**Homework 4:  MapReduce**

**Implementation**

**jobtracker & task tracker:**

**map & reduce:**

job tracker在map與reduce的階段會分別與task trackers進行三次溝通。

1. 接收task trackers的job request:

當job tracker接收到request後，會根據data locality去分配工作。透過便利全部的jobs，當發現存在job的node id跟request node id相同的話，就會優先將該job分配給該node。反之，就會根據FIFO的原則，分配job給node。

1. 當全部的jobs都被指派出去：

當全部的jobs都被指派出去，job tracker如果再接收到job request，就會發送訊息給task trackers，告之已經沒有工作要做了，讓task tracker不用在發送job request。

1. 發送job完成訊息給job tracker:

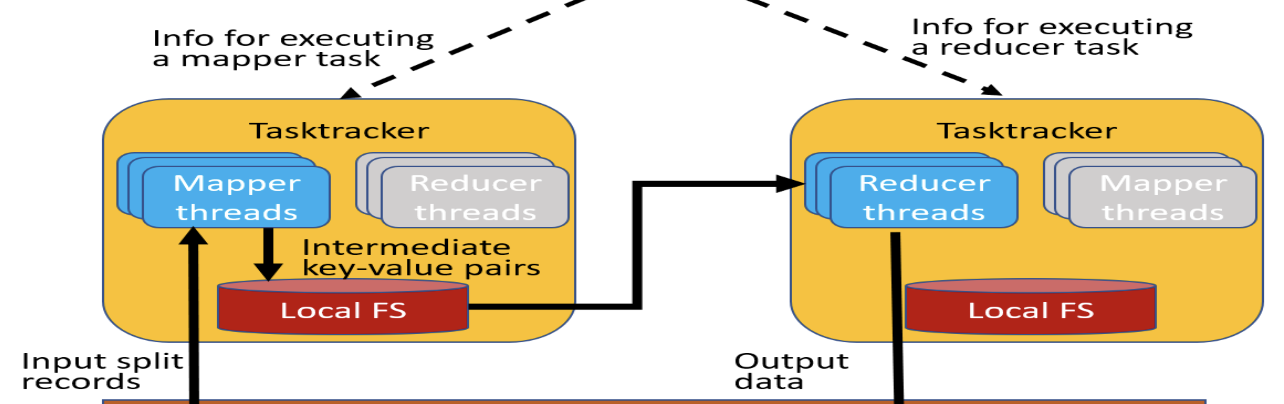
當jobs都完成之後，每個task trackers會再傳送jobs的訊息給job tracker，像task id、execution time等等。方便接下來的紀錄。

**mapper threads & reducer threads**

1. 程式一開始會先create很多threads，並讓這些threads去job queue拿task執行。當task trackers接收到工作之後，會將job submit給thread pool。當threads在拿task時，首先會將mutex 鎖住，接下來會檢查task queue是否有task，如果queue是空的，就會wait on一個condition variable，直到有新的task被加入到task queue時，才會重新signal condition variable，讓threads重新拿到新的task。
2. 當task tracker向job tracker要求job request時，也會用mutex跟condition variable來操控。每次在發送request前，會先檢查每個threads是否都有job，如果每個job都有工作的話，就會wait on condition variable，直到有threads空出來，才會重新signal這個condition variable。

**Intermediate file**

從spec可以看到，當mapper threads處理完一個task時，就會產生intermediate key-value pairs，我會將他先儲存成file的形式。當job tracker接收到task tracker的完成訊息後，job tracker就會先創建N的reducer檔案，並讀取每個mapper task產生的intermediate key-value pairs，並依照partition function分配給特定的reducer。



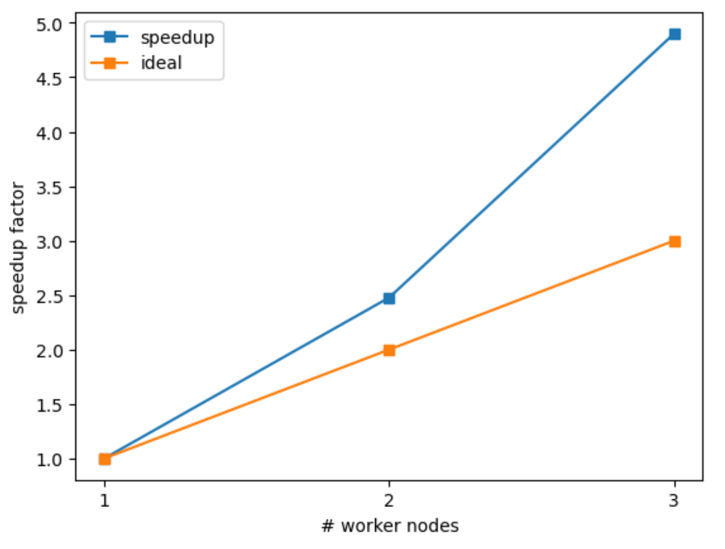
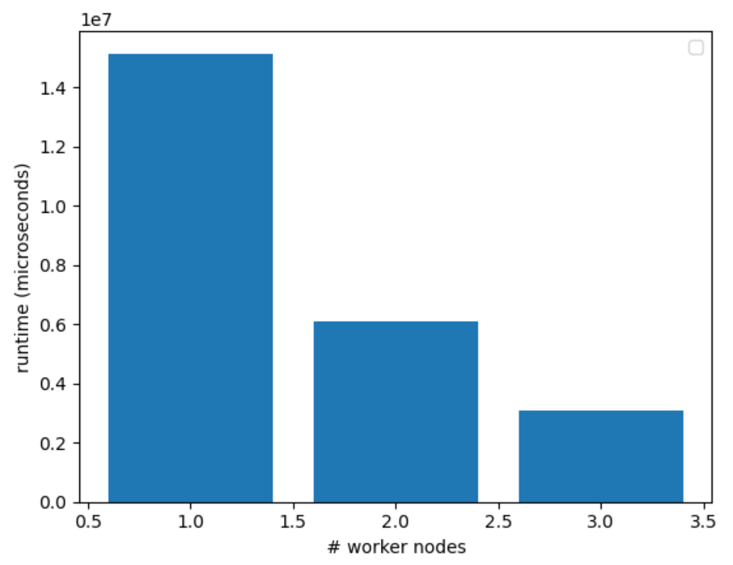
**Experiment**

1. System spec: 學校的系統環境

一張含有 文字 的圖片

自動產生的描述

1. Speedup Factor & Time Profile

我用的測試參數是testcases/04.json，並調整node的數量。

因為有一個node固定要是job tracker，因此worker node最大數量為3。從speedup factor的圖表上可以看到，當worker nodes的數量增多的時候，speedup factor也有隨之上升。且隨著worker nodes持續增加，speedup的程度甚至超過ideal的線條，我推測是因為有考慮data locality。因此可以了解到，程式的scalability非常好。我另外也把time profile給畫出來，可以看到runtime大幅度的下降。

**conclusion**

透過這一次的實驗，我除了更加了解mapreduce的整體架構之外，對於MPI與pthread的操作也更加熟悉。對於data locality對於程式的影響也有更加深刻的認識。