國立交通大學

資訊科學系

碩士論文

一個利用分散式資料庫建置高效能網格運算架構的方法

Towards a High-Performance Grid Computing Infrastructure —A Distributed Databases Approach

研究生：郭訓宏

指導教授：陳俊穎　博士

中華民國　九十三　年七月

一個利用分散式資料庫建置高效能網格運算架構的方法

Towards a High-Performance Grid Computing Infrastructure —A Distributed Databases Approach

研 究 生：郭訓宏　　Student：Hsun-Hung Kuo

指導教授：陳俊穎　　Advisor：Jing-Ying Chen

國 立 交 通 大 學

資 訊 科 學 系

碩 士 論 文

A Thesis

Submitted to Department of Computer and Information Science

College of Electrical Engineering and Computer Science

National Chiao Tung University

in partial Fulfillment of the Requirements

for the Degree of

Master

in

Computer and Information Science

July 2004

Hsinchu, Taiwan, Republic of China

中華民國九十三年七月

一個利用分散式資料庫建置高效能網格運算架構的方法

學生：郭 訓 宏　　指導教授：陳 俊 穎　博士

國 立 交 通 大 學

資 訊 科 學 系

摘　　要

格網運算能夠集合許多儲存及計算設備的計算能力，並提供一致的存取方法。對格網架構的發展者而言，目標在一個穩定及安全的方法下，建立一個能夠有效率分配與協調分散及異質性資源的機制。相對的，對格網程式發展者而言，最主要的挑戰為針對網格架構所提供的功能作最適當的運用。通常的情況下，程式設計者需要將問題切割成較小的工作，並要計畫如何對資料作適當的操作及傳輸。若程式執行對記憶體的需求超過一般個人電腦的能力，這樣的分割勢必是需要的，但是相對的會讓程式本身更為複雜。本文論述了藉由資料庫特有的技術，能夠減輕程式設計者在開發程式時對於記憶體管理的負擔。簡單而言，我們使用個別的關連式資料庫當作計算節點，利用資料庫的儲存及計算能力，將各節點串連組合成一個分散式計算平台。除此之外，我們定義了一個通用的資料表綱要，能夠儲存複雜的資料結構，並能夠有彈性地轉換成其他易於計算的表格狀資料結構。我們認為結合這些想法，能夠建構一個有效簡化格網應用程式發展的平台，有助於格網運算的發展。

Towards a High-Performance Grid Computing Infrastructure —A Distributed Databases Approach

Student：Hsun-Hung Kuo　　Advisor：Dr. Jing-Ying Chen

Department of Computer and Information Science

National Chiao Tung University

Abstract

Grid Computing promises to provide uniform, non-expensive access to computing power through aggregation and utilization of potentially unlimited number of storage and computing devices. For Grid infrastructure developers, this goal amounts to creating effective mechanisms that can allocate and coordinate distributed, heterogeneous resources in a robust and secure manner. For Grid application developers, on the other hand, the main challenge is to make best use of the facilities provided by the infrastructure. Typically, a developer needs to divide a problem into smaller pieces, and plan for appropriate data manipulation and transfer among them. Such divide-and-conquer effort is essential when required memory space is beyond the capabilities of individual machines, but complicated when the infrastructure provides only low-level facilities. This thesis describes database-specific techniques that can relieve developers from complicated memory management. Simply speaking, we use individual relational databases as computational nodes for their storage and computation capabilities, and connect them together into a distributed computing platform. In addition, we define a generic schema capable of storing complex data structures, and mechanisms that allow flexible translation between the schema and other computation-friendly tabular structures. We argue that together these constructs form an attractive platform that can greatly simplify Grid application development, thus contribute to the general Grid Computing community in a useful way.

誌　　謝

隨著本文的完成，也代表著在新竹六年的求學生涯告一段落。

在交大的這兩年，首先要感謝我的指導教授—陳俊穎博士，在研究領域對我不倦的教誨，讓我不斷地學習與成長。此外，還要感謝兩位口試委員，袁賢銘博士以及劉興民博士，不吝於對我論文上的指教。

當然，要感謝的人太多，實驗室的同窗好友，建宏、阿吉、舜禹，給予我的支持與協助，更是讓我能夠順利完成學業的最佳伙伴。大學社團的好友，珮芬、翔慧、慶媽，總是能在我低潮的時候給予我最溫暖的問候與祝福。室友敦致的照顧，讓我兩年租屋生活無後顧之憂。以及大學班上的好友們，一起相互漏氣才能一同進步。

最後要感謝我的家人，家人的體諒與包容，讓我的求學之路可以走的平穩順遂。父母親的慈愛與關懷，讓我自在的學習成長，兩個姊姊的督促與鼓勵，亦是驅使我向前的動力。

夢想就要起飛，生命中嶄新的一頁正要開始。祝福我的師長朋友們，在人生的旅途上，一路平安、快樂！

目　錄

表目錄

圖目錄


E
S
A
1896

第一章　緒論

近年來網際網路快速的成長，以及儲存設備硬體技術的成熟，使得資料藉由高速網路的傳遞分享與儲存備份，變得更加容易與方便。然而在現今的科學研究，正思考著如何對巨量資料建立有效率之儲存、管理、模擬、重構、傳佈以及分析的共享機制。例如，在生物領域方面，生物資訊專家則正發展生物網格技術(BioGrid [20])，以探究並解決長久以來生物學中所面臨到科學運算的挑戰。

寬頻網際網路的發展，同時對高速計算與應用產生了衝擊，那就是跨平台的演算與合作，也就是所謂的格網運算(grid computing [2][4])。格網運算最主要的目的是要善用閒置伺服器能量，從事更多的工作，來彙整與有效運用分散之資源，以提供對一個需要大量計算問題的求解方案。並用以挑戰規模尺度更大，以及進行更複雜之應用與科學之發現與探索。格網運算的環境提供了一個虛擬的計算環境，並能夠在其中執行應用程式。而要在網格上撰寫程式，程式撰寫者必須要能夠管理平行的、異質的、動態的運算資源及其效能。雖然目前有工具可以幫助使用者發展格網程式，但仍無法適當地有效率的管理機器間的合作或是處理異質性的機器與資料。

當使用者編寫一個平行處理格網程式，需要花費心思將程式平行化，舉例而言，程式執行時，會將運算資料載入進記憶體中，在大量資料運算情形下很容易超過記憶體的限制，因此程式中必須要做適當的記憶體管理。除此之外，格網也將網路服務(Web Service)整合至其中，不同的服務可能會來自相異的系統，且各服務有著各自定義的介面及檔案格式，所以當使用者的程式想要利用這些服務時，程式當中也需要管理資料的轉換。為了格網將來發展的完備及其架構能夠完成的能力，上面所提出的問題值得被討論。

我們提出了建置分散式的資料庫環境來處理上述的問題。由於資料庫系統本身已具有對機器資源的管理，開發資料庫端可執行的程式，可解決一般程式開發者對於記憶體管理的問體。另一方面，使用資料庫的內儲程式，對於大量資料的運算，能夠減少原本計算所需的大量時間，來提供高效能服務。

我們利用生物資訊所面臨到的需求當作來系統設計的出發點，並當作我們系統實作的目標。由於生物資訊資料結構複雜性高，生物資訊資料的概念複雜度及確認生物資料之間的相關聯性困難，若要在關連性資料庫之下建立表格，則會需要依照不同的儲存情形來建立不同的表格。為了一種情況來定義一次表格，對使用者來說是個麻煩且費時的動作，並且與表格相關的演算法，也會受到影響。於是我們提出了使用 generic data-type schema 來儲存資料，無論資料的格式欄位為何，讓使用者無須為了定義資料表格的欄位而費神。

在建立各資料庫之後，我們將資料庫實作成服務，服務間利用 XML 訊息透過 socket 機制來傳遞。除了訊息之外，各資料庫之間也能夠將資料輸入輸出並互相傳遞，來達到資料共享的目的。使用者能夠編寫在資料庫端執行的程式，將機器上的資料及運算分散至其他資料庫上，來達到平行運算的目的。

本文除了第一章緒論外，在第二章相關背景中，介紹與本研究相關的背景知識。第三章系統設計，闡述了系統設計理念。第四章系統實作，說明系統實作的細節。第五章實驗結果，以內儲程式與 Java 程式做效能比較並分析結果。第六章相關研究，討論了與我們動機及想法類似的研究。第七章結論，總結本文與未來發展。

第二章　相關背景

在現今的科學運算中，資料密集以及高效能需求的應用，需要有效率地在分散在大範圍的計算資源中管理及傳遞巨量資料。研究領域涵跨了高能物理、氣候模型、地震預測、天文研究以及生物資訊等等，這些應用包括實驗數據的分析及模擬。在這些研究領域的應用，大量的資料必須要能夠被分享與共用，研究人員要能夠將實驗結果的資料從一台機器傳送至遠端另一台機器作運算分析。

而目前新興的研究"格網運算"[2][4]，其概念正是基於資源的共享以及在動態的環境中有效率的解決問題。而資源的共享，不僅只是資料檔案的互相交換，更希望是直接對電腦、軟體、資料或其他能夠共同合作解決問題的資源。而資源間的共享的限制，更要能夠讓各資源的管理者控制。

2.1 格網運算

目前電腦計算強調著合作(collaboration)，資料共享(data sharing)，計算能力共享(cycle sharing)以及一些其他包含著分散式資源的互動形式。加上軟體、硬體、以及其他資源不斷的個人化，使得一些希望能夠透過分散式、廣域網路來取得並分享資源及服務的應用程式開始發展，於是相關的需求也開始被提出。格網運算 (Grid Computing)指的是藉由高速網路大規模整合的電腦系統，可按照使用者的需求提供資料處理功能，或提供單台機器與一小群機器所無法達成的功能。而格網運算與一般常見分散式運算的差異在於，它著重於大規模的資源共享，創新的應用，以及對工作效能的注重。簡單的說，將閒置、分散的電腦或個別 CPU 藉由網路整合其運算能力，成為一個獨立處理資訊的系統，格網將打破個人電腦運算的界線。過去幾年來，格網運算逐漸受到學術研究社群的注意，高能物理、生物資訊、地球觀測、全球變遷以及數位典藏等領域皆積極探究網格技術的應用。

以生物資訊為例，目前(2004/06)[9]已有 167 個基因組(genome)，包含了人類的基因組，已經完成了定序解碼，而這些資料儲存在共用的資料庫裡，資料量並以將近每年兩倍的速度成長著。對於這些大量的資料以及一些複雜的演算法，如蛋白質的折疊(folding)結構的研究或是 3D 結構的預測，要在一個可接受的時

間內完成運算，將其運算平行化是一個重要的方法。在將格網技術應用到生物領域上，目前已有一些正在進行中的生物格網計畫，包括 myGrid [19]，Asia Pacific BioGrid Initiative [20]，North Carolina BioGrid [21]，Canadian BioGrid，EUROGRID project [22]以及 Biomedical Informatics Research Network。

由於希望能夠利用網格帶來的計算能力，因此撰寫在網格環境上執行的應用程式，引起許多開發者的注意。一個格網應用程式，可看做是一個工作的集合或是一個利用網格資源完成複雜計算的任務，通常大部分的程式中的工作仍需要程式發展者的控制。所以對於發展格網應用程式，程式的執行與網格架構元件間的互動有著極大的關連。由於一個網格的環境可能是大規模、分散在各地，並且是異質的，所以設計網格應用程式是一種挑戰。非網格應用程序在相對穩定、定義良好並且通常是專用的環境中運行，而具有網格功能的應用程序的運行環境則是一種動態的、有時還是鬆散定義的並且強烈依賴於網絡的環境。

一個網格應用程式，除了要具備一般循序(sequential)程式的能力與特性外更要包含平行化及分散程式的特點。除了協調控制多樣的行程以及分散的資料，程式設計者要能夠管理一個通常是異質且動態的環境中的計算，伴隨著因為頻寬限制與記憶體使用的加深而造成的延遲。所以在格網程式的撰寫，資源管理就顯得相當重要。一般機器的資源除了記憶體、硬碟空間，還包括了程式執行時所需要的硬體設備。然而格網程式在格網環境中，將被擴散到實質上開放式的環境，因此有著更多的資源如機器、行程、儲存設備、網路、服務等都是可被獲得的。這些資源會是分散在廣大的區域使用共用的架構的異質環境。在單一處理器的環境下，作業系統負責了行程的管理，或是可以由程式來控制。而在分散式環境中，尤其是在網格環境的異質性下，更使得平行程式的編寫更加複雜。所以在編寫格網程式中，將程式中行程的平行化，如同步化行程的平行需要對平行執行緒的產生與終止，皆需要程式發展者明確的控制與管理。

The Globus Project[1]對資源管理、安全、資訊服務及資料管理等網格計算的關鍵理論進行研究，開發能在各種平臺上運行的網格計算工具軟體，幫助規劃和組建大型的網格試驗平臺，開發適合大型網格系統運行的大型應用程式。Globus 定義了許多方便撰寫格網程式的函式，Globus Toolkit 3 (GT3) 便是其中

之一，讓使用者用來撰寫網格程式。Globus Toolkit 提供了許多對格網服務的元件，它包含了諸如 GridFTP [6]之類的工具。GridFTP 構建在標準 FTP 協定之上，使用者可以使用 GridFTP 工具來移動資料文件，而不必在所涉及的每個節點上都要執行一遍登錄過程。這個工具提供了第三方文件傳送，這樣一個節點就可以啟動另兩個節點間的文件傳送。Toolkit 是 Globus 一項重要的成果，全球網格套件包含了監視、尋找以及管理資源的軟體服務及函式庫。

由於網格環境是由許多異質的資源所組成，所以在網格之間的互動行為必須要透過一個一致的機制，因此由 Global Grid Forum[25]定義了一個能夠支持在不同的資源間具有通透性的開放式標準，稱為 Open Grid Services Architecture (OGSA)。

OGSA [3]是一種基於網格服務的分佈式互動和計算體系結構，用來確保異質系統間的互通性，這樣不同類型的系統就可以進行通信、共享訊息。OGSI (Open Grid Services Interface)則定義了共通的協定及介面，而 OGSI 是利用 OGSA 為基礎的格網間互通的一個標準。

OGSI 規範定義了網格服務並建立在標準 Web Service [27] 之上。 OGSI 利用了 XML 與網路服務描述語言（Web Services Description Language，WSDL）這些網路服務機制，為所有網格資源指定標準的介面、行為與互動。OGSI 進一步擴展了網路服務的定義，提供了動態的、有狀態的和可管理的網路服務的能力，這在對網格資源進行標準化時都是必需的。

