行政院國家科學委員會專題研究計畫　成果報告

協同計算之網路資源管理(3/3)

計畫類別：個別型計畫
計畫編號：NSC92-2213-E-002-006-
執行期間：92 年 08 月 01 日至 93 年 07 月 31 日
執行單位：國立臺灣大學電機工程學系暨研究所

計畫主持人：顏嗣鈞

計畫參與人員：陳建良、 許華祐、 康師誠 、柯信豪

報告類型：完整報告
報告附件：出席國際會議研究心得報告及發表論文
處理方式：本計畫可公開查詢

中　華　民　國 93 年 11 月 1 日

行政院國家科學委員會補助專題研究計畫 **■ 成 果 報 告**
□期中進度報告

協同計算之網路資源管理(3/3)

計畫類別：■ 個別型計畫　　□ 整合型計畫
計畫編號：NSC　92－2213　－E　－002－006　－
執行期間：　　92 年 8 月 1 日 至　　93 年 7 月 31 日

計畫主持人：顏嗣鈞
共同主持人：
計畫參與人員： 陳建良、 許華祐、 康師誠 、柯信豪

成果報告類型(依經費核定清單規定繳交)：□精簡報告　■完整報告

本成果報告包括以下應繳交之附件：
□赴國外出差或研習心得報告一份
□赴大陸地區出差或研習心得報告一份
■出席國際學術會議心得報告及發表之論文各一份
□國際合作研究計畫國外研究報告書一份

處理方式：除產學合作研究計畫、提升產業技術及人才培育研究計畫、列管計畫及下列情形者外，得立即公開查詢
□涉及專利或其他智慧財產權，□一年□二年後可公開查詢

執行單位：國立台灣大學電機工程系

中　華　民　國　　93　年　　10　月　　20　日

中文摘要

本研究的主要目的在發展一套跨資源的分散式資源搜尋與管理系統，並結合刀鋒式伺服器進行實作。在第一年，本團隊完成一些理論上的分析，將網路上資源量化並以 ER 派區網路（ER Petri net）語言來描述網路上各種不同的資源以及資源上的限制 … 等，加上引進了配對演算法（matchmaking）解決一些變動環境極大的網路資源分配問題，網路資源的配置可以藉由配對演算法所造成的配對樹中就可以反映出一些系統相關特性。第二年，本團隊規劃了一個以 Linux 為主軸的系統來模擬協同運算環境，利用修改一些公開共享軟體並將理論部份實做化，完成我們所提出的資源管理服務。第三年，本團隊將原本建構的系統移植到刀鋒式伺服器上，討論如何在上面處理高計算量的工作，以求得較佳效率。我們修改配對演算法，來充分利用刀鋒式伺服器的高速頻寬，減少配對階段鎖耗費的時間與資源。並且採用類似網際網路上避免碰撞時所使用的概念，將每次進行配對、配置的工作以指數方式成長／遞減，並提出一套有效的配置演算法。最後並實際在刀鋒式伺服器上進行實驗，與其他指標實驗（benchmark）比較。

關鍵詞：分散式系統、刀鋒式伺服器、ER 派區網路、配對演算法、配對樹、協同計算。

英文摘要

The main purpose of our project is to develop a distributed resource-discovery and resource-management system. Starting from the first year, we have surveyed and analyzed currently worldwide activities about resource management. By introducing the ER Petri net, we established our system model to further describe the resource language and its limitation. When a job tells its requirement, matchmaking algorithm provides us the way to allocate resources with respect to their different availability. For the second year, we designed and implemented our resource management system architecture on PCs (with OS of RedHat Linux 8.0) to simulate the collaborative environment. We modified some open-source software and algorithms in order to create the system. During the third year, we transfer our resource management system to the blade server. We studied its properties such as high bandwidth, high computing, and parallelism. For better performance, we further re-formulate the matchmaking algorithm. Together with the idea of CSMA/CA, we resulted to a more efficient matchmaking algorithm and tested several benchmarks on the blade server. Conclusion and testing results are all provided in the report.

Keyword: resource management system, ER Petri net, matchmaking algorithm, collaborative environment, blade server, CSMA/CA.

目錄

第一部份、前言

近來由於網路在教育與學術研究的蓬勃發展，出現協同計算的觀念，它與分散式系統的差別在於分散式系統完全地控制系統裡的所有資源，而協同系統卻包含了許多的資源擁有者與使用者，使用者可以在某些規範下尋找並使用屬於或不屬於他的任何資源，這些規範就是本計畫的題目：資源管理。

協同計算（Collaborative Computing）是一個術語，我們將會用它來取代一般傳統群體軟體（groupware）的觀念，而群體軟體就我們所知，是指在非同步方面合作，以電子郵件、討論區及一些相似性的軟體幫助我們以電子手段交換資訊。協同運算的觀念可以將一些網路的資源分享出來給所需要的人使用，而這些資源是可以即時方式所取得。『叢集式處理系統』(Clustering System)，通常指的是把好幾台相同系統的電腦（或不同機型的電腦）用高速網路連結在一起，形成一個大電腦系統。在問題計算量常常非常龐大的今天，如何能將多台電腦串聯一起、共同計算工作已是一項重要課題。在叢集式處理系統的環境底下，現今最新的應用首推以刀鋒式伺服器（Blade Server）架構為主的系統，如圖 1。藉由把所有的伺服器系統的硬體如處理器、記憶體、硬碟機，以及網路連線整合到單一的擴充卡，或者所謂的『刀鋒』(Blade）上。

(A)　　(B)

圖 1、刀鋒式伺服器---（A）刀鋒部分、（B）伺服器主機。

分散式運算已行之有年，也能夠順利的將來自不同地方的相異資源做有效的分配、利用。刀鋒式系統的資源則多半屬於同質性（homogeneous)，甚至是完全相同。因此可以省去許多花費在以特定語言去描述、定義資源的功夫，而將節省下來的時間拿來進行工作運算本身；在演算法設計上，也因為各個資源並沒有太大本質上的差異，故可以簡化許多變數、以更簡單的程序而獲得更佳的結果。再者，刀鋒式伺服器其內部相互傳輸速度，比一般的網際網路速度高上許多（1Gbps)，而且沒有斷線或選擇路線（routing）的問題；有此優勢，所有因訊

息傳遞所造成的額外支出均可降低至被忽略的程度。除此之外，更可利用刀鋒式伺服器這項特有的優勢，對演算法及工作排程和工作分配上做一些修改，以充分利用資源、增加工作效率。舉凡如資源性質的異同性，聯繫各資源之間的網路型態、速度，皆指出在刀鋒式伺服器的管理上，其排程（scheduling)、配對（matchmaking)、配置（allocation）演算法均可再由分散式運算的現有方法再加以改進。

協同計算體現了網路科技的最大願景，那就是：『更緊密地結合計算平台，更緊密地結合人們』。網路上有著數以萬計的電腦具備著形形色色的軟硬體平台，從國家實驗室的巨大平行電腦到南極探險隊拿在手上的 PDA，從最普遍的視窗作業環境到特別用途的衛星作業系統，複雜、多樣並且分佈廣泛的資源，使得資源管理成為協同計算環境中非常重要的一項核心技術，好的資源管理技術更可以倍數地提升協同計算系統的可用性、即時調整的彈性、可靠度、執行效率與能力。

第二部份、研究目的

本研究的主要目的在發展一套跨資源的分散式資源搜尋與管理系統，並結合刀鋒式伺服器進行實作。在第一年，本團隊完成一些理論上的分析，將網路上資源量化並以 ER 派區網路（ER Petri net）語言來描述網路上各種不同的資源以及資源上的限制…等，加上引進了配對演算法（matchmaking）解決一些變動環境極大的網路資源分配問題，網路資源的配置可以藉由配對演算法所造成的配對樹中就可以反映出一些系統相關特性。第二年，本團隊規劃了一個以 Linux 為主軸的系統來模擬協同運算環境，利用修改一些公開共享軟體並將理論部份實做化，完成我們所提出的資源管理服務。第三年，本團隊將原本建構的系統移植到刀鋒式伺服器上，討論如何在上面處理高計算量的工作，以求得較佳效率。我們修改配對演算法，來充分利用刀鋒式伺服器的高速頻寬，減少配對階段鎖耗費的時間與資源。並且採用類似網際網路上避免碰撞時所使用的概念，將每次進行配對、配置的工作以指數方式成長／遞減，並提出一套有效的配置演算法。最後並實際在刀鋒式伺服器上進行實驗，與其他指標實驗（benchmark）比較。

本團隊針對工作排程與配置進行探討研究，提出一套演算法來處理刀鋒式伺服器在處理高計算量的工作時的系統運作（圖 2），並且將各部分元件的功能加以整合，形成完整的輸入輸出架構。本團隊在研究方法上包含下列幾個重點：提出一套針對刀鋒式伺服器所設計，將排程、配對、配置三者整合的演算法，探討其實作方面的考量，同時舉實例示範該演算法執行過程，最後並分析其複雜度。在實作部分，將對於刀鋒式伺服器如何接收工作輸入、對工作須如何描述，對工作進行處理的流程，其間如何與資料庫、PVM、以及其他的程式、服務相互溝通，一直到工作輸出結果，對整體系統架構及流程做個完整的說明，並舉出實際應用的例子以茲說明（以矩陣 LU 分解和一維熱擴散方程式），以印證基礎理論以及演算法分析。最後提出結論和未來工作展望。

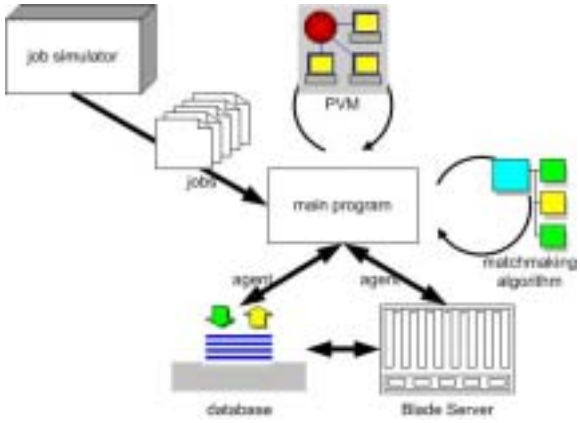


圖 2、系統運作示意圖。

第三部份、文獻探討

1、派區網路

派區網路（Petri nets）是一種數學工具，兼有圖形模型與數學模型兩種功能，常用來描述圖論中狀態與事件之間的關係。在離散動態事件系統中，派區網路更是一個非常方便轉換工具，可將許多實際系統中的複雜性質加以化簡，更進一步利用當中已知的數學特性，瞭解系統的可能變化。派區網路可以視為一種特殊形式的二分式有向圖（bipartite directed graph），主要由三種物件所構成：狀態（place）、轉換（transition）、方向（directed arc）。一個基本的派區網路可以藉由上述三種物件所形成，但是，考慮到更深層實際系統的動態行為，我們引進第四個物件：標記（token），標記的存在與否，代表著在一個狀態中是否具備有相應的條件存在。以下是一個派區網路定義：

***Definition 1**: A Petri net* is a 5-tuple PN = （ P, T, I, O, M_0 ）

- $P = \{ p_1, p_2, \ldots, p_m \}$ is a finite set of places
- $T = \{ t_1, t_2, \ldots, t_n \}$ is a finite set of transitions
- $I: （ P \times T ） \rightarrow N$ is an input function that defines directed arcs from places to transitions
- $O: （ T \times P ） \rightarrow N$ is an output function that defines directed arcs from transitions to places
- $M_o : P \rightarrow N$ is the initial marking; $P \cup T \neq \phi$, $P \cap T = \phi$, and N is a set of nonnegative integers

可到達性（reachability）是一種研究離散動態事件系統的基礎性質，直觀上來說，可到達性反映出系統當中是否具備有某種特定形式狀態或行為，跟一開始系統上的標記分佈（marking）有關。每當一個轉換發生的時候，標記隨之變化，標記分佈也跟著轉變。換言之，在一連串時序性轉換發生後，標記分佈最終也會到達某種分佈狀態。我們說，一個標記分佈 M_i 具有可到達性，代表系統可由初始標記分佈（initial marking）M_0 開始，藉由一連串時序性轉換發生，導引派區網路由 M_0 到 M_i。

我們雖然可以用派區網路描述一個離散動態事件系統，但是當系統具備有時間因子的時候，普通的派區網路便顯得有些功力不足，高階的派區網路便在文獻中廣為討論。在諸多高階的派區網路中，ER 派區網路（ER Petri net）是一個

