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
  - ```
    * * * * * /usr/bin/python3 /home/ec2-user/migrate.py >> /home/ec2-user/migrate_log.txt 2>&1
    ```
  - 我是以 crontab 的方式讓其每 60 秒執行一個 migrate.py
  - ```
    /usr/bin/crontab -u ec2-user /home/ec2-user/cronjob
    ```
    - 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

- ```python
  file_list = os.listdir(dir_path)

  # 定期檢查是否有檔案需要 migrate
  for filename in file_list:
      migrate_ip = isNeedMigrate(my_ip, filename)

      if not migrate_ip:
          continue
      else:
          migrate(filename, migrate_ip)
          time.sleep(3) # 避免同時太多上傳請求
  ```
  ```

- 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)
-

```
def isNeedMigrate(my_ip, filename):

    filepath = filename
    slashs = [i for i, c in list(enumerate(filepath)) if c == '/']
    if len(slashs) != 0:
        filename = filename[max(slashs) + 1:]

    client = new_client(my_ip, 5057)
    h = hash(filename)
    # print("Hash of {} is {}".format(filename, h))

    node = client.call("find_successor", h)
    node_ip = node[0].decode()

    if node_ip != my_ip: # 找到新的存放位置，要 migration
        if not is_file_exist(filename, node_ip): # 避免把備份刪除

            return node_ip
        else:
            return False
    else:
        return False
```

  - 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

- ```
  def migrate(file_name, migrate_ip):

      # migrate
      base_path = "/home/ec2-user/files/"
      simple_upload(base_path + file_name, migrate_ip)

      # delete the file
      base_path = "/home/ec2-user/files/"
      os.remove(base_path + file_name)

      print("Migrate", file_name, "to", migrate_ip, "...")
  ```

  - 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

- ```
  filename = "a.txt"
  my_ip = subprocess.check_output(["curl", "-s", "ht

  filepath = filename
  slashs = [i for i, c in list(enumerate(filepath))
  if len(slashs) != 0:
      filename = filename[max(slashs) + 1:]

  client = new_client(my_ip, 5057)
  h = hash(filename)
  print("Hash of {} is {}".format(filename, h))

  node = client.call("find_successor", h)
  node_ip = node[0].decode()

  print("a.txt should be in node:", node_ip)
  ```

  - 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

```
[ec2-user@ip-172-31-27-82 ~]$ python3 find_ip.py
Hash of a.txt is 607204990
a.txt should be in node: 172.31.30.190
```

- 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

```
[ec2-user@ip-172-31-27-82 ~]$ cd files/
[ec2-user@ip-172-31-27-82 files]$ nano a.txt
[ec2-user@ip-172-31-27-82 files]$ cat a.txt
hi
```

- 因此在 .82 這個 Node 新增 a.txt，當作搬移的檔案

```
[ec2-user@ip-172-31-30-190 ~]$ cd files/
[ec2-user@ip-172-31-30-190 files]$ ls
[ec2-user@ip-172-31-30-190 files]$
```

- .190 目前是沒有任何檔案的

```
[ec2-user@ip-172-31-27-82 ~]$ python3 migrate.py
Hash of a.txt is 607204990
Upload IP: 172.31.30.190
My IP: 172.31.27.82
Uploading file to http://172.31.30.190
Migrate a.txt to 172.31.30.190 ...
```

- 如圖所示，發現 a.txt 需要 migrate，因此 upload 到指定 IP

```
[ec2-user@ip-172-31-27-82 ~]$ cd files
[ec2-user@ip-172-31-27-82 files]$ ls
[ec2-user@ip-172-31-27-82 files]$
```

- 上傳完成後，檢查 .82 的確沒有 a.txt 了

