COEN242 PA2 - Top K words in a big dataset

Hui-yu Liu, Ching-Yueh Huang

I. Introduction

In the previous work, we used python to find top K words in a big dataset, including 300MB, 2.5GB and 16GB. This report is another way using Hadoop MapReduce to implement it. The goal is to find the Top 100 frequent words and the words having more than 6 characters using MapReduce. In addition, we will be utilizing the nlargest function from Python's heapq module to efficiently analyze large datasets and gain insights into the most commonly used words in a given text corpus.

This report aims to introduce the method we utilized in our function and setting files. We will first provide a detailed explanation of the method and how it was implemented. In addition, we will discuss some observations and optimization techniques that we adopted during the testing phases to improve the performance of our method. Furthermore, we will summarize the results obtained from our experiments and provide insights into the effectiveness of our approach. Finally, we will conclude our work by discussing the implications of our findings and potential future improvements.

II. Methods

To perform word count using Hadoop MapReduce, we began by creating a mapper to split each line of text into individual words and convert them to lowercase. This was done because all the stopwords provided were in lowercase. By converting all the words to lowercase, we made it easier to identify the stopwords. The output of the mapper was a set of key-value pairs where the word was the key and the value was set to 1. To optimize performance, we also implemented a combiner to aggregate the intermediate results for each key before they were sent to the reducer.

To meet the requirement of retrieving the top 100 words with each word having more than 6 characters, we wrote another mapper program. Here, we also converted each word to lowercase and counted the number of occurrences of each word using counters. After aggregating the results, we passed them to the reducer for further processing of the data. In the reducer program, we created a dictionary to store the key-value pairs. Moreover, we implemented ‘heapq’ module with ‘nlargest’ function, which is used to return the n largest elements from the key-value dataset. The algorithm of heap queue works by maintaining the the heap size, and iterating over the collection, adding each element to the heap and then popping the smallest element off if the heap size exceeds the given number. This algorithm has the time complexity of O(log n), proving that it is an efficient way to maintain a heap in memory. In terms of space efficiency, since only a small portion fo the input data needs to be stored in memory at any given time, the heap queue is space efficient. To get the top k element from the input data, we customized our heapq algorithm with ‘nlargest’ function, which allowed us to sort the data based on a specified key and handle any ties between elements. This approach proved to be the effective solution for sorting and filtering data in Python.

Next, we used Hadoop Streaming to run the MapReduce job on a Hadoop cluster. We specified the number of reducers and partitioned the data based on the first two characters of each word. And end up writing a reducer function to sum up the value of each key to get the total count for each word. In addition, we changed some variables in mapred-site.xml to optimize the whole process. For example, we increase the ‘mapreduce.task.io.sort.mb’ parameter which results in improved performance with more memory available for the map output buffer and reduced amount of data that needs to be spilled to disk during the sort and shuffle phase. It also reduced disk I/O. With more memory available for the map output buffer, the amount of data that needs to be spilled to disk during the sort and shuffle phase is reduced. This resulted in faster job completion times. Plus, the setting parameter increased parallelism with more memory available for sorting. This improved overall job throughput by allowing more tasks executing in parallel. However, we were aware of the tradeoffs involved in increasing this parameter, such as requiring more memory, which could be problematic if the cluster had limited resources., which can be a problem if the cluster has limited resources. So we carefully monitored the test result and adjusted the the parameter by calculating the most suitable requirements based on our resources and the efficiency of MapReduce tasks accordingly.

III. Analysis

A. MapReduce Optimization

When Map starts generating output, it does not simply write the data to disk, as frequent disk operations can cause severe performance degradation. It is more complex in that the data is first written to a buffer in memory and some pre-sorting is done to improve efficiency.

Each Map task has a circular memory buffer used to write the output data. The default size of this buffer is 100MB, and the exact size can be set via the io.sort.mb property. When the amount of data in the buffer reaches a specific threshold (io.sort.mb * io.sort.spill.percentage, where io.sort.spill.percentage is 0.80 by default), a background thread is started to spill the contents of the buffer to disk. During the spill process, the output of Map will continue to be written to the buffer, but if the buffer is full, Map will be blocked until the spill is complete. spill thread will do a second quick sort on the buffer before writing it to disk, first sorting by the partition the data belongs to, and then sorting by Key in each partition. in each partition. The output consists of an index file and a data file.

