Hadoop

Thejas Raju Ravi Govind Raju

1 Introduction

Big data refers to a very large collection of data that is characterised by the velocity at which data is being generated, volume of the data being generated, variety of data, and variability or flow of the data across systems.

The programming model utilised for processing Big Data sets is called MapReduce. As the name suggests, it consists of two main procedures Mapping and Reducing. The sorting and filtering of data is performed by the Mapper. The Mapper reads the input files and outputs key-value pairs which is passed onto the reducer. The summarising of data is performed by the Reducer. Here, the reducer performs some aggregation procedure over a list of values that share a common key. The result of this is passed to the output file as key-value pairs.

Hadoop is a software that allows using multiple nodes across networks to handle and perform computations across a large volume of data. Big data tasks can be performed using Hadoop framework as it implements the MapReduce programming model. Hadoop uses a Distributed File System which facilitates the access storage and access of data across networks in a convenient format. Hadoop Commons module provides the tools and utilities required to process Big Data. The system resources are managed by the YARN module.

2 Requirements

The system is required to perform Big Data jobs using the Hadoop framework which implements the MapReduce programming model. The data to be analysed is an archive of debates from the European Parliament. The data set is ideal and qualifies as a BigData set as it possess a huge volume of data which is of a wide variety. The dataset being used is a subset of a bigger data corpus which consists of 2,218,201 sentences and 53,974,751 words. The variety is introduced due to the wide range of topics that were debated in the European Parliament which provides a stage for different views and opinions for expression. This introduces various words and their semantics.

The system is expected to be capable of processing the dataset to analyse and provide insights into the text present in them. The system should be able to find all the different words that are present in the dataset and provide the total count of these unique words. This count can be used to understand volume of the data being analysed. Based on the result, user can decide whether there is sufficient data present for analysis or if more data should be included.

Dataset is an archive of debates which consist of various separators which are used to implement grammatical syntax of the language. These separators include punctuations (such as periods, question marks, semicolons etc.), whitespaces, tab spaces, and new-line characters. These do not convey any special meaning in the analysis and introduce overhead. For example, words ‘new.’ and ‘new,’ are counted as two different words initially which is incorrect. Hence, the system is expected to omit separators to increase accuracy and precision of the count of words present.

The data consists of words which occur multiple times across multiple documents. However, some terms tend to occur less frequently and often provide a better comprehension of the dataset. The system should be able to extract the count of only words which occur less than four times across the entire dataset being analysed.

Words which occur more frequently can also provide an insight into the data. For example two set of words me/my/mine/I and us/our/ours/we can help study whether the debates consist of more singular possessive pronouns or plural possessive pronouns. System should be capable of providing the total individual count of different set of words.

As the dataset contains debates, some topics may consist of niche terms whose appearances are restricted to a single document. These unique terms are effective to communicate the subjects or themes being discussed. The application is expected to be able to express terms which appear one or more times only in a single document.

As mentioned earlier, multiple words may occur across multiple documents. Some words may occur more frequently than others. On close examination, it can be seen that often the words which occur with a high frequency are words which are generally considered as stop-words. Stop-words include words such as a, an, the, or, is, these etc. These words occur very frequently but often convey no special meaning. Hence, the system should remove these stop-words from the dataset before analysis.

Considering a specific term and iterating the words which appear after it provides a better understanding of the context or semantics of the term chosen. In order to perform this analysis, the system must be capable of extracting five words that appear most after a term related to a political concept.

3 Design

The application is setup on a Virtual Machine running Ubuntu operating system with one gigabyte of memory and two processors allocated as system resources. The Hadoop setup files are installed and configured along with JDK tools which are essential to run BigData jobs. The dataset to be analysed is downloaded to the local system and uploaded to a directory in the Hadoop File System (HDFS) for analysis.

The count of total number of unique words in a file is obtained by modifying the WordCount program. A static variable ‘wordCount’ is initialised in the Reducer and set to zero. The variable ‘wordCount’ is incremented each time the reduce method is invoked. The Mapper returns a key-value pair and the reduce method increases the value of ‘wordCount’ by one for each key encountered. The final value of ‘wordCount’ is printed to the output file using a clean-up method defined in the Reducer. The clean-up function is used as this is the final step in execution and hence the total value unique words can be printed in the output file. The variable is defined as ‘static’ which ensure there is only one copy of the variable and prevents it from being initialised at

each invocation of the Reducer. In the Driver class, the job.setCombinerClass() method is omitted as count of all unique words is to be found.
The number of unique words found are 342,207.

