設計以 Linux 容器為基礎之金融程式交易平台

[1]林俊宏 [2]蔡政修
國立中山大學資訊工程學系
[1]lin@cse.nsysu.edu.tw, [2]caijhengsiou@gmail.com

摘要

雲端開發平台之建置經常需要花費大量的時間安裝與設定環境，當雲端服務系統需要更新或者遷移時所需付出的成本將隨著系統擴增而急速上升。因此，我們希望藉由將預先安裝好的金融程式交易服務平台包裝成 Turnkey 的型式，用戶只需要簡易安裝 Turnkey 的執行環境－Docker，並將 Turnkey 掛載於其上，便能快速建置與部署專屬的雲端金融服務平台，如此不論用戶想要更新或者遷移整個服務平台，只需要將更新後的 Turnkey 重新掛載或者遷移至新的雲端環境，不再需要重新安裝與設定所有機器，有效的減少雲端開發平台重新建置時所必須付出的龐大成本。
Keywords: 雲端開發平台、程式交易、Turnkey、Docker

Abstract

Cloud development platform development with traditional virtual machine has always been too complex, too expensive, and too slow, therefore we hope to reduce costs by replacing traditional virtual machine with docker container. We build the cloud development platform which can run R language program on the docker container and make it to a turnkey. The user can update, migrate and run the turn key quickly without reinstalling or resetting the host. It will reduce the cost of re-building the cloud development platform effectively.
In the cloud service platform, the computing resource share very precious. We design a dynamic provisioning mechanism for them so as to avoid redundant compute nodes. We adjust the number of compute nodes by timeout mechanism. When the running job is timeout, it means the system is too busy, and then we need to provide more compute nodes. When the utilization of compute nodes are too low, we will remove some compute nodes to reduce waste of resources.

In order to make our platform services have more portability, we implement our platform service on the web server. The users can using the platform service including the IDE (Rstudio) and the function of deployment on the PC, netbook or any mobile device(e.g. smart phone, tablet PC). The user will not be limited by hardware device and it will increase the portability of system effectively.

1. 前言

海量資料的儲存與運算需要消耗大量的硬體資源，透過雲端平台服務業者所提供的資源，不論是政府、企業、學校或者是個人，皆能夠輕易的在服務平台之上開發與部署運算策略。然而，不論是知名雲端服務公司 Google 所開發的 GAE(Google App Engine)[1]，或者是軟體大廠 Microsoft 所提供的 Azure[2]雲端服務平台，皆無法保證安全問題，也因此造成了許多企業仍選擇自行承擔更新與維護系統之成本，建立私有的雲端平台。

為了降低建置成本，本系統藉由將開發完成之雲端金融服務平台封裝成 Turnkey 的型式，並利用 Docker 強大的部署能力，將 Turnkey 掛載於其上，如此只需要將更新後的 Turnkey 重新掛載於平台之上，便能快速形成新的雲端平台環境。

本系統基於提供金融公司良好的程式交易策略開發與計算平台，透過網頁化的服務模式，用戶能夠在任何有網路之處使用雲端平台所提供的服務，並利用動態調整運算節點數量的方式，減少雲端平台運算資源的無端消耗。

2. Related Work

A. Linux Container

Linux Container 是一種不需要透過 Hipervisor 模擬硬體裝置，而是直接使用實體機器資源的虛擬化技術，透過 Linux Container，能夠在不提供額外 kernel instance 的情形之下創建出近似於標準 Linux 的系統環境。

Linux Container 允許使用者在單一主機之作業系統上產生多個獨立的 sendboxes，並透過 cgruop [3] 與 AppArmor 等系統管理工具進行管理、控制以及隔離不同虛擬機器與實體機器之間的資源運用。

