MapReduce

107062103 王依婷

Implementation

Map

Job tracker

在 Mapping 的時候，Job tracker 要負責的是接收 request for mapping task，並且根據 Locality 來分配。

- `Data locality-aware scheduling algorithm`：

  這邊的實作就是，會紀錄每一個 chunk 的 locality number，並且後面有 request 來的時候，就會去看有沒有哪一個 chunk 的 locality number 跟來的人是一樣的，如果都沒有就拿第一個，如果有就回傳那個

而當今天 mapping task 被拿完，就會 send 一個 signal 讓每個 process 知道 mapping task 做完了

Task tracker

在 Mapping 的階段，process 會去 Job tracker 拿 mapping task，直到他可用的 thread 都在做事才會 sleep，而當然，只要有 thread 做完事，都會去 signal Process。而每個 mapping task 都是由一個 pthread 來負責，他會去讀取他需要讀的 chunk，並且把他 parse 成每一個單字，接著也會將同樣的單字利用 map 把同樣的頻率加起來。最後

會把每個 chunk 的檔案都先寫進一個 `chunk file` 中，之後再讓 Job tracker 做 Shuffle 並且切分 reducing task。也會計算每個 Mapping task 的時間。

Shuffle & Intermediate

在 Shuffle 階段，因為我會先拿到每個 chunk fIle 的 word frequency，所以我會讀過每個 chunk 的 word frequency，並且利用 (`word[0]` % reducer number + 1)，來得到每個 reduce task 的 input key value pairs。

Reduce

Job tracker

他會等其他 process 來 request reducIng task，直到所有 reducing task 發送完，也會跟前面 mapping task 一樣，發送一個 end signal 來告知，不需要繼續索取了

Task tracker

在 Reducing 階段，每個 process 都會去跟 Job tracker 要 Reducing task，要到後，就是拿到一個 intermediate file，他會先做 Sorting，目前在這次作業我是先照著字典序來排序，這邊的 sort 用 std::sort。再來會把 word 完全相同的 group 在一起，。最後則是 Reducing step，他會把後面連接的單字，全部加起來，變成一個 map<string, int>的結果。最後再根據這個結果，來寫出這個 reduce task 的 Output。

Encounter Challenge

在實作上，已經對於MPI不熟悉，所以常常會有 Send 和 Recv 無法對上的狀態，有可能是 tag，有可能是 data number，都有可能導致錯誤，有發現一個解決辦法是，一直等到他被 kill 後，會顯示說是哪裡接收不到訊息的資訊跑出來，就可以利用那邊 debug。

還有這次也經歷過痛苦的 segmentation fault 的階段，中間也有利用過 fsanitize=address 的 compiler flag 來幫助 Debug，發現是在 string assigning 的時候出錯，後來上網查才發現有人說，C++的東西不要用 c 的 malloc 要用 new。

