國 立 交 通 大 學

電 機 與 控 制 工 程 學 系

碩 士 論 文

求取網格運算最大可靠度的

資源配置演算法

The Resource Allocation Algorithm for Maximizing the

Reliability of Grid Computing Network

研 究 生：林成梓

指導教授：林心宇　教授

中 華 民 國 九 十 六 年 十一 月

求取網格運算最大可靠度的資源配置演算法

The Resource Allocation Algorithm for Maximizing the Reliability of Grid Computing Network

研 究 生：林成梓　　Student：Cheng-Zih Lin

指導教授：林心宇　　Advisor：Shin-Yeu Lin

國 立 交 通 大 學

電 機 與 控 制 工 程 學 系

碩 士 論 文

A Thesis

Submitted to Department of Electrical and Control

Engineering College of Electrical Engineering

and Computer Science

National Chiao Tung University

in partial Fulfillment of the Requirements

for the Degree of Master

in

Electrical and Control Engineering

November　2007

Hsinchu, Taiwan, Republic of China

中華民國九十六年十一月

求取網格運算最大可靠度的資源配置演算法

研究生：林成梓　　指導教授：林心宇　博士

國立交通大學電機與控制工程學系

摘要

網格運算系統不同於常見的分散式運算系統是因為它著重於大規模的資源分享和為了完成服務的開放式架構。實際上在網格系統可擴張的基礎下，全域網格技術和 Globus Toolkit 是朝向開放式網格服務架構(OGSA)發展，以便於各機構可以提供他們所擁有的服務和整合他們的資源。本篇論文目標是求取最佳資源配置使得網格運算可以得到最大可靠度。我們先介紹網格服務可靠度的模型和估算可靠度的方法，接下來說明網格服務的資源配置和它的最佳化模型，並提出序的最佳化方法(OO)來解決網格中資源配置問題。最後使用兩個例子來比較現存中使用基因演算法(GA)和我們的方法所得到的解和運算時間的差異。

The Resource allocation algorithm for maximizing the reliability of grid computing network

Student: Cheng-Zih Lin Advisor: Dr. Shin-Yeu Lin

Institute of Electrical and Control Engineering

National Chiao-Tung University

Abstract

In this thesis, Grid computing system is different from conventional distributed computing systems by its focus on large-scale resource sharing and open architecture for service. The global grid technology and the Globus Toolkit in particular, are evolving toward an open grid service architecture(OGSA) with a grid system provides an extensible infrastructure so that various organizations can offer their own services and integrate their resources. Hence, this thesis aims at solving the problem of optimally allocating services on the grid to maximize the grid service reliability. We introduce the model of the reliability of grid service and evaluation methods first, and then explain the resource allocation of the grid service and its optimization model, and propose solving the resource allocation problem in order optimization method (OO). Use two example come in comparing in existence using genetic algorithm (GA) and our method, operation of time that method get of us finally.

誌謝

首先我要感謝林心宇老師在於我研究期間給予多方面的指導與協助，不論是在課業上，老師教導我許多重要的基礎知識與觀念。在研究上，傳授我做研究的精神與方法。在爲人處事上，老師的一舉一動，讓我學習到認真與負責的態度。從老師身上所學到的一切，讓我獲益良多，使我能在未來更嚴峻的環境中，繼續往前進步。

感謝啓新學長、謝興學長、士程學長、榮壽學長、建文學長、紹興學長與暐升、崇翔、庭瑜、至祥和 807 實驗室的好伙伴們給予我的寶貴意見和鼓勵。承蒙鄭木火老師與李福進老師的蒞臨指導，使本篇論文更爲精進。

感謝我的女朋友惠文，在我唸書期間，陪在我身旁，給我鼓勵與支持，讓我能有信心和快樂的繼續堅持下去。

最後感謝我最親愛的家人在背後給我的支持，你們是我最重要的驕傲與依靠，使我無後顧之憂的度過這兩年，順利完成學業。

目　　錄

表　目　錄

圖　目　錄

第一章 緒論

1.1 研究動機與目的

網格(Grid)運算系統已經被提出且應用在一個新的領域，有別於傳統的分散式運算系統。是因為網格運算系統注重於大規模的資源分享、創新的應用和在相同的案例中，有別於分散式系統的高執行方針[1-3]，因此網格運算系統是一種廣域分散式系統。在網格原理的基礎下，現實生活中的特定問題如：調整資源分配與動態且大規模的分散式程式問題的處理[4]。

1997 年，美國加州柏克萊大學的天文實驗室發起了一項有趣的科學研究計畫：SETI（Search For Extraterrestrial Intelligence)，想要分析天文望遠鏡所接收的無線電訊號，從中探索是否隱含外星生物的傳遞訊息。由於所收集的無線電訊號量十分龐大，必須同時建置好幾臺超級電腦，才有能力分析所擷取到的微弱外太空訊號，然而柏克萊天文實驗室在當時，並沒有足夠的經費來投入這個計畫。於是他們想到了一個替代方案，號召全世界的個人電腦使用者，參與這個計畫：每個人只要在電腦中下載並安裝一個 SETI@home 的螢幕保護程式，當電腦處於閒置狀態的時候，SETI@home 程式即伴隨著螢幕保護程式執行而啟動，此時從柏克萊大學天文實驗室中的伺服器，即會透過網路傳送一些天文訊號資料至 SETI@home 的程式中，進行運算及分析，完成分析與運算後，再由網路傳送回天文實驗室中的伺服器。

在資料分析的期間，如果個人電腦的使用者重新回到工作崗位上時，SETI@home 程式即會隨著螢幕保護程式的停止而結束，並暫停目前的運算分析動作，一直到下次 SETI@home 再隨著螢幕保護程式被啟動才會再繼續分析。

這樣透過網路達到「分散運算需求到多臺主機」的做法，我們即稱為「網格運算（Grid Computing)」。柏克萊天文實驗室成功地號召了二百萬臺以上的電腦在使用者閒暇之餘，進行天文無線資訊的分析。由於分析的動作是在電腦閒置的時候才進行，因此這樣的作法並不會造成任何使用上的不便。但就柏克萊天文實驗室而言，這二百萬臺的個人電腦整體所提供的總運算能力，可以媲美數臺超級電腦所達成的運算能力。

「找出閒置的 IT 剩餘資源即時回收，並透過網路再利用於其他需要 IT 資源的地方，以增加 IT 運算資源的使用率，減少 IT 成本的支出」，這樣一個 IT 資源循環、再利用的做法就是「網格運算」吸引人的地方。網格運算的理念，起初

是因分享的想法而不是為了交換資訊而提出,但現在相當於直接存取資源包含電腦，軟體和硬體等等。

最近，廣域網格技術(global grid technologies)和於特定的 Globus Toolkit，朝向於開放式網格服務架構(OGSA)發展，使網格系統可以提供一種可以擴展的基礎架構,使各種組織可以提供他們所擁有的服務和整合可靠的特定資源[5]。當一個組織想要在網格上提供一種新的服務，為了使服務可靠度最大化，他們也許會希望可以讓他們的資源分佈最佳化。

一般而言，網格服務需要使用一組資源來可靠地完成任務，資源管理系統(RMS)是網格的中樞，在網格服務的管理上，RMS 扮演著重要的腳色。他是服務請求和服務提供的媒介，控制 RMS 來存取和使用資源[6]。那些服務由一個或數個資源管理站來管理。資源管理站可以經由網域網路來交換資訊，以得到他們所需要的資源。因此，為了提供網格服務，分佈所需資源在網格上是必須的。資源可以是軟體程式，硬體設備等等。為了使提供網路服務最佳化，系統在花費的限制下也許想知道有多少多餘的資源可以提供和如何分佈他們於網格上,可以使網格可靠度最佳化。

1.2 研究方法與論文架構

本篇論文參考[3]裡的網格服務系統的虛擬模型與估測演算法來估算網格服務可靠度，提出在網格服務分析問題的最佳化數學模型和應用基因演算法(GA)來解決問題，然後修改演算法和最佳化數學模型，最後應用序的最佳化(OO)來節省運算時間。

相同的演算法已經被研究於傳統的小規模分散式系統，[7]是最早研究分散式程式/系統可靠度的論文之一。他們建構可靠度數學模型用於分散式系統，但無法直接執行於網格服務可靠度的分析，這是因為假設在運算時，連結與節點的特性是不變的。在此研究之後，延伸此研究的可靠度數學模型，更近一步研究[8-14]。假設在傳統小規模的分散式系統的運算特性是固定的，其交換資訊的大小和處理時間可以忽略，也許是一個好的近似。然而在網格系統中，因為網格的廣域和動態特性，所以經由網際網路，交換資訊量的大小使運算時間與通訊時間不能忽略。因此，原本已存在的數學模型和演算法用於小規模系統的可靠度分析，不能簡單的應用在網格系統和服務來研究可靠度。

因此我們研究分析網格運算系統的可靠度。從[3]中考慮他們的大規模和大範圍特性。由這個結果的延伸，首先制定本篇論文的數學模型，分析，估測和網

格服務可靠度。利用這些規範，然後提出網格服務分佈的最佳化模型和發展 GA 來解決這些問題。

本篇論文其餘的主題如下：第二章節描述網格系統與服務的特性和提出網格服務可靠的模型和估測演算法。第三章節介紹網格服務的資源配置和他的最佳化模型，然後使用 GA 來解決問題。第四章節修改第二章節中的演算法和修改第三章節中的最佳化模型因條件限制而提出的補償運算子。第五章節使用兩個例子來顯現出模擬程序和演算法的效用。第六章節為本篇論文的結論。

第二章 網格服務與穩定度

當公司、顧客、政府和其他機構中在網格中經由連結，有效地利用資源時，網格就開始相當的令人關注。許多專家相信全域網格將是網際網路的後繼者，它是介於軟體和服務的一個獨立層次，在作業系統上將不同的系統連接在一起，允許它們共享資源。透過廣泛共用計算、應用和網路儲存等能力，來共用文件和多媒體檔案，使得每台終端設備都具有超越自身極限的工作能力，協作能力變得更強大。如此一來，需要大量電腦資源的問題就得以解決。現在讓我們進一步探究網格運算的好處：

- 解決複雜問題：

科學發展不斷推動網格技術的演進，這是由於超級電腦也有其極限。為了分析分子碰撞的資料、比對天氣資料和建造一座虛擬天文台，三個主要的網格工程因而成立，分別是美國的 TerraGrid 、英國的 National Grid 、以及一個透過 SURFnet 的荷蘭網格連結。

- 啟動行動裝置：

網際網路擴大了個人電腦的影響範圍，人們利用網路的通訊功能讓伺服器工作，例如存儲備份檔案、個人網頁服務和反垃圾郵件。然而，個人電腦、 PDA 或其他的手持式裝置，卻對共用平台上各種程式尚不具備存取功能，而網格運算具有將這項能力帶到上述設備中的可能。在無線世界中，網格能使一些非常簡單的裝置，例如呼叫器等，透過網路獲得更大的運算能力。

- 緊密協同合作：

如今大多數線上協同作業能讓人們討論同一份文件，而網格運算可以增加更多應用的便利性。另外，透過檢測如財務運用、住宅修繕、事件計劃等，甚至個人也會從已確定的標準方案，或由那些有著相似需求的人所制定之選擇中獲益。

Michael Schrage(麻省理工學院媒體實驗室研究員) 在「嚴肅的遊戲」中說道，要真正使一個雛形開始運用，就必須創造一個雛形本身和其他事物的新關係。許多模擬，包括設計原理都能透由網格運算變成可能，如此可以鼓勵新關係和新團體的產生。

除了新能力之外，還有一個潛在的節約成本效益，例如開發剩餘時間 (電腦的閒置時間) 並行處理其他問題，或是經由比較簡單的裝置，來存取和使用所需要的應用功能，使資訊技術在未來更為經濟實用。如此一來電視會議發展成

