

CS560 Programming Assignment 2 Report

Parker Diamond, Ksenia Burova

March 31st, 2018

NOTE: Sections 1 and 2 use CloudLab systems while Sections 3-5 use the VolSec Lab server. Be wary of any discrepancies in commands and file locations!

Overview

In this project, the goal is to gain practical experience on MapReduce programming through the implementation of MapReduce parallel data processing system on the CloudLab cloud computing platform. This will allow us to observe the performance implications of parallel data processing. The main task of this project is to implement an important component of an online search engine - the capability for users to search full-text from a document. For the first step, this assignment involves creating a full inverted index of the contents of the document used as input. The collection of the complete works of Shakespeare will be used as testing input. For the second step, we are going to learn how to write a query program on top of our full inverted file index.

1 Platforms and resources used

There were three distributed system platforms used for the purposes of this assignment: CloudLab, Hadoop MapReduce, and Spark.

We were given a link to complete works of Shakespeare. But all of the .txt files in that source had enormous amount of junk text attached in the beginning of each file. We assumed that could influence results of our project so we preferred to use another source that can be found [here](#).

2 Identifying and removing stop words

For this part of the assignment, we have to write a word count function to perform a word count over a corpus of text files and to identify stop words. We chose to use MapReduce to generate a list of stop words based on frequency

There are two programs to count words: `mapper.py` and `reducer.py`. `stopwords.py` generates the list of stop words.

- `map_stop_words.py` - reads and filters input, handles all the punctuation and capitalization. We mentioned that besides regular punctuation there are many other possible special characters can be found in file. We handle removal of all the special characters, plus we check for words ending

with apostrophes (e.g. 'ed, 's, 'st, 'll and etc.). Mapper basically outputs every single word found in files with count of 1 for each.

- `reduce_stop_words.py` - counts occurrence of each word. There is nothing different from regular reduce function.
- `stopwords.py` - specifies the limit for word frequency, iterates through all the words with total counts, sorts them in decreasing order and saves the top 'limit' words as a list of stop words into 'list.txt'. It returns the frequency of the least popular word on this list as a threshold.

We run first two functions in *hadoop* by using this command:

```
hadoop jar /usr/local/hadoop-2.7.3/share/hadoop/tools/lib/hadoop-streaming-2.7.3.jar \  
-files /users/username/map_stop_words.py,/users/username/reduce_stop_words.py \  
-mapper map_stop_words.py -reducer reduce_stop_words.py \  
-input /user/username/books/ \  
-output /user/username/output
```

After reduction is saved into `/user/username/output/part-00000` file on hdfs, we can access it to use the data to calculate stop words. We run the program this way:

```
user@resourcemanager:~$ python stopwords.py < part-00000  
Threshold: 1308
```

Let's take a look at some of the words in `list.txt`:

```
user@resourcemanager:~$ cat list.txt  
the  
and  
i  
to  
of  
a  
you  
my  
that  
in  
is  
not  
with  
he  
me  
it  
for  
his  
be  
this  
your  
.  
.  
.  
.  
such  
out
```

```
where
who
give
some
these
th
too
```

So, seems like the list is pretty accurate, since these are the common stop words. The stop words are written to a file that is passed to the map tasks that will construct the inverted index.

3 Building the inverted index

The inverted index is constructed with MapReduce and written to a flat file. The input files are assumed to be under 64MB in size, which guarantees that every map task receives a complete file and that no file is split between multiple map tasks. Since Shakespeare's individual works are simple plain text files (and not remarkably long), we create a file for each work that is well under 64MB.

3.1 Map

The mapper program for the inverted index construction is contained in `map.py`, and it performs two tasks:

1. Removing stop words and punctuation
2. Indexing the word by file name and line number

The file name is retrieved from the environment variable `map_input_file`, which is defined by Hadoop at the start of the MapReduce program. The file name is assumed to be splittable (no spurious slashes in the name) such that only the relative path is output. Since the map task is written in Python, it uses the streaming JAR to receive a file over standard input. For each file it receives, it reads the file from stdin line by line, keeping a 1-indexed count of which line it is on during the reading. For each word on the line, split by whitespace, it performs three tasks:

1. Removes apostrophes from the word
2. Removes other miscellaneous punctuation from the word
3. If the word is not empty and not a stop word, index it.

The map program then prints the index of the word in the file in the following format: word, file, line_number

3.2 Reduce

The reducer code for the inverted index construction is in `reduce.py` and is extremely simple – it maintains a set of the 3-tuples (stored as a line) it has seen, sorts them, and then prints them. It uses a set because there may be a word that is used twice on one line in the same document, and we do not care about such cases, so a set eliminates duplicates.