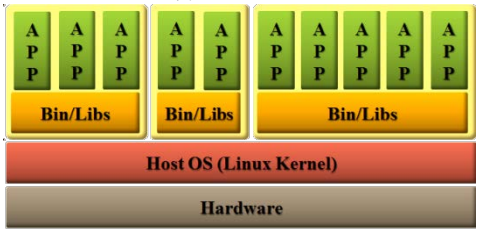

圖 1:Linux Container Architecture

B. Docker

Docker 能夠在 Linux 系統上迅速創建 container

[4]。透過封裝 payload 的方式，Docker 可以在不同的 virtual machines、bare-metal 伺服器、OpenStack clusters 之上運行完全相同的 container，不需要重新建立 container 之中的開發環境與應用程式，有利於雲端開發平台之建置與轉移。

Docker 不僅使得 container 的建立、修改、發布、搜尋以及執行變的更為容易，也讓開發人員可以選擇手動建立或者透過 Dockerfile 自動產生的方式建置其所需的 containers。除此之外，研發人員也可以透過 Docker Commit Function 將修改過的 container 儲存成一個新的 image，並藉由將新產生的 image push 至 Central Registry 或者是儲存成 tarball 檔等方式，使得新產生的 image 能夠分享與轉移至其他實體機器，Figure 2 為 Docker 基本的運作原理。

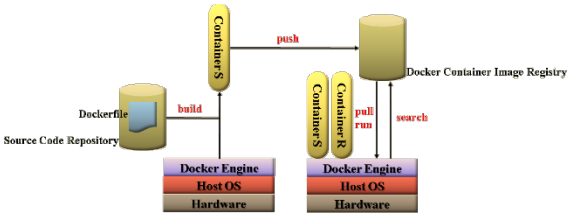

圖 2: Docker Basic Functions

C. R 與 Rstudio

1) R

R 是一種 open source 統計語言，主要用於統計分析與繪圖。R 提供包含線性模組、非線性模組、統計測試、time-series 分析、分類以及巨集等多種統計技術[5]，並透過其繪圖能力將統計結果以圖表方式呈現，讓使用者易於分析統計結果。相對其他同類語言，R 的特色在於:

1. 有效的資料處理與保存機制
2. 針對陣列與矩陣運算提供一套計算工具
3. 提供一系列連貫且完整的資料分析工具
4. 繪圖工具可以直接對資料進行分析並顯示
5. 包含 conditionals、user-defined recursive function、loops、input 以及 output，是一種相當完善、簡潔且高校能的程式語言

2) Rstudio

RStudio 是一款用於開發 R 程式的整合式開發環境(Integrated Development Environment)，其改良原有的 R 使用介面，輔助 R 程式設計者開發程式與測試。

RStudio 可分為 Desktop 與 Server 兩種版本，其中 Desktop 版的 RStudio 可安裝於 Microsoft Windows、Mac OS X 以及 Linux 等作業系統之上，提供程式開發人員在 local 端開發自己的程式；Server 版的 RStudio 則安裝並執行於遠端的 Linux server 之上，使用者透過瀏覽器連線至遠端的 RStudio server 進行專案開發與測試。RStudio 所提供的功能包含:

1. Syntax highlighting 編輯器
2. Code completion 編輯器
3. 直接在編輯器上執行與測試程式，執行方式包含 Line、Seclition(from beginning to line、frome line to end、function definition、all)、File、etc.
4. 自訂工作環境(console、source、plots、workspace、history、etc.)
5. 上傳及下載檔案

3. System Overview

本系統主要由五個 contiainers 與一個 storage 所構成:

- ☐ 部屬節點(deployment node):
  - 提供 web 介面供使用者操作
- ☐ 計算節點(compute node):
  - 運算
  - 新增與刪減運算節點
  - 偵測並自動安裝尚未安裝之 R 套件
- ☐ 中央控制節點(Management node):
  - 紀錄運算資源
  - 分配運算資源
- ☐ 開發節點(development node):
  - 提供網頁開發環境-Rstudio
- ☐ 資料庫節點(mariadb node):
  - 儲存使用者資訊
  - 儲存應用程式資訊
- ☐ 儲存節點(storage):
  - 儲存程式以及投資策略