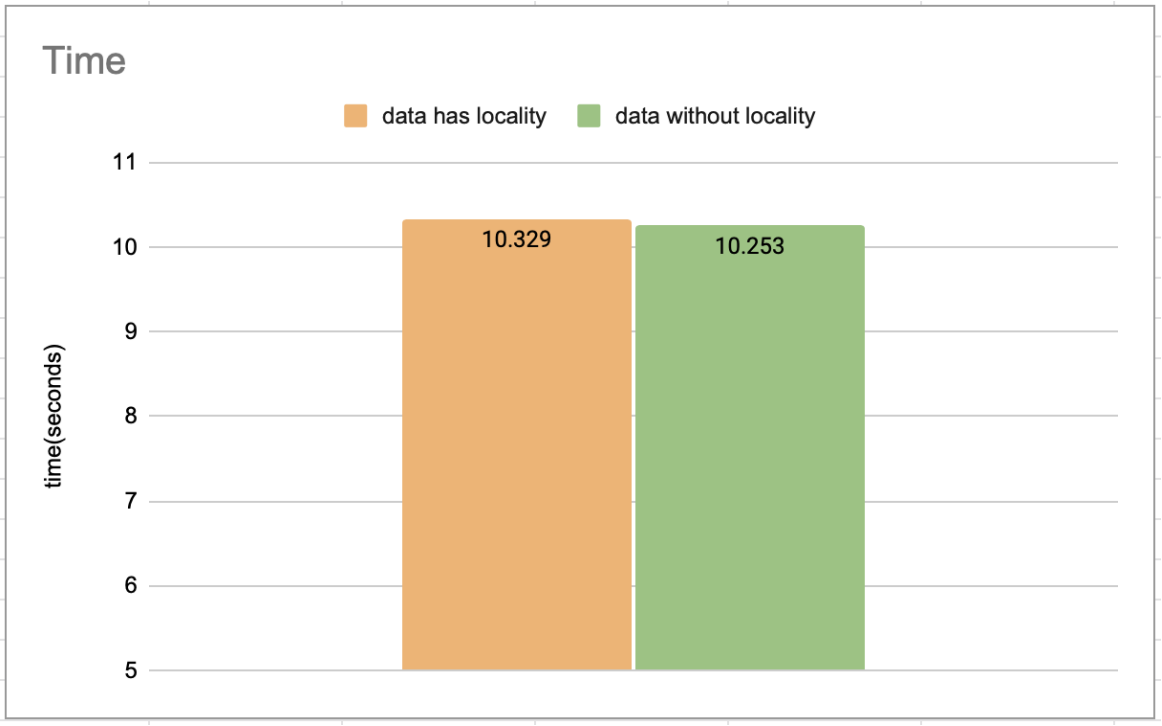
Analysis and some Experiment

System Spec

在這次作業中使用課程中提供的 apollo server。

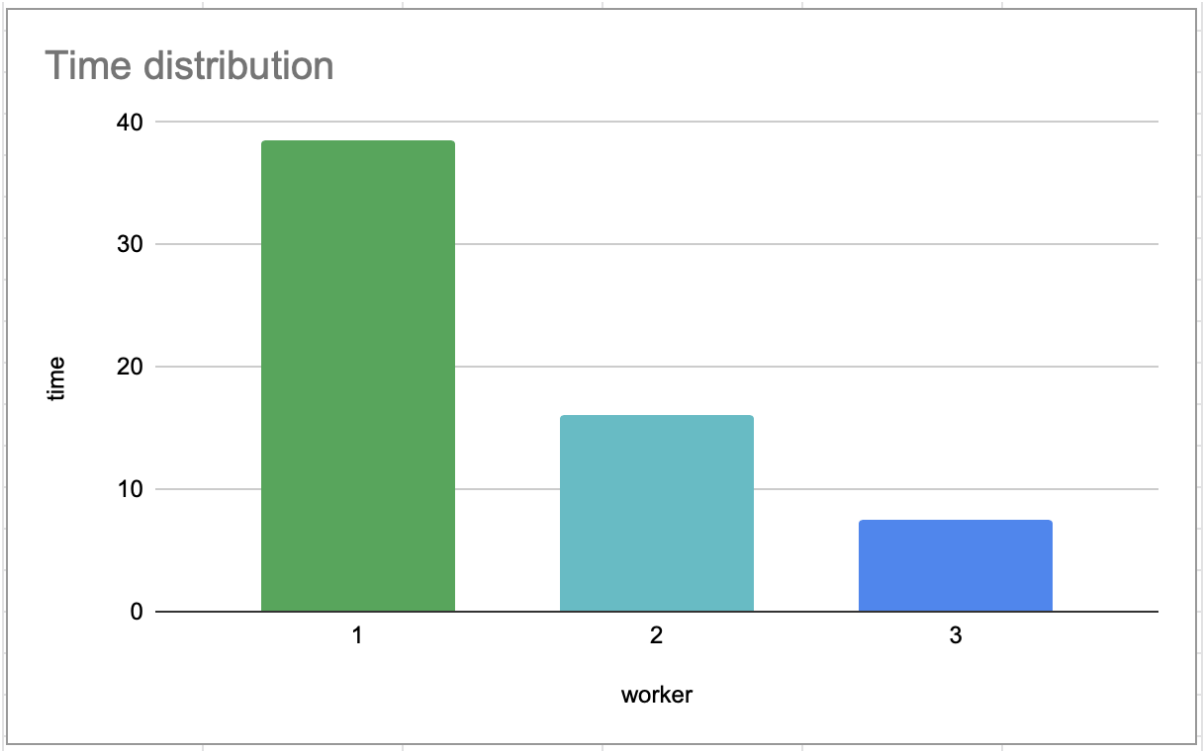
Data locality

- 可以看到下圖的 locality 並沒有造成太大的影響，推測是因為計算的時間太短了，應該要讓 mapping task 可以拉長，才能看出與 sleep 的差別，否則就會如下圖，雖然有 data locality 但可能還比不上其他人都去 sleep 只讓一個 process 的 thread 來做。