3.3 MapReduce

We now combine their functionality with map reduce. We first run the Hadoop MapReduce streaming JAR with our mapper and reducer and stop word list:

```
cubia@erebus:~/MapReduce
stopwords.py stop_words.txt
[cubia@erebus MapReduce]$ sudo rm -rf output/
[cubia@erebus MapReduce]$ sudo -E hadoop jar /usr/lib/hadoop-3.0.0/share/hadoop/tools/lib/hadoop-streaming-3.0.0.jar -files map.py,reduce.py,stop_words.txt -mapper map.py -reducer reduce.py -input Shakespeare/ -output output
```

Hadoop MapReduce then splits up the inputs (Shakespeare's works) into records and gives them to the map tasks. They perform the filtering and indexing. Hadoop MapReduce then shuffles them into bins based on their keys and gives the shuffled outputs to the reducer tasks, which sort and emit the results. The result summary is given below:

```
cubia@erebus:~/MapReduce
IO_ERROR=0
WRONG_LENGTH=0
WRONG_MAP=0
WRONG_REDUCE=0
File Output Format Counters
  Bytes Written=11155509
2018-03-31 21:32:56,093 INFO mapred.LocalJobRunner: Finishing task: attempt_local47010517_0001_r_000000_0
2018-03-31 21:32:56,093 INFO mapred.LocalJobRunner: reduce task executor complete.
2018-03-31 21:32:56,571 INFO mapreduce.Job: map 100% reduce 100%
2018-03-31 21:32:56,572 INFO mapreduce.Job: Job job_local47010517_0001 completed successfully
2018-03-31 21:32:56,590 INFO mapreduce.Job: Counters: 30
File System Counters
  FILE: Number of bytes read=174402311
  FILE: Number of bytes written=363017873
  FILE: Number of read operations=0
  FILE: Number of large read operations=0
  FILE: Number of write operations=0
Map-Reduce Framework
  Map input records=182689
  Map output records=571865
  Map output bytes=11179805
  Map output materialized bytes=12323787
  Input split bytes=4105
  Combine input records=0
  Combine output records=0
  Reduce input groups=565662
  Reduce shuffle bytes=12323787
  Reduce input records=571865
  Reduce output records=565662
  Spilled Records=1143730
  Shuffled Maps =42
  Failed Shuffles=0
  Merged Map outputs=42
  GC time elapsed (ms)=189
  Total committed heap usage (bytes)=84077969408
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=5510268
File Output Format Counters
  Bytes Written=11155509
2018-03-31 21:32:56,590 INFO streaming.StreamJob: Output directory: output/
[cubia@erebus MapReduce]$
```

The inverted index is then output to `output/part-00000`, and it is assumed to be at this location when the inverted index is queried. A `tail -n 30` on the output file will gives us the files and line numbers of

the last 30 words (ascending order) that appear in the corpus.

```
cubia@erebus:~/MapReduce
[cubia@erebus MapReduce]$ tail -n 30 output/part-00000
zeal,R2.txt,3899
zeal,R3.txt,3792
zeal,R3.txt,3924
zeal,TGV.txt,1537
zeal,TNK.txt,813
zeal,Tim.txt,1791
zeal,Tim.txt,3391
zeal,Tit.txt,740
zeal,Tro.txt,3591
zeal,Tro.txt,3754
zeal,WT.txt,3982
zealous,AWW.txt,2334
zealous,Jn.txt,1122
zealous,Jn.txt,502
zealous,LLL.txt,3180
zealous,R3.txt,3775
zealous,Son.txt,461
zeals,Tim.txt,742
zed,Lr.txt,1803
zenith,Tmp.txt,439
zephyrs,Cym.txt,3504
zir,Lr.txt,4501
zir,Lr.txt,4514
zo,Lr.txt,4507
zodiac,Tit.txt,864
zodiacs,MM.txt,433
zone,Ham.txt,5374
zounds,1H4.txt,1014
zounds,1H4.txt,4457
zwaggered,Lr.txt,4506
[cubia@erebus MapReduce]$
```

