以混合式負載平衡策略提昇三階層式點對點網路拓樸之執行效能

嚴國慶　王淑卿*　陳秀芳　王順生
朝陽科技大學
{kqyan; scwang*; s9514624; sswang}@cyut.edu.tw

摘要

由於科技的進步及網路的普及，使得點對點計算逐漸成為分散式應用的主流。但由於點對點計算主要是透過分散的節點合作完成一個大型的工作，因此如何將工作有效的分配到每一個節點上，使系統中每個節點的工作達成負載平衡，是一個值得探討的議題。本研究在一個三階層式點對點網路拓樸架構下，提出混合式負載平衡排程演算法以進行工作的配置，透過門檻值的設定，使得每個需要執行的工作能快速的被分配到適當的節點上，除有效的改善每個節點的工作負擔外，還可依據工作的特性來選擇最合適的節點，提供三階層式點對點網路拓樸之負載平衡與執行效率的品質保證。

關鍵詞：分散式系統、點對點計算、排程演算法、行動代理人、負載平衡

Abstract

With the great technology advanced and the well popularization of Internet, P2P computing has gradually become the mainstream of distributed applications. However, in this study, a two-phase scheduling algorithm under a three-level P2P network is advanced. The proposed scheduling algorithm can utilize better executing efficiency and maintain the load balancing of system.

Keywords: Distributed system, P2P computing, Scheduling, Load balancing.

1. 前言

網路最主要的功能，就是將許多不同的電腦或資源連接在一起，並扮演這些電腦或資源之間溝通的管道。這些電腦擁有自己的 CPU、時脈、記憶體、匯流排以及周邊裝置等，而將這些電腦集合在一起，除了可以提高系統的計算能力，更可以共享系統中的資源。為達成這個目的，目前常用的架構為分散式系統[7]。

分散式系統概略可以分為主從式系統與點對點計算兩類。主從式系統的架構是一種集中式的管理方式，架構中以伺服器為中心，提供各類資訊內容、電子郵件及資訊搜尋等服務。然而主從式架構最大的缺點是伺服器一旦發生故障，則整個主從式系統可能發生故障或癱瘓[7]。

點對點計算(Peer-to-Peer computing，簡稱 P2P)又稱對等式網際網路技術，是一種網路新技術[1]。點對點計算透過網路中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[12]。從計算模式上來說，點對點計算打破了傳統的主從式模式，每個在網路中的節點都可以擔任 Client 的工作，也可以擔任 Server 的工作。因此，使用點對點計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。同時因為資源是分佈在每個節點上，所以可以運用每個節點的一些資源協同合作完成一個大的工作[12]。

但是如何運用點對點計算的優點，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。因此，本研究在一個三階層式點對點網路拓樸架構下，提出混合式負載平衡排程演算法，結合 OLB 排程演算法與本研究所提出之 LBMM 排程演算法及行動代理人機制進行工作的配置，藉以提高工作完成效率與達到善用節點資源之目的。

本文第二節為文獻探討，說明相關的拓樸架構與分散式系統管理及排程演算法；第三節將介紹本研究所提出之 LBMM 排程演算法；第四節則說明究環境與流程；第五節將說明研究的方法與架構；第六節為結論與未來工作。

2. 文獻探討

在本節中，將分別說明點對點計算相關的拓樸架構與排程演算法。

2.1 網路拓樸架構

在網路的環境中，只要電腦間彼此可以透過通訊媒體互相連結，即能構成一個網路架構[1]。在網路架構中，每台電腦(節點)連結的方式，即所連結的形狀稱之為「拓樸」，這些拓樸可以依據節點排列的形狀而加以分類。目前應用在點對點計算架構的網路拓樸，可區分為星狀拓樸、環狀拓樸與階層式拓樸[1]。

