2017 PP Hw5

105062570 林子瑜

Instruction

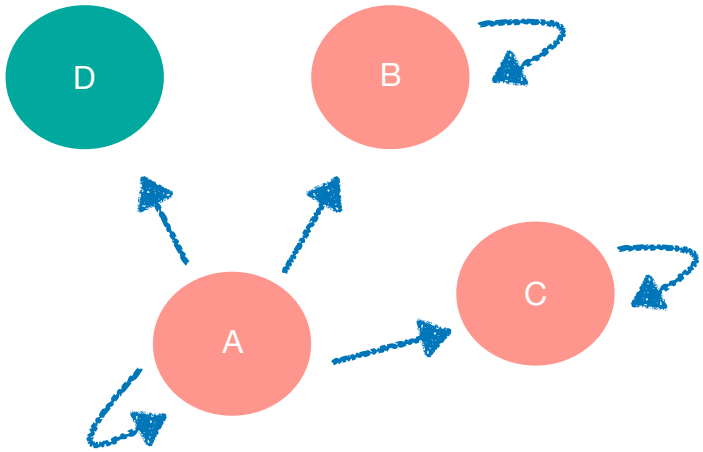
1. Compile：已寫好 Makeflie，直接下 make 就好
2. Run：$ sh run.sh $input_file $output_dir [$iteration] [$num_reducer] [$dataset_size]

Implementation

整支程式被分成 6 個 job：

1. Parse
    - Mapper

        將原始 xml 檔案 parse 出 title 以及對應到的 links，並且為了要將 missing link 拔掉，這邊將除去 missing link 的問題 model 成 graph 的問題。為了要知道哪些 link 是不存在的，把 title 視為 parent node，而指出去的 links 視為 child node，因此這邊做的事情就是由 parent node 傳送 msg 給 child，並且為了後續能夠證明自己存在，也要傳送訊息給自己。如下圖所示。



Graph

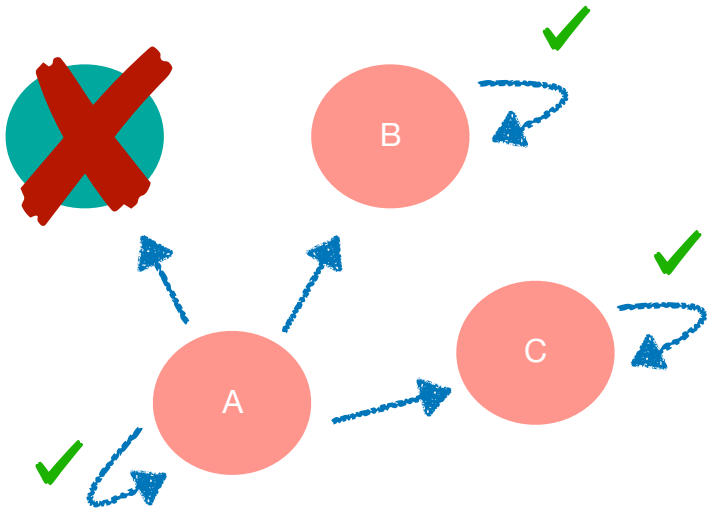
[A：B, C, D]

[B：]

[C：]

    - Reducer

        reduce phase 最主要的工作是篩掉 missing link，在收到所有訊息後每個 node 會將自己的 parent 收集好存成 Text 格式，並且在寫進檔案前先檢查是否有收到剛剛在 map 階段自己傳給自己的訊息，若無則代表自己不存在是 missing link，那就不寫進檔案，反之則寫入檔案。



Graph

[A：B, C, D]

[B：]

[C：]

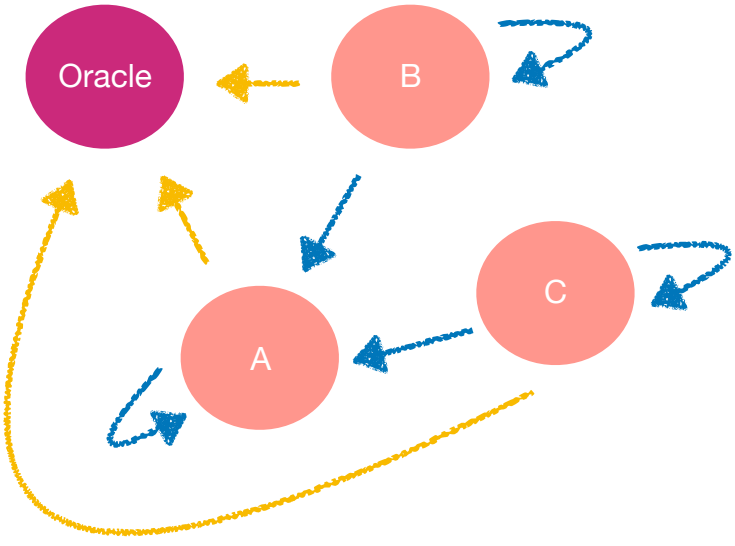
- Combiner

純粹優化，讓 mapper 寫入暫存檔的數量變少。若是看到自己傳給自己的，就直接 pass 到 reducer，若是看到 parent link 則先行收集存成 Text 再傳給 reducer。

