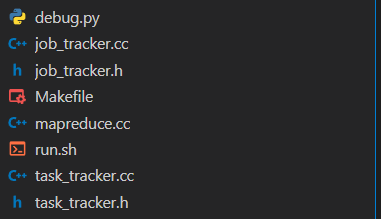
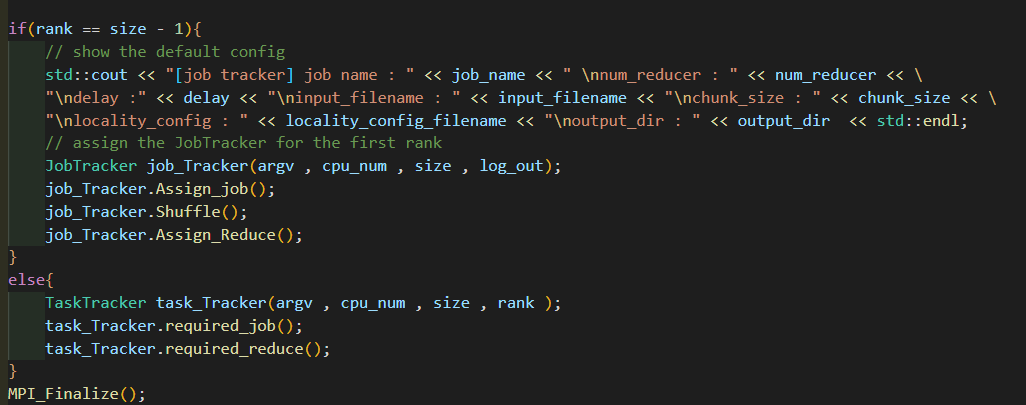
|  |  |  |  |
| --- | --- | --- | --- |
| Hw3 report | | | |
| 學號: 109062233 | 姓名: 蘇裕恆 |  | 繳交時間: |

**Implementation :**

****

我總共創造了幾個file，分別為job\_tracker , mapreduce 與 tasktracker，在比較容易debug的形況下比較符合真實在會出現的，而非只有一個單一的file。



在map reduce裡面，我將最後一個node設為 job tracker，而剩餘的node都為tasktracker。

我們可以從spec裡面發現。

“The jobtracker is responsible for generating the map tasks, reducing tasks of a

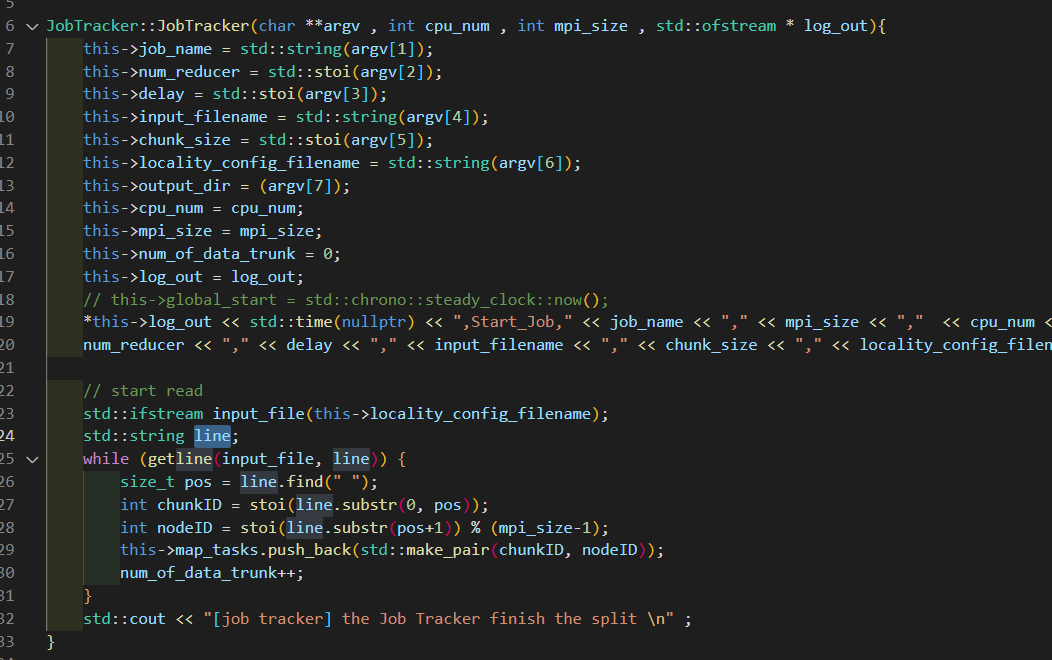
MapReduce job and following the data-locality scheduling principle to dispatch tasks

on worker nodes for execution.”