星狀拓樸(Star Topology)建立的方式是必須先建立一個伺服端的主機，而伺服端的主機會提供特定的服務讓客戶端使用，如資料庫系統、網頁伺服

器系統和很多簡單的分散式系統都是以星狀拓樸的架構提供服務。這種架構的應用方式是將所有的資料都集中存放在只有單一的中央伺服端中，並提供給很多的客戶端直接連接，以取得所需要的資料。而在點對點計算架構中，可使用這種星狀拓樸的服務型態，提供很多的客戶端搜尋資料。然而在點對點系統架構中，伺服端只提供很多的客戶端做資料的搜尋，並沒有提供客戶端直接在伺服端做資料的存取[1]。

由於星狀拓樸的服務方式只有一台伺服端的設備，因此可以服務客戶端的數量有限。為解決星狀拓樸所產生之問題，因此將多個伺服器連結起來，形成一個環狀拓樸(Ring Topology)來平衡伺服端的負荷，便可提高服務客戶端的數量。這種建立拓樸的方式，就是節點與節點之間的連線需依據成本來考量，以選擇最佳的連接節點。Markus 等學者，就以環狀連結拓樸的概念，運用在點對點系統架構中，將節點連結形成環狀拓樸，避免網路中的節點盲目搜尋資料，減少網路搜尋訊息的數量[5,6]。另外，為了防止單一鏈結的環狀拓樸中會發生連結斷裂，因此在每個節點中另外建立一條備份連結(Backup link)，形成多連結(Muti-ring)的環狀拓樸，使訊息傳送封包遺失率相對較低。不過，當環狀拓樸節點需要搜尋資訊時，如果環狀網路拓樸的節點數非常多時，可能造成繞送時間過久，影響搜尋的效能。

在網路的相關應用中，階層式系統的使用已有很長的一段歷史，如領域名稱伺服器(Domain Name Server；DNS)所形成的拓樸，就是採用階層式的網路架構。階層式拓樸(Hierarchical Topology)的形成方式，主要會有一個名稱伺服器(Root name sever)來負責驗證的機制，每個下層節點都需要上層節點的驗證許可，依照這樣的方式形成樹狀的拓樸[9,11]。其優點是每一個節點只須記錄其上一節點之位置，因此可有效降低記錄節點資料。

總而言之，點對點計算相關的拓樸架構各有其優缺點，在本研究中將以三階層式網路拓樸做為研究的架構。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，不會有星狀式拓樸只有一台伺服端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，因此可有效降低記錄節點資料。

2.2 排程演算法

不同的排程方法各有其不同的特性，有些排程方法對於某些特定的應用情況有利，有些則否。因此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。因此，本節將探討不同排程演算法之特性並分別說明。

OLB(Opportunistic Load Balancing)排程演算法試圖讓每一部電腦都保持忙碌的狀態，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分配給目前可以用的電腦進行執行。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[2,4]。

MET(Minimum Execution Time)排程演算法試圖讓每個工作可以獲得最好的電腦支援，不考慮電腦目前的工作量，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[3,5]。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，不適用於異質性電腦系統之應用[2,3]。

MCT(Minimum Completion Time)排程演算法將目前具有最小完成時間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[2,3]。

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[8]。

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[8]。

由於 OLB 排程演算法簡單且容易實行，並能使所有的節點盡可能地都處於工作狀態，因此本研究將在三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節點的執行時間。

3. LBMM 排程演算法

Min-min 排程演算法執行時考慮到所有工作的最小完成時間，但因 Min-min 排程演算法只考慮每一個工作在節點上的完成時間而未考慮每個節點的負載狀況，因此可能造成有些節點總是非常忙碌而有些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節點的執行時間。

除此之外，由第二節的說明，可以得知多階層式點對點網路拓樸可有效降低記錄節點資料之成本。然而若階層過高亦將導致網路管理成本過高，因此本研究中將以三階層式網路拓樸做為研究的架構。換言之，本研究假設在一個三階層式的點對點網路拓樸環境下，所有可提供計算之資源節點，如圖 1 所示分成三個階層，階層 1 為分配工作之管理者，階層 2 為分配工作之子管理者，階層 3 為可利用的資源節點。