遠距離工作將成為常態，線上售貨員將更有效率，整合無線通訊和埋藏在無生命物體的智慧型晶片和甚至將給人們帶來快速且特別的轉變。並且OSGA能夠經由分散，由不同成分組成，動態虛擬組織來整合服務和資源。不管是在一間公司範圍內或擴充到外部的資源分享和服務提供者的關係[1]。為了分析網格的服務可靠度，常態的網格結構和網格服務可靠度的定義將於下面子章節說明。

2.1 名詞與變數的命名

為了建立網格運算服務的可靠度的數學模型,首先定義名詞與變數如下:

RM	資源管理器
RMS	資源管理系統
RN	根節點
RST	資源展延樹
OGSA	開放式網格運算服務架構
GA	基因演算法
MRST	最小資源展延樹
a_{ij}	第 i 個節點與第 j 種資源之間的關係
α	向量 $\{a_{ij} \mid i \in [1,N], j \in [1,K]\}$
B	提供新服務而所需花費的預算上限
c_i	使用第 i 筆資源而所需的費用
element_i	在 MRST 裡的第 i 個要素(節點或連結)
E_j	MRST_j 是可靠的事件
$L(i,j)$	在節點 i 與節點 j 之間的連結
N_t	網格運算服務中,MRST 的總數
$R_c(j)$	第 j 個節點的可靠度方程式
$R_L(i,j)$	連結 $L(i,j)$ 的可靠度方程式
R_{MRST}	MRST 的可靠度
R_s	網格運算服務的可靠度
Q_j	與第 j 個節點相連的一組節點
$T(j)$	第 j 個節點的總通訊時間
$T_w(\text{element}_i)$	第 i 個元素的工作時間
β_i	能整合在節點 i 上的最大資源總數

γ_{ij}	在 GA 裡的染色體上的位元
γ	在 GA 裡的染色體：$=\{\gamma_{ij}=0,1 \mid a_{ij}=d, i\in[1,N], j\in[1,K]\}$
$\lambda(\text{element}_i)$	第 i 個元素的失敗率
σ_{ij}	把資源 R_j 配置在節點 G_i 上
σ	配置矩陣 $\{\sigma_{ij} \mid i\in[1,N], j\in[1,K]\}$

2.2 網格服務的描述

全域網格系統一般如圖1.描述。OSGA[5]能夠從各種機構整合服務和資源。任何使用者進入網格，當他們可以成功連結和在他們之間正確的交換資訊時，便可以利用那些服務和資源。由全域網路連結這些包含資源的裝置如圖 1.所示。這裡資源代表的是一種單位，可以是軟體程式，硬體設備或韌體設施等等。

顧客可以經由網格系統的中樞，也就是RMS的控制來使用遠端資源[15]。當使用者想要利用可靠的服務，或在網格中存取可靠的資源，他們將請求指令優先送到RMS，因為他們不知道哪裡會提供資源或服務。RMS將會識別那些要求後，然後把他們與在網格系統裡提供的服務相匹配。然後這些服務將經由RMS的控制來存取遠端的所需資源。當使用那些遠端資源後，完成服務後，結果將傳回RMS，那些使用者將經由網格服務得到最終結果。通常RMS是由許多RM站構成的，也許他們是分散的[6]。這些許多的RM站可以為使用者所提出的要求提供服務，他們可以互相支持。如果任何一個RM站能成功完成服務，使用者就可以成功地得到服務的結果，因此可以從服務的配合來觀察可靠度。那些RM站也許是由不同成分所構成，是由於不同位置，佈署和其他多樣化的因素來制定，所以在分析上是困難的。

上面那一段是介紹使用者使用網格運算服務的一般程序。換句話說，從提供服務的觀點，也可以稱為虛擬組織[5]，他們的目標是將他們所提供的服務達到高可靠度。服務設計的策略，高可靠度的服務是最重要的策略之一，例如如何在網格系統中配置服務所需的資源，和有多少閒置資源可以使用而不會違反預算限制。

爲了達成這個目標的最佳化，網格可靠度應該被優先研究和估測，這個部份將由下面子章節2.2和2.3來介紹。

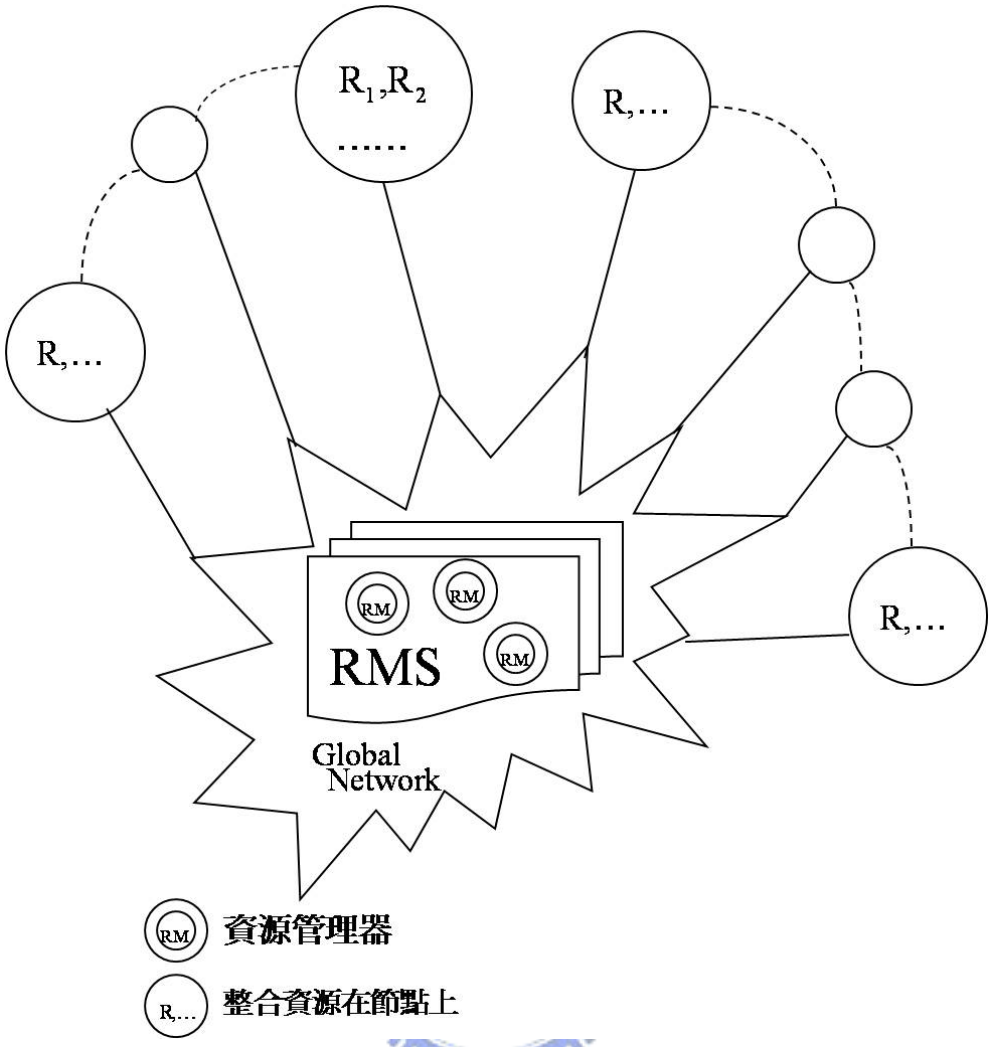


圖1. 網格運算系統

2.3 穩定度模型和網格服務的分析

小規模的分散式程式/系統已經被廣泛的研究[7，9，11，13，14，16]。然而，這些常見的模型有共通的假設：（1）由實體連結(電纜)和節點(訊號處理機)來組成網路拓撲是靜態的，而不用考慮原件的動態改變和邏輯結構。(2) 節點和連結的操作特性是固定的，而不用考慮頻寬和資訊的論點。這些假設不適用於網格服務可靠度模型，因此需要被適當的修正。

在一個網格系統中，在遙遠的兩端點間交流可以在理論上被中斷，雖然他們之間存在實體連結，但是一些遠端資源的使用權利也許會變卦。舉例說明，電腦A是一台伺服器，和電腦B(個人電腦)是相連的。但在現實生活中，有可能因為某些因素，譬如網路線的損毀，或者電腦A改變了電腦B登入的權限而拒絕電腦B進入伺服器。而導致電腦A、B雖然實際上是相連接，但彼此卻不相通。為了解決這

些問題，我們的數學模型使用虛擬結構來取代實體結構，將現實生活中的不確定性的因素給排除掉，以便於探討與研究。在安排之後，在RMS的控制下，網格服務需要一組資源來達成。因此，我們可以將那些處理器/裝置給虛擬化成節點上的資源。RM站可以處理被要求的服務，被虛擬化成根節點(RN)。不同於其他的節點，RN收到從使用者那邊的服務要求，將要求識別/轉換，找出恰當的服務。然後開始調節遠端資源來一起工作，然後將那些結果收集起來傳回RN，最後輸出結果給使用者。因為多數的RN可以並行運作，如果任何一個成功地達成任務，服務就是可靠的。兩節點之間的直接通訊通道可以虛擬成連結，代表不只是實體連結也是邏輯連結(換言之兩節點能夠直接交換彼此的資訊)。虛擬化圖形模型的優點就是簡化網格覆蓋在廣大區域上，可能有許許多多實體處理器和電纜。虛擬圖形模型的範例如圖2.所示。注意： 一個節點允許包含多項資源而且RN也允許提供可靠的資源。

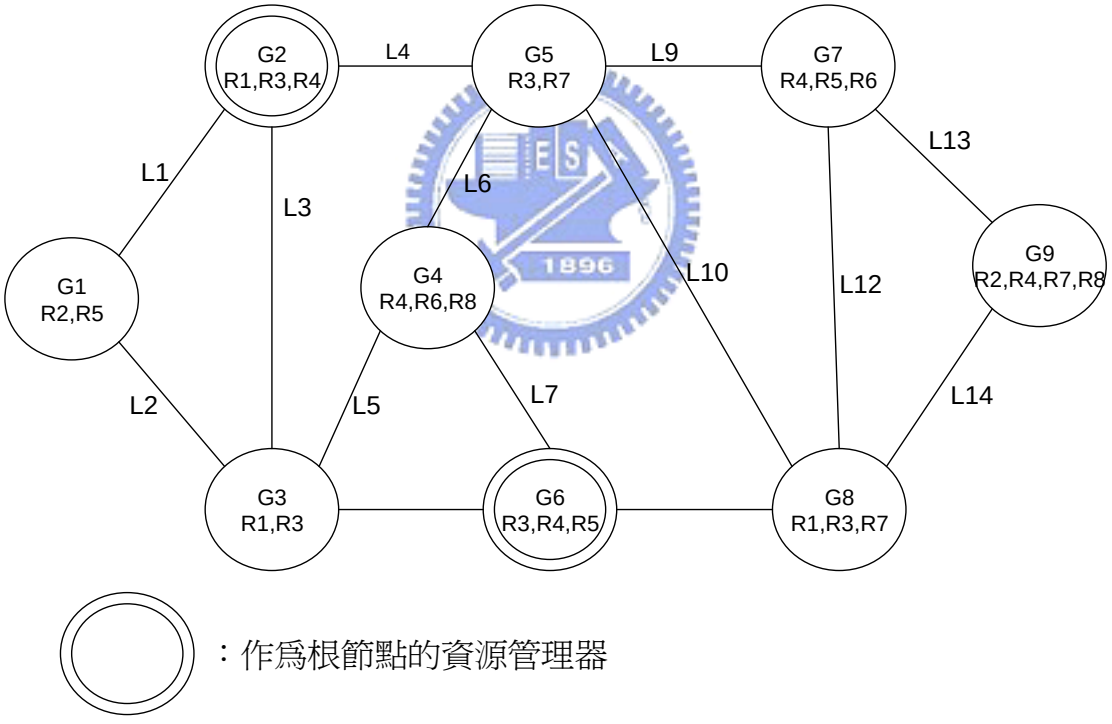


圖2. 使用資源(R1,R2,…,R8)來達成網格服務的虛擬圖形模型

虛擬節點和連結能成功運作的機率，不能單單只是設定個固定值，在過去常見模型中，常見的設定數值是0.9[7，9，11，13，16]。當然成功運作的機率會受到各種條件影響。因此，我們的模型將因各種條件而造成對動態網格運算的影響考慮進來。(1) 在連結上，模型考慮可用的頻寬，從不同位置交換資訊，競爭共同連結的頻寬和連結失敗率的影響。(2) 在節點上，除了以上資訊的傳達，節