If a Combiner is set up, it will run on top of the sorted output, which is a Mini Reducer that runs on the node itself performing the Map task, first doing a simple Reduce on the Map output, making the Map output more compact, with less data being written to disk and transferred to the Reducer.

The spill file is stored in the directory specified by mapred.local.dir and deleted after the Map task is completed.

A new spill file is created every time the data in memory reaches the spill threshold, so there may be multiple spill files by the time the Map task finishes writing its last output record. Before the Map task finishes, all spill files will be merged and sorted into one index file and one data file. This is a multi-way merge process, and the maximum number of merges is controlled by io.sort.factor (default is 10). If Combiner is set and the number of spill files is at least 3 (controlled by the min.num.spills.for.combine property), then Combiner will be run to compress the data before the output files are written to disk.

Compressing the data written to disk is usually a good way to do this, as it makes the data write to disk faster, saves disk space, and reduces the amount of data that needs to be transferred to the Reducer. The default output is not compressed, but it can be enabled by simply setting mapred.compress.map.output to true. The library used for compression is set by mapred.map.output.compression.codec.

When the spill files are consolidated, Map will delete all temporary spill files and tell TaskTracker that the task is complete. The number of threads used to transfer the partitions data is controlled by tasktracker.http.threads, which is set for each TaskTracker, not a single Map, and defaults to 40.

For Intermediate data, this is always a good idea to use LZO compression. Every Hadoop job that generates a non-negligible amount of map output will benefit from intermediate data compression with LZO. Although LZO adds a little bit of overhead to the CPU, it saves time by reducing the amount of disk IO during the shuffle.

Name	Default	Changed
mapreduce.task.io.sort.mb	100	220
mapreduce.task.io.sort.factor	10	100
mapreduce.map.sort.spill.percent	0.8	0.9
mapred.map.child.java.opts	-Xmx200m	-Xmx1024m
mapred.reduce.child.java.opts	-Xmx1024m	-Xmx1024m
mapreduce.reduce.input.buffer.percent	0.0	0.2
mapred.job.shuffle.merge.percent	0.66	0.8
min.num.spills.for.combine	3	5

Data changed in mapred-site.xml

B. Cases of experiment

In Hadoop, the input data is divided into blocks, typically 64MB or 128MB in size. For a 16GB input, assuming a block size of 128MB, you would have approximately 110 blocks. Each block will be processed by a mapper. We will give some cases below to check how different cases work.

1) Case 1

When setting only one reducer and allowing the number of mappers to be defined by the data size, the system will automatically determine the optimal number of mappers based on the input data.

```
hadoop jar
$HADOOP_HOME/share/hadoop/tools/lib/hadoop-streaming-3.3.5.ja
r \
-file <path_to_mapper.py> \
-mapper <path_to_mapper.py> \
-file <path_to_reducer.py> \
-reducer <path_to_reducer.py> \
-file <path_to_stopword.txt> \
-input <hadoop_path_to_data_16GB.txt> \
-output /output/16gb/case1
```

	Job Overview
Job Name:	streamjob5080891716127857877.jar
User Name:	sandy
Queue:	default
State:	SUCCEEDED
Uberized:	false
Submitted:	Fri May 12 21:02:59 PDT 2023
Started:	Fri May 12 21:03:05 PDT 2023
Finished:	Fri May 12 21:50:25 PDT 2023
Elapsed:	47mins, 19sec
Diagnostics:	
Average Map Time	1mins, 19sec
Average Shuffle Time	24mins, 4sec
Average Merge Time	2mins, 12sec
Average Reduce Time	15mins, 8sec

ApplicationMaster

Attempt Number	Start Time	Node	Logs
1	Fri May 12 21:03:01 PDT 2023	172.31.94.170:8042	logs

Task Type	Total	Complete
Map	115	115
Reduce	1	1

Attempt Type	Failed	Killed	Successful
Maps	0	1	115
Reduces	0	0	1

As you can see, for Case 1, there are 115 mappers and one reducer, we spend most of the time in shuffle and reduce. The total time of it is about 47min. We can also check the MapReduce FrameWork in Counter, the spilled records of mapper is more than 2,900,000,000. Each map spill for about 28MB, so that's why we tried to change mapreduce.task.io.sort.mb from default 100 to 200.

