以改良式 Min-Min 排程演算法 提供動態階層式雲端運算網路拓樸之負載平衡

嚴國慶 [1] 王淑卿 [2]* 王順生 [3] 曾莉雅 [4] 陳慶維 [5]
{[1]kqyan; [2]*scwang; [3]sswang; [5]s9814612} @cyut.edu.tw
[4]d918305@oz.nthu.edu.tw
*: 聯絡人
朝陽科技大學

摘要

由於現今電腦網路頻寬及硬體設備相關技術的持續發展演進，使得網際網路的相關應用更加蓬勃發展。架構在網際網路的雲端運算為一新興的分散式系統之概念，其使用較低效能的主機來達到高可靠性及高效率的運算能力。由於雲端運算概念的興起，使得網際網路服務的應用，已透過雲端運算擴大其服務層面的廣度及服務內容的深度。但由於雲端運算主要是透過分散的服務節點(電腦或資源)協力合作完成一個大型的工作，也就是將工作分割成若干個子工作，再將這些子工作分配給這些分散的節點進行服務。因此，如何將工作有效的切割並分配到每一個服務節點上，才不會造成某些服務節點負擔過重，而某些服務節點卻是閒置的情況，則是一個值得探討的議題。本研究在一個動態階層式雲端運算網路拓樸架構下，提出混合式排程演算法，結合OLB(Opportunistic Load Balancing)排程演算法與本研究所提出之 LBMM(Load Balance Min-Min)排程演算法進行工作的配置，使得每個需要執行的工作都能被分配到適當的資源並有效的改善每個服務節點負擔與降低資源的浪費，提供動態階層式雲端運算網路拓樸之負載平衡。

關鍵詞： 分散式系統、雲端運算、排程演算法、負載平衡

Abstract

Network bandwidth and hardware technology are developing rapidly, resulting in the vigorous development of the Internet. A new concept, cloud computing, uses low-power hosts to achieve high reliability. Cloud computing increases the number of user applications on the Internet. The cloud computing, an Internet-based development in which dynamically scalable and often virtualized resources are provided as a service over the Internet has become a significant issue. In a cloud computing environment, the connected topology is not very significant that uses low-power hosts to achieve high reliability and ensure the ability to be better. Research must be focused on how distributed systems can provide better reliability and fluency. The cloud computing refers to a class of systems and applications that employ distributed resources to perform a function in a decentralized manner. Cloud computing is to utilize the computing resources (service nodes) on the network to facilitate the execution of complicated tasks that require large-scale computing. Thus, the selecting nodes for executing a task in the cloud computing must be considered, and to exploit the effectiveness of the resources, they have to be properly selected according to the properties of the task. This study proposed a hybrid scheduling to maintain the execution performance and the load balancing of the system.

Keywords: Distributed System, Cloud Computing, Scheduling, Load Balancing

1. 前言

由於現今電腦網路頻寬及硬體設備相關技術的持續發展演進，使得網際網路的相關應用更加蓬勃發展。架構在網際網路的雲端運算為一新興的分散式系統之概念，其使用較低效能的主機來達到高可靠性及高效率的運算能力[3,4]。雲端運算目前主要應用於商業用途上，在雲端運算的系統內，服務的提供主要是經由電腦間一同合作來執行[6]。然而，由於

現今網路設備的蓬勃發展，以及網路頻寬的日漸增大，使得各種通信和計算技術得以整合，如電信、多媒體通訊、訊息傳輸技術、模擬建構等[8]。透過網路技術的快速發展，各種過去應用於個人終端設備上之相關程序，如：多媒體影音的欣賞等，已能出現於各種網路平台上。在雲端運算的環境中，使用者可以從中得到系統所提供之許多便利的應用程序[9]。

2007 年末 Google 在網路的相關服務，運用雲端運算概念，提供用戶在最短時間內獲得服務的需求[18]。雲端運算是以使用者為導向的服務，其主要的運作模式是使用者發出要求服務的訊息後，由位於多處的電腦系統來處理其要求之服務[15]。所以，在整個處理的過程中，任何一個使用者發送服務要求的訊息後，並不知道整個訊息處理的流程及經過，而在傳統網際網路的圖形表達方式通常都是以一朵類似雲朵的形狀來呈現，因此，雲端運算之“雲”的意思即是指其意，圖 1 所示就是雲端運算的簡略示意圖。

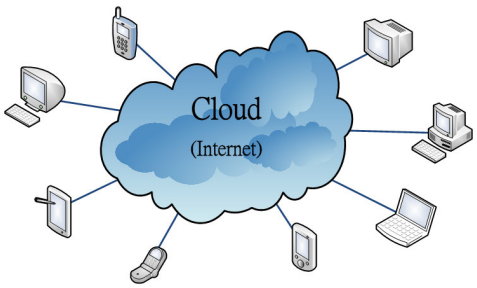

圖 1、雲端運算概念意示圖

除了 Google 提供雲端運算的相關服務外，微軟與 Yahoo 在分析雲端運算的發展趨勢後，也跟進將雲端運算應用在網頁服務上。隨著雲端運算的熱潮蔓延，國內外科技企業公司陸續將雲端運算應用於服務或技術上[20]。如趨勢科技公司將雲端運算概念導入，應用於該公司的防病毒軟體上，透過雲端運算將防病毒軟體作用在網際網路上，開發出雲端保全應用程式，包括：軟體更新、病毒碼更新、即時防護及待解決的病毒問題等，皆可由遠端的伺服器節點處理，以提升防毒軟體的即時性與安全性。商業管理服務著名的 Salesforce.com 科技公司將雲端運算應用於 ERP 與 CRM，提供企業透過此公司的客戶管理服務，更迅速回應客戶的服務、解決客戶的問題以及即時與客戶連線[19]。Salesforce.com 以更低的成本提供企業更快的服務，包括：新的產品支援、服務或流程、以及即時進行更新。亦即，Salesforce.com 將企業的需求透過雲端提供服務，因此企業不再需要額外的成本進行硬體或軟體的管理。但在 2009 年的 2 月及 5 月發生 Gmail 當機事件，甚至 9 月份發生兩次，引發全球用戶的反彈。當機事件背後所隱藏的負載不平衡之問題，是本研究探討雲端運算負載平衡的主要動機。