相當方便描述一個離散動態事件系統具備有時間因子導向的派區網路，它提供了對離散動態事件系統更強的模型化與分析能力，詳細的 ER 派區網路可在參考文獻當中找到。

2、配對演算法

配對（圖 3）的基本觀念很簡單，就是在一群提供服務者（service provider）與使用服務者（customer）當中，利用配對者（matchmaker），建構一個有效的“配對”機制，讓正確的提供服務者能找到正確的使用服務者。一般配對的機制，主要分成幾個部分：

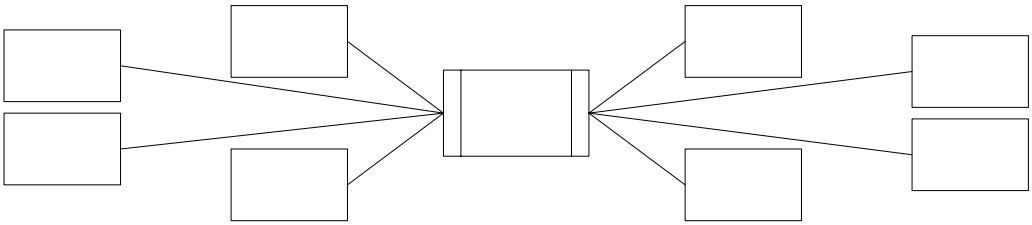


圖 3、配對概念圖。

（1）描述語言（description language）

一個用來指定、形容、描述所有參與配對者（包含提供服務者和使用服務者）的特質（characteristic）、限制（constrains）、以及優先順序（preference）的語言。例如分類式廣告（classified advertisement，classad）就是一種常用的描述語言。

在特質部分，用來敘述本身的性質，例如每一片刀鋒（blade）的特質可能包括它的ＣＰＵ時脈、記憶體大小、硬碟大小、作業平台、軟體套件等等。在限制部分，用來描述對於要和自己配對者的要求。例如每一片刀鋒的限制可能包括：只有當負載（load）小於一定門檻的時候才接受新的工作、只接受記憶體需求在 128MB 以下的工作。優先順序甚至可以指定只接受來自特定客戶端（或客戶群）的工作。下頁圖 4 為 classad 的一個例子。

```
[
Type        =  “Blade”
Disk        =  “40000”              //MByte
Memory      =  “1024”               //MByte
Arch        =  “INTEL”
OS          =  “LINUX”
MIPS        =  “4800”
Rank        =  “1”
Name        =  “192.168.10.25”
LoadAvg     =  “0.0”
Constrains  =  “NULL”
]
```

圖 4、classad。

(2) 配對協定 (Matchmaker protocol)

配對協定包含了公佈協定(publish protocol)以及通知協定(notification protocol)。公佈協定用來規定代理人（agents）如何跟配對者（matchmaker）溝通、發出廣告，通知協定則規定了如何接收通知。配對者完成一組配對之後，會依照通知協定去通知雙方相關資訊。

(3) 配對演算法 (Matchmaking algorithm)

配對演算法是配對者（matchmaker）用來進行、造成配對的工具。簡單說，配對者將送來的 classad 中，依演算法找出其中相關的屬性（attribute）內容來進行配對。在這個過程中，演算法會將不同的屬性賦予不同的意義，以達成某些特殊目的。例如將屬性當中的 Constrains，其意義是對於配對的對象加以限制；而 Rank 則可以讓不同的配對對象有不同的優先權。圖 5 為配對演算法示意圖。

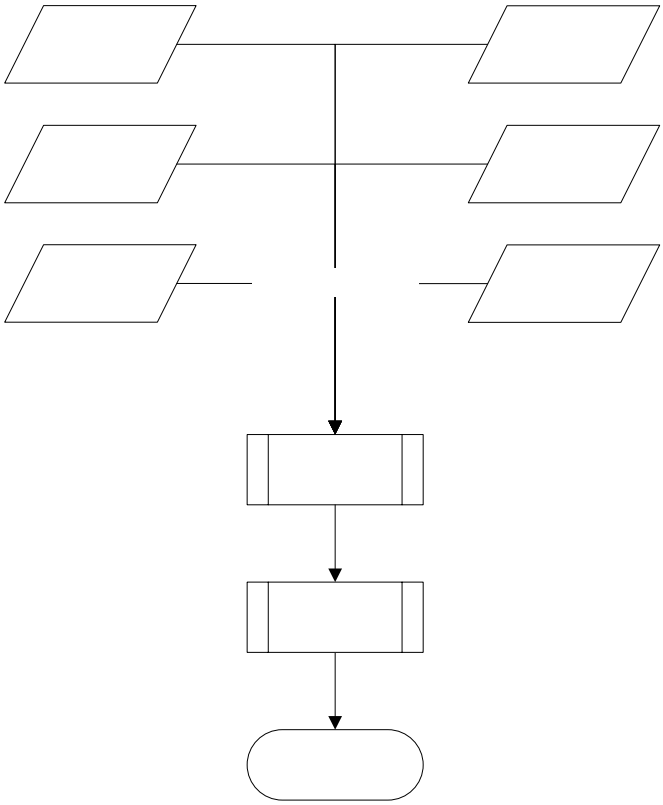


圖 5、配對演算法示意圖。

(4) 要求協定 (Claiming Protocol)

要求協定一般是用來規定被配對到的兩方要如何溝通、開始建立工作關係。使用要求協定的動機之一，是配對不等於配置（allocation)。將 A 和 B 兩方配對在一起之後，只代表 A 可以被配置給 B。A 和 B 將會在要求協定之下去驗證對方的狀態以及詳細狀況，來決定是否真正的開始合作。圖 6 為要求協定操作的示意流程圖。

使用要求協定的好處，是可以使得各個資源的狀態不需要常常更新，可以節省訊息傳遞的額外開支 (overhead)。伺服器端不需要擁有各個資源的最新狀態，因為配對之後，雙方會再度比對當時的最新狀況。然而這些優勢在刀鋒式伺服器的高速頻寬之下，顯得不再像原本的那樣有效果。事實上，為了充分利用刀鋒式伺服器的高速頻寬，反而應該在不太耗費多餘資源的前提下，盡量維持各資源的最新狀態，減少最後仍舊失敗的配對。此外，略過要求協定還可以少去認證（authentication）的步驟。以往每一次配對成功之後，雙方尚需要再度認證；而刀鋒式伺服器僅需要於傳送工作時，伺服器與傳送者進行認證即可，因為各片刀鋒是在內部的子網路運作，伺服器與各片刀鋒不需認再證一次。

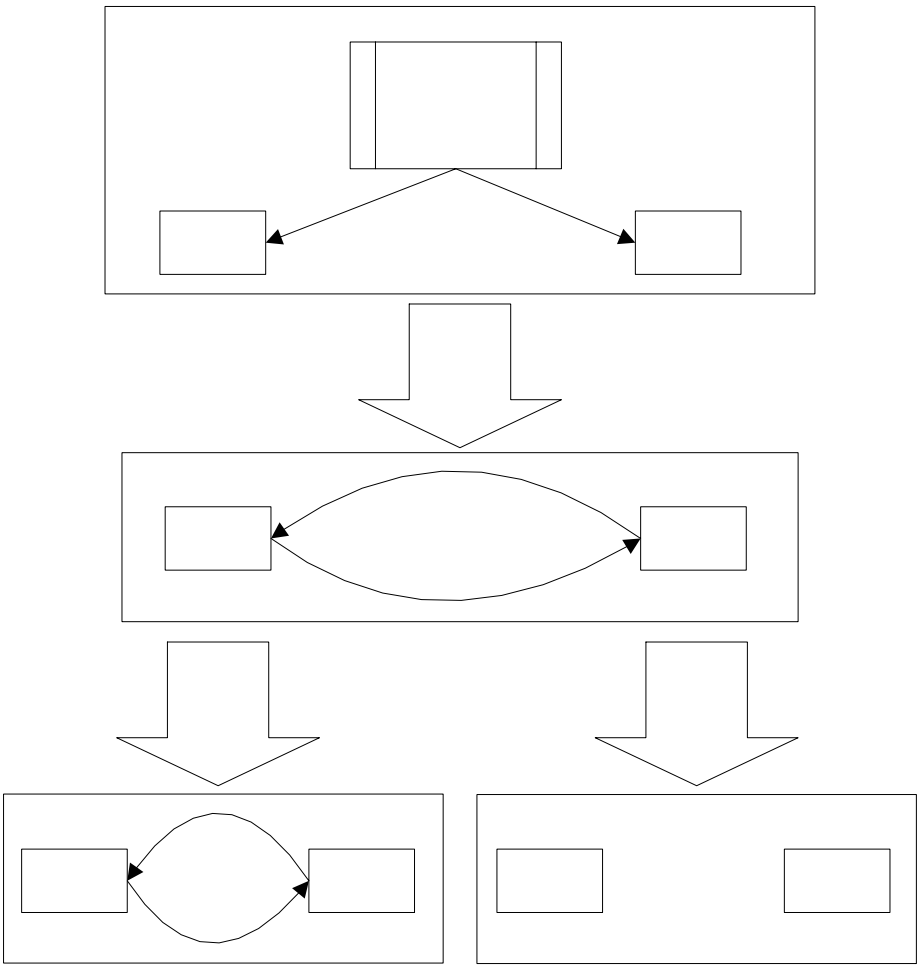


圖 6、要求協定操作示意圖。

3、Amdahl’s Law

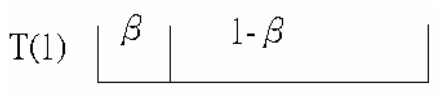
Amdahl's Law 是一個在使用多顆ＣＰＵ或多台機器去平行運算、解決問題時，速度上比單獨使用一顆ＣＰＵ去執行，增加速度成效的定律。一個程式的速度是它所花費在執行上的時間，這可以隨著時間被量測出來。速度增加率 *S* 的定義就是把該程式以單獨一顆ＣＰＵ去連續性（sequential）地執行所花費的時間，除以該程式以多顆ＣＰＵ去平行（parallel）地執行所花費的時間：

$$\text{Speedup（速度增加率）：}\ S = \frac{T(1)}{T(k)}$$

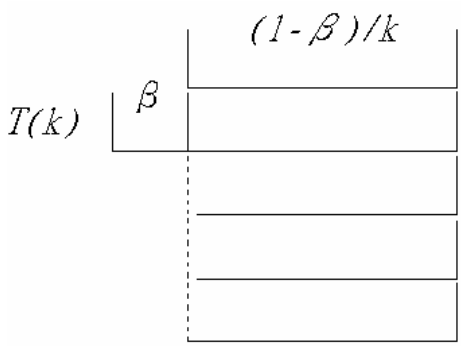
其中 *T*（1）表示僅用一顆ＣＰＵ所花的時間，*T*（k）則表示以 *k* 個ＣＰＵ去進行運算所花的時間。效率就是將 *S* 除以所使用的ＣＰＵ數量 *k*。

一個工作或一個程式，通常可以區分為兩部分：連續性部分和平行性部分。連續性部分一次僅能由單顆ＣＰＵ執行。假設某個工作由單一ＣＰＵ執行時要花費 t，其中連續性部分佔的比例是 β，則使用 k 顆ＣＰＵ的去執行該工作要花費的時間會是 $\beta t+\frac{(1-\beta)t}{k}$，得 $S=\frac{k}{\beta k+(1-\beta)}$。

單一ＣＰＵ



多顆ＣＰＵ



因此，對同一件工作，持續增加參與運算的ＣＰＵ數目並不能一直有效地減少執行時間，反倒只是減低效率。為了要減小 β 值、降低連續性部分所佔的比例，我們可以：對演算法作最佳化，增加網路頻寬、減少需要傳遞的資料量、降低訊息傳遞時間，增大工作的內容量，將延遲（latency）最小化、或是增加總產量（throughput）。

第四部份、研究方法

1、演算法介紹

本章依照演算法流程，分三部份來說明：首先在 1.1 說明計劃/排程（Scheduling），其重點在於工作取出量（Job Fetching Window Size）的決定；接著 1.2 探討配對（Matchmaking）；1.3 則是著重於配置（Allocation）的決定和演算法如何去選擇配置組合的過程與分析。本演算法主要針對刀鋒式伺服器在處理高計算量的工作之需所設計。