點的工作時間應該被包含在裡面。因此，模型結合這些條件會更接近真實的網格環境而且可以用來處理易變的廣大區域通訊。我們的模型是基於假設在這些節點與連結失敗的發生滿足 *Poisson process* [17]。

Poisson process 是隨機程序的一種，是以事件的發生時間來定義的。它滿足以下條件：

1. 在兩個互斥（不重疊）的區間內所發生的事件的數目是互相獨立的隨機變數。

2. 在區間[$t, t+\tau$]內發生的事件的數目的機率分佈為：

$$P[(N(t+\tau)-N(t))=k]=\frac{e^{-\lambda\tau}(\lambda\tau)^k}{k!} \qquad k=0,1,\ldots$$

其中 λ 是一個正數，是固定的參數，通常稱為抵達率（arrival rate）或強度（intensity）。是單位時間(或單位面積)內隨機事件的平均發生率。當 λ 或 τ 越小，發生事件的機率越低。因此在論文中，網格系統為了完成服務，經由連結來交換所需的資源，能成功的運作，代表就是未發生失敗的事件，即是k=0。因此可靠度 $P[(N(t+\tau)-N(t))=0]=e^{-\lambda\tau}$，$\lambda$ 在這裡代表的是失敗率，可靠度會隨著失敗率 λ 和運作時間 τ 的減少而增加。

因此起初失敗的發生會隨著失敗率的參數呈指數分佈。而且，我們的模型也假設不同項的失敗彼此間是獨立的。這是一個在大規模網格系統中真實的論點，是一個好的近似。因為資源被配置在節點上，大部份彼此節點是相距遙遠的。

在細節上，為了特定的服務，RM 站可以在不同資源中交換各種數量的資訊。這些程式和資源也許會分散在不同的網點上，藉由網際網路的相連，由網格上可靠的運算元件和經由通訊通道來交換資訊而完成程式的運算。在處理程式時，失敗的發生也許會在運算處理器上或通訊的通道上。網格運算系統的結構，一般如圖 3. 所示。

在圖 3. 中，假設 M 筆程式 $P_1,P_2,\ldots,P_M$ 散佈在網格運算系統上。每筆程式相對的所需處理時間為 $t(1),t(2),\ldots,t(M)$。這些程式也許會交換資訊以得到所需的資源來協助完成運算。這些資源為 $R_1,R_2,\ldots,R_h,\ldots,R_H$。$N$ 個節點(節點 1 到節點 N)經由網際網路相連。每個節點相當於運算元件，也許會運算一組程式或包含一組資源。"$RES-n$"代表一組資源在節點 n 上。"$PRG-n$"代表一組程式在節點 n 上運作 $(n=1,2,\ldots,N)$。

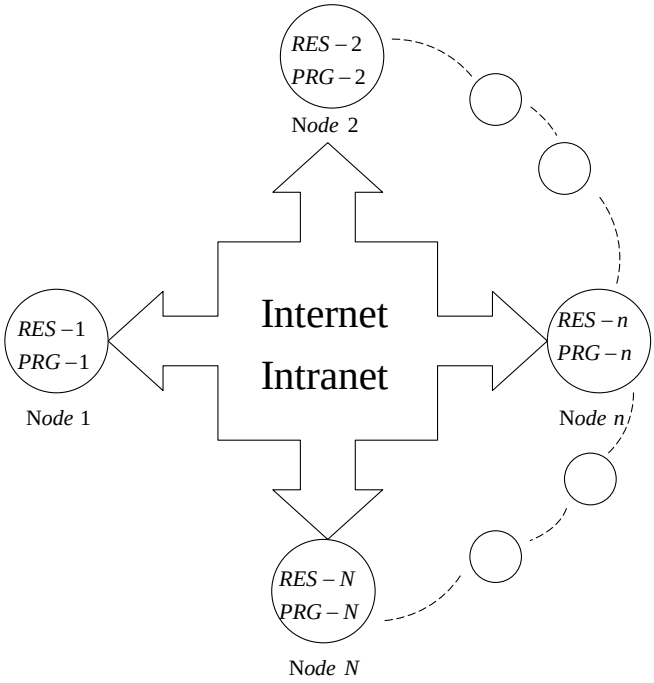


圖 3. 網格運算系統的結構

在節點 *i* 與節點 *j* 之間的連接有兩種方式。第一種是直接連接，意思是兩個節點能經由網際網路直接連接。第二種是非直接連接，如果兩節點間不存在直接相連的的通道或者通道是不通的，須藉由其他的節點來達成連接。舉例來說，一個可靠的資源只配置在特定的節點上或只能被特定的使用者來存取，所以如果其他的節點想要連接此資源或其他的使用者想要使用此資源，他們只能經由特定的節點或使用者來收取/送出資訊來完成與此資源相連。

專有名詞"*link*"被使用在這裡代表的是和節點直接相連。因此，直接相連能由圖 4. 來表示。

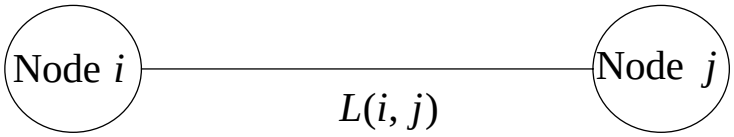


圖 4. 兩節點間的直接連接

link L(*i*, *j*) 也許是像電纜那樣實體的連接，或者是像網際網路那樣虛擬的連接(非實體連結)。然而，不論是哪種連結，在兩節點之間，他們應該存在經由連結交換資訊的平均速度，稱為 *S*(*i*, *j*)。

通常在相同連結上，傳送或收取大量的資訊會比小量的資訊有著較大的失敗機會。如果失敗的發生滿足 *Poisson process* ，那麼交換越大量的資訊則需要

更多的通訊時間，則會使任務失敗的機率更大。不同的程式可能會在相同資源上交換不同量的資訊。更詳細的說，資源管理站為了完成特定的服務而可以從不同節點收集各種所需的資源，其中 D_h 是在一個資源管理站與資源 R_h $(h=1,2,\cdots,H)$ 爲了執行服務而交換的總資訊量。

依照圖 4.，在節點 i 與節點 j 之間的通訊時間，由符號 $T_c(i,j)$ 來表示。經由連結 $L(i,j)$ 交換總資訊量，由符號 $D(i,j)$ 來表示。連結的平均速度，由符號 $S(i,j)$ 來表示。$T_c(i,j)$ 可以由 $D(i,j)$ 與 $S(i,j)$ 推導得到，如下式

$$T_c(i,j)=D(i,j)/S(i,j) \tag{1}$$

可以假設失敗發生在節點上，節點 n $(n=1,2,\ldots,N)$，連結 $L(i,j)$ $(i,j=1,2,\ldots,N)$，各自均滿足在運作階段的 Poisson processes[12-13]。節點 n 的失敗率，由符號 λ_n 來表示，而連結 $L(i,j)$ 的失敗率，由符號 $\lambda_{i,j}$ 來表示。在連結時，不論是連結或是連結的兩節點，當中任何一處發生失敗，就代表此連結是失敗的。因此在節點 i 和節點 j 之間，經由連結 $L(i,j)$ 成功連結的機率可以表示如下式

$$R_c(i,j)=\exp\{-(\lambda_i+\lambda_j+\lambda_{i,j})\cdot T_c(i,j)\} \tag{2}$$

其中 $T_c(i,j)$ 可以經由(1)式運算求得。

同樣地，程式執行時，在執行程式的節點上發生任何失敗將讓所提供的服務失敗。節點 n 上的程式 P_m 在時間 $t(m)$ 內能運算成功的機率

$$R_p(m,n)=\exp\{-\lambda_n\cdot t(m)\} \tag{3}$$

在圖 2. *link* $L(i,j)$ 用來交換資訊的可靠度可以由下式表達

$$R_L(i,j)=\exp\{-\lambda_{i,j}\cdot T_c(i,j)\} \tag{4}$$

$T_c(i,j)$ 可以經由(1)式求得。因此，節點 G_j 的總通訊時間可以由下式表達

$$T(j)=\sum_{i\in D_j} T_c(i,j) \tag{5}$$

D_j 在這裡指的是在 MRST 裡與節點 G_j 通訊的節點。因此節點 G_j 在通訊上的可靠度方程式可以由下式表達

$$R_c(j)=\exp\{-\lambda_j\cdot T(j)\} \tag{6}$$

最終，節點 n 上，程式 P_m 在處理時間 $t(m)$ 內能成功執行的可靠度，可以經由(3)式得到。

由於每個RM站特性的不同，不同RM站(RN)在控制相同的服務時，甚至可能有不同的處理時間和失敗率。在節點上的處理時間能被加在通訊時間上，然後上述(6)式可以被直接用於計算節點的可靠度。上述情況是爲了考慮不同節點的差異性和RM站在網格中因不同的失敗率，處理時間，通訊時間和其它變數的影響。這裡我

們相信，會比單一假設(0.9)更接近實際情況[18]。

基於以上所描述的網格運算系統。定義網格服務可靠度如下：

網格服務可靠度：資源管理系統收到使用者的所要求的服務，蒐集遠端資源來協助完成，能成功執行程式在網格環境中的機率。換句話說，RN在網格運算環境中管理使用者所要求的服務，能成功地執行而不會失敗；任何一組資源(用來完成服務)在工作時是可靠的；連結在通訊時也是可靠的。

2.4 網格運算服務可靠度的估算

爲了估測所供給的服務的可靠度，圖形定理(graph theory)被應用在這裡。一組虛擬節點和連結組成的展延樹，RN收集到所需資源來執行被要求的服務。有些展延樹可以支配其他的展延樹。網格運算服務可靠度完全由這些展延樹的可靠度來決定。因此，展延樹從起始點RN經由連結到其他節點搜尋為了完成服務而所需的資源(換言之，RN站管理網格運算服務)定義為資源展延樹(RST)。因此在展延樹上的節點持有爲了完成網格運算服務而所需的資源。在所有RST當中，其中他們有些RST也許可以支配其他RST。換句話說，早前的RST失敗，其後的RST也必定失敗。因此，在網格可靠度分析上，觀察到支配RST的集合是最重要的。如此支配RST的展延樹稱為最小資源展延樹(MRST)，定義如下。

MRST：對於在RN上執行給定的服務而言，MRST_i 是 RST_i 的最小子樹，稱為 RST_j。即是 $\mathrm{RST}_j \subset \mathrm{RST}_i$。

從圖2.的例子來說。假設在RM上或根節點 N_2 需要 R_1, R_2, R_3 和 R_4 這四種資源來達成服務。下列有四條 *RST* 來完成此服務，分別是 $\{G2, L1, G1\}$；$\{G2, L1, G1, L2, G3\}$；$\{G2, L3, G3, L2, G1\}$；$\{G2, L4, G5, L9, G7, L13, G9\}$；這四棵RST裡有三棵是MRST。分別是 $\{G2, L1, G1\}$；$\{G2, L3, G3, L2, G1\}$；$\{G2, L4, G5, L9, G7, L13, G9\}$，其中 $\{G2, L1, G1, L2, G3\}$ 不是MRST，是因為 $\{G2, L1, G1\}$ 是 $\{G2, L1, G1, L2, G3\}$ 的子樹[3]。