由於雲端運算是透過參與者的計算能力和頻寬來進行應用程式之執行，而不是把所有需要執行的工作都聚集在較少的幾台伺服器上[1,5,14]。從計算模式上來說，雲端運算打破了傳統的主從式模式，使用雲端運算可避免傳統主從式架構中，因只有一個集中管理伺服器，而導致伺服器負荷過重的問題。除此之外，亦能有效改善在傳統主從式架構伺服器上之計算能力與儲存能力的要求。同時因為相關的服務資源是分佈在每個服務節點上，所以可以運用每個服務節點的一些資源協同合作完成一個大的工作[1]。

但是如何運用雲端運算的優點，讓需要運算的工作能在最短的時間內分配到最適當的資源則是一重要議題。本研究運用雲端運算的特性，將使用者的需求適當的分配給服務節點，以期使系統中每一個服務節點能達到較佳的負載平衡，除此之外更將提供每一位使用者快速的服務品質。因此，本研究在一個動態階層式雲端運算網路拓樸架構下，提出混合式排程演算法，結合 OLB(Opportunistic Load Balancing)排程演算法與本研究所提出之 LBMM(Load Balance Min-Min)排程演算法進行工作的配置，藉以提高工作完成效率與達到善用服務節點資源之目的。

本文第二節為文獻探討，說明雲端運算的相關應用與排程演算法；第三節將說明動態階層式雲端運算網路拓樸的架構；第四節將介紹本研究所提出之 LBMM 排程演算法；第五節說明研究的方法與架構；第六節為結論。

2. 文獻探討

在本節中，將分別說明雲端運算的相關應用與排程演算法。

2.1 雲端運算的相關應用

近年來，由於網際網路提供媒體化的服務，越來越多的網站不僅提供文字瀏覽，更進一步提供照片以及影音資訊，使越來越多的網站都需要不斷得提升網站的瀏覽品質，同時也需要不斷擴充實體設備，以提供使用者更多元

化的應用與更流暢的使用環境，如部落格(Blog)與 YouTube 等。

在 2007 年末，由 Google 所提出雲端運算以使用者為導向的概念，掀起網路供應商在網路服務領域上另一種新格局的出現[9]。雲端運算是個概念而不是技術，是由分散式系統衍生出的概念，透過網際網路將大量的運算處理程序分解成無數個小的子程序，再交由多部伺服器組成的巨大系統，經由搜尋、運算分析，將處理的結果回傳給用戶端。而由 2008 年以來，越來越多科技公司也紛紛宣布研發有關雲端運算概念的相關技術及相關應用，如 Yahoo、微軟(Microsoft)、亞馬遜(Amazon)網路書店等，使得雲端運算的概念成為網際網路一個新的應用趨勢。

雲端運算利用 “As a service” 的網路技術，以大量的運算或儲存資源(服務節點)提供使用者多元化的服務。由於，雲端運算網路平台的供應商，使用虛擬化的資訊技術基礎建置和服務，以提供不同的使用者多元化之應用服務需求。除此之外，使用者也具有自行客製化個人服務的能力。因此，不管是使用者所擁有的網路儲存總容量或所需的軟體運算能力，都能透過雲端運算網路平台的供應商獲得，因此可以降低使用者建置硬體設備所需的成本。

在雲端運算的相關應用中，Google 透過雲端運算環境，不斷增大免費的電子郵件信箱(e-Mail)容量及免費的網路空間。除此之外，Google 也推出網路平台上之應用功能，如 Google 文件，其可以提供線上編輯文件的功能，同時也能將所編輯的文件存放於帳戶內的空間；Google Earth 則能觀看地球任何一個地方的衛星雲圖、3D 影像、地圖等；而今年在 Google Labs 更推出一個新的搜尋技術，名為語音搜尋技術(Gaudi：Google Audio Indexing)，可以透過關鍵字搜尋包含關鍵字的影片或音樂檔案[18]。

亞馬遜則經由 Amazon Web Service(AWS)的雲端運算環境提供了許多應用程序，包括：Cloud Computing(EC2)服務，Sample Storage Service（S3）和 CloudFront[16,17]。有了 AWS 的服務，使用者可以要求需要的計算能力、資料儲存的空間和其他服務，並取得一具彈性的資訊技術基礎設施服務的需求[16,17]。在 AWS 平台上，允許使用者能夠租用雲端運算服務的基礎設施和應用平台。亞馬遜則可以依照使用者實際使用的狀態，向使用者收取使用費用[16,17]。

Zeus 科技公司所提出的虛擬化的雲端運算資訊技術基礎建設如圖 2 所示[21]，透過將一大型的服務資源分群成小批量群組處理，以提供其服務效能。而 Zeus 科技公司，所開發的 Zeus Extensible Traffic Manager(ZXTM)，可以讓使用者從任何雲端運算環境上產生、管理和提供高性能的線上服務的軟體。換言之，ZXTM 可以提供平台上的使用者能夠更快、更可靠、更安全、且易於管理的相關服務[21]。ZXTM 軟體可以安裝在任何支援 Linux、Solaris 或 FreeBSD 作業系統的伺服器。此外，ZXTM 軟體因在軟體設計的過程中，內嵌有可提供並協調使用者服務的功能，因此亦提供伺服器得以執行使用者所提出之服務需求的工作分派。因此，透過 ZXTM 軟體除了可以提高雲端運算的服務規模外，更可提供使用者更有效率且更具成本效益的服務，其概念如圖 3 所示。