為能在三階層式的點對點網路拓樸下，使各節

點之工作負載達到平衡，且能有效的降低每個節點的執行時間，因此本研究結合 OLB 與 LBMM 排程演算法，以達到此目標。LBMM 執行步驟如圖 2 所示。本研究在三階層式點對點網路拓樸架構下，提出混合式負載平衡排程演算法，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節點上，並考慮所有工作在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。

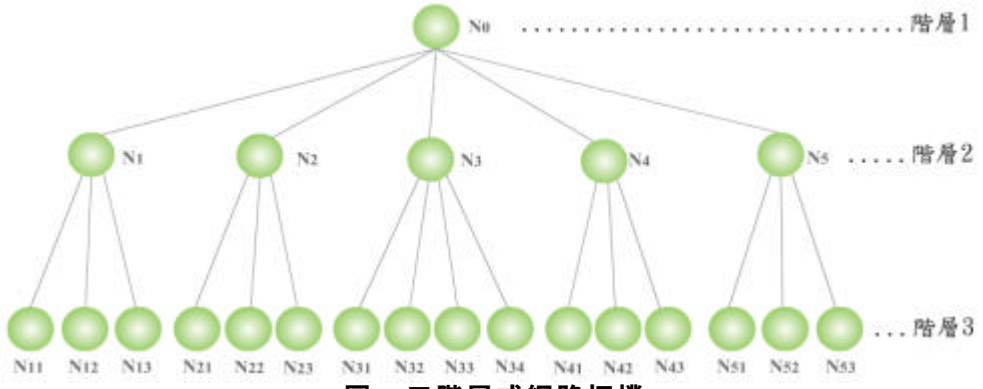

圖 1 三階層式網路拓樸

Step 1:	針對各個子工作分別在不同的節點上找尋可以使用的最小執行時間之資源節點，並形成一個 Min-Time 資源節點集合。
Step 2:	再從 Min-Time 資源節點集合中選出其中最小執行時間的節點。
Step 3:	將子工作分配給節點。
Step 4:	將被完成的子工作從任務集合中刪除。
Step 5:	將被分配到執行子工作的節點重新排在所有資源節點的最後。
Step 6:	重複 Step 1 到 Step 5，直到所有的子工作完成。

圖 2 LBMM 執行步驟

4. 研究方法與流程

在點對點系統中，由於節點的組成是在一個異質性的環境上，亦即每個節點執行工作的能力不盡相同，因此在選擇節點執行工作時，不僅需考慮節點 CPU 的使用率，還需考慮其他影響節點有效性的因素，因此如何有效的選擇節點來執行工作是本研究另一個重要目標。

由於使用者交付執行的工作會有不同的特性，因此需要某些特定的資源，如在執行生物序列組合時，對記憶體會有大量需求[12]，而為了執行每項工作時都能達到其最佳執行效能，因此必須針對其工作之特性，採取不同考慮因素之決策變數，也就是說，依據工作執行時所需的資源作為其決策變數。而本研究主要考量節點是否可支援大量的運算能力與是否可快速地將工作分配到執行節點上的整個執行效能，因此對於 CUP 剩餘量、記憶體剩餘量與傳輸速度有其限制，因此決策變數可定義為：

V_1 = CPU 剩餘量；

V_2 =記憶體剩餘量；

V_3 =傳輸速度。

由於不同的工作所需要計算的複雜度不一，為避免某些能力較差的節點分配到需要高度複雜且大量的運算工作，而導致選擇到不適合的節點，造成執行效能的降低，因此，本研究藉由加入限制條件來提高整體的執行品質。其中，管理者利用“子管理者門檻值”挑選適合的子管理者。由於在階層式點對點網路拓樸架構下，由同一個子管理者所管理的群組可利用的資源節點(階層 3)，必已滿足“子管理者門檻值”方能彙聚在同一群組中。待選定子管理者後，接著將以“可利用的資源節點門檻值”來選擇最佳執行節點。

