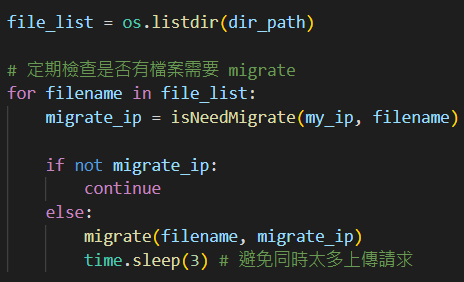
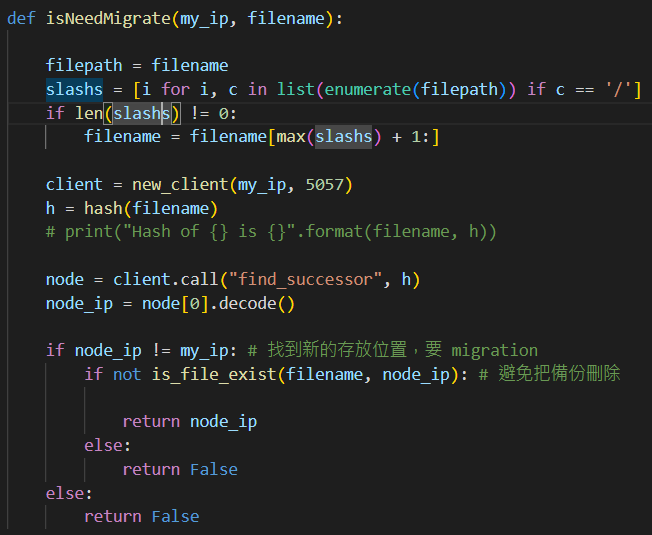
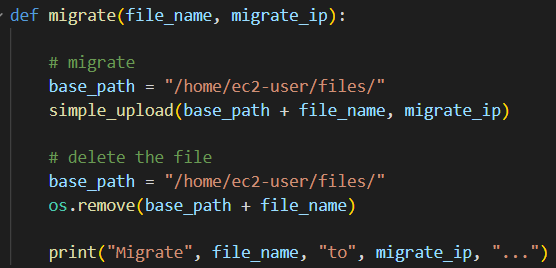
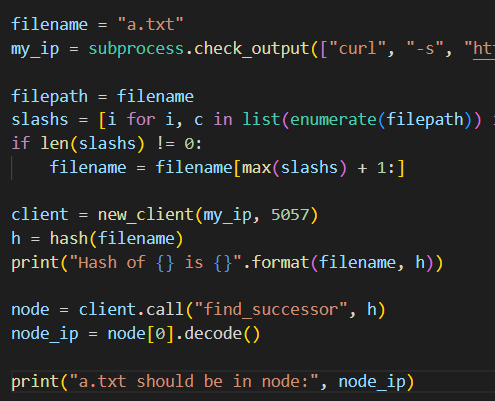
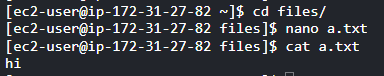
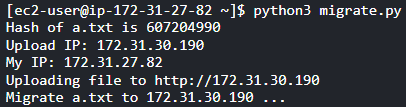
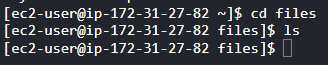
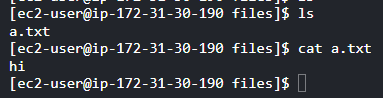
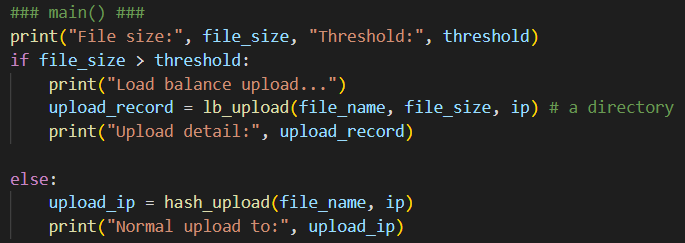
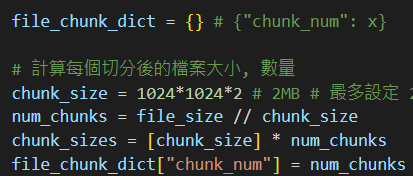
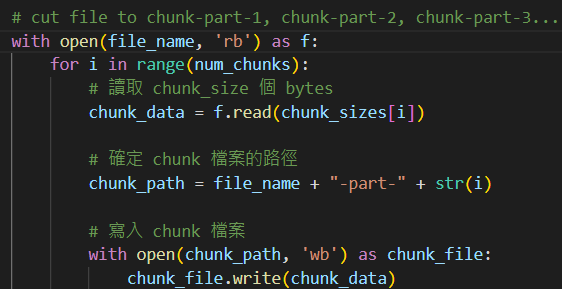
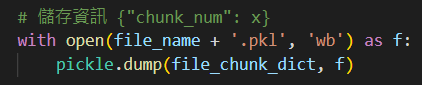
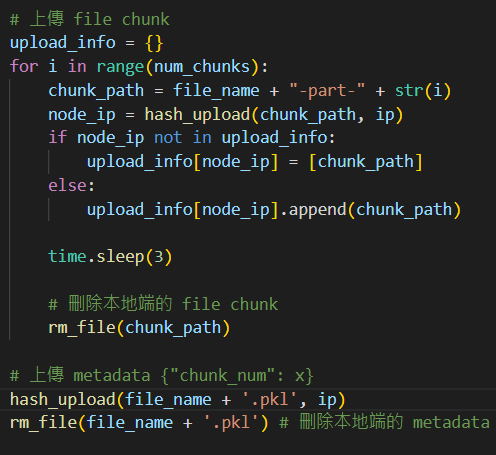
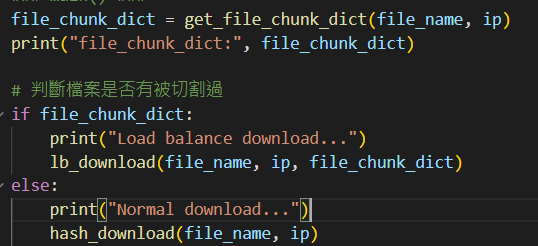
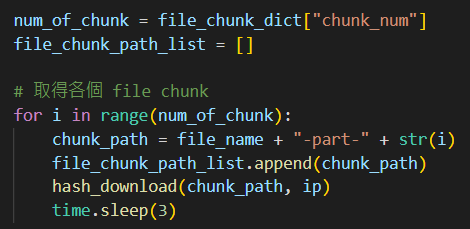
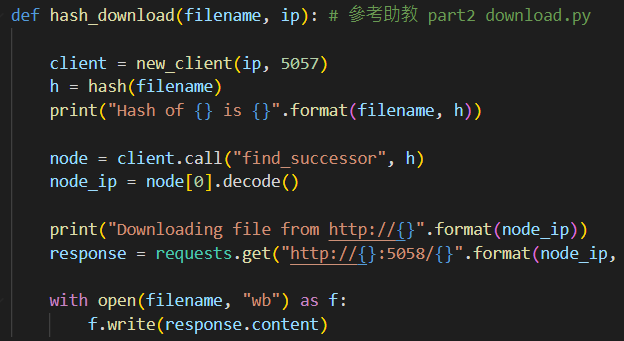
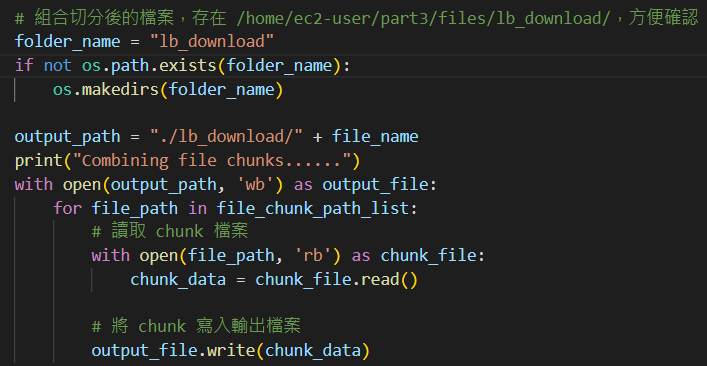
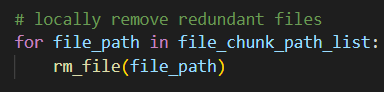
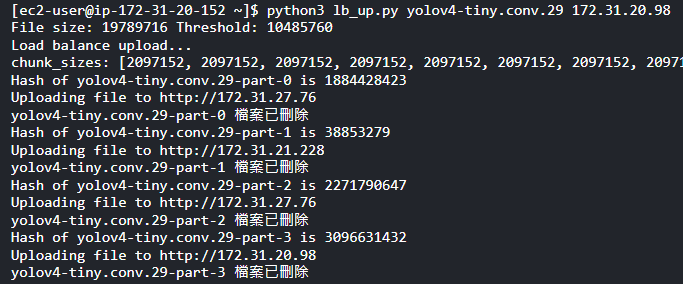
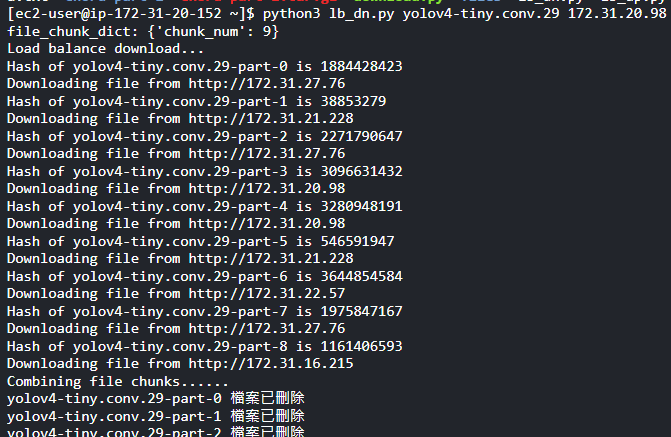
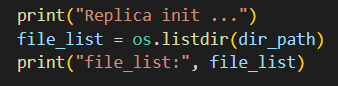
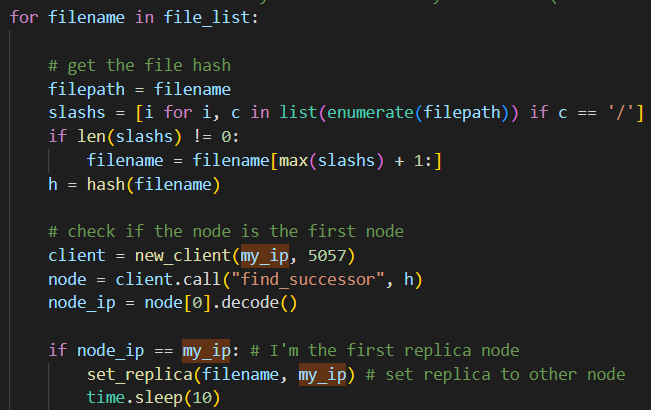
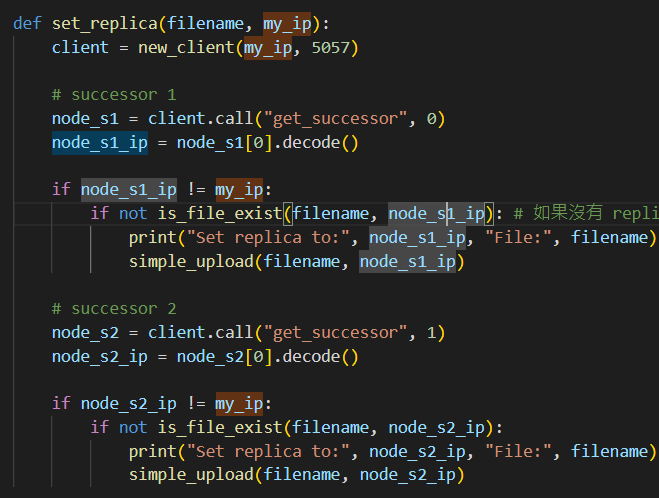
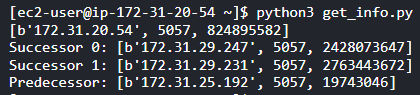
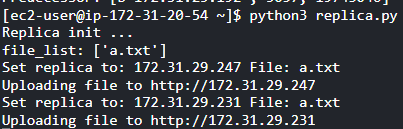
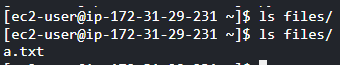
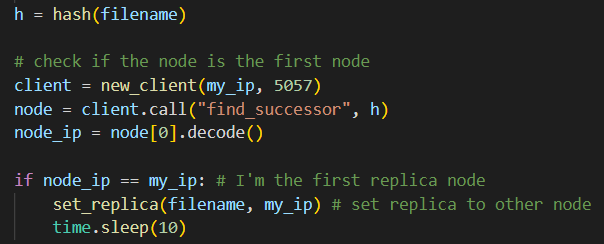