MRST 的可靠度是由操作 MRST 來完成給定服務的機率來決定的，稱為 R_{MRST}，有三個部分: (1)在通訊中，被包含在 MRST 中，所有連結的可靠度。(2)在通訊中，被包含在 MRST 中，所有節點的可靠度。(3)在程式處理時間之內，所有根節點在運算程式時的可靠度。

因此，MRST 的可靠度可由方程式(3，4，6)推導

$$R_{MRST} = R_p(m,n)\cdot \prod_{L(i.j)\in MRST} R_L(i,j)\cdot \prod_{G_j\in MRST} R_c(j)$$
$$= e^{-\lambda_n\cdot(t(m)+T(n))}\cdot \prod_{L(i.j)\in MRST} e^{-\lambda_{i,j}\cdot T_c(i,j)}\cdot \prod_{\substack{G_j\in MRST \\ j\neq n}} e^{-\lambda_j T(j)} \quad (7)$$

從方程式(7)，我們定義一參數為MRST裡的連結與節點的”工作時間”。連結的工作時間是在連結時的通訊時間，可以由(1)式求得。根節點的工作時間是在根節點的總通訊時間和程式處理時間的總合，換句話說就是(7)式的 $t(m)+T(n)$ 。其他節點的工作時間是相對節點的通訊時間。$T(j)$ $(G_j \in MRST)$ 的總通訊時間可以由(5)式求得和給定的程式 P_m 決定節點的處理時間，$t(m)$。MRST 的節點與連結在這裡被定義為” *element* ”。假設在一棵MRST裡共有K個 *element*。因此 $element_i$，(i=1,2,…,K) 定義為MRST裡的第 *i* 個 *element*。於是第 *i* 個 *element* 的工作時間和失敗率分別被定義為 $T_w(element_i)$ 和 $\lambda(element_i)$。結合(2)式，(4)式和(7)式的MRST的可靠度可以簡單的表示如下

$$R_{MRST} = \prod_{i=1}^{K} \exp\{\lambda(element_i)\cdot T_w(element_i)\} \quad (8)$$

在這方程式，如果所有 *element* 的工作時間和失敗率已經得到，則可以算出MRST的可靠度。因此找出所有MRST和決定他們的 *element* 工作時間是取得網格運算程式/系統可靠度的第一個步驟。在RN上搜尋為了完成服務的所有MRST的演算法，演算法詳細的過程在**附錄A**。重複的使用此演算法，可以找出所有各自從RN出發的MRST，找出使服務完成的資源，搜尋MRST的演算法，簡略的定義如下：

Step 1. 使用者對網格運算系統請求服務，稱為 S_1，開始於管理 S_1 的RN沿著可能的連結，去搜尋在其他節點上所需的資源和記錄每個 *element* 組成的搜尋路徑和 *element* 的工作時間。

Step 2. 當所有的所需資源已找到，MRST即找到。

Step 3. 使用深度優先搜尋(DFS)演算法來重複 *Step 1* 和 *Step 2*，直到搜尋到所有的MRST。

Step 4. 換到另一個同步管理 S_1 的RN，重複上面所述的三個步驟直到所有RN完成服務。將所有完成 S_1 的MRST存入一向量裡面，為 $MRST(S_1)$。

在找出所有 MRST 後，任何一條在 $MRST(S_1)$ 裡的 MRST 在工作時，如果他的每個 *element* 都能正常運作，能提供來成功地運算給定的程式。因此網格運算服務的可靠度可以寫成數學式，如下所示

$$\mathrm{R}_s = \Pr(\bigcup_{j=1}^{N_t(P_m)} E_j) \quad (9)$$

令 N_t 為能在 $MRST(S_1)$ 裡的 MRST 總數。E_j 代表 $MRST_j$ 能成功執行所要求的服務，$(j=1,2,\ldots,N_t(P_m))$。經由條件機率，在(9)式所考慮的事件可以分解成彼此互斥事件，如下所示

$$R_s = \Pr(E_1) + \Pr(E_2)\cdot\Pr(\bar{E}_1 \mid E_2) + \ldots + \Pr(E_{N_t(P_m)})\cdot\Pr(\bar{E}_1 \Lambda \bar{E}_2 \Lambda \ldots \Lambda \bar{E}_{N_t(P_m)-1} \mid E_{N_t(P_m)}) \quad (10)$$

在這裡， $\Pr(\bar{E}_1 \mid E_2)$ 指的是 $MRST_2$ 是在運作狀態時而 $MRST_1$ 已經失敗的條件機率。

因此(10)式的網格服務可靠度可以用來估算兩件不同事件的機率。第一事件是指的是 $MRST_i$，而第二件事件指的是 $MRST_j$ $(j=1,2,\ldots,i-1)$。當 $MRST_i$ 在運作狀態時，所有在它之前的樹，也就是 $MRST_j$ 是全處於失敗狀態的。

第一事件的機率，$\Pr(E_i)$，可以經由(8)式明確的計算得到。第二事件的機率，$\Pr(\bar{E}_1 \Lambda \bar{E}_2 \Lambda \ldots \Lambda \bar{E}_{i-1} \mid E_i)$，可以經由我們的演算法2得到，演化法詳細的過程在**附錄B**。簡略地描述演算法2的兩個步驟：

Step 1. 識別所有使 $MRST_j$ $(j=1,2,\cdots,i-1)$ 失敗，而 $MRST_i$ 能成功運算的條件元件。定義條件元件，稱為 $element_k$（任何 $MRST_j$ 被包含在裡面，$j=1,2,\cdots,i-1$），條件元件運作在開始到結束的這段時間內，當 *element* 失敗，可以導致 $MRST_j$ 失敗而不會影響到 $MRST_i$。為了使 $MRST_i$ 正常運作，$MRST_j$ 裡的 $element_k$ 運作的工作時間必須大於在 $MRST_i$ 裡相同 $element_k$ 的工作時間(如果 $MRST_i$ 未包含 $element_k$，將他的工作時間視為0)，然後使用二元選擇樹(binary search tree)。

Step 2. 搜尋這些確定的elements可能的組合，可以使事件 $\bar{E}_1 \Lambda \bar{E}_2 \Lambda \cdots \Lambda \bar{E}_{i-1} \mid E_i$ 發生和計算這些機率的組合。這些機率總合就是 $P_r(\bar{E}_1 \Lambda \bar{E}_2 \Lambda \cdots \Lambda \bar{E}_{i-1} \mid E_i)$ 的結果。

因此，由(5)式計算 $P_r(E_i)$ 和由演算法2(附錄B)來計算 $P_r(\bar{E}_1 \Lambda \bar{E}_2 \Lambda \cdots \Lambda \bar{E}_{i-1} \mid E_i)$，$i=1,2,\cdots,N_t$。代入(7)式，就可以得到網格服務的可靠度。

第三章 網格運算服務分布的最佳化模型

用來估算網格服務可靠度的模型和演算法，現在能夠被用來研究和解決：一機構整合可靠的服務在網格裡的最佳化問題，為了使網格服務可靠度在預算範圍內最大化。

3.1 網格服務的資源配置

OGSA 能夠使各種機構提供他們的服務和在網格上配置資源[1]。分析上面章節所述，RMS 使用一些遠端的資源來完成使用者所需求的服務。在網格上，從服務提供者提供新的服務，可以看出完成網格服務而所需的資源配置在節點上。例如，整合硬體設備，安裝軟體程式和分享資料庫等等。accessible nodes 代表這些節點可以被用來整合在新的網格服務上一個或數個資源。

在廣域網格中的資源管理系統(RMS)，能夠整合處理器、資料庫、應用伺服器與應用軟體等網格元件。讓它們彼此互相合作來完成使用者所要求的服務。這些元件就是我們所謂的資源。資源的供應就是將資源配置到有需要的地方(節點)上，為了完成使用者所要求的服務而將資源動態地配置。

舉個例子來幫助說明:某島國位於大陸板塊交疊的地帶，導致島上經常發生大小規模的地震。因為地震所帶來的傷害與損失是非常巨大的，也因此預測地震的發生，做好防震措施，將傷害降至最低，紀錄地質活動是非常重要的。

在國家地震工程研究中心，研究人員欲分析今年與過去幾年的地質活動，來預測近期是否會有發生大地震的可能。因此研究中心必需記錄今年全國各個主要斷層帶的地質活動，如圖 5.所示。研究中心有數顆硬碟(R1，R2，R3 動態配置)來紀錄各大斷層帶的地質活動的情形，這樣的配置是為了避免硬碟毀損或者因為其他因素造成記錄停擺，而導致紀錄失真，將收集完的資料存放到資料庫(R4 固定配置)，這就是所謂的硬體資源配置。

接下來，研究人員要分析這幾年的地質活動，這是數筆龐大的資料(R5，R6 動態配置)，所以必須藉由許多台電腦來幫助分析與運算，如圖 6.所示，經由網際網路，找尋到願意提供閒置資源的處理器，將資料(R5，R6)配置上去來完成分析。最後將分析結果送回研究中心的資料庫(R4)，同樣地，這樣的資源配置是為了確保分析計算成功，這就是所謂的軟體資源配置。

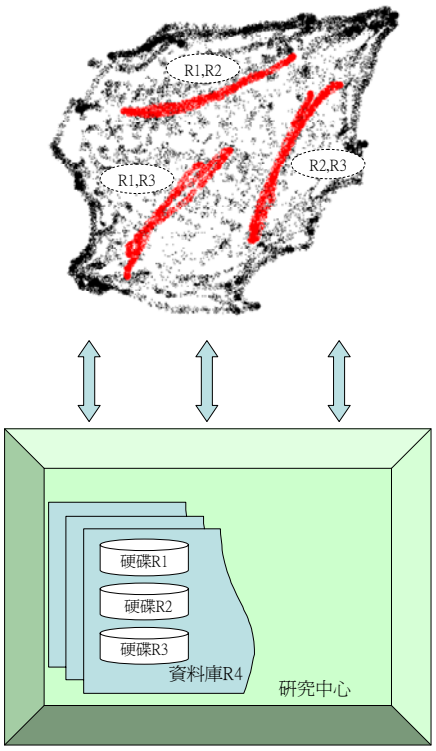


圖 5. 記錄地質活動

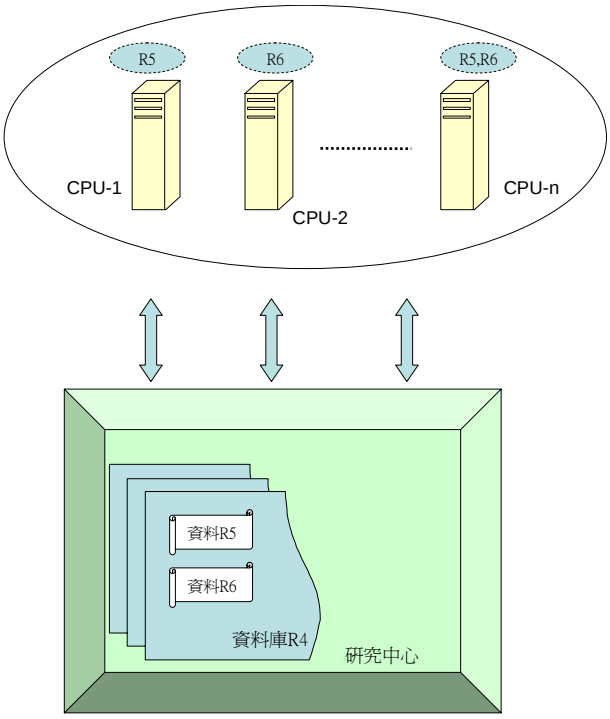


圖 6. 分析資料

看完上面的例子，接下來我們所探討的是資源管理系統該如何動態配置資源，在有限的預算下，讓網格服務可靠度最大化。因此在機構中，誰要提供新的資源是常見的問題。意思是爲了使網格運算服務可靠度最大化，如何在 accessible nodes 上使資源的配置最佳化。實際上，服務提供者也許想知道他們該使用多少閒置的資源和如何配置資源在各個 accessible nodes。