4.1 子管理者門檻值的設定

由於所要執行的工作需求對 CPU 剩餘量、記憶體剩餘量、傳輸速度等條件有基本需求，因此必須將上述因素設定為限制式。假設在一特定的應用中，所要執行的工作有如下之需求：

(1) CPU 剩餘量 = 490 MB/s

(2) 記憶體剩餘量 = 203698 KB/s

(3) 傳輸速度 = 7.09 MB/s

為了能執行此特定的工作，因此以這三個需求因素做為“子管理者門檻值”，當子管理者的 CPU 剩餘量、記憶體剩餘量及傳輸速度都跨過這個門檻值時，才會是 OLB 挑選的候選子管理者之一。

由於每個工作的特性不同，需要的限制條件也會依據其工作的特性來決定，所以每個工作都會有不同的限制條件。因此，管理者利用“子管理者門檻值”挑選適合的子管理者，並使用 OLB 排程法將待執行的工作分派給適當的子管理者。

4.2 可利用的資源節點門檻值的設定

當子管理者節點跨過“子管理者門檻值”時，表示該子管理者節點的下層節點皆有能力執行

此工作。而由於點對點系統中每個節點都是動態的組成，因此可能某些節點目前是處於閒置的狀態，但下一秒鐘就處於忙碌的狀態，以致提高工作的執行時間，影響整個系統的效能，所以透過“可利用的資源節點門檻值”的限制條件，來選擇最佳執行節點，其步驟分為下列四點：

步驟(1)：計算每個子工作的平均執行時間。

步驟(2)：當子工作需要執行的時間=平均執行時間時，則可正常執行。

步驟(3)：當子工作需執行的時間>平均執行時間時，最小執行時間會被設定為“8”(無窮大)表示無法執行，並讓其他已執行過的節點重新進入系統參與工作的執行。

步驟(4)：重複步驟(1)~(3)，直到工作執行完畢。

本研究主要在階層架構下，利用排程演算法快速的將工作分配出去，並利用門檻值的方式篩選出有效的節點。也就是說，首先會透過管理者將需要執行的工作利用佇列方式存放，利用 OLB 排程演算法的特性將工作分配給下一層節點，而由於每個交付執行的工作有不同的特性，所以選擇節點的限制條件也不同，因此利用“子管理者門檻值”依據工作特性來選擇適合工作的節點(子管理者)。而子管理者接收到工作後，會先將工作以邏輯大小切割成子工作，再利用 LBMM 排程演算法的特性將工作分配給第三階層的節點(工作執行者)，而為了避免某些節點執行時間過長而影響到系統的執行效率，所以利用“可利用的資源節點門檻值”有效的選出最佳的執行節點。本研究所提出的架構，其整個運作機制流程如圖 3 所示。

5. 研究假設與實例

依據上述提出的方法，本節將說明如何在三階層式網路拓樸中，利用 OLB 與 LBMM 排程演算法以及如何選擇最合適的工作執行節點，有效的提供三階層式點對點網路拓樸架構排程機制，達到快速的將工作分配到適當的節點上執行，並說明排程演算法之執行效能與節點的負載。

本研究的基本想法是當工作進入三階層式點對點網路拓樸時，根節點(管理者)會收集所有工作的資訊並依序存放到佇列中，同時派遣行動代理人收集節點的相關資訊。接著，依據此工作特性與條件限制分配給下一階層(子管理者)。而當子管理者收到要執行需求之工作時，會將分配到的工作切割成若干個子工作，再依據節點能力將工作分配給下一階層之節點(資源)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[10]。
3、工作具可分割性，且每一個節點皆可將分配到的子工作執行完成。
4、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節點執行。