2. Build graph

- Mapper

child 回傳 ack 給 parent 以讓 parent node 在 reduce 階段能夠得知 real link 有哪些。並且自己傳送訊息給自己以讓 dangling node 能夠保留住。最後則是每個人都要傳訊息給一個 Oracle node，讓其在 reduce 階段時可以獲取所有人的訊息。



Graph

[A：B, C, D]

[B：]

[C：]

- Reducer

reduce 階段首先會初始化每個人的 PR 值 = 1 / N，N 的獲取是由在 Parse 階段時讀取了 hadoop 內建計算 input record 的 counter，有幾個 page N 就是多少。接著 parent

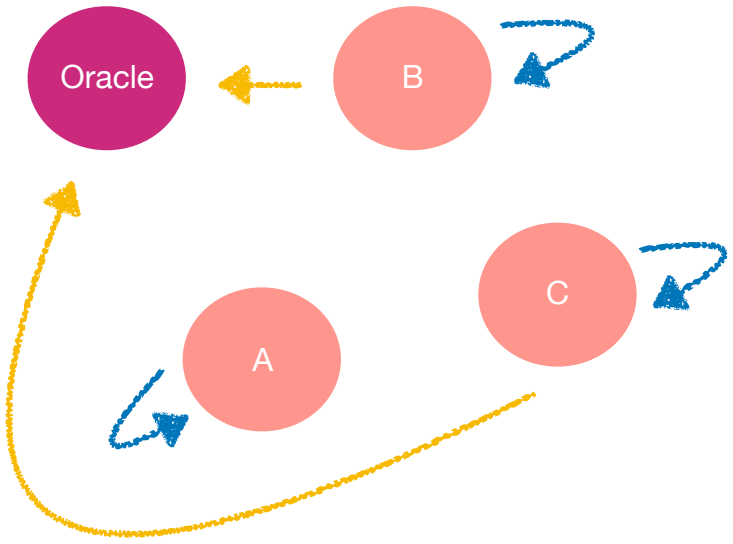
node 將 child 回傳的訊息收集起來，組成自己的 outgoing link。而 Oracle node 也是把所有人的訊息收集成自己的 link 存起來。

- Combiner

  也是純優化用，若是看到自己傳給自己的就直接 pass 過去，而若是看到 child，就先行做 reduce，減少寫入暫存檔的 pair。

3. Collect
   - Mapper

     在 Collect 階段做的事情是收集 dangling node 的 PR 並傳給 Oracle node，這是由於只有 Oracle node 知道所有別人的訊息，可以藉著此點事先算好該 iteration 所有 dangling node 的總和，之後再由 Oracle node 傳送給所有其他人。因此在 map 階段時，會先判斷若是 dangling node 就傳送自己的 PR 給 Oracle node，並且將自身訊息傳給自己以保留整張 graph 的資訊。



Graph

[A：B, C, D]

[B：]

[C：]

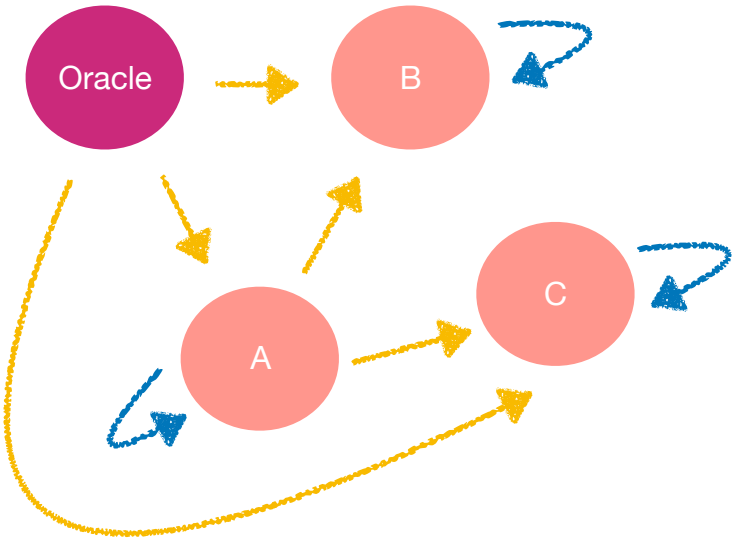
   - Reducer

     reduce 階段就是將傳給 Oracle node 的 PR 值相加起來，而其他為了 graph 資訊的訊息就原封不動地寫入檔案裡。

4. Rank
   - Mapper

     Rank 階段則是執行真正的 update PR 的工作。在 map 時每個人會將自己的 PR 值萃取出來，將其除以 #outgoing link，最後傳送訊息給所有 outgoing link。那這邊值得一提