因此，我在jobTracker裡面總共有三個階段，AssignJob ( 代表著assign map task) , Shuffle ( 進行hash跟並將他結果存在home directory ) 和 Assign\_Reduce ( 代表著assign reduce task )。

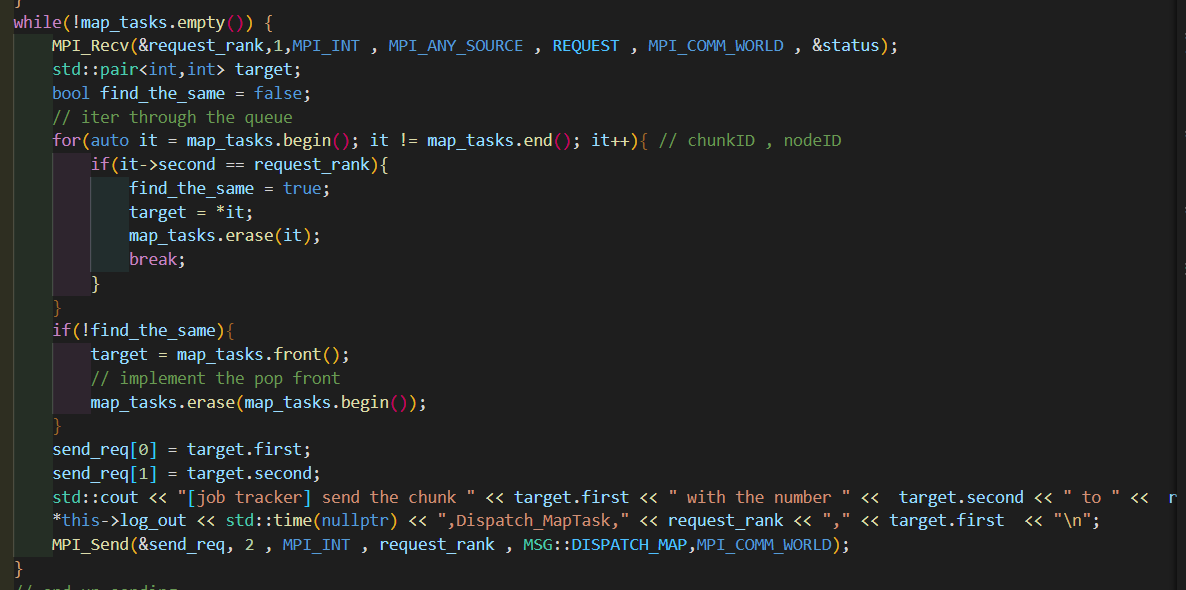
同時，taskTracker 會一直先required map job ( require job) 與 required reduce ( 尋找reduce job )。

我們先進入job tracker



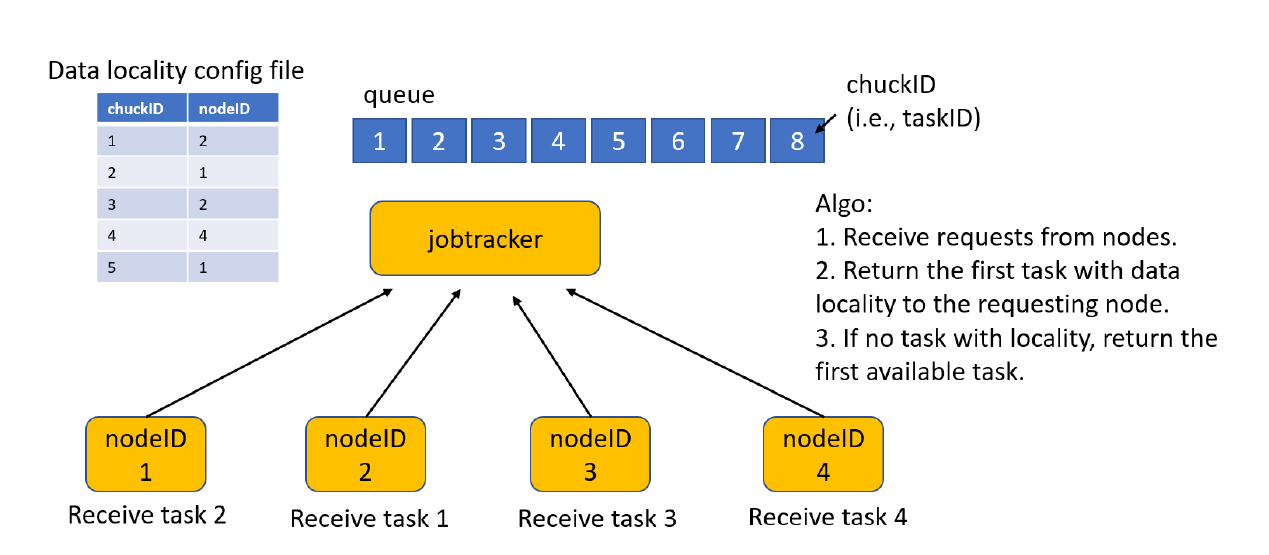
我們可以看到，除了一般的initialize以外，我們會直接從input line readline近來，並將他用空格做分開 因為(If the nodeID is larger than the number of worker nodes, mod the nodeID by the number of worker nodes.) 所以我們會將他mode (size – 1 ) 這也是為甚麼我將 job tracker設在node為size – 1得原因，最後將他統一放入map\_tasks裡面 ( 代表總共有幾個map tasks要做 ) 在同時記錄有幾個data trunk。