The number of unique words found increases when non-alphanumeric characters such as punctuations, tab spaces, line breaks, and other symbols are removed. The map method is modified such that each string token is checked and replaceAll() function is implemented to remove all occurrence of punctuations, tabs, line breaks and other symbols. The replaceAll() function takes two parameters, where the first argument specifies the characters to be replaced. They are passed as a regular expression and the second parameter consists of empty quotes which specifies that the characters are only removed and not replaced with anything else. In the Reducer, a static variable keeps count of each word that is encountered. At the end of execution, a cleanup() method is defined to pass the total wordCount calculated to the output file.
The number of unique words found after removing non-alphanumeric characters are 635,535 which is more than count in Task-1.

To extract only words occurring less than four times, modification to the reduce function is made. Here, the variable 'sum' keeps track of total number of occurrences of each key. The variable is initialised to zero and this initialisation is done for every invocation of the reduce function. The 'sum' value is incremented by one for each value present with the same key. The 'sum' is then subjected to a conditional statement. If the value of 'sum' is less than four, the value of a static variable 'wordCount' is increased each time. If the value of 'sum' is greater than four, the 'wordCount' value is not increased. At the end of execution of the Reducer, a cleanup() method is defined to pass the total wordCount calculated to the output file.
The number of words that appear less than four times are 840,736.

The comparison of the count of two sets of words me/my/mine/I and us/our/ours/we is done by making modifications to the map function. Here, each token is analysed to check if they belong to one of the set of words given. If the token belongs to any one of them, the respective key-value pair is generated where the key is the set of words the token belongs to, and the value one for each occurrence. The reducer is invoked only twice as there are only two keys present. The reducer uses a counter to increment the value at each occurrence of the key (set of words). This pair is then passed as output to the file.
The number of occurrences of me/my/mine/I and us/our/ours/we are 718,883 and 704,456 respectively. The terms me/my/mine/I occur more frequently in the dataset.

To count the number of terms occurring in a single document, modifications are made to both the map and reduce methods. In the map method, variable files is used to store the names of the files being analysed. The mapper outputs a key-value pair which consists of the word as key and the document where it occurs as the value. In the reduce method, a HashMap is used to store each occurrence of a file, the HashMap key consists of the file names and the value includes the count of the word returned as key from the map method. A conditional statement checks the number of entries in the HashMap for each word. If the word has a single entry, a static variable is

incremented each time for each word. This variable value is passed as output to the file.
The number of terms that appear only in a single document are 10,259.

To study the impact of removing stop-words on the word-count, a dictionary of stop-words was used. This was saved in the local directory containing the Java program. A setup method is declared in the Mapper where the file is read using the Scanner function, and the stop-words present in the file are saved to a string HashSet variable 'stopWord'. In the map function, each token being analysed is matched to check if it is present in the 'stopWord' HashSet. If present, it indicated that the token is a stop-word and no action is to be taken. If the token is not present in the HashSet, the static variable containing wordCount value is incremented and the total count of occurrences is passed as output to the file.
The total number of words are 1,093,904. The number of words found increases when compared to results of Task-1.

The political concept chosen is "justice". To extract five words that are present after the occurrence of "justice", changes to the map function are made. The first token being analysed at each instance is stored in a string and checked to see if the token matches the political concept chosen. If a match is found and there are more tokens are present following this word, a key-value pair is generated such that the key is the token occurring after " justice" and value is the count of the word. Another counter is declared to keep in check that no more than five tokens are returned for each match found.

4 Challenges Faced

The dataset provided could not be uploaded as individual files to the HDFS as attempt to do so resulted in system crashes or system hangs which lasted for excess periods. To work around this, all the text files present in the dataset were combined to a single file using the 'cat' command. This single file was used in the analysis of all tasks except Task-5. For Task-5 a dataset containing a total of five text files chosen from the dataset were uploaded to the HDFS and used for analysis.

