1. List the highlights of your implementation

➢ Map phase

- **Scheduler**

首先從 locality configuration file 中讀出 chunkID、nodeID 並依序 push 至個別的 vector 中，每次收到 worker 的 request 後將 map task chunkID 傳到 worker 手上，而因為要記錄 Scheduler 所擁有的 map task 中有哪些已經分派出去，因此我用一個一維陣列(allocation marker)做表示，分配出去的 chunk 紀錄為 1，而 0 是紀錄尚未被分派的 chunk。而到最後，若 scheduler 已經沒有 chunk 需要被處裡，則傳送 termination condition (-1) 給每個 worker。當 scheduler 將 termination condition 全部傳送給每個 worker 完畢時，scheduler schedule map task 的工作即結束，而 worker 收到 termination condition 後，也隨著結束 map phase 的工作。

- **Worker**

收到 chunkID 後開始對該 chunk 執行：

Input split function：

從檔案中讀取須處理的 chunk 的資料，並將一個 chunk 拆解成一個個 line(record)再傳送給 map function 做處理。

Map function：

將每個 line 產生出 key - value pairs，實作方式為開出兩個 vector，

一個負責裝 key，一個裝 value。

Partition function：

將 key 餵給 partition funciton，回傳的值為 reducerID，在新開一個 vector 將 reducerID 存進去。

下圖為 partition function，可依據不同目的、需求修改 return 值。

```
int partition_function(string word, int num_reducer){
  return (word.length() % num_reducer);
}
```

這樣一來，就可以知道 key-value pair 和每個 key 會對應到哪個 reducerID。最後，再將其存入 intermediate file 中，同個 line(record)但不同 reducerID 的 key-value pair 會放到不同的 intermediate file 中，其格式為 chunkID + line no + reducerID.txt。

➢ Reduce phase：

- **Scheduler**

Shuffle：

Scheduler 透過 map phase 產生出來的 intermediate file 做 shuffle，將相同 reducerID 的 file 整合在一起，再將其寫入新的檔案中，格式為 reducerID.txt。

因此，每個 file 就可形成一個 reducer task 給 reducer 做處理。

Send reducer task：

做法和 Send map task 的做法相似，但因為 Send reducer task 沒有 locality aware 的問題，因此如果 worker 發出 request，就將 reducerID 由小的先開始傳。而結束 schedule reducer task 的做法和結束 schedule map task 相同。

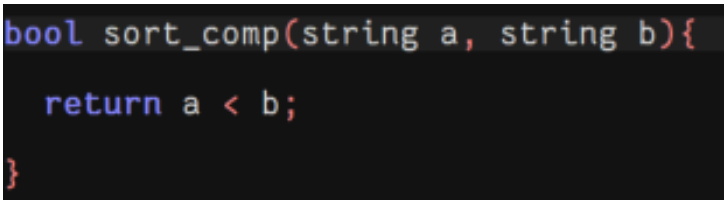
- **Worker**

收到 scheduler send 的 reducer task 後讀取該 file 並依序執行下列 function：

Sort function：

將每個讀取出來的 key-value pair 進行 sorting，透過調整 sorting comparator 來決定要如何排序。

下圖為 sort comparator，可依照自己需求(EX：升冪、降冪排列)做調整

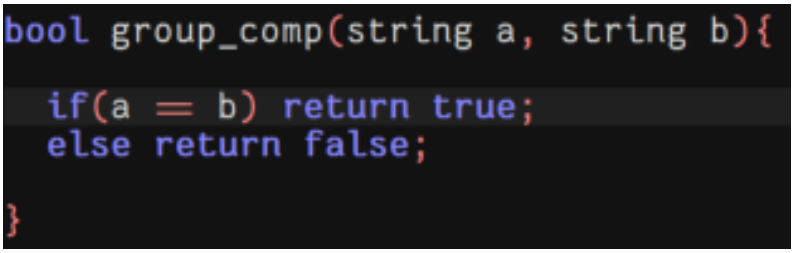


Group function：

在對排序好的 key-value pair 進行 grouping，實作上是掃描每個 key 並對 key 兩兩比較判斷是否要 group 在一起，並開出一個一維 grouping marker array 紀錄每個 key 是屬於哪一個 group。至於要如何判斷要不要進行 grouping，由 group comparator 決定，其 return 值為 true or