JobT tracker::Assign\_job()

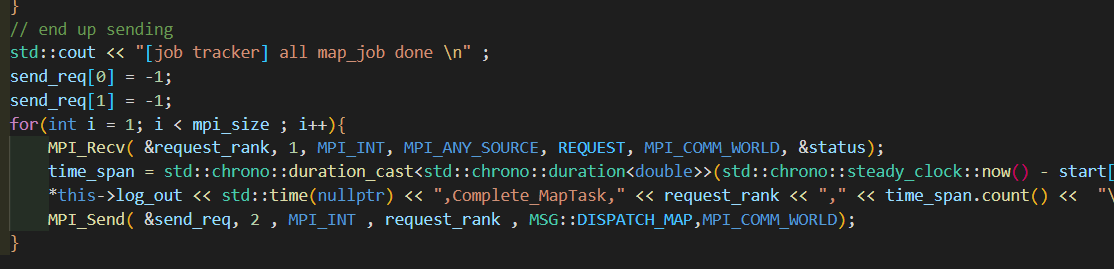


基本上，map task跟reduce task的模式很接近，我都是使用thread pool的方式來實作。

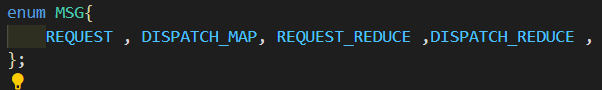
在job tracker裡面，根據spec，他會一直接收到request。因此，我們會需要做mpi\_recv。而這邊需要進行的是將nodeID相同的在一起，我這邊是用 std::vector<std::pair<int, int>> 來實作，因為vector的data type比較適合做中間index的pop。