2) Case 2

To enhance the efficiency of the data processing workflow, we conducted an analysis to identify areas for improvement. In Case1, where no combiner was employed, the execution time for the Reduce phase was significant. In an effort to mitigate this, we introduced a combiner, which operates as an intermediate step between the mapper and reducer stages.

```
hadoop jar
$HADOOP_HOME/share/hadoop/tools/lib/hadoop-streaming-3.3.5.ja
r \
-file <path_to_mapper.py> \
-mapper <path_to_mapper.py> \
-file <path_to_reducer.py> \
-reducer <path_to_reducer.py> \
-file <path_to_combiner.py> \
-combiner <path_to_combiner.py> \
-file <path_to_stopword.txt> \
-input <hadoop_path_to_data_16GB.txt> \
-output /output/16gb/case2
```

Job Overview	
Job Name:	streamjob2613114076023928026.jar
User Name:	sandy
Queue:	default
State:	SUCCEEDED
Uberized:	false
Submitted:	Sat May 13 09:52:06 PDT 2023
Started:	Sat May 13 09:52:13 PDT 2023
Finished:	Sat May 13 10:20:42 PDT 2023
Elapsed:	28mins, 29sec
Diagnostics:	
Average Map Time	1mins, 14sec
Average Shuffle Time	22mins, 18sec
Average Merge Time	5sec
Average Reduce Time	12sec

ApplicationMaster

Attempt Number	Start Time	Node	Logs
1	Sat May 13 09:52:08 PDT 2023	172.20.205.169:8042	logs

Task Type	Total	Complete
Map	115	115
Reduce	1	1

Attempt Type	Failed	Killed	Successful
Maps	0	1	115
Reduces	0	0	1

By incorporating the combiner, we observed a remarkable reduction in overall execution time, with a decrease of approximately 40% compared to Case1. This improvement can be attributed to the combiner's ability to locally aggregate and compress the intermediate key-value pairs generated by the mapper, reducing the data volume transferred over the network and optimizing the reducer's workload. The utilization of a combiner not only minimizes network congestion and data transfer, but also optimizes the reducer's workload by performing an initial aggregation step locally.

3) Case 3

For Case3, our objective was to evaluate the impact of employing multiple reducers on reducing the execution time of the data processing task. By distributing the workload across multiple reducers, we aimed to parallelize the computation and potentially achieve a faster overall processing time. In this experiment, we increased the number of reducers and monitored the execution time to assess the effectiveness of this approach. By dividing the data into multiple partitions and assigning each partition to a separate reducer, we expected to leverage parallel processing capabilities and potentially achieve a significant reduction in execution time.

```
hadoop jar
$HADOOP_HOME/share/hadoop/tools/lib/hadoop-streaming-3.3.5.ja
r \
-D mapred.reduce.tasks=50 \
-file mapper.py \
-mapper mapper.py \
-file reducer.py \
-reducer reducer.py \
-file combiner.py \
-combiner combiner.py \
-file stopword.txt \
-input /wordcount/input/data_16GB.txt \
-output /wordcount/output/16gb/case3-temp \
&&\
hadoop jar
$HADOOP_HOME/share/hadoop/tools/lib/hadoop-streaming-3.3.5.ja
r \
-file sort_map.py \
-mapper sort_map.py \
-file sort_reduce.py \
-reducer sort_reduce.py \
```

```
-file combiner.py \
-combiner combiner.py \
-input /wordcount/output/16gb/case3b-temp \
-output /wordcount/output/16gb/case3b
```