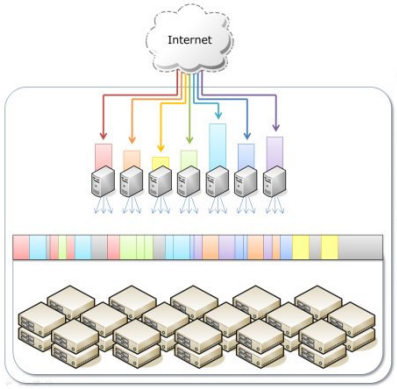

圖 2、虛擬化的雲端運算資訊技術基礎建設

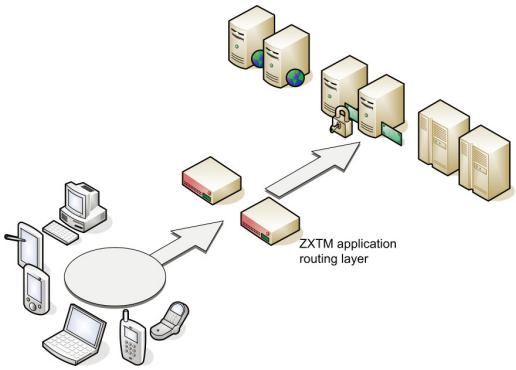

圖 3、ZXTM 高效能負載平衡之示意圖

由於目前在雲端運算的相關應用中，各種網路拓樸各有其優缺點，其中以階層式拓樸是目前最為相關應用所採用[2,13]。主要原因為在階層式拓樸中，其工作可以依階層的分配給下一階層，因此不會產生因只有透過一台伺服

器提供服務，造成服務資源數量有限之問題，且每一個節點只須記錄其上一節點之位置，所以可有效降低記錄節點資料。因此，本研究將在一個動態階層式雲端運算網路拓樸架構下，提出混合式負載平衡排程演算法進行工作的分配，使需要被執行的工作能快速的分配到適當的服務節點上，除有效的改善每個服務節點的工作負載外，還可依據工作的特性來選擇最合適的服務節點，提供各服務節點之負載平衡與執行效率的品質保證。

2.2 排程演算法

在各種相關的研究中，為了提高系統執行的效率及維持系統之負載平衡，學者們大多會運用各種不同的排程演算法以達到善用資源及提升效能的目的。然而不同的排程方法各有其不同的特性，在選擇排程方法以進行應用時，必須考慮到各種排程方法的特性及適用的範圍[5,7,11]。因此，在本節中將探討不同排程演算法之特性並分別說明。

2.2.1 OLB(Opportunistic Load Balancing)

OLB 排程演算法試圖讓每一部電腦都保持忙碌的狀態，因此不考慮各個電腦目前的工作量，而以任意的順序將尚未被執行的工作(Task)分配給目前可以用的電腦進行執行。因此，OLB 排程演算法最大的優點是相當簡單，但卻因為未考慮每個工作的期望執行時間(Expected task execution time)，所以整體而言所將獲得完成時間(Makespan)非常的差。

2.2.2 MET(Minimum Execution Time)

MET 排程演算法則試圖讓每一個工作可以獲得最好的電腦之支援，因此不考慮電腦目前的工作量，而以任意的順序將可以得到最短執行時間的電腦分配給尚未被執行的工作。MET 排程演算法可能導致整個系統中各電腦間負載的不平衡，因此不適用於異質性電腦系統之應用。

2.2.3 MCT (Minimum Completion Time)

MCT 排程演算法將目前具有最小完成時間(Minimum completion time)的電腦以任意的順序分配尚未被執行的工作，但仍可能有部份的工作無法獲得最小的執行時間(Minimum execution time)。

2.2.4 Min-Min

Min-min 排程演算法針對每一個未排程的工作建立最小的完成時間，並將工作指派給可提供最小完成時間的電腦進行處理，因對工作或電腦都取最小的完成時間(Minimum-minimum completion time)，因此稱之為 Min-min 排程演算法。

Min-min 排程演算法的優點是會考慮到所有工作的最小完成時間，但也因為需要考慮到所有工作的最小完成時間而必須花費額外的計算成本。換言之，Min-min 排程演算法的精神便在於總完成時間最小的最佳組合為分派工作的依據[5,11]。

由於 OLB 排程演算法簡單且容易實行，並能使所有的服務節點盡可能地都處於工作狀態，因此本研究將在動態階層式網路拓樸的中間階層使用 OLB 排程演算法，進行工作的分配並將工作切割成若干個子工作。而為了能提供系統中各服務節點工作之負載平衡，本研究將改善 Min-min 排程演算法，期能有效的降低每個服務節點的執行時間。

3. 動態階層式雲端運算網路拓樸

雲端運算讓使用者可以透過網際網路直接連接到雲端運算環境的服務節點，以進行相關應用服務或檔案的共用與交換。另一方面，雲端運算在深度搜索、分佈計算、協同工作等方面也有相當的貢獻[10]。