如果找不到一樣的，就從一開始pop出一個item來send出去，給request rank ( node ) 。

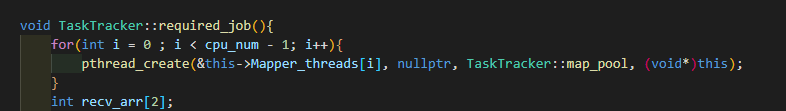


同時，如果今天data 都已經分配出去並且處理完成，那就是將 -1 送出給request node。

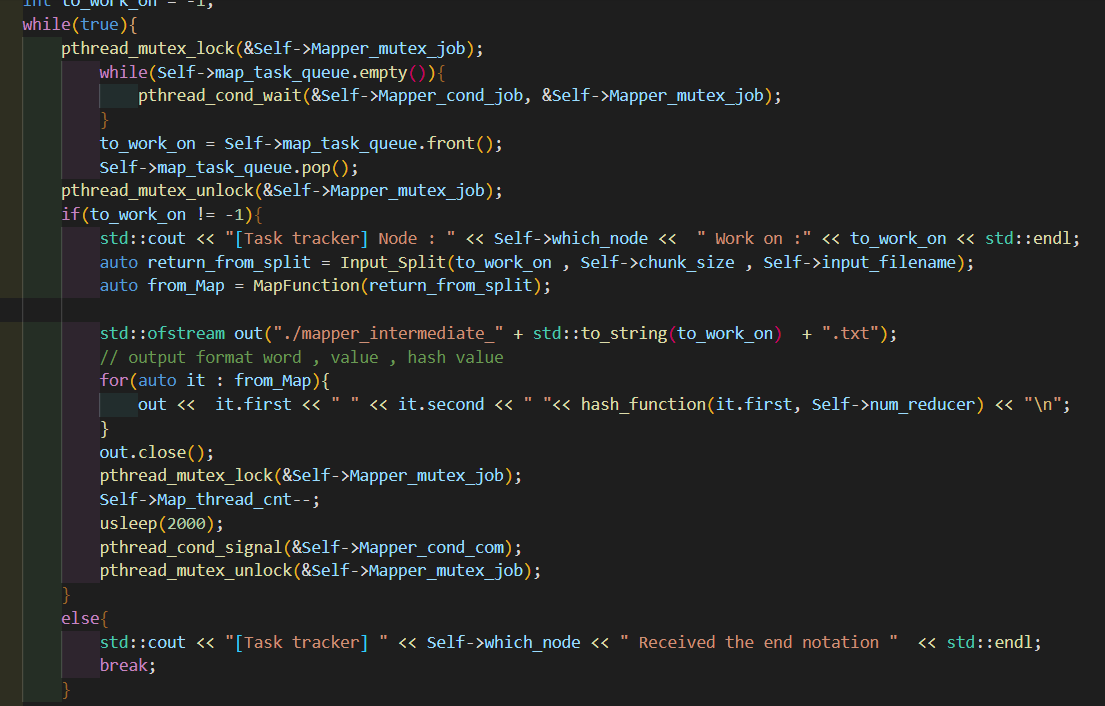


在實作上，我使用了enum來做為哪中type的send。在美觀的同時也保證了程式的可讀性。

將到他怎麼assign job ，就必須提及task tracker是如何send request的。

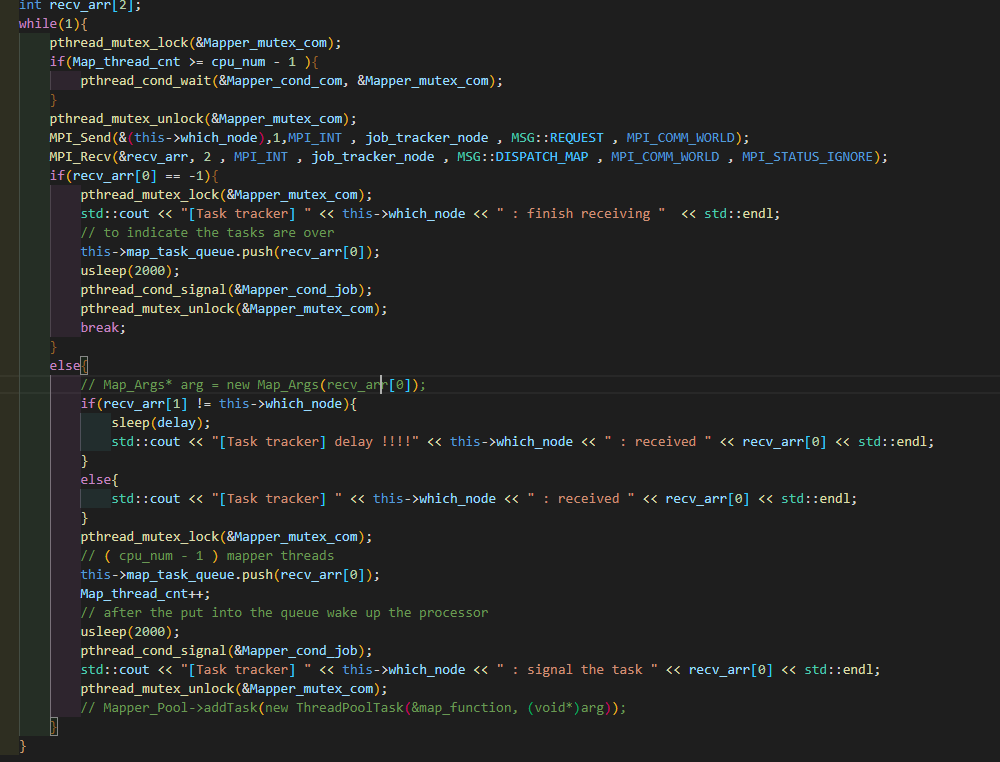


在tasktracker，我們會先將mapper thread全部assign到map\_pool裡面。這邊沒有使用助教給的threadpool是因為當時誤會了，因為遇到了一點困難，到最後使用手刻。帶difficulty也會提及。



在pool裡面，若今天要處理的queue是empty的話，他就會call condition wait並將threa put to sleep。相反的，有task要執行的話，他就會先去fetch那個task id ( 注意，因為我們在 JOB TRACKER結束後設為 -1 ，因此若是task\_queue裡面有 -1 則代表整體結束。並會退出整個while loop。) 接著將得到的資料已 ( word , value , hash value) 的方式存到intermediate file。

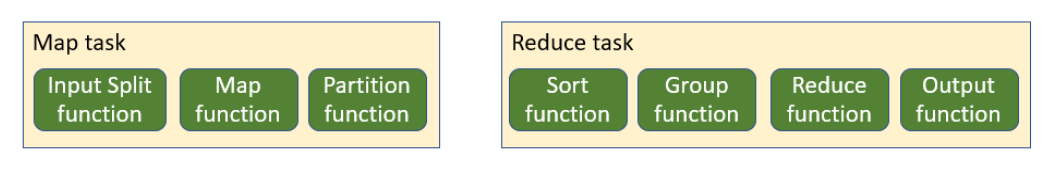
為了確保程式的正確性，我們在找到task與作整體而言有幾個map thread的部分，都是在critical section進行。



