CloudProgramming HW2-PageRank Report

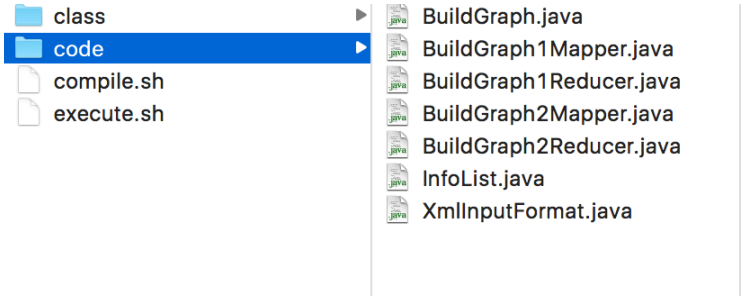
學號姓名：101062231 林展逸

Hadoop Version

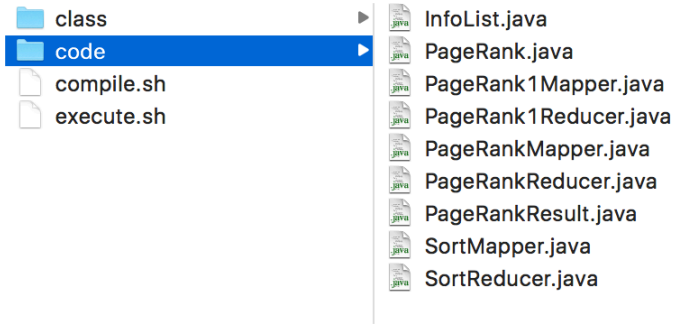
1. **File**

   共有兩個部分，第一部分是 BuildGraph，第二部分是 PageRank

   BuildGraph:



   PageRank:



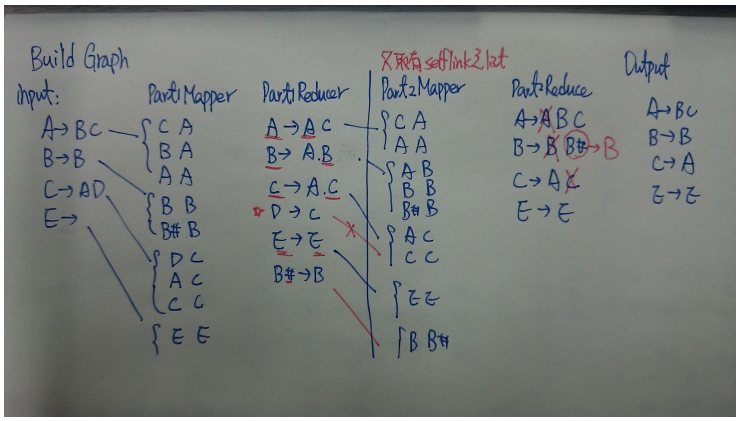
2. **Instruction**

   Compile: sh compile.sh

   Run: sh execute.sh

3. **Implementation**

   (1).首先是 Buildgraph 的部分



Mapper1 將<link,title>及<title,title>作為 key-value 送出去

如果有自己指向自己的 page 則做個記號變成<link+”#”,title> 避免等等被刪掉

Reducer1 收集起來如上圖

Mapper2 將 input 吃進來後只處理有 selflink 或是有做記號的，故 D->C 被捨棄

Reducer2 收集起來後將剛剛<title,title>去掉同時若有做記號的 link+”#”則還原為 link.

即可去掉不存在的 node

(2).再來是 PageRank 得部分

Mapper:

input 分別是 Page, PR, diff, links

output: [驚嘆號表示此 page 存在],[“|”如圖上的註解],[“#”為上一次的 PR]

```
/*
sample input:
-----------------------------------------
Page_A  1.0 0.0
Page_B  1.0 0.0 Page_A
Page_C  1.0 0.0 Page_A  Page_D

sample output:
-----------------------------------------
Page_A !
Page_A #1.0

Page_B !
Page_A Page_B 1.0 1 //Page_B link to Page_A
Page_B |Page_A
Page_B #1.0

Page_C !
Page_A Page_C 1.0 2
Page_D Page_C 1.0 2
Page_C |Page_A  Page_D
Page_C #1.0
```