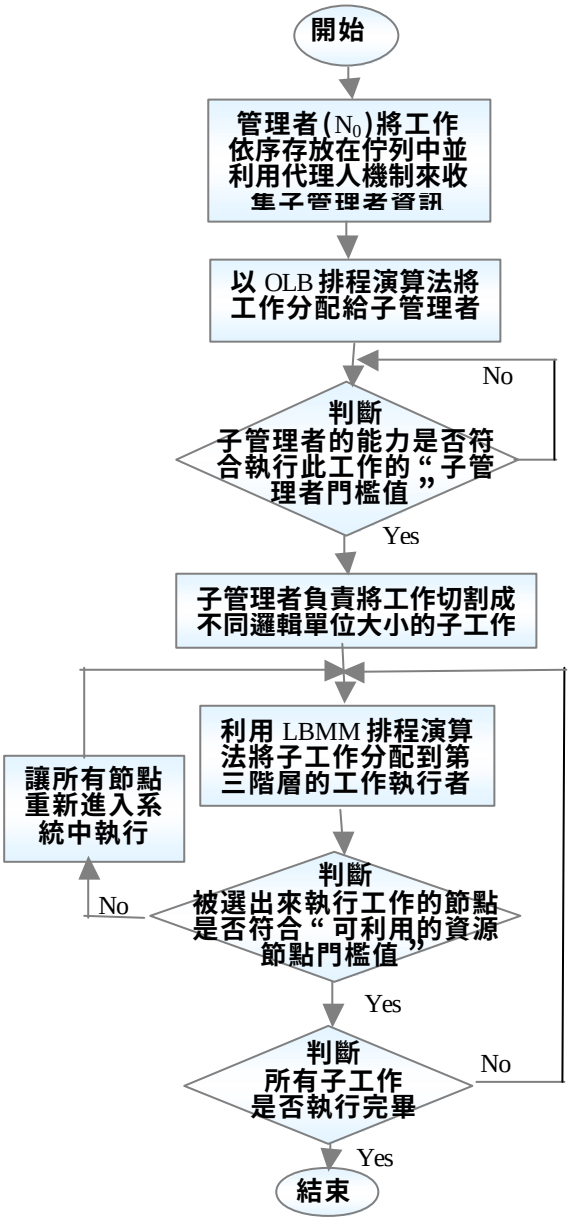

圖 3 運作機制流程

本研究在三階層式網路拓樸架構下，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節點之完成時間分配其資源。

假設目前有四個工作，分別為 A、B、C、D、E 需被執行，其執行步驟如下說明：

步驟一：節點 N_0 收集要執行的工作 A、B、C、D 及 E，並存放到工作佇列中(圖 4)，方便於工作執行時，可依佇列中之順序分配給下一階層。同時派出行動代理人收集節點(子管理者)的相關資訊(CPU 剩餘量、Memory 剩餘、傳輸速率等)，如表 1 所示。

A	B	C	D	E

圖 4 工作佇列

表 1 各節點的相關資訊

	CPU 剩餘量 (MB/s)	Memory 剩餘量(MB/s)	傳輸速率 (MB/s)
N_1	503	456	29.04
N_2	250	350	12.46
N_3	490	203	7.09
N_4	750	230	23.25
N_5	128	322	21.03

步驟二：以 OLB **排程演算法將工作佇列中待執行之工作依序分配給子管理者，亦即依工作的限制條件將工作** A **分配給節點** N_1、N_2、N_3、N_4、**或** N_5；**繼之，再依工作的條件限制選擇合適節點將工作** B **分配給節點** N_1、N_2、N_3、N_4、**或** N_5(**去除掉已執行工作** A **的節點)；工作** C、D、E **依此類推。**

步驟三：由於工作 A **的“子管理者門檻值”為“子管理者限制條件** A **”：**

(1) CPU **剩餘量** = 500 MB/s
(2) **記憶體剩餘量** = 256 MB/s
(3) **傳輸速度** = 20 MB/s

其中，節點 N_1 **滿足“子管理者限制條件** A **”；節點** N_2 **的** CPU **剩餘量與傳輸速率都不符合“子管理者限制條件** A **”；節點** N_3 **的** CPU **剩餘量、**Memory **剩餘與傳輸速率皆不符合“子管理者限制條件** A **”，因此將工作** A **分配給節點** N_1。

步驟四：當子管理者(N_1, N_2, N_3, N_4, N_5)**接收到不同工作(**A, B, C, D, E)**時，則將每個工作以邏輯為單位分割成若干個子工作。**