的是 Oracle node 也會執行這個動作，而他的 PR 就是上一個 Collect 階段所收集的所有 dangling node 的 PR，而其 outgoing link 是所有人，因此 #outgoing link = N。除此外，還會將自己的資訊傳給自己以保留整張 graph 的資訊。



Graph

[A：B, C, D]

[B：]

[C：]

- Reducer

  reduce 階段就是各自把收到的 PR 加起來，套上 PR 的公式得到新的 PR，更新過後把新 PR 及原本 graph 資訊寫到檔案裡。另外，在這邊每個人在更新 PR 前會先計算改變的差值，並將其傳給特殊的 Error node，這是為了等等計算 error 用的。

- Combiner

  純優化用

5. Judge

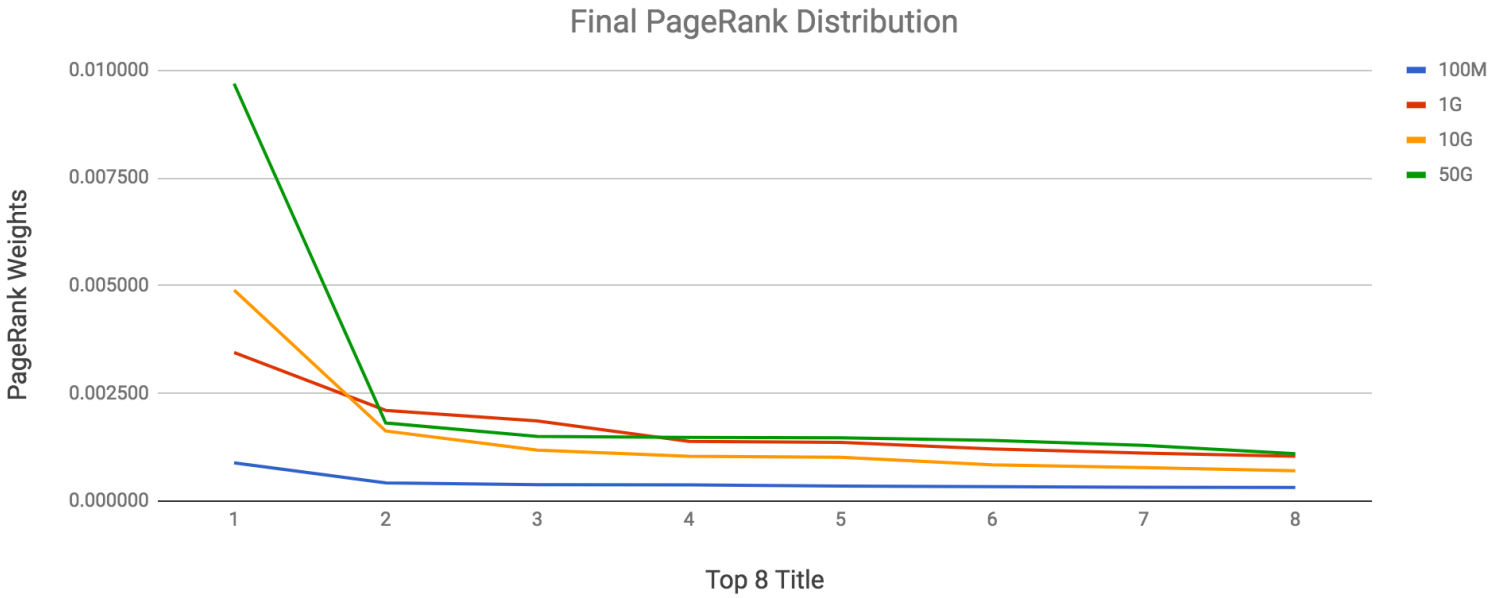
   將上一個階段產生的 Error node 的值做 reduce 以得到 total error，並利用 counter 做紀錄，若是 error < 0.001 就結束程式。

6. Sort

   實作一個 Node class 並 implements WritableComparable，其 sort 策略是先比較 PR 值再比較 title。而 Sort 階段的工作就是在 map phase 將 PR 和 title 分離出來，用 Node class 存成 key 值，接著就是在 reduce 階段 write 到檔案裡。