Reducer:

```
/*
sample input:
-----------------------------------------
Page_A !
Page_A #1.0
Page_A Page_B 1.0 1 //Page_B link to Page_A
Page_A Page_C 1.0 2

Page_B !
Page_B #1.0
Page_B |Page_A


Page_C !
Page_C #1.0
Page_C |Page_A

Page_D Page_C 1.0 2

sample output:
-----------------------------------------
Page_A  1.425   X.X
Page_B  0.15    X.X Page_A
Page_C  0.15    X.X Page_A,Page_D
*/
```

```
for (Text value : values){
    pageWithRank = value.toString();

    if(pageWithRank.equals("!")) {
        isExistingWikiPage = true;
        continue;
    }

    if(pageWithRank.startsWith("|")){
        links = "\t"+pageWithRank.substring(1);
        continue;
    }

    if(pageWithRank.startsWith("#")){
        originPageRankString = pageWithRank.substring(1);
        originPageRank = Double.valueOf(originPageRankString);
        continue;
    }

    split = pageWithRank.split("\\t");

    double pageRank = Double.valueOf(split[1]);
    double countOutLinks = Double.valueOf(split[2]);

    sumShareOtherPageRanks += (pageRank/countOutLinks);
}

double newRank = ((1-damping)/titleNumber) + (damping * sumShareOtherPageRanks) + (damping * (danglingRankSum/titleNumber));
```

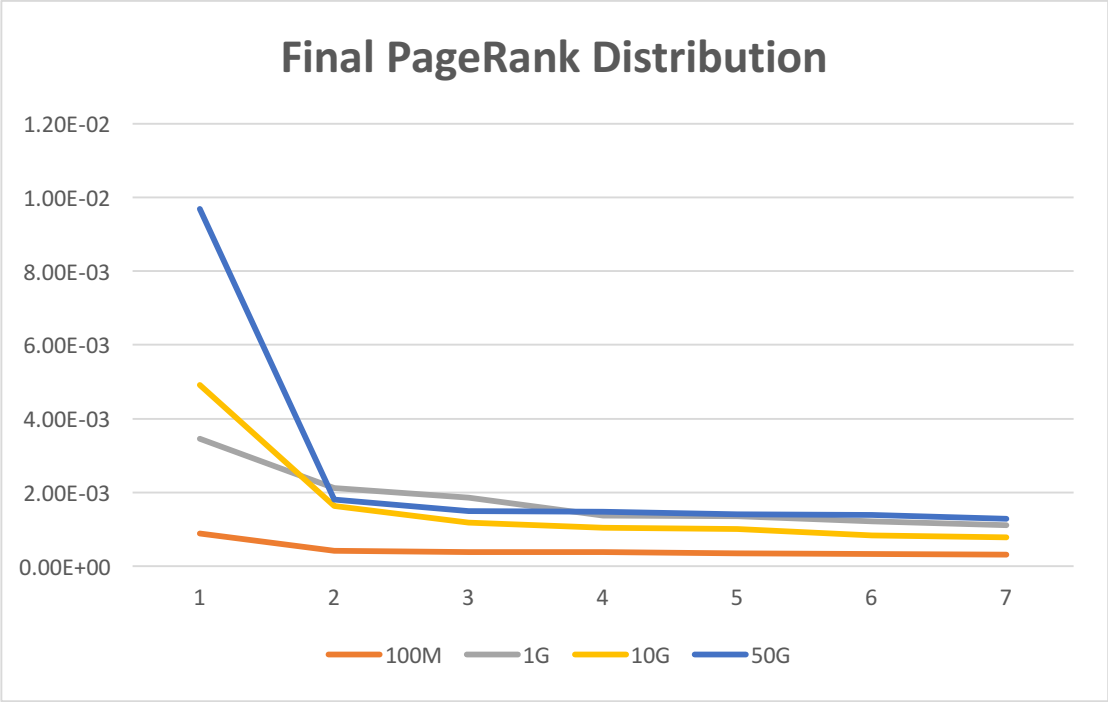
若為”|”則表示有 link 到此 page 把他的 PR/outLinkCount 算出來加起來