4.1 Task-1 Unique words in the corpus

The general WordCount program outputs all unique words with their respective counts. However, to output only the total number of unique words found, a counter had to be used. This counter had to be declared as static so that only one copy of the variable existed and did not lead to being initialised to zero for each new word encountered. A context.write() in the reduce method would output the count at every instance. To avoid this and obtaining a single total count at the end of the execution, a cleanup() method is declared which holds the context.write() method. In the job driver, the setCombinerClass() was omitted.

4.2 Task-2 Removing separators to reduce number of unique terms

Removing separators required matching and as tokens can pose challenges, the token being analysed was stored in a string variable. The string replaceAll() method was used to find matching characters and replace them. Here, the matching punctuations, white space characters and other symbols were only removed and not replaced with anything else.

4.3 Task-3 Finding words which appear less than 4 times

This task could be completed with using a counter to keep track of how many times a word has occurred in the dataset. The reduce function was modified by adding a conditional statement to check the value of 'sum' variable which wrote to the output file if value is lesser than four, otherwise, the word was skipped.

4.4 Task-4 Finding the frequency of set of words

In order to find the frequency of given set of words, the tokens posed a challenge to perform matching. Hence, each token was converted to a string and equals() function returns true if a match was found. Two conditional statements whose expression is a combination of equals() function and logical operator OR was used. One for each set of words. The wordCount was incremented if a word belonging to either of the sets was encountered.

4.5 Task-5 Listing terms which appear in only one single document

This task required using additional java libraries such as FileSplit to read the names of individual files being analysed and HashMap library to store the file names and its occurrence. The names of files were read in the map method where the key-value returned consists of word as key and the file name as value. The reduce method was modified to create a HashMap such that each file name was entered as a key in the HashMap and the count of the word was entered as value of the HashMap. To obtain words which are unique to a single document, a conditional statement was defined which checks the size of the HashMap. A HashMap of size one implies that word is unique to a single file and wordCount value was incremented.

4.6 Task-6 Removing stop-words from the dataset

To read stop-words from a file and store them, additional java libraries were imported. A function set-up is defined in the Mapper which is instantiated once. Here, the file containing stop-words is scanned and the words are stored in a HashSet. In the map method, each token is subjected to a check to confirm whether the word is a stop-word by looking up the HashSet. If a match is not found, the word and the count were passed as key-value pair to the reducer. Otherwise, the word was considered as a stop-word and was not passed to the output.

4.7 Task-7 Computing five words appearing the most after a political concept term

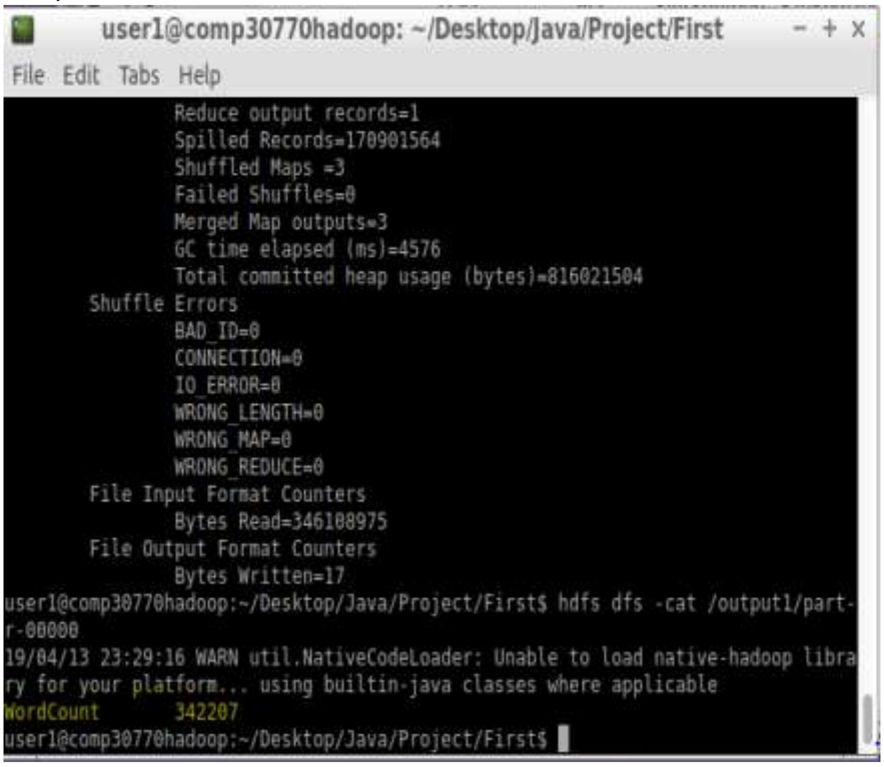
While searching for matching tokens to the political concept chosen, it was noticed that the tokens may be present in different cases in the dataset and hence all the characters present in the token being analysed was converted into lower case before matching. The token may also be followed by punctuations and other characters which may result in a mismatch. In order to avoid missing the match, startsWith() method is used which returns true for every token which begins with the political term being analysed.