	Job Overview
Job Name:	streamjob9167437224214791307.jar
User Name:	sandy
Queue:	default
State:	SUCCEEDED
Uberized:	false
Submitted:	Sun May 14 12:08:21 PDT 2023
Started:	Sun May 14 12:08:30 PDT 2023
Finished:	Sun May 14 12:58:08 PDT 2023
Elapsed:	49mins, 37sec
Diagnostics:	
Average Map Time	55sec
Average Shuffle Time	45sec
Average Merge Time	0sec
Average Reduce Time	1sec

ApplicationMaster

Attempt Number	Start Time	Node	Logs
1	Sun May 14 12:08:26 PDT 2023	100.76.111.243:8042	logs

Task Type	Total	Complete	
Map	115	115	
Reduce	50	50	
Attempt Type	**Failed**	**Killed**	**Successful**
Maps	0	0	115
Reduces	0	0	50

However, upon analyzing the results, we found that the expected improvement in execution time was not realized. Despite using multiple reducers, the overall processing time did not demonstrate a substantial reduction compared to previous cases.

This outcome suggests that the specific characteristics of the data, the nature of the computation, or other factors may have limited the benefits of employing multiple reducers in this particular scenario. Further analysis is required to understand the underlying reasons for the lack of performance improvement and to identify potential optimizations that could enhance the efficiency of the data processing task.

Based on the results of our experimentation, it can be concluded that Case 2, which involved the utilization of a combiner in the data processing pipeline, exhibited better performance compared to Case 3, which employed multiple reducers.

In Case 2, the introduction of the combiner allowed for partial aggregation of intermediate key-value pairs within the mapper phase itself. This reduced the amount of data transferred between the mappers and reducers, leading to a more efficient data processing workflow. As a result, the overall execution time was significantly reduced by approximately 40% compared to Case 1.

IV. Conclusion

The primary objective of this project was to efficiently process a large dataset using the Hadoop Data Processing Framework - MapReduce, and extract the top 100 words in parallel across clusters. To achieve this, we implemented nlargest and heapq algorithms, along with the sort_mapper and sort_reducer files, which helped streamline the sorting process and identify the most frequently occurring words.

After running several test cases, we identified areas for improvement in our program, including adding a combiner function, making parameter changes in the .xml files, and setting up the Hadoop LZO environment to enhance performance further. We were able to achieve significant reductions in processing time by optimizing hardware and software performance. For instance, in case one from part three, we increased the amount of data that can be spilled to disk during the sort and shuffle phase, reducing processing time from over an hour to just 47 minutes. In case two, by adding a combiner function and strictly coding it, we were able to achieve a 40% reduction in processing time.

This project demonstrates the power of MapReduce in processing large datasets and the various techniques that can be employed to optimize performance. Throughout this project, we have gained insights into how MapReduce processes large volumes of data in a parallel and scalable manner, while also simplifying complex computations through effective management of Mappers and Reducers. By constantly evaluating and improving our methods, we can continue to refine our approach and tackle even more complex big data challenges in the future.

V. Reference

Apache Hadoop 3.3.5 – MapReduce Tutorial
Hadoop Streaming
Pusukuri, Kishore. COEN 242 Class Slides. Santa Clara University
Hadoop LZO
Hadoop Mapreduce Client