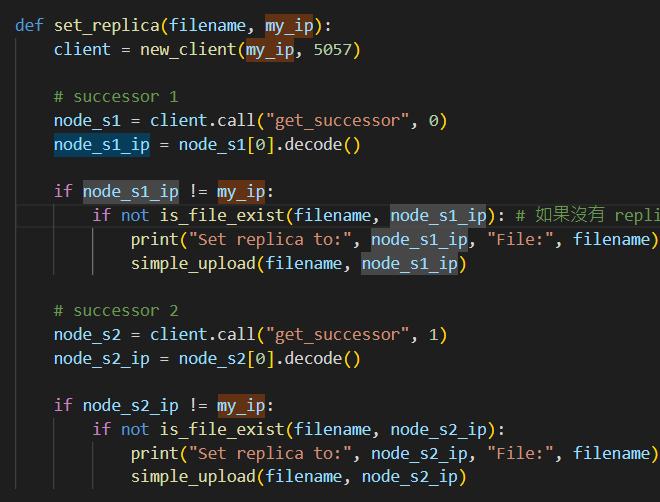
**Correctness - Data Migration (33%)**

* System components
  + migrate.py
    - 每個chord node都會有migrate.py，每60秒會監控 /home/ec2-user/files/ 這個資料夾，確認裡面的資料是否需要進行migrate
      * 這是一個trade-off，監控時間越短系統越不會出現來不及migrate的情形，但成本也越高。
  + Cronjob
    - 
    - 我是以crontab的方式讓其每60秒執行一個migrate.py
    - 
      * 可以將這一行加入AWS launch template的user-data中，這樣chord透過auto scaling group啟動時就會自動將migrate排進定時任務中
* System functionalities
  + 
    - 每個Node監控自己儲存上傳檔案的資料夾 (dir\_path)，預設是/home/ec2-user/files
      * 定期將每個檔案抓出來確認，看需不需要migrate
      * 1分鐘檢查一次，過多的檢查會浪費資源
        + 這是一個trade-off，監控時間越短系統越不會出現來不及搬移的情形，但成本也越高。
    - 以下介紹兩個主要的函數
      * isNeedMigrate(my\_ip, filename)
      * migrate(file\_name, migrate\_ip)
    - 
      * 將當前檔案拿去hash，並用find successor確認是否應該是找到自己，如果不是就代表需要migrate
    - 
      * 這裡很單純，就是將檔案upload到剛剛find successor找到的migrated IP，把檔案傳過去後再把本地的檔案刪除
* Experiment
  + 這裡我們用一個 find\_ip.py來協助實驗進行
  + 
    - 假設我們要進行migrate的檔案是 a.txt，用find\_ip.py事先找出使用find successor後會應該要放在哪個node，把這個node當作之後才加入的node，然後將檔案migrate到這個node