過去有關分散式網路拓樸的研究架構，包括：星狀式、環狀式、階層式等網路拓樸。然而，現今的網路應用已十分的多元化，這些網路拓樸之研究結果已不完全符合雲端運算的需求。主要原因在於星狀式網路拓樸雖然其製作簡單，但這種架構的應用方式是將所有的資料都集中存放在單一的中央伺服端中，提供給很多的使用者端直接連接，以存取所需要的資料。因此，單一伺服端的負荷可能過高。而環狀式網路拓樸，將多個伺服器連結起來，形成一個環狀拓樸來平衡伺服端的負荷，可提高服務客戶端的數量。但當節點間某個鏈結中斷時，將造成整個網路故障。而階層式網路拓樸透過一個名稱伺服器(Root Name Sever)提供負責驗證的機制，每個下層節點都需要上層節點的驗證許可，因此每個節點只需知道上一節點位址即可，不需記錄網路中每個節點的位址。但當其中一個節點離開或失效時，則子鏈結與父鏈結的關係將會斷線，影響到整個網路的連線[12]。

在本研究中提出之動態階層式雲端運算

網路拓樸架構，所考慮的是在雲端運算環境下，如何讓雲端運算計算之工作，有效的配置資源，使分散在不同位置上擁有不同之記憶體、CPU、時脈與輸入/輸出設備的異質性服務節點(Heterogeneous Service Node)達到真正的負載平衡，而又不會浪費資源。

動態階層式雲端運算網路拓樸主要分為二階層式架構與三階層式架構，其依據系統目前的工作量(即所需執行的工作數目)，來決定工作要如何分配給分散式系統中的每一個服務節點，也就是說，當工作量很小時，只需要二階層式雲端運算網路拓樸的架構就可完成任務，而當工作量很大時，則會採用三階層式雲端運算網路拓樸，說明如下。

3.1 二階層式雲端運算網路拓樸

在二階層式雲端運算網路拓樸上主要是由第一階層的服務管理者(Service Manager)與由多個可利用的服務節點(Service Node)所組成的服務群集(Service Block)之第二階層。當進入系統執行的所有工作量小於 k 時(k 為第二階層之服務群集最大負載上限)，則服務管理者會將工作切割成若干個子工作，並將這些子工作有效的分配到下層服務節點上執行，如圖 4。

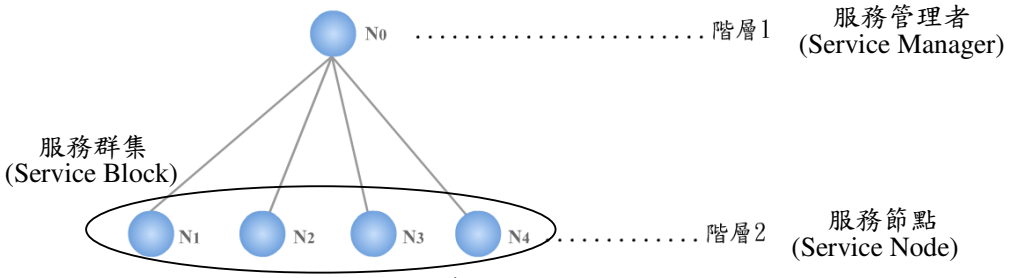

圖 4、二階層式雲端運算網路拓樸

3.2 三階層式雲端運算網路拓樸

三階層式雲端運算網路拓樸主要是由多個二階層式雲端運算網路拓樸所組成，每一個二階層式網路拓樸都是由一個服務管理者與一個服務群集所構成，因此本研究稱階層 1 為分配工作需求之需求管理者(Request Manager)，階層 2 為分配實際服務之服務管理者，階層 3 為可執行工作之服務節點。當進入系統執行的所有工作量大於 k 時(k 為第二階層之服務群集最大負載上限)，則系統會自動採用三階層式雲端運算網路拓樸。需求管理者會將進入系統的所有工作收集起來，並依順分配給服務管理者，服務管理者會將工作依邏輯大小切割成若干個子工作，在分配給最下層的可執行工作之服務節點。

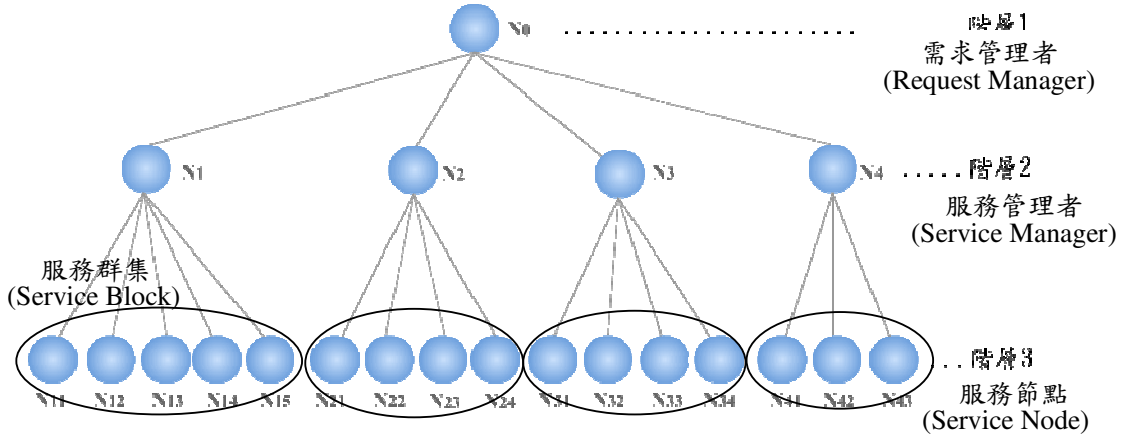

圖 5、三階層式雲端運算網路拓樸

動態階層式雲端運算網路拓樸可依據工作量的需求動態的調整網路拓樸，藉此可有效降低記錄節點資料之成本與避免不必要的資源浪費。然而若階層過高亦將導致網路管理成本過高，因此本研究中將以動態階層式網路拓樸做為研究的架構。