再回到function本身，整體而言。會先看整體map thread是否大於上限 ，若是，則將整個condition variable 進入wait。並將他halt掉。

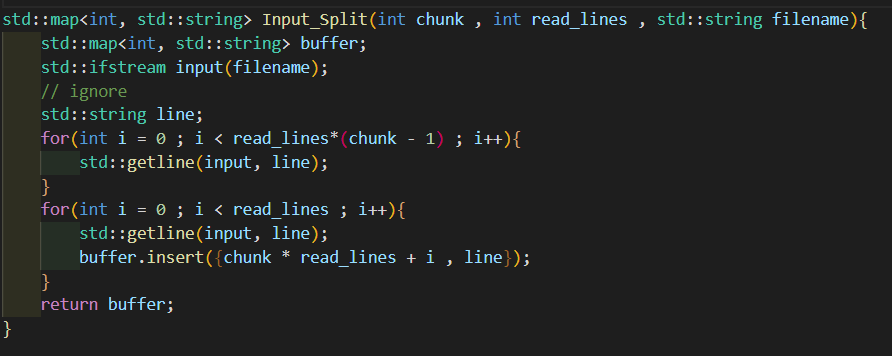
不然的話，會一直send request到本來的job tracker裡面。接著，接收從 job tracker出來的request，並且做一個delay的行為 (optional ) 。

若是今天他有將task push進去queue裡面，則會同時signal cond var。

****

接著講一下各自設定的function

Input split function:

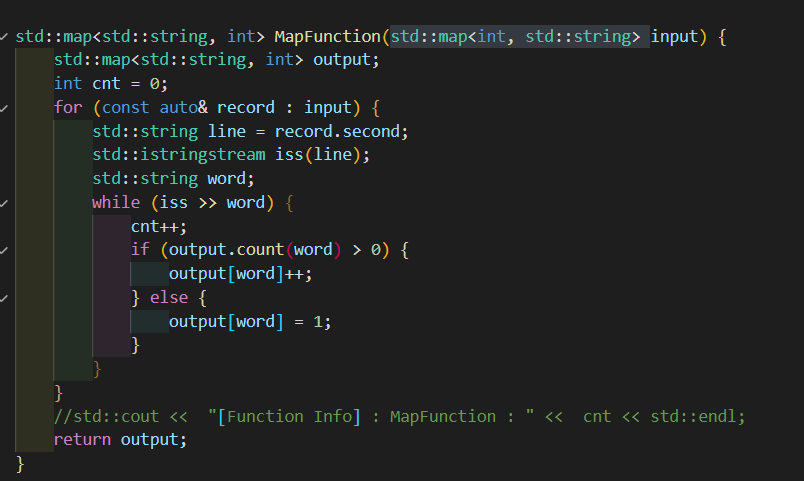


Input : num to be read , filename

Output : std::map<int, std::string>

作法先getline讀到應該要讀取的地方 ( 第一個for loop ) 接著在將他跑過需要讀的lines並將他傳到buffer裡面。

Map function:

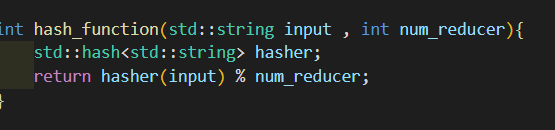


You can assume the data type of the input records is int (i.e., line#), and string (i.e., line text), and the data type of the output records is string (i.e.,word), and int (i.e., count).