      * 因為每次auto scaling group的IP都不一定，這樣node的串接也不一定，為了實驗上的方便才這樣做，不然很難找到一個node加入時剛好是要migrate的node
    - 
      * 從 .82 這個Node可以看到此檔案在 .190這個Node加入後，檔案應該要migrate過去
    - 
      * 因此在 .82 這個Node新增a.txt，當作搬移的檔案
    - 
      * .190 目前是沒有任何檔案的
  + 
    - 如圖所示，發現a.txt需要migrate，因此upload到指定IP
  + 
    - 上傳完成後，檢查 .82 的確沒有 a.txt 了
  + 
    - 而 .190 則收到 a.txt 了

**Load balance - File Chunks (33%)**

* System components:
  + lb\_up.py: 用來進行 load balance upload
  + lb\_dn.py: 用來進行 load balance download
  + yolov4-tiny.conv.29 : 19M的大檔案，用來當作範例
* System functionalities
  + lb\_up.py
    - 
      * 如果檔案過大(門檻=10M) 就用 load balance 的方式上傳
    - 
      * 將數據切成2MB的chunk，並且將chunk數量存進一個dict裡面當作metadata
    - 
      * 將資料依照剛剛的計算來切分檔案，並用-part-來分辨是第幾個chunk
    - 
      * 將metadata (共有幾個chunk)的資訊存成.pkl
    - 
      * 最後實際把這些檔案經過hash之後，上傳到指定node
      * Metadata的格式是{"chunk\_num": num\_of\_chunk}，用來幫助lb\_dn.py
  + lb\_dn.py
    - 
      * 先嘗試取得metadata (chunk dict)，如果有就代表此檔案有經過切割，需要以load balance的方式下載
    - 
    - 
      * 透過metadata可以得知共有多少chunk，依據這個資訊經過hash之後的值用find successor找到指定node下載
    - 
      * 下載完後將各個chunk組合，並存在 ./lb\_download/裡面，方便進行檢查