4. LBMM(Load Balance Min-Min)排程演算法

Min-min 排程演算法執行時會考慮到所有工作的最小完成時間，但是其最大的缺點是因為 Min-min 排程演算法只考慮到每一個工作在服務節點上的完成時間而沒有考慮到每個服務節點的負載狀況，因此可能造成有些服務節點總是非常得忙碌而有些服務節點則是閒置的情況。所以在本研究中提出一個 LBMM(Load Balance Min-min)排程演算法，改善 Min-min 之負載不平衡的狀況，且更能有效的降低每個服務節點的執行時間。

為能在動態階層式的雲端運算網路拓樸下，使各服務節點之工作負載達到平衡，且能有效的降低每個服務節點的執行時間，因此本研究結合 OLB 排程演算法與 LBMM 排程演算法，以達到此目標。LBMM 排程演算法所使用的變數定義於表 1，LBMM 排程演算法的執行步驟如表 2 所示，LBMM 排程演算法如圖 6 所示。

表 1、LBMM 的變數定義

變數名稱	意義
x	需執行的工作總個數
N_i	各工作經切割後納入各任務集合中的總子工作數，$1 \le i \le x$
y	各服務管理者所能管控的服務節點個數
M_j	可執行子工作之服務節點集合內服務節點的總個數，$1 \le j \le y$
α	子工作的編號，$1 \le \alpha \le N_i$
γ	服務節點的編號，$1 \le \gamma \le M_j$

表 2、LBMM 執行步驟

Step1：	針對各個 N_i 個子工作分別在 M_j 個服務節點上找尋可以使用最小執行時間之服務節點，並形成一個 Min-time 服務節點集合。
Step2：	再從 Min-time 服務節點集合中選出其中最小執行時間的服務節點 γ。
Step3：	將子工作 α 分配給服務節點 γ。
Step4：	將被完成的子工作 α 從任務集合中刪除。
Step5：	將被分配到執行子工作 α 的服務節點 γ 重新排在所有服務節點的最後面。
Step6：	重複 Step1 到 Step5，直到所有的子工作完成。

```
int exe-time[1..Ni][1.. Mj];
/*array 中存放每個子工作 Ni 在每個服務節點
                              Mj 上所須的執行時間*/
int Min-time[1..n][1..3];
          /*建立 Min-time 資源集合陣列 */
int min=∞;
int Min-Min-time; /*最後的最小完成時間值*/
int  α, γ;
for ( α = 1  to  Ni )
{
  for ( γ = 1  to  Mj )
  {
    if ( exe-time[ α ][ γ ] < min )
     {
       min=exe-time[ α ][ γ ];
   /*找出每一個子工作 Ni 的最小執行時間*/
     }
  }
  for (int k = 1 to Ni )
            /*建立 Min-time 資源集合陣列*/
  {
    Min-time[k][1]=min;
/* array1 存放每個子工作 Ni 的最小執行時間*/
    Min-time[k][2]= α;
              /* array2 存放子工作的 id*/
    Min-time[k][3]= γ;
              /* array2 存放服務節點的 id*/
  }
}
for (int k = 1 to Ni )
{
  if ( Min-time[k][1] < min )
     {
       min= Min-time[k][1];
 /*找出 Min-time 資源集合陣列中的最小執行
                                  時間*/
        α = Min-time[k][2];
        γ = Min-time[k][3];
```

```
        }
}
Min-Min-time = min ; /*最小的完成時間值*/
Node = γ;                    /*挑選出之服務節點
γ*/
Subtask = α;          /*被執行之子工作α*/
將 Subtask α從任務集合中刪除
將 Node γ 重新排在所以服務節點的最後面
```

圖 6、LBMM 之演算法

本研究在動態階層式雲端運算網路拓樸架構下，提出混合式排程法，結合 OLB 排程演算法與本研究所提出的 LBMM 排程演算法之特性，讓工作可平均的分配到各個服務節點上，並考慮所有工作在服務節點上執行的最小完成時間，讓工作皆可在最短的時間內被完成。

5. 研究方法與架構

依據第 4 節本研究所提出的 LBMM 排程演算法，本節將詳細的說明如何在動態階層式雲端運算網路拓樸架構中，利用 OLB 排程演算法與本研究所提出之 LBMM 排程演算法，達到快速的將工作分配到適當的資源上，並詳細分析比較 Min-min 排程演算法與 LBMM 排程演算法之執行效能與服務節點的負載狀況。

5.1 研究設計

本研究的基本想法是當有工作進入動態階層式雲端運算網路拓樸架構時，需求管理者將收集所有的工作需求並依序存放到佇列中。接著，依據工作的大小來決定利用的階層式架構，並且參照此階層式網路拓樸，將工作分配到下一階層之節點以進行工作之執行。

本研究假設執行的環境如下：

1、 傳輸時間是可掌握的。
2、 每個工作所需執行的時間可以被預測[12]。
3、 工作具可分割性，且每一個服務節點皆可以將分配到的子工作執行完成。
4、 服務節點的總數量大於工作切割成的子工作之總數量，即每個子工作都可以對應到一個服務節點以進行執行。

本研究在動態階層式雲端運算網路拓樸架構下，先以 OLB 排程演算法進行工作的分配並切割成若干個子工作，再依 LBMM 之排程演算法依據每個服務節點之完成時間分配其資源。

假設目前有四個大型的工作，分別為 A、B、C、D，需要被執行，因此，本研究利用三階層式雲端運算網路拓樸架構進行工作的配置，即依據每個服務節點之完成時間分配其資源，其步驟如下說明：