Deployment 透過 Nginx 與 php-fpm 的組合建立 web 伺服器提供使用者網頁上傳與部署介面，使得使用者能夠在任何有網路的地方使用此環境；Compute 提供 R 程式的運算環境，並透過郵件伺服器 postfix 發送運算完成訊息，讓使用者能夠即時收到通知；Management 則提供節點管理的機制，透過 Redis[6]資料庫紀錄可用資源以及資源使用情形，並負責分配運算資源給部署節點；Development 包含 RStudio 套件，提供使用者網路開發投資策略之環境；最後，Database 提供 MariaDB[7]資料庫，讓系統能夠記錄使用者與投資策略等相關資訊。

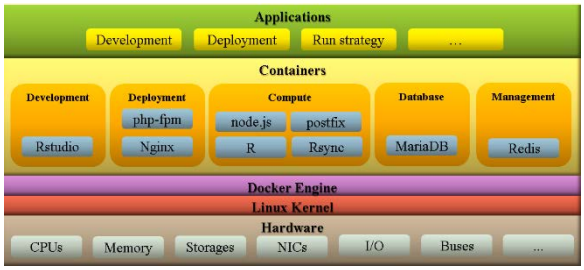

圖 3: System Architecture

本系統主要由中央控制節點分配運算資源，部署節點則在獲取中央控制節點所配置的資源之後，將工作發送至各運算節點計算，並將以網頁方式呈現運算的結果；而運算節點負責資料的運算處理與分析和提供運算節點數量的調節機制，流程如 Figure 4 所示。

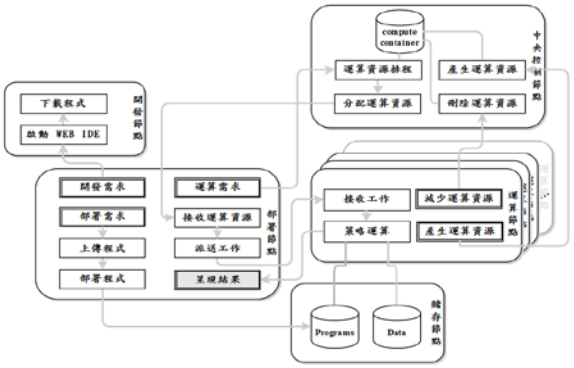

圖 4: System Work Flow

4. System Implementation

A. Deployment node

Deployment node 為雲端服務平台與使用者溝通的主要媒介，分別提供程式開發、部署程式與執行策略三個模組，當 Deployment node 接收到要求時，將會針對不同需求發送訊息到相對應的模組，使得每一位使用者的需求能夠得到相對應的處理與回應，Deployment node 運作流程如 Figure 5 所示。

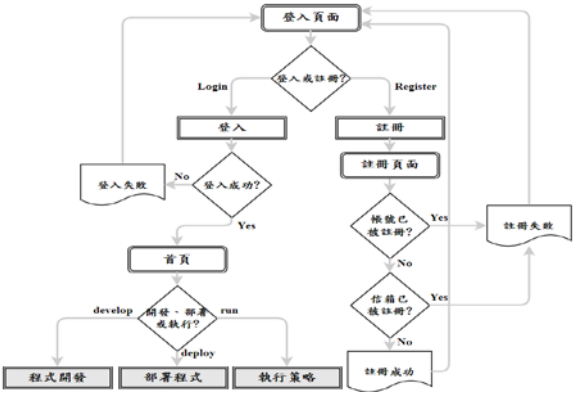

圖 5: 使用者操作概念示意圖

web 伺服採用輕量化的 Nginx[8]。Nginx 是一個高效能的 HTTP 伺服器， Nginx 具有占用記憶體少以及穩定性高等優勢[9]。本系統以 php-fpm 套件取代傳統運用於 Apache 之上的 mod_php，Nginx 透過 fastcgi_pass 將 php 請求交由 php-fpm 套件處理，而其本身只負責處理靜態請求，效能方面 Nginx 與 php-fpm 的組合優於 Apache 與 mod_php。