    - 
      * 移除用完的chunk
* Experiment
  + Pre-requirement
    - 先開好數個chord node (我這裡開5個)
    - 準備好一個大於10M的大檔案來觸發lb\_up.py設的threshold
      * 
  + lb\_up.py
    - 
      * 如圖所示，lb\_up.py會先將檔案切割後並進行hash之後再用find\_successor()找到node來上傳檔案，上傳完後也會刪除本地切割出來的file chunk
    - 
      * 程式碼最下方也會印出上傳的詳細資訊，方便我們檢查
    - 
    - 
    - 
    - 
    - 
      * 可以看到總共有0~8 (9個file chunk)上傳至5個Node
      * 並且有一個Node會收到.pkl的metadata，用來輔助下載
  + lb\_dn.py
    - 
      * 可以看到lb\_dn.py會先取得metadata(總共有多少file chunk)並印出來，然後將每個file chunk都先拿去hash並用find successor()找出儲存的node，找到之後再下載
      * 最後把各個file chunk組合起來後再把不會用到的這些file chunk給清除乾淨
    - 
      * 為了方便檢查，程式會自動create一個資料夾lb\_download，可以看到下載的檔案在裡面
      * 
        + 檢查後也可以發現，他跟我們原始檔案一樣是19M

**Fault Tolerance - Replication (33%)**

* System components
  + replica.py
  + 每個chord node都會有replica.py，每60秒會監控 /home/ec2-user/files/ 這個資料夾，確認裡面的資料是否需要進行備份
    - 這是一個trade-off，監控時間越短系統越不會出現來不及備份的情形，但成本也越高；這裡設定60秒是假設3個replica不會在60秒內一次全部壞掉。
  + Cronjob
    - 
      * 可以連同migrate.py寫在一起
    - 我是以crontab的方式讓其每60秒執行一個replica.py
    - 