false。

下圖為 group comparator，使用者可以依照偏好的 grouping 方式進行修改。



Reduce function：

透過掃描 grouping marker array 將同個 group 的 value 全部加起來。

Output function：

會後將 final key-value pair 存入檔案中，格式為 job name + reduce task ID.out。

2. Detail your implementation of the whole MapReduce program.

整個程式就像是一個流水線，每個步驟，每個 function 基本上是各自獨立的，但是讓人最頭痛的部分就是同步化問題，感覺只要牽扯到同步化的問題就是會陷入 debug 的泥沼裡。當初在設計如何 schedule task 時就覺得這部分跟 HW2 很像，都是要 dynamic 的進行 task 的分配，但不同之處是這次的作業要考量到 Data locality-aware 得部分。

data locality-aware：因為 chunkID、nodeID 是依序被 push 在個別的 vector 中，因此越前面的位置越新，所以當 work 送出 request 時，scheduler 會從 vector 最前端開始找，希望能找出該 worker 應該負責的 chunk，若沒有，則找出 vector 最前面且非該 worker 應該處理的 chunk 傳送給 worker，因此會進行兩次的掃瞄。而傳送時會傳入兩個資料，一是 chunkID，告訴 worker 需要處理的 chunk 編號，另一是該 chunk 原本要分配的 nodeID，透過此 nodeID，可以與收到此 chunkID 的 worker 做比較，若 ID 不相同，則呼叫 sleep()來模擬傳輸時間。

```
if(recvTask[1] != workerID){

  cout<<"worker:"<< workerID <<" ready to sleep"<<endl;
  sleep(delay);
  cout<<"worker:"<< workerID <<" finish sleep"<<endl;

}
```

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

data locality：

Test data	#nodes	#cpus	Delay	Chunk size	字數
04.word	4	8	0.01 second	10	25180

透過修改 data locality configuration file 來突顯 Strength of data locality，將 delay 設小希望能在很短的執行時間內也能看出時間上的差異。

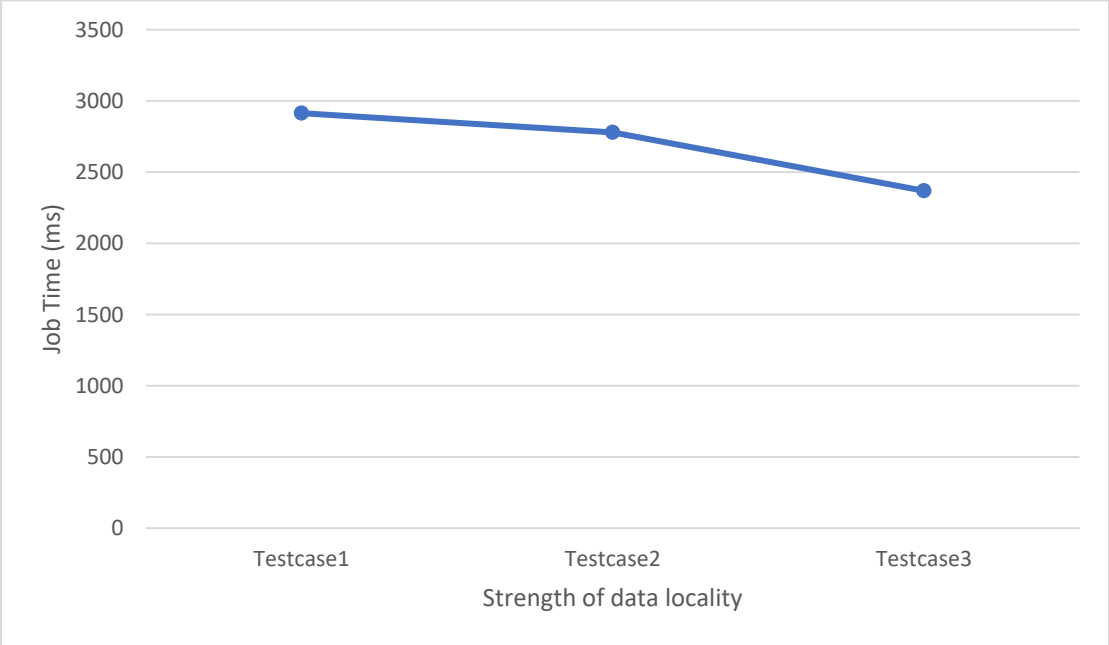
實驗資料為下列三種 configuration file：

Testcase1：分配 80% chunckID 給 node0

Testcase2：分配 50% chunckID 給 node0

Testcase3：chunkID 三個 node 平分

理想情況下，Testcase1 中 node1、node2 相較其他 case 更容易被分配到 node0 該負責的 chunkID，因此會呼叫 sleep 較多次而拖慢執行時間。