使用者資訊儲存方面，為避免相同帳號被大量註冊，同一組帳號與電子郵件信箱只允許使用者註冊一次。

1) 程式開發

本系統在程式開發模組中架設伺服器版本的 R 語言整合式開發環境 Rstudio，在於 Rstudio 登入方面，為了避免使用者需要二次登入所造成的不便利性，藉由將加密過後的帳號密碼傳送至開發節點，使得使用者不需要再次輸入帳號密碼，即可自動登入開發系統。除此之外，我們採用了 NIS 帳號共享技術，在使用者註冊帳號的同時，系統除了將使用者資訊儲存到系統資料庫之外，也會向 NIS 伺服器提出註冊申請，避免無法登入的情形。

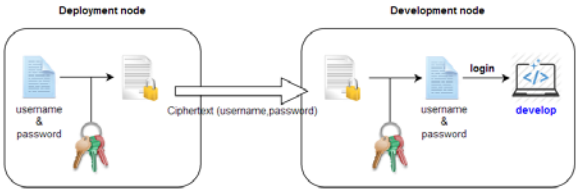

圖 6: 開發節點與部署節點帳號傳遞圖

2) 部署程式

透過 XMLHttpRequest 物件將投資策略上傳至雲端平台伺服器，並將策略部署於儲存節點之中，再透過 NFS 技術將投資策略與所有運算節點共享，在收到使用者的執行需求時能夠快速執行。

本系統依據每位使用者的帳號進行分類，各使用者只能瀏覽自己的投資策略，以避免互相竊取彼此策略的情形發生。此外，系統將投資策略的相關資訊寫入資料庫，以利於資料取得的方便性。

3) 執行策略

中央控制節點在接收到執行策略模組所提出的資源需求時，其會從已經註冊完成的運算節點列表中取出可以運用的計算資源，並將相關資訊回傳給執行策略模組。本系統採用 RR (Round-robin)排程演算法來分配運算節點，以達到負載平衡。

B. Compute node

Compute node 為主要的運算單元，其接收使用者的策略運算需求，提供 R 語言的執行環境，並在接收到運算需求時執行使用者的投資策略，最後回傳運算結果以及發送 mail 通知使用者。

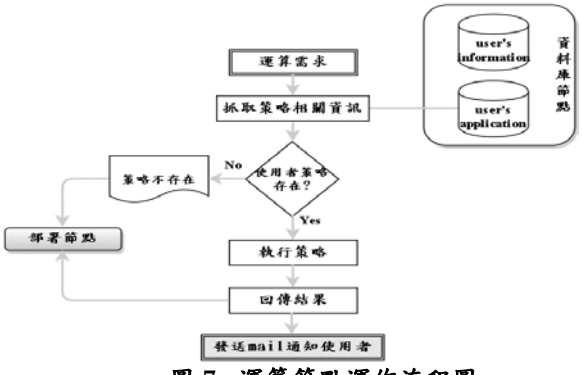

圖 7: 運算節點運作流程圖

表 1:
Compute-Node-Disposition-Contorl-Algorithm

```
Function Compute-Node-Disposition-Control-Algorithm
Input: N, N_cur_job, N_containers, γ, bomb_flag
Output: Retain or remove container
For each Job_finished
    If ((N_cur_job <= N * γ) and (bomb_flag == 0))
        Set bomb flag to 1 and Sleep T_countdown
        If (N_containers <= N_container_lower_bound)
            Retain container
        Else
            If (N_cur_job > 0)
                Wait unfinished jobs are completed
            Else
                Remove container
            End If
        End If
    Else If ((N_cur_job > N * γ) and (bomb_flag == 1))
        Set bomb flag to 0 and kill time bomb
    End If
End For
```

