以改良式 Min-Min 排程演算法 提昇三階層式點對點網路拓樸之負載平衡

嚴國慶 王淑卿 陳秀芳
朝陽科技大學企管系 朝陽科技大學資管系 朝陽科技大學資管系
kqyan@cyut.edu.tw scwang@cyut.edu.tw s9514624@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(Peer-to-Peer computing)逐漸成為分散式應用的主流，它不僅可以提供大量的運算資源，協助無法在單一電腦上解決之複雜計算問題之外，更可達到資源有效的整合與共享。但由於點對點計算主要是透過分散的節點(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在不同地理位置上的節點作運算。因此，如何將工作有效的切割並分配到每一個節點上，才不會造成某些節點負擔過重，而某些節點卻是閒置的情況，則是一個值得探討的議題。本研究在一個三階層式點對點網路拓樸架構下，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節點負擔，提供三階層式點對點網路拓樸之負載平衡。

關鍵詞：分散式系統、點對點計算、排程演算法、負載平衡。

Abstract

In recent years, network bandwidth and quality has been drastically improved, even much faster than the enhancement of computer performance. The Peer-to-Peer computing refers to a class of systems and applications that employ distributed resources to perform a function in a decentralized manner. peer-to-peer (P2P) computing is to utilize the computing resources (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 P2P 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, Peer-to-Peer 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(Distribution System)[9,15]。

分散式系統概略可以分為主從式系統(Client/Server System)與點對點計算(Peer-to-Peer computing，簡稱 P2P)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[9,16]。

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[2]。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[1,15,17]。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[1]。

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；第四節說明研究的方法與架構；第五節為結論。

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[2]。在網路架構中，每台電腦(節點)連結的方式，也就是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[2]，分述於以下各小節。

2.1.1 星狀拓樸（Star Topology）

星狀拓樸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(Web server)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，如圖 1 所示。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，如 Napster 即為典型的星狀拓樸網路架構。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[2]。

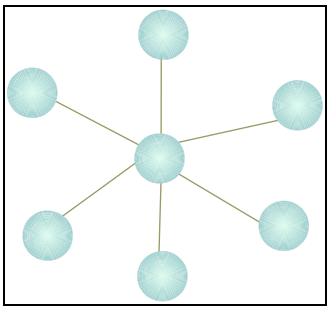

圖 1、星狀拓樸

2.1.2 環狀拓樸（Ring Topology）

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，形成一個環狀拓樸來平衡伺服端的負荷，便可提高服務客戶端的數量，如圖 2 所示。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[7,8]。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。

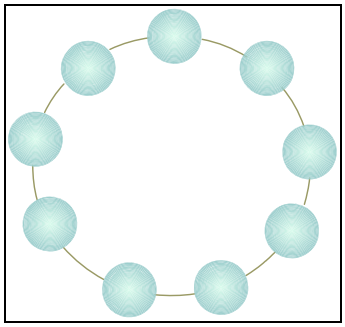

圖 2、環狀拓樸

2.1.3 階層式拓樸（Hierarchical Topology）

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(Tree)的拓樸[11,13,14]。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，圖 3 即為一多階層式拓樸。

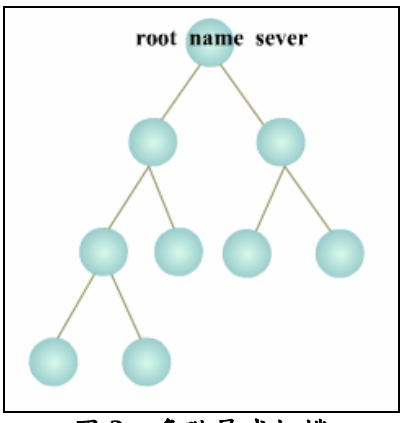

圖 3、多階層式拓樸

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。

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[3-6]。

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[4,7]。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[3,4]。

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[7]，但仍可能有部份的工作無法獲得最小的執行時間(Minimum execution time) [3,4]。

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[1,10]。

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[1,10]。

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(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節點上的完成時間而沒有考慮到每個節點的負載狀況，因此可能造成有些節點總是非常得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。

在此，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 4 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。

