HW1.Inverted_Index and Retrieval Report

101062231 林展逸

1. How to run

總共有兩個 folder，一個是 InvertedIndex 一個是 Retrieval

compile:

sh ./InvertedIndex/compile.sh

sh ./ Retrieval/compile.sh

在執行前需要將 InvertedIndex.jar Retrieval.jar 及 inuut put 到 HDFS

另外在 HDFS 創造 output、tableOutput、scoreOutput 三個資料夾

run:

sh ./InvertedIndex/execute.sh

sh ./ Retrieval/ execute.sh

Retrieval 在執行時 query 範例格式

search “cat”

search “cat or dog”

search “cat and dog not bird”

2. Design

整個過程大致分成兩個部分

- 一個是先利用 InvertedIndex 針對 Input 創造 Inverted Index Table
- 再來利用 Retrival 讓使用者輸入 query，讀進 Table 找出使用者要找的 word，算出包含該 word 的 file 之 weight，在排序印出結果。

InvertedIndex 部分：

跑兩次 mapreduce，第一次計算 tf，第二次計算 df

第一次：

自定義<key,value> = <I2Key,I2Value>

I2Key 包含 file, word;

I2Value 包含 offset, termfrequency

InverteIndex/mapper

利用 pattern and matcher 將 input file 裡面的單字找出來，過濾掉其他符號，同時算出該單字的 offset

InverteIndex/Combiner
如果是同一個單字且同一個 file(Key 相同)則將 offset add 到 I2Value 的 offset 裡面
InverteIndex/Reducer
最後求出 I2Value 裡面的 offset ArrayList.size();即可求得該單字在該 file 的 termfrequency

第二次：
自定義 class wordInfo 包含 fileName, termfrequency, ArrayList<Long> offset;
自定義<key,value> = <TableKey,TableValue>
TableKey 包含 word;
TableValue 包含 offset, termfrequency, ArrayList<WordInfo> WordInfos;
InverteIndex2/mapper
將第一次的檔案讀回來這次以 word 為 key，以 TableValue 為 value
InverteIndex2/Combiner
類似第一次的做法，如果是同一個字(key 相同)將 WordInfo add 進 TableValue 裡
InverteIndex2/Reducer
最後求出 TableValue 裡面的 WordInfo ArrayList.size();即可求得該單字在該 file 的 doqfrequency.

Retrieval 部分：
跑一次 mapreduce，找出 query 中的單字資訊，並且計算每個 file 的 score
自定義 class TermFileInfo 包含 fileName, score, ArrayList<Long> offset;
自定義 value Scoreboard 包含 ArrayList<TermFileInfo> TermFileInfos;
Retrieval/mapper
從 conf param 取得使用者的單字，同時讀入前面的 tableOutput
過濾掉不要的字，只保留要的字的 information
Retrieval/mapper
計算出每個單字的 score，做成<key, value>=<word, scoreboard>

回到 main class 後，把針對 query 中的 or and not 把不符合要求的 file 去掉或是把不同單字卻重複 reference 到的 file score 相加起來形成最後的結果
在排序後將結果印出來。

3. Question

1.

(1).在整個過程中，Inverted Index 部分跑兩次 mapreduce，Retrieval 跑一次 mapreduc

(2).在 Retrieval 的地方應該可以再多跑一次 mapreduce，過濾出需要的 file 以及計算出他真正的 score 後排序印出

(3).

缺點：如果在跑一此 mapreduce 則會因為需要再讀寫一次 Disk 中的資料而導致 latency 更長。

優點：我這邊的做法是直接利用 java 的 hash 以及 treemap 來做，可能有點不符合作業規定。所以多跑一次 mapreduce 應該較好。

2.

(1).我多實做了 AND/NOT

(2).整個過程為困難的部分，我認為是讀寫檔案，因為需要做多次 Mapreduce 故有很多次檔案讀寫，但是要將前一次的檔案讀回來，再做成 <key,value>需要相當大量的 String parsing 來取得所需要的資訊。

另外為了有效率的算出自己要的資訊，所以這部分 key value 的 type 和資料結構就很花了很大量的時間設計。

3.我利用 Java 的 pattern matcher 來將不需要的符號去掉

matcher.find() 有下一個單字

matcher.start() 找到 offset

```
String  inputStr   = value.toString();
String  patternStr = "[A-Za-z]+";
Pattern pattern    = Pattern.compile(patternStr);
Matcher matcher    = pattern.matcher(inputStr);;
while (matcher.find()) {
    Long tmp = key.get() + (long)matcher.start();
    ArrayList<Long> arr = new ArrayList<Long>();
    arr.add(tmp);

    I2Value  offset    = new I2Value (arr, (Double)0.0);
    I2Key wordEntry    = new I2Key(String.valueOf(fileId), matcher.group());
    context.write(wordEntry, offset);

}
```