Compute-Node-Disposition-Control-Algorithm 主要的目的在於動態調整計算節點數量。Compute node 在每一個運行於其上的工作完成時皆會自行檢查目前在自身節點中運行的工作情形，並評估是否保留或者是關閉 Compute node，當目前在節點中運行的工作數量小於節點最大可同時運行數量 N 乘上 γ (0.1 <γ< 0.3)時，則評估為節點使用率偏低，適合關閉。當一個 Compute node 被評估為關閉對整體系統效益較大時，我們便啟動一個倒數計時器，如果在 Tcount down 秒內有新的工作被配置到該運算節點，工作數量若超過 N 乘上 γ，則停止並刪除此倒數計時器；否則，在 Tcountdown 秒之後，Compute node 會通知中央管理節點，將運算節點資訊從 Compute node list 中移除，並在該節點中的所有工作執行完畢後，關閉 Compute node。

C. Security

本系統為了避免投資策略在傳輸過程中被竊取，在投資策略上傳至雲端服務平台之前，系統會先要求使用者賦予即將上傳之投資策略一組認證碼，並利用 AES key 加密認證碼加密過的投資策略，以及利用 RSA key 加密 AES key，再將加密過後的 AES key 與投資策略傳送至雲端服務平台之中，提升系統安全性。當使用者想要執行其投資策略時，系統會要求使用者輸入相應認證碼，當輸入正確的認證碼時，該策略才會正常的被執行。

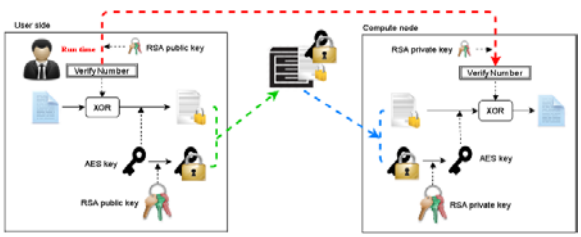

圖 8: 策略加密流程圖

D. 工作排成機制

為避免在資源不足時，中央控制節點仍將工作分配到運算節點，提供工作排程機制[10]，以避免運算主機因無法負荷而丟棄部分工作的情形發生。

為避免客戶無法在第一時間獲取重要資訊，本系統藉由各運算節點自行監控系統資源狀況的方式，動態調整節點中可運行的工作數量，當運算節點目前工作量超過最大可同時運行的工作數量 N 時，運算節點會送出訊息告知中央控制節點目前運算資源已經滿載，以避免工作繼續往該運算節點配送。相反的，當運算節點工作數量小於最大可同時運行的工作數量 N 乘上 β (0.1 <β< 0.3)時，則表示運算節點已經釋放足夠的資源以接收新的工作。

在於工作配置方面，中央控制節點採取傳統的 RR(Round Robin)機制，並以 FCFS(First-Come, First-Served)方式處理大量的工作需求。

本系統單一運算節點可接受的工作數量調整機制參考 TCP[11] congestion windows 控制封包傳輸量之方式[12]，系統藉由工作實際的執行時間觀察系統目前的負荷狀況調整系統可執行的工作數量。為了避免短時間內工作數量過於頻繁的被更改，本系統設計了一個緩衝時間Δ，唯有在前一次更新時間與目前時間相差超過Δ時，才會進行工作數量的調整。詳細工作排程演算法如 Table 2 所示。

表 2: Job-Scheduler-Algorithm

```
Function Job-Scheduler-Algorithm
Input: Job_new, N, N_cur_job ,  Job_life_time,  α ,  β , busy_flag
Output: The system is free or busy
For each Job_new
    If (N_cur_job <= N)
      Increase N_cur_job, set Job_start_t to current time and start Job_new
      When Job_new finishs
          If (Job_start_t > Job_update_N_t + Δ)
              If (Job_run_time > Job_life_time)
                  N = N *  α
              Else
                  N = N + 1
              End If
              Decrease N_cur_job
          End If
          If
              If ((busy_flag == 1) and (N_cur_job < N *  β ))
                  Set busy flag to 0 and add this compute node to compute node list
            on manager again
              End If
          End If
    Else
        If (busy_flag==0)
      Set busy_flag to 1 and remove this compute node from compute node list on
      manager temporarily
        End If
        Add Job_new to wating queue
    End If
```