與 Web Service 相較，網格服務讓程式編寫者利用 factory/instance 的機制來解決 Web Service 的 stateless 限制及缺少生命週期(life cycle)的管理。網格服務另外還使用 service data 來描述服務本身。service data 包含 state information 及 service metadata。除此之外，相較於 Web Service 的 URI，網格服務利用 Grid Service Handle (GSH) 來讓其他使用者知道服務的位置，並且利用 Grid Service Reference (GSR) 來告知如何與此網格服務連接溝通。

是否能夠將程式有效率的切割並分散至各端點，通常決定在於問題本身的性質。有些問題能夠容易的平行化，而有些問題則是必須要以循序的方式來解決。這其間的差異需要程式設計者的巧思來分析判別。而提到將運算工作平行化並分散至網路各地執行，利用閒置伺服器的運算功能，最早提出的大型計畫為SETI@home。 SETI [10]，尋找外星智慧計畫 (Search for ExtraTerrestrial Intelligence)是一項國際性的科學計畫，利用無線電望遠鏡(telescope)接收來自太空中自然產生的窄頻寬(narrow-bandwidth)訊號，來求證在地球之外是否還有其他的智慧生命存在，在 1999 研究人員將計畫發布至網路上，利用由網路上許多電腦所組成的虛擬超級電腦，來完成大量資料的分析。此計畫的成功也證明了公用資源計算(public-resource computing)的可行性。

SETI 一項很重要的工作便是將從望遠鏡上取得的原始資料，切割成小量的工作分散給世界給地的電腦執行。這項任務是交由一台伺服器來完成，每份工作的大小是 0.25Mb，而將切割任務的工作交由一台主機的目的是為了減少儲存設備的存取步驟。

SETI 發展了一個負責管理資料與結果的伺服器[11]，主要功能是將工作單位分配給使用者。他利用 HTTP 的協定，使用者在下載了工作之後，執行分析，並回傳結果，只有在與主機連線時需要網路的連線。所以 SETI@home 的計算模型並不複雜，收集到的訊號資料分割成固定大小的工作單元，藉由網路分散給各地的用戶端程式執行，在運算產生結果後回傳給伺服器並取得新的工作單元。圖(1)解釋了如何將原始資料切割成給使用者執行的工作單元。

而 SETI 的工作分配，以現階段來說，是將同一份資料(就是工作單位)寄給一些使用者，只要有接獲兩份以上的結果回傳，就將該份工作單位從電腦中移除。這樣的做法可以加速新資料的取得以及加速將龐大的資料分離成每一個工作單位的速度。SETI@home 的用戶端程式利用 C++開發完成，客戶端的程式可以當成背景行程執行，或是當作螢幕保護程式。

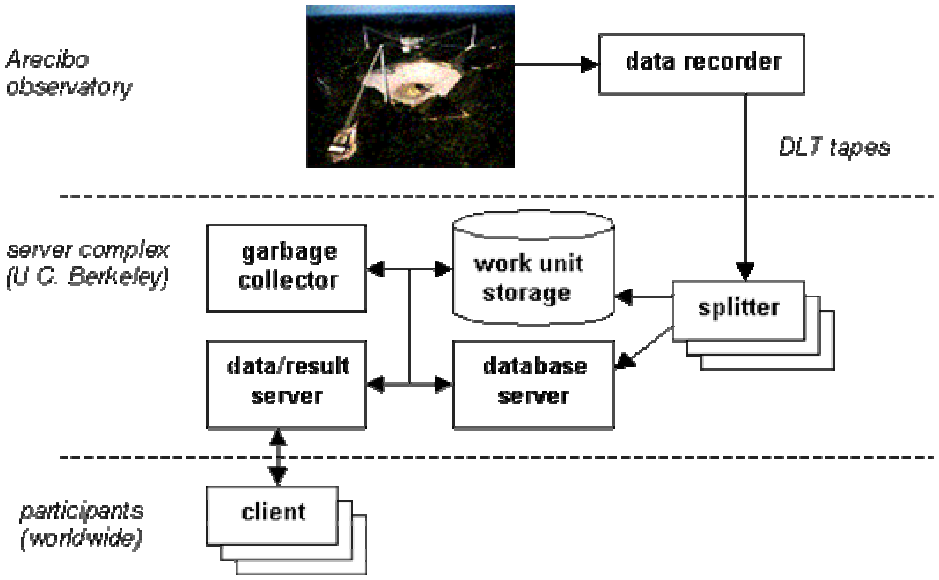


圖 1　SETI 架構 [12]

與網格應用程式相比，SETI 將計算工作平行化的方法較為簡單，僅將原始資料分割成同樣大小的工作，讓使用者下載至個人電腦上執行。在網格環境下，由於要考慮程式的普遍性及面對動態異質的環境的狀況，一般的問題需要更特別的平行化方法，程式設計者會有更多的考量。

2.2 問題歸納

綜合前述目前發展格網程式時所面臨到的難題，由於格網程式必須要管理計算環境，他們通常是平行的、動態的、以及異質的，加上程式會需要動態的及有彈性的結合所需要的資源與服務，雖然目前有一些工具提供建立格網程式的方法，但仍無法有效率的管理資源間動態的結合，或處理異質的機器。因此除了對一般的資料作運算操作外，程式設計者更要去設計與遠端服務的互動，以及與資料來源及硬體計算資源的管理。包括了記憶體、硬碟空間，以及所需控制的硬體設備。

而格網計算通常會應用在大規模、高效能的計算上，想要獲得高效能的結果，需要在計算過程中對所有參與的運算機器，對計算有著平衡的分佈。在某些情況下，這些將運算平行的機制可由程式設計者來管理，而有些則無法手動達

成。因此目前格網程式還沒有一個有效率的程式模組(programming model)能夠減少在動態、異質的環境下將程式大量平行化所產生的負擔。

所以一個設計良好的程式模組必須要兼具有著高效能計算與能夠彈性的結合與管理資源。程式模組也會影響著軟體的生命週期：設計、實做、除錯、操作等等，所以一個好的程式模組也要能夠提供有效率的發展工具，如 debugger、效能的監控等等。

2.3 目標與想法

我們提出了利用建置一個分散式的資料庫環境來解決上述的問題。在我們的分散式環境中，可想成是由許多分散在網路上的許多節點所組成。每個節點皆能夠提供計算的功能，且每個節點能夠將資料準備(Data Provision)成特定的格式資料，此格式的資料除了能夠利用網路在各節點間傳遞，更能夠儲存在各節點內。我們也定義了一種程序語法，讓使用者能夠在程序中將計算工作指定至某台機器上執行，將運算工作平行化。程序中我們也定義了一些常用到的數學運算，讓使用者能夠更省時的編寫程序。

以生物資訊應用為例，在生物資訊的資料分析方面，大量激增的資料，使用資料庫儲存是必然的趨勢。然而由於生物資訊的資料變異性大，若使用一般關連式資料庫定義表格的方法，將會顯得繁瑣及重複。為了解決儲存生物資訊資料的難題，目前常見的方法為建立一個邦聯式的資料庫(Federal Database)。邦聯式資料庫的作法為在資料庫與資料庫之間，建立一個共通的介面，但是各資料庫本身的資料型態與綱要仍是各異。所以當一個生物學家需要一份來自兩個不同資料庫的資料作分析時，選取所需的資料之後再將其整合成一份資料，勢必需要繁瑣的步驟。

因此我們提出了一套 generic data-type schema 設計，利用此法儲存資料，使用者只要利用 XML 定義好欲儲存的實體類型(Entity Type)與實體(Entity)，無須再對儲存不同的資料來定義資料表綱要。加上由於系統間各資料庫有著一致的資料表綱要，所以在其他的資料庫上也能方便的分享與取得資料。

而關連式資料庫，在經過這幾十年的發展，除了資料儲存與取得的功能之外，目前的幾間公司的資料庫系統，如 Oracle、SQLServer 等等，皆提供了資料庫內部的資料運算功能，使用者只要定義好內儲程式(Stored Procedure)並儲存在資料庫內，就能夠對資料庫內的表格進行運算。內儲程式能夠長時間存放在資料庫內，透過程式的呼叫或特定事件的觸發來執行。在對大量資料進行運算時，一般程式會面臨到實體記憶體不足的問題，而降低整體運算速率，並影響電腦的其他服務。而利用資料庫的內儲程式 對資料庫內的資料進行運算，資料量的大小對於運算的速率影響，則不會有劇烈的變化。利用 stored procedure 來撰寫程式，除了可以減輕應用程式的負擔之外，並提供了重複使用性及對資料庫的功能有較佳的控制性。

基於上述理由，我們發展讓使用者編寫利用資料庫內儲程序來完成工作的程序，由於資料庫系統本身會對機器的資源做調配，保持系統的穩定性，因此使用者在編寫程序時不必在另外花費心思對機器資源的管理作控制。

利用建置分散式資料庫的方法，因為資料庫本身對機器資源的控制管理，我們發展一種能夠讓使用者簡便的編寫程序來完成大量資料的運算，也能夠將計算工作分配至其他的機器上，且利用 XML 編寫程序，也讓使用者能夠清楚的定義工作的分配與流程。透過這種機制，來完成一個類似網格運算環境的架構，而能夠有效率的處理大量資料的儲存與運算。

第三章　系統架構與設計

在我們的設計的平行程式的執行環境，可想成是由許多分散在網路上的節點所組成。每個節點可能包含了資料庫系統，也可能僅是一個網路服務。在每個節點上皆能夠提供計算的功能，且包含資料庫的節點能夠將資料準備(Data Provision)成特定的格式資料，此格式的資料除了能夠利用網路在各節點間傳遞，更能夠儲存在資料庫內。我們也定義了一種程序語法，讓使用者能夠在程序中將計算工作指定至某個節點上執行，並將結果回傳，達到運算工作平行化。在程序中我們也預先定義了一些常用到的數學運算，讓使用者能夠更省時的編寫程序。

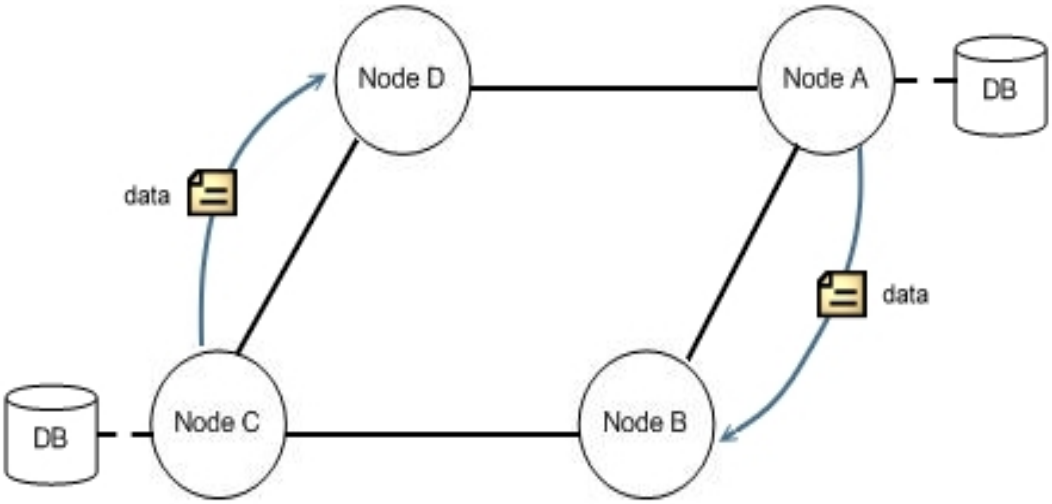


圖 2　Programming Model

3.1 HiDB 系統架構

我們發展了一套資料庫系統，稱為 HiDB (High-Performance DataBase)，本節說明 HiDB 的架構設計。

如圖(3)所示，在每個機器端點，除了資料庫之外，還有著一個以 Java 實做的服務。服務負責與資料庫的連接，除了將使用者的要求傳遞給資料庫之外，服務與服務間則能夠利用 socket 來傳遞以 XML 為基礎的訊息。

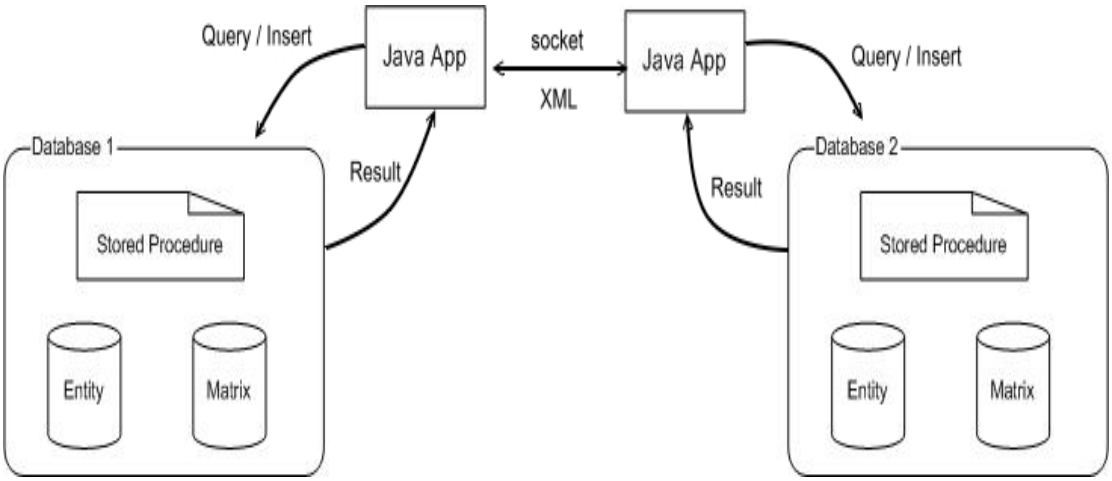


圖 3　HiDB Architecture

除了儲存與瀏覽資料之外，使用者可以將機器 A 上的資料輸出(Export)成利用資料庫備份機制產生的 dmp 檔案，再利用我們所提供的底層連線機制，將資料輸入(Import)至指定的機器 B 上。藉由此法，有效率的資料備份與共享能夠在我們的系統中達成。

我們利用了資料庫所提供的呼叫內儲程式的功能，讓使用者來定義自己所要完成的一段程式，如果程式中有特定的運算，可以利用呼叫資料庫內已有的內儲程式來完成，而無須重複定義編寫，每台機器上所擁有的內儲程式端看使用者如何定義。

因此當某台機器要做某種運算而機器上沒有此演算法時，便可以將欲運算的資料傳遞且輸入至另一台亦具有此運算服務的機器上完成運算，最後再將其結果以相同的方法傳遞回來。如此在面對大量資料運算之時，能夠將資料分散出去運算來達到平行化的處理。