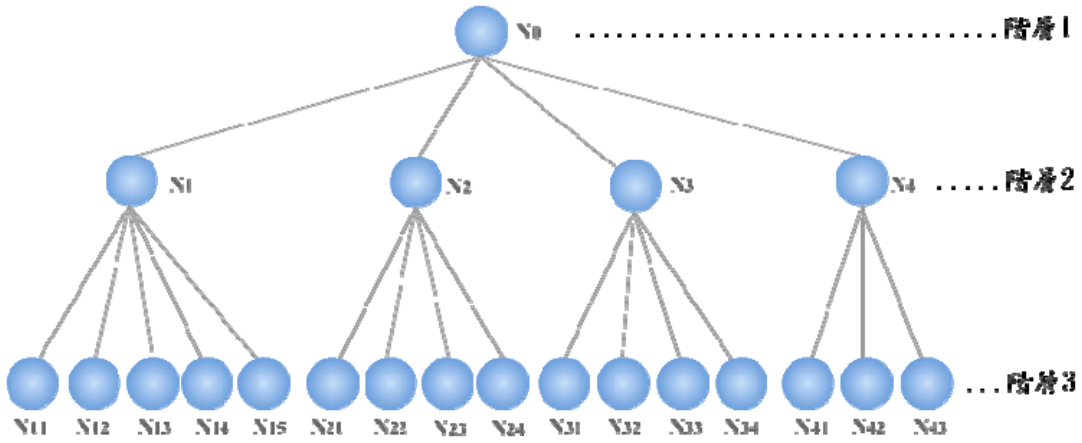

圖 4、三階層式網路拓樸

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 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 5 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	各子管理者所能管控的資源節點個數
M_j	可執行子工作之資源節點集合內節點的總個數，$1 \le j \le y$
α	子工作的編號，$1 \le \alpha \le N_i$
γ	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 個資源節點上找尋可以使用最小執行時間之資源節點，並形成一個 Min-time 資源節點集合。
Step2：	再從 Min-time 資源節點集合中選出其中最小執行時間的節點 γ。
Step3：	將子工作 α 分配給節點 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 的節點 γ 重新排在所有資源節點的最後面。
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 在每個資源節點
                        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 存放資源節點的 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;              /*挑選出之節點γ*/
Subtask = α;          /*被執行之子工作α*/
將 Subtaskα從任務集合中刪除
將 Nodeγ重新排在所以節點的最後面
```

圖 5、LBMM 之演算法

本研究在三階層式點對點網路拓樸架構下，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。

4. 研究方法與架構

依據上述所提到的，本節將詳細的說明如何在三階層式網路拓樸架構中，利用 OLB 排程演算法與本研究所提出之 LBMM 排程演算法，且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節點的負載狀況。

4.1 研究設計

本研究的基本想法是當有工作進入三階層式點對點網路拓樸架構時，根節點(管理者)會收集所有的工作需求並依序存放到佇列中。接著，依據此工作需求的順序分配給下一階層(子管理者)。而當子管理者收到要執行需求之工作時，會將此分配到的工作切割成若干個子工作，並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[12]。
3、 工作具可分割性，且每一個節點皆可以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以進行執行。

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。

若目前有四個工作，分別為 A、B、C、D，需要被執行。本研究所提出之混合式排程法，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，其步驟如下說明：

步驟一：節點 N0 將要執行的工作 A、B、C、及 D 加以收集，並存放到工作佇列中如圖 6，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 6、工作佇列

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，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：當子管理者(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 7 所示，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。

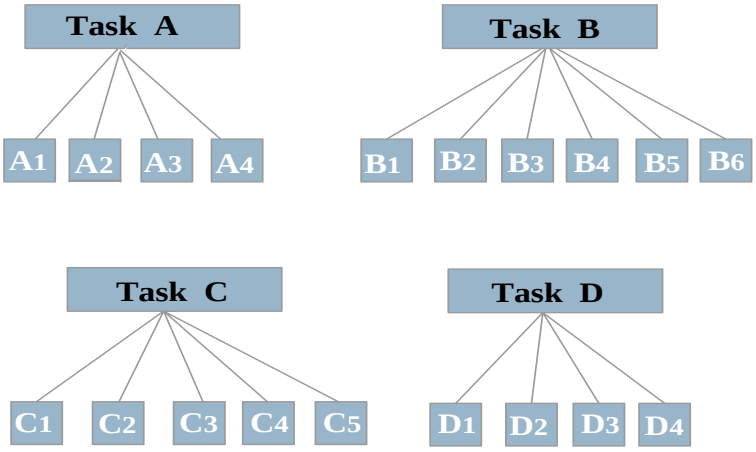

圖 7、將工作切割成子工作

步驟四：利用 LBMM 排程演算法，首先計算出該子工作分配到 M_j 個不同資源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	9	19	18
A3	11	22	14	24	25
A4	10	27	13	21	19

$$\text{Min-Time} = \begin{bmatrix} A1, N13 \end{bmatrix} = \begin{bmatrix} 8 \end{bmatrix} = \begin{bmatrix} A2, N13 \\ A3, N11 \\ A4, N11 \end{bmatrix} = \begin{bmatrix} 9 \\ 11 \\ 10 \end{bmatrix} \quad (1)$$

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 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(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	11	22	24	25
A4	10	27	21	19

$$\text{Min-Time}=\begin{bmatrix}A2,N15\\A3,N11\\A4,N11\end{bmatrix}=\begin{bmatrix}18\\11\\10\end{bmatrix} \quad (2)$$

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 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(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，即完成子管理者(N1)的分配工作，其他管理者依此類推。

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

$$\text{Min-Time} = [A3,N12] = [22] \quad (4)$$

上述方法，除了將子工作依序分配給下一階層，讓每個節點的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。