-
```
[ec2-user@ip-172-31-30-190 files]$ ls
a.txt
[ec2-user@ip-172-31-30-190 files]$ cat a.txt
hi
[ec2-user@ip-172-31-30-190 files]$
```
    - 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
    -
```
### main() ###
print("File size:", file_size, "Threshold:", threshold)
if file_size > threshold:
    print("Load balance upload...")
    upload_record = lb_upload(file_name, file_size, ip) # a directory
    print("Upload detail:", upload_record)

else:
    upload_ip = hash_upload(file_name, ip)
    print("Normal upload to:", upload_ip)
```
        - 如果檔案過大(門檻=10M) 就用 load balance 的方式上傳
    -
```
file_chunk_dict = {} # {"chunk_num": x}

# 計算每個切分後的檔案大小, 數量
chunk_size = 1024*1024*2 # 2MB # 最多設定
num_chunks = file_size // chunk_size
chunk_sizes = [chunk_size] * num_chunks
file_chunk_dict["chunk_num"] = num_chunks
```

- 將數據切成 2MB 的 chunk，並且將 chunk 數量存進一個 dict 裡面當作 metadata

```
# cut file to chunk-part-1, chunk-part-2, chunk-part-3...
with open(file_name, 'rb') as f:
    for i in range(num_chunks):
        # 讀取 chunk_size 個 bytes
        chunk_data = f.read(chunk_sizes[i])

        # 確定 chunk 檔案的路徑
        chunk_path = file_name + "-part-" + str(i)

        # 寫入 chunk 檔案
        with open(chunk_path, 'wb') as chunk_file:
            chunk_file.write(chunk_data)
```

- 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

```
# 儲存資訊 {"chunk_num": x}
with open(file_name + '.pkl', 'wb') as f:
    pickle.dump(file_chunk_dict, f)
```

- 將 metadata (共有幾個 chunk)的資訊存成.pkl

```
# 上傳 file chunk
upload_info = {}
for i in range(num_chunks):
    chunk_path = file_name + "-part-" + str(i)
    node_ip = hash_upload(chunk_path, ip)
    if node_ip not in upload_info:
        upload_info[node_ip] = [chunk_path]
    else:
        upload_info[node_ip].append(chunk_path)

    time.sleep(3)

    # 刪除本地端的 file chunk
    rm_file(chunk_path)

# 上傳 metadata {"chunk_num": x}
hash_upload(file_name + '.pkl', ip)
rm_file(file_name + '.pkl') # 刪除本地端的 metadata
```

- 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

◆
```
file_chunk_dict = get_file_chunk_dict(file_name, ip)
print("file_chunk_dict:", file_chunk_dict)

# 判斷檔案是否有被切割過
if file_chunk_dict:
    print("Load balance download...")
    lb_download(file_name, ip, file_chunk_dict)
else:
    print("Normal download...")
    hash_download(file_name, ip)
```

- 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

◆
```
num_of_chunk = file_chunk_dict["chunk_num"]
file_chunk_path_list = []

# 取得各個 file chunk
for i in range(num_of_chunk):
    chunk_path = file_name + "-part-" + str(i)
    file_chunk_path_list.append(chunk_path)
    hash_download(chunk_path, ip)
    time.sleep(3)
```

◆
```
def hash_download(filename, ip): # 參考助教 part2 download.py

    client = new_client(ip, 5057)
    h = hash(filename)
    print("Hash of {} is {}".format(filename, h))

    node = client.call("find_successor", h)
    node_ip = node[0].decode()

    print("Downloading file from http://{}".format(node_ip))
    response = requests.get("http://{}:5058/{}".format(node_ip,

    with open(filename, "wb") as f:
        f.write(response.content)
```

- 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

```
# 組合切分後的檔案，存在 /home/ec2-user/part3/files/lb_download/，方便確認
folder_name = "lb_download"
if not os.path.exists(folder_name):
    os.makedirs(folder_name)

output_path = "./lb_download/" + file_name
print("Combining file chunks......")
with open(output_path, 'wb') as output_file:
    for file_path in file_chunk_path_list:
        # 讀取 chunk 檔案
        with open(file_path, 'rb') as chunk_file:
            chunk_data = chunk_file.read()