5 Conclusion

The BigData set provided was analysed using the Hadoop framework by implementing the MapReduce programming model. It can be seen that Hadoop simplifies the process of analysing such huge data. The dataset used was unstructured but Hadoop could manage it efficiently which shows its flexibility. Hadoop’s robustness was evident as it showed that by tweaking a raw program such as WordCount by removing stop-words and other separators could provide better analysis of the data. It provides a good platform for drawing statistics such as limiting the counting to words appearing a specific number of times. Observations can also be made by limiting counting to only a certain pre-determined words. Lastly, the accelerated speed of execution for such large datasets can be observed due to the distributed framework.

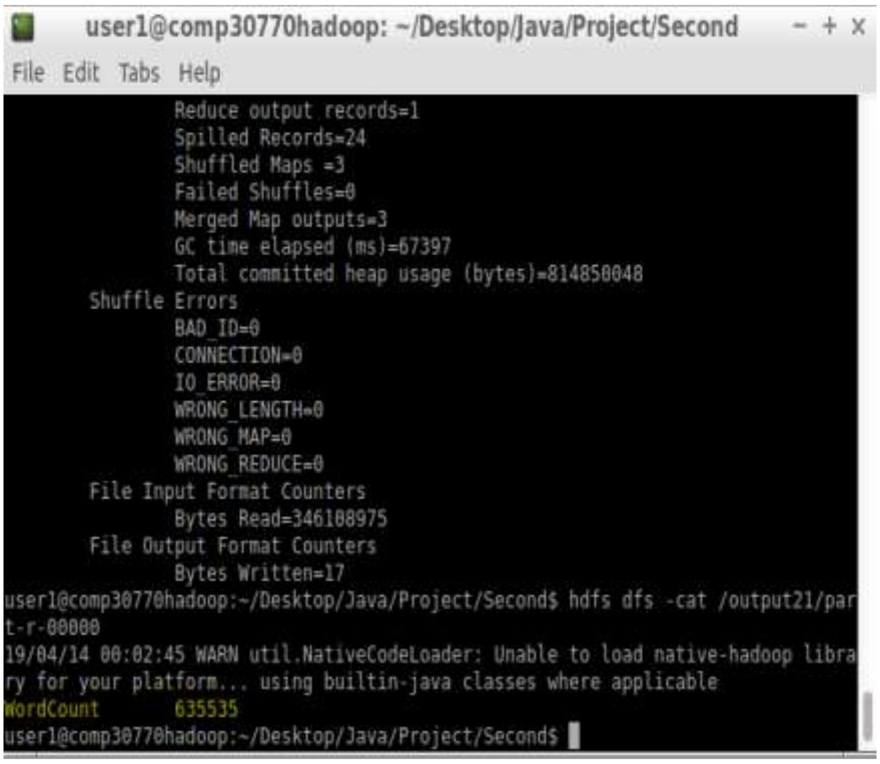
Appendix

- TASK-1 O/P:



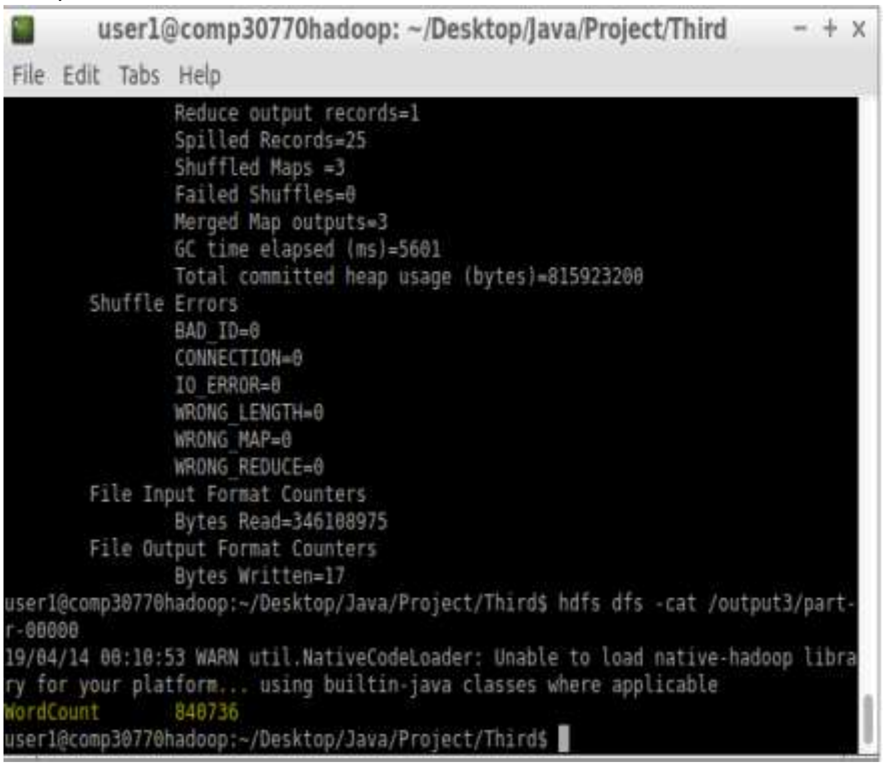
```
                Reduce output records=1
                Spilled Records=170901564
                Shuffled Maps =3
                Failed Shuffles=0
                Merged Map outputs=3
                GC time elapsed (ms)=4576
                Total committed heap usage (bytes)=816021504
        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=346108975
        File Output Format Counters
                Bytes Written=17
user1@comp30770hadoop:~/Desktop/Java/Project/First$ hdfs dfs -cat /output1/part-
r-00000
19/04/13 23:29:16 WARN util.NativeCodeLoader: Unable to load native-hadoop libra
ry for your platform... using builtin-java classes where applicable
WordCount       342207
user1@comp30770hadoop:~/Desktop/Java/Project/First$
```

- TASK-2 O/P:



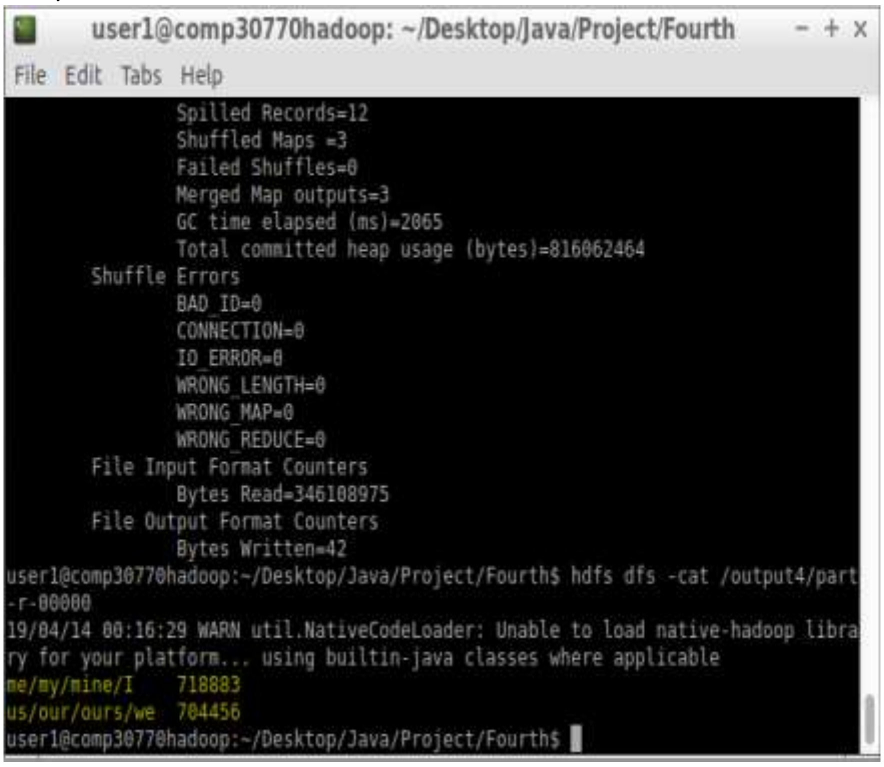
```
                Reduce output records=1
                Spilled Records=24
                Shuffled Maps =3
                Failed Shuffles=0
                Merged Map outputs=3
                GC time elapsed (ms)=67397
                Total committed heap usage (bytes)=814850048
        Shuffle Errors
                BAD_ID=0
                CONNECTION=0
                IO_ERROR=0
                WRONG_LENGTH=0
                WRONG_MAP=0
                WRONG_REDUCE=0
        File Input Format Counters
                Bytes Read=346108975
        File Output Format Counters
                Bytes Written=17
user1@comp30770hadoop:~/Desktop/Java/Project/Second$ hdfs dfs -cat /output21/par
t-r-00000
19/04/14 00:02:45 WARN util.NativeCodeLoader: Unable to load native-hadoop libra
ry for your platform... using builtin-java classes where applicable
WordCount       635535
user1@comp30770hadoop:~/Desktop/Java/Project/Second$
```