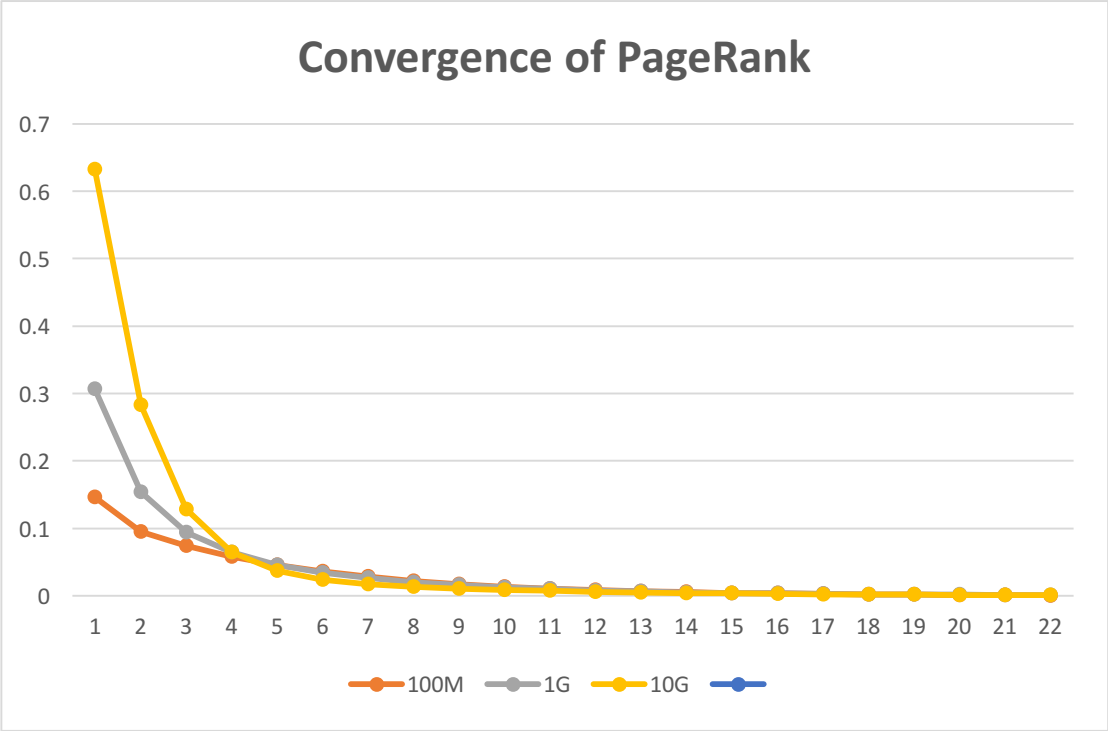
最後照 spec 公式即可算出 PageRank，同時算該 page 的 newPR-originalPR 當 error<0.001 即跳出 iteration

(3).

最後就是將 Page 依照 PR 排序印出

4. Experiment & Analysis





P.S Hadoop 版本在執行 50G 的資料量時會出現

java.lang.OutOfMemoryError: Java heap space 錯誤

應該是因為資料處理得不夠好導致中間就把 memory 用完

因此結果沒有 50G 的實驗數據

100M: job_1463345558261_1926

1G: job_1463345558261_2100

10G: job_1463345558261_2854

Apach Spark Version

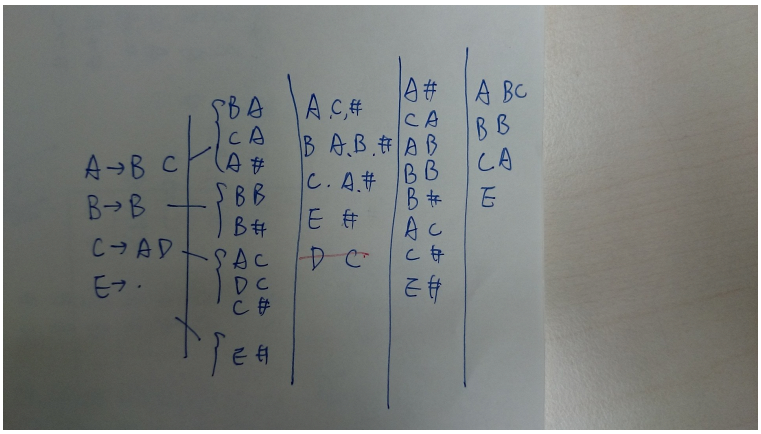
1. **Instruction**

   sbt pacjage

   spark-submit --class PageRankSpark --num-executors 30 target/scala-2.10/page-rank-spark_2.10-1.0.jar input(ex:hdfs:///shared/HW2/sample-in/input-100M)