步驟一： 需求管理者 N0 將需要執行的工作 A、B、C、及 D 分別收集，並存放到工作佇列中如圖 7，以方便工作執行時，可依佇列中之順序分配給下一階層。

A	B	C	D

圖 7、工作佇列

步驟二： 需求管理者以 OLB 排程演算法將工作佇列中待執行之工作依序分配給服務管理者，即將工作 A 分配給節點 N1；將工作 B 分配給節點 N2；將工作 C 分配給節點 N3，依此類推。

步驟三：當服務管理者(N1, N2, N3, N4)依序接收到工作(A, B, C, D)時，則開始將每個工作以邏輯為單位分成 N_i 個子工作。如圖 8 所示，若工作 A 被分割成 4 個子工作，工作 B 被分割成 6 個子工作，工作 C 被分割成 5 個子工作，…等。

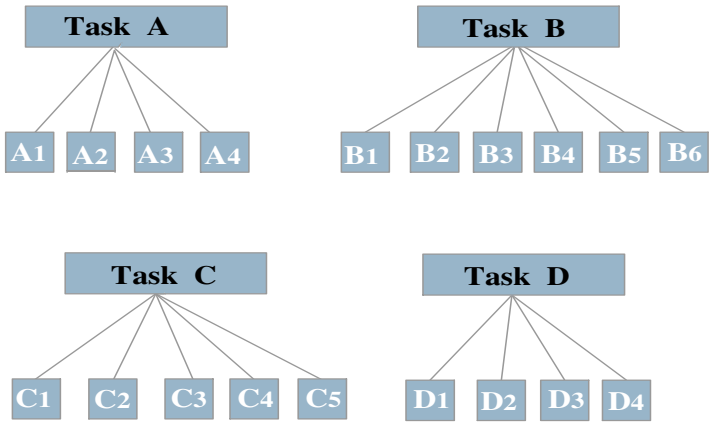

圖 8、將工作切割成子工作

步驟四： 服務管理者利用 LBMM 排程演算法，首先計算出該子工作分配到 M_j 個不同服務節點上的最小執行時間如表 3。假設子工作 A1 在節點 N13 上的執行時間為最小，則可記為 Min-Time =(A1, N13)= 8，其中 Min-Time 是一個含 M_j 個元素的一維陣列，代表一個由各個最小執行時間組成的集合，如式(1)所示。再從 Min-Time 陣列中找最小值，這裏為(A1, N13)，其對應的工作號是 A1，對應的節點號是 N13。因此把子工作 A1 交給節點 N13 執行，並將 A1 從任務集合中刪除，同時更新工作分配前每個節點執行時間，如表 4 所示。

表 3、工作分配前每個節點執行時間(第一次)

節點 / 子工作	N11	N12	N13	N14	N15
A1	12	16	8	11	14
A2	19	20	17	19	18
A3	17	22	19	24	25
A4	10	27	13	21	19

$$\text{Min-Time} = \begin{bmatrix} A1, N13 \\ A2, N13 \\ A3, N11 \\ A4, N11 \end{bmatrix} = \begin{bmatrix} 8 \\ 17 \\ 17 \\ 10 \end{bmatrix} \quad (1)$$

步驟五： 從表 4 可以看到，由於已經把子工作 A1 分配給節點 N13，所以 A1 會從集合中被刪掉，而節點 N13 會重新排在所有節點的最後面，等到所有節點都被分配到工作時才會又重新進入排程中。接著會在選出每個子工作的最小執行時間(Min-Time)，形成一個 Min-time 服務節點集合，如式(2)所示。再從 Min-Time 陣列中找出最小值，這裏為(A4, N11)，其對應的子工作為 A4，對應的節點為 N11，因此把子工作 A4 交給節點 N11 去執行，將 A4 從任務集合中刪除，同時更新工作分配前每個節點執行時間如表 5 所示。

表 4、工作分配前每個節點執行時間(第二次)

節點 / 子工作	N11	N12	N14	N15
A2	19	20	19	18
A3	17	22	24	25
A4	10	27	21	19

$$\text{Min-Time} = \begin{bmatrix} A2, N15 \\ A3, N11 \\ A4, N11 \end{bmatrix} = \begin{bmatrix} 18 \\ 17 \\ 10 \end{bmatrix} \quad (2)$$

步驟六： 從表 5 可以看到，由於已經把子工作 A4 分配給節點 N11，所以 A4 會從集合中被刪掉，而節點 N11 會重新排在所有節點的最後面，等到所有節點都被分配到工作時才會又重

新進入排程中。接著繼續選出每個子工作的最小執行時間(Min-Time)，形成一個 Min-time 服務節點集合，如式(3)所示。再從 Min-Time 陣列中找出最小值，這裏為(A2, N15)，其對應的子工作為 A2，對應的節點為 N15，因此把子工作 A2 交給節點 N15 去執行，將 A2 從任務集合中刪除，同時更新工作分配前每個節點執行時間如表 6 所示。

表 5、工作分配前每個節點執行時間(第三次)

節點 / 子工作	N12	N14	N15
A2	20	19	18
A3	22	24	25

$$\text{Min-Time}=\begin{bmatrix}A2,N15\\A3,N12\end{bmatrix}=\begin{bmatrix}18\\22\end{bmatrix} \quad (3)$$

步驟七： 從表 6 可以看到，由於已經把子工作 A2 分配給節點 N15，所以 A2 會從集合中被刪掉，而節點 N15 會重新排在所有節點的最後面，等到所有節點都被分配到工作時才會又重新進入排程中。最後把剩下的最後一個子工作 A3 交給在此最短時間完成的節點 N12 執行，即完成服務管理者(N1)的分配工作，其他服務管理者依此類推。

表 6、工作分配前每個節點執行時間(第四次)

