Map-Reduce Approach to Anagram Problem

Identifying the Anagrams in a Text File

What are anagrams ?

MARY is a word and **ARMY** is another word which is formed by re arranging the letters in the original word MARY

- **MARY** and **ARMY** are ***anagrams***

- **POOL** and **LOOP** are ***anagrams***. There could a lot of such examples.

Note : We are interested in finding out anagram combinations from a text document which does not contain irrelevant gibberish words

Problem Statement

To identify and list all the anagrams found in a document. Eg A book (a novel)

Input file name : **sample.txt** (a file in text format) and has 2 lines in the file.

File contents: mary worked in army
the loop fell into the pool

Expected output : (must contain all the anagrams)

mary army

loop pool

Output of the Record Reader

This is going to the be output of the record reader after reading the first line of the file

Contents of the file : mary worked in army
loop fell into the pool

KEY	VALUE
file offset	**entire line of the file**
0	mary worked in army

The above (key-value pair) is now going to be fed into the mapper as an input.

Programming the Mapper

Mapper is programmed do the following

Step 1: Ignore the key from the record reader

Step 2: Split the words in the value (the full line)

mary works in army

[mary] [works] [in] [army] (the line is split)

Step 3: Compute the word length of each word

Step 4: Output the word length as key and original word as value . The sample output of mapper would look like

KEY	VALUE
4	**mary**
5	**works**
2	**in**

Step 5: Repeat the above steps for all the words in the line

Output of the Mapper After Processing the Entire File

KEY	VALUE
4	mary
6	worked
2	in
3	the
4	army
4	loop
4	fell
4	into
3	the
4	pool

Output After Sorting the Keys

KEY	VALUE
2	in
3	the
3	the
4	mary
4	army
4	loop
4	fell
4	into
4	pool
6	worked

Output After Shuffling the Keys (aggregation of duplicate keys)

KEY	VALUE
2	in
3	the, the
4	mary , army , loop, fell , into, pool
6	worked

This is the input to the REDUCER

KEY	VALUE
2	in
3	the, the
4	mary , army , loop, fell , into, pool
6	worked

What can we do in the Reducer now to identify the Anagrams ?

KEY	VALUE
2	in
3	the, the
4	mary , army , loop, fell , into, pool
6	worked

- **Pick one word at a time from the list of values for every key value pair**
- **Check if the same combination of letters are present in every other word in the list**

i.e the letters m,a,r and y is present in amry, if true then mary and army are anagrams

- **How to revolve *the* and *the* as both contain the same combination of alphabets ?**

Its simple ..we can choose do a string comparison and if the strings are identical then we can ignore them !

Problem With This Approach

- This looks like a solution however has several challenges
- Consider the below key value pair

    4 mary , army , loop, fell , into, pool

- To compare the alphabet combinations m,a,r and y is present in one other word takes 4 X 4 = 16 comparisons
- 16 comparison operation multiplied by number of words in the value list = 16 X 6 = 96 comparison operations
- What is the list it too long ? This just worsens the computation time in the event of large data sets (big data)
- Reducer is overloaded here !
- What seemed as a solution, is not so practical approach for BIG DATA set

An alternate approach would be .....

Output of the Record Reader

This is going to the be output of the record reader after reading the first line of the file

Contents of the file : mary worked in army
loop fell into the pool

KEY	VALUE
file offset	**entire line of the file**
0	mary worked in army

The above (key-value pair) is now going to be fed into the mapper as an input.