- TASK-3 O/P:



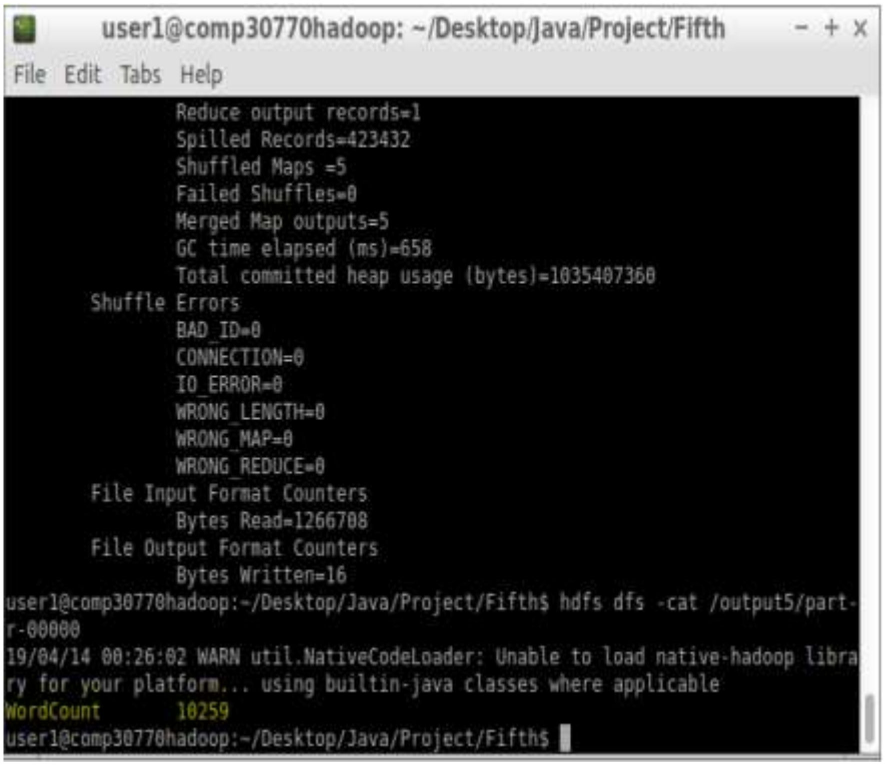
```
                Reduce output records=1
                Spilled Records=25
                Shuffled Maps =3
                Failed Shuffles=0
                Merged Map outputs=3
                GC time elapsed (ms)=5601
                Total committed heap usage (bytes)=815923200
        Shuffle Errors
                BAD_ID=0
                CONNECTION=0
                IO_ERROR=0
                WRONG_LENGTH=0
                WRONG_MAP=0
                WRONG_REDUCE=0
        File Input Format Counters
                Bytes Read=346108975
        File Output Format Counters
                Bytes Written=17
user1@comp30770hadoop:~/Desktop/Java/Project/Third$ hdfs dfs -cat /output3/part-
r-00000
19/04/14 00:10:53 WARN util.NativeCodeLoader: Unable to load native-hadoop libra
ry for your platform... using builtin-java classes where applicable
WordCount       840736
user1@comp30770hadoop:~/Desktop/Java/Project/Third$
```