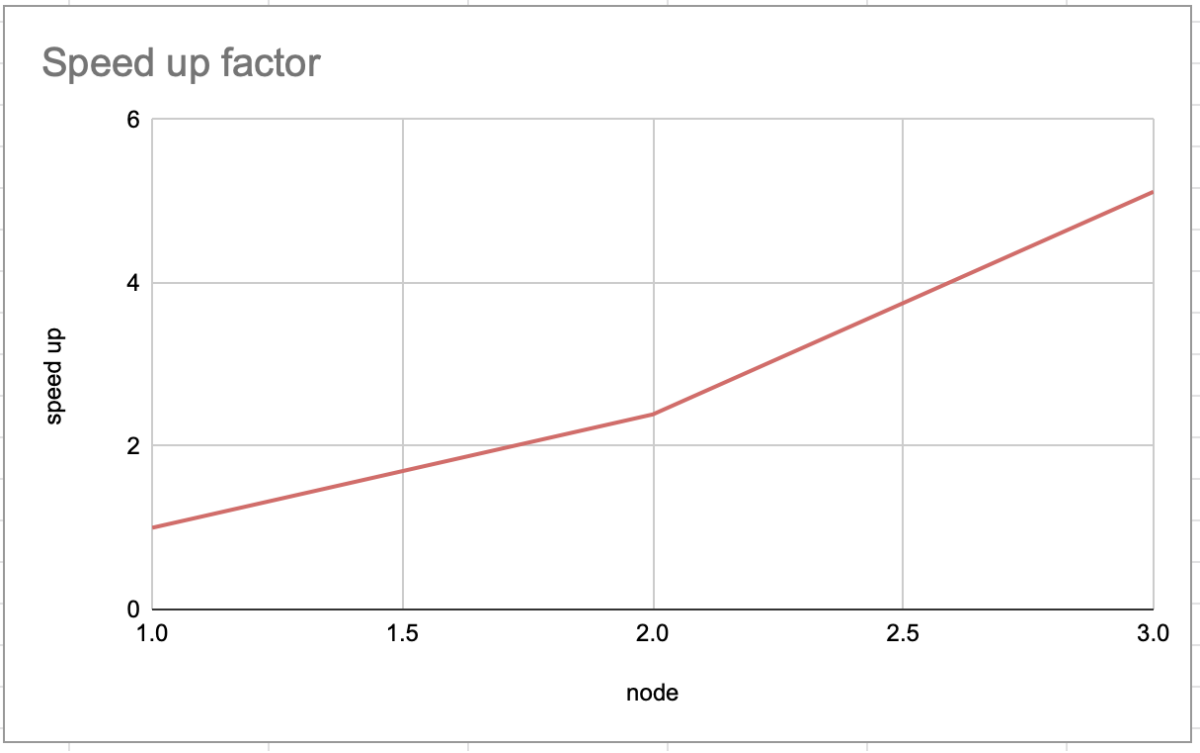
Time profile

- 可以看到隨著使用的 node 數量增加(worker 的數量)，時間也可以從原本的將近40秒，下降到大約5秒就可以完成。



Speedup factor

- 可以看到 speed up 的結果，在有兩台 worker 的時候，可以快大約 2.3 倍，而在 3 台 worker 的情況下可以有 5.1 倍的 speed up。



Conclusion

在這次作業中，第一次試了 pthread + MPI 的組合，雖然過程中因為期末壓力，導致時間有點緊迫，但這次做出來也是十分有成就感，這次也額外瞭解了 MapReduce 的架構，未來在使用 MapReduce 的時候，也可以不只是盲目的使用，也是能了解背後實作的原理以及 architecture。