1.1 計劃/排程（Scheduling）

1.1.1 工作佇列（Job Queue）

在伺服器啟動之時，同時也準備了一個佇列來接收、暫存工作。當工作被客戶端送到刀鋒式伺服器、要求運算的時候，都先被放置在佇列中等候處理、配置。若佇列溢滿時，則表示該刀鋒式伺服器已負荷量過大，或工作量的短期爆量過大。而佇列區的大小，可參考伺服器以往工作狀態，或是工作來源的情況再做決定。定得太大除了浪費儲存空間之外，工作可能等待時間過長而失去時效性；若能在第一時間就回報客戶端，表示刀鋒式伺服器的佇列已滿，則客戶端尚可另覓他法去處理。若定得太小，則佇列溢滿與佇列被掏空的狀態發生頻率都會太高，容易發生先是工作大量被忽略，隨後卻又發生佇列區無工作、資源閒置的窘境。一般而言，我們可以將客戶端所能接受的合理等待時間，乘上以目前資源去估計所能負荷的工作處理速率，來求得較佳的佇列區大小。

1.1.2 等待週期（Job Fetching Interval）

在實作上，由於必須取得各個客戶端的即時資訊，例如ＣＰＵ使用率、記憶體使用率、硬碟可用空間等等，故至少須經過一段適當長度的等待週期。若週期過短，容易發生數據統計錯誤的問題。例如原本閒置的ＣＰＵ，在剛剛接收到一個工作之時，雖然馬上就開始進行運算，但其ＣＰＵ使用率的統計並不會立刻就變成百分之百，而是視其工作平台的統計更新速率。以 Linux RedHat 為例，大約需要五至十秒鐘，ＣＰＵ使用率才會漸漸由零上升至百分之百。除了硬體上之限制，等待週期的大小也會對演算法執行成效有所影響。我們將在詳細介紹配置（allocation）的演算法之後，再來探討這個議題。

1.1.3 工作取出量（Job Fetching Window Size）

故經過一段適當長度的等待週期之後，由佇列中取出若干個工作來進行配對以及配置的動作。工作取出量若太小，則容易發生有資源被閒置，降低工作效率的現象；工作取出量若太大，除了增加演算法計算的時間之外，工作彼此之間競爭資源的現象也更明顯，容易造成飢餓（starvation），或者為了避免這些現象而增加一些額外支出（overhead）。工作取出量的詳細影響將在後面進一步探討。因此，如果能夠決定一個適當的工作取出量，對於增進整體的工作效率會有相當大的幫助。這裡我們採用網際網路上所使用的一個概念，來決定取出的工作數量：以指數成長的方式，來動態改變工作取出量。

工作取出量的改變觀念如下：假設工作取出量為m，意即從佇列中取出前m個工作來進行配對與配置。若此m個工作均能夠順利被配置到資源、進行運算，除非全部資源均已配置，否則即表示目前的工作取出量尚無法充分利用所有的資源，也就是仍舊有資源浪費的現象產生。因此必須增加工作取出量。這裡我們直接將工作取出量增加兩倍。若此m個工作，即使在最佳的配置情況下，仍有超過一半的工作無法被執行運算，那表示目前的工作取出量已超過所有資源總共能負荷的量的兩倍以上。此時我們將工作取出量減半。若被已被配置到資源的工作數量，佔工作取出量的 50%~100%，則表示目前的工作取出量對現有資源而言差不多就是其負荷量。而那些在此次配對中無法獲得足夠資源的工作，將被重新放回佇列中；但不是放至佇列尾端重新排隊，而是依照這些工作在被取出之前的相對先後順序，再度放回佇列的前面。如此可避免工作因為一次配對失敗，就必須等待不合理的長時間。圖 7 為工作取出量之變化示意圖。

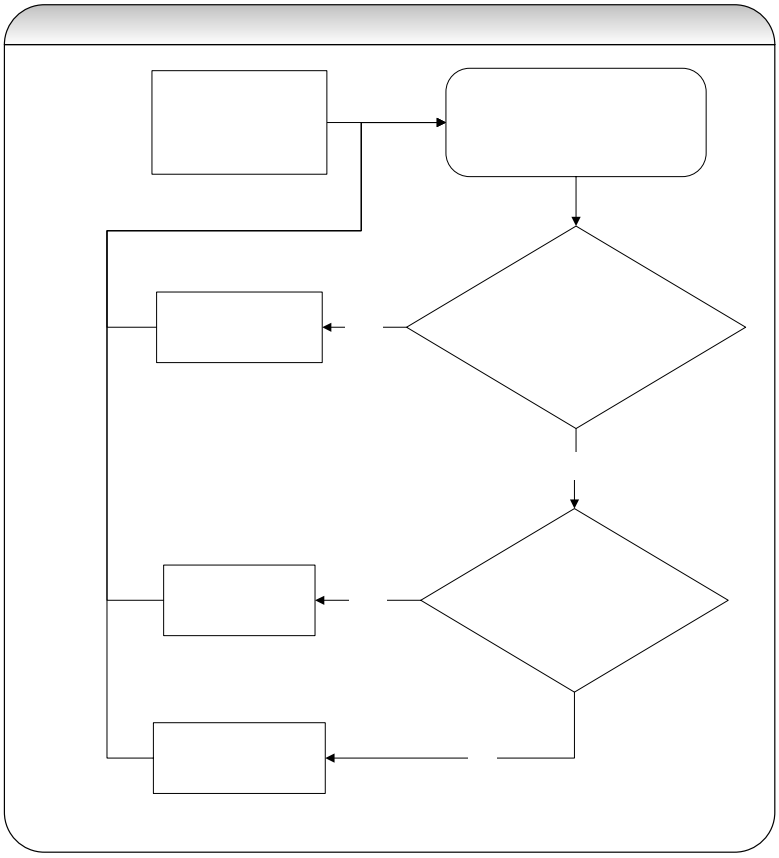


圖 7、工作取出量之變化示意圖。

1.2 配對 (Matchmaking)

1.2.1 要求條件

在配對階段，將以各個工作所要求的條件為根據，來尋找能夠符合該工作需求的資源。一般常見的要求條件，不外乎：ＣＰＵ時脈、ＣＰＵ使用率、平均負載量、可用之記憶體大小、硬碟可用空間大小、作業平台、軟體套件，以及所需要的資源數目（ＣＰＵ數目）等等。在刀鋒式伺服器的環境下，最需要考慮的就是ＣＰＵ使用率，以及記憶體大小。

1.2.2 蒐集資訊

實作上，主伺服器會要求各客戶端，每隔一定的週期就將自身的各種資源狀態等資訊，傳送至主伺服器，再將這些資訊儲存在資料庫中以方便查詢。當各工作開始配對時，僅需要在主伺服器上進行查詢的動作，將符合該工作需求的所有資源都找出並記錄下來。如此可避免每次配對都需要多花費時間去等待客戶端蒐集即時資訊，同時也可生成各客戶端的狀態紀錄。

配對階段完成之時，每一個工作都已經找到所有能夠滿足其要求條件的資源。至於要使用這些資源當中的哪些—或者是全部—來進行運算，則是在配置階段（allocation phase）才進行選擇、決定。圖 8 為簡單的配對演算法概念。

```
Receiving clients status every Info Fetching Interval
for each job
  for each resource
    if  (the resource meet the job's request)
      marked it in a matching-table
```

圖 8、配對演算法。

1.3 配置（Allocation）

進入配置階段之時，最需要留意的是各工作要求的資源數量（number of requested resources，RR）和配對資源數量（matched resources，MR)。每次配置時，可能的選擇方法都有很多種組合。我們以求出一個能夠滿足、執行最多工作數的配置組合為目標。

定義 1 （可滿足性、可配置性）

可滿足性（Satisfiability，S_i）：

各個工作的可滿足性為『配對資源數量－需求資源數量』，S_i = MR-RR。

可配置性（Allocatability，A_i）：

資源 R_i 的可配置性為『所有與該資源配對的工作（matched jobs）的 S_i 總和』。

$$A_i = \sum_{\{k \mid J_k, R_i \text{ matched}\}} S_k$$

1.3.1 資源配置（Resource Allocation）的選擇方法

進行配置的先後，依照各工作的需求資源數量（RR）大小來決定，RR 較小者優先進行配置的動作。但 S_i 小於零的工作則表示此輪配置無論如何無法滿足該工作，故不予考慮、將資源分配給其餘工作以期能滿足最多工作數目。若各個工作有優先權，則依照優先權來決定配置順序，優先權相同時才依照 RR 考慮順序。若有多個工作，其 RR 均相同，則以 S_i 較小者優先。若 RR、S_i 均相同，則以『與該工作所有 matched 之資源的 A_i 總和』來決定順序。如果以上三條件均無法決定出先後順序，則依照工作原本在佇列中的順序進行配置。以下頁圖 9 為例，Job 2（2）表示 Job 2 的 RR 為 2，因此在此八個 Job 中，Job 2，6，7，8 這四個 RR 最小的工作將先被挑出。下一步依 S_i 挑出 Job 2 和 6，最後比較 A_i 總和：Job 6（8+1）＜Job 2（11+5）而決定第一個配置 Job 6。

Jobs (RR)	Job 1 (4)	Job 2 (2)	Job 3 (4)	Job 4 (5)	Job 5 (4)	Job 6 (2)	Job 7 (2)	Job 8 (2)	
S_i	1	0	2	3	1	0	1	5	A_i
Node 1			matched		matched				3
Node 2	matched		matched		matched			matched	9
Node 3	matched		matched	matched	matched			matched	12
Node 4	matched				matched				2
Node 5	matched	matched	matched	matched				matched	11
Node 6		matched	matched	matched					5
Node 7				matched		matched		matched	8
Node 8				matched			matched	matched	9
Node 9			matched				matched		3
Node 10				matched				matched	8
Node 11	matched			matched	matched				5
Node 12						matched	matched		1
Node 13				matched				matched	8

圖 9、資源分配舉例。

當一個工作開始進行配置時，演算法必須從該工作配對（matched）的資源當中，選擇出 RR 個資源來配置給此工作。選擇的依據是按照 A_i 來選擇。A_i 小的資源會優先被選擇、配置。以圖 9 為例，假設目前正在配置 Job 4，則 Node 6, 7, 10, 11, 13 會被選擇、配置給 Job 4。若 A_i 相同時，則依照資源編號順序選擇。圖 10 則為決定工作配置先後順序的示意圖。

完成一個工作的配置之後，即重複上述兩步驟，選擇出下一個要配置的工作並選出該工作將使用的資源。當此輪工作取出量內的所有工作，均已被配置或 S_i 小於零時，此時演算法已求出一組配置方法。可以開始將工作指令傳送到各個資源去執行，同時開始等待週期；或者繼續求其他的配置組合以找出最佳解。

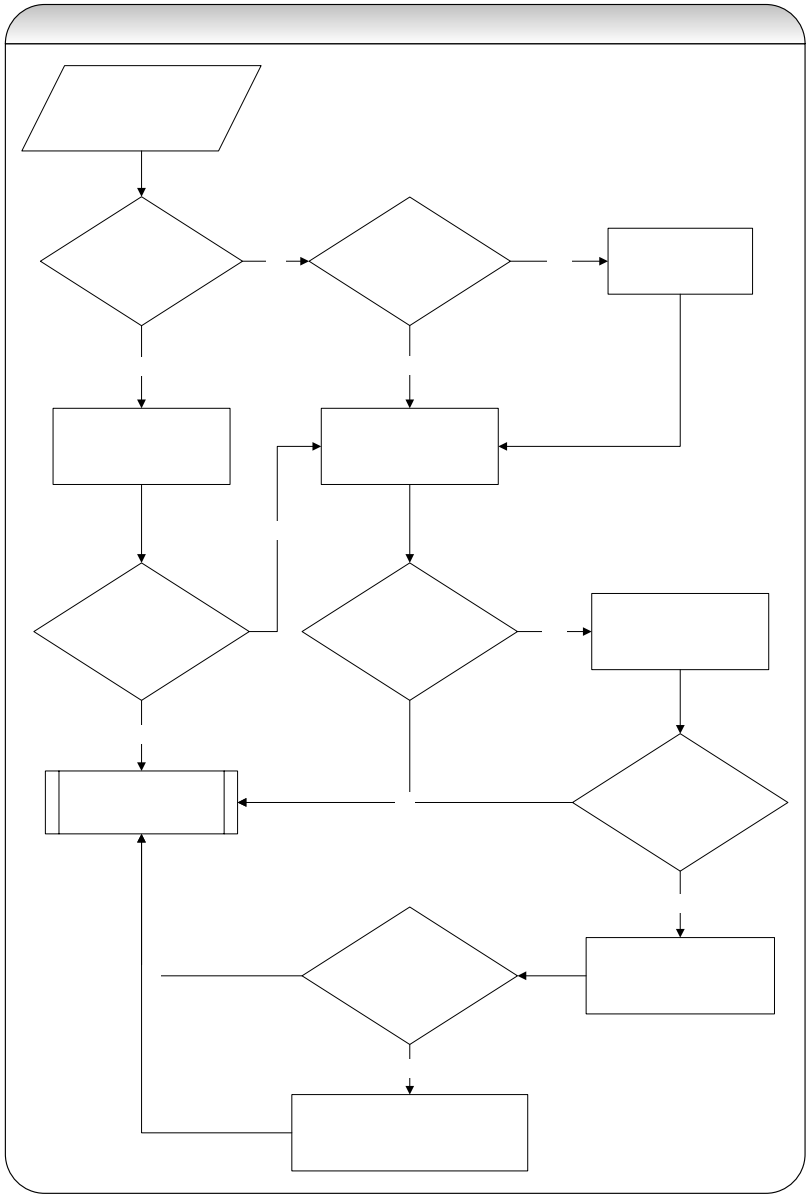


圖 10、工作配置先後順序的示意圖。

1.3.2 依實際需要決定不同的演算法抉擇

在此m個工作中，如果有上一次配置或前幾次配置時配置失敗的工作，將優先被考量。隨著配置失敗次數增加，其被考量的優先權也會增加。有些工作可能較為特殊，需要特別大量（可能是全部）的資源才能滿足其需求；此類工作一旦進入佇列，除非當時工作數目較少，否則相當容易配置失敗，幾次下來其優先權會變成最高。但是每次進入配置階段時，可用的資源數目並不是固定的、並不一定是全部，有些資源可能仍在執行之前分派給它的運算。也就是說，即使擁有了該次配置的最高優先權，即使可以將此次配置階段所有的資源都納為己用，也還是有可能無法滿足該工作之最低需求。此時有兩種方法可供選擇：