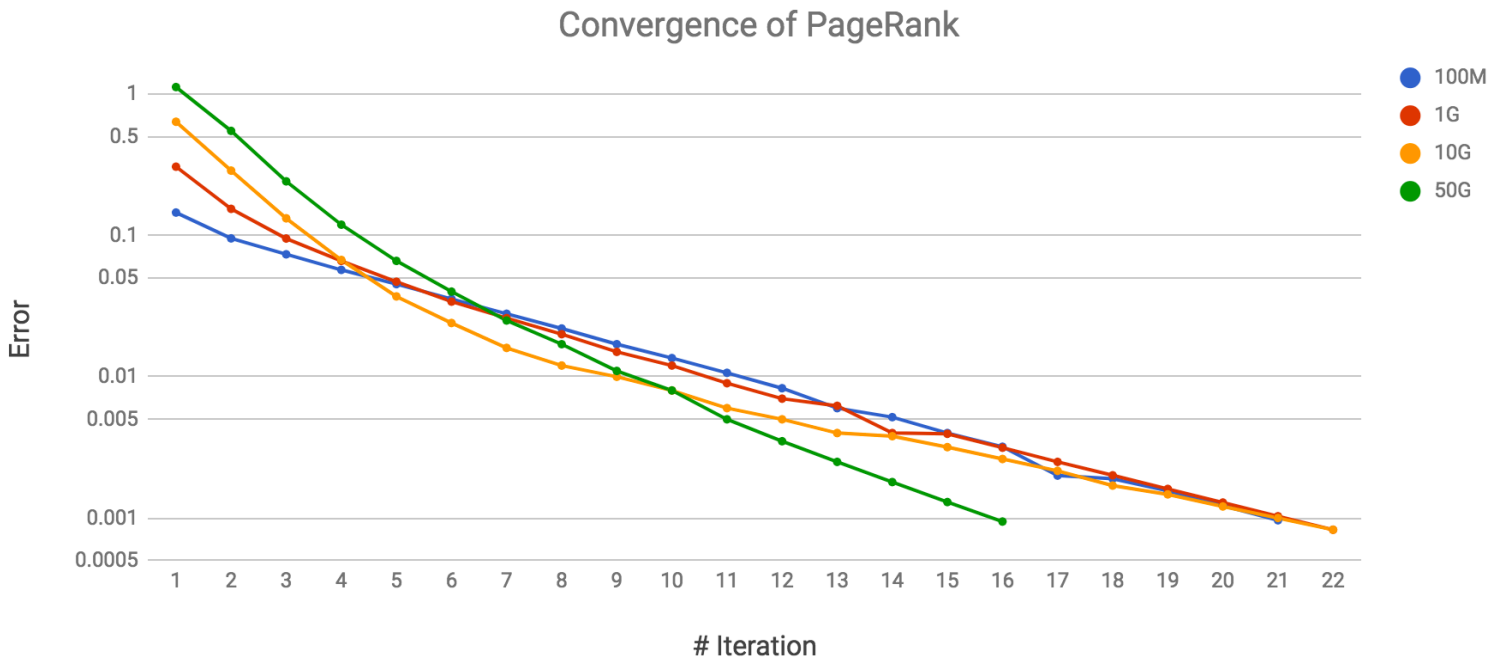
Experiment

1. PageRank weight distribution

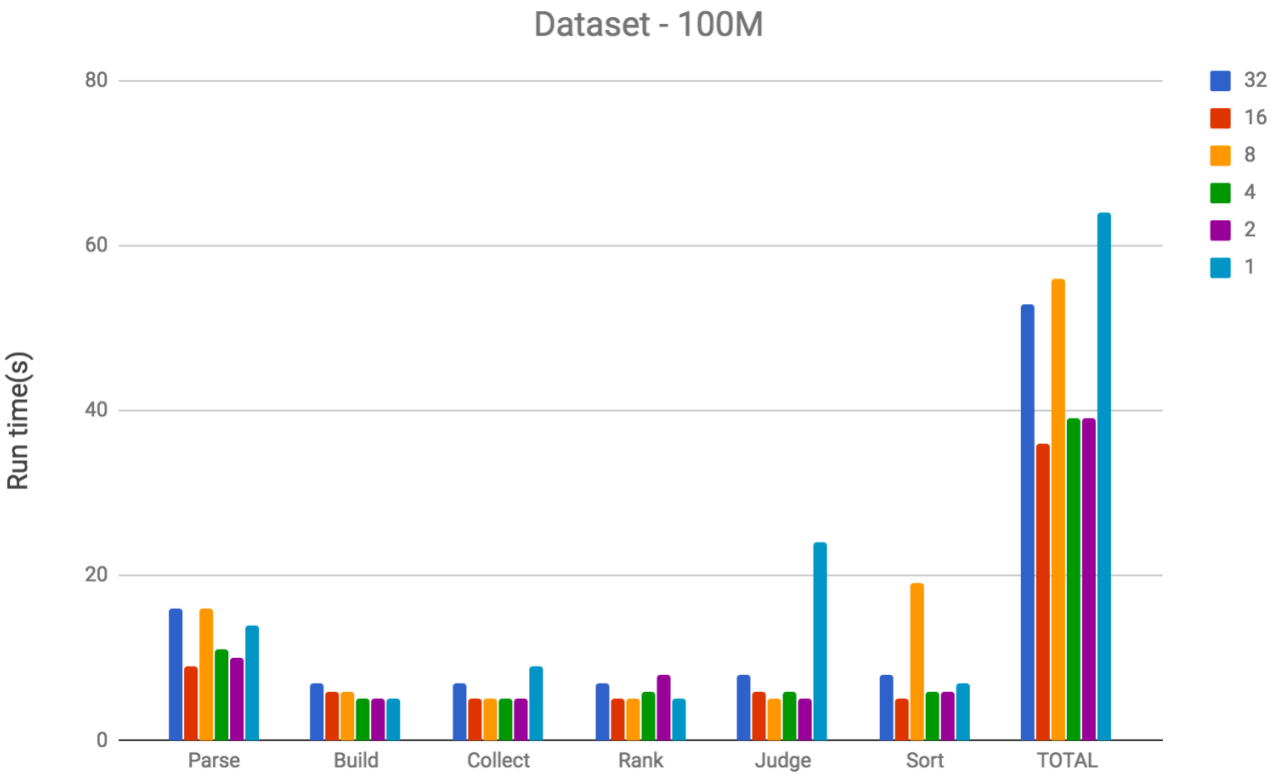


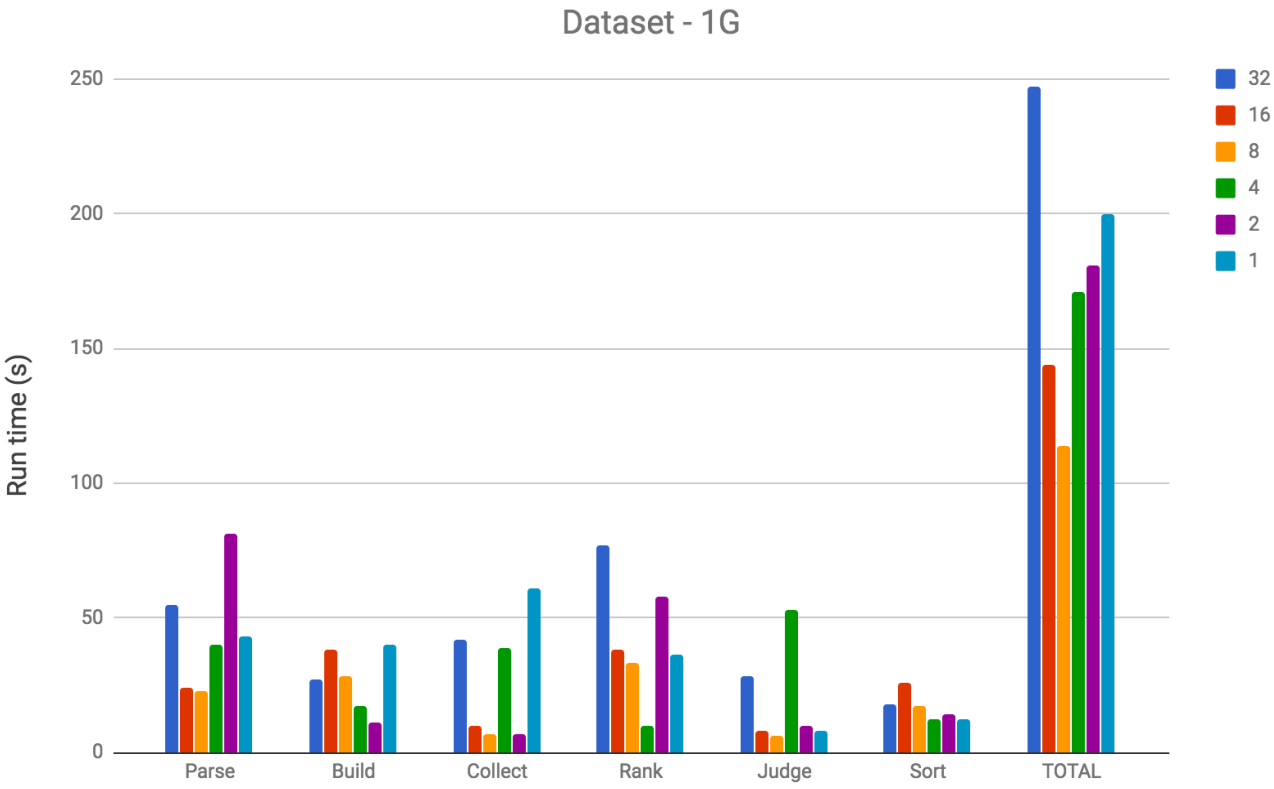
	1	2	3	4	5	6	7	8
100M	0.000889	0.000422	0.000382	0.000379	0.000351	0.000338	0.000323	0.000319
1G	0.003447	0.002107	0.001862	0.001386	0.001363	0.001212	0.001113	0.001042
10G	0.004898	0.001627	0.001186	0.001042	0.001016	0.000843	0.000777	0.000705
50G	0.009684	0.001814	0.001500	0.001478	0.001471	0.001410	0.001295	0.001100