We can now double check that these line numbers are correct by executing a `head -n [line_number] [file] | tail -n 1` and checking for the existence of the word on that line. See below that for the word "zeal" appears on the line and document as indicated by the inverted index contained in `output/part-00000`.

```

cubia@erebus:~/MapReduce
[cubia@erebus MapReduce]$ tail -n 30 output/part-00000
zeal,R2.txt,3899
zeal,R3.txt,3792
zeal,R3.txt,3924
zeal,TGV.txt,1537
zeal,TNK.txt,813
zeal,Tim.txt,1791
zeal,Tim.txt,3391
zeal,Tit.txt,740
zeal,Tro.txt,3591
zeal,Tro.txt,3754
zeal,WT.txt,3982
zealous,AWW.txt,2334
zealous,Jn.txt,1122
zealous,Jn.txt,502
zealous,LLL.txt,3180
zealous,R3.txt,3775
zealous,Son.txt,461
zeals,Tim.txt,742
zed,Lr.txt,1803
zenith,Tmp.txt,439
zephyrs,Cym.txt,3504
zir,Lr.txt,4501
zir,Lr.txt,4514
zo,Lr.txt,4507
zodiac,Tit.txt,864
zodiacs,MM.txt,433
zone,Ham.txt,5374
zounds,1H4.txt,1014
zounds,1H4.txt,4457
zwaggered,Lr.txt,4506
[cubia@erebus MapReduce]$ head -n 3899 Shakespeare/R2.txt | tail -n 1
Ours of true zeal and deep integrity.
[cubia@erebus MapReduce]$ head -n 3792 Shakespeare/R3.txt | tail -n 1
Of thy devotion and right Christian zeal.
[cubia@erebus MapReduce]$ head -n 3924 Shakespeare/R3.txt | tail -n 1
If you refuse it, as in love and zeal
[cubia@erebus MapReduce]$ █

```

4 Query the inverted index

The software to query the inverted index is contained in `query.py`. The query program is straightforward. The program is run with the command `python query.py [mapreduce-output]`. The program simply reads the mapreduce output file passed by command line and builds a Python dictionary of the words. Each word is a key which corresponds to a list of values, which are document/line number pairs. Since a Python dict uses a hashmap for mapping keys to values, the look-up time is constant time. Since the corpus of the inverted index is rather small, the construction of the dictionary is fast. For larger corpi, a distributed solution might be more appropriate.

The program then prompts the user to query the locations of a word in an infinite loop until EOF is given.

```

cubia@erebus:~/MapReduce
[cubia@erebus MapReduce]$ ls
hadoop_commands.txt  map.py  output  query.py  README.md  re
[cubia@erebus MapReduce]$ python query.py output/part-00000
Word to search for: zeal
('1H4.txt', '3609')
('1H4.txt', '4382')
('2H4.txt', '2065')
('2H4.txt', '3176')
('2H4.txt', '4778')
('2H6.txt', '2258')
('3H6.txt', '4406')
('Ado.txt', '1263')
('H5.txt', '965')
('H8.txt', '1484')
('H8.txt', '3189')
('Jn.txt', '1181')
('Jn.txt', '1314')
('Jn.txt', '2245')
('Jn.txt', '3352')
('Jn.txt', '840')
('LLL.txt', '2394')
('LLL.txt', '2548')
('LLL.txt', '4010')
('LLL.txt', '4011')
('MV.txt', '3890')
('R2.txt', '133')
('R2.txt', '3899')
('R3.txt', '3792')
('R3.txt', '3924')
('TGV.txt', '1537')
('TNK.txt', '813')
('Tim.txt', '1791')
('Tim.txt', '3391')
('Tit.txt', '740')
('Tro.txt', '3591')
('Tro.txt', '3754')
('WT.txt', '3982')
Word to search for:

```

Querying the word 'zeal' shows us the documents and line numbers where that word appears – we saw a few of these earlier. A user can continue querying for words after.

5 Query with Spark

We integrate the inverted index query system with Spark using PySpark – a Python API for Spark. We want users to be able to query for words interactively, so we cannot run a standalone script on PySpark since it will redirect STDIN. Instead, we create a wrapper for our query system and load it as a module in PySpark.

The code for the Spark integration is in `sparkWrapper.py` and is similar to the code in `query.py`. In `sparkWrapper.py`, though, the inverted index dictionary is built on a Spark RDD. To run the queries on Spark, start the `pyspark` executable and then enter the following commands on the interactive terminal:

```

>>> import sparkWrapper
>>> sparkWrapper.start(spark)

```

These commands will load in the Python file as a module and pass the Spark session to the query system. The user will then enter the same query loop as in section 4:

[illegible]