2. **Implementation**

   (1).首先是 parse 的部分



   因為 sacla 語言的關係所以從前面的作法做了小小的改進

   變成一樣<link,title>,<title,"#">作為<key,value>然後 reduce 時刪掉沒有"#"記號的那個，接下來再反過來<title,link><title,"#">//此為了避免因為 dangling 在 map 時因為沒有 outlink 而沒有考慮進來所以還是先多加了一個冗余的 outlink 最後 reduce 後把冗余的"#"filter 掉

   再來是 PageRank 的部分

   先做出一個 RDD[title, (PageRank, Array[outLinks])]

   計算 danglingRankSum：

   從 RDD filter 出 Array.length == 0 的項然後 map(pagerank).reduce(_+_)

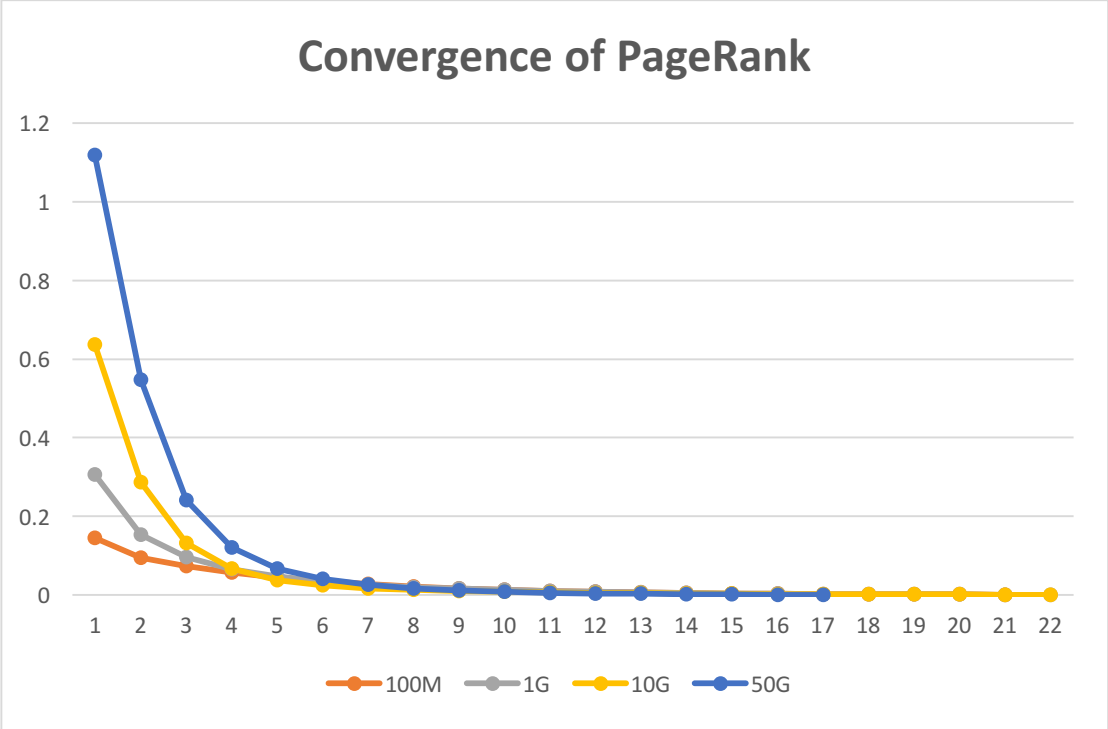
   計算 PageRank:

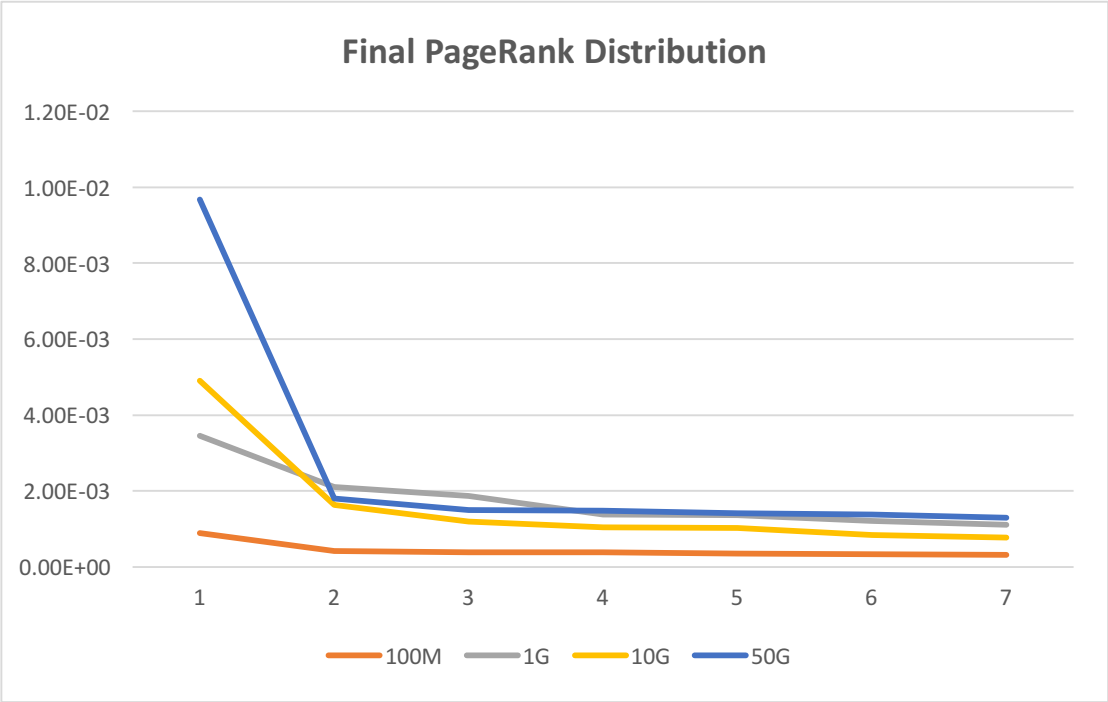
   從 RDD 裡面取出每個 page 的 outlink 然後 map 成<outlink,XXX>，XXX 為指向他的 title 該 pageRank/linkCount

   在多加<title,YYY>，YYY 為公式中另外兩項，如此一來 reduce 加總時就可以把分數算出來

```
//計算PageRank
var tmpPageRankList = pageRankList.map(list => {
  list._2._2.map { outLinks => (outLinks, damping * list._2._1 / list._2._2.length) }
            .+:(list._1, (1 - damping)/totalNum + damping * danglingRankSum)
}).flatMap(x => x).reduceByKey(_ + _);
```

3. Experiment & Analysis





100M: application_1463345558261_5193

1G: application_1463345558261_4412

10G: pplication_1463345558261_4430

50G: application_1463345558261_4509

Compare Two Implementation

在寫 code 方面，因為 spark 並沒有像 hadoop 那麼死必須寫 mapper＆reducer class，同樣的工作在 hadoop 可能要分開寫好幾個 file 並且定義好中間的 key,value 型態。但是在 spark 中可以直接一行完成。
另外 Hadoop 算 PageRank 每一次的 iteration 都需要將結果寫回 HDFS 導致浪費許多時間在讀檔寫檔，但是 spark 採用 Transformation 的方法使算過得可以 keep 在記憶體得效率更高。所以無論在寫程式的過程還是執行的過程 Spark 都是某種程度上優異於 hadoop

Experience & Conclusion

首先是實驗結果的部分，關於 Page Rank Distribution 可以發現在 100M 的 input 時，前 10 名的 Page Rank 差距都差不多，不過當 data 越大例如到 10G、50G 時，前 10 名的 Page Rank 差距就很大。我認為原因是，當 data 少時，不太容易出現很多 page 都 link 到的 node。但是當 data 變大時就容易出現一個 page 被很多 node link 到也因此該 page 的 Page Rank 會特別大。
另外是 convergence，可以從結果發現不管資料量多大，幾乎都可以在 21,22 次 runs 後將 error 縮小到 0.001。