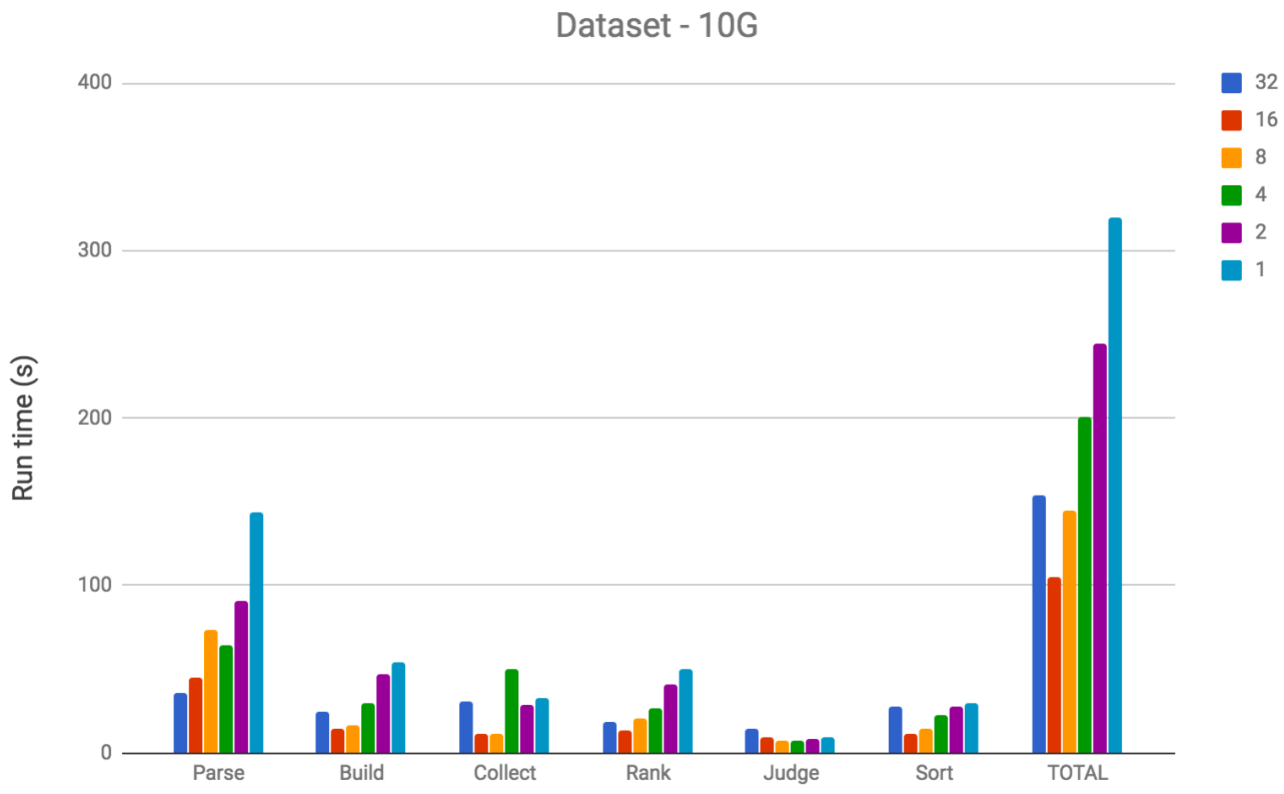
2. Converge rate

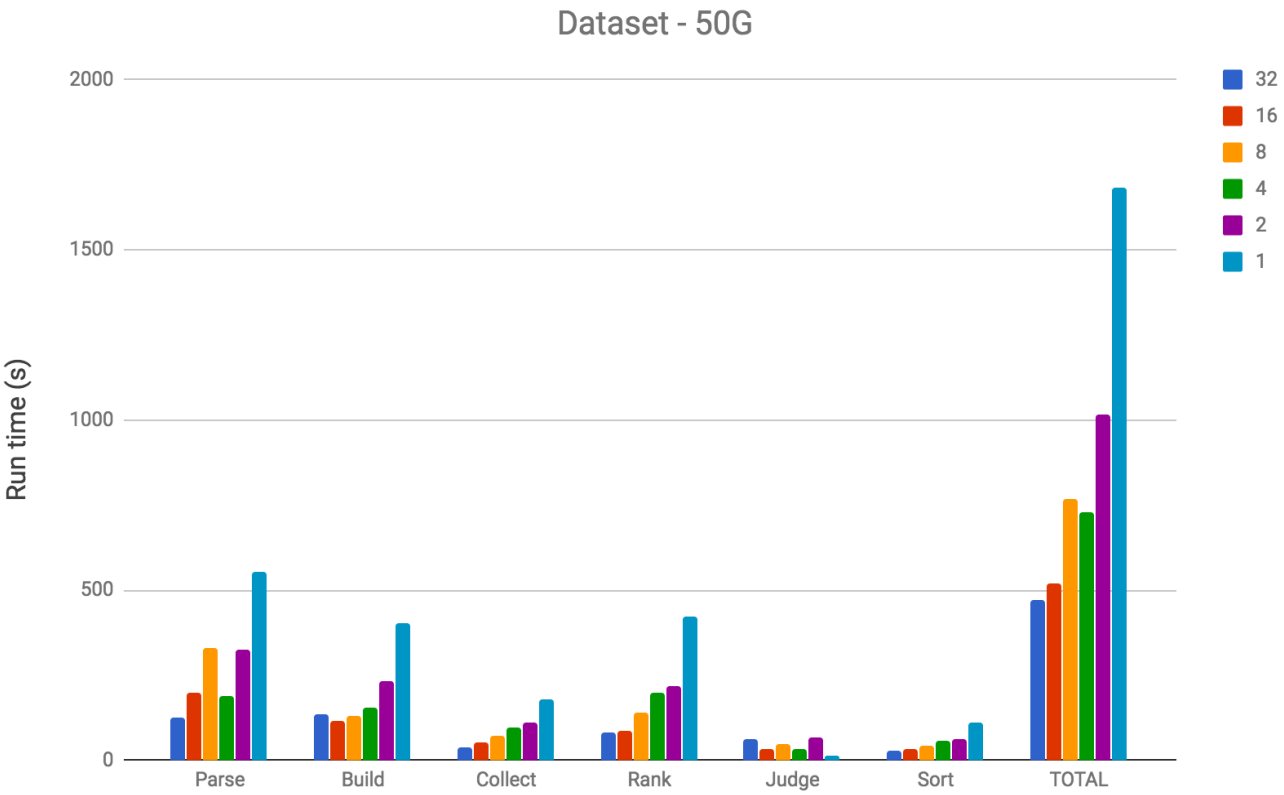


3. Different number of Reducers