We can check that our output matches that of the previous section again.

```

cubia@erebus:~/MapReduce
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
Welcome to

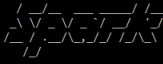
  ____
 /  _ \
/_/_/ \_/_/ version 2.3.0

Using Python version 3.6.4 (default, Jan 5 2018 02:35:40)
SparkSession available as 'spark'.
>>> import sparkWrapper
>>> sparkWrapper.start(spark)
Query the Location of a Word: zeal
('1H4.txt', '3609')
('1H4.txt', '4382')
('2H4.txt', '2065')
('2H4.txt', '3176')
('2H4.txt', '4778')
('2H6.txt', '2258')
('3H6.txt', '4406')
('Ado.txt', '1263')
('H5.txt', '965')
('H8.txt', '1484')
('H8.txt', '3189')
('Jn.txt', '1181')
('Jn.txt', '1314')
('Jn.txt', '2245')
('Jn.txt', '3352')
('Jn.txt', '840')
('LLL.txt', '2394')
('LLL.txt', '2548')
('LLL.txt', '4010')
('LLL.txt', '4011')
('MV.txt', '3890')
('R2.txt', '133')
('R2.txt', '3899')
('R3.txt', '3792')
('R3.txt', '3924')
('TGV.txt', '1537')
('TNK.txt', '813')
('Tim.txt', '1791')
('Tim.txt', '3391')
('Tit.txt', '740')
('Tro.txt', '3591')
('Tro.txt', '3754')
('WT.txt', '3982')
Query the Location of a Word:

```


A user can also enter CTRL-D to raise an EOFError and exit the input loop. This will allow the user to make additional modifications to the RDD (located in the `sparkWrapper.inv_index` object) via the interactive terminal without reloading the inverted index.

```
[cubia@rebus MapReduce]$ /bin/gypspark
Python 3.6.4 (default, Jan 5 2018, 02:35:40)
[GCC 7.2.1 20171224] on linux
Type "help", "copyright", "credits" or "license" for more information.
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
Welcome to

 version 2.3.0

Using Python version 3.6.4 (default, Jan 5 2018 02:35:40)
SparkSession available as 'spark'.
>>> import sparkWrapper
>>> sparkWrapper.start(spark)
Inverted Index RDD is in object "inv_index"
Query the Location of a Word: Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
    File "/home/cubia/MapReduce/sparkWrapper.py", line 19, in start
      line = input('Query the Location of a Word: ')
EOFError

>>> print(dir(sparkWrapper.inv_index))
['_add', '_class', '_delattr', '_dict', '_dir', '_doc', '_eq', '_format', '_ge', '_getattr', '_getnewargs', '_gt', '_hash', '_init', '_init_subclass',
'_le', '_lt', '_module', '_ne', '_new', '_reduce', '_reduce_ex', '_repr', '_setattr', '_sizeof', '_str', '_subclasshook', '_weakref', '_bypass_serializer',
'_computeFractionForSampleSize', '_defaultReducePartitions', '_id', '_is_pipelinaable', '_jrdd', '_jrdd_deserializer', '_jrdd_val', '_memory_limit', '_pickled', '_prev_jrdd', '_prev_jrdd_deseriali',
'_reerialize', '_to_java_object_rdd', '_aggregate', '_aggregateByKey', '_cache', '_cartesian', '_checkpoint', '_coalesce', '_cogroup', '_collect', '_collectAsMap', '_combineByKey', '_context', '_count',
'_countApprox', '_countApproxDistinct', '_countByKey', '_countByValue', '_ctx', '_distinct', '_filter', '_first', '_flatMap', '_flatMapValues', '_fold', '_foldByKey', '_foreach', '_foreachPartition', '_fullOuter',
'_join', '_func', '_getCheckpointFile', '_getNumPartitions', '_getStorageLevel', '_glom', '_groupBy', '_groupByKey', '_groupWith', '_histogram', '_id', '_intersection', '_isCheckpointed', '_isEmpty', '_isLocallyCh',
'_isCheckpointed', '_is_cached', '_is_checkpointed', '_join', '_keyBy', '_keys', '_leftOuterJoin', '_localCheckpoint', '_lookup', '_map', '_mapPartitions', '_mapPartitionsWithIndex', '_mapPartitionsWithSplit', '_mapV',
'_values', '_max', '_mean', '_meanApprox', '_min', '_name', '_partitionBy', '_partitioner', '_persist', '_pipe', '_preservesPartitioning', '_prev', '_randomSplit', '_reduce', '_reduceByKey', '_reduceByKeyLocally', '_r',
'_repartition', '_repartitionAndSortWithinPartitions', '_rightOuterJoin', '_sample', '_sampleByKey', '_sampleStddev', '_sampleVariance', '_saveAsHadoopDataset', '_saveAsHadoopFile', '_saveAsNewAPIHadoopDataset',
'_saveAsNewAPIHadoopFile', '_saveAsPickleFile', '_saveAsSequenceFile', '_saveAsTextFile', '_setName', '_sortBy', '_sortByKey', '_stats', '_stddev', '_subtract', '_subtractByKey', '_sum', '_sumApprox', '_take',
'_takeOrdered', '_takeSample', '_toE', '_toDebugString', '_toLocalIterator', '_top', '_treeAggregate', '_treeReduce', '_union', '_unpersist', '_values', '_variance', '_zip', '_zipWithIndex', '_zipWithUniqueId']
>>>
```