A．盡量滿足該工作之需求。也就是說，寧可將目前的資源先暫時閒置著，等待其他的資源完成目前的運算工作、成為可用資源之後，再將這些資源一起配置給這類需求量特別大的工作。此法的明顯缺點就是資源必須閒置等待，且等待時間無法保證長短。

B．忽略此類工作。當這類工作因為要求的資源太過龐大，以至於經過等待一定次數的配置之後都無法成功被滿足時，表示除非採用 A.法，否則目前可用資源無法滿足該工作。為了增加整體效率、不影響其他工作等待時間，要求資源太大的工作若無法在一定回合內被成功的配置到資源，將被忽略、自佇列中刪去（並通知客戶端，告知此工作已接近此台刀鋒式伺服器之所能提供之服務極限）。此法缺點在於雖然實際上總體資源是可以執行一些工作，但卻可能因為佇列中的工作數目增加，而將這類工作忽略。至於要採用哪一種方法，則視實際需要或偏好來決定。

1.3.3 找出最佳解（滿足最多工作數）

依照上面的演算法步驟所求出的解，不一定保證是最佳解、能夠滿足最多的工作數。在 worst case 裡頭，我們的演算法跟暴力法是一樣的、必須將所有組合都計算出來，才能找到最佳解；但是在其他的案例裡，除非最佳解裡頭剛好包含的都是 RR 最大（或較大）的工作，否則演算法在選擇不同配置組合的過程中，在已經找到最佳解之後的嘗試，大都不必執行到最後就會被跳過，尤其離最佳解越遠（即滿足工作數量和最佳解比起來少很多）的組合方法，由於會使得很多工作的 S_i 小於零，而當 S_i 小於零的工作數量超過最佳解時，這次的嘗試就會被跳過。

整體而言，暴力法較本演算法的優點在於每種配置組合方式都不需要多花時間去計算求得，缺點就是每一種組合都必須從頭跑到尾。若各個工作的需求差異不小，或者各個資源之間的相異性頗多的情況下，會使得各工作的 RR 和 S_i、各資源的 A_i 數字上都有一定程度的差距，分布上也不會很類似。在這種情況下，本演算法比暴力法的優勢會更明顯。

1.3.4 避免飢餓（Starvation）

然而，為了避免飢餓產生，上述措施仍舊不足。因此我們引進優先權（priority）的觀念。

1.3.5 演算法分析

在本節開始探討之前，先將探討的範圍區分為三種：在單位時間內：各客戶端送來各個工作，設其 RR 總和為『總需求量』(Total requests，TR)；完成工作、再度成為可用狀態的資源個數令其為『總釋放量』(Free resources，FR)。
第一種情況（Case 1)：總需求量大於總釋放量，TR＞FR。
第二種情況（Case 2)：總需求量趨近於總釋放量，TR　FR。
第三種情況（Case 3)：總需求量小於總釋放量，TR＜FR。

(1)　Case 1（TR＞FR)：

在這種情況下，可用資源總量並不足以滿足所有的工作需求，勢必有些工作會配置失敗、必須放回佇列，同時增加其優先權。長期處於 case 1，會使得這種具有高優先權工作的數量增加，致使資源分配上變成變相的 FCFS：一開始配置失敗的工作，其優先權逐漸增加；同時其他配置成功的工作開始執行。接下來的配置階段，這些優先權較高的工作，又會導致新被納入工作取出量之內的工作配置失敗（因為資源都已優先被配置給前面優先權較高的工作了)。如此惡性循環，每個工作都必須在進入工作取出量的範圍內之後，幾乎都必須等待相當周期之後才有機會拿到足夠的資源。大部分的資源都是被高優先權的工作所『保留』，等最久的最優先保留，然後依照等待時間長短（優先權）進行保留。所謂『保留』意思就是即使配置給那些工作，由於資源尚未到達該工作之 RR，這些資源仍舊處於閒置狀態。所以即使這些資源可以集中分配給某些工作、使之能夠配置成功開始進行運算，但由於工作取出量太大，使得資源大部分被浪費。如果為了避免資源浪費而取消優先權，卻又會發生飢餓（starvation）的問題。

(2)　Case 2（TR　FR)

這是最佳的狀況，我們採用會變化的工作取出量，目的就是能夠盡量處於這種狀況。然而實際情況的 TR、FR 都會隨著時間不斷改變，固定工作取出量並不能保證長期都能處於這種供需平衡的狀態。若工作型態改變或是有資源退出時，工作取出量可能都需要調整。

(3)　Case 3（TR＜FR)

這種情況表示 X 太小，使得可用資源在滿足所有 X 個工作之後，仍舊有多於資源閒置。長期處於 case 3，表示一直都有資源沒有被利用—即使這些資源可能可以滿足其他還在佇列裡等候的工作。這樣很明顯的降低了整體的工作效率以及資源使用率，並不是一個好的選擇。

1.4. 時間複雜度（Time Complexity）

假設此次從佇列中取出、進行處理的工作數量為m，資源總數為ｎ。在排程階段的動作，都可在常數時間之內完成，其複雜度為Ｏ（１）。在配對階段，建置配對表格需要Ｏ（ｍｎ）的時間。在配置階段，要選出每一種配置組合，選出 RR 最小的工作需要Ｏ（m），之後要選出 S_i 最小者也需要Ｏ（m），接著選 A_i 總和最小的工作需要Ｏ（m），以上總計仍舊是Ｏ（m）。選出要配置的工作之後，從其配對（matched）的資源當中選出 RR 個 A_i 最小的工作來進行配置，平均需要Ｏ（ｎｌｏｇｎ）的時間，因為與之配對的資源數量並無上限、且各工作之間可重複配對同樣資源，故最多可以有Ｏ（ｎ）個資源與該工作配對。而以 Quick sort 去將ｎ個資源依 A_i 排序要Ｏ（ｎｌｏｇｎ）。選出要配置給該工作的資源之後，要再去修改配對表格，這個修改表格的步驟，在每一種配置組合（allocation-combinations）之中，總共要花去Ｏ（ｍｎ）：每個工作要修正的資源數量是『配對資源數量（matched resources，MR)』個，而每個資源要修正的 S_i 與 A_i，已被配置的資源需要修改所有與此資源配對的工作之 S_i，這最多會花掉Ｏ（m）個動作；沒有被配置的資源則需要修改 A_i，這需要Ｏ（１）的時間。由於每個資源在一種配置選擇之下，只會配置給一個工作，所以修改的 S_i 動作至多執行ｎ次，時間總共最多花費Ｏ（m） ｘｎ=Ｏ（ｍｎ）。每個資源修改 A_i 的次數至多m次（因為共m個工作進行配置，每個工作至多一次），故修改 A_i 的時間最多也是Ｏ（ｍｎ）。兩者合計還是Ｏ（ｍｎ）。所以，選擇一個工作出來進行配置，再選出要配置的資源，選擇完畢之後的再修改表格，重複這些步驟直到將m個工作都處理完成、形成一種配置組合，總計所需要的時間複雜度是：ｍ〔Ｏ（m）+Ｏ（ｎｌｏｇｎ)〕+Ｏ（ｍｎ）＝Ｏ（ｍｎｌｏｇｎ），worst case 是Ｏ（ｍ$ｎ^2$)。

2、系統架構

在使用者將工作送到刀鋒式伺服器，到工作被完成、將結果回傳給使用者的整個流程中，除了刀鋒式伺服器本體之外，整個系統所牽涉到的部分還包括了：系統之輸入輸出、工作佇列（job queue)、排程（scheduling）演算法、資源狀態蒐集（resources status collection)、資料庫、配對（matchmaking)、配置（allocation）演算法、平行虛擬機器（PVM, Parallel Virtual Machine)，其間之互動關係如圖 11。

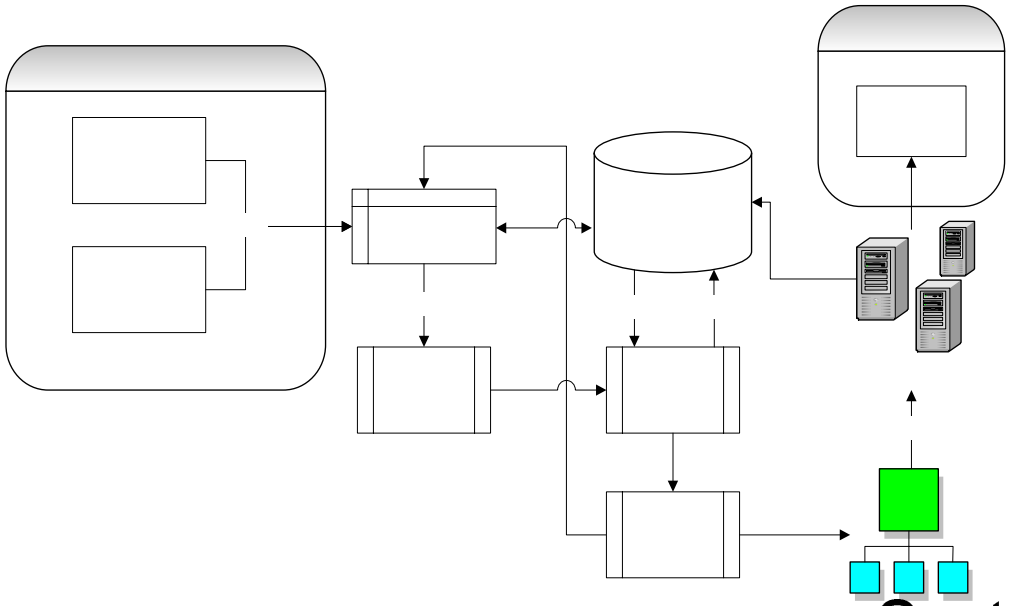


圖 11、系統架構圖。

2.1 系統之輸入及輸出

2.1.1 系統輸入

User send jobs

整個系統的輸入，就是各個使用者所送來的工作，或是由工作模擬器所生成的工作；簡而言之就是等待被處理的工作。而這些工作的描述，包含了有工作類型、資源需求（分為ＣＰＵ、記憶體，以及資源總數三部分），以及各種不同類型工作其自身的參數：

J

(1) **工作類型**：表明該工作屬於何種類型的工作，以使刀鋒式伺服器能夠執行相對應之程式，並要求該類型工作所需的參數。

Job simulator

(2) **資源需求**：

A. ＣＰＵ最低時脈：表示該工作僅能在ＣＰＵ不低於此時脈之資源上執行。

B. 可用記憶體大小：表示該工作僅能在記憶體不小於此大小之資源上執行。

C. 要求資源數量（RR）：表示該工作必須被配置到 RR 個資源才能順利執行。

(3) **工作參數**：執行該工作時所需要用到的參數。以矩陣的 LU 分解（LU Decomposition）為例，工作參數需要包含矩陣的維數。

Job type	=	“ LU ”	
Minimum CPU speed	=	“ 1000 ”	//MHz
Minimum memory required	=	“ 16 ”	//MByte
Requested resources	=	“ 1 ”	
Parameters			
Matrix dimension	=	“ 1000 ”	