3.2 Generic Data-Type Schema

在關連式資料庫中，資料表(Table)是資料庫內最重要的單位。使用者在使用資料庫儲存資料之時，首先要選擇存入的表格，若表格尚未存在，則需依其需要來定義每個表格的欄位的名稱及類型。每個表格的欄位定義我們稱為資料表綱要(schema)，每個要存入此表格的資料，皆需要符合此綱要的定義。

定義 generic data-type schema 的目的，除了對於儲存資料變異性大的資料，讓使用者可以不必重複定義資料表綱要外，這個一致的資料表綱要，也提供了使用資料庫內儲程式來完成運算的便利性。由於在我們的實作當中，所有資料庫皆會使用此綱要，因此在撰寫資料庫內儲程式時，對於資料的選取，能夠簡單的完成，並且若將本地端的資料庫內儲程式移至遠端其他的資料庫中執行，則不會因為表格定義的差異，而導致無法執行。

3.2.1 實體 (Entity) 與實體類別 (Entity Type)

在 generic data-type schema 中，概念上最基本的單位為一個實體(Entity)。我們將欲存入的資料視為一個實體，使用者在建立實體之前，必須要先定義此實體屬於何種類型，如此才能產生實體間的關連性。

在我們系統中，每個實體在資料庫有著兩樣最基本的資訊，序號 (id) 以及名稱 (name) 。序號是實體儲存在資料表中所對應產生唯一的號碼，名稱則儲存使用者對每個實體的稱呼。每個實體還會有參數 (Attribute) ，並且實體與實體之間會伴隨著由使用者所定義的關係。除此之外，每個實體會有一個類別(Type)，類別限制了每個實體所能夠擁有的參數，以及與其他實體間的關係。

由於類型的定義是要給所有屬於這個類型的實體去遵循，且類別是利用 XML 來定義並產生。所以每個類型除了自己定義的參數之外，我們另外定義了五個布林(boolean)值的參數，has_Image 、has_Ints 、has_Floats 、has_Strs 、has_Refs，讓系統能夠知道每個類型會包含著什麼資料型態的資料。當使用者要去定義一個類型，必須要給予這五個參數設定值。當使用者要儲存屬於這個類型的資料時，系統會去檢查此類型這五個參數的值，而決定需要將資料存放至哪些對應的表格中。

表(1)是利用 xml 格式來定義的一個類型結構。在類型的定義中，需要為此類型決定名稱，此類型會包含的參數，以及上述的五個布林值的參數。而這個類型會包含的子類型，子類型的名稱、參數，子類型與親代類型之間的 tag，且每個子類型包含的五個布林值的參數，都是需在這裡同時定義。表(1)的類型之間的示意圖我們以圖(4)來表示。

```
<type name= "Parent">
    <attr name="A" />
    <has ints = "false"
        strs = "true"
        image = "false"
        floats = "false"
        refs = "false"/>
    <child name="Child1"
        tag="a">
        <attr name="B" />
        <has ints = "false"
            … />
    </child>
    <child name="Child2"
        tag="b">
        <attr name="C" />
        <has ints = "false"
            … />
    </child>
</type>
```

表 1　Type 的 Scheme

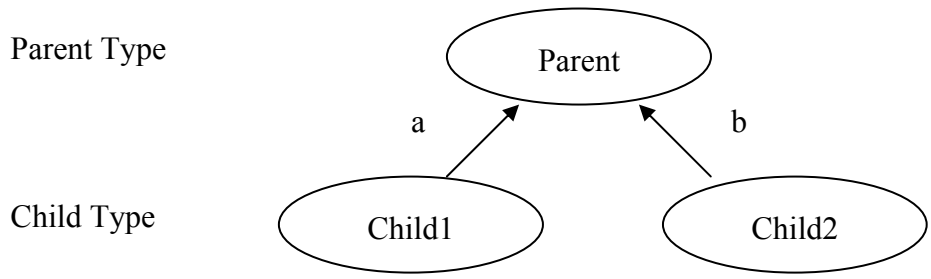


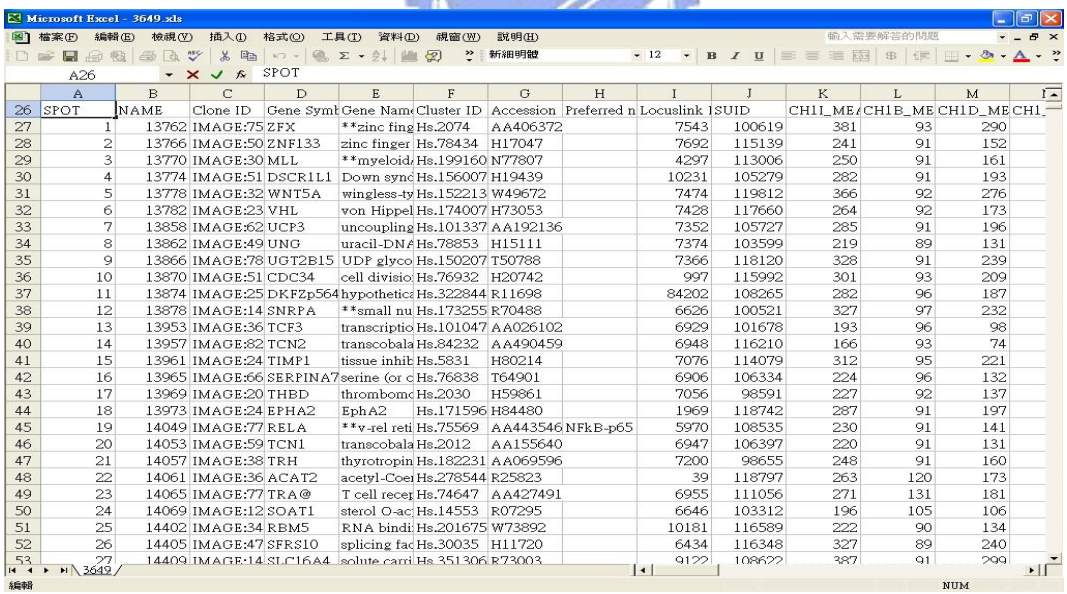
圖 4　Type 示意圖

在利用 XML 定義完類型之後，類型與其子類型之間，會產生一個樹狀結構的概念，每個親代會有不限個數的子代，且親代與子代之間有著由使用者定義的標籤，因此在資料的選取時，能夠藉由親代與子代間的標籤，來完成特定目的的

資料選取。

接下來我們利用一個完整的例子來說明如何建立實體及實體類型。我們選擇了利用微陣列矩陣(microarray)[16][17]的資料，從建立實體、儲存資料到編寫分析程序的過程，皆能夠在資料庫端內完成。在附錄(一)中，我們對微陣列矩陣做了簡介。

一個微陣列矩陣在經過機器掃瞄(scan)之後，會產生一個原始檔案(raw data)以 excel 檔案格式儲存，其格式如圖(5)所示。此檔案是利用 ScanAlyze 這個軟體 Scan 晶片後所產生。由於掃瞄後的檔案參數繁雜，若將此檔案儲存至資料庫表格，依照關連式資料庫的概念，將每個直行對應到資料表的一個欄位，就需要近 70 個對應的欄位，其中更包含許多的空白欄位。在一個原始檔案就有將近萬筆資料的情況下，只要使用者儲存較多的原始檔案，其中所浪費的空間將難以想像。若能夠讓使用者依照每次的實驗定義，只去儲存所需要的條件的資料，則整個資料庫的空間及效能，能夠獲得較好節省與提升。


Microsoft Excel - 3649.xls

	A	B	C	D	E	F	G	H	I	J	K	L	M
26	SPOT	NAME	Clone ID	Gene Symb	Gene Name	Cluster ID	Accession	Preferred n	Locuslink I	SUID	CH1I_ME	CH1B_ME	CH1D_ME
27	1	13762	IMAGE:75	ZFX	**zinc fing	Hs.2074	AA406372		7543	100619	381	93	290
28	2	13766	IMAGE:50	ZNF133	zinc finger	Hs.78434	H17047		7692	115139	241	91	152
29	3	13770	IMAGE:30	MLL	**myeloid/	Hs.199160	N77807		4297	113006	250	91	161
30	4	13774	IMAGE:51	DSCR1L1	Down sync	Hs.156007	H19439		10231	105279	282	91	193
31	5	13778	IMAGE:32	WNT5A	wingless-ty	Hs.152213	W49672		7474	119812	366	92	276
32	6	13782	IMAGE:23	VHL	von Hippel	Hs.174007	H73053		7428	117660	264	92	173
33	7	13858	IMAGE:62	UCP3	uncoupling	Hs.101337	AA192136		7352	105727	285	91	196
34	8	13862	IMAGE:49	UNG	uracil-DNA	Hs.78853	H15111		7374	103599	219	89	131
35	9	13866	IMAGE:78	UGT2B15	UDP glyco	Hs.150207	T50788		7366	118120	328	91	239
36	10	13870	IMAGE:51	CDC34	cell divisio	Hs.76932	H20742		997	115992	301	93	209
37	11	13874	IMAGE:25	DKFZp564	hypothetica	Hs.322844	R11698		84202	108265	282	96	187
38	12	13878	IMAGE:14	SNRPA	**small nu	Hs.173255	R70488		6626	100521	327	97	232
39	13	13953	IMAGE:36	TCF3	transcriptio	Hs.101047	AA026102		6929	101678	193	96	98
40	14	13957	IMAGE:82	TCN2	transcobala	Hs.84232	AA490459		6948	116210	166	93	74
41	15	13961	IMAGE:24	TIMP1	tissue inhib	Hs.5831	H80214		7076	114079	312	95	221
42	16	13965	IMAGE:66	SERPINA7	serine (or c	Hs.76838	T64901		6906	106334	224	96	132
43	17	13969	IMAGE:20	THBD	thrombomo	Hs.2030	H59861		7056	98591	227	92	137
44	18	13973	IMAGE:24	EPHA2	EphA2	Hs.171596	H84480		1969	118742	287	91	197
45	19	14049	IMAGE:77	RELA	**v-rel reti	Hs.75569	AA443546	NFkB-p65	5970	108535	230	91	141
46	20	14053	IMAGE:59	TCN1	transcobala	Hs.2012	AA155640		6947	106397	220	91	131
47	21	14057	IMAGE:38	TRH	thyrotropin	Hs.182231	AA069596		7200	98655	248	91	160
48	22	14061	IMAGE:36	ACAT2	acetyl-Coe	Hs.278544	R25823		39	118797	263	120	173
49	23	14065	IMAGE:77	TRA@	T cell recep	Hs.74647	AA427491		6955	111056	271	131	181
50	24	14069	IMAGE:12	SOAT1	sterol O-ac	Hs.14553	R07295		6646	103312	196	105	106
51	25	14402	IMAGE:34	RBM5	RNA bindi	Hs.201675	W73892		10181	116589	222	90	134
52	26	14405	IMAGE:47	SFRS10	splicing fac	Hs.30035	H11720		6434	116348	327	89	240
53	27	14409	IMAGE:14	SLC16A4	solute carri	Hs.351306	R73003		9122	108622	387	91	299

圖 5　Microarray Raw Data

首先，使用者必須對要建立的實體類型做定義，我們將微陣列矩陣視為一個實體，因此需要對矩陣定義一個”array”的類型，且為了利用 generic data-type schema 儲存資料的便利，我們定義在原始資料中每個直行是一個實體，故定義

了”column”這個類型，column 會是 array 的子類型。由於為了將來對微陣列矩陣的原始資料作分析時，需要將這些數據轉變成矩陣類型，所以我們也定義了”RowHead”這個類型，也就是列索引這個類型，作為在資料轉換產生矩陣時的一個索引對照。綜合這些條件，對微陣列矩陣其完整的類型定義如下。

```
<type name="Array">
    <attr name="Date" />
    <attr name="Owner" />
    <has ints="false"
       strs="false"
       image="true"
       floats="true"
       refs="false"/>
    <child name="Column"
        tag="b" >
        <has ints="false"
            strs="false"
            image="false"
            floats="false"
            refs="false" />
    </child>
    <child name="RowHead"
        tag="a">
        <has ints="false"
            strs="true"
            image="false"
            floats="false"
            refs="false" />
    </child>
</type>
```

表 2　type xml 定義

而在表(2)中所定義的型態，在將其對應到實際的微陣列矩陣的資料後，型態間的樹狀概念與原始資料的關係，可以用圖(6)來說明。

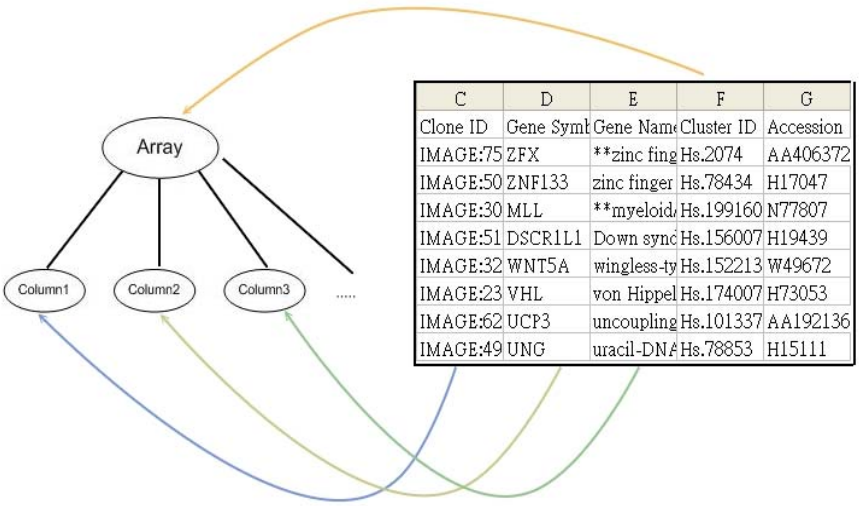


圖 6　Entity 與 Microarray Raw Data 的對應圖

在定義完 Array 的類型之後，每當我們要對存入資料庫存入一個微陣列矩陣的資料時，便需要去建立一個屬於這個資料的實體。表(3)是一個屬於 Array 類型實體的定義。在表(2)中類型的定義僅指出這個類型會有”Date”以及”Owner”這兩個參數，屬於這個類型的所有實體，皆會包含這兩個參數，所以在實體的 XML 定義時，需要給這兩個參數設定值。在實體產生之後，使用者才能針對這個實體去儲存屬於這個實體的資料。

```
<entity name="Array1">
    <attr name="Date" value="2004/01/01"/>
    <attr name="Owner" value="Poseidon"/>
</entity>
```

表 3　Entity Xml 定義

3.3 程序 (Procedure)