E. 自動檢查並安裝 R 套件

R 語言是一種由眾多套件所形成的統計語言，如果套件不曾被安裝，則必須讓其自動安裝該套件，並在安裝成功後開始執行使用者的運算策略。

表 3: 自動偵測與安裝 R 套件之函式

```
# Check a package and auto install
lib <- function (pkg){
    mirror_location <- "http://cran.csie.ntu.edu.tw/"
    if (!pkg %in% rownames(installed.packages())){
        res <- try(install.packages(pkg,repos=mirror_location))
        if (inherits(res, "try-error")){
            q(status=4)
        }
    }
    suppressMessages(library(pkg,character.only=TRUE))
}
```

本系統提供一個 lib 函式(如 Table 3 所示)，該函式取代傳統的 library 函式，進行套件檢查、安裝並載入相關套件，在本系統之中，使用者必須透過此函式載入欲使用之套件，而非傳統的 library 函式。函式呼叫方式如下:

```
lib ( PACKAGE_NAME )        # PACKAGE_NAME: 欲載入的 package 名稱
```

5. Simulated Result

A. 實驗一

使用四種不同的工作總數，平均每一個工作執行時間約 2.0~2.5 秒，找出運算節點最佳的運行工作數量初始值 N。

1) 測試內容如下:

–工作總數量 :1000, 2000, 3000, 4000 (個)

–N : 3, 5, 10, 20, 50, 100, 200 (個)

2) 預期輸出結果:

–工作完成總時間

3) 實驗結果:

由 Figure 9 可知，本系統在測試主機上最佳的 N 值為 5，當 N 值為 5 時，執行工作的總時間最少，也因此我們將系統的 N 初始值設定為 5。除此之外，我們從 Figure 9 中發現，隨著 N 值增加，CPU 的平均使用率也會隨之上升，當 N 值達到 10 時，CPU 的平均使用率已經接近於 100%，此時如果再增加 N 的數量，工作完成總時間將隨之增加；然而，N 值也並非越小越好，當 N 值過小時，因 CPU 資源沒有被充分運用，反而造成工作執行的總時間不減反增的現象。

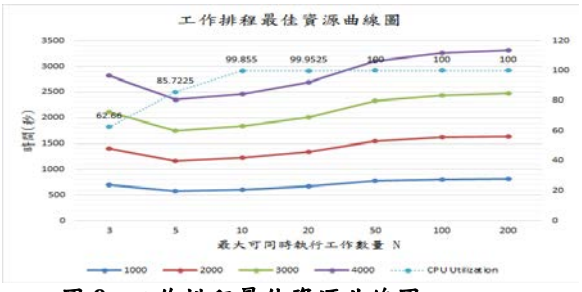

圖 9: 工作排程最佳資源曲線圖

B. 實驗二

本實驗主要從系統管理者的角度來觀察系統效能變化，主要目的是在尋找系統中最佳的 α 值與 job life time 數值，為了減少實驗複雜度，實驗以 2500 個執行時間接近 2.0~2.5 秒的工作為測試對象，並觀察不同 α 值與 job life time 之間的系統效能變化，每次實驗皆以 20 分鐘為標準，測試 20 分鐘內系統能夠順利完成的工作數量。

1) 本實驗的測試內容如下:

– α : 0.5, 0.6, 0.7, 0.8, 0.9

– job life time: 3, 5, 10, 15, 20, 30, 60 (秒)

2) 預期輸出結果:

– 20 分鐘內工作完成之數量

3) 實驗結果:

由 Figure 10 我們可以看出，在有了工作排程機制之後，我們 1200 秒內最少能夠完成 1513 個工作，是沒有排程前的 2.5 倍，甚至於最多能夠完成 3 倍以上的工作量(約為 2012 個工作)，由此可知，工作排成機制對於提升系統效能有了顯著的功效。

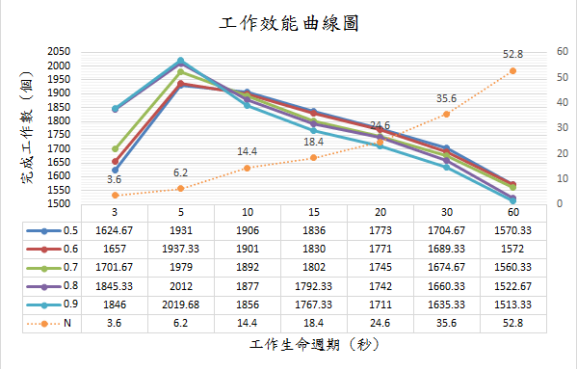

圖 10:工作效能曲線圖

除此之外，由 Figure 10 可知，job life time 的大小將會影響系統的 QoS，job life time 設定的越大，系統所能同時服務的工作數量越多，相對的 QoS 也就越小，例如在本次實驗中，當 job life time 超過單一工作完成時間兩倍時，因為可同時服務的工作數增加，將造成整體系統的效能下降(此論點已由實驗一得證)；然而，job life time 也並非設定的越小越好，當 job life time 越接近單一工作完成時間時，timeout 將頻繁發生，嚴重影響系統 throutput。我們由實驗二的結果得知，當 job life time 設定為單一工作完成時間的兩倍且 α 值設定為 0.9 時，系統能夠獲得最佳的效能，因此本系統將排程中的 α 值設定為 0.9，且 job life time 的初始值設定為 5 秒鐘。

C. 實驗三

本實驗從使用者的角度觀察系統最佳的 α 值與 job life time 數值，實驗以 1000 個執行時間接近 2.0~2.5 秒的工作為測試對象，並觀察不同 α 值與 job life time 之間的使用者平均等待時間的變化。

1) 本實驗的測試內容如下:

– α : 0.5, 0.6, 0.7, 0.8, 0.9

– job life time: 3, 5, 10, 15, 20, 30, 60 (秒)

2) 預期輸出結果:

– 使用者平均等待時間

3) 實驗結果:

本實驗針對 1000 個同時送出的運算需求進行測試，找出最差的使用者平均等待時間。由 Figure 11 得知，當系統的 job life time 在設定為單一工作完成時間的兩倍時，使用者的平均等待時間最小，此點與實驗二的結果不謀而合，我們推測原因在於 token 數的變化情形，當 job life time 設定為單一工作執行時間的兩倍時，系統的平均 token 數會接近於系統

的最佳 token 數 5，也因此獲得最佳的系統效能。另外，從 Figure 11 中我們也發現，當 job life time 設定超過單一工作執行時間的兩倍時，使用者平均等待時間會有大幅度的變化，主要原因是動態 token 數的變化，因為 job life time 設定的越大，token 的變化量也相對越大，造成使用者平均等待時間會有大幅度的變化。

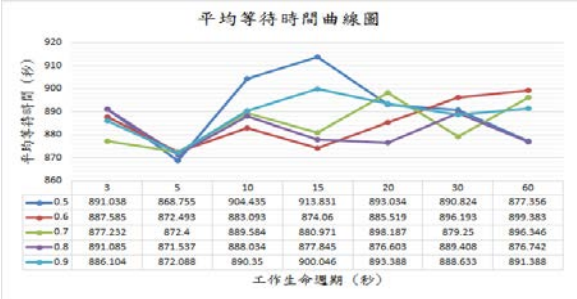

圖 11: 平均等待時間曲線圖

由 Figure 11 得知，在實驗三中使用者最佳的平均等待時間約略為 870 秒，如果系統採取依序分配工作的策略，在每一個工作完成之後才分派另一個工作進入執行，則第一個工作等待時間為 0 秒，第二個工作等待時間為 2 秒，依此類推，第 n 個工作等待時間為 2*(n-1)秒，則平均等待時間為 (2*((n*(n-1))/2))/n = (n-1) = 999 秒，遠大於系統的平均等待時間，由此可知本系統的架構有助於提升系統的效能。