節點 / 子工作	N12	N14
A3	22	24

$$\text{Min-Time} = \begin{bmatrix}A3,N12\end{bmatrix} = \begin{bmatrix}22\end{bmatrix} \quad (4)$$

上述方法，除了將子工作依序分配給下一階層，讓每個提供服務的服務節點之工作負擔盡可能的平均之外，同時也利用 LBMM 排程演算法考慮整體的完成時間，改善了 OLB 排程演算法沒有考慮到執行時間的缺點同時也改善了 Min-min 排程演算法之負載不平衡的缺點，讓需要被執行的工作可以在最短的時間內被分配到最適當的資源。

5.2 研究結果與分析

為驗證本研究所提出的 LBMM 排程演算法確實可以使各個服務節點達到更有效的負載平衡，因此將以原有的 Min-min 排程演算法與本研究所提出之 LBMM 進行實例的分析說明，探討子工作分配到不同服務節點上之負載狀況與每個服務節點所需的最小完成時間。

(1) Min-min 排程演算法

依據本研究所提出之研究架構，利用 Min-min 排程演算法進行工作之分配，則可以找出每一個子工作在任一服務節點上的最小執行時間如表 7 所示。從表中可以知道子工作 A1 會分配給節點 N13，需要的最小執行時間為 8；子工作 A2 會分配給節點 N13，需要的最小執行時間為 17，依此類推。將上述結果用圖形呈現，則可以更清楚的得知每個服務節點的負擔情況，如圖 9 所示。

表 7、Min-min 之工作分配狀態

子工作	節點	Min-Time
A1	N13	8
A2	N13	17
A3	N11	17
A4	N11	10

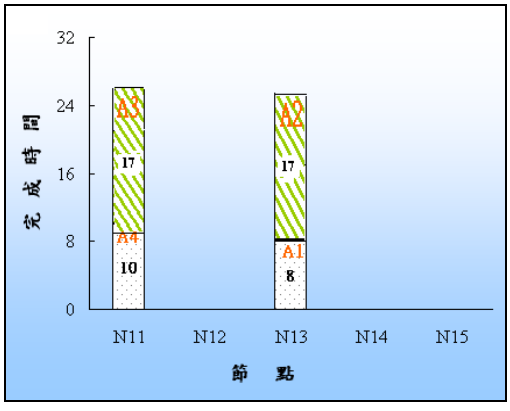

圖 9、使用 Min-min 排程演算法之各服務節點的負載狀況

由圖 9 中可以看出，如果採用 Min-min 排程演算法，則會將所有的子工作都分配到節點 N11 及 N13 二個服務節點上執行子工作的計算。因此可以發現，節點 N12、N14 與 N15 是處於閒置的狀態，造成某些服務節點可以執行工作卻沒有被分配到工作，而某些服務節點處於忙碌狀態卻又被分配到工作增加其負擔。在這個例子中，先以 OLB 排程演算法進行工作的分配並切割成若干個子工作，再使用 Min-min 排程演算法，完成所有工作所需的時間為 27 單位。

(2) LBMM 排程演算法

依據本研究所提出之研究架構，利用 LBMM 排程演算法進行工作之分配，則可以找出每一個子工作在任一服務節點上之最小執行時間如表 8 所示。從表 8 中可以知道子工作 A1 會分配給節點 N13，需要的最小執行時間為 8；子工作 A2 會分配給節點 N15，需要的最小執行時間為 18，依此類推。將上述結果用圖形呈現，則可以更清楚的得知個節點的負擔情況，如圖 10 所示。

表 8、LBMM 之工作分配狀態

子工作	節點	Min-Time
A1	N13	8
A2	N15	18
A3	N12	22
A4	N11	10

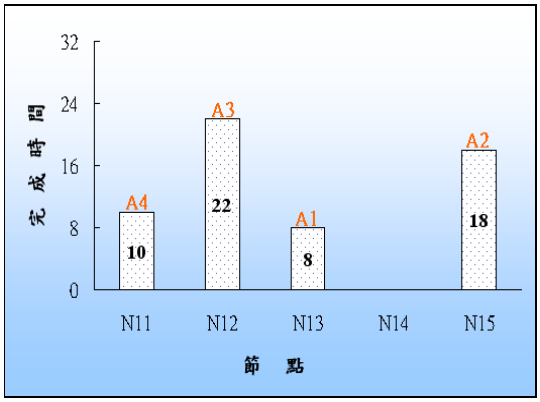

圖 10、使用 LBMM 排程演算法之各服務節點的負載狀況

由圖 10 中可以看出，採用 LBMM 排程演算法，則每一個服務節點都可以被分配到一個子工作，不會發生某些服務節點可以執行工作卻沒有被分配到工作，而某些服務節點處於忙碌狀態卻又被分配到工作的現象，也因為每個服務節點都可以平衡的被分配到工作，所以使得每一個服務節點的執行時間，與整體的完成時間也很平均。在這個例子中，先以 OLB 排程演算法進行工作的分配並切割成若干個子工作，再使用 LBMM 排程演算法，完成所有工作所需的時間為 22 單位。

由上述結果可知，採用 Min-min 排程演算法所需要的最小完成時間為 27，而採用 LBMM 排程演算法的最小完成時間則為 22，由此結果可以說明 LBMM 排程演算法除了可以平均的分配工作，達到系統負載的平衡之外，而其整體的工作的完成時間也與 Min-Min 排程演算法差不多，有效的提升系統整體的效能。