相較於在格網環境上撰寫應用程式，我們系統提出了讓使用者編寫能夠在資料庫端執行的程序。使用者利用 XML 來編輯一段程序，在編輯程序的過程裡，我們提供了<if>的條件判斷，<loop>的迴圈控制，以及<call>來呼叫已存在的

oracle 內儲程式，並且允許使用者在程式中插入檢查點<check point>，檢查點的插入，讓程序執行在過程中可以顯示目前執行的進度。在編譯完成之後，系統會將此 XML 的程序轉譯成 oracle 內部可執行的內儲程式並儲存之。讓使用者下次可直接執行此程式而無須再次編輯。

```
<proc name = " Example " >
    <arg name="A" />
    …
    <if>
        <begin>
            …
        </begin>
        …
    </if>

<checkPoint label=" Point " />
    <loop>
        …
        <endloop>
            …
        </endloop>
    </loop>

    <call name=" Procedure Name " >
        …
    </call>

</proc>
```

表 4 Procedure Scheme

在表(4)中看到的是一個利用 XML 來編輯一段程序的大綱，在這段程序一開始便定義了這個程序在執行時所需要傳入的參數，這些參數在程式中會不斷地被使用。在標籤<if>中，使用者必須在<if>的子標籤<begin>中，定義這個判斷式會開始執行的條件，當判斷式條件成立後才會繼續開始標籤內的程式。標籤<call>

的屬性”name”，則代表被呼叫的內儲程式在資料庫內的名稱，同樣的，每個內儲程式也有自己定義的傳入參數值，也需要在此給定。標籤<loop>則是定義一個迴圈的程式，其中的<endloop>標籤，定義著這個迴圈結束執行的條件。標籤<checkPoint>則是提供給使用者一個類似 debugger 的機制，label 這個屬性，讓使用者定義程式執行的一個標記，當這個內儲程式在執行時，系統會有一個使用者介面來顯示目前執行到的檢查點。因此檢查點的設計，能夠向使用者報告目前程式執行的程度。當程式中呼叫到其他的內儲程式，且此內儲程式中也有包含檢查點時，亦能夠同時在使用者介面上顯現出來。

上述以<proc>為標籤來定義的程序，系統會將其轉譯成資料庫內可執行的內儲程式，並且儲存進本地端的資料庫。除此之外，我們還定義了另一種以<prog>為標籤的程序。

```
<prog name= "test">
    <call name=" procedure name" >
        <hidb name=" server 1 "
             … />
        <arg A ="" />
        …
    </call>

    <copy  tblid=””>
        <hidb name="server 2"
             …/>
    </copy>
</prog>
```

表 5　Prog Scheme

與以<proc>定義的程序最大的不同，<prog>定義的程序，可以看做是一群工作的集合，在定義完後後並不會編譯成內儲程式儲存。表(5)中我們看到的範例，在<prog>中的每個子標籤，可以看做是一個個獨立的工作，每個工作可以是

呼叫資料庫內的內儲程式，或是傳送資料至遠端資料庫。<call>這個標籤是要去呼叫 name 這個屬性代表的存在資料庫內的內儲程式，而其中的<arg>的子標籤則是指定這個內儲程序執行時所需要的參數，子標籤<hidb>則是指定了要去對哪一台資料庫主機進行呼叫這個動作，標籤內的參數給定了對資料庫建立連線的資訊。接下來的<copy>標籤，則會完成資料傳輸與儲存資料兩個動作，這個工作會將本地端內的矩陣類型的資料產生.dmp 檔案，在傳輸到指定的機器上並會同時完成輸入至資料庫的動作，而達到資料複製的目的。同樣的使用者必須要在子標籤<hidb>中定義目的地主機。

利用<prog>編輯的程序，讓使用者決定了要將哪份工作分派至哪一台主機，或是要將資料傳送至某台主機，也可以利用遠端主機的內儲程式對來自本端輸入的資料作運算，來達到計算分散化平行化的目的。編輯程序完整的 DTD 在附錄(二)。

在經由<proc>標籤所定義的程序在編譯之後，會轉譯成資料庫可執行的內儲程式並儲存在資料庫內，為了對應儲存在資料庫內部的內儲程式，我們定義了程序(procedure)這個類型。在 Oracle 的 DBMS 環境下，當使用者建立了內儲程式後，除了直接呼叫執行之外，並無法利用資料庫一般的 DML(Data Management Language)來選取，使用者無法得知此資料庫內已存在哪些內儲程式可應用，所以我們除了讓使用者以 XML 定義內儲程式之外，也將定義的 XML 程式以及編譯後的內儲程式，一同儲存在資料表中。表(6)為程序這個類型的定義。

```
<type name = "Procedure">
    <has ints = "false"
        strs = "true"
        image = "false"
        floats = "false"
        refs = "false"/>
</type>
```

表 6　Procedure Type 定義

3.4 矩陣類型 (Matrix Type)

由於利用 generic data-type schema 來儲存資料，存放在在資料庫的是中各實體與類型之間的複雜關係。而要將各實體所包含的資料做資料分析或運算，則需要一些資料選取的資料庫操作，因此會增加系統執行的時間。所以我們定義了另一個類型的資料，矩陣類型(Matrix Type)。存放在資料庫內各實體的資料，皆能夠讓使用者選取，並經由程式的轉換而成矩陣類型。我們希望能夠利用矩陣類型的特性，來完成使用者定義的資料運算或分析。

定義矩陣這個類型，對於在運算過程中所需要的矩陣運算，有著極大的幫助。我們將一個矩陣分成以每個直行(column)為單位，所以在定義類型的時候，定義了直行這個類型並將直行定義成矩陣的子類型。由於定義矩陣這個類型的目的是為了方便運算，我們將儲存在資料庫的資料型態定義為浮點數。圖(7)便是一個矩陣類型的例子，表(7)是矩陣類型的定義。

Tbl id = 1

Row Id \ Column Id	1	2	3	4	5	6	7
1	34.30	28.32	24.25	23.04	17.58	15.74	10.47
2	36.94	30.50	26.12	24.82	18.93	17.41	11.58
3	39.59	32.69	27.99	26.60	20.29	19.08	12.69
4	41.94	34.64	29.66	28.18	21.49	20.27	13.48
5	44.64	36.86	31.56	29.99	22.88	21.94	14.59
6	48.10	39.72	34.01	32.32	24.65	23.85	15.86
7	51.90	42.86	36.70	34.87	26.60	26.24	17.45
8	55.89	46.15	39.52	37.56	28.64	28.38	18.87
9	60.27	49.77	42.61	40.50	30.89	30.77	20.46

圖 7 Matrix 範例

```
<type name = "Matrix">
    <has ints = "false"
        strs = "false"
        image ="false"
        floats = "true"
        refs = "false"/>
    <child name = "Column" >
        <has ints = "false"
            strs = "false"
            image = "false"
            floats = "true"
            refs = "false" />
    </child>
</type>
```

表 7　Matrix Type 定義

以微陣列矩陣為例，要從 array 的資料庫實體中取得數值，必須要經過前處理(preprocess)的步驟。表(8)是我們以微陣列矩陣的前處理為例，使用者可以選擇要從哪些 array 實體選擇資料，由於每個 array 內的資料是以直行(column)實體作為儲存的單位，因此使用者下一步便需要決定從各 array 實體中來選擇哪些直行的資料。在完成選擇之後，會利用各 array 的列索引的實體來當作此矩陣(Matrix)的索引值，當 array 的列索引相同時，資料會放矩陣的在同一列上，若不同則會將資料新增至矩陣的下一列，如此來完成一個可運算的矩陣類型。

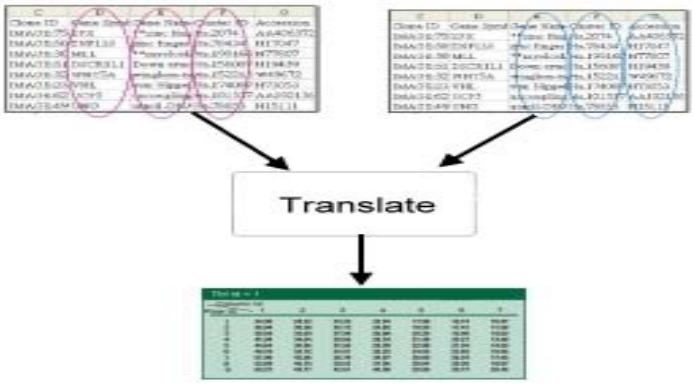


圖 8　preprocess 圖示

```
<preprocess>
    <hidb name="server1"
         … />
    <array id= " ">
       <col  name=" A "/>
       <col  name=" B "/>
        …
   </array>
   <array id="">
        …
    </array>
    …
</preprocess>
```

表 8　preprocess xml

3.5 資料傳輸

完成了每個獨立的資料庫的建置之後，我們的目的是希望能夠將各資料庫串聯起來，資料庫之間的資料能夠共享，並且能夠將運算平行化，而成一個分散式的資料庫服務系統。

在我們的系統中，我們有一個以 Java 實做的服務(Service)來負責使用者與資料庫之間的溝通，每個資料庫旁都會伴隨一個服務。在服務與服務之間，利用 Socket 來傳遞訊息以及資料，而訊息是以 XML 格式來定義。舉例來說，要將資料庫之間的資料共享，使用者可以將某個矩陣型態的資料輸出(export)，而為了效率的提升，我們使用了 Oracle 的資料備份(Recovery)機制，所以資料在輸出後，是以 oracle 備份資料的.dmp 檔案格式存在。在產生了備份檔案之後，使用者便可利用 XML 格式的指令來將檔案傳輸到指定的機器上，並會自動的將資料使用 Oracle 資料備份的機制，輸入(import)至相同的表格。於是使用者便可利用傳送的資料來進行所需要的資料分析。

```
<prog>
<copy  tbl_id = ‵″>
    <hidb name="poseidon"
          host="140.113.88.45"
          port="1119"
          path="db"  />
</copy>
</prog>
```

表 9　copy 檔案範例

在表(9)中可以看到，使用者若要對遠端的資料庫進行操作，則必須在<hidb>這個標籤中指定目的地的機器，包含了主機位置、接受埠及路徑。這個操作會將指定資料庫的矩陣的 id，輸出成.dmp 格式的檔案，並傳遞到以<hidb>標籤內指定的機器上並將備份的資料載入至資料庫內，所以在指令中也需要以 file 這個標籤來說明欲傳輸的資料在本地端電腦中的位置。

第四章　系統實作

在本章中，我們將對第三章提出的設計，做更進一步的闡述，並以相關的範例來作詳細的說明。

4.1 資料儲存

我們的資料庫服務核心，是使用在上一章介紹過 generic data-type schema。接下來我們要說明，系統如何對定義好的實體儲存到資料庫中。

4.1.1 記錄 (Record)

在我們定義的 generic data-type schema 中，我們將每筆要存入的資料視為一筆記錄(record)，每筆插入的資料都會儲存在 Rec 這個表格內。而一個記錄除了儲存在 Rec 表格外，其本身亦有可能會包含其他的資訊，以微陣列矩陣的記錄來說，一個原始資料的記錄除了數值之外，還有可能包括了實驗日期、擁有者以及實驗設計等等的參數。我們將這些資訊存放到另一個叫 Attr 的表格，視為這個記錄的參數。每個記錄可以有零到多個的參數，由於是將記錄與參數分別存放到兩個表格內，我們利用此兩表格不會重複(unique)的 record id 來當作鍵值(foreign key)，來維持兩個表格間的一致性。

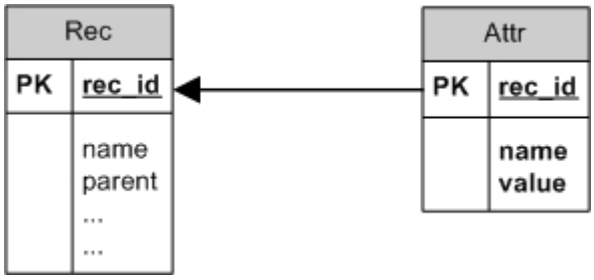


圖 9　Rec 表格與參數表格

一個記錄除了參數之外，以微陣列矩陣的原始資料而言，其中更有著數十萬筆的數值需要存入。若將一個點的數值對應到一個資料表的一個值組(tuple)，對於將巨量的資料存入至資料庫來說來說，更是一個龐大的負擔。所以在我們的設計中，一個記錄除了參數之外，對於每個記錄而言，還有著與資料的資料型態

相對應的表格，來儲存這個記錄所會包含的資料。我們預設了三種資料型態，包含了字串(String)、整數(Int)以及浮點數(Float)。以微陣列矩陣的資料來說，由於將原始資料中的每個點的值視為一筆資料太過繁雜，因此我們將原始資料的一個直行(column)定義為一個記錄，系統會依照這個記錄所包含的資料的型態，將資料存放到相對應的表格中，舉例來說，若資料本身的型態是字串型態，則會將資料儲存到 Strs 的表格中，同理，若資料包含的是浮點數，則會將資料儲存到 Floats 的表格中。在儲存的過程中，系統會依照資料本身的順序建立索引並依序存入，以便將來從資料庫將資料取出時能夠不失去原本的順序。而 Strs、Ints 與 Floats 這三個表格，也是利用利用不重複 record id 來當作與 Rec 表格之間的鍵值，來維持表格間的一致性。

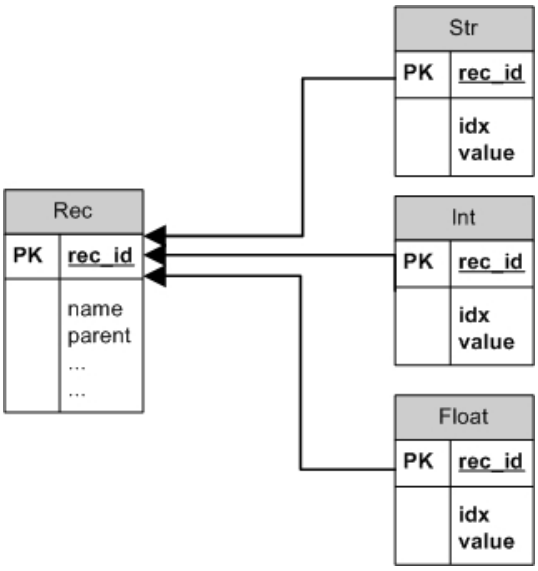


圖 10　Rec 表格與資料表格

4.1.2 親代(Parent)

在定義完實體類型之後，我們可以方便地依照使用者所需要的類型來取得或儲存資料。然而，當類型的數目增加後，類型與類型之間亦可能產生關連性。以微陣列矩陣來說，我們將原始資料的直行定義成一個類型，而矩陣也定義成一個類型，一個矩陣內包含直行，所以直行屬於矩陣。這種類型之間的關係，我們希望能保存下來。所以在 Rec 表格中，我們定義了親代(parent)這個欄位。