        # 將 chunk 寫入輸出檔案
        output_file.write(chunk_data)
```

- 下載完後將各個 chunk 組合，並存在 ./lb_download/裡面，方便進行檢查

```
# locally remove redundant files
for file_path in file_chunk_path_list:
    rm_file(file_path)
```

- 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
    -
```
[ec2-user@ip-172-31-20-152 ~]$ ls -lh yolov4-tiny.conv.29
-rw-rw-r-- 1 ec2-user ec2-user 19M Apr 26 02:26 yolov4-tiny.conv.29
```
- lb_up.py

- ◆
```
[ec2-user@ip-172-31-20-152 ~]$ python3 lb_up.py yolov4-tiny.conv.29 172.31.20.98
File size: 19789716 Threshold: 10485760
Load balance upload...
chunk_sizes: [2097152, 2097152, 2097152, 2097152, 2097152, 2097152, 2097152, 2097
Hash of yolov4-tiny.conv.29-part-0 is 1884428423
Uploading file to http://172.31.27.76
yolov4-tiny.conv.29-part-0 檔案已刪除
Hash of yolov4-tiny.conv.29-part-1 is 38853279
Uploading file to http://172.31.21.228
yolov4-tiny.conv.29-part-1 檔案已刪除
Hash of yolov4-tiny.conv.29-part-2 is 2271790647
Uploading file to http://172.31.27.76
yolov4-tiny.conv.29-part-2 檔案已刪除
Hash of yolov4-tiny.conv.29-part-3 is 3096631432
Uploading file to http://172.31.20.98
yolov4-tiny.conv.29-part-3 檔案已刪除
```

  - 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

- ◆
```
Upload detail: {'172.31.27.76': ['yolov4-tiny.conv.29-part-0', 'yolov4-tiny.conv.29-part-2', 'yolov4-tiny.co
v4-tiny.conv.29-part-3', 'yolov4-tiny.conv.29-part-4'], '172.31.22.57': ['yolov4-tiny.conv.29-part-6'], '172
```

  - 程式碼最下方也會印出上傳的詳細資訊，方便我們檢查

- ◆
```
[ec2-user@ip-172-31-20-98 ~]$ ls files/
yolov4-tiny.conv.29-part-3  yolov4-tiny.conv.29-part-4
```

- ◆
```
[ec2-user@ip-172-31-21-228 ~]$ ls files/
yolov4-tiny.conv.29-part-1  yolov4-tiny.conv.29-part-5
```

- ◆
```
[ec2-user@ip-172-31-16-215 ~]$ ls files/
yolov4-tiny.conv.29-part-8  yolov4-tiny.conv.29.pkl
```

- ◆
```
[ec2-user@ip-172-31-22-57 ~]$ ls files/
yolov4-tiny.conv.29-part-6
```

- ◆
```
[ec2-user@ip-172-31-27-76 ~]$ ls files/
yolov4-tiny.conv.29-part-0  yolov4-tiny.conv.29-part-2  yolov4-tiny.conv.29-part-7
```

  - 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

