,194,134 bytes with stopwords. This can lead to faster data processing and increased scalability as the size of the data decreases without stopwords as you can see the File Output Bytes Written=425910 with stopwords is higher than the File Output Bytes Written=433133 bytes without stopwords. Finally, as there's less data to process there's less usage of the resources like memory. Hence removing the stop words helps in overall increase increase in the performance of the system.

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?

- Number of stopwords used in gist\_stopwords.txt: 851

- Data without stop words:
  - Total reduce input groups (keyspace size): 36,981
- Data with stop words:
  - Total reduce input groups (keyspace size): 37,698

```

Select kyo@MININT-7B4IC2U:/mnt/c/users/tarun/desktop/dic/hw2/count_w_stp
Killed map tasks=1
Launched map tasks=5
Launched reduce tasks=1
Data-local map tasks=5
Total time spent by all maps in occupied slots (ms)=25279
Total time spent by all reduces in occupied slots (ms)=2682
Total time spent by all map tasks (ms)=25279
Total time spent by all reduce tasks (ms)=2682
Total vcore-milliseconds taken by all map tasks=25279
Total vcore-milliseconds taken by all reduce tasks=2682
Total megabyte-milliseconds taken by all map tasks=25885696
Total megabyte-milliseconds taken by all reduce tasks=2746368
Map-Reduce Framework
  Map input records=174482
  Map output records=507357
  Map output bytes=6209159
  Map output materialized bytes=7223903
  Input split bytes=525
  Combine input records=0
  Combine output records=0
  Reduce input groups=36981
  Reduce shuffle bytes=7223903
  Reduce input records=507357
  Reduce output records=36981
  Spilled Records=1014714
  Shuffled Maps =5
  Failed Shuffles=0
  Merged Map outputs=5
  GC time elapsed (ms)=762
  CPU time spent (ms)=12110
  Physical memory (bytes) snapshot=2097188864
  Virtual memory (bytes) snapshot=53352487728864
  Total committed heap usage (bytes)=1871183872
  Peak Map Physical memory (bytes)=435761152
  Peak Map Virtual memory (bytes)=1370885080064
  Peak Reduce Physical memory (bytes)=243027968
  Peak Reduce Virtual memory (bytes)=551231320064
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=8154751
File Output Format Counters

Select kyo@MININT-7B4IC2U:/mnt/c/users/tarun/desktop/dic/hw2/count_w_stp
HDFS: Number of bytes written=433133
HDFS: Number of read operations=20
HDFS: Number of large read operations=0
HDFS: Number of write operations=2
HDFS: Number of bytes read erasure-coded=0
Job Counters
  Killed map tasks=1
  Launched map tasks=5
  Launched reduce tasks=1
  Data-local map tasks=5
  Total time spent by all maps in occupied slots (ms)=43392
  Total time spent by all reduces in occupied slots (ms)=3724
  Total time spent by all map tasks (ms)=43392
  Total time spent by all reduce tasks (ms)=3724
  Total vcore-milliseconds taken by all map tasks=43392
  Total vcore-milliseconds taken by all reduce tasks=3724
  Total megabyte-milliseconds taken by all map tasks=44433408
  Total megabyte-milliseconds taken by all reduce tasks=3813376
Map-Reduce Framework
  Map input records=174482
  Map output records=1236725
  Map output bytes=12194134
  Map output materialized bytes=14667614
  Input split bytes=525
  Combine input records=0
  Combine output records=0
  Reduce input groups=37698
  Reduce shuffle bytes=14667614
  Reduce input records=1236725
  Reduce output records=37698
  Spilled Records=2473458
  Shuffled Maps =5
  Failed Shuffles=0
  Merged Map outputs=5
  GC time elapsed (ms)=1073
  CPU time spent (ms)=14590
  Physical memory (bytes) snapshot=2067482752
  Virtual memory (bytes) snapshot=5915289108480
  Total committed heap usage (bytes)=1956642816
  Peak Map Physical memory (bytes)=472326144
  Peak Map Virtual memory (bytes)=1418684137472
  Peak Reduce Physical memory (bytes)=192131072
  Peak Reduce Virtual memory (bytes)=462440660992
Shuffle Errors
  BAD_ID=0
  CONNECTION=0
  IO_ERROR=0

```

Without STOP                          With STOP

Without stop words, the keyspace is 36,981, while with stop words, it is 37,698. The keyspace size is proportional to the number of unique words in the dataset.

**So total number of stop words reduced:  $37,698 - 36,981 = 717$ .**

## 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 keyvalue pair per key.**

- **How much data will each mapper have to parse?**
- **What is the size of your keyspace?**
- **What is the maximum number of key-value pairs that could be communicated during the barrier between the mapping and reducing?**
- **Assume you are running one reducer per site. On average, how many key-value pairs will each reducer have to handle?**

**Answer:**

- **How much data will each mapper have to parse?**

Given,

- 100 TB of input data
- Number of sites the input data is being spread = 10
- Number of mappers = 20

