Parallel Programming
Homework 3: MapReduce

學號：111065510 黃韋慈

I. Implementation

1. List the highlights of your implementation

i. 讓在同一個 node 的所有 thread 寫入本地檔案

在 MapReduce 的過程中，在讓每個 thread 去進行 map 的處理後需要把文件資料寫入本地的檔案，這部分在一開始我是使用一般 c++的寫入方法，每當 thread 進行完畢後就會將結果寫在 output 的路徑中額外增加的檔案裡，然而在實作過程中發現這樣有些麻煩，會造成檔案過於繁多，因此想到在 hw2 中使用 pthread lock 去動態分配資料的方法，在這邊也是採取相同的方法，由於所有 thread 都是共用一塊記憶體，因此當有 thread 需要去進行寫入時，就會先去檢查記憶體是否有該筆單字的資料，並透過 pthread lock 的機制去進行紀錄並確保資料不會出現問題．

ii. 動態分配 thread 所需要處理的資料

由於每個 node 所拿到的資料不一，thread 處理的資料數也會不一，為了讓負載平衡，在使用 thread 去進行計算前會先針對 node 所分配的所有單字數去進行數量的分配，並將每個 thread 需要處理的資料打包放進 array 裡，這樣 thread 在計算時只需要去讀取相應的位置即可，此外這邊的分配方法也如前面作業相同，都是在

將所有資料平均分配後，將多出來的資料由第一個依序增加一個，以保障每個 thread 處理的資料能差不多．

iii. 讓其他 node 能接收未知數量的資料

這點在一開始十分苦惱，因為每個 node 會處理數目不一的資料，而在 MPI_Recv 卻又需要明確定義出會接收多少筆，以往資料數都是讓各個 node 去根據他們自身的 rank 搭配計算，而由於現在資料的讀取及計算都是由 jobTracker 去進行，因此在一開始我透過 MPI_Recv、MPI_Send 在資料真正的傳送前會先去傳送資料的數目，然而後來在搜尋 MPI 的過程，有發現他們有特別提供相應的解決方法就是透過 MPI_Get_Count、MPI_Probe 的方式先去接收資料數，再進行真正的資料接收，如此傳送方就只需要傳送一次即可．

iv. 在不同 node 間傳送字串

由於在一開始採用的方式是由 jobTracker 進行資料的讀取及分配，並非由各自 node 自行去讀取，因此就需要將需處理的資料去進行分配，就需要去傳輸單字，然而 MPI 不支援 string，之前的作業也是使用傳送數字為主，但因爲 MPI 收取傳送都需要有 buffer 去做，因此這邊嘗試用 char array 的方式去做，而由於因為過去很少使用 char 相關型態因此在設定大小時有小卡住．

2. Detail your implementation of the whole MapReduce program

i. 首先在定義完相關參數後，就會開始區分 jobTracker、TaskTracker 去做．其中 rank=0 是 jobTracker，其餘都是 TaskTracker，因此會有 size-1 個 TaskTracker．

ii. jobTracker 會先去讀取 loc 檔案，並將其中的 chunk id 和 node id 以 map 的方式進行存放．

iii. jobTracker 接著會去讀取 input 檔案，並且根據一開始輸入的 chunk size 去將行數做組合．

iv. 接著 TaskTracker 會主動向 jobTracker 以 MPI_Send 的方式傳送自己的 id 以告知 jobTracker 可以進行分派工作．

v. jobTracker 在收到 TaskTracker 傳送的 id 後就會先去檢查說這個 TaskTracker 是否還有 locality，如果還有就會發送相應的資料．如果沒有，則代表此 TaskTracker 可以去幫其他 TaskTracker 處理，而在處理之前會先去檢查這個 TaskTracker 是否有 sleep 過，若沒有則會先 sleep，若有 jobTracker 就會發送其他資料給他進行處理．這邊 jobTracker 都有用 struct 去進行記錄各個 task 以及 TaskTracker 的狀態，包括是否還有 locality，還有哪一個 task 尚未被處理等等的內容．

vi. TaskTracker 在收到要處理的資料以後，會根據參數開啟 pthread，然後進行資料數的處理，將資料平均的分配進 array 以便後續讓

pthread 進行處理．

vii. 在 TaskTracker 的 pthread 進行處理的過程中，會將資料存入該 node 的記憶體中，並透過 pthread lock 方式確保資料的正確．

viii. 在 thread 都處理完裡就會 join，而該 node 就會對 thread 處理完畢的資料進行 reduce，合併相同文字的數量，並進行 sort．