假設一機構能在網格上提供服務，此網格服務被設計配置k筆資源，以 R_i $(i=1,2,\cdots,k)$ 來表示。這些資源可以是軟體程式，硬體元件或韌體設備。機構能使用每種型態的數筆閒置資源，但是使用閒置的資源會導致額外的花費，總支出的預算上限為B。此外，在網格上每種型態的閒置資源應該至少要有一筆。假設機構能在網格裡的 N 個accessible nodes $(G_1,G_2,\cdots,G_N)$ 上配置資源和同時有 $G_1,G_2,\cdots,G_M$ ，也就是 RN_s （i.e. RM站）來管理服務。每一節點上所分配到的資源數量應有所限制，所以我們使用 β_i $(i=1,2,\cdots,N)$ 來代表每一節點上所配置的資源最大上限數。同樣地，有些資源也許已經固定配置在某些節點上，有些資源也許可以被允許選擇某些節點來配置或不允許配置在某些節點上。舉個例子來幫助說明，下面圖7. 是網格運算技術在生態研究上的應用。

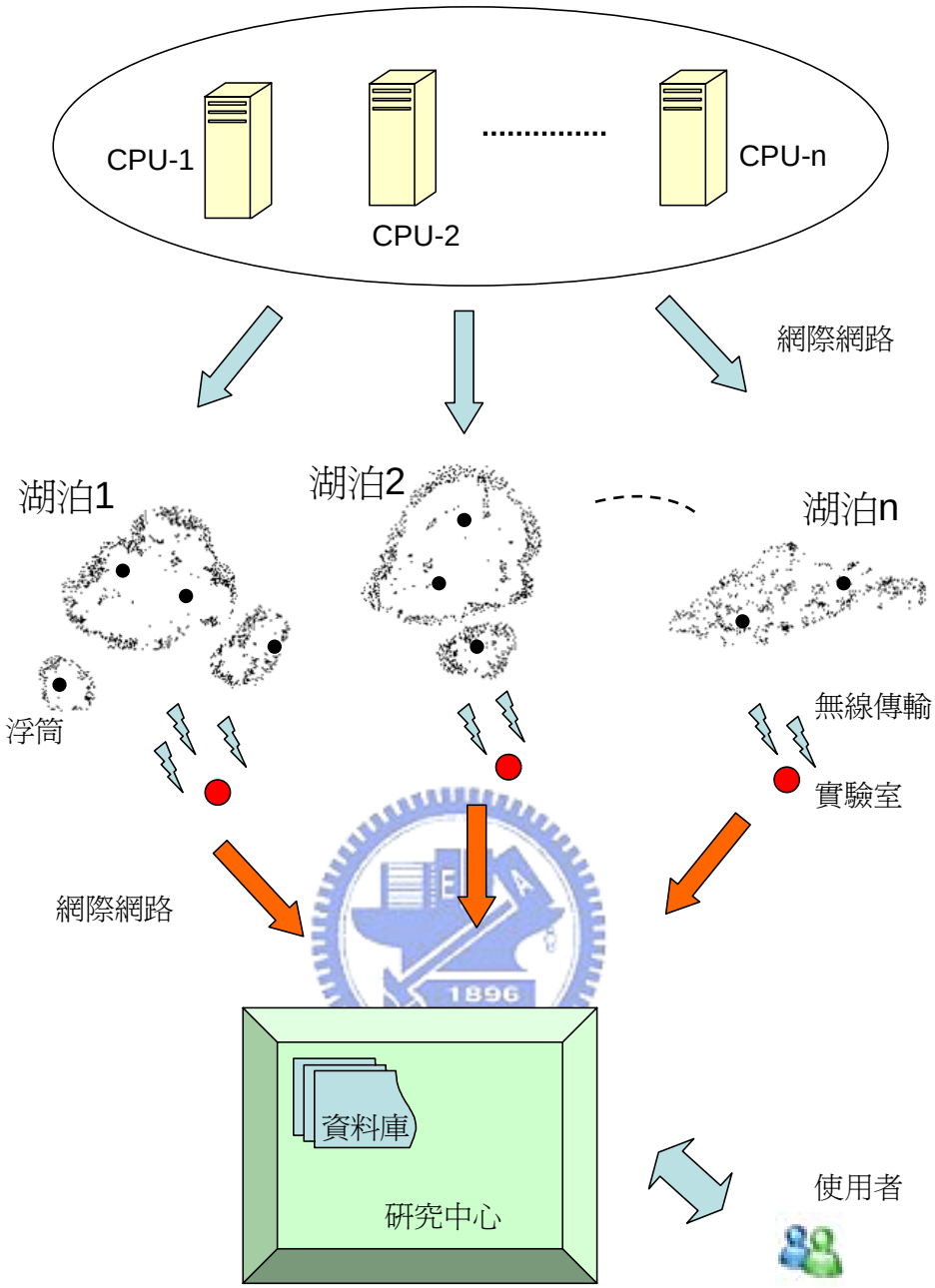


圖7. 網格運算應用於追蹤湖泊代謝過程

各國科學家為追蹤湖泊代謝過程，在湖泊架設浮筒（buoy），浮筒上再架設氣溫、相對濕度、風速、風向以及溶氧量儀器，資料可設定每小時/分鐘蒐集一次。

而這些現地蒐集到的資料，可利用無線傳輸方式即時傳送到實驗室，研究人員透過電腦的監控而得知資料是否能正確無誤的傳送進來，若訊號中斷或資料異常發生時，便可在發現問題的第一時間內儘快予以解決，以減少資料損失的風險；許多複雜的生態系動態模式，可以連結數以萬計的個人電腦共同運算。

資料最後再傳送到研究中心的資料庫。如此各地的研究人員便可在網際網路

上擷取資料以及進行分析。這個計畫將網格技術中強調的**分散**、**遠端**、**分享**等概念，在生態研究上實現。

假設我們要比較湖泊 1(R1)與湖泊 2(R2)的代謝過程，將資料收集到研究中心的資料庫(R3)，連結數以萬計的個人電腦(R4)來共同運算複雜的生態系動態模式，將結果輸出給使用者分析與研究。因此在湖泊 1 的實驗室(虛擬成節點)裡就不會有湖泊 2(R2)的資料，研究中心(虛擬成節點)也有運算與分析的能力(R4)。因此，我們使用 a_{ij} 來描述節點 G_i 與資源 R_j 之間的關係。其中 $i=1,2,\cdots,N$；$j=1,2,\cdots,K$。a_{ij} 有三個值(−1，0 或 1)。如果 $a_{ij}=-1$，代表機構能自由地選擇是否分配資源 R_j 於節點 G_i 上。如果 $a_{ij}=0$，代表資源 R_j 不允許配置於節點 G_i 上。 如果 $a_{ij}=1$，代表資源 R_j 已經固定配置於節點 G_i 上。然後 α 是決定資源 R_j 與節點 G_i 之間關係的代表矩陣，$\{a_{ij}\,|\,i\in[1,N],\ j\in[1,K]\}$。

在這些條件下，機構使用每種閒置資源和配置他們在網格上的節點裡。下一子章節會描述為了使新服務在網格上能使可靠度最大化的最佳化模型。

3.2 最佳化模型

σ_{ij} 代表資源(R_j)與節點(G_i)之間的關係。$\sigma_{ij}=0$，意思是 R_j 未配置在 G_i 上。 $\sigma_{ij}=1$，意思是 R_j 已配置在 G_i 上。σ 被定義為一矩陣 $\{\sigma_{ij}\,|\,i\in[1,N],\ j\in[1,K]\}$，代表是爲了完成網格服務而配置所需的資源在節點上的策略。因此，涉及服務的節點和連結的架構，可以由矩陣 σ 的配置策略來決定網格服務的可靠度。網格服務的可靠度可由 $R_s(\sigma)$ 來表示，可以由(13)式算出。因此最佳化的問題變成找尋 σ 的最佳解，使網格服務可靠度的最大化。最佳化模型如下所示：

$$Given \quad a_{ij}$$

$$Decision\,\text{var}\,iables: \sigma=\{\sigma_{ij}=0,1\,|\,i\in[1,N],\ j\in[1,K]\} \tag{11}$$

$$Objective\ \ function\ :\ Maximize\ \ R_s(\sigma) \tag{12}$$

$$Subject\ \ to:\ a_{ij}\neq -1,\ \ then,\ \ \sigma_{ij}=a_{ij}$$

$$i=1,2,\ldots N \quad and \quad j=1,2,\ldots,k \tag{13}$$

$$\sum_{i=1}^{N}\sigma_{ij}\geq 1\ ,\quad j=1,2,\cdots,K \tag{14}$$

$$\sum_{j=1}^{K}\sigma_{ij}\leq \beta_i \quad i=1,2,\cdots,N \tag{15}$$

$$\sum_{i=1}^{N}\sum_{j=1}^{K}c_j\sigma_{ij}\leq B \tag{16}$$

一開始先依代表矩陣a，確定某些資源R_j已經固定配置或不允許配置於節點G_i上。使得Constraint(16)被a所限制，代表節點與資源間的關係：如果$a_{ij}=1$(換句話說，資源R_j已固定配置於節點G_i上)， σ_{ij}的值被設定為1。 如果$a_{ij}=0$(換句話說，資源R_j未配置於節點G_i上)， σ_{ij}的值被設定為0。在限制條件(14)當中，$\sum_{i=1}^{N}\sigma_{ij}$代表資源$R_j$被配置在網格上的總數，應該至少為1。在限制條件(15)當中，$\sum_{j=1}^{K}\sigma_{ij}$代表配置在節點$G_i$上的資源總數，應當不能超過上限值$\beta_i$。在限制條件(16)當中，$c_j$代表使用第$j$種資源所需的費用。所以$\sum_{i=1}^{N}\sum_{j=1}^{K}c_j\sigma_{ij}\le B$代表使用所需資源來完成網格運算服務的花費，且花費不能超過上限B。