So, Let's calculate the data per site = Total input data / sites

- $100\text{TB} / 10 = 10 \text{ TB of data per site}$

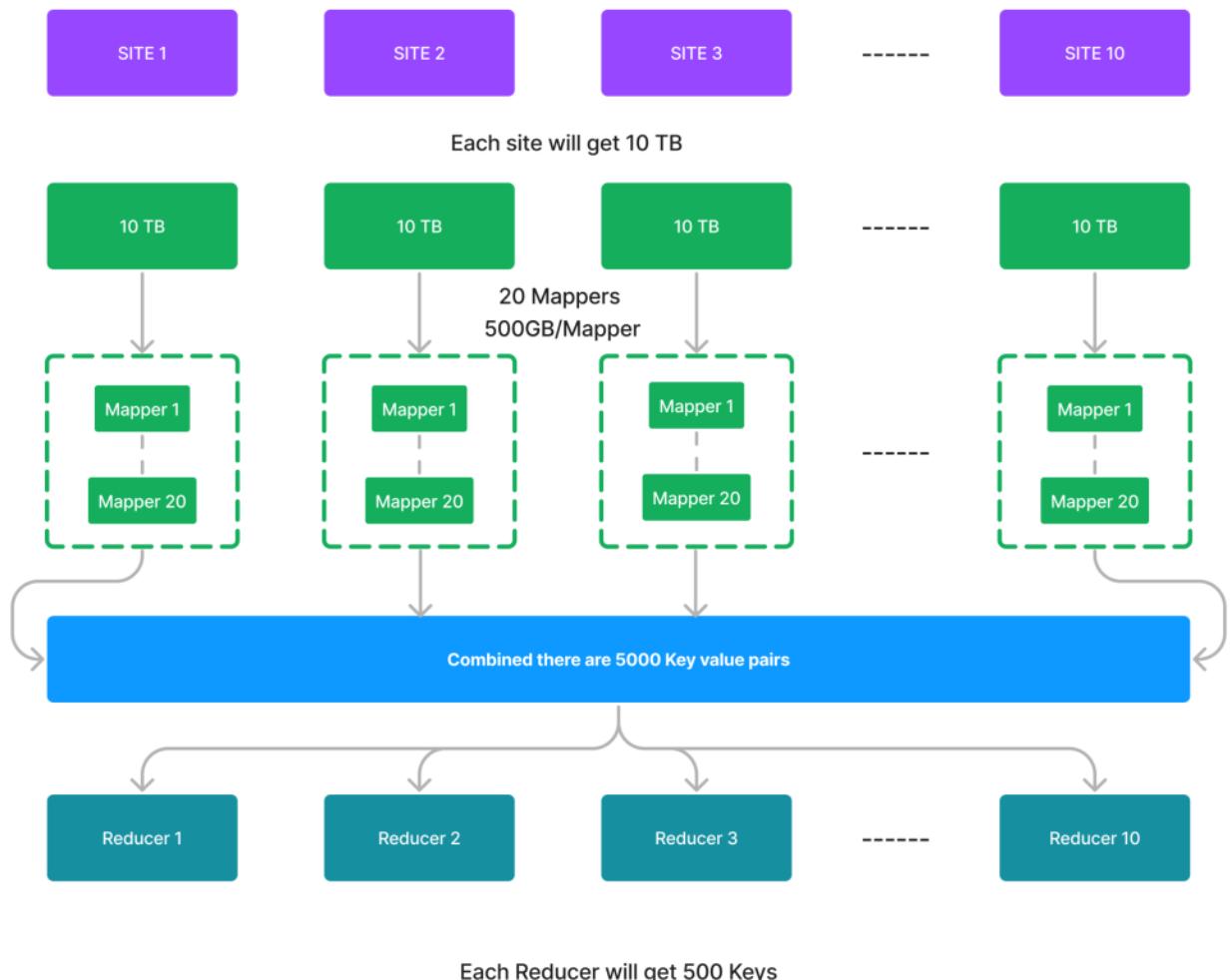
Data that each mapper needs to parse = Data given per site / the number of mappers

- $10\text{TB} / 20 = 500\text{GB}$

SO each mapper will have 500GB or 500000MB of data to parse.

- **What is the size of your keyspace?**
  - 25 as all the words are being excluded.
- **What is the maximum number of key-value pairs that could be communicated during the barrier between the mapping and reducing?**

- There are 25 keys per mapper
  - There are 20 Mappers
  - There are 10 sites
  - So As each mapper may output at most 1 key pair
  - total key value pairs = 25 keys per mapper x 20 mappers per site x 10 sites = 5000 key value pairs.
- **Assume you are running one reducer per site. On average, how many key-value pairs will each reducer have to handle?**
    - Since there are 5000 key value pairs for and 10 sites, so each reducer will handle 500 key value pairs worth of data.
6. **Draw the data flow diagram for question 5. The diagram should be similar to the diagram shown in the lecture. On your diagram, label the specific quantities you got for 5a,b,c, and d.**



@bhanucha  
On Figma

## References

- Easiest way to sort a list of words by occurrence. (n.d.). Stack Overflow. Retrieved November 6, 2023, from <https://stackoverflow.com/questions/2366443/easiest-way-to-sort-a-list-of-words-by-occurrence>

- *Enhancing the word count sample a little bit.* (2012, October 28). Altocumulus. <https://yossidahan.wordpress.com/2012/10/28/enhancing-the-word-count-sample-a-little-bit/>
- *How to copy file from HDFS to the local file system.* (n.d.). Stack Overflow. Retrieved November 6, 2023, from <https://stackoverflow.com/questions/17837871/how-to-copy-file-from-hdfs-to-the-local-file-system>
- *MapReduce Tutorial.* (n.d.). Apache.org. Retrieved November 6, 2023, from <https://hadoop.apache.org/docs/stable/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html>
- *[Solved]-Remove punctuation and HTML entity with hadoop Wordcount in java-Java.* (n.d.). Hire Developers, Free Coding Resources for the Developer. Retrieved November 6, 2023, from <https://www.appsloveworld.com/java/100/2159/remove-punctuation-and-html-entity-with-hadoop-wordcount-in-java>
- *Stop words list.* (n.d.). CountWordsFree. Retrieved November 6, 2023, from <https://countwordsfree.com/stopwords>