Programming the Mapper

Mapper is programmed do the following

Step 1: Ignore the key from the record reader

Step 2: Split the words in the value (the full line)

mary works in army

[mary] [works] [in] [army] (the line is split)

Step 3: sort each word in dictionary order (lexicographic ordering)

mary after sorting would become ***amry***

Step 4: Output the sorted word as key and original word as value . The sample output of mapper would look like

KEY	VALUE
amry	**mary**

Step 5: Repeat the above steps for all the words in the line

Output of the Mapper after processing the entire file

KEY	VALUE
amry	mary
dekorw	worked
in	in
eht	the
amry	army
loop	loop
efll	fell
inot	into
eht	the
loop	pool

Output after sorting the keys

KEY	VALUE
amry	mary
amry	army
dekorw	worked
eht	the
eht	the
in	in
inot	into
loop	loop
loop	pool
efll	fell

Output after shuffling the keys (aggregation of duplicate keys)

KEY	VALUE
amry	mary, army
dekorw	worked
eht	the,the
in	in
inot	into
loop	loop, pool
efll	fell

Observation and Inference

KEY	VALUE
amry	mary, army
dekorw	worked
eht	the,the
in	in
inot	into
loop	loop, pool
efll	fell

ANY BULBS LIGHTING UP ?

LOOK CLOSER !

There are some keys with more than one value. We need to only look at such key, value pairs

amry	**mary ,army**
eht	**the , the**
loop	**loop , pool**

Logic to list the anagrams

amry	mary ,army
eht	the,the
loop	loop,pool

Problem : We need to only print the values belonging to keys “**amry**” and “**loop**” since only their values qualify for anagrams.

We need to ignore the values belonging to the keys “**eht**” since its corresponding values do not qualify for being anagrams.

How to ignore the non anagram values ?

greatlearning
Learning for Life

amry	mary ,army
eht	the ,the
loop	loop, pool

We need to program the following into the reducer

Step 1: Check if the number of values are > 1 for each key

Step 2: Compare the first and second value in the values list for every key, if they match, ignore them.

key	val1	val2
eht	the	the

Step 3: If the values don't match in step 2. Compose a single string comprising of all the values in the list and print it to the output file. This final string is the KEY and value can be NULL (do not print anything for value)

KEY = "mary army" VALUE =" "

Step 4: Repeat the above steps for all the key value pairs input to the reducer.

Final Output of the Reducer

mary army

loop pool

The above output is for the case of a file with just 2 lines of data.

What if the file is 640MB in size ?

How does map reduce help in speeding up the job completion ?

HOW DOES MAP REDUCE SPEED UP THE PROCESSING ?

What is the input file used in this example is 640MB instead of just containing 2 lines ?

- The HADOOP framework would first split the entire file into 10 blocks each of 64MB
- Each 64MB block would be treated as a single file
- There would be one record-reader and one mapper assigned to each such block
- Output of all the mappers would finally reach the reducer (one reducer is used by default) however we can have multiple reducers depending on degree of optimization required

Why MapReduce?

- Scale *out* not scale *up:* MR is designed to work with commodity hardware
- *Move* code where the data is: cluster have limited bandwidth
- *Hide* system-level details from developers: no more race condition, dead locks etc
- Separating the *what* from *how:* developer specifies the computation, framework handles actual execution
- *Failures* are common and handled automatically
- Batch processing: access data sequentially instead of random to avoid locking up
- Linear Scalability: once the MR algorithm is designed, it can work on any size cluster
- *Divide* & *Conquer*: MR follows Partition and Combine in Map/Reduce phase
- High-level *system details*: monitoring of the status of data and processing
- Everything happens on top-of a *HDFS*

Use case of MapReduce?

- Mainly used for searching keywords in massive amount of data
- Google uses it for wordcount, adwords, pagerank, indexing data for Google Search, article clustering for Google News
- Yahoo: "web map" powering Search, spam detection for Mail
- Simple algorithms such as grep, text-indexing, reverse indexing
- Data mining domain
- Facebook uses it for data mining, ad optimization, spam detection
- Financial services use it for analytics
- Astronomy: Gaussian analysis for locating extra-terrestrial objects
- Most batch oriented non-interactive jobs analysis tasks

Summary

Flow of map reduce using an example of word count

Anagram problem using map reduce

Why map reduce

Use cases of map reduce