Top100 Words for 16GB.txt

said 16983038
would 5829109
one 5827854
new 5619251
also 4618231
us 4602724
people 4302078
last 4096906
year 4011803
two 3964144
first 3839807
mr 3746747
years 3637652
time 3635760
could 3374880
like 3058690
government 2520240
says 2417351
may 2393105
get 2346949
many 2344542
back 2295703
million 2255486
three 2250200
president 2193692
even 2175400
made 2130243
make 2104945
told 2092903
since 2041708
world 2039952
percent 2022481
police 2014250
well 1965773
still 1956831
state 1948101
way 1936600
much 1878812
say 1866770
day 1831556
week 1829329
home 1827482
take 1785403
per 1779050
work 1770896
going 1744159
think 1670557
company 1665814
good 1635409
dont 1629480
next 1605312
including 1594887
see 1561688
states 1543254
another 1536907
group 1521302
go 1508945
public 1506218
de 1499434
house 1494732
around 1482330
city 1474866
former 1461398
part 1446451
second 1433375
national 1415826
obama 1405174
know 1386991
want 1379621
game 1371927
four 1371032
news 1368259
found 1366151
end 1353926
right 1336449
court 1330434
team 1329924
united 1310698
need 1307171
report 1299343
country 1295488
help 1289802
according 1281617
business 1275145
market 1267854
life 1256423
long 1242001
set 1232190
months 1227463
man 1226580
best 1211261
come 1208192
cent 1203960
officials 1195387
family 1186525
left 1178734
high 1177794
show 1172154
health 1163668
york 1161016

Top100 Words(>6) for 16GB.txt

government	2520240
million	2255486
president	2193692
percent	2022481
company	1665814
including	1594887
another	1536907
national	1415826
country	1295488
according	1281617
business	1275145
officials	1195387
american	1129214
international	1116052
financial	1097729
support	1080825
children	1072801
however	1072387
security	1058065
billion	1049368
without	1039688
political	1018915
european	1018667
whether	1014215
yearold	981085
tuesday	969122
already	960627
minister	942536
information	927619
wednesday	920376
reported	917579
expected	916793
economic	910616
thursday	910329
something	896500
military	893360
several	877379
companies	869826
washington	866772
members	861953
service	853089
federal	852384
campaign	820329
statement	817540
economy	816287
british	813789
university	806441
services	799299
director	788859
important	773502
working	766645
players	755612
countries	740881
decision	732901
earlier	732872
different	722896
general	722268
research	716089
capital	705038
saturday	702016
believe	701746
industry	700059
department	699764
executive	696821
quarter	696623
following	665860
although	662413
foreign	654706
despite	652919
official	643514
council	639609
election	635236
announced	632756
looking	630482
possible	629503
thought	628703
control	628021
current	624576
program	624140
getting	623774
interest	612373
outside	605728
england	599716
minutes	599437
spokesman	596504
continue	594807
increase	588451
private	586730
authorities	584908
meeting	578137
history	576197
together	575985
results	575269
community	571257
reports	565968
leaders	565284
problem	562464
hospital	559119
process	556804
markets	555948

Top100 Words for 2.5GB.txt

said	2616266
one	949555
would	917212
new	852822
also	727934
last	700735
people	688828
us	670794
mr	659562
year	654244
de	643122
two	637405
first	616295
years	608882
time	602016
could	548856
like	484097
says	419576
government	405470
back	398012
get	397140
may	372870
police	371212
per	367298
la	360237
three	360087
told	359229
many	358774
made	344154
even	335035
make	333551
well	326187
world	319405
million	318478
since	318416
way	316957
still	312846
day	305766
home	304861
week	299482
much	296828
cent	294331
work	294328
le	284535
take	284214
going	283188
good	278834
say	278437
dont	277780
think	274485
next	269513
state	264274
around	259091
see	258524
including	250803
president	250617
go	249882
another	249601
found	245247
company	244010
former	239811
public	236849
part	235478
court	234684
group	234072
team	232873
man	231083
know	230360
game	230229
city	227040
house	225066
want	224799
second	223861
end	222602
right	221258
four	220644
life	218625
national	216146
family	214071
best	207276
news	206894
need	205295
help	204715
set	202012
come	201761
got	201136
long	200086
according	199469
business	199119
left	197811
les	196404
et	195339
months	193875
really	193662
used	193435
market	192509
percent	191929
im	191567
big	189219
place	188953