圖 12、矩陣 LU 分解的工作。

上圖 12 為某一個矩陣 LU 分解的工作，其輸入系統時的描述。下頁圖 13 為使用者手動輸入三個矩陣 LU 分解的工作時的畫面。

圖 13、矩陣 LU 輸入。

2.1.2 系統輸出

當工作被執行完成之後，其結果會顯示於螢幕上，或者儲存於紀錄檔中。每個工作輸出結果包括了下列項目：

(1) **工作編號：**工作編號能讓使用者由眾多的工作當中，分辨出何者為自己所送至刀鋒式伺服器之工作，並檢視此工作是由哪台或哪些資源所執行，以及相關參數。

(2) **機器（資源）代號：**各個資源在回報工作結果時，都會標明自身的代號，以方便追蹤運作狀況以及紀錄查詢。

(3) **執行時間**：包含了各台機器開始執行此工作的日期與時間、工作完成的日期與時間，以及該機器花費在執行此工作上的總時數。

(4) **工作內容**：顯示出此工作的內容、參數等。

圖 14 為某一個矩陣 LU 分解的工作，其完成時系統輸出之描述。

```
Job number              =  " 0 "
Executed by             =  " /home/lab355/project/pvm/LU15 "
Job parameters
  Matrix dimension      =  " 120 "
  Requested resources   =  " 1 "
Resource ID             =  " LU15 "
Run time
  Start at              =  " 6/26 15:21:26 "
  End at                =  " 6/26 15:21:26 "
  Total                 =  " 62170 μs "
```

圖 14、矩陣 LU 輸出。

圖 15 為圖 14 當中的三個工作執行完成時之畫面。

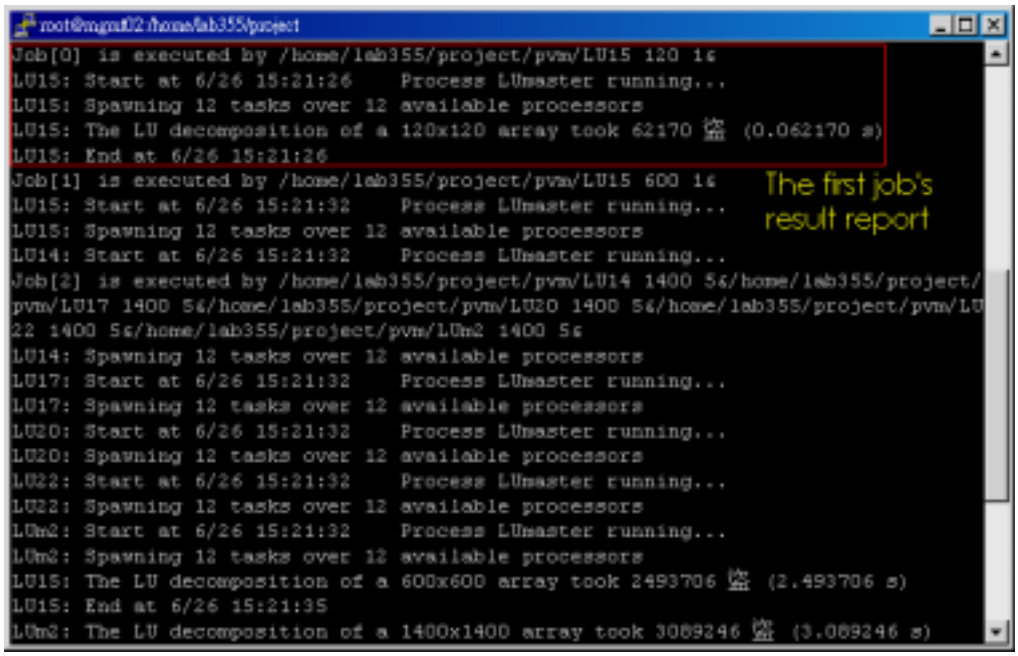


圖 15、矩陣 LU 結束畫面。

2.2 架構及流程

整個系統開始運作之前，刀鋒式伺服器的主伺服器上必須先行啟動 PVM、MYSQL Server，然後執行一個負責接收客戶端即時資訊之行程（process）。當所有準備工作完成之後，即可準備接收工作。

2.2.1 資料庫

主伺服器必須先啟動資料庫的服務（service），並且建立所需的資料庫以及主伺服器連接至資料庫時所使用的使用者身分、同時設好相關權限；後面這個步驟僅需要執行一次，因為資料是存放於硬碟中，機器重新啟動之後設定並不會改變。此處我們使用 MYSQL Server。資料庫啟動之後，接著才能夠啟動負責接收客戶端即時資訊的行程，同時紀錄各個資源的參數（如ＣＰＵ資訊）。在整個系統執行的過程中，資料庫將接收來自主伺服器的各種命令，包括了有建立資料表（create table）、查詢（query）、更新欄位內容（update）、增刪列（insert/delete row）等等。

2.2.2 蒐集最新資訊

在以往的系統中，要隨時保持各資源的最新資訊是一項需要許多額外支出的舉動，因為這牽扯到資訊傳遞，更與網路速度的快慢息息相關。然而在刀鋒式伺服器內部子網路速度高達 1Gbps 的情況下，我們可以隨時保持各刀鋒的最新狀態，卻不會增加太多負擔。

在系統能夠開始接收來自使用者或是工作模擬器的工作之前，必須先在主伺服器上啟動一個行程，專門用來接收、處理各資源所送來的狀態訊息。這些訊息內容目前包括了資源代號（目前以 IP 代表）、ＣＰＵ使用率、可用之記憶體大小。接收到這些訊息之後，主伺服器上的這個行程會將資料從訊息中解讀出來後，連線至資料庫，建立需要的資料表（table）或者更新相關欄位內容。

而各個資源，要成為可使用的資源，讓主伺服器知道該資源的存在、以加入配對和配置等階段之前，必須先啟動一個行程，專門負責每隔固定週期就將自身的即時狀態傳遞給主伺服器。這個行程剛開始執行時，會先將資源本身ＣＰＵ的基本資料送至主伺服器，接著會從本地端的系統上定時地蒐集所需要的資訊，再將資料封裝成一個物件（object），而後傳送至主伺服器。以上兩種行程（主伺服器上的行程和各個資源上的行程），都是以背景模式執行，因此並不會有結果可以顯示、輸出給使用者。圖 16 為蒐集最新資訊的流程圖。

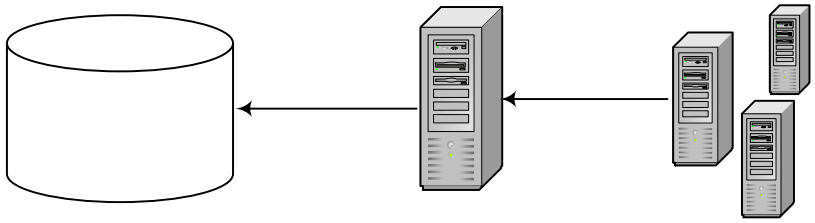


圖 16、資料庫更新。

2.2.3 工作佇列

當工作依照對應輸入系統之後，首先會被儲存至工作佇列中。若為使用者輸入之工作，將會告知使用者該工作之佇列編號（也就是工作編號）。此後要對工作進行排程、配置等作業時，或是未被配置到的工作，都是從工作佇列中取出或放入。

2.2.4 工作處理

接下來進行的動作，主要就是開始進行前面所敘述的演算法。首先由前次配置的結果，或是依照初始值，求得此次要從工作佇列中取出的工作數量。接著從工作佇列中取出工作，然後進行配對。在配對階段，主伺服器會依照各個工作所指定的需求條件，向資料庫進行查詢，找出符合該工作需求的資源。同時會依照各資源目前的ＣＰＵ使用率狀況來進行篩選，ＣＰＵ使用率高於一定門檻的資源將不會被視為符合條件。與一般的配對（matchmaking）所不同的是，在此階段並未直接找出、指派各個工作所將使用的資源，而僅是將各個工作可以使用的資源候選人全數找出並儲存。配對完成之後，隨即開始配置資源至各個工作。此時演算法會選擇出各個工作所將使用的資源。若有工作配置失敗，則該工作會被重新儲存回工作佇列中。

2.2.5 PVM

當工作已經決定由哪些資源來執行之後，接下來的步驟就是要透過 PVM，使那些資源可以執行配置到的工作。主伺服器必須在要將工作配置給各資源之前，先行啟動 PVM 的服務。而各個資源也必須先加入 PVM 的主機清單（host list）中，才能夠透過 PVM 來傳遞、執行工作。在一切設定就緒之後，主伺服器在本機上執行該工作所需的程式、同時給予所需參數，此程式即會透過 PVM，在各片刀鋒上產生（spawn）子程式來進行運算。子程式與主伺服器上之母程式之間的訊息傳遞，以及程式執行完畢時的結果回傳，均依照 PVM 內現有之協定溝通、傳遞。

第五部份、結果與討論

1、學理討論

當分散式環境資源的越來越多以及其資源使用的限制也越來越多時，傳統的資源配置管理系統就顯得沒有這麼彈性了。分散式環境分成兩個角色，一個是分散式資源管理，另一個是分散式資源擁有者。資源管理包含了各種不同的資源和不斷產生的新資源，而分散式資源擁有者則規定資源所能夠使用的條件。要去解決這樣的資源分配問題是一件很困難的事，所以我們引進了 Gangmatching 演算法，如圖 17。來解決這個問題。Gangmatching 是利用分類式廣告（classad）的架構來達成的，分類式廣告架構分成五個部分：分類廣告的規格，廣告的協定，配對演算法，配對的協定和提出要求的協定。其中 Gangmatching 演算法是依照優先順序大小排列的分類廣告中選出所要的資源，然後使用由上至下回溯演算法（top-down backtracking algorithm）來安排資源分配的問題。如此可以解決一些變動環境極大的網路資源分配問題。

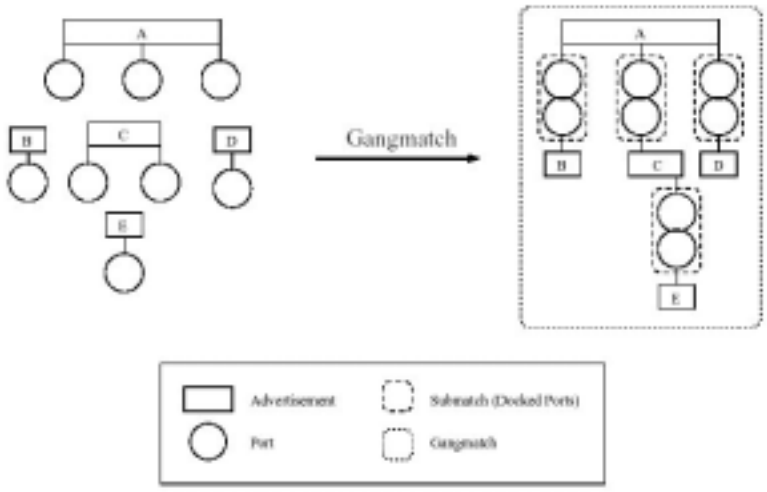


圖 17、Gangmatch 演算法概念。

另外我們使用 ER 派區網路語言，如圖 18。來描述網路上各種不同的資源以及資源上的限制…等並配合使用 Gangmatching 演算法。藉由這樣的轉換可以將網路資源配置的問題當成是一個 ER 派區網路，並且我們可以在這個問題裡面來討論可到達性的問題。在派區網路中，網路資源的配置可以藉由 Gangmatching 所造成的 gang tree 中就可以反映出一些特性。當然 gang tree 對於 Gangmatching 演算法的複雜度有著極大影響。