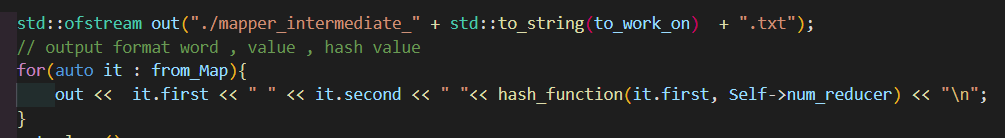
Input : num to be read , filename

因為之前是一行一行的讀近來，因此我們就會用一個 map 來看他有沒有到讀過，如果有的話就加一 沒有的話就 = 1

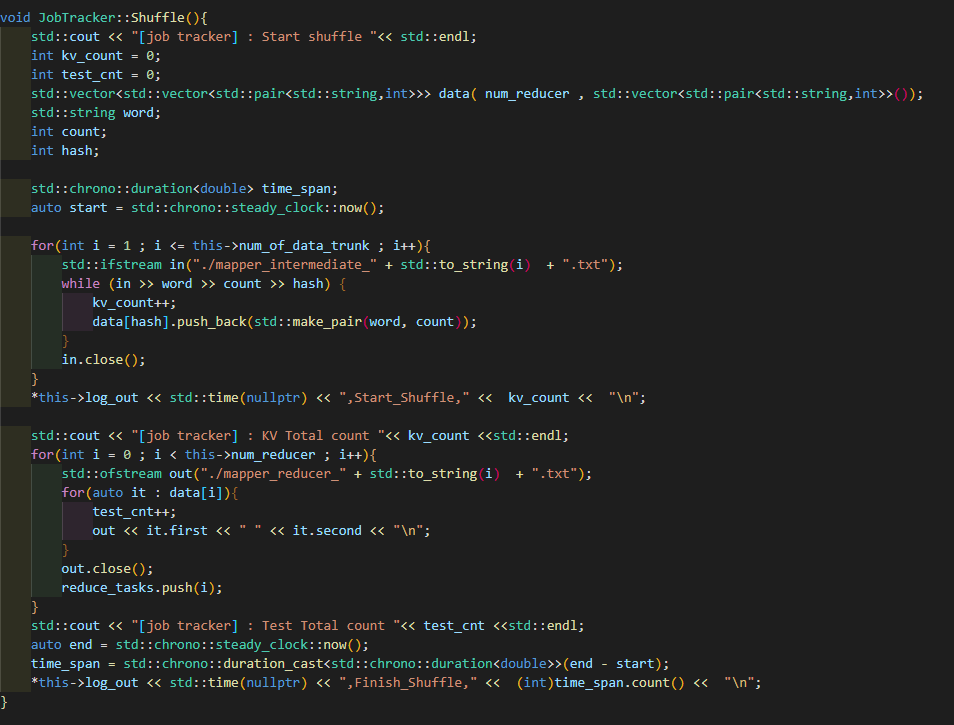
Partition function:



我使用的是 std::hash 他可以將任意input轉乘hash。並且在最後會 % num\_reducer。



注意 : 在mapper 的最後，他會將intermediate file傳出去，並在之後回收使用。

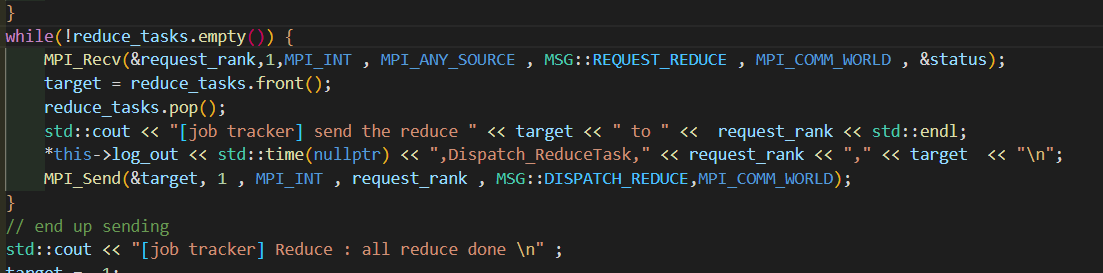


在shuffle這邊，我們會先將 word , count , hash傳進去，而他的data tpye為。

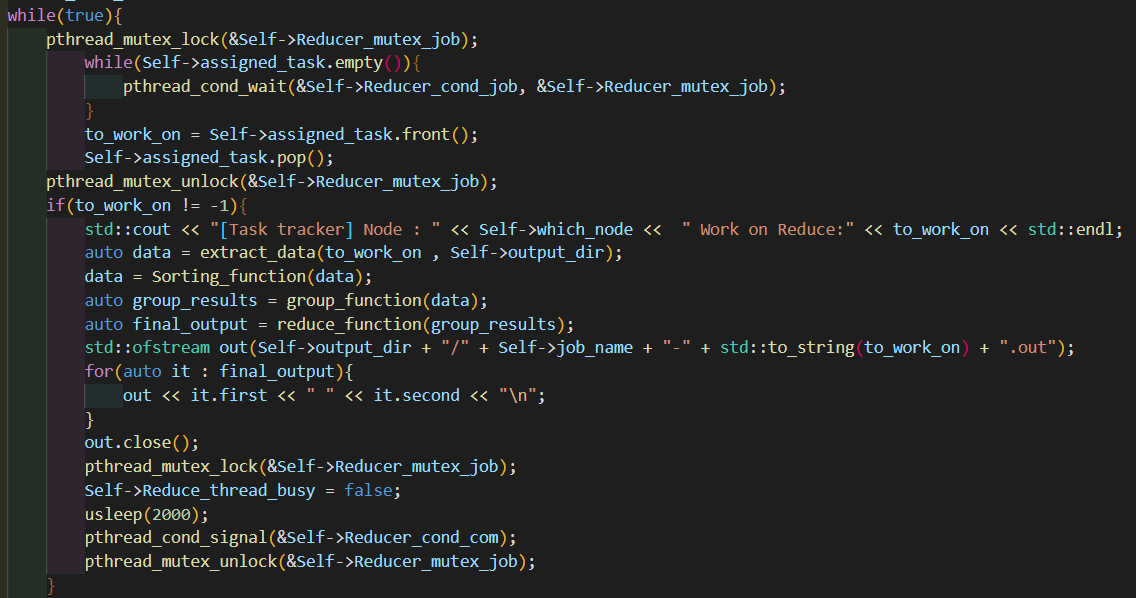
std::vector<std::vector<std::pair<std::string,int>>>