- TASK-4 O/P:



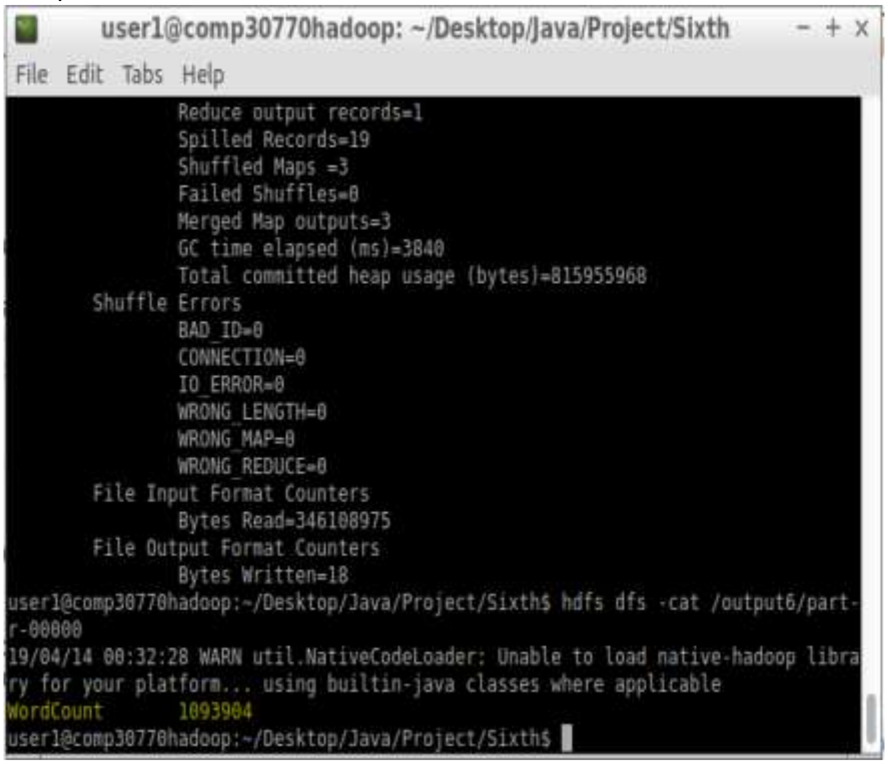
```
                Spilled Records=12
                Shuffled Maps =3
                Failed Shuffles=0
                Merged Map outputs=3
                GC time elapsed (ms)=2065
                Total committed heap usage (bytes)=816062464
        Shuffle Errors
                BAD_ID=0
                CONNECTION=0
                IO_ERROR=0
                WRONG_LENGTH=0
                WRONG_MAP=0
                WRONG_REDUCE=0
        File Input Format Counters
                Bytes Read=346108975
        File Output Format Counters
                Bytes Written=42
user1@comp30770hadoop:~/Desktop/Java/Project/Fourth$ hdfs dfs -cat /output4/part
-r-00000
19/04/14 00:16:29 WARN util.NativeCodeLoader: Unable to load native-hadoop libra
ry for your platform... using builtin-java classes where applicable
me/my/mine/I    718883
us/our/ours/we  704456
user1@comp30770hadoop:~/Desktop/Java/Project/Fourth$
```

- TASK-5 O/P:



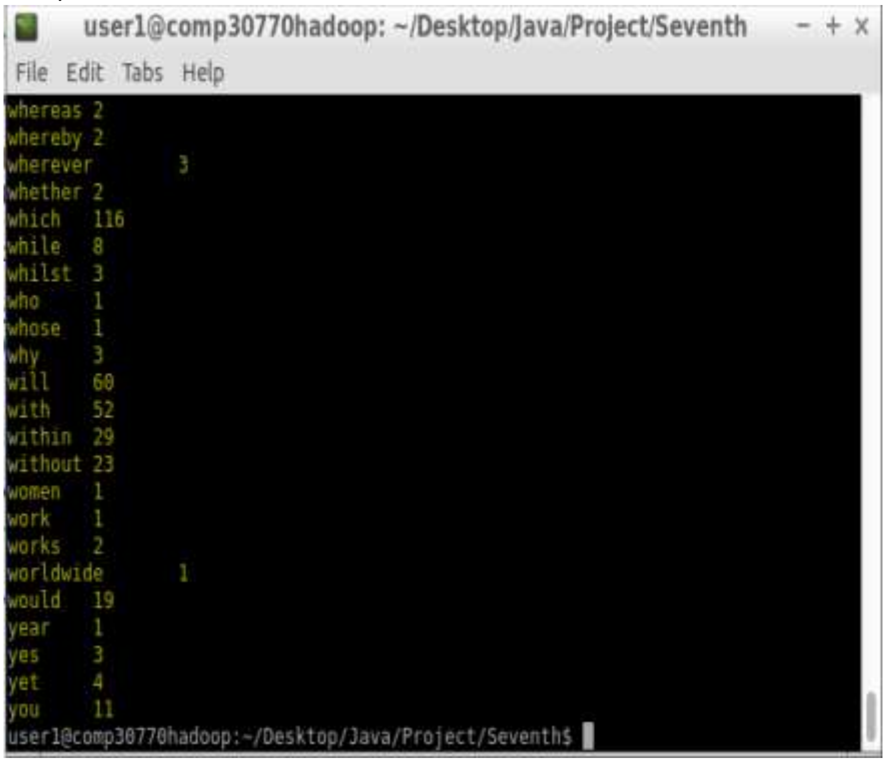
```
                Reduce output records=1
                Spilled Records=423432
                Shuffled Maps =5
                Failed Shuffles=0
                Merged Map outputs=5
                GC time elapsed (ms)=658
                Total committed heap usage (bytes)=1035407360
        Shuffle Errors
                BAD_ID=0
                CONNECTION=0
                IO_ERROR=0
                WRONG_LENGTH=0
                WRONG_MAP=0
                WRONG_REDUCE=0
        File Input Format Counters
                Bytes Read=1266708
        File Output Format Counters
                Bytes Written=16
user1@comp30770hadoop:~/Desktop/Java/Project/Fifth$ hdfs dfs -cat /output5/part-
r-00000
19/04/14 00:26:02 WARN util.NativeCodeLoader: Unable to load native-hadoop libra
ry for your platform... using builtin-java classes where applicable
WordCount       10259
user1@comp30770hadoop:~/Desktop/Java/Project/Fifth$
```

- TASK-6 O/P:



```
                Reduce output records=1
                Spilled Records=19
                Shuffled Maps =3
                Failed Shuffles=0
                Merged Map outputs=3
                GC time elapsed (ms)=3840
                Total committed heap usage (bytes)=815955968
        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=346108975
        File Output Format Counters
                Bytes Written=18
user1@comp30770hadoop:~/Desktop/Java/Project/Sixth$ hdfs dfs -cat /output6/part-
r-00000
19/04/14 00:32:28 WARN util.NativeCodeLoader: Unable to load native-hadoop libra
ry for your platform... using builtin-java classes where applicable
WordCount       1093904
user1@comp30770hadoop:~/Desktop/Java/Project/Sixth$
```

➢ TASK-7 O/P:



```
whereas 2
whereby 2
wherever        3
whether 2
which   116
while   8
whilst  3
who     1
whose   1
why     3
will    60
with    52
within  29
without 23
women   1
work    1
works   2
worldwide       1
would   19
year    1
yes     3
yet     4
you     11
user1@comp30770hadoop:~/Desktop/Java/Project/Seventh$
```

References

http://hadooptutorial.info/mapreduce-programming-model/

http://www.statmt.org/europarl/

http://bigdataprogrammers.com/get-distinct-words-file-using-map-reduce/

http://b03hadoop.blogspot.com/2015/11/executing-word-count-program-in-hadoop.html

https://stackoverflow.com/questions/49140121/hadoop-mapreduce-context-write-changes-values

https://acadgild.com/blog/building-inverted-index-mapreduce

https://brightspace.ucd.ie/d2l/le/content/35457/viewContent/329388/View

https://stackoverflow.com/questions/29178258/using-hashset-to-store-a-text-file-and-read-from-it

https://www.geeksforgeeks.org/hashset-in-java/