6. Conclusion

本論文的目的在快速地建立自己專屬的金融運算服務環境，而不需要經過繁瑣的安裝程序。本系統主要由中央控制節點、部署節點以及數個運算節點所組成，透過 Docker 強大且快速的部署能力，使用者只需掛載事先建置好的服務平台映像檔於系統之上和簡單的網路參數設定，便能夠在極短的時間內建置好屬於自己的運算環境。

本系統在使用者上傳投資策略之前會預先將投資策略進行加密，再進行傳送，以確保策略的機密性。本系統為了維護服務品質，在各運算節點中皆提供一個工作排程機制，依據實際工作完成的時間，適時地調整節點中運行的工作數量，藉此來提升系統整體效能。由論文實驗過程我們發現，系統同時運行的工作數量並非越多越好，每部機器會因為不同的 CPU 數量、記憶體大小等因素而有最佳的同時運行之工作數量，並從實驗結果得知，在於 throughput 的表現上。

參考文獻

[1] “Google APP Engine – Google Developers”, https://developers.google.com/appengine/

[2] “Azure: Microsoft’s Cloud Platform”, http://azure.microsoft.com/en-us/

[3] “LXC”,https://help.ubuntu.com/lts/serverguide/lxc.html#lxc-cgroups

[4] “Docker”, https://docs.docker.com/

[5] “ R”,http://www.r-project.org/

[6] “Redis”, http://redis.io/

[7] “MariaDB”, https://mariadb.org/

[8] “Nginx”,http://nginx.com/

[9] “apache / lighttpd / nginx 三大開放 WEB 伺服器比較”, http://mike7120.blogspot.tw/2011/02/apache-lighttpd-nginx-web.html

[10] Michael L. Pinedo, “Scheduling: theory, algorithms, and systems Forth Edition”, 2012.

[11] W.Richard Stevens, “TCP/IP Illustrated, Volume 1: The Protocols”, March, 2011.

[12] W. Stevens, “TCP Slow Start, Congestion Avoidance, Fast Retransmit, and Fast Recovery Algorithms”, January 1997.

[13] 姜林杰祐,“程式交易：觀念、方法、技術與解決方案”, 新陸書局出版, 2009.

[14] H. Shafi, P.J. Bohrer and J. Phelan, Design and validation of a performance and power simulator for PowerPC systems. IBM Journal of Research and Development,47 5–6 (2003), pp. 641–652

[15] S. Patidar, D. Rane and P. Jain, “Challenges of Software Development on Cloud Platform,” World Congress on Information and Communication Technologies (WICT), Mumbai, 11-14 December 2011, pp. 1009-1013.

[16] Shuai Zhang; Shufen Zhang; Xuebin Chen. Cloud Computing Research and Development Trend. Future Networks, 2010. ICFN '10. JAN 2010. Pages 93-97

[17] W. Liu, “Research on Cloud Computing Security Problem and Strategy,”2nd International Conference on Consumer Electronics, Communications and Networks, YiChang, 21-23 April 2012, pp. 1216-1219.

[18] C. Tsai, U. Lin, A. Chang and C. Chen, “Information Security Issue of Enterprises Adopting the Application of Cloud Computing,” 6th International Conference on Networked Computing and Advanced Information Management (NCM), Seoul, 16-18 August 2010, pp. 645-649.

[19] Liu, Wentao. "Research on cloud computing security problem and strategy." Consumer Electronics, Communications and Networks (CECNet), 2012 2nd International Conference on. IEEE, 2012.