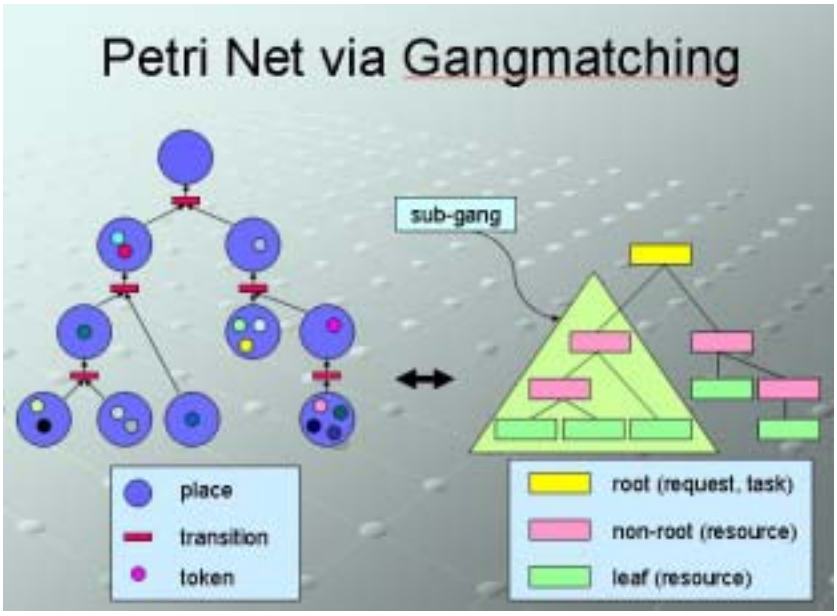


圖 18、ER 派區網路與 Gangmatching。

根據本團隊的研究，資源管理中首重各種資源之性質定義，針對其性質將本系統設計成可以妥善管理各種資源，我們可以對所有資源進行『保留』、『配置』以及『使用』等動作，利用資料庫的架構妥善規劃，使個別資源都可以在不影響整個系統的前提下，獨立地被加入或移出本系統。藉由工作模擬器的設計，產生多樣式的工作群，此時，工作排程與資源分配便扮演著舉足輕重的關鍵。在兼顧效率與公平的考量下，以智慧型代理人為工具，動態更新各項資源即時資訊。隨著各種不同需求的提出或改變，資源必須能夠被隨時動態重分配並立即使用，本系統主要利用配對演算法，這是資源管理中常運用的一種資源分配方法，利用廣告配對的想法，將{供}與{需}作最佳的組合。分散式的環境下，本系統必須具備足夠的容錯能力，可以適時地偵測並且更正錯誤。

使用者可以透過搜尋介面將自己可用資源和想用資源公告在伺服器上，此搜尋介面是由 QT 所建構而成並可達到跨平台性。之後藉由伺服器中管理資源排程演算法來決定資源該如何分配使用，接下來如何讓各個電腦之間進行溝通就變成一個重要的因素，還有將工作傳送給不同的電腦來進行計算的任務是本團隊所需要考慮的，故我們以 PVM（Parallel Virtual Machine）這個軟體來進行測試，PVM 軟體可以使各個電腦具有互相溝通計算之能力。因此我們利用 PVM 來達成平行運算的功能。當系統發生錯誤時，從最接近現在的檢查點重新開始工作，可是協同式計算環境中因為資源使用者並不擁有該項資源，所以資源很可能會被收回，或是工作中的某個伙伴發生當機，使得工作必須中斷，這樣子的中斷通常很突然且難以處理，故這也是我們嘗試要去解決的問題。協同計算環境是一個極易發生錯誤的環境，這樣的環境對安全性更是一大考驗，加強容錯能力與防止駭客入侵實在是很重要的課題。最後可以透過系統監測，指的是資源與使用者的使用率，

使用方式，計算行為等等，這些資料經過整理統計之後可以供作我們調整系統設計及管理策略之用。

2、實驗環境及實驗數據

我們在 IBM 所提供的刀鋒式伺服器上進行實驗。參與的刀鋒（blades）共計 12 片，作業系統為 Linux RedHat，每片刀鋒的ＣＰＵ為 Intel P4-Xeon 2.4GHz（400MHz FSB，512KB L2，HTT-enabled），記憶體為 1GB ECC PC2100 DDR266 SDRAM。主機連外頻寬為 100Mbps，伺服器內部各刀鋒之間的頻寬為 1Gbps。軟體套件上我們使用到的有 PVM (Parallel Virtual Machine) version 3.4.4, MYSQL Server version 3.23.58。

而模擬的工作內容，我們以分別以矩陣的 LU 分解（LU Decomposition），以及解一維的熱擴散（heat diffusion）方程式來作為計算內容。LU 分解是將一個矩陣 A 分解成一上三角矩陣 U 與一下三角矩陣 L，可以用來解 *Ax=b* 的方程式。熱擴散方程式則是在計算熱量在介質上擴散的情況，一維方程式則表示計算熱量在一條線上的擴散情況。一維熱擴散方程式為：

$$\frac{\partial A}{\partial t} = \frac{\partial^2 A}{\partial x^2} \qquad (1)$$

其離散形式的方程式為：

$$\frac{A_{i+1,j} - A_{i,j}}{\Delta t} = \frac{A_{i,j+1} - 2A_{i,j} + A_{i,j-1}}{\Delta x^2} \qquad (2)$$

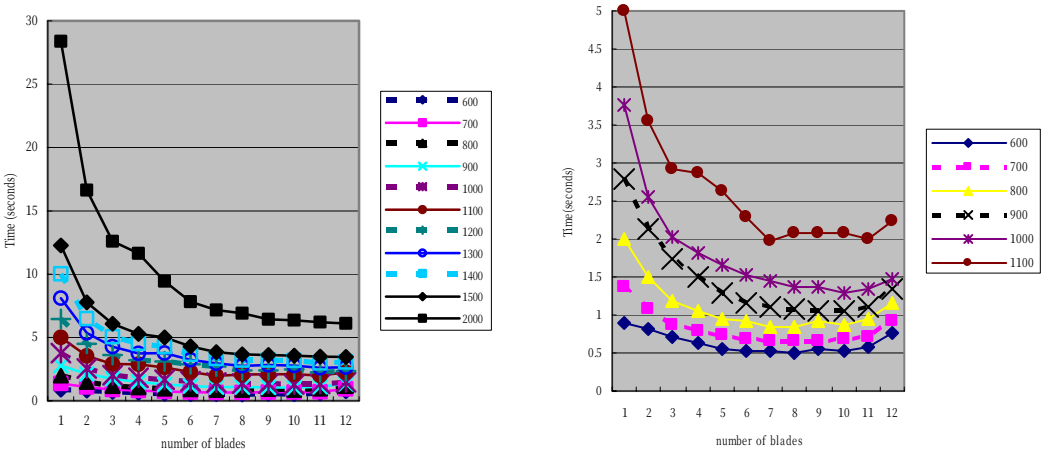
依照（2）式，給定邊界條件（boundary conditions）即可運算。

各個工作的需求以及矩陣大小我們以亂數決定。不過在進行各種不同狀況的測試時，我們使用的是參數相同的同一組工作，以求得較客觀的數據。

2.1 矩陣 LU 分解

由於速度增加率（speedup）並不是隨著使用的ＣＰＵ呈線性成長，因此我們為了找出最佳的效率，做了一連串的實驗。下圖 19（A）顯示的是各種不同大小矩陣，隨著使用刀鋒數目改變，其花費時間的折線圖。圖 19（B）是將圖 19（A）時間小於 5 秒的部分取出來的放大圖，以詳細觀察較小矩陣的時間變化。

表 1 為以矩陣的 LU 分解來進行 Amdahl's Law 之完整實驗數據。



(A) 原圖　　(B) 放大

圖 19、不同刀鋒與時間的折線圖-矩陣 LU 分解。

Calculation time (second)												
Problem Size	Number of blades											
	1	2	3	4	5	6	7	8	9	10	11	12
600	0.889	0.809	0.700	0.643	0.564	0.538	0.526	0.508	0.543	0.534	0.580	0.759
700	1.367	1.081	0.878	0.796	0.727	0.685	0.660	0.648	0.670	0.678	0.717	0.921
800	2.005	1.501	1.172	1.061	0.959	0.919	0.853	0.847	0.921	0.860	0.949	1.154
900	2.785	2.123	1.736	1.501	1.291	1.161	1.096	1.080	1.054	1.044	1.108	1.335
1000	3.770	2.553	2.028	1.814	1.671	1.528	1.436	1.371	1.372	1.296	1.342	1.472
1100	5.001	3.541	2.911	2.875	2.624	2.280	1.970	2.069	2.083	2.085	1.998	2.245
1200	6.452	4.491	3.598	3.195	3.116	2.868	2.704	2.396	2.396	2.474	2.253	2.338
1300	8.103	5.352	4.252	3.740	3.753	3.219	2.963	2.745	2.821	2.784	2.608	2.640
1400	10.027	6.487	5.047	4.509	4.258	3.660	3.339	3.196	3.205	3.175	2.975	3.059
1500	12.267	7.771	6.089	5.282	4.997	4.297	3.840	3.659	3.616	3.569	3.475	3.449
2000	28.397	16.622	12.580	11.622	9.444	7.819	7.171	6.904	6.436	6.352	6.204	6.102

表 1、矩陣 LU 分解實驗數據。

由數據可知，計算量小的工作（矩陣大小 800 以下），能夠使其工作時間最短的ＣＰＵ數量約為 7~8；中等計算量的工作（矩陣大小在 900~1400），能夠使其工作時間最短的ＣＰＵ數量約為 9~11；需要龐大運算的工作，能夠使其工作時間最短的ＣＰＵ數量則在 12 以上。圖 20 為 Speedup 曲線圖，表 2 為 Speedup

數據。

Speedup												
Problem Size	Number of blades											
	1	2	3	4	5	6	7	8	9	10	11	12
600	1	1.099	1.27	1.383	1.576	1.652	1.69	1.75	1.637	1.665	1.533	1.171
700	1	1.265	1.557	1.717	1.88	1.996	2.071	2.11	2.04	2.016	1.907	1.484
800	1	1.336	1.711	1.89	2.091	2.182	2.351	2.367	2.177	2.331	2.113	1.737
900	1	1.312	1.604	1.855	2.157	2.399	2.541	2.579	2.642	2.668	2.514	2.086
1000	1	1.477	1.859	2.078	2.256	2.467	2.625	2.75	2.748	2.909	2.809	2.561
1100	1	1.412	1.718	1.739	1.906	2.193	2.539	2.417	2.401	2.399	2.503	2.228
1200	1	1.437	1.793	2.019	2.071	2.25	2.386	2.693	2.693	2.608	2.864	2.76
1300	1	1.514	1.906	2.167	2.159	2.517	2.735	2.952	2.872	2.911	3.107	3.069
1400	1	1.546	1.987	2.224	2.355	2.74	3.003	3.137	3.129	3.158	3.37	3.278
1500	1	1.579	2.015	2.322	2.455	2.855	3.195	3.353	3.392	3.437	3.53	3.557
2000	1	1.708	2.257	2.443	3.007	3.632	3.96	4.113	4.412	4.471	4.577	4.654

表 2、矩陣 LU 分解 Speedup 數據。

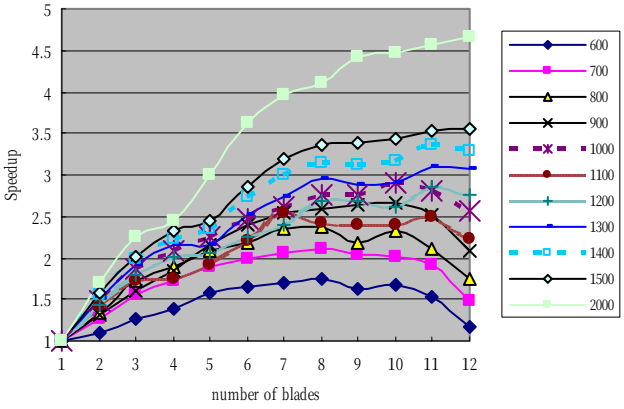
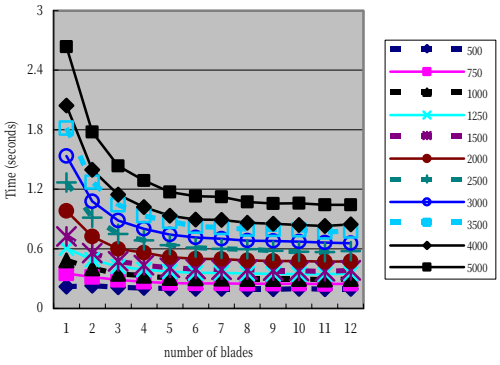
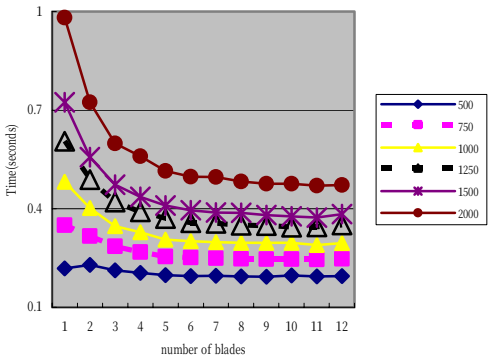


圖 20、矩陣 LU 分解 Speedup 折線圖。

2.2 一維熱擴散方程式

圖 21 (A) 顯示的是各種不同大小的熱擴散方程式，隨著使用刀鋒數目改變，其花費時間的折線圖。下頁圖 21（B）是將圖 21（A）時間小於 1 秒的部分取出來的放大圖，以詳細觀察較小方程式的時間變化。表 3 為以一維熱擴散方程式來進行 Amdahl's Law 之完整實驗數據。