- ```
  [ec2-user@ip-172-31-20-152 ~]$ python3 lb_dn.py yolov4-tiny.conv.29 172.31.20.98
  file_chunk_dict: {'chunk_num': 9}
  Load balance download...
  Hash of yolov4-tiny.conv.29-part-0 is 1884428423
  Downloading file from http://172.31.27.76
  Hash of yolov4-tiny.conv.29-part-1 is 38853279
  Downloading file from http://172.31.21.228
  Hash of yolov4-tiny.conv.29-part-2 is 2271790647
  Downloading file from http://172.31.27.76
  Hash of yolov4-tiny.conv.29-part-3 is 3096631432
  Downloading file from http://172.31.20.98
  Hash of yolov4-tiny.conv.29-part-4 is 3280948191
  Downloading file from http://172.31.20.98
  Hash of yolov4-tiny.conv.29-part-5 is 546591947
  Downloading file from http://172.31.21.228
  Hash of yolov4-tiny.conv.29-part-6 is 3644854584
  Downloading file from http://172.31.22.57
  Hash of yolov4-tiny.conv.29-part-7 is 1975847167
  Downloading file from http://172.31.27.76
  Hash of yolov4-tiny.conv.29-part-8 is 1161406593
  Downloading file from http://172.31.16.215
  Combining file chunks......
  yolov4-tiny.conv.29-part-0 檔案已刪除
  yolov4-tiny.conv.29-part-1 檔案已刪除
  yolov4-tiny.conv.29-part-2 檔案已刪除
  ```
  - 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
- ```
  [ec2-user@ip-172-31-20-152 ~]$ ls lb_download/
  yolov4-tiny.conv.29
  ```
  - 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
  - ```
    [ec2-user@ip-172-31-20-152 lb_download]$ ls -lh yolov4-tiny.conv.29
    -rw-rw-r-- 1 ec2-user ec2-user 19M Apr 29 01:41 yolov4-tiny.conv.29
    ```
    - 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
  - ```
    * * * * * /usr/bin/python3 /home/ec2-user/migrate.py >> /home/ec2-user/migrate_log.txt 2>&1
    * * * * * /usr/bin/python3 /home/ec2-user/replica.py >> /home/ec2-user/replica_log.txt 2>&1
    ```
    - 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
  - ```
    /usr/bin/crontab -u ec2-user /home/ec2-user/cronjob
    ```
    - 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

- ```
  print("Replica init ...")
  file_list = os.listdir(dir_path)
  print("file_list:", file_list)
  ```
  ```

- 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

```
for filename in file_list:

    # get the file hash
    filepath = filename
    slashs = [i for i, c in list(enumerate(filepath)) if c == '/']
    if len(slashs) != 0:
        filename = filename[max(slashs) + 1:]
    h = hash(filename)

    # check if the node is the first node
    client = new_client(my_ip, 5057)
    node = client.call("find_successor", h)
    node_ip = node[0].decode()

    if node_ip == my_ip: # I'm the first replica node
        set_replica(filename, my_ip) # set replica to other node
        time.sleep(10)
```

- 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

```
def set_replica(filename, my_ip):
    client = new_client(my_ip, 5057)

    # successor 1
    node_s1 = client.call("get_successor", 0)
    node_s1_ip = node_s1[0].decode()

    if node_s1_ip != my_ip:
        if not is_file_exist(filename, node_s1_ip): # 如果沒有 repli
            print("Set replica to:", node_s1_ip, "File:", filename)
            simple_upload(filename, node_s1_ip)