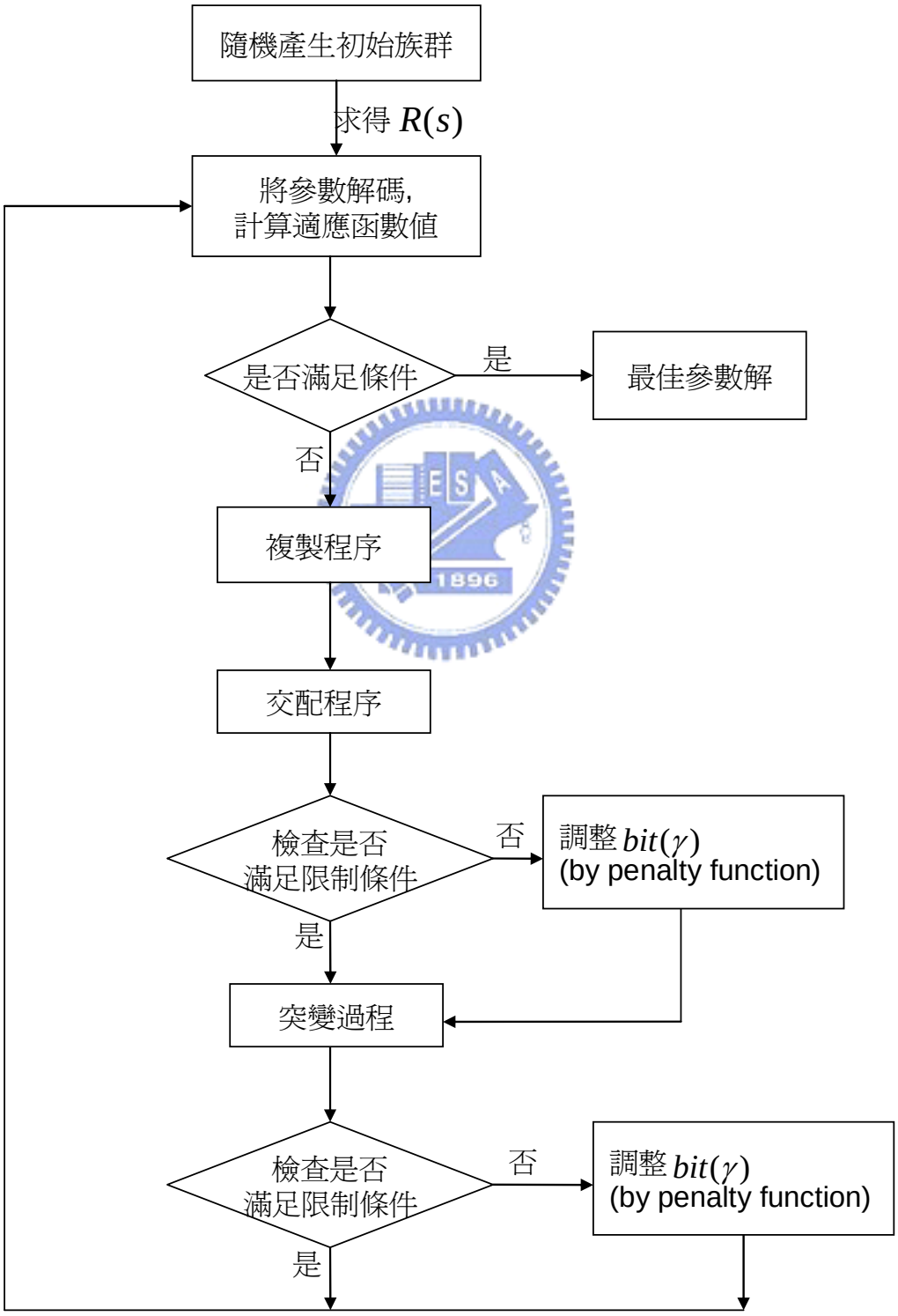
因為$\sigma=\{\sigma_{ij}=0,1|i\in[1,N],j\in[1,K]\}$，是一段長度為$N\times K$的二進制碼。因此，整個解空間為$2^{N\times K}$。解空間會隨著資源和節點總數成指數型地增加。同時地，計算每件實例(服務的配置)的條件機率也會有時間消耗。快速找尋這配置的所有MRSTs 和所有事件$\bar{E}_1\Lambda\bar{E}_2\Lambda\cdots\Lambda\bar{E}_{i-1}|E_i$可能發生的組合，用(13)式來估算條件機率。MRSTs的總數由節點總數和節點之間的連結來決定。由MRSTs的總數來決定(13)式運算的複雜度。為了解決這個最佳化的問題，在常見的網格運算服務中，徹底地搜尋演算法是沒有效率的，舉例來說，如果一個網格運算服務包含10種資源和 10 個節點，解空間就為2^{100}，無法在任何可以接受的時間內做徹底的搜尋。

因此，用適當的探索演算法做為手段來解決如此複雜的問題。下一子章節是參考[19]，使用GA來解決這個最佳化的問題。但是這方法會導致花費過多的時間在計算每一個資源配置所對應的可靠度，主要是因為它必須完整找出所有可以完成服務的MRST。雖然越多條的MRST可以更精確地計算可靠度，但是也會導致大量的運算。為了改善此問題，因此我們提出了序的最佳化(OO)方法。我們的方法分成兩個階段，在第一階段中，我們已較粗略的模式來計算可靠度，亦即以前三條MRST來計算可靠度，然後用GA來決定不錯的50組資源配置。第二階段我們針對所選出的50組資源配置，精確計算每一資源配置的可靠度--即完整的找出所有MRST來計算可靠度。於是在第二階段中所求得的50組中最佳資源配置及為我們所得的解。

3.3 G.A.

GA 是一種推測的最佳化技術，在人工智慧的領域迅速的發展。GA 是由達文西的物競天擇定理得到啟發，被廣泛地使用在許多領域上[20-24]。

3.3.1 流程圖



3.3.2 配置(編碼)、fitness evaluation

網格運算服務的資源配置最終解就是配置向量σ。根據限制條件(13)，有些在σ上的位元已經由$a_{ij} \neq -1$來決定。將這些已經決定配置的位元作上記號，其它的位元爲二進位的變數，由向量$\gamma = \{\gamma_{ij} \in \{0,1\} \mid a_{ij} = -1, i \in [1,N], j \in [1,K]\}$來代表。因此相對應於$\gamma$，染色體就是一列二進制碼(0 或 1)。一列當中的每個成員，γ_{ij}的配置相當於基因。如果$\gamma_{ij} = 1$，他的意思是資源R_j被設計分配在節點G_i上。如果$\gamma_{ij} = 0$，他的意思是資源R_j被設計不分配在節點G_i上。因此σ的最終分佈解可以被γ和那些$a_{ij} \neq -1$(代表那些位元值已經被決定)的值來決定。有些基因被編碼成γ，限制條件(13)將滿足下列 GA 的步驟。染色體的適應函數決定了解的品質。在網格運算服務配置問題當中，解的品質由網格運算服務可靠度(在目標方程式(12))來表示。爲了計算網格運算服務可靠度$R_s(\sigma)$，配置向量σ應該被優先決定，由γ和$a_{ij} \neq -1$來組合。因此，網格運算服務可靠度可以由(7)計算出。使用在我們 GA 裡的適應函數定義如下

$$Fitness = \frac{-A}{\ln(R_S(\sigma))} \qquad (17)$$

A 為常數正數和$R_S(\sigma)$為網格運算服務可靠度。因為$R_S(\sigma)$的值介於 0 和 1 之間，$\ln(R_S(\sigma))$將趨近於 0，則$R_S(\sigma)$趨近於 1，適應值將趨近於無限。因此，在(17)式能得到一個很大的適應值，則$R_S(\sigma)$接近於 1。

在所有染色體所得的適應值之下，我們使用輪盤式選擇法在這問題上來決定基因池。

3.3.3 補償運算子(違背限制)

有些方法可以用來處理限制條件，舉例來說，處罰方程式(penalty function)，附加修正項，拋棄無法實現的解等等。在此案例中，尋找合理的處罰方程式是困難的。因為我們有數個限制條件，且每個都沒有共同相似之處，以便於來簡單的使用可靠的權重。如果他們使用處罰方程式，每一項加入處罰項。那會更加困難地發現有理且有效的方法來把它們結合在一起。拋棄不適應的解的方法是用在於不適應的解是相當少的，而在這裡是不適用的。當不適應的解出現相當頻繁時，補償運算子(Repair operators)是效率高且有效的，可以調整適應值而不需太多費力的計算。在這裡描述一件例子來驗證。因此我們選擇補償運算子來調整染色體的適應值。由於染色體編碼，限制條件(13)在此 GA 裡會滿足。對於限制條件(14)-(16)，補償運算子會在下列三個步驟給定。

Step 1. 對於限制條件(14)而言，如果$\sum_{i=1}^{N}\sigma_{ij}<1$，換句話說第 j 個資源未配置在此網格上，補償運算子會隨機選擇一節點G_i，將資源R_j配置在上面，然後將γ_{ij}設定為 1。

Step 2. 對於限制條件(15)而言，如果$\sum_{j=1}^{K}\sigma_{ij}>\beta_i$，隨機選擇在節點$G_i$上的資源$\sum_{j=1}^{K}\sigma_{ij}-\beta_i$並將他們去除掉(換句話說，$R_x$是被選擇的資源，將相對的$\gamma_{ix}$設為 0)。若經由這個補償運算子而違反 *Step 1*，將跳回 *Step 1*，剩下的，跳到 *Step 3*。

Step 3. 對於限制條件(16)而言，如果$\sum_{i=1}^{N}\sum_{j=1}^{K}c_j\sigma_{ij}>B$，選機選擇一節點$G_i$，在節點之間配置最多的閒置資源$R_j$，然後將$\gamma_{ij}$設為 0。反覆此步驟直到$\sum_{i=1}^{N}\sum_{j=1}^{K}c_j\sigma_{ij}\le B$。

3.3.4 參數的選擇（初始族群、交配率、突變、世代）

在 GA 裡有些參數，如交配率，突變率，族群大小，演化終止前的演化世代數。這些參數機會影響 GA 的效用。因此選擇恰當的參數是很重要的。然而參數是困難的被決定而沒有經過任何的實驗。通常參數需要經由實驗的執行結果來調整。下面有一些規則有助於參數的能快速的被選擇而不須經由太多次的實驗。

交配率決定了交配運算的頻率，對最終的最佳解有顯著的影響。一般，交配率應該要很高，所以我們介紹一段範圍來選擇近似的交配率，大約在 80%和 95%之間。

突變是爲了避免 GA 收斂的太快而陷入區域最佳值。但是突變不應當太常發生，因為若突變率定的太高，這樣 GA 將會變成在隨機搜尋。突變率決定了突變運算，在一般情況下，突變率應該設的很低。所以我們介紹一段範圍來選擇適當的突變率，大約在 0.5%和 5%之間。

族群的大小是另一項重要的參數。也許是令人驚訝的，非常大的族群通常無法改善 GA 的結果(意思是找出解的速度)。好的族群大小通常在 20-30 之間，然而在這個問題上的最佳族群大小是在 50-120 之間。

在終止 GA 運算前的演化世代數也是一項重要的參數。一般而言，越多的演化世代數，會得更佳的最終解。然而太多的演化世代有時候在時間上是沒有效益

的，尤其在很大的複雜網格上，因為通常在前面的階段就會搜尋到完善的解，然後這個解經由好幾次的演化仍然會保持差不多的結果，直到得到另一個更佳的解為止。因此，從完善和運算時間的觀點，這個問題上的最佳演化世代，我們建議在 100-200 之間。

第四章　以序的最佳化為基礎的演算法

4.1 MRST 的搜尋

從演算法 2 的範例中，得知網格服務可靠度 GSR=0.9915。這是由所搜尋到的六條 MRST 所求得的的可靠度。若我們只用搜尋到的前三條 MRST 來估算可靠度，得到 GSR=0.9574，比起原先 GSR=0.9915，降低約 0.034 的可靠度。在網格中，當能完成服務的 MRST 越多，越能使網格服務可靠。但相對的，系統服務可靠度的估算也得花掉更多的時間，因為從圖 14.計算 $\Pr(\bar{E}_1 \Lambda \bar{E}_2 \,|\, E_3)$ 的二元樹，可以看出當條件元件 CV 的數目會隨著 MRST 的增加而增加，當條件元件 CV 越多時，會使二元樹展延越廣，而計算 $\Pr(\bar{E}_1 \Lambda \bar{E}_2 \cdots \bar{E}_{i-1} \,|\, E_i)$ 所花費的時間會越多，但卻只能增加少許的網格服務可靠度。當網格越大時，也就是能搜尋到越多的 MRST 時，更能看出搜尋所有 MRST 和搜尋前三條 MRST 這兩種方法，來估算可靠度所花費的時間的顯著差異。