這樣的定義方法提供了類型間單向的連結，讓有親子關係的類型其中的子類型，在親代這個欄位填入其父代類型的 record id，於是我們便能夠階層式的從父代的類型，一代一代的找出與其有關連的子代類型，同時，亦能夠找出屬於各類型的記錄，將類型之間的關係完全表達，而完成關連性的儲存。

4.1.3 標籤 (Tag)

由於一個親代的類型，其可能會有多個的子代類型，因此使用 parent 這個欄位，僅能表示出類型之間有著親代與子代的關係，若要列出親代類型的子代類型，系統會將親代類型的所有子代類型都選取。為了解決此問題，我們在 Rec 表格中定義了標籤(Tag)這個欄位。所以當使用者定義一個類型時，對於其每個子代的類型，需要多定義一個 tag 的參數，使得我們在選取了一個父代類型後，可以決定在他的子類型中，只選出 tag 是"a"或是某個特定符號的子代類型，讓整個資料庫內的資料的關連性更加完備。

完成上述的定義之後，我們的 Rec 表格包含了類型、親代、以及 tag，每個記錄有著自己的參數以及相對應資料型態的資料，並且屬於某個類型的屬性，而每個類型之間有著親代與子代的關係，所以屬於各類型的記錄亦能有著同樣對應的關係。再加上能夠使用 tag 來選定每個類型特定的子代類型，來完成我們的 generic data-type schema 的底層儲存架構。

4.2 程序(Procedue)

在上一章我們說明了利用<proc>及<prog>兩種標籤來編輯程序的方法，接下來我們用一個實例來說明。

由於為了讓使用者在編寫程序時，能夠更快速方便的處理一些基本的數學運算，我們定義了一些基本矩陣的運算，以資料庫內部提供的內儲程式來實作，讓使用者可以直接使用。這些運算包括了矩陣的相加(table_add)、矩陣相減(table_sub)、矩陣相乘(table_mul)、以及取得矩陣內指定位置的值(getvalueat)。除了這些矩陣數學運算之外，由於在某些演算法的計算過程中，會產生大量的計算暫存資料。為了提高計算效率以及避免硬體空間的不足，我們將計算過程中所產

生需要儲存的暫存值，存放在暫時性表格中(temporary table)。

暫時性表格的特點為，能夠在使用者執行程式時，動態地配置表格空間，且資料只有在使用者程式與資料庫的工作階段(session)仍保持連線時才會存在，當使用者與資料庫的工作階段結束，暫時性表格中的資料也會自動地清除。如此在計算過程中，使用者無需為了計算中暫時性資料的儲存與清除而去費心，亦能減少硬碟空間的浪費。

接下來我們便利用上述所提出的方法來編輯程序，使用一些已經預先定義好的運算，以及條件與迴圈的控制，來完成一個複雜的演算法運算。接續微陣列矩陣的例子，在將資料建立實體與實體類型，並存入資料庫後，並利用前處理的方法產生了矩陣類型的資料後，我們下一步便希望能夠對此矩陣作出有意義的分析。因此我們以我們提出的程序的編輯方法，利用 XML 的格式，來完成 k-means clustering 的分析。k-means 演算法的 pseudo-code 如表(10)。

Initialization

1. Set k cluster centers.

Main Loop

1. Calculate distances from all elements to all cluster centers
2. Assign every element to the nearest cluster
3. Recalculate every cluster center
4. Repeat main loop until no cluster center will move

表 10　k-means pseudo-code

在 k-means 的運算一開始，必須要先給定初始的群中心，接下來所有的資料點要計算與各個群中心的距離，並將其指定到距離最近群中心所屬的那個群集。在每個資料點都指定群集之後，重新計算每個群的群中心，並重複每點計算與群中心距離的步驟，直至群中心不再改變為止。

```
<proc name=”kmeans”>
    <arg tbl=”inputTable”/>
    <table name=”temp1”>
        <createTable name = “inputTable” />
    </table>
    <table name=”seed”>
        <createRow>
            <getValueAt tbl=” inputTable” row=”1” col=”1” />
        </createRow>
        <createRow>
            <getValueAt tbl=” inputTable” row=”2” col=”2” />
        </createRow>
    </table>
    <Loop>
        <table name=”temp2”>
            <call name=”table_sub”>
                <arg name=”temp1”/>
                <arg name=”seed”/>
            </call>
        </table>
        <table name=”temp3”>
            <call name=”table_mul”>
                <arg name=”temp2”/>
                <arg name=”temp2”/>
            </call>
        </table>
        <table name=”temp4”>
            <call name=”table_sqrt”>
                <arg name=”temp3”/>
                <arg name=”temp3”/>
            </call>
        </table>
        <table name=”seed_new”>
            <createRow>
                <calMean tbl=”temp4” groupby=”clus” />
            </createRow>
        </table>
```

```
        <table name=”result”>
            <call name=”table_sub”>
                <arg name=”seednew”/>
                <arg name=”seed”/>
            </call>
        </table>
    <table name=”seed”>
        <copy name = “seednew” />
    </table>
    <endLoop>
            <if>
            <check tbl="result" row="2" col="1"  op="<" value="0"/>
            </if>
    </endLoop>
        </Loop>
    <return table = “temp4”/>
</proc>
```

表 11　k-means procedure example

在表(11)可以看到利用程序編輯 k-means clustering 的方法，利用了許多已事先定義好的矩陣運算來完成。在我們的方法中，一開始便先將欲運算的資料存放在一個新的暫存表格，接下來並讓使用者定義初始的群中心。在決定了各群的中心之後，所有的資料值皆須與群中心計算距離，在這裡我們是先讓資料與群中心相減，之後再對相減的值取平方並開根號，而這些計算過程的值，皆會另外存放在暫存表格。在算完距離之後，我們對計算結果重新計算群中心，並將新的群中心的值存放在新的群中心表格中。藉由比較新舊群中心的距離，若中心已沒有移動，則會結束迴圈，完成演算法的執行。

除了在資料庫端執行的 k-means 內儲程式，我們亦利用符合附錄(二)所定義的 DTD 來編寫一個平行 k-means 程式[26]。在表(12)中列出了平行 k-means 的 pseudo-code。程式一開始會將欲運算的資料以及初始群中心，由一台主機分配給各參與運算的機器，各機器利用所得到的部分資料來進行 k-means 的運算，並將

運算結果的群中心傳回至主機。在得到各機器的運算結果後，主機器再計算新的群中心，並重複之前的步驟，直到群中心不變為止。圖(11)為平行 k-means 程式的示意圖。附錄(三)中包含了完整的程式。

Master Process

1. Send each subset and cluster centers to each of the *K* slaves
2. Receive *K* resulting subsets from *K* slaves

Slave Process

1. Receive a subset data *P* and centers from master process
2. Compute cluster centers of the subset *P*
3. Send the calculating result to master

表 12　平行 k-means pseudo-code

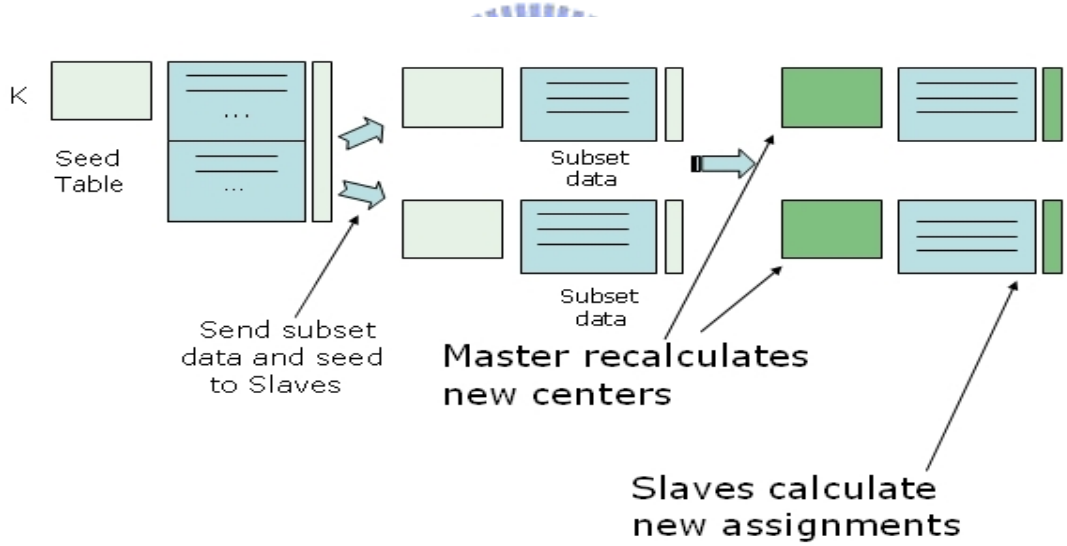


圖 11　平行 k-means 示意圖

```
<table name="seed">     <!-- initial seed table -->
    <createRow>
            <getValueAt tblid="inputTable" row="1" col="1" />
    </createRow>
    <createRow>
        <getValueAt tblid="inputTable" row="2" col="2" />
    </createRow>
</table>
```

表 13　平行 k-means 程式片段 1

在表(13)中是我們利用了符合附錄(二)定義的 DTD 所編輯的平行 k-means 程式的片段。在這個片段內，我們讓使用者可以自輸入的表格中，指定位置來當作 k-means 程式開始執行時的初始群中心，將其以矩陣型態儲存，並可以指定這個初始群中心的變數。

```
<par>
      <task>
            <assign id="idA">
               <copy tblid="parTable1" >
                   <hidb name="poseidon" host="140.113.88.45"
                         port="1119" path="db"/>
               </copy>
            </assign>
     </task>
     <task>
           <assign id="idB">
              <copy tblid="parTable2" >
                     <hidb name="poseidon" host="140.113.88.140"
                           port="1119" path="db"/>
              </copy>
           </assign>
     </task>
</par>
```

表 14　平行 k-means 程式片段 2

表(14)的程式片段說明了程式執行的下一個步驟。在定義了初始群中心之後，主機會傳送將欲在各機器上執行運算的部分資料，傳遞到各機器上。在這個片段中的標籤 par 之內，讓使用者可以在此定義需要平行化的工作，亦即在標籤 par 之內的各個工作，會平行的分配給各指定的機器去執行。而每個工作，則是以標籤 task 來定義，工作可以是傳遞資料至遠方的資料庫，或是去呼叫指定機器上的資料庫內儲程式。因此每個 task 的標籤內，必定要包含 hidb 這個子標籤，來決定此工作會分配到哪一台機器上去執行。而在 task 標籤中的子標籤 assign，則是代表了指定變數的意義。無論是傳遞資料或是呼叫內儲程式，在執行之後資料庫端都會產生一個與儲存新資料對應的 id，而這個 id 在程式的內部需要繼續使用，所以我們利用 assign 這個標籤來完成指定變數的功能。

```
<Loop>
       <par>     <!-- partition jobs to remote clients -->
           <task>
               <copy tblid="seed" >
                    <hidb name="poseidon" host="140.113.88.45"
                          port="1119" path="db"/>
               </copy>
            </task>
            <task>
               <copy tblid="seed" >
                    <hidb name="poseidon" host="140.113.88.140"
                          port="1119"  path="db"/>
               </copy>
            </task>
        </par>
        <par>
            <task>       <!-- each client executes k-means -->
               <assign id="id1">
                  <call name="Kmeans" >
                     <tbl id = "idA" />
                     <hidb name="poseidon" host="140.113.88.45"
                       port="1119"  path="db"/>
                  </call>
               </assign>
            </task>
            <task>
               <assign id="id2">
                  <call name="Kmeans" >
                      <tbl id = "idB"/>
                      <hidb name="poseidon" host="140.113.88.140"
                            port="1119" path="db"/>
                  </call>
               </assign>
            </task>
       </par>
```

表 15　平行 k-means 程式片段 3

在完成對各機器的部分資料傳遞之後，程式便進入迴圈控制部分。表(15)的程式片段，在第一個 par 這個標籤中，平行的對各機器傳遞定義完成的初始群中心表格。在各機器都接收到執行運算的資料以及初始群中心後，迴圈內的第二個 par 的標籤中，利用之前已經指定好的矩陣的變數，來呼叫各機器上的內儲程式，各自完成運算，並將運算結果的表格 id 指定到變數中。

```
<table name = "result2">  <!-- get client's result -->
    <get tblid= "id2">
        <from>
            <hidb name="poseidon"  host="140.113.88.140"
                  port="1119"  path="db"  />
        </from>
        <to>
            <hidb name="poseidon" host="140.113.88.45"
                  port="1119"  path="db"  />
        </to>
    </get>
</table>
<table name="new_seed">   <!—calculate new cluster center -->
   <call name="calMean">
      <tbl id="id1"/>
      <tbl id="result2"/>
        <hidb name="poseidon" host="140.113.88.140"
             port="1119" path="db"/>
   </call>
</table>
```

表 16　平行 k-means 程式片段 4

在各機器完成運算之後，主機需要得到各機器所運算的新的群中心，因此在表(16)的第一個 table 標籤中，我們利用了 get 這個標籤，可以從以 from 標籤指定的機器上，將資料傳遞至以 get 標籤指定的機器中。在第二個 table 標籤中，則是利用自各機器得到的計算結果，來計算新的群中心。

```
    <endLoop>   <!-- check end condition -->
       <if>
        <table name="result">
           <call name="table_sub">
               <arg name="new_seed"/>
               <arg name="seed"/>
           </call>
        </table>
        <check tbl="result" row="2" col="1"  op="<" value="0"/>
       </if>
    </endLoop>
    <table name = "seed">
        <copy tblid="new_seed" >
           <hidb name="poseidon" host="140.113.88.45"
                port="1119" path="db"/>
        </copy>
    </table>
</Loop>
```

表 17　平行 k-means 程式片段 5

在表(17)中，endLoop 這個標籤，則是定義了迴圈結束的條件。利用新計算出的群中心與一開始的初始群中心的相減，來檢查群中心的位置是否不會改變。如果群中心已不再移動，則會結束迴圈並結束程式，反之，則會將新的群中心的值指定到 seed 這個變數，並繼續執行迴圈，直到滿足結束條件。