6. 結論及未來工作

在雲端運算的環境下，由於組成的服務節點是以分散式的分式進行運算與提供服務。因此，如何建置一個網路拓樸架構與有效的利用這些服務節點資源，是一個值得深入探討的議題。因此本研究依據動態階層式雲端運算網路拓樸的架構，結合 OLB 排程演算法與 LBMM 排程演算法進行工作的配置，透過 OLB 排程演算法讓每個服務節點都是處於工作狀態，進而實現負擔平衡。除此之外，並利用 LBMM 排程演算法使得每個工作在運算服務節點上的執行時間最小，且達到整體的完成時間最小，提升整體的效能，且透過動態階層式雲端運算網路拓樸評估需要執行工作的服務節點數量，動態的調整拓樸，使得資源可有效的被控制，與降低系統控管成本。

在本研究中，更因利用階層式的網路拓樸架構，使得計算完成的資訊可由第二階層的子管理者進行整合的工作之後，再回傳給管理者，實現了動態階層式雲端運算網路拓樸之負載平衡與善用資源之目的。

未來工作則會考慮到服務節點處於動態的環境下所面臨的問題，例如，在動態階層式的網路架構下，當服務節點損壞時，可能造成上一階層無法知道下一階層的位置而使得工作無法有效的分配，以致影響執行的效率。因此，如何有效的維護與管理服務節點，並達到容錯能力則是未來研究方向之一。

誌謝

這篇論文是國科會計畫(NSC95-2221-E-324-019-MY3 與 NSC97-2221-E-324-007-MY3)研究成果的一部份，我們在此感謝國科會經費支持這個計畫的研究。

參考文獻

[1] 鍾添曜、蔡嘉鴻，*「點對點服務搜尋拓樸演算法之設計」*，元智大學資訊工程研究所，碩士論文，2003。

[2] Armstrong, R., Hensgen, D. and Kidd, T., "The Relative Performance of Various Mapping Algorithms is Independent of Sizable Variances in Run-time Predictions," *7th IEEE Heterogeneous Computing Workshop (HCW '98)*, pp. 79-87, 1998.

[3] Aymerich, F.M., Fenu, G. and Surcis, S., "An Approach to A Cloud Computing Network," *1st International Conference on the Applications of Digital Information and Web Technologies*, pp. 113-118, August 4-6, 2008.

[4] Birman, K., Chockler, G. and Renesse, R., "Toward A Cloud Computing Research Agenda," *ACM SIGACT News*, Vol. 40, No. 2, pp. 68-80, June, 2009.

[5] Brauna, T.D., Siegelb, H.J., Beckc, N., Bölönid L.L., Muthucumaru Maheswarane, Albert, I.R., Robertsong, J.P., Theysh, M.D., Yaoi, B., Hensgenj, D. and Freundk, R.F., "A Comparison of Eleven Static Heuristics for Mapping a Class of Independent Tasks onto Heterogeneous Distributed Computing Systems," *Journal of Parallel and Distributed Computing* , Vol. 61, Issue 6 , pp. 810-837, 2001.

[6] Buyya, R., Yeo, C.S. and Venugopal, S., "Market-oriented Cloud Computing: Vision, Hype, and Reality for Delivering IT Services as Computing Utilities," *10th IEEE International Conference on High Performance Computing and Communications*, pp. 5-13, September 25-27, 2008.

[7] Freund, R.F., Gherrity, M., Ambrosius, S., Campbell, M., Halderman, M., Hensgen, D., Keith, E., Kidd, T., Kussow, M., Lima, J. D., Mirabile, F., Moore, L., Rus,t B., and Siegel, H. J., "Scheduling Resources in Multi-user, Heterogeneous, Computing Environments with SmartNet," *7th IEEE Heterogeneous Computing Workshop (HCW'98)*, pp. 184-99, 1998.

[8] Grossman, R.L., Gu, Y., Sabala, M. and Zhang, W., "Compute and Storage Clouds Using Wide Area High Performance Networks," *Future Generation Computer Systems*, Vol. 25, No. 2, pp. 179-183, February, 2009.

[9] Grossman, R.L., "The Case for Cloud Computing," *IT Professional*, Vol. 11, No. 2, pp. 23-27, March, 2009.

[10] Junginger, M. and Lee, Y., "The Multi-Ring Topology - High-Performance Group Communication in Peer-to-Peer Networks," *Proceedings of Peer-to-Peer Computing*, pp. 49- 56, 2002.

[11] Ritchie, G. and Levine, J., "A Fast, Effective Local Search for Scheduling Independent Jobs in Heterogeneous Computing Environments," *Journal of Computer Applications*, Vol. 25, Issue 5, pp. 1190-1192, 2005.

[12] Satyanarayanan, M., "Pervasive Computing: Vision and Challenges," *IEEE Personal Communications*, Vol. 8, No. 4, pp. 10-17, 2001.

[13] Vouk, M.A., "Cloud Computing- Issues, Research and Implementations," *Information Technology Interfaces*, pp. 31-40, June 23-26, 2008.

[14] Wang, L.H., Tao, J. and Kunze, M., "Scientific Cloud Computing: Early Definition and Experience," *10th IEEE International Conference on High Performance Computing and Communications*, pp. 825-830, September 25-27, 2008.

[15] Weiss, A., "Computing in the Clouds," *netWorker*, Vol. 11, No. 4, pp. 16-25, December, 2007.

[16] Amazon.com: Online Shopping for Electronics, Apparel, Computers, Books, DVDs & More, December 2009, http://www.amazon.com/.

[17] Amazon Web Services, December 2009, http://aws.amazon.com/.

[18] More Google Product, December 2009, http://www.google.com/options/.

[19] Salesforce. com http://www.salesforce. com/tw.

[20] What is Cloud Computing?, December 2009, http://www.zeus.com/cloud_com puting/cloud.html.

[21] Zeus http://www.zeus.com/.