    # successor 2
    node_s2 = client.call("get_successor", 1)
    node_s2_ip = node_s2[0].decode()

    if node_s2_ip != my_ip:
        if not is_file_exist(filename, node_s2_ip):
            print("Set replica to:", node_s2_ip, "File:", filename)
            simple_upload(filename, node_s2_ip)
```

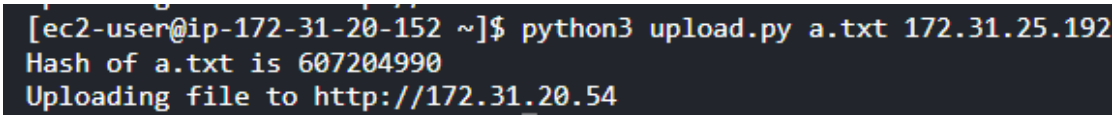
- 持有 file 的第一個 node 要像後面兩個 successor 放置 replica，

  如果已經有檔案就不要重複放置

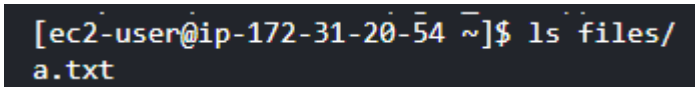
➔ Experiment

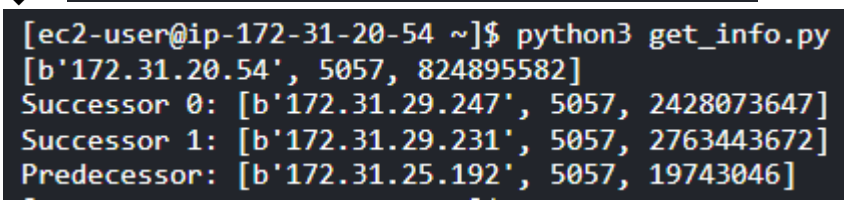
- 為了方便實驗，我們直接模擬發現需要進行 replica 的情況，而不是

  等她每 60 秒再維護一次系統

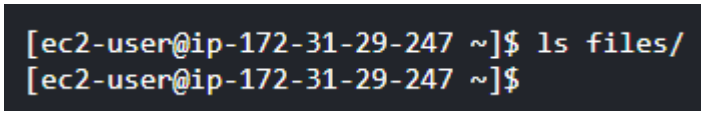
- 

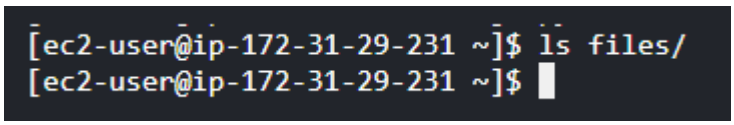
  - 首先，我們將 a.txt 上傳自 chord node

  - 

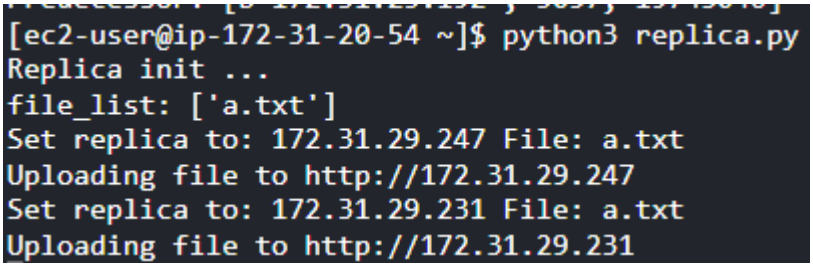
- 

  - 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

- ```
  [ec2-user@ip-172-31-29-247 ~]$ ls files/
  [ec2-user@ip-172-31-29-247 ~]$ ls files/
  a.txt
  ```
- ```
  [ec2-user@ip-172-31-29-231 ~]$ ls files/
  [ec2-user@ip-172-31-29-231 ~]$ ls files/
  a.txt
  ```
  - 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?

- ```python
  h = hash(filename)

  # check if the node is the first node
  client = new_client(my_ip, 5057)
  node = client.call("find_successor", h)
  node_ip = node[0].decode()

  if node_ip == my_ip: # I'm the first replica node
      set_replica(filename, my_ip) # set replica to other node
      time.sleep(10)
  ```
  - 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
  ```

```python
def set_replica(filename, my_ip):
    client = new_client(my_ip, 5057)

    # successor 1
    node_s1 = client.call("get_successor", 0)
    node_s1_ip = node_s1[0].decode()

    if node_s1_ip != my_ip:
        if not is_file_exist(filename, node_s1_ip): # 如果沒有 repli
            print("Set replica to:", node_s1_ip, "File:", filename)
            simple_upload(filename, node_s1_ip)

    # successor 2
    node_s2 = client.call("get_successor", 1)
    node_s2_ip = node_s2[0].decode()

    if node_s2_ip != my_ip:
        if not is_file_exist(filename, node_s2_ip):
            print("Set replica to:", node_s2_ip, "File:", filename)
            simple_upload(filename, node_s2_ip)
```

- 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