由於微陣列矩陣資料分析完的結果，資料是以數值型態的方式儲存，無法直接做判讀，所以我們將結果圖形化的呈現。

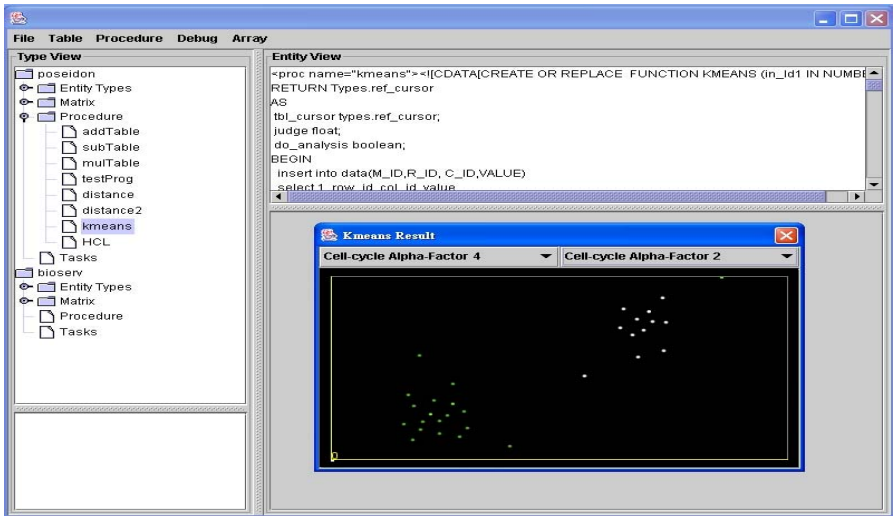


圖 12　K-means Analysis Result

4.3 分散式資料庫環境建置

在建立了分散式的資料庫系統之後，我們實做了 HiDB 的系統，讓使用者能夠在同一個使用者介面中，瀏覽系統內所有的資料庫，而在程式的執行之前，使用者必須建立一個 config 的初始設定檔案，讓程式知道欲連線的資料庫的名稱、位置等資訊以建立連線。

表(18)是一個建立連線的 config 檔，讓系統知道在程式開始執行要去跟哪些資料庫建立連線，在這個設定檔的範例當中，有兩個我們欲連接的資料庫，分別以<hidb>的標籤定義，而各個資料庫在連線時所需的資訊，包括了主機名稱、位置、使用者名稱、密碼以及資料庫的驅動程式，皆需要清楚完整的定義。在完成連線後，便能夠在一個主介面上瀏覽所有資料庫內的資訊。圖(13)則是整個使用者介面的概觀。

```
<cfg>
   <hidb name="poseidon"
         user="system"
         password="manager"
         url="jdbc:oracle:thin:@140.113.88.45:1521:test1"
         driver="oracle.jdbc.driver.OracleDriver"/>
   <hidb name="bioserv"
         user="system"
         password="manager"
         url="jdbc:oracle:thin:@140.113.88.140:1521:bioserv"
         driver="oracle.jdbc.driver.OracleDriver"/>
</cfg>
```

表 18　Config 檔

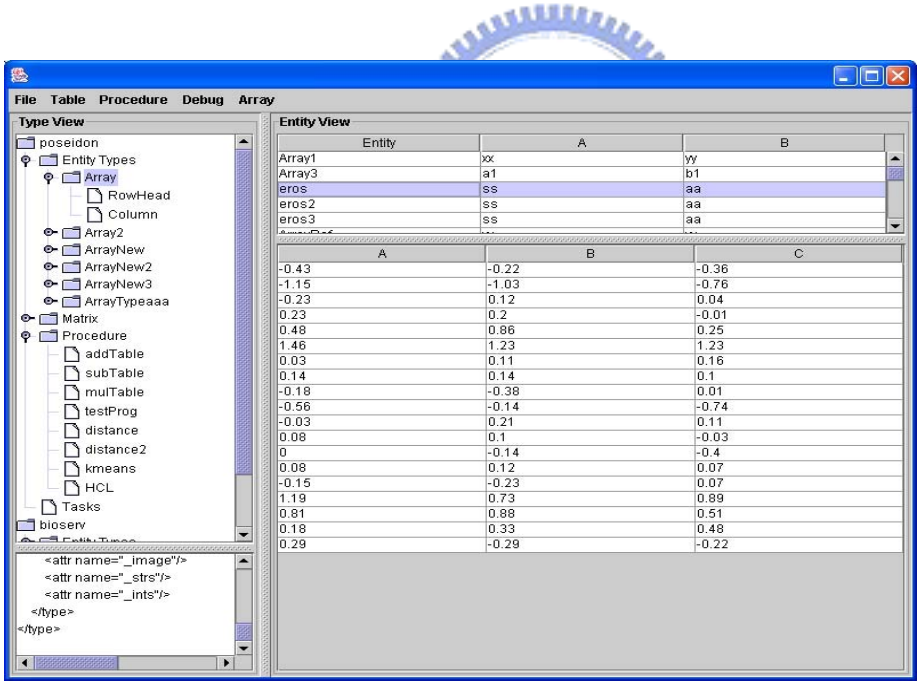


圖 13　HiDB Console

第五章　實驗結果

為了瞭解我們系統的可行性，我們設計了一些實驗來檢驗我們系統的效能。由於系統是建置在各資料庫之上，因此我們預期系統在處理大量資料時，能夠維持一個穩定的執行狀態，在一般的運算上，也能夠有著可接受的效率。我們實驗測試的環境為：

CPU：Intel Pentium4　2.0G

Memory：256 Mb

Hard Disk：Maxtor 80Gb

OS：Windows XP

Database：Oracle 9.1

5.1 矩陣加法的比較

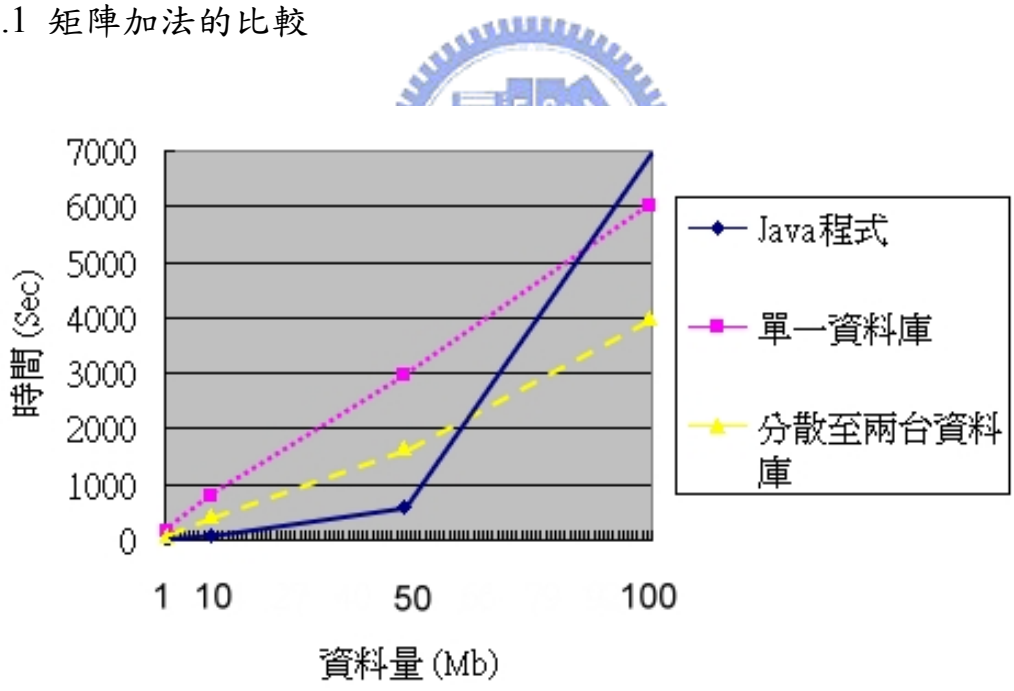


圖 14　矩陣加法效能比較

圖(14)顯示的是利用 Java 程式執行矩陣加法 10 次，以及利用單一資料庫的內儲程式來執行矩陣加法 10 次，以及將資料分散至兩台資料庫利用內儲程式各自執行矩陣加法 5 次運算的結果比較。在資料量小於 50Mb 時，明顯的 Java

程式相較於資料庫內儲程式的運算有著較高的效能。但是在資料量增加成為 100Mb 後，由於矩陣加法在兩個矩陣相加後，需將結果存至第三個矩陣，所以程式執行時會需要 3 倍的計算空間，已超過了機器的實體記憶體，即使在執行之前改變 Java 執行環境(Java Runtime Environment)的旗標來增加 heap 和 stack 的大小，但是 Java 程式在單一資料庫做完運算的時間內，仍無法結束運算。相較之下，雖然資料庫的運算在資料量較小時，需要較多的時間來完成運算，但是在資料量增加到一定大小後，還能穩定的完成運算。

5.2 矩陣乘法的比較

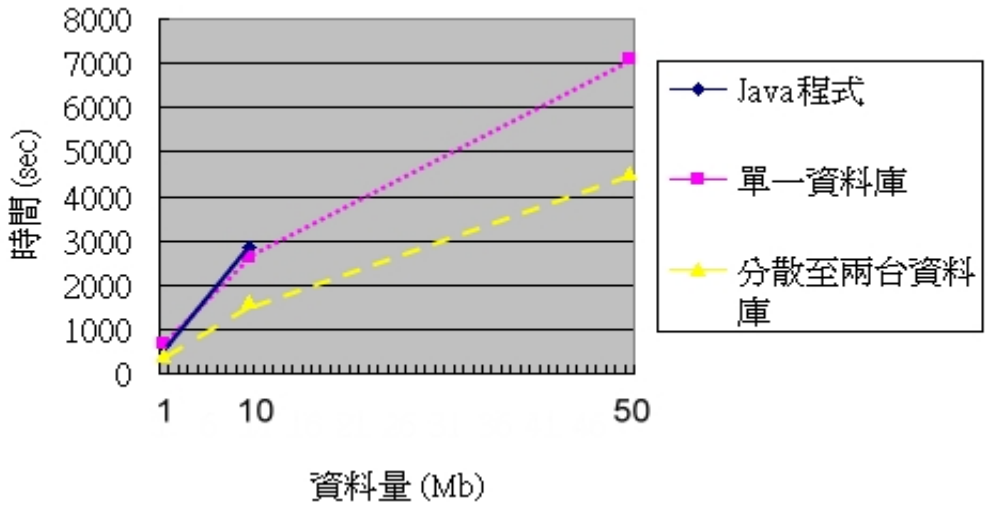


圖 15　矩陣乘法效能比較

除了矩陣加法之外，我們還測試了更耗費資源的矩陣乘法，其結果如圖(15)所示。對一個 N 維矩陣來說，矩陣乘法中矩陣的每一個位置的值需要經過 N 次的乘法及 N-1 次的加法所獲得，隨著資料量的增加，比起矩陣加法，計算的複雜度更是可觀。以 1Mb 的資料量來說，Java 程式的實做，矩陣乘法所花費的時間將近是矩陣加法的 100 倍，且當資料量增加到為 50Mb 時，程式執行時會發生超出記憶體的錯誤而無法執行。然而使用內儲程式執行的矩陣乘法，在執行上所花的時間並沒有如同 Java 程式那般暴增。

其間的差異，經過我們的分析，由於資料庫內儲程式能夠直接對儲存矩陣

的資料表格做存取，且矩陣存放在我們定義的 tbl 表格中，每個位置的值皆有 row_id 與 column_id 來作為資料表的鍵值，而這兩個欄與列的索引在矩陣的運算上，提供了相當程度的便利，加速了矩陣乘法的運算。當資料量增大時，資料庫的運算依然保持穩定的效能，並不像一般程式容易造成執行失敗的情形發生。因此在面對大量的資料且在某些大量計算的情形下，以資料庫來完成運算，會較一般使用程式來的更加的有效率及穩定。

5.3 資料傳輸的比較

我們建立的環境中，在服務與服務之間，底層訊息傳遞的機制是利用 socket 來傳送 XML 的訊息，由於各獨立的資料庫能夠將資料輸出成備份格式，並指定傳入至特定的資料庫上，我們希望檢視我們所利用的資料傳輸的機制，與目前常用的方法相較之下，是否會造成效能上的負擔。以格網環境為例，資料傳輸的服務利用了 GridFTP 這個通訊協定，而 GridFTP 則是建立在目前通用的 FTP 協定之上，所以我們使用了 socket 與 FTP 來傳輸資料，作了簡單的實驗。

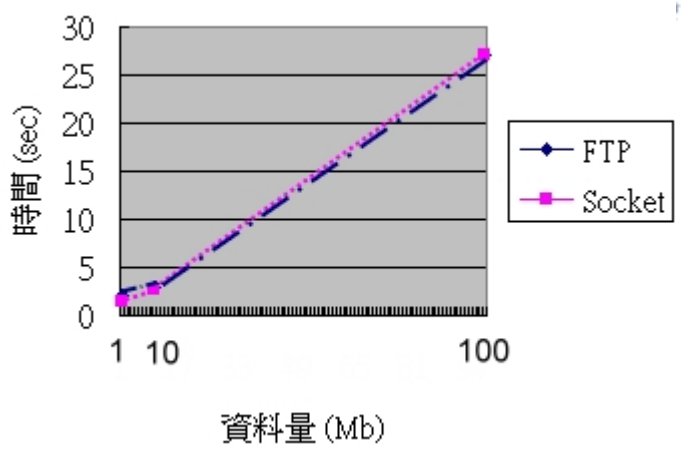


圖 16　資料傳輸效能比較

結果顯示，我們使用 socket 來傳遞資料，與在格網環境中使用的 FTP 相較，資料量增加的情況下，兩者皆有著類似的效能，因此我們的系統在資料傳輸的效能上，也能保持一個穩定的狀態。

5.4 k-means 程式比較

在我們的系統中提供了利用 XML 來完成程序的編寫，並由於我們的系統架構能夠支持平行程式的運算，所以我們編寫了 k-means 的平行程式，希望來比較在我們系統的環境下將程式平行化並執行，與利用單獨資料庫的內儲程式來完成運算之間的效能的差別。