(A) 原圖　　(B) 放大

圖 21、不同刀鋒與時間的折線圖-熱擴散。

Calculation time (second)												
Problem Size	Number of blades											
	1	2	3	4	5	6	7	8	9	10	11	12
500	0.219	0.229	0.213	0.205	0.198	0.195	0.196	0.194	0.193	0.197	0.194	0.195
750	0.350	0.317	0.286	0.268	0.255	0.252	0.250	0.248	0.247	0.247	0.246	0.248
1000	0.482	0.402	0.347	0.328	0.306	0.301	0.298	0.296	0.296	0.296	0.290	0.295
1250	0.605	0.489	0.421	0.391	0.370	0.358	0.356	0.350	0.349	0.344	0.345	0.351
1500	0.724	0.557	0.473	0.436	0.410	0.395	0.389	0.388	0.381	0.377	0.373	0.384
2000	0.982	0.724	0.599	0.560	0.515	0.498	0.497	0.483	0.476	0.476	0.470	0.472
2500	1.269	0.913	0.750	0.685	0.638	0.613	0.600	0.595	0.579	0.571	0.566	0.577
3000	1.535	1.077	0.884	0.803	0.742	0.714	0.703	0.683	0.680	0.674	0.664	0.657
3500	1.815	1.273	1.042	0.935	0.874	0.826	0.814	0.799	0.794	0.791	0.768	0.779
4000	2.043	1.398	1.145	1.021	0.932	0.896	0.893	0.863	0.853	0.841	0.830	0.845
5000	2.637	1.777	1.435	1.287	1.173	1.129	1.123	1.073	1.055	1.059	1.043	1.043

表 3、熱擴散實驗數據。

圖 22 為 Speedup 曲線圖。與矩陣 LU 分解相較，其 Speedup 較低，原因是因為在解一維熱擴散方程式的過程中，經常必須互相交換邊界條件，因此訊息傳遞所花費的時間佔整體時間的比例較高，也就是 Amdahl's Law 當中無法加速的連續性部分。大多數的方程式在刀鋒片數達到 12 的時候，Speedup 都已經開始些微下降。下頁表 4 為 Speedup 數據。

Speedup												
Problem Size	Number of blades											
	1	2	3	4	5	6	7	8	9	10	11	12
500	1	0.956	1.028	1.068	1.106	1.123	1.117	1.129	1.135	1.112	1.129	1.123
750	1	1.104	1.224	1.306	1.373	1.389	1.4	1.411	1.417	1.417	1.423	1.411
1000	1	1.199	1.389	1.47	1.575	1.601	1.617	1.628	1.628	1.628	1.662	1.634
1250	1	1.237	1.437	1.547	1.635	1.69	1.699	1.729	1.734	1.759	1.754	1.724
1500	1	1.3	1.531	1.661	1.766	1.833	1.861	1.866	1.9	1.92	1.941	1.885
2000	1	1.356	1.639	1.754	1.907	1.972	1.976	2.033	2.063	2.063	2.089	2.081
2500	1	1.39	1.692	1.853	1.989	2.07	2.115	2.133	2.192	2.222	2.242	2.199
3000	1	1.425	1.736	1.912	2.069	2.15	2.183	2.247	2.257	2.277	2.312	2.336
3500	1	1.426	1.742	1.941	2.077	2.197	2.23	2.272	2.286	2.295	2.363	2.33
4000	1	1.461	1.784	2.001	2.192	2.28	2.288	2.367	2.395	2.429	2.461	2.418
5000	1	1.484	1.838	2.049	2.248	2.336	2.348	2.458	2.5	2.49	2.528	2.528

表 4、熱擴散 Speedup 數據。

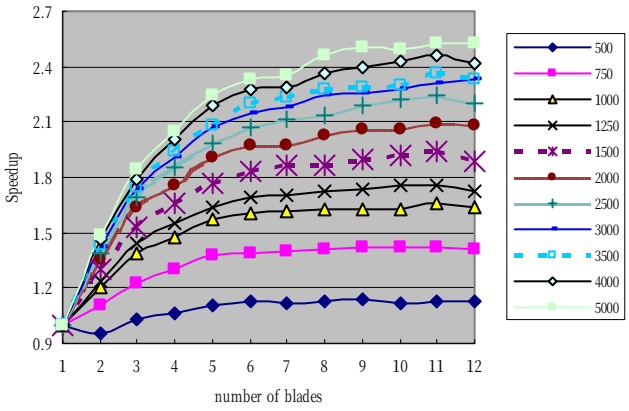


圖 22、熱擴散 Speedup 折線圖。

3、實驗分析

3.1 等待週期（Job Fetching Interval）的影響

首先說明一下各個名詞的定義。**總分配時間**代表的是從第一個工作開始被配置，到最後一個工作被配置出去，所花費的時間。**總工作時間**代表的是從第一個工作開始被配置，到最後一個工作被完成，所花費的時間。以上兩者的差距主要

在於最後一批工作配置出去之後，到一些計算量比較龐大工作都計算完成之間的時間。**問題大小上限（maximum problem size）**則表示：在矩陣的 LU 分解中，矩陣 A 的大小是從 1 到大小上限之間以亂數決定；一維熱擴散方程式的大小也是從 1 到大小上限之間以亂數決定。**刀鋒使用時數**則代表該片刀鋒花費在計算工作上的時間。**總執行時數**則是所有刀鋒的使用時數的總合。下面將以兩種工作，分別以等待週期 6 秒和 30 秒各進行一次測試，並對其結果加以分析。

3.1.1 矩陣 LU 分解

（A）對照組

工作總數=400，刀鋒總數=12，問題大小上限=2000，等待週期=6 秒，工作取出量由演算法決定。總分配時間=1373 秒，總工作時間=3125 秒，總執行時數=18692.8 秒，各刀鋒使用時數如表 5：

Blade	使用時數（秒）	Blade	使用時數（秒）
M3	338.5	30	1225.9
25	2540.8	31	1517.4
26	1303.4	32	1199.5
27	1437.5	33	806.7
28	1122.3	34	2572.7
29	2155.9	35	2472.2

表 5。

（B）實驗組

工作總數=400，刀鋒總數=12，問題大小上限=2000，等待週期=30 秒，工作取出量由演算法決定。總分配時間=2315 秒，總工作時間=3950 秒，總執行時數=20071.8 秒，各刀鋒使用時數如表 6：

Blade	使用時數（秒）	Blade	使用時數（秒）
M3	2488.1	30	2468.7
25	3158.1	31	897.3
26	968.8	32	1087.9
27	1341.0	33	880.0
28	1646.9	34	2183.7
29	1434.2	35	1517.1

表 6。

3.1.2 一維熱擴散方程式

(A) 對照組

工作總數=400，刀鋒總數=12，問題大小上限=5000，等待週期=6 秒，工作取出量由演算法決定。總分配時間=372 秒，總工作時間=375 秒，總執行時數=1457.7 秒。

(B) 實驗組

工作總數=400，刀鋒總數=12，問題大小上限=5000，等待週期=30 秒，工作取出量由演算法決定。總分配時間=1087 秒，總工作時間=1088 秒，總執行時數=2176.3 秒。表 7 為等待週期之實驗數據比較表。

	矩陣 LU 分解			一維熱擴散方程式		
等待週期	總分配時間	總工作時間	總執行時數	總分配時間	總工作時間	總執行時數
6 秒	1373 秒	3125 秒	18692.8 秒	372 秒	375 秒	1457.7 秒
３０秒	2315 秒	3950 秒	20071.8 秒	1087 秒	1088 秒	2176．３秒

表 7、等待週期之實驗數據比較。

3.1.3 分析

在矩陣 LU 分解的部分，將等待週期拉長到比每個工作所需的執行時間還長的情況下（單片刀鋒執行矩陣大小=2000 所需平均時間為 28.4 秒，參見表 1），會使得執行時間明顯增加。2.1.1 當中對照組的 M3 使用時數偏少的原因是在於 M3 本身是 management node，必須處理分配工作以及 PVM 和 MYSQL 等其他程式，使得在配置階段時，容易因為ＣＰＵ使用率高過門檻值而不被配置。而在 2.1.2 中則因為每次週期之後，幾乎所有的工作均已完成，故無此現象。而在一維熱擴散方程式的實驗上，總工作時間更是明顯增加至將近三倍。

3.2 工作取出量（Job Fetching Window Size）的影響

下面將以兩種工作，分別以工作取出量由演算法決定、工作取出量固定為 1（FCFS）、工作取出量固定為 12、工作取出量固定為 6、各進行一次測試，並對其結果加以分析。

3.2.1 矩陣 LU 分解

(A) 對照組

工作總數=400，刀鋒總數=11，問題大小上限=2000，等待週期=6 秒，工作取出量由演算法決定。總分配時間=1425 秒，總工作時間=3176 秒，總執行時數=18577.4 秒，各刀鋒使用時數如表 8。

Blade	使用時數 (秒)	Blade	使用時數 (秒)
M2	-	18	3011.7
13	785.0	19	2458.5
14	1254.9	20	1484.6
15	1118.9	21	2601.9
16	1165.1	22	1238.5
17	1305.2	23	2153.1

表 8。

(B) Fixed window size

- First come first serve

工作總數=400，刀鋒總數=11，問題大小上限=2000，等待週期=6 秒，工作取出量固定為 1，也就是 first come first serve。總分配時間=2434 秒，總工作時間=4102 秒，總執行時數= 18524.3 秒，各刀鋒使用時數如表 9。

Blade	使用時數 (秒)	Blade	使用時數 (秒)
M2	-	18	357.9
13	968.9	19	53.0
14	1780.4	20	3182.5
15	1790.2	21	1365.5
16	901.5	22	2509.0
17	3456.0	23	2159.4

表 9。

- Large window size

工作總數=400，刀鋒總數=11，問題大小上限=2000，等待週期=6 秒，工作取出量固定為 12。總分配時間=1662 秒，總工作時間=3449 秒，總執行時數=18586.6 秒，各刀鋒使用時數如表 10。

Blade	使用時數 (秒)	Blade	使用時數 (秒)
M2	-	18	1453.8
13	899.7	19	1236.4
14	2474.4	20	1233.6
15	1347.9	21	2492.7
16	1495.5	22	2685.4
17	2151.1	23	1116.1

表 10。

- Medium window size

工作總數=400，刀鋒總數=11，問題大小上限=2000，等待週期=6 秒，工作取出量固定為 6。總分配時間=1496 秒，總工作時間=3246 秒，總執行時數=18579.9 秒，各刀鋒使用時數如表 11。

Blade	使用時數 (秒)	Blade	使用時數 (秒)
M2	-	18	1503.9
13	1228.7	19	2502.3
14	2675.9	20	1323.8
15	1257.0	21	1445.0
16	2161.1	22	740.1
17	2618.3	23	1123.8

表 11。

3.2.2 一維熱擴散方程式

(A) 對照組

工作總數=400，刀鋒總數=12，問題大小上限=5000，等待週期=6 秒，工作取出量由演算法決定。總分配時間=372 秒，總工作時間=375 秒，總執行時數=1457.7 秒。實驗數據為引用 3.2.1 中之結果。

(B) Fixed window size

- First come first serve

工作總數=400，刀鋒總數=12，問題大小上限=5000，等待週期=6 秒，工作取出量固定為 1，也就是 first come first serve。總分配時間=2400 秒，總工作時間=2404 秒，總執行時數= 768.8 秒。

- Large window size

工作總數=400，刀鋒總數=12，問題大小上限=5000，等待週期=6 秒，工作取出量固定為 12。總分配時間=390 秒，總工作時間=392 秒，總執行時數= 1580.1 秒。

- Medium window size

工作總數=400，刀鋒總數=12，問題大小上限=5000，等待週期=6 秒，工作取出量固定為 6。總分配時間=410 秒，總工作時間=411 秒，總執行時數= 1285.0 秒。表 12 為工作取出量之實驗數據比較表。

	矩陣 LU 分解			一維熱擴散方程式		
工作取出量	總分配時間	總工作時間	總執行時數	總分配時間	總工作時間	總執行時數
由演算法決定（對照組）	1425 秒	3176 秒	18577.4 秒	372 秒	375 秒	1457.7 秒
FCFS	2434 秒	4102 秒	18524.3 秒	2400 秒	2404 秒	768.8 秒
固定為 12	1662 秒	3449 秒	18586.6 秒	390 秒	392 秒	1580.1 秒
固定為 6	1496 秒	3246 秒	18579.9 秒	410 秒	411 秒	1285.0 秒

表 12、工作取出量之實驗數據比較。

3.2.3 分析

首先說明為何在矩陣 LU 分解的實驗中，參與執行工作的刀鋒數從原本的 12 降為 11。原因是因為在進行 large window size 這一項實驗時，若 management node 也加入執行工作的可用資源，會由於負載過大、記憶體不敷使用，使得 PVM、MYSQL 等程式出現異常，進而影響工作的配置、發生工作遺失的錯誤。這些現象在工作取出量會隨時改變的情況下並不會發生。然而為了數據上能夠比較，因此將之改為僅以 11 片刀鋒參與執行工作。

矩陣 LU 分解的部分，在總執行時數上，各種設定均約略相同（差距最大約千分之三），差異值可視為程式誤差。而不論是總分配時間或是總工作時間，FCFS 的工作效率均明顯較差：總分配時數增加了 70.8%，總工作時間增加了 29.2%。將工作取出量固定為略大於可用資源總量，由於容易發生必須將資源暫留下來、避免某些工作發生飢餓，因此整體時間較對照組有略為拖長：總分配時數增加了 16.6%，總工作時間增加了 8.6%。這個總花費時間拖長現象，會隨著工作取出量所固定的值，超過總體可用資源越多而越明顯。而將工作取出量固定為可用資源

總量之半數，由於與理想值之誤差較小，效率也比另外兩種設定均高出一些，與對照組差距不大：總分配時數增加了 5.0%，總工作時間增加了 2.2%。

在一維熱擴散方程式的部分，總執行時數容易隨著同一時間正在解的方程式數量而有所改變，原因是因為所有的邊界條件都需要透過 PVM 傳遞。而每解方程式是以遞迴的方式進行，每計算一輪就必須交換一次邊界條件，而 PVM 是由 management node 提供的服務，當訊息量大的時候處理時間就會增加，間接造成解方程式的時間增加、總執行時數增加。也因此可以由表 12 中發現，一維熱擴散方程式的總執行時數與工作取出量是呈正相關。由於熱擴散方程式所需的工作時間較短，因此 FCFS 的工作效率相當於每六秒處理完一個工作，總分配時間和總工作時間都在對照組的六倍以上。同時也由於單個工作的工作量較小，而使得適當的工作取出量較大，介於 6~12 且較接近 12（由數據可看出）。將工作取出量固定為可用資源總量（12）時，總分配時數增加了 4.8%，總工作時間增加了 4.5%。將工作取出量固定為可用資源總量之半數（6）時，總分配時數增加了 10.2%，總工作時間增加了 9.6%。

3.3 可用資源總數的影響

參與工作配對、配置的可用資源總數，也就是刀鋒片數，已經分別以 12 片和 11 片進行過實驗。下面是以 6 片刀鋒所作的實驗，以觀察可用資源總數與執行時間的相互關係。

(A) 矩陣 LU 分解

工作總數=400，刀鋒總數=6，問題大小上限=2000，等待週期=6 秒，工作取出量由演算法決定。總分配時間=2903 秒，總工作時間=4615 秒，總執行時數=18351.6 秒，表 14 為與對照組之數據比較表。

刀鋒總數	總分配時間	總工作時間	總執行時數
1 2	1373 秒	3125 秒	18692.8 秒
6	2903 秒	4615 秒	18351.6 秒

表 14。

由以上數據可知，由於資源總數減半，總分配時間與對照組相較變為 2.1 倍，可用資源總數與總分配時數呈線性關係；不是剛好兩倍的原因是在於演算法執行的次數變多，使得總分配時間略為增加。而總工作時間與總分配時間，其之間的差距代表的是，從最後一批工作被配置出去，到全部工作均已完成的時間。這段時間主要是由一些計算量龐大、耗時較久的工作所影響，因此與資源總數較

無關聯。由各項實驗也可發現這個差距也幾乎均為定值，表示每次最後才執行完成的工作，其所耗費時間比起其餘工作就是多出那段固定的時間差距。

(B) 一維熱擴散方程式

工作總數=400，刀鋒總數=6，問題大小上限=5000，等待週期=6 秒，工作取出量由演算法決定。總分配時間=594 秒，總工作時間=596 秒，總執行時數= 1118.8 秒。表 15 為與對照組之數據比較表。總分配時間增加了 59.7%，總工作時間增加 58.9%。總執行時數由於同一時間執行之方程式數目減少而跟著減少。

刀鋒總數	總分配時間	總工作時間	總執行時數
1 2	372 秒	375 秒	1457.7 秒
6	594 秒	596 秒	1118.8 秒

表 15。

第六部份、計畫成果自評

- 根據本計畫的進度，本團隊對第一年研究的統整如下：

一、各種資源發現模式的研究與理論驗證。
二、各種共同排程模式的研究與理論驗證。
三、各種撮合模式的研究及理論驗證。
四、資源管理架構的研究與效能評估。
五、 找出適合大中小型協同式計算環境之最佳網路架構。

前四項主要在資源管理的一些核心技術進行深入瞭解，並且試著提出可行之解決方案，這方面我們已經完成，並且進一步發現資源的利用與分配可以利用 ER 派區網路做更有效率的排程，至於第五項我們研究了許多種的網路架構，大小不同的計算環境所需的網路架構並不相同，如何找到一個合適共通的平衡點，著實是一個好的研究問題。

- 根據本計畫的進度，本團隊對第二年研究的統整如下：

一、研究資訊統整。
二、制訂完整可行之系統架構。
三、研究設計三項資源管理服務。
　甲、監控。
　乙、排程。
　丙、配對。
四、制訂富有調整彈性之協定。
五、演算法的設計與運用。
六、整合發展必要的所需軟體。
七、檢查點與安全性的探討。

- 根據本計畫的進度，本團隊對第三年研究的統整如下：

在第三年的規劃中，本團隊針對所設計的系統，做更進一步的測試與評估，使本系統於效能上，能有一個量化的資源管理機制。提出一套針對刀鋒式伺服器所設計，將排程、配對、配置三者整合的演算法，探討其實作方面的考量，同時舉實例示範該演算法執行過程，最後並分析其複雜度。在實作部分，將對於刀鋒式伺服器如何接收工作輸入、對工作須如何描述，對工作進行處理的流程，其間如何與資料庫、PVM、以及其他的程式、服務相互溝通，一直到工作輸出結果，

對整體系統架構及流程做個完整的說明，並舉出實際應用的例子以茲說明（以矩陣 LU 分解和一維熱擴散方程式），以印證基礎理論以及演算法分析。以指數成長的方式來隨著實際情況改變工作取出量，的確比固定一個數值來得有效率。且若實際情況的變化起伏越大，則此法較固定數值之優勢會愈加明顯。然而此法仍舊有其缺陷，若工作的性質、『總需求量』(Total requests，TR)變化並不大，依此演算法所求出的工作取出量穩定值，會是 2 的乘冪中比 TR 大的最小乘冪，並不是十分精確的值。若 TR 越大，則絕對誤差越大。此外，雖然我們找出了 Amdahl's Law 在刀鋒式伺服器上頭之理想值，但工作本身將分配給幾台刀鋒一起進行運算，目前的方法是由客戶端決定，伺服器僅照要求處理，尚未做到視工作內容、大小而主動調整分配的資源數目。演算法上，最佳解仍舊無法保證能夠迅速求出，在 worst case 時甚至與暴力法同樣效果。實作上，目前使用的 PVM 功能並非特別強大，在指定特定電腦群組的功能上有待加強。

在未來的研究規劃中，本團隊將針對所設計的系統，首先將先完成介面更新的目標。在測試與除錯後，便進一步進行其他相關議題（如：容錯、平行系統、…）研究，整合至本資源管理機制系統，提升系統能力與效能。

第七部份、參考文獻

[1] Condor Web Site Http://www.cs.wisc.edu/condor

[2] Jim Basney and Miron Livny, "*Deploying a High Throughput Computing Cluster*", in High Performance Cluster Computing, Rajkumar Buyya, Editor, Vol. 1, Chapter 5, Prentice Hall PTR, May 1999.

[3] Miron Livny, Jim Basney, Rajesh Raman, and Todd Tannenbaum, "*Mechanisms for High Throughput Computing*", SPEEDUP Journal, Vol. 11, No. 1, June 1997.

[4] Jim Basney, Miron Livny, and Todd Tannenbaum, "*High Throughput Computing with Condor*" , HPCU news, Volume 1(2), June 1997.

[5] D. H. J Epema, Miron Livny, R. van Dantzig, X. Evers, and Jim Pruyne, "*A Worldwide Flock of Condors : Load Sharing among Workstation Clusters*" Journal on Future Generations of Computer Systems Volume 12, 1996.

[6] Scott Fields, "*Hunting for Wasted Computing Power*" 1993 Research Sampler, University of Wisconsin-Madison.

[7] Rajesh Raman, Miron Livny, and Marvin Solomon, "*Resource Management through Multilateral Matchmaking*", Proceedings of the Ninth IEEE Symposium on High Performance Distributed Computing (HPDC9), Pittsburgh, Pennsylvania, August 2000, pp 290-291.

[8] Rajesh Raman, Miron Livny, and Marvin Solomon, "*Matchmaking: Distributed Resource Management for High Throughput Computing*", Proceedings of the Seventh IEEE International Symposium on High Performance Distributed Computing, July 28-31, 1998, Chicago, IL.

[9] Jim Basney and Miron Livny, "*Managing Network Resources in Condor*", Proceedings of the Ninth IEEE Symposium on High Performance Distributed Computing (HPDC9), Pittsburgh, Pennsylvania, August 2000, pp 298-299.

[10] Jim Basney and Miron Livny, "*Improving Goodput by Co-scheduling CPU and Network Capacity*", International Journal of High Performance Computing Applications, Volume 13(3), Fall 1999.

[11] D. Angulo, I. Foster, C. Liu, and L. Yang. *Design and Evaluation of a Resource Selection Framework for Grid Applications.* Proceedings of IEEE International Symposium on High Performance Distributed Computing (HPDC-11), Edinburgh, Scotland, July 2002.

[12] G. A. Geist, J. A. Kohl and P. M. Papadopoulos, *PVM and MPI: a Comparison of Features*, Calculateurs Paralleles Vol. 8 No. 2 (1996).

[13] I. Foster and C. Kesselman. *"The Grid: Blueprint for a New Computing Infrastructure"*, Morgan Kaufmann, San Fransisco, 1999.

[14] J. L. Peterson. *Petri Net Theory and the Modeling of Systems*. Prentice -Hall International, 1981.

[15] C. Ghezzi, D. Mandrioli, S. Morasca, and M. Pezze. *"A unified high-level Petri net formalism for time-critical systems."* IEEE Transactions on Software Engineering, 17:160-172, 1991.

[16] T. Murata. "Petri nets: Properties, analysis and applications." Proceedings of the IEEE, 77(4):541-580, April 1989.

[17] J. Wang. *Time Petri nets: Theory and application.* Kluwer Academic Publishers, 1998.

[18] MPI Forum. *MPI: A message– passing interface standard.* Interna- tional Journal of Supercomputer Application, 8 (3/4) : 165-416, 1994

[19] V. S. Sunderam. *PVM: A framework for parallel distributed computing.* Concurrency: Practice & Experience, 2 (4), 1990.

[20] Charlie Lai, Gennady Medvinsky, B. Clifford Neuman. *"Endorsements, licensing, and insurance for distributed system services."* In Proceedings of the Second ACM Conference on Computer and Communications Security, pages 170-175, November 1994.

[21] Jerome H. Saltzer, David P. Reed, and David D. Clark. *"End-to-end arguments in system design."* ACM Transactions in Computer Systems 2, 4, pages 277-288, November 1984.

[22] Jim Pruyne and Miron Livny. *"Interfacing Condor and PVM to harness the cycles of workstation clusters."* Journal on Future Generations of Computer Systems, Volume 12, 1996.

[23] Keith Decker, Mike Williamson, Katia Sycara. *"Matchmaking and Brokering."* In Proceedings of the Second International Conference on Multi-Agent Systems (ICMAS-96), December 1996.

[24] Santi Saeyor and Mitsuru Ishizuka. *"Cooperative Matchmaking of Requests Among Distributed Change Monitoring Service Agents."* In Proceedings of 1999 IEEE Pacific Rim Conference on Communications, Computer and Signal Processing (PACRIM'99), August 1999.

[25] VUB Parallel Computing Laboratory. The Computer Science service of the faculty of Applied Science of the Vrije Universiteit Brussel. Similar results are available at http://parallel.vub.ac.be/

[26] S.C. Chin. *Analysis of Resource Management via Timed Petri Nets.* Master Thesis, Electrical Engineering Department of National Taiwan University, June 2002.

[27] M.C. Pong. *Job Scheduling, Matchmaking, and Allocation on Blade Servers.* Master Thesis, Electrical Engineering Department of National Taiwan University, July 2004.