Introduction to Massive Data Analysis

Homework 2 Report

Student ID: 105062635
Name: 吳浩寧

1. INTRODUCTION

這次的作業要使用 MapReduce 來實作 Google 的 PageRank，假設整個網路的架構是個有向圖，每個網頁是一個 vertex，連向其他網頁的超連結則是 edge，要得知每個網頁的重要性，可以想像越常被導向，或被越重要的網頁連向的網頁，得到的 PageRank 分數應該越高。因此，我們假設所有網頁的分數總合恆為 1，初始分數都相等，使用 Adjacency Lists 來表示彼此連接的關係，每個相鄰的 vertex 可以平均地分得來源網頁的分數。但當遇到有 Dead End 或 Spider Trap 的情況，會導致分數匯集到這些網頁上送不出去，最後所有的分數都歸零，因此必須有較小的機率可以 teleport 到任意網頁，並在每輪做完後進行 normalize 以避免此情況。

因此最終計算公式如下：

$$r_j'^{new} = \sum_{i \to j} \beta \frac{r_i}{d_i} + (1-\beta)\frac{1}{N} \qquad r_j^{new} = r_j'^{new} + \frac{1-\sum r_j'^{new}}{N}$$

2. IMPLEMENTATION

可以指定的相關參數如下：

NODES	總結點數	**BETA**	公式中的 β
TOPN	分數最高的前幾名	**MAXROUND**	最多執行幾輪
PRECISION	結果精確度(小數點後幾位，輸入 0 使用 double 原本精確度)		

1.) Job 1

Map Function:

接收兩個 input 檔案，分別為 directed graph，和前一輪產生所有網頁的 rank。如果資料來自 rank 檔案，則送出(id, score)，為了和來自 graph 的資料區分，將 score 前面加上一個字元 R 以供辨識；若資料來自 graph 檔案，每讀一行除了送出(id, neighbor id)外，還必須送出(neighbor id, “TO”)，因為我們 page id 並非從 0 開始往下編號，若只用 FromNodeId 作為 key，後面的程序無法得知所有 vertex 有使用到的 id。

input		output	
(graph file)	(virtual rank file)	1 2	3 TO
1 2	1 0.2	1 3	4 TO
1 3	2 0.2	2 4	1 TO
2 4	3 0.2	3 1	5 TO
3 1	4 0.2	3 4	1 R0.2
3 4	5 0.2	3 5	2 R0.2
3 5		5 1	3 R0.2
5 1		5 4	4 R0.2
5 4		2 TO	5 R0.2

Reduce Function:

將每個 page、舊的 rank 與所有鄰點串接在一起。若是第 1 輪進入此 reducer，由於尚無舊的 rank 檔案，因此須透過 context.getConfiguration() 取得目前進行了幾輪運算，並在第 1 輪使用初始值 1/N。

output
1 0.2 2 3
2 0.2 4
3 0.2 1 4 5
4 0.2
5 0.2 1 4

2.) Job 2

Map Function:

除了計算 value $\beta\frac{r_i}{d_i}$與$(1-\beta)\frac{1}{N}$並送出(id, value)，還必須送出(“SUM”, value) 以供 reducer 計算所有 page rank 的總和$\sum r_j^{new}$；(“SUM”, id)則將所有有使用到的 page id 傳給 reducer，並在 id 前加上字元 N 以供辨識。

output	
2 0.08	4 0.053
SUM 0.08	SUM 0.053
3 0.08	5 0.053
SUM 0.08	SUM 0.053
SUM N1	SUM N3
4 0.16	SUM N4
SUM 0.16	1 0.08

SUM N2	SUM 0.08
1 0.053	4 0.08
SUM 0.053	SUM 0.08

Reduce Function:

收集所有(id, value)以計算$r_j'^{new} = \sum_{i \to j} \beta \frac{r_i}{d_i} + (1-\beta)\frac{1}{N}$，收集所有("SUM", value)以計算$\sum r_j'^{new}$，並將$\sum r_j'^{new}$與所有("SUM", id)中的 page ids 串接在一起。

output
1 0.173
2 0.12
3 0.12
4 0.333
5 0.093
SUM 0.84 4 2 5 1 3

3.) Job 3

Map Function:

在此計算 value $\frac{1-\sum r_j'^{new}}{N}$，同上一階段計算好的 $r_j'^{new}$送出(id, value)。

output	
1 0.173	4 0.032
2 0.12	2 0.032
3 0.12	5 0.032
4 0.333	1 0.032
5 0.093	3 0.032

Reduce Function:

計算最終結果$r_j^{new} = r_j'^{new} + \frac{1-\sum r_j'^{new}}{N}$，並根據精確度將去除多餘的位數。

output
1 0.205
2 0.152
3 0.152
4 0.365
5 0.125

4.) main()

在主程式裡除了循環執行 job 1~3，為了達到判斷結果是否收斂，每次做完 job 1，就必須將結果使用 hdfs.rename()搬到其他位置，直到 job 3 執行完後，利用 hdfs.getFileChecksum()比較新舊檔案的 checksum，來決定是否停止程式。

計算完最終結果後會將 Top N 的 pages 輸出，我直接讀取一遍最終輸出的檔案，每讀完一行便將該行加到 priority queue，並 override compare function，以該行的 rank 作為排序依據，若新增元素後 queue size 超過設定值即將最小的那行剔除。

3. RESULT

經測試後發現，作業範例中 5 個 vertices 的 input 在精確度到小數點後第 3 位時，實際在第 7 輪就會收斂了。

針對 10876 個 vertices 的 input p2p-Gnutella04.txt，我測試了不同精確度設定下的收斂情況，結果如下：

PRECISION	7	10
LAST ROUND	13	20
CONVERGENCE	yes	yes
RESULT	1056 6.321E-4 1054 6.292E-4 1536 5.242E-4 171 5.118E-4 453 4.958E-4 407 4.848E-4 263 4.796E-4 4664 4.708E-4 261 4.63E-4 410 4.614E-4	1056 6.321432E-4 1054 6.291889E-4 1536 5.24102E-4 171 5.117887E-4 453 4.957661E-4 407 4.848811E-4 263 4.796437E-4 4664 4.706709E-4 261 4.629469E-4 410 4.614139E-4

PRECISION	0 (double)
LAST ROUND	20
CONVERGE	no
RESULT	1056 6.32143145666554E-4 1054 6.291888494620049E-4 1536 5.2410207124257E-4 171 5.117886671691846E-4 453 4.957660193584136E-4

	407	4.8488104972304745E-4
	263	4.7964371946164095E-4
	4664	4.70670862000855E-4
	261	4.6294685525522823E-4
	410	4.614139376722708E-4

PRECISION	0 (double)	
LAST ROUND	36	
CONVERGE	yes	
RESULT	1056	6.3214314566671l6E-4
	1054	6.291888494619177E-4
	1536	5.241020712425744E-4
	171	5.117886671691387E-4
	453	4.95766019358337E-4
	407	4.8488104972317E-4
	263	4.796437194617451E-4
	4664	4.706708620008177E-4
	261	4.629468552551962E-4
	410	4.614139376721918E-4

可以發現其實在精確度到小數點後 7 位，即可以正確地找到前幾名的結果，而且在第 13 輪就會收斂，速度快很多；若使用 double 原本的精確度，則須進行 36 輪的計算。

4. EXPERIENCE

這次遇到比較大的問題包括以下幾點：嘗試在每次 MapReduce 同時處理多件工作，以減少 jobs 數量；更了解如何使用 Hadoop 相關的 FileSystem API 以利在程式中針對檔案進行讀寫、搬移、比較等工作；如何找到 Top N 的 pages；如何在 main function 與 Mapper、Reducer 間傳送參數；另外檔案輸入格式並不如一開始所想：將每個 page 從 0 依序編號下來，而是不連續、起始點也不一定，因此為了讓 job 知道所有的 page ids 也花了些功夫。

不過根據我這次的寫法，在 job 2 的時候須將所有的 page ids 串接起來，因此記憶體須可容納此大小的資料量，最後判斷檔案相等的部分若有多個 reducers 會產生多個 output file，這次也沒有處理此情況，希望以後有時間可以再做改進。