      * 可以將這一行加入AWS launch template的user-data中，這樣chord透過auto scaling group啟動時就會自動將migrate排進定時任務中
      * 這樣也可以避免python執行while迴圈浪費資源
* System functionalities
  + 
    - replica.py 會監控 /home/ec2-user/files/ 這個資料夾，所以會先將其中所有的檔案取出來放進變數file\_list裡面
  + 
    - 每次都會用find successor (hash(file name))去查看當前自己是持有此file的第一個node，如果自己是第一個node就要負責維護所有replica
  + 
    - 持有file的第一個node要像後面兩個successor放置replica，如果已經有檔案就不要重複放置
* Experiment
  + 為了方便實驗，我們直接模擬發現需要進行replica的情況，而不是等她每60秒再維護一次系統
  + 
    - 首先，我們將a.txt上傳自chord node
    - 
  + 
    - 用get\_info()取得successor資訊，來確認等等誰應拿到replica
    - 
    - 
      * 目前兩人都還沒有replica
  + 
    - 這時我們直接執行replica.py，模擬電腦監控到需要放replica的event，他就會上傳檔案到指定位置
    - 
    - 
      * 可以看到他們倆個都拿到replica了
  + 如何確保 kill 之後還是會有3份replica?
    - 
      * 因為如上圖所示，系統會由**擁有replica的第一個node來負責維護，可以由find successor確認自己是否為第一個node**，當前第一個node被砍掉後用file successor()找到的node就會變成原本的第二個node，改由原本第二個node當first node來維護replica
        + 假如前兩個node都掛了，第三個node經過find successor()就會發現自己變第一個node，以此類推
    - 
      * 當確認自己是當前第一個node之後，就會使用set replica的功能去幫後面兩個successor放置replica，如果後面兩個人已經有replica就不會重複放置
      * 這樣就可以確保整個系統有3份replica