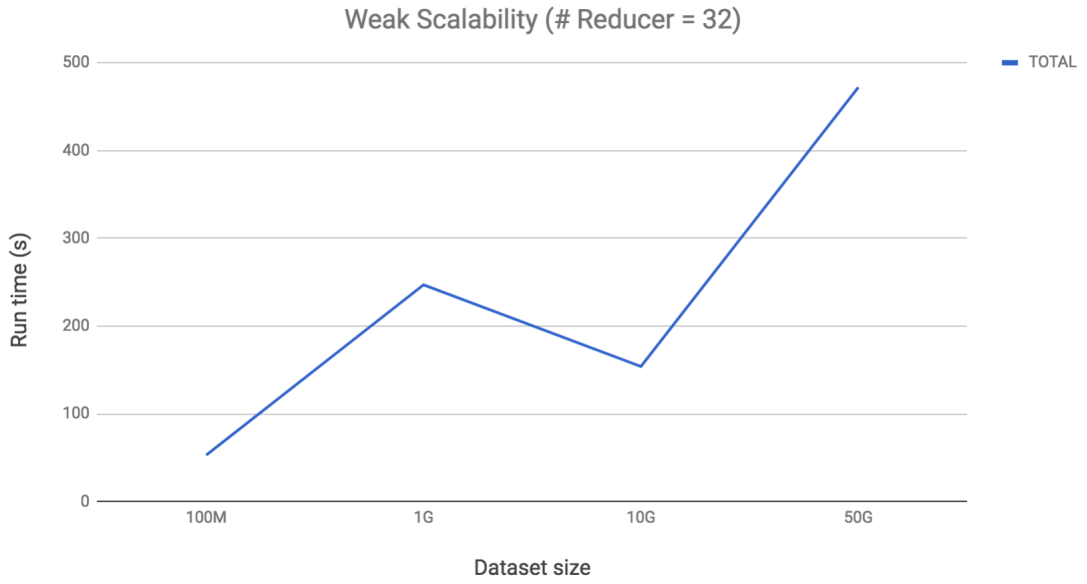
這邊的實驗設計是只跑一個 iteration，記錄每個階段的執行時間，而最後的 TOTAL 是把每個階段的執行時間做總和。可以由上圖歸納出以下幾點觀察：