步驟五：利用 LBMM **排程演算法，計算出該子工作分配到** C **個不同資源節點上的最小執行時間如表** 2 **所示。假設子工作** A1 **在節點** N_{13} **上的執行時間為最小，則可記為** Min-Time =(A1, N_{12})= 14，**其中** Min-Time **是一個含** C **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。**

步驟六：計算每個子工作的門檻值(Avg)，**並與每個子工作的最小執行時間比較，如工作** A1 **則為** 14=24，**執行時間在門檻值範圍內，因此正常運作；**A2 **則為** 12=17，**執行時間在門檻值範圍內，工作亦正常運作，依此類推。**

步驟七：再從 Min-Time **陣列中找最小值，此例中為(**A2, N_{12})，**其對應的工作是** A2，**對應的節點是** N_{12}。**因此把子工作** A2 **交給節點** N_{12} **執行，並將** 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** 3 **所示。**

表 2 工作分配前每個節點執行時間(第一次)

節點 / 子工作	N_{11}	N_{12}	N_{13}	N_{14}	Avg
A1	18	14	38	26	24
A2	14	12	24	18	17
A3	26	18	66	42	41
A4	19	20	24	36	24.75

$$\text{Min-Time}=\begin{bmatrix} A1, N_{12} \\ A2, N_{12} \\ A3, N_{12} \\ A4, N_{11} \end{bmatrix}=\begin{bmatrix} 14 \\ 12 \\ 18 \\ 19 \end{bmatrix} \quad (1)$$

步驟八：從表 3 **得知，由於已經把子工作** A2 **分配給節點** N_{12}，**所以** A2 **會從集合中被刪掉，而節點** N_{12} **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。接著會再選出每個子工作的最小執行時間(**Min-Time)，**並與一開始計算出來的門檻值(**Avg)**作比較，因每個子工作執行時間都在門檻值範圍內，因此正常運作，並形成一個** Min-Time **資源節點集合，如式(**2)**所示。再從** Min-Time **陣列中找出最小值，這裏為(**A1, N_{11})，**其對應的子工作為** A1，**對應的節點為** N_{11}，**因此把子工作** A1 **交給節點** N_{11} **去執行，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(第二次)

節點 / 子工作	N_{11}	N_{13}	N_{14}	Avg
A1	18	38	26	24
A3	26	66	42	41
A4	19	24	36	24.75

$$\text{Min-Time}=\begin{bmatrix} A1, N_{11} \\ A3, N_{11} \\ A4, N_{11} \end{bmatrix}=\begin{bmatrix} 18 \\ 26 \\ 19 \end{bmatrix} \quad (2)$$

步驟九：從表 4 **可知，由於已把子工作** A1 **分配給節點** N_{11}，**所以** A1 **從集合中刪掉，而節點** N_{11} **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。接著繼續選出最小執行時間(**Min-Time)**的子工作，並與一開始計算所得的門檻值(**Avg)**進行比較，因子工作** A4 **的執行時間在門檻值範圍內，因此可正常運作。但子工作** A3 **的執行時間超過門檻值(**42>41)，**因此最小執行時間會暫時被設定為“** 8 **”，表示無法執行，並重新形成一個** Min-Time **資源節點集合，如式(**3)**所示。再從** Min-Time **陣列中找出最小值，這裏為(**A4, N_{13})，**其對應的子工作為** A4，**對應的節點為** N_{13}，**因此把子工作** A4 **交給節點** N_{13} **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(第三次)

節點 / 子工作	N_{13}	N_{14}	Avg
A3	66	42	41
A4	24	36	24.75

$$\text{Min-Time}=\begin{bmatrix} A3, N_{14} \\ A4, N_{13} \end{bmatrix}=\begin{bmatrix} \infty \\ 24 \end{bmatrix} \quad (3)$$