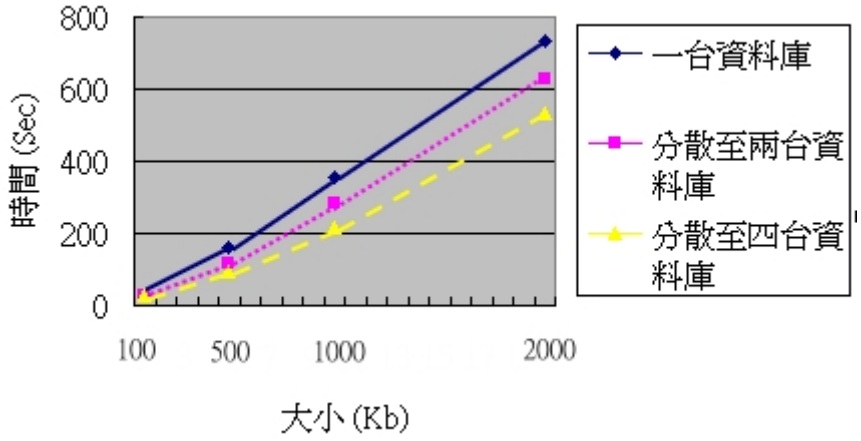


圖 17 k-means 比較

由實驗結果可以看出，將資料分散至兩台資料庫上執行所需的時間，與在單一資料庫上執行的時間相比，並非理想中的二分之一；同樣的，分散至四台資料庫執行的時間，也大於單一資料庫上執行所需時間的四分之一。分析差異造成的原因，由於在計算完成後，尚須將結果傳回至主機，且在整個程式的執行過程中，需要資料的傳遞時間計算在內，以及執行運算的各台機器的狀態不一，在平行計算後，主程式需要得到每一台運算完的結果，才能繼續程式的執行，因此若有某台機器的執行速度較慢，則會降低程式整體的執行速度。加上由於在平行 k-means 程式一開始，需要從主機上將運算的資料經由資料庫輸出(Export)、傳遞、以及資料庫輸入(Import)至各機器上，這個部分也佔去部分執行時間，所以我們將平行 k-means 程式中，一開始傳輸部分資料的時間扣除，單純比較運算之間的速度差異，實驗結果在圖(18)。

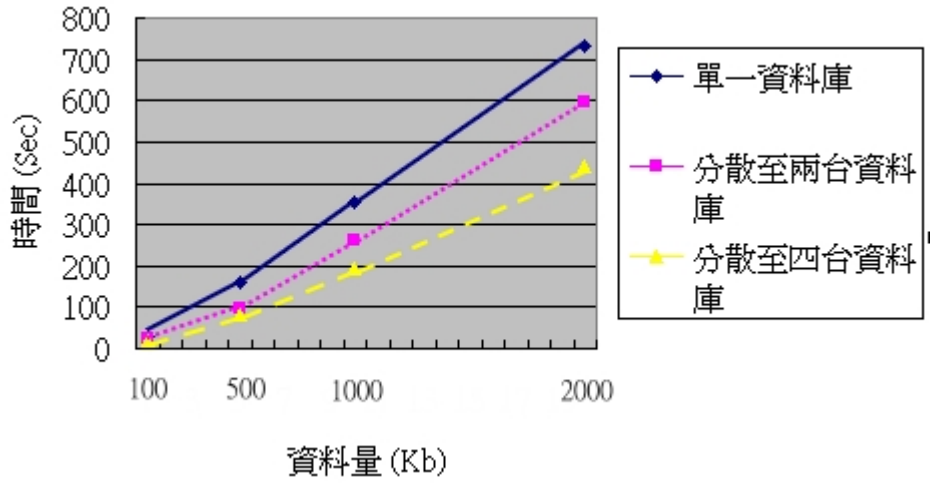


圖 18　平行 k-means 運算比較

整體而言，將運算切割並分散至多台機器上執行，在執行大量運算時，由於各台機器僅需要處理部分的原始資料，因此在計算時間上能夠有效的減少。相較於在單一的機器上運作，花費較短的時間而能提供較佳的效能。

第六章　相關研究

本章我們要介紹在目前現有的研究計畫中，與我們系統有著類似目標與動機的研究。

無論是在生物資訊領域，或是一般企業組織，各部門單位使用不同資料庫管理系統來儲存和取得其重要資料，這幾乎是不可避免的，但這些系統常常來自不同的廠商，通常情況下，會需要通過一個公用介面來使用兩個機構中經過組合的資料。在這些情形下，資料庫的聯邦技術通過提供異構資料的統一介面有效地解決這一問題。

在參考文獻[13]中，便是以生物資訊為研究領域，探討在對各處理生物資訊資料且分散的資料庫所建立的邦聯，發展一個中介的程式(mediator)。此中介程式能夠存取各分散資料庫的資料，並會利用預先收集好各資料庫的資訊與關係，建立一個整合式的資料表綱要，讓使用者能夠對這個整合式的綱要進行任意的資料庫的詢問(query)。若當有新的資料庫加入此邦聯，則需要對此中介程式增加資料庫本身的描述資訊，才能重新整合並擴展整合式的綱要。

與其相較，我們建立的分散式資料庫環境中，由於各個資料庫皆使用相同的 generic data-type schema 設計，所以在對資料庫的詢問上，不需要另外建立一個整合式的綱要，而能夠直接對資料庫作操作。加上我們的系統中使用一致的綱要，讓使用者在編寫程序時，對各個資料庫都能有著一致性的操作，而不像邦聯式的資料庫環境，對資料庫任何的操作皆需要透過中介程式。因此我們認為我們建置的分散式資料庫系統，有著較好的一致性與操作性。

而對於資料庫內的資料分析，另一個常見的方法為線上即時分析(On-Line Analytic Processing,OLAP)。 線上即時分析是決策支援必要的元件，因為傳統資料庫無法預測歷史性資料回歸分析、儲存方式速度太慢以及散佈各地的不同平台，查詢時需要結合(join)多個資料表，造成系統整體效能低落。加上傳統的資料庫技術並未提供組織決策市場的分析功能，於是資料倉儲的技術就是在這樣的情勢下所產生的。

[15]這篇論文中，研究了將資料探勘與線上時分析的整合應用，並將其發展成為工具。在他們的研究中提出，由於資料探勘需要對整合過且經過前處理的資料上進行，因此資料倉儲的特性能夠符合此項要求。加上線上即時分析提供的基本功能如 drilling、pivoting、slicing 等等對資料立方體(data cube)的操作，讓使用者可以在不同的層面的概念上作資料探勘。雖然如此，但是在作資料探勘時，由於目前已有許多的資料探勘的函數，針對各個不同的資料領域，使用者必須要能夠選擇適當的資料探勘函數，否則也無法獲得有意義的資訊。

而我們的系統在資料的儲存方式，是以 generic data-type schema 的概念儲存，因此資料庫內資料皆是以複雜的實體與實體類型的關係存在，並不會建立資料方體來提供分析。但是我們提出了矩陣類型，讓實體資料能夠被選取且結合成一個矩陣，並能夠利用此矩陣來作資料分析及一般運算。在資料分析方面，上述的論文提出了利用資料倉儲建立資料方體，並在其上以 OLAP 的方法來切割，當成資料探勘的領域，由於資料探勘是利用已定義完成的函數來對其資料作分析，所以函數的選擇相當重要。而我們的系統則是讓使用者編輯資料庫端能執行的程序，且程序提供了一些基本的運算，讓使用者能夠利用編輯的程序來完成複雜的演算法。

在[23]這篇論文中，提出了整合 P-GRADE 這個程式發展環境與加入檢查點的功能，能夠將平行程式的設計以及批次模式的執行程式，整合至 grid 的環境之下。在他們提出的程式模型下，平行程式要被當作能夠在各分散的節點上執行的一個工作，並且要有一個機制來決定在格網中最適合提供服務的節點。除此之外，在格網的環境下執行平行程式，另一項重要的考量便是錯誤的發現與解決，要能夠在格網分散的環境下找出平行程式執行過程中可能發生錯誤的連結。但由於格網環境的複雜性，所以目前並沒有一個具體可行的解決方法。因此使用檢查點來當作除錯的機制，則是目前較多人使用的方法。

在我們的系統中，使用者編輯程序的部分，我們亦提供了插入檢查點的功能，讓程序執行在過程中可以顯示目前執行的進度，檢查點的設計，能夠向使用者報告目前程式執行的程度。對於平行程式的除錯，是一項重要且基本的除錯機制。

在參考文獻[24]中，作者整理了目前在格網計算的環境上，所提出的網格程式模型(Programming Model)語言以及工具。在網格程式的發展中，可攜性、跨平台，資源的尋找，效能的要求以及容錯，皆是網格程式模型的需求。而在工具與語言方面，則依照其類型的不同，分為狀態共享 (Shared-State) 模型，訊息傳遞(Message-Passing)模型，混合 (Hybrid) 模型，點對點 (Peer-to-Peer) 模型，架構化元件 (Component) 模型，網路服務(Web Services)模型，以及協同(Coordination)模型。我們的系統在程序的編寫過程中，可以利用呼叫已定義好的內儲程式，來完成特定的運算，而這些運算可以被獨立成網路服務來單獨提供服務，因此在[24]的分類之下，我們的系統屬於網路服務模型。

第七章　結論

總結本文，對於個別的資料庫，有別於一般關連式資料庫中資料表綱要的設計，本文提出了使用 generic data-type 的設計來代替資料庫的資料表綱要，讓使用者在儲存資料時，無須重複定義綱要，只需要將欲存入的資料以 XML 來定義實體以及實體類型，且在各分散的資料庫之間都能夠通用。

相較於在格網環境上撰寫應用程式，我們提出了讓使用者編寫能夠在資料庫端執行的程序。使用者利用 XML 來編輯一段程序，程序提供了矩陣的基本運算外，還提供了條件判斷、迴圈控制、插入檢查點等功能，並能夠轉譯成資料庫端能執行的內儲程式。另一種格式的程序，除了讓使用者能夠呼叫已存在的內儲程式外，還能夠將工作指派給遠端的資料庫來進行運算，也能夠將本地端的資料傳遞到指定的機器上。

在建置完各分散的資料庫之後，我們利用 Java 實做了一個服務，每個資料庫皆會伴隨著一個服務，服務與服務之間使用 Xml 來溝通，並利用 socket 來傳遞訊息。此外，我們對資料庫內矩陣類型的資料使用 oracle 的備份機制，輸出成 dmp 格式檔案，並能夠利用程序的控制，傳輸到指定的機器上並完成輸入，來達到資料共享與備份的功能。

我們的系統目前的實作完成了建置分散式資料庫初步的環境。對於各資料庫間運算工作的排程、運算資源的預測及網路頻寬的偵測等等，都是我們接下去發展的目標，以期能夠達到一個類似格網環境，能夠平行運算且資料共享機制完全的一個高效能分散式資料庫運算環境。

附錄一 微陣列矩陣晶片簡介

自從人類基因序列被大量定序解碼後，許多與生物相關的資訊研究，被各領域的研究人員所提出，包括了大量的微陣列矩陣(Microarray)的實驗資料，基因序列的定序，基因功能的預測，以及將基因資訊應用在藥物研發與疾病治療。而如何處理儲存這些大量的資料，並從中去發掘並分析出對生物學家有幫助的資訊，則是現今生物資訊研究的重點

生物晶片的概念起源於二十世紀 80 年代後期，歐美許多研究單位體認到結合微電子、微機械、生命科學和生物訊息等的綜合產物---生物晶片 ，其發展和應用必將會二十一世紀帶來一場生物技術革命。總體來說，生物晶片研究在已有許多重大成果，如為陣列矩陣 (Gene Chip, DNA Chip or Microarray)、蛋白質晶片 (protein chip) 、微流體晶片 (Microfluidics) 及多功能處理晶片 (Lab-on-a-chip)。微陣列矩陣(Microarray)是一種可以同時平行測量成千上萬機音的 mRNA 含量，並間接說明基因的表現程度。在雙色的 cDNA 生物晶片(two-dye cDNA Microarray)中，更進一步比較兩種個體，組織或細胞基因表現程度的差異。

微陣列矩陣製程一般可分為兩大類：(一) In-situ 以光罩法（光蝕刻法）為主:仿半導體製程，生物探針直接在晶片上合成，(二)Ex-situ 以點片法(Spotting)為主：生物探針合成後再由點片裝置點到晶片上。點片的方式又可分針點式(pin)、噴墨式 (Inject Printing) 與壓電式(Piezoelectric-printing)。微陣列矩陣之製備將 cDNA，以微陣列點樣系統佈放於晶片上即完成。於反轉錄過程中，將 Biotin-dUTP 或 Dig-dUTP 標記於第一股互補去氧核醣核酸(cDNA)上，並與微陣列進行一般之雜交反應(Hybridization)，最後再顯現訊號。 微陣列矩陣的訊號來自核酸標的物所攜帶的螢光顯色基團或顯色基團，利用螢光基團是目前普遍使用方式，靈敏度高並可作定量分析，但其螢光物質材料費高且需要昂貴的雷射掃描儀 (laser scanner)讀取訊號與分析。如圖(19)

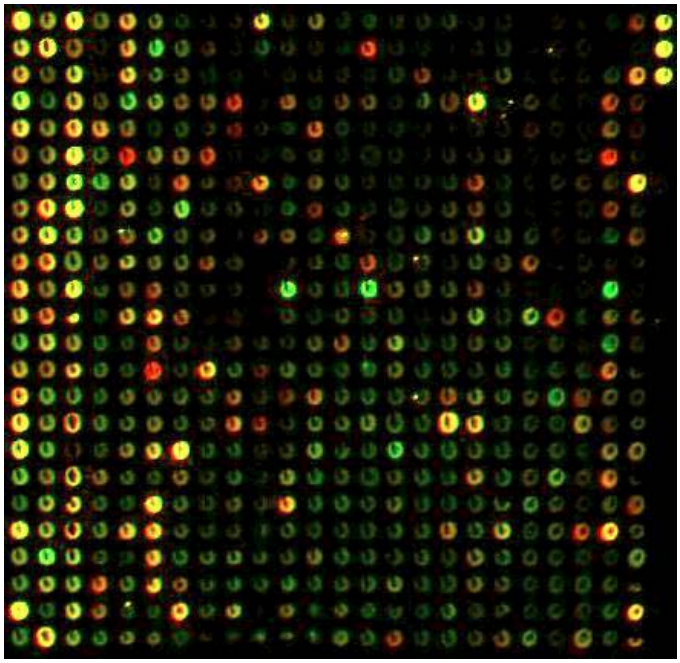


圖 19　微陣列矩陣影像

現階段，由於包含主動元件的自動化生物晶片(Lab-on-a chip)的技術尚未成熟，因此晶片反應完成後的影像處理和分析需經過額外的步驟來進行。由於被記錄的晶片顯像只是類比式(analog)的影像，所以如何將此資料轉換為有用的數據，亦成為生物資訊學(bioinformatics)重要的應用課題之一。簡單來說，基本的影像處理，例如 segmentation、 normalization 、quantification 等步驟是必需的。然而，在資料處理和分析的過程中，以上每一個步驟皆會產生無可避免的誤差。在層層的累積之下, 誤差值可左右實驗分析結果甚巨。此外，在每片晶片高達數萬個引子樣本內，數值高(在 2 以上)的對比反應(high ratio expression)極有可能是高斯分佈(Gaussian distribution)下的巧合，也就是假陽性(false positive)的錯誤訊號。直到目前，這些挑戰尚需有效的方法來具體克服。