4.2 補償運算子的修正

違背限制條件(14)-(16)，運用修正運算子(repair operator)來修正。在實際情況中，我們可能遇到的問題如下：在網格上，違背了限制條件(14)，即 $\sum_{i=1}^{N} \sigma_{ij} < 1$，換句話說第 j 個資源未配置在此網格上，隨機選擇一節點 G_i，將資源 R_j 配置在上面，使限制條件(14)滿足。跳到 *Step 2.* 若節點 G_i 違背了限制條件(15)，即 $\sum_{j=1}^{K} \sigma_{ij} > \beta_i$，隨機選擇在節點 G_i 上的資源 $\sum_{j=1}^{K} \sigma_{ij} - \beta_i$ 並將他們去除掉，使限制條件(15)滿足。跳回 *Step 1.* 檢查是否滿足限制條件(14)。若經由這個修正運算子而違反限制條件(14)，則必須繼續修正。如此反覆修正，直到限制條件(14),(15)均滿足。有可能導致花費許多時間調整配置的資源於網格裡的節點上。

因此，我們可以在一開始配置資源時，強制每種資源 R_j（$j = 1, \cdots, K$）至少都有一項固定配置在網格的節點上。這樣一來就會自動滿足限制條件(14)，即 $\sum_{i=1}^{N} \sigma_{ij} \geq 1$。若節點 G_i 違背限制條件(15)時，調整節點 G_i 的資源，就不會有在網格上，缺少所需資源 R_j（即 $\sum_{i=1}^{N} \sigma_{ij} < 1$）的情形發生。

另一個方法，當節點G_i違背限制條件(15)時，比較節點G_i上所配置到的資源R_j（$j=1,\cdots,m$）在網格裡，何者所配置的數量最多，選擇並加以刪除，可以大大降低違背限制條件(14)的機會。

我們由搜尋前三條 MRST 所求得網格服務可靠度，經由 GA 運算來決定網格環境中，資源的配置。最後再用 OO 來選擇 50 組配置好的環境，找出所有 MRST 來計算出完整的網格服務可靠度。最後在比較這兩個方法所求得網格服務可靠度和估算 GSR 所花費時間的差異。

第五章　範例

5.1 範例 1

假設一機構想要提供新的網格服務而需要配置 4 種資源(R1，R2，R3，R4)在網格上，在此網格上有 8 個 accessible nodes(N1,N2,…,N8) ，當中包含一個資源管理裝置(N1)，可以收到使用者的要求來管理服務。此虛擬網路架構的節點配置和連結方式如圖 8.所示。

服務的處理時間由 RN(G1)來控制，可以收到使用者的要求來管理服務。總預算上限 B 為 150K(NTD)。節點的失敗率如表 1.所示，連結的頻寬和失敗率如表 2.所示。每個節點上所配置的資源數$\beta_i(i=1,2,...,8)$的上限設為 2。表 3.為每種資源量的大小和使用此資源而所需的花費。

為了尋找到一個好的資源配置解，以 GA 為基礎下的程式，在”P4 1.6G”處理器的環境下，使用”Borland C++ 5.0”來運算。我們由實驗來調整 GA 的參數，最終選擇參數的組合如下所示：族群大小，100；交配率，0.9；突變率，0.02；演化世代數，100。

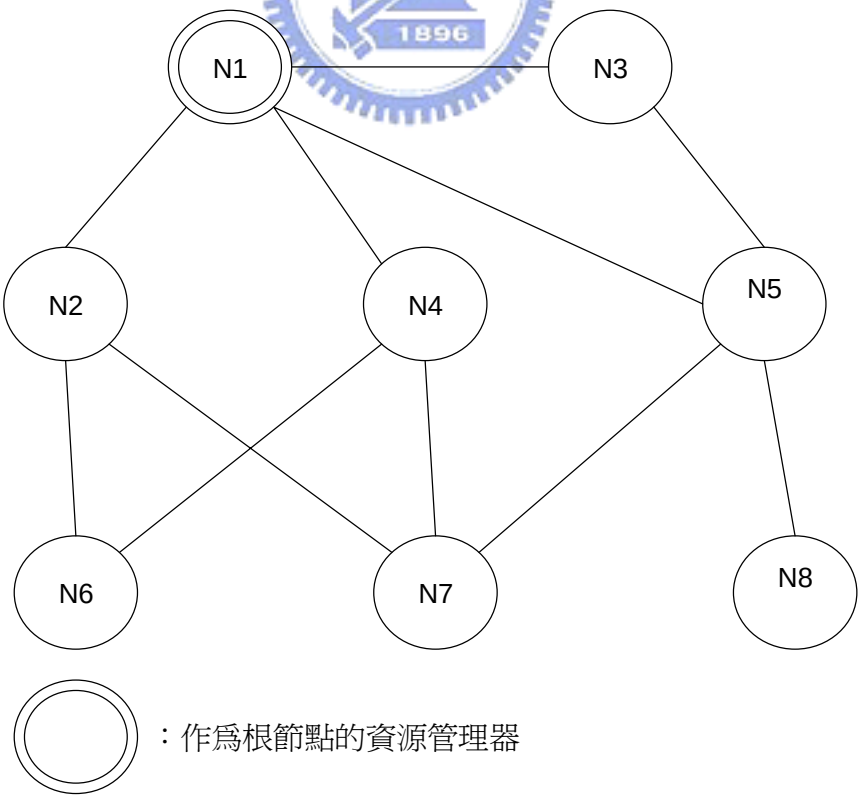


圖 8. 虛擬網路架構 1

表 1. 節點的失敗率

節點	N1	N2	N3	N4	N5	N6	N7	N8
失敗率	0.001	0.002	0.003	0.004	0.005	0.003	0.002	0.004

表 2. 每個連結的速度與失敗率

連結	L(1,2)	L(1,3)	L(1,4)	L(1,5)	L(2,6)	L(2,7)
速度 $S(i,j)$ (Kbps)	20	15	10	25	30	35
失敗率 $\lambda_{i,j}$	0.001	0.002	0.002	0.001	0.005	0.004
連結	L(3,5)	L(4,6)	L(4,7)	L(5,7)	L(5,8)	
速度 $S(i,j)$ (Kbps)	40	45	50	35	45	
失敗率 $\lambda_{i,j}$	0.006	0.006	0.005	0.004	0.005	

表 3. 資源量與花費

程式	執行時間 (sec)	所需資源	資源量 (Kbit)	花費 (NTD)
P1	50	R1,R2, R3,R4	300,200, 100,50	12K,7K, 14K,26K

表 4. 執行結果

	可靠度最大值	執行時間(sec)
GA	0.9602	20141.39
OO	0.9539	43.703

表 5. GSR 為最大的配置

	節點	N1	N2	N3	N4	N5	N6	N7	N8
GA	資源	R1,R3		R2		R2,R3	R2,R4	R2,R4	R4
OO	資源	R1,R3	R2	R4	R2		R2,R4	R2,R4	R3

使用 GA 經由 100 次世代演化後，和使用 OO 所得結果經由統計如表 4.所示，最佳分佈如表 5.所示。其中使用 GA 所求得的可靠度最佳值為 0.9602 且執行時間為 20141.39 秒，花費為 146K(NTD)。經由 OO 所得的網格服務可靠度最佳值為 0.9539，花費為 146K(NTD)。雖然可靠度低於原本約 0.006，但執行時間為 43.703 秒，節省了約 460 倍的時間。

5.2 範例 2

假設一機構想要提供新的網格服務而需要配置 4 種資源(R1，R2，R3，R4)在網格上，在此網格上有 6 個 accessible nodes(N1，N2，…，N6)，當中包含兩個資源管理裝置(N1，N3)，可以收到使用者的要求來管理服務。此虛擬網路架構的節點配置和連結方式如圖 9.所示。

服務的處理時間由 RN(N1，N3)來控制，可以收到使用者的要求來管理服務。總預算上限 B 為$150K。節點的失敗率如表 6.所示，連結的頻寬和失敗率如表 7.所示。每個節點上所配置的資源數 $\beta_i (i=1,2,...,6)$ 的上限設為 2。

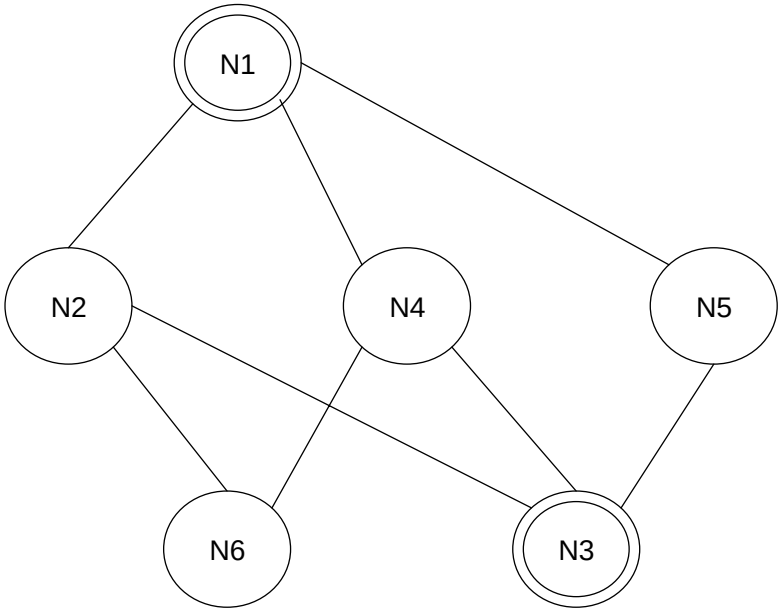


圖 9. 虛擬網路架構 2

表 6. 節點的失敗率

節點	N1	N2	N3	N4	N5	N6
失敗率	0.001	0.002	0.002	0.004	0.005	0.003

表 7. 每個連結的速度與失敗率

連結	L(1,2)	L(1,4)	L(1,5)	L(2,3)
速度 $S(i,j)$ (Kbps)	9	6	8	14
失敗率 $\lambda_{i,j}$	0.001	0.002	0.001	0.004
連結	L(2,6)	L(3,4)	L(3,5)	L(4,6)
速度 $S(i,j)$ (Kbps)	12	6	9	16
失敗率 $\lambda_{i,j}$	0.005	0.005	0.004	0.006

表 8. 執行結果

	可靠度最大值	執行時間(sec)
GA	0.9879	37219.219
OO	0.9845	93.234

表 9. GSR 為最大的配置

	節點	N1	N2	N3	N4	N5	N6
GA	資源	R1,R3	R1,R2	R2,R4	R1,R2	R1,R3	R4
OO	資源	R1,R3	R1	R2,R4	R2	R1,R3	R2,R4

使用 GA 經由 100 次世代演化後，和使用 OO 所得結果經由統計如表 8.所示，最佳分佈如表 9.所示。其中使用 GA 所求得的可靠度最佳值為 0.9879 且執行時間為 37219.219 秒，花費為 149K(NTD)。經由 OO 所得的網格服務可靠度最佳值

為 0.9845，花費為 137K(NTD)。雖然可靠度低於原本約 0.003，但執行時間為 93.234 秒，節省了約 400 倍的時間。

第六章 結論

在廣域分散式系統中，網格技術是很重要的發展方向。如何在有限的預算下，使資源有效的配置在網格系統中，讓網格服務可靠度最佳化也是很重要的問題。本篇論文討論網格服務可靠度模型和演算法的估算，使用 GA 來有效地找出資源最佳化配置，然後呈現了一個經由配置後，可以使網格服務可靠度最大化的最佳化模型。再來提出了修正方法和使用 OO 來找出數筆資源配置，雖然減少了些許的網格服務可靠度，但有效的節省工作時間。最後，使用兩個範例來說明與比較執行結果。

附錄 A.