4.2 研究結果與分析

為驗證本研究所提出的 LBMM 排程演算法的確可以達到更有效的負載平衡狀態，因此在這邊將以原有的 Min-min 排程演算法與本研究所提出之 LBMM 作簡單的分析說明，探討子工作分配到不同節點上之負載狀況與每個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節點的負擔情況，如圖 8 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	21
A4	N11	10

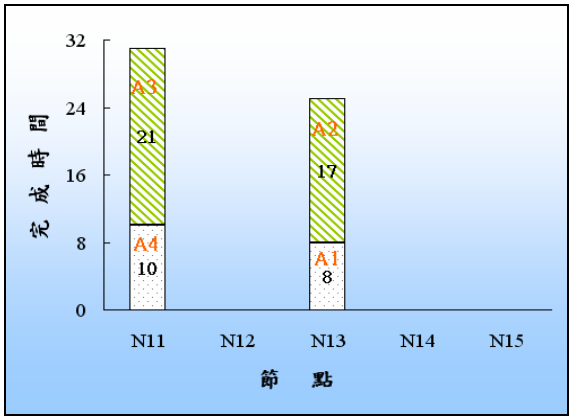

圖 8、使用 Min-min 排程演算法之各節點的負載狀況

由圖 8 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 二個資源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，造成某些節點可以執行工作卻沒有被分配到工作，而某些節點處於忙碌狀態卻又被分配到工作增加其負擔，呈現負載不平衡的情形，使得整體的完成時間增長。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 31 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 9 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

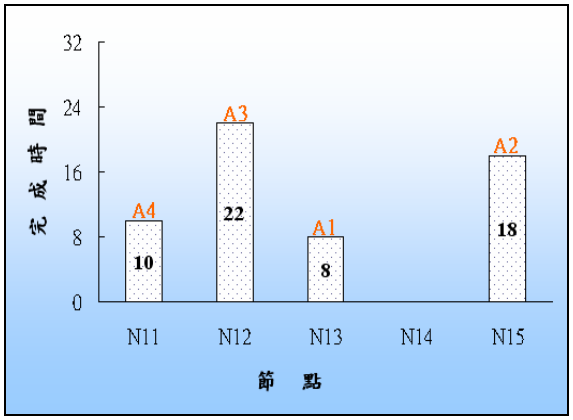

圖 9、使用 LBMM 排程演算法之各節點的負載狀況

由圖 9 中可以看出，採用 LBMM 排程演算法，則每一個節點都可以被分配到一個子工作，不會發生某些節點可以執行工作卻沒有被分配到工作，而某些節點處於忙碌狀態卻又被分配到工作的現象，也因為每個節點都可以平衡的被分配到工作，所以降低了每一個節點的執行時間，使得整體的完成時間也跟著降低。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 31，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，也可以有效的降低工作的完成時間，提升系統整體的效能。

5. 結論及未來工作

在點對點計算的環境下，由於節點都是分散在各處。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，提升整體的效能。

在本研究中，更因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，實現了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] 曾文華、魏天宇，“網格計算環境中任務調度演算法的研究，”廈門大學碩工學位論文，pp. 1-36，2005。

[2] 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。

[3] 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.

[4] 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.

[5] Freund, R.F. and Siegel, H.J., “Heterogeneous Processing,” *IEEE Computer*, 26(6), pp. 13-17, 1993.

[6] 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.

[7] 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.

[8] Lv, Q., Cao, P., Cohen, E., Li, K. and Sponsor, S. S. “Search and Replication in Unstructured Peer-to-peer,” *Series-Proceeding-Section-Article*, ACM Press, pp. 84-95, 2002.

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

[10] 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.

[11] Shin, K., Lee, S., Lim, G., Yoon, H. and Ma, J. S., “Grapes: Topology-based Hierarchical Virtual Network for Peer-to-Peer Lookup Services,” *Parallel Processing Workshops Proceedings*, pp. 159 -164, 2002.

[12] Yan, K. Q., Wang, S.C., Chang, C.P., and Lin, J.S., 2006, “A Hybrid Load Balancing Policy underlying Grid Computing Environment,” *Computer Standards & Interfaces*, Vol. 29, Issue 2, pp. 161-173, February 2007.

[13] Zhang, H., Rao, S.G., Seshan, S. and Chu, Y.H., “A Case for End System Multicast,” *IEEE Journal Selected Areas Communication*, Vol.

20, pp. 1456 -1471, 2002.
[14] Francis, P., Pryadkin, Y., Radoslavov, P., Govindan, R. and Lindell, B., "Yoid Tree Management Protocol (YTMP) Specification," http://www.icir.org/yoid/.
[15] Napster, http://www.napster.com/
[16] P2P, http://wiki.csdn.net/index.php/P2P, 2005.
[17] Peer-to-Peer, http://www.intsci.ac.cn/users/luojw/P2P/ch01.html.