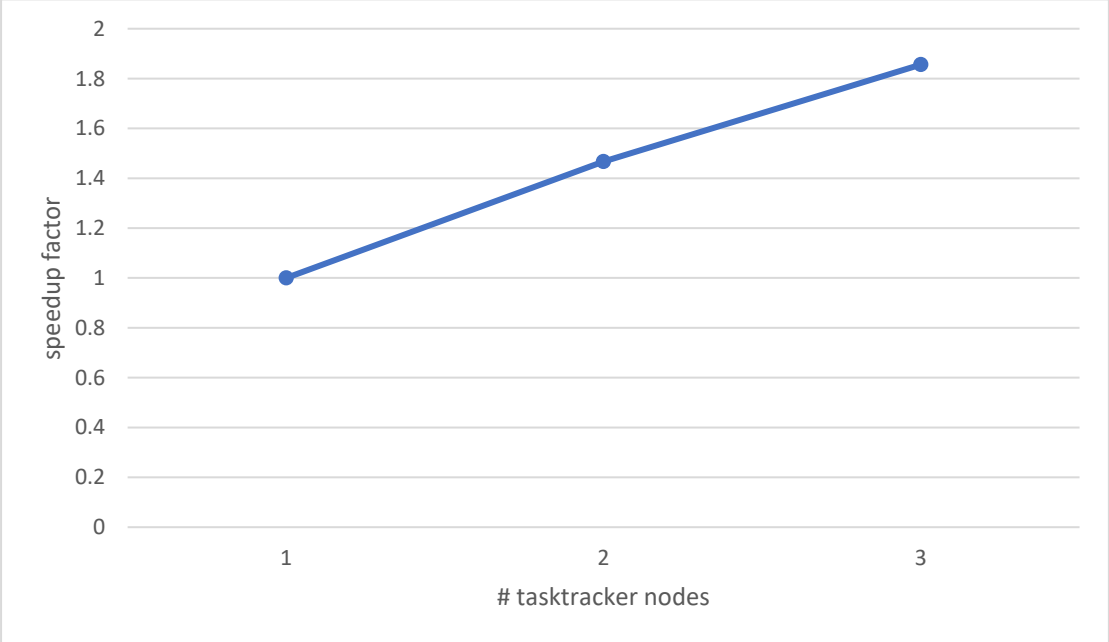
透過對上圖的觀察明顯可以看出若 chunkID 的 locality 越是分佈的均勻，越能減少 sleep 所花費的時間進而縮減整個程式執行的時間。否則若分配的不平均，則其他 node 則需要先呼叫 sleep 才能對 chunk 進行處理，時間會被拖慢。若是在分散式系統中則會因為傳輸時間變多造成效能下降，因此可見 data

locality 的重要性。

Scalability：

Test data	#nodes	#cpus	Delay	Chunk size	字數
04.word	4	8	0 second	10	25180

其他參數不變，將 delay 設為 0 秒(也就是假設每個 node 處理 chunk 不需要額外花 transfer time)。目的是希望看到固定的資料下透過 node 增加是否能夠提升 speedup factor。



由圖中可知，再沒有額外 transfer time(sleep)的情況下，透過增加 node 數量可以更加平行地執行分派下來的 task，使執行時間降低。

4. Experience & conclusion

- What have you learned from this homework ?

這次的作業沒有需要運動到什麼複雜的技術，但是以作業需實作的流程、功能來說我覺得相當複雜，光是作業的 spec 就花了我大半時間才把他搞懂，而在實作過程中遇到了很多檔案的操作，不熟練的我遇到了很多的困難，還有許多資料的搬移、同步化的處理，需要花點心思才解決的出來，但也因為這次的作業讓我更了解 map reduce 是如何運作的，流程是什麼、每個步驟要幹嘛，都透過這次的作業有進一步的了解，雖然這次是用 cluster 進行分散是系統的模擬，但透過 sleep 來模擬資料的傳輸，在運作過程中也能體會到 map reduce 跑在分散式系統上的感覺。