我們在附錄 A 中，詳細地說明如何找尋能完成服務的 MRST 的演算法，包含演算法程序與範例。

Algorithm 1. 開始於給定的初始節點，沿著可能的連結去搜尋在其他節點上所需的資源和紀錄搜尋路徑，直到所有所需資源被找到，MRST 即找到，紀錄 MRST。

假設圖形裡有配置好的 n 個節點，m 種資源。

I. 名詞解釋

1. TV (Track vector)

用來記錄 完成 MRST 的路徑。

TV = ($t_1 t_2 \cdots t_n$)

未拜訪過的節點設為 d， TV = (dd…d)

起始節點設為 0

t_i 拜訪其鄰近且尚未拜訪過的節點 t_j， t_i(parent)= t_j(new)

- 若起始節點為 t_1， TV = (0d…d)
- t_1 拜訪其鄰近且尚未拜訪過的節點 t_2， TV=(01d…d)
  t_2 拜訪其鄰近且尚未拜訪過的節點 t_4， TV=(01d2d…d)
  依此類推。
- Delete(TV)　將紀錄的 追蹤路徑 刪除掉

2. RV (Resource vector)

用來判斷是否找到 完成 MRST 所需要的資源。

RV = ($r_1 r_2 \cdots r_j \cdots r_m$)

若所需找尋的資源為r_j

則r_j設為 1，其餘設為 0， RV =（00…1…0）

在拜訪節點V_i時，檢查V_i上是否有所需的資源r_j。

- 若存在r_j，則r_j設為 0。

當 RV =（00…0），則完成 MRST 的找尋。

- 若不存在r_j，則往V_i連接的鄰近且未拜訪過的節點尋找r_j。

- RV(TV) 追蹤的路徑上所搜尋到的資源。

II. 詳細描述

給定起始節點V_i，更新 TV，RV，j = 0。

拜訪V_i的所有相鄰且未拜訪過的節點V_a,…,V_e。

記錄每條路徑且檢查是否收集到所有所需資源。

V_a若收集到所需資源，則完成 MRST 的搜尋，i=i+1。

V_b若未收集到所需資源，照V_i的方式，沿著V_b繼續往下搜尋。

若V_b不存在未拜訪過的相鄰節點，則搜尋失敗，此路徑無法得到 MRST。

若 i>=K 或搜尋完所有從V_i出發的 MRST，則找到最多 K 條 MRST。

III. 執行步驟

Procedure MRST(V，E)

Step1. 輸入起始節點V_i，求得 TV 並置入*佇列*。 j=0。

Step2. $N = TV_{front}$; //令*佇列首*的 TV 為 N

Delete(TV_{front}); //釋放(刪除)TV_{front}

RV(N); //依紀錄的路徑而所得到的資源

Step3. 檢查路徑是否收集到所有所需資源

若未收集到， go to ***Step4***。

若收集到，則完成 MRST 的搜尋，j=j+1， go to ***Step5***。

Step4. V_N 為紀錄的路徑的最末端節點

拜訪 V_N 的所有相鄰且未拜訪過的節點 V_m， $m=(a,\cdots,e)$，

並依序紀錄路徑（將 $TV_a,\cdots,TV_e$ 依序存入***佇列***）。

若 V_N 不存在相鄰且未拜訪過的節點，

則此路徑無法產生 MRST。

Step5. 若{（ j > = K ）或（搜尋完所有從 V_i 出發而產生的 MRST(***佇列為空***)）}

則找到最多 K 條 MRST， stop。

else, go to ***Step2***。

IV. 範例說明

如下圖 10.所示:

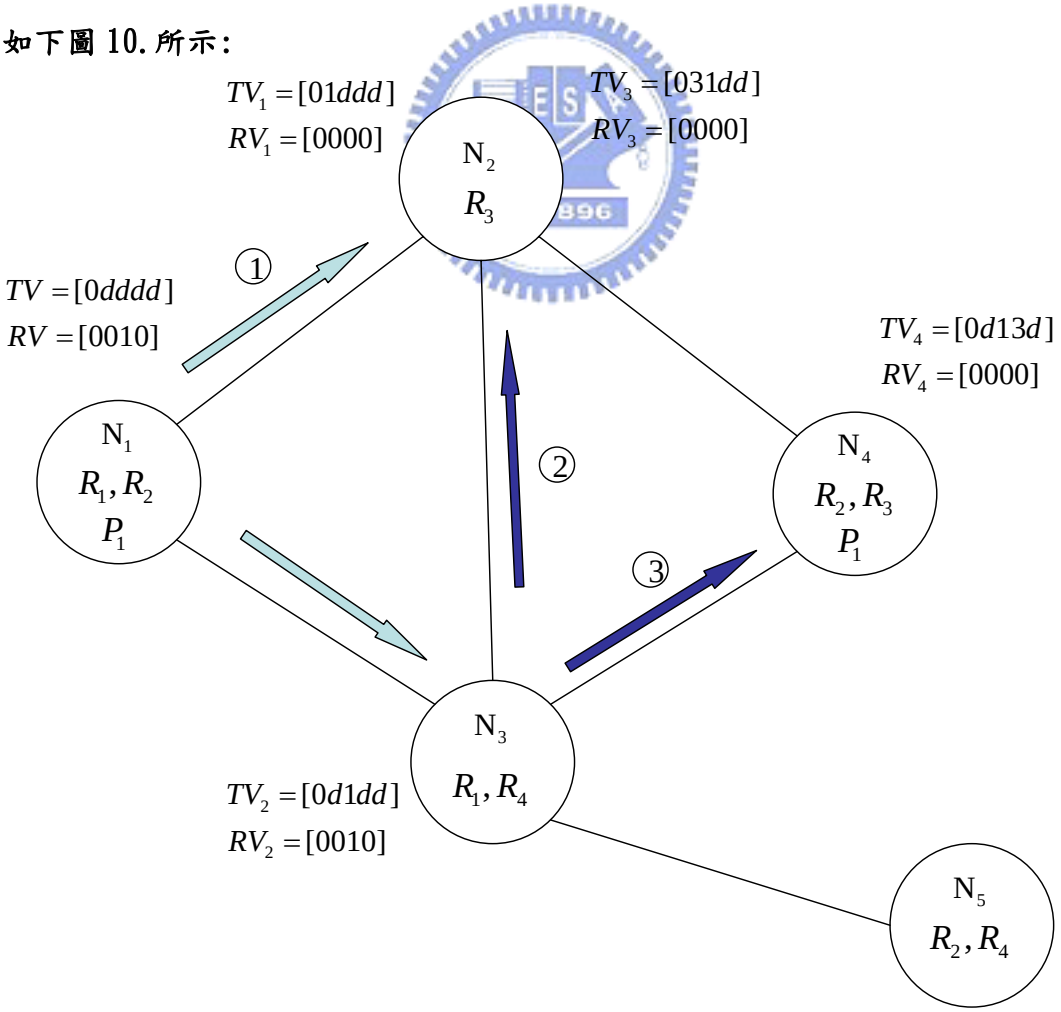


圖 10. MRST 的說明圖

說明:

使用者對 RN(N_1)要求服務(執行 P_1)，執行 P_1 需要資源 R_1, R_2, R_3 來協助完成

找尋可以完成執行 P_1 的 MRST

1. TV=(ddddd)， RV=(1111)　//將 TV，RV 初始化

2. 輸入起始節點(N_1)， 記錄 TV=(0dddd)，

放入*佇列*(queue)　//TV_queue [(0dddd)]

N=TV_{front}(0dddd)， 刪除TV_{front}(0dddd)　// TV_queue []

3. RV(N)=(0010)，欠缺資源 R_3

4. 拜訪 N_1 的所有相鄰且未拜訪過的節點 N_2, N_3

並依序紀錄路徑 (01ddd)， (0d1dd)

依序存入*佇列*(queue)　//TV_queue [(01ddd)，(0d1dd)]

N=TV_{front}(01ddd)， 刪除TV_{front}(01ddd)　// TV_queue [(0d1dd)]

5. RV(N)=(0000)， 找尋到 MRST， i=1

N=TV_{front}(0d1dd)， 刪除TV_{front}(0d1dd)　// TV_queue []

6. RV(N)=(0010)，欠缺資源 R_3

7. 拜訪 N_3 的所有相鄰且未拜訪過的節點 N_2, N_4, N_5

並依序紀錄路徑 (031dd)，(0d13d)，(0d1d3)

依序存入*佇列*(queue)　//TV_queue [(031dd)，(0d13d)，(0d1d3)]

N=TV_{front}(031dd)， 刪除TV_{front}(031dd)　// TV_queue [(0d13d)，(0d1d3)]

8. RV(N)=(0000)， 找尋到 MRST， i=2

N=TV_{front}(0d13d)， 刪除TV_{front}(0d13d)　// TV_queue [(0d1d3)]

9. RV(N)=(0000)， 找尋到 MRST， i=3

N=TV_{front}(0d1d3)， 刪除TV_{front}(0d1d3)　// TV_queue []

10. RV(N)=(0010)，欠缺資源R_3

11. N_5不存在相鄰且未拜訪過的節點

佇列(queue)為空，終止搜尋 MRST

12. $MRST_1$：1->2

$MRST_2$：1->3->2

$MRST_3$：1->3->4

附錄 B.

我們在附錄 B 中，詳細地說明如何計算 $\Pr(\overline{E}_1 \Lambda \overline{E}_2 \Lambda \cdots \overline{E}_{i-1} | E_i)$ 的演算法，包含演算法程序與範例。

Algorithm 2. 計算 $\Pr(\overline{E}_1 \Lambda \overline{E}_2 \Lambda \cdots \overline{E}_{i-1} | E_i)$

I. 名詞解釋

1. EV(Element Vector)

   用來記錄完成 TV 所用到的元件(節點與連結)。

   EV=$(e_1 e_2 \cdots e_n)$

   1 代表相對應的元件被包含在已搜尋過的路徑中，0 則不是。

   初始值 EV=(00…0)

2. WV(Operating time Vector)

   用來記錄相對應 EV 上每個元件所需的工作時間

   WV=$(w_1, w_2, \cdots, w_n)$

   初始值 WV=$(0,0,\cdots,0)$

3. MRSTE[j,k]

   當找尋到 MRST 時，MRSTE=EV

4. MRSTW[j,k]

   用來記錄相對應 MRSTE 上每個元件所需的工作時間

5. CEV(Conditional Elemt Vector)

   用來記錄使 $MRST_1, \cdots, MRST_{i-1}$ 其中一條失敗，而 $MRST_i$ 成功運作的條件元件

6. CV

   用來記錄條件元件結果

   $k \in [1, m]$

   CV[k]＝0 代表條件元件運作失敗

CV[k]=1 代表條件元件運作成功

7. OV

用來記錄相對應 CV 運作結果

$k \in [1, m]$

OV[k]=0 代表 $MRST_k$ 運作失敗

OV[k]=1 代表 $MRST_k$ 運作成功

II. 詳細描述

找出所有使 $MRST_j$ $(j = 1, 2, \cdots, i-1)$ 失敗，而 $MRST_i$ 能成功運作的條件要素。經由 Algorithm 1 找出 $MRST$ 後，記錄 $MRSTE[j,k]$ 和 $MRSTW[j,k]$，$(j = 1, 2, \cdots, i$；$k = 1, 2, , \cdots, K)$，其中 K 是在網格系統中，元件(節點和連結)的總數。

在 $MRST_1, \cdots, MRST_{i-1}$ 中，將與 $MRST_i$ 相同且工作時間大於 $MRST_i$ 的元件，記錄下來。在這些相同元件中，依工作時間，由小到大依序排序並記錄之。滿足這些條件的元件，稱為條件元件(能使 $MRST_1, \cdots, MRST_{i-1}$ 其中一條失敗，而 $MRST_i$ 成功運作)。

假設條件元件共有 m 個，其中 $MRST_i$ 的元件 j，工作時間為 $w_{i,j}$。若 $MRST_{i-1}, MRST_{i-2}$ 的元件 j 的工作時間分別為 $w_{i-1,j}$，$w_{i-2,j}$ 均大於 $w_{i,j}$，且 $w_{