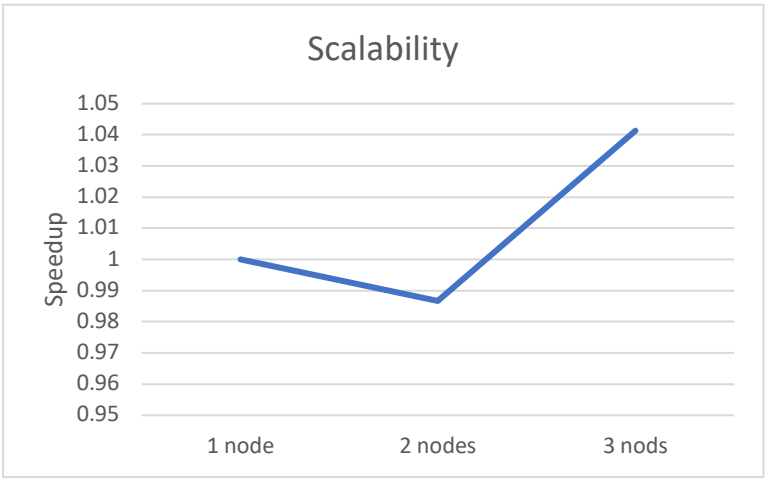
ix. TaskTracker 再回傳通知 jobTracker 已處理完畢，並發起新的需求給 jobTracker．

x. 而當 jobTracker 發現所有的 task 都被處理過後就會停止指派任務．

xi. 最後 TaskTracker 會將 sort 好的資料傳送給 rank=1 的 TaskTracker，讓他對所有資料再進行加總並排序，最後寫到 output 的檔案中．

3. Conduct the experiments to show the performance impact of data locality, and the scalability

以下實驗使用 testcase6 進行實驗．由於 TaskTracker 是 size-1，而最多只能開到 4 個，因此 TaskTracker 最多只會有 3 個 node．



在有兩個 node 時會稍微有下降的趨勢，猜測有可能是因為其中有 sleep 的情況所造成的原因，而 3 node 會呈現上升就如我原本預期能夠得到更快的結果。

II. Experiments and conclusion

1. What have you learned from this homework?

這次的作業我覺得主要是更了解 c++的資料型態，以往都是使用 array 為主，但因為這樣會很麻煩，因為沒有 key-value 的方式在其中，會額外增加許多 loop 去做遍歷。而在每個 TaskTracker 的 thread 做完存放到 local 檔案中這部分，一開始還是會像以前繼定的概念一般想說就開個檔案來存即可，但回想到前幾次的作業，包括更了解 thread、memory 這些概念後發現其實有更簡單的方式可以進行，這些知識其實都對我自身的思考模式有蠻大的改變，讓我在做的過程中能更好的從 CPU、memory 的角度去思考，都讓我獲益匪淺。

2. Feedback

這次的作業其實比起 hw3 簡易了許多，雖然乍看覺得十分複雜，但只要釐清了一步一步試驗就覺得還好。

這學期以來學到了很多平行化的程式和概念，讓我更能從較為底層的東西去思考，而非僅僅只是在算法上進行優化．同時，也因為這堂課讓我從原本沒寫過 c++到現在已經可以做出想達成的事，實在是獲益良多！也很感謝助教與老師不論是在課堂或是 lab 的盡心教導！真的十分感謝！雖然很累，但很慶幸能修習這堂課！