因為我們需要依據hash的結果將reducer的結果產生到home directory。接著再讓reducer使用這個file並做出相對應的處理。

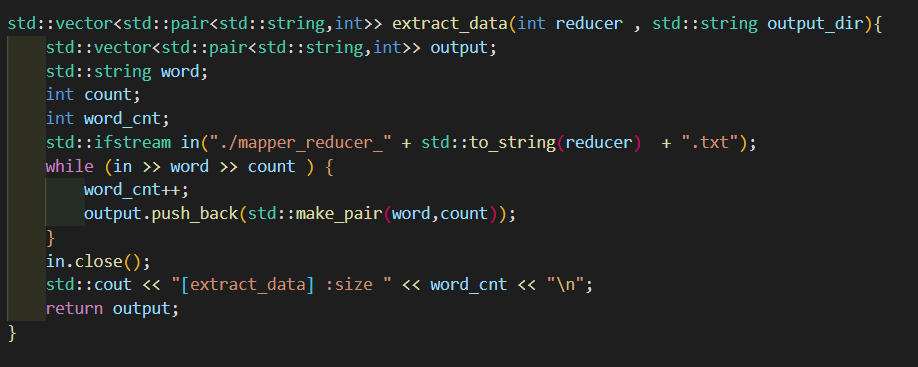
待完成後就可以準備將reducer task 傳給reducer。



做法跟map task一樣，唯一差別是不需要將因為沒有locality，因此不需要傳node數量給tasktracker。



因為這邊做法，基本上就會map task一樣。但唯一的差別就是他們的function不一樣。



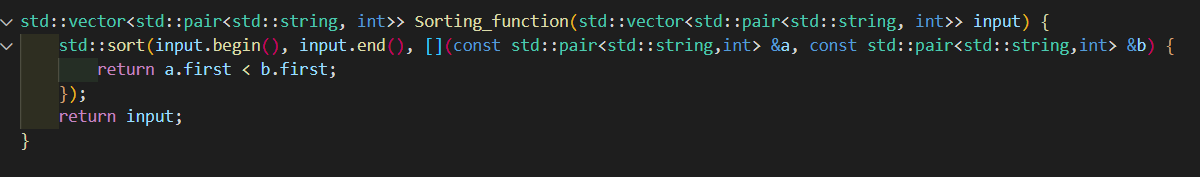
首先 extract data ,

Input : reducer number

Output: std::vector<std::pair<std::string,int>>

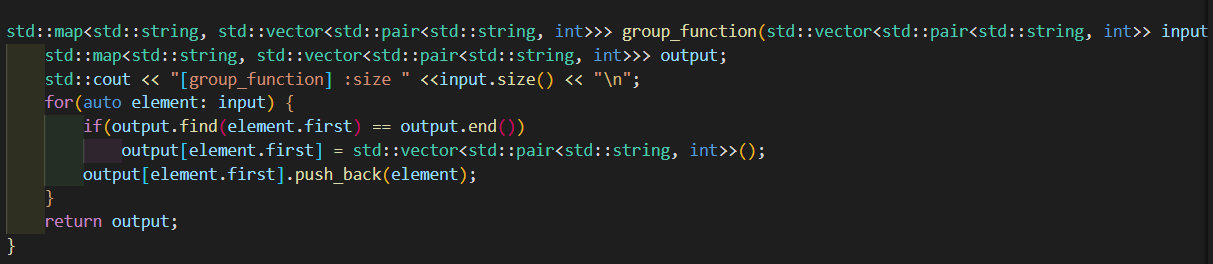
基本上就是將之前的資料讀出來，(每個reducer讀取需要負責的hash檔案的file)

Sort function:



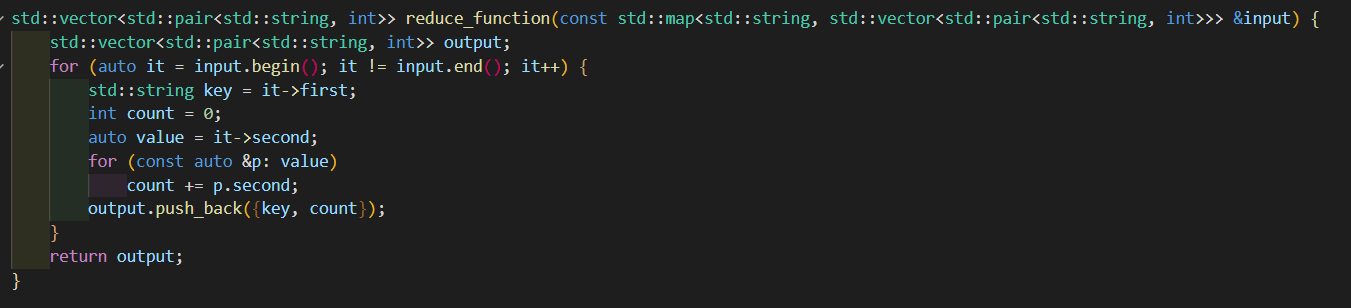
依據字典續將word由小排到大。

Group function:



用一個map將word設為key，並將一樣的key放到一起。並且回傳

Reduce function:



將同一個group裡面的東西加在一起，並且回傳一個 vector<string , int> 當成最終答案

最後再將整個reduce完的東西寫出去，整個mapreduce就完成了。

**challenges in your implementation :**

1. 型別問題:

在class裡面，若要class pthread的話



他的型別必須是要static void\* 而也是因為一開始搞不懂這點，因此才會拒絕選擇使用ta的code。 ( 最後手刻到一半才發現好像差不多)

“ When you pass static void\* map\_function(void\* arg) as the first argument to the ThreadPoolTask(void\* (\*f)(void\*), void\* arg) function, the type of the argument is void\* (\*)(void\*).

So, ThreadPoolTask function is expecting a function pointer as the first argument and the static void\* map\_function(void\* arg) is a function pointer that meets that requirement.

On the other hand, if void\* map\_function(void\* arg) is passed as the first argument , the ThreadPoolTask function will receive a function, not a function pointer and it will not match the expected argument type and it will cause a compile error. “

1. Time measurement :

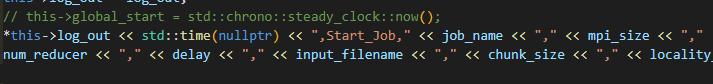
也是同樣的問題，若是在construct的時候將begin設為

// this->global\_start = std::chrono::steady\_clock::now();

並在最後destuctor的時候將他相減，不知道為甚麼他就是不會work。

Solution : 在main裡面做calculation

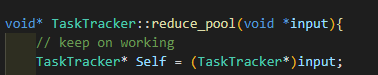
1. Log File generating



一開始也是出現了很多問題，因為就算是用 ofstream(…,std::ios::app) 他一樣會overwrite掉。造成log file無法正確。( 因為在main裡面也需要將時間算出來並寫進去 )

最後，就是將log file當成parameter傳入class內，才解決這個問題。

1. 不同的型別問題



在static 裡面，因為有諸多限制，因此若直接call class裡面的variables 會有error。因此，找了許多文獻後，發現要將自己放到裡面，之後call的時候從 self來抓，才不會有問題。

1. Delay問題

因為有時候太快的話會產生一些work on : -2345646

推測是因為太快了，我在signal前都加了usleep 來保證他不會因為還沒有sync就抓取資料。

**Experiments**

以下的testcase base:

Experiment 1 : {

    "NODES": 4,

    "CPUS": 8,

    "JOB\_NAME": "TEST04",

    "NUM\_REDUCER": 9,

    "DELAY": 1,

    "INPUT\_FILE\_NAME": "04.word",

    "CHUNK\_SIZE": 10,

    "LOCALITY\_CONFIG\_FILENAME": x

}

Experiment 2 : {

    "NODES": x

    "CPUS": 8

    "JOB\_NAME": "TEST06",

    "NUM\_REDUCER": 9,

    "DELAY": 0

    "INPUT\_FILE\_NAME": "06.word",

    "CHUNK\_SIZE": 20,

    "LOCALITY\_CONFIG\_FILENAME": "06.loc"

}

1. data locality

雖然說有差，但說到底的話就只是因為一個多做了一次sleep 因而造成這樣的差距。(因為程式跑太快，因此當一個卡住以後，另外一個就把剩下的做完了) 不過我們還是可以看到scalibility的差距

1. scalability

可以看到他的時間依據worker數下降了

但是他的scalability不好，推測是因為我有call usleep因此讓她無法完全的展示scalablity。

**Experience & conclusion:**

1. What have you learned from this homework?

學到了很多關於c++ class的細節，與一個mapreduce framework是如何被實踐的。