Top100 Words(>6) for 2.5GB.txt

government	405470
million	318478
including	250803
president	250617
another	249601
company	244010
national	216146
according	199469
business	199119
percent	191929
children	186405
country	185323
yearold	174923
however	171875
minister	171512
support	160054
without	159819
whether	157119
international	155251
something	150041
already	147866
players	147136
officials	145338
security	143445
expected	142353
american	140006
billion	139672
financial	135819
british	134480
political	133593
information	133546
australia	133494
australian	133022
university	132830
service	132019
several	127788
working	126900
tuesday	126216
members	126102
federal	125761
director	125292
companies	123237
statement	123219
military	122988
thursday	122559
different	122406
research	121958
wednesday	121808
services	120953
earlier	120110
decision	120034
following	118936
important	116119
reported	113628
executive	112042
despite	112028
hospital	111788
thought	111391
economic	108564
looking	108465
industry	108024
economy	107246
department	105927
england	105272
saturday	104744
getting	103814
believe	103005
although	102923
minutes	102270
outside	99852
washington	99784
possible	97903
capital	97281
control	96887
european	96292
general	96204
campaign	94970
started	94334
foreign	93788
together	93721
continue	93269
morning	92841
interest	92423
michael	92310
announced	92247
history	91906
playing	90794
council	90489
evidence	90275
forward	89655
community	89601
official	89275
current	89202
spokesman	87907
building	87419
private	87286
released	86908
countries	86402
parents	86343
election	85852

Top100 Words for 300MB.txt

european	318743
mr	210690
would	181912
also	180118
commission	172783
must	156856
president	152134
union	130344
states	130270
member	126301
parliament	122059
report	119173
like	111176
council	107720
one	102551
europe	95645
countries	93490
us	91379
eu	86935
need	84622
policy	82662
important	81684
new	81139
people	78712
time	78348
rights	69717
therefore	68313
support	68006
however	65871
take	64840
make	64650
work	62135
economic	60152
committee	59456
political	56396
made	55424
first	54666
many	53998
social	53696
way	53401
believe	52845
development	52466
commissioner	51201
say	50620
proposal	50250
market	49889
fact	48978
debate	48595
human	48400
group	46352
question	45847
think	45818
agreement	45493
years	45489
even	45344
issue	44464
citizens	44331
point	44227
future	43162
order	43091
situation	42969
public	42716
financial	42361
well	42255
right	42231
measures	42147
two	42045
cannot	42028
could	41419
possible	41337
community	41307
want	41292
today	40822
national	40257
international	40222
country	39653
directive	39575
said	39528
state	39025
within	38778
already	38700
much	38680
cooperation	38527
good	38321
common	38316
vote	38217
world	38038
part	37862
clear	37847
members	37768
mrs	37471
may	37147
still	36947
year	36773
particular	36433
use	36111
know	36086
see	35543
system	34983
energy	34781

Top100 Words(>6) for 300MB.txt

european	318743
commission	172783
president	152134
parliament	122059
council	107720
countries	93490
important	81684
therefore	68313
support	68006
however	65871
economic	60152
committee	59456
political	56396
believe	52845
development	52466
commissioner	51201
proposal	50250
question	45847
agreement	45493
citizens	44331
situation	42969
financial	42361
measures	42147
possible	41337
community	41307
national	40257
international	40222
country	39653
directive	39575
already	38700
cooperation	38527
members	37768
particular	36433
security	33259
information	32899
without	32726
process	32647
protection	31263
problem	30630
example	30564
gentlemen	30454
amendments	30308
problems	29849
government	29573
rapporteur	29433
particularly	29247
programme	29047
necessary	28937
resolution	28790
position	27628
respect	27523
whether	26696
framework	26478
presidency	26427
services	25850
regulation	25611
proposals	25403
amendment	25115
certain	24890
adopted	24128
legislation	23702
different	23433
strategy	22929
institutions	22850
decision	22848
present	22830
progress	22575
subject	22475
continue	22407
opinion	22225
employment	22066
transport	21805
working	21633
general	21518
concerned	21513
another	21287
towards	21233
welcome	21101
research	20969
something	20907
approach	20826
environment	20733
commissions	20637
current	20384
forward	20348
resources	20189
conditions	20158
clearly	20086
finally	20040
together	20040
including	20030
industry	20024
principle	20000
procedure	19964
fundamental	19823
authorities	19651
importance	19645
proposed	19548
specific	19452
policies	19429