步驟 10：從表 5 得知，由於已把子工作 A4 分配給節點 N_{13}，所以 A4 從集合中刪掉。而因為 A3 的最小執行時間超過門檻值，在步驟 9 中曾暫時被設定為“ 8 ”，此時除將其回覆為原執行時間“ 42 ”外，更讓所有已完成執行的節點重新進入系統中，如表 6 所示。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 N_{12} 執行，即完成子管理者(N_3)的分配工作，其他工作依此類推。

表 5 工作分配前每個節點執行時間(第四次)

節點 / 子工作	N_{14}	Avg
A3	42	41

表 6 工作分配前每個節點執行時間(第五次)

節點 / 子工作	N_{11}	N_{12}	N_{13}	N_{14}	Avg
A3	26	18	66	42	41

$$\text{Min-Time} = [A3, N_{12}] = [18] \qquad (4)$$

本研究所提出的方法，除將子工作依序分配給下一階層，讓每個節點的工作負載儘可能的平衡外，同時利用 LBMM 排程演算法考慮整體的完成時間，改善 OLB 排程演算法未考慮到執行時間的缺點，也改善了 Min-min 排程演算法之負載不平衡的缺點。而在節點能力評估上，本研究利用行動代理人機制，即時掌握節點的資訊，並利用門檻值的設定進行節點能力的篩選，讓每個工作都可以尋找到適合的資源且要被執行的工作也可以在最短的時間內完成執行，有效降低因選擇到不適合的節點導致執行時間延長，並達到資源的最佳配置。

6. 結論及未來工作

在點對點計算的環境下，由於節點分散在各處，且是動態的組成。因此，如何告訴每台電腦其連結的方式，並給予一個拓樸架構與如何有效的利用這些節點資源，是一個值得深入探討的議題。因此本研究依據三階層式網路拓樸的架構，結合 OLB 排程演算法與 LBMM 排程演算法同時利用行動代理人機制收集節點相關資訊以進行工作的配置，透過 OLB 排程演算法讓每個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節點上的執行時間最小，且達到整體的完成時間最小，並透過階層式的架構來設定二階門檻值，即可依工作特性來篩選合適的節點，讓處理忙碌的節點或是執行效率較差的節點無法進入系統執行工作，因此考慮到在異質性環境下，每個節點的能力與其忙碌程度，讓工作除了可以快速的被分配到合適的節點上之外，更可以快速的被執行完成，提升整體系統的效能。

在本研究中，利用三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，實現三階層式點對點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節點處於動態的環境下所面臨的問題，例如，在三階層式的網路架構下，當節點離開或是損壞時，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節點，並達到容錯能力則是未來研究方向之一。

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，2003.

[2] R. Armstrong, D. Hensgen and T. Kidd, “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] T.D. Brauna, et al., “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.

[4] R.F. Freund, et al., “Scheduling Resources in Multi-user, Heterogeneous, Computing Environments with SmartNet,” 7th IEEE Heterogeneous Computing Workshop (HCW'98), pp. 184-99, 1998.

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

[6] Q. Lv, et al., “Search and Replication in Unstructured Peer-to-peer,” Series – Proceeding – Section - Article, ACM Press, pp. 84-95, 2002.

[7] M. Ripeanu, “Peer-to-Peer Architecture Case Study: Gnutalla Network,” Peer-to-Peer Computing Proceeding, pp. 99-100, 2001.

[8] G. Ritchie, and J. Levine, “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.

[9] K. Shin, et al., “Grapes: Topology-based Hierarchical Virtual Network for Peer-to-Peer Lookup Services,” Parallel Processing Workshops Proceedings, pp. 159 -164, 2002.

[10] K.Q., Yan, et al., “A Hybrid Load Balancing Policy underlying Grid Computing Environment,” Computer Standards & Interfaces, Vol. 29, Issue 2, pp. 161-173, 2007.

[11] H. Zhang, et al., “A Case for End System Multicast,” IEEE Journal Selected Areas Communication, Vol. 20, pp. 1456 -1471, 2002.

[12] P2P, http://wiki.csdn.net/index.php/P2P, 2005.