① Dataset 的大小會強烈影響到不同 Reducer 數目影響效能的情形，這邊推測是因為當資料量不夠大時，其他變因像是 cluster 的穩定度之類的細微差異都會影響到實驗結果。像是在 100M 的實驗中幾乎看不出什麼規律。

② 除了 100M 的資料外，其餘的實驗能夠大致看出當 Reducer 增加到一定的數量後會達到最佳的 performance，若超過這個 threshold 則會導致效能下降，那這點觀察也很合理，因為若是資料量不夠大卻硬是增加 Recucer 數目的話可能會因為 I/O cost 的增加而導致效能下降。

③ 只觀察 10G, 50G 的實驗結果會發現 Judge 階段的執行時間都很一致，這是因為在實作上 Judge 的 Reducer 永遠只會有一個，此結果也符合預期。

4. Weak scalability



那可以很明顯地看出當 Dataset 數量上升，執行時間的趨勢大致隨之上升，然而在 10G 時抖了一下，那推測原因是因為在 1G vs 10G 時變因有兩個：增加 Reducer 數量帶來好壞處、problem size 增加，而抖一下應該就是兩者帶來效果相互抵消後的差異。

Conclusion

這次作業要求我們使用 hadoop MapReduce 實作 pagerank，那其實我覺得本次作業最難的地方是在 preprocessing input 的部分，像是處理 missing value ，重點在於要將這樣的問題想成是 graph 的問題才比較能夠找到解法，因為一開始一直糾結在要怎麼存一個超大 hashmap 才能夠知道哪些 link 不存在，那這樣的出發點一開始就錯了一定會跑超慢或是根本做不到，但後來有想到把問題 model 成 graph 的問題後就迎刃而解。

另外覺得其實這樣一直 iteration 的問題利用 map/reduce 來說好像有點笨，或許使用 Spark 會比較適合。

最後則是在這樣的 framework 下 programming，對於該架構本身的流程一定要非常清楚，不然會超容易遇到 bug 不知道該怎麼解決，那這點我認為是老師上課比較缺乏的部分，像是在 java 中的 mapper 究竟在 call 了 map function 之後做了什麼，reuse 了哪些 variable 之類的，我個人是認為挺重要的啦，或許可以不用細講但應該至少要稍微提一下會比較好。

Application ID

只提供第一個 iteration，因為全部貼上太多了@@

1. 100M

Phase	**Application ID**
Parse	application_1513147375415_12714
Build graph	application_1513147375415_12733
Collect	application_1513147375415_12734
Rank	application_1513147375415_12735
Judge	application_1513147375415_12737

2. 1G - 32

Phase	**Application ID**
Parse	application_1513147375415_12967
Build graph	application_1513147375415_12968
Collect	application_1513147375415_12970
Rank	application_1513147375415_12972
Judge	application_1513147375415_12974

3. 10G

Phase	**Application ID**
Parse	application_1513147375415_13295
Build graph	application_1513147375415_13297
Collect	application_1513147375415_13300
Rank	application_1513147375415_13303
Judge	application_1513147375415_13305

4. 50G

Phase	Application ID
Parse	application_1513147375415_16735
Build graph	application_1513147375415_16754
Collect	application_1513147375415_16767
Rank	application_1513147375415_16781
Judge	application_1513147375415_16799

5. 1G - 32

Phase	Application ID
Parse	application_1513147375415_17535
Build graph	application_1513147375415_17544
Collect	application_1513147375415_17552
Rank	application_1513147375415_17560
Judge	application_1513147375415_17569
Sort	application_1513147375415_17573

6. 1G - 16

Phase	Application ID
Parse	application_1513147375415_17587
Build graph	application_1513147375415_17595
Collect	application_1513147375415_17603
Rank	application_1513147375415_17608
Judge	application_1513147375415_17617
Sort	application_1513147375415_17620

7. 1G - 8

Phase	Application ID
Parse	application_1513147375415_17662
Build graph	application_1513147375415_17666
Collect	application_1513147375415_17673
Rank	application_1513147375415_17679
Judge	application_1513147375415_17684
Sort	application_1513147375415_17688

8. 1G - 4

Phase	Application ID
Parse	application_1513147375415_17698
Build graph	application_1513147375415_17703
Collect	application_1513147375415_17708
Rank	application_1513147375415_17714
Judge	application_1513147375415_17719
Sort	application_1513147375415_17722

9. 1G - 2

Phase	Application ID
Parse	application_1513147375415_17741
Build graph	application_1513147375415_17752
Collect	application_1513147375415_17756
Rank	application_1513147375415_17760
Judge	application_1513147375415_17768
Sort	application_1513147375415_17772

10.1G - 1

Phase	Application ID
Parse	application_1513147375415_17784
Build graph	application_1513147375415_17789
Collect	application_1513147375415_17793
Rank	application_1513147375415_17800
Judge	application_1513147375415_17804
Sort	application_1513147375415_17807