附錄二 Program xml Document Type Definition

```
<!ELEMENT prog (arg*, body)>
<!ATTLIST arg name CDATA #REQUIRED>
<!ELEMENT body(%stmt;)+ >
<!ENTITY % stmt "assign |
       call | copy |  par |
       table | while | if ">

<!ELEMENT assign (call | copy) >
<!ATTLIST assign id CDATA #REQUIRED>

<!ENTITY % var "tbl" >
<!ELEMENT tbl EMPTY >
<!ATTLIST tbl id CDATA #REQUIRED>

<!ELEMENT hidb EMPTY >
<!ATTLIST hidb
  name CDATA #REQUIRED
  host CDATA #REQUIRED
  port CDATA #REQUIRED
  path CDATA #REQUIRED >

<!ELEMENT call ((%var)+ ,hidb ) >

<!ELEMENT copy (hidb)>
<!ATTLIST copy tblid CDATA #REQUIRED>

<!ELEMENT par (task+) >
<!ELEMENT task ((%stmt;)+ )>

<!ELEMENT while ((%stmt;)+ , endWhile )>
<!ELEMENT endWhile (if ) >

<!ELEMENT if (check ) >
<!ELEMENT check EMPTY>
<!ATTLIST check
  tbl CDATA #REQUIRED
  row CDATA #REQUIRED
  col CDATA #REQUIRED
```

```
  op CDATA #REQUIRED
  value CDATA #REQUIRED >

<!ELEMENT table (createRow+ |
       call | get ) >

<!ELEMENT createRow (getValueAt)>
<!ELEMENT getValueAt EMPTY>
<!ATTLIST getValueAt
  tbl CDATA #REQUIRED
  row CDATA #REQUIRED
  col CDATA #REQUIRED >

<!ELEMENT get (hidb)>
<!ATTLIST get tblid CDATA #REQUIRED>
```

附錄三 平行 k-means 程式

3.1 分散至兩台機器的平行程式

```
<prog name="parallel_kmeans">
   <arg name="inputTable"/>
   <arg name="parTable1"/>
   <arg name="parTable2"/>
   <body>
   <table name="seed">    <!-- initial seed table -->
     <createRow>
          <getValueAt tblid="inputTable" row="1" col="1" />
     </createRow>
     <createRow>
      <getValueAt tblid="inputTable" row="2" col="2" />
     </createRow>
   </table>
   <par>         <!-- transfer data to remote clients -->
       <task>
            <assign id="idA">
                <copy tblid="parTable1" >
                    <hidb name="poseidon" host="140.113.88.45"
                          port="1119" path="db"/>
                </copy>
            </assign>
      </task>
      <task>
           <assign id="idB">
               <copy tblid="parTable2" >
                   <hidb name="poseidon" host="140.113.88.140"
                         port="1119" path="db"/>
               </copy>
           </assign>
       </task>
   </par>

   <Loop>
      <par>     <!-- partition jobs to remote clients -->
          <task>
               <copy tblid="seed" >
```

```
                <hidb name="poseidon" host="140.113.88.45"
                port="1119" path="db"/>
            </copy>
        </task>
        <task>
            <copy tblid="seed" >
                <hidb name="poseidon" host="140.113.88.140"
                    port="1119"  path="db"/>
            </copy>
        </task>
    </par>
    <par>
        <task>      <!-- each client executes k-means -->
            <assign id="id1">
                <call name="Kmeans" >
                    <tbl id = "idA" />
                    <hidb name="poseidon" host="140.113.88.45"
                        port="1119"  path="db"/>
                </call>
            </assign>
        </task>
        <task>
            <assign id="id2">
                <call name="Kmeans" >
                    <tbl id = "idB"/>
                    <hidb name="poseidon" host="140.113.88.140"
                        port="1119" path="db"/>
                </call>
            </assign>
        </task>
    </par>

    <table name = "result2">  <!-- get client's result -->
        <get tblid= "id2">
        <from>
            <hidb name="poseidon"  host="140.113.88.140"
                port="1119"  path="db"  />
        </from>
        <to>
```

```
            <hidb name="poseidon" host="140.113.88.45"
                  port="1119"  path="db"  />
          </to>
          </get>
        </table>

        <table name="new_seed">   <!—calculate new cluster center -->
        <call name="calMean">
           <tbl id="id1"/>
           <tbl id="result2"/>
           <hidb name="poseidon" host="140.113.88.140"
                port="1119" path="db"/>
        </call>
        </table>
        <table name = "seed">
          <copy tblid="new_seed" >
          <hidb name="poseidon" host="140.113.88.45"
               port="1119" path="db"/>
          </copy>
        </table>
        <endLoop>   <!-- check end condition -->
          <if>
          <table name="result">
          <call name="table_sub">
              <arg name="new_seed"/>
              <arg name="seed"/>
           </call>
          </table>
          <check tbl="result" row="2" col="1"  op="&lt;" value="0"/>
          </if>
        </endLoop>
    </Loop>

    </body>
</prog>
```

3.2 分散至四台機器的平行程式

```
<prog name="parallel_kmeans_for4">
   <arg name="inputTable"/>
   <arg name="parTable1"/>
   <arg name="parTable2"/>
   <arg name="parTable3"/>
   <arg name="parTable4"/>
   <body>
   <table name="seed">
     <createRow>
         <getValueAt tblid="inputTable" row="1" col="1" />
     </createRow>
     <createRow>
         <getValueAt tblid="inputTable" row="2" col="2" />
     </createRow>
   </table>
      <par>
       <task>
            <assign id="idA">
                 <copy tblid="parTable1" >
                 <hidb name="poseidon" host="140.113.88.45"
                       port="1119"  path="db"  />
                </copy>
            </assign>
      </task>
      <task>
           <assign id="idB">
              <copy tblid="parTable2" >
              <hidb name="bioserv" host="140.113.88.140"
                    port="1119"  path="db"  />
              </copy>
           </assign>
       </task>
        <task>
           <assign id="idC">
              <copy tblid="parTable3" >
              <hidb name="ares"  host="140.113.88.36"
                     port="1119"  path="db"  />
              </copy>
```

```
        </assign>
    </task>
     <task>
        <assign id="idD">
            <copy tblid="parTable4" >
            <hidb name="hebe"  host="140.113.88.126"
                 port="1119"  path="db"  />
            </copy>
        </assign>
    </task>
</par>
<Loop>
   <par>
       <task>
            <copy tblid="seed" >
            <hidb name="poseidon"  host="140.113.88.45"
                  port="1119"  path="db"  />
            </copy>
      </task>
      <task>
            <copy tblid="seed" >
            <hidb name="bioserv"  host="140.113.88.140"
                  port="1119"  path="db"  />
            </copy>
      </task>
      <task>
            <copy tblid="seed" >
            <hidb name="ares"  host="140.113.88.36"
                   port="1119"  path="db"  />
            </copy>
      </task>
      <task>
            <copy tblid="seed" >
            <hidb name="hebe"  host="140.113.88.126"
                   port="1119"  path="db"  />
            </copy>
      </task>
  </par>
```

```
<par>
   <task>
          <assign id="id1">
             <call name="Kmeans" >
                <tbl id = "idA" />
                <hidb name="poseidon"  host="140.113.88.45"
                      port="1119"  path="db"  />
             </call>
          </assign>
    </task>
   <task>
          <assign id="id2">
             <call name="Kmeans" >
                <tbl id = "idB"/>
                <hidb name="bioserv"  host="140.113.88.140"
                      port="1119"  path="db"  />
             </call>
          </assign>
   </task>
   <task>
          <assign id="id3">
             <call name="Kmeans" >
                <tbl id = "idC"/>
                <hidb name="ares"  host="140.113.88.36"
                      port="1119"  path="db"  />
             </call>
          </assign>
   </task>
   <task>
          <assign id="id4">
             <call name="Kmeans" >
                <tbl id = "idD"/>
                <hidb name="hebe"  host="140.113.88.126"
                      port="1119"  path="db"  />
             </call>
          </assign>
   </task>
</par>
```

```
<table name = "result2">
<get tblid= "id2">
      <from>
           <hidb name="bioserv"  host="140.113.88.140"
                 port="1119"  path="db"  />
       </from>
       <to>
           <hidb name="poseidon"  host="140.113.88.45"
                 port="1119"  path="db"  />
       </to>
</get>
</table>

<table name = "result3">
<get tblid= "id3">
      <from>
           <hidb name="ares"  host="140.113.88.36"
                 port="1119"  path="db"  />
       </from>
       <to>
           <hidb name="poseidon"  host="140.113.88.45"
                 port="1119"  path="db"  />
       </to>
</get>
</table>

<table name = "result4">
<get tblid= "id4">
      <from>
           <hidb name="hebe"  host="140.113.88.126"
                  port="1119"  path="db"  />
       </from>
       <to>
           <hidb name="poseidon"  host="140.113.88.45"
                 port="1119"  path="db"  />
       </to>
</get>
</table>
```

```
        <table name="new_seed">
        <call name="calMean4">
            <tbl id="id1"/>
            <tbl id="result2"/>
            <tbl id="result3"/>
            <tbl id="result4"/>
            <hidb name="poseidon"  host="140.113.88.45"
                  port="1119"  path="db"  />
        </call>
        </table>
       <table name = "seed">
          <copy tblid="new_seed" >
          <hidb name="poseidon" host="140.113.88.45"
               port="1119" path="db"/>
          </copy>
       </table>
        <endLoop>
         <if>
           <table name="result">
              <call name="table_sub">
                   <arg name="new_seed"/>
                   <arg name="seed"/>
               </call>
           </table>
           <check tbl="result" row="2" col="1"  op="&lt;" value="0"/>
         </if>
       </endLoop>
     </Loop>
   </body>
</prog>
```

參考文獻

[1] *The Globus Project Web Page*. http://www.globus.org.

[2] I. Foster, C. Kesselman, S. Tuecke. The Anatomy of the Grid: Enabling Scalable Virtual Organizations. In *International Journal of Supercomputer Applications*, 15(3):6-7, 2001.

[3] I. Foster, C. Kesselman, J. Nick, S. Tuecke. The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration. In *Open Grid Service Infrastructure WG, Global Grid Forum*, June 2002.

[4] I. Foster, C. Kesselman. Computational Grids. *Chapter 2 of "The Grid: Blueprint for a New Computing Infrastructure"*, Morgan-Kaufman, 1999.

[5] A. Chervenak, I. Foster, C. Kesselman, C. Salisbury, S. Tuecke. The Data Grid: Towards an Architecture for the Distributed Management and Analysis of Large Scientific Datasets. In *Journal of Network and Computer Applications*, 23:187-200, 2001

[6] B. Allcock, J. Bester, J. Bresnahan, A. Chervenak, I. Foster, C. Kesselman, S. Meder, V. Nefedova, D. Quesnel, S. Tuecke. Secure, Efficient Data Transport and Replica Management for High-Performance Data-Intensive Computing. In *IEEE Mass Storage Conference*, 2001.

[7] I. Foster, C. Kesselman, J.M.Nick, et al. Grid Services for Distributed System Integration. In *Computer archive Volume 35 , Issue* 6. p.37-46. June 2002.

[8] B. Allcock, J. Bester, J. Bresnahan, A. L. Chervenak, I. Foster, C. Kesselman, S. Meder, V. Nefedova, D. Quesnal, S. Tuecke. Data Management and Transfer in High Performance Computational Grid Environments. In *Parallel Computing Journal*, Vol. 28 (5), p. 749-771. May 2002.

[9] *Genomes OnLine Database*. http://www.genomesonline.org

[10] *SETI@home: Search for Extraterrestrial Intelligence at home.* http://setiathome.ssl.berkeley.edu/

[11] E. Korpela, D. Werthimer, D. Anderson, J. Cobb, M. Lebofsky. SETI@Home : Massively Distributed Computing for SETI. In *Science and Engineering archive* Volume 3, Issue 1, p.78-83. January 2001

[12] P. Anderson, J. Cobb, E. Korpela, M. Lebofsky, D. Werthimer. SETI@home: An Experiment in Public-Resource Computing. In *Communications of the ACM, Vol. 45 No. 11,* p.56-61. November 2002.

[13] J. Graham, L. Kemp, N. Angelopoulos, and P.Gray. Architecture of a Mediator for a Bioinformatics Database Federation. In *IEEE Transactions on Information Technology in Biomedicine,* Vol. 6, Issue 2, p. 116-122, JUNE 2002.

[14] O. Akinde, H. Böhlen, T. Johnson, L. Lakshmanan, D. Srivastava. Efficient OLAP Query Processing in Distributed Data Warehouses. In *Information Systems* Volume 28 , Issue 1-2 p.111-135. March 2003

[15] J. Han. OLAP Mining: An Integration of OLAP and Data Mining. In *Proc. 1997 IFIP Conference on Data Semantics (DS-7)*, Leysin, Switzerland, pp. 1-11. Oct.1997

[16] A. Schulze and J. Downward. Navigating gene expression using microarrays. In *technology review NATURE CELL BIOLOGY.* Volume 3, p.190-195. Auguest 2001.

[17] C.A. Afshari. Perspective: microarray technology, seeing more than spots. In *Endocrinology* 143: 1983-1989, June 2002.

[18] C. J. Date. An Introduction to Database Systems. /Pearson/Addison Wesley.

[19] myGrid.
A project developing experiments in biology on a Grid.
http://www.mygrid.org.uk/

[20] *Asia Pacific BioGRID*.
A project created a distributed database for a prototypic DataGrid.
http://www.apbionet.org/grid/

[21] *The North Carolina BioGrid project*. http://www.ncbiogrid.org/

[22] *EUROGRID project*. http://www.eurogrid.org/

[23] P. Kacsuk. Parallel Program Development and Execution in the Grid. In *International Conference on Parallel Computing in Electrical Engineering.* p.131 Septemper 2002.

[24] C. Lee, D. Talia. Grid Programming Models: Current Tools, Issues and Directions. Chapter 21 of Grid Computing: Making the Global Infrastructure a Reality. F. Berman, G. Fox and T. Hey (eds.), Wiley, p. 555-578. 2003.

[25] GGF, *Global Grid Forum.* http://www.gridforum.org/

[26] K. Stoffel, A. Belkoniene. Parallel *k/h*-Means Clustering for Large Data Sets. In *Proceedings of the European Conference on Parallel Processing EroPar99*, p. 1451-1454. September 1999.

[27] Web Services Architecture Working Draft, World Wide Web Consortium (W3C), February 2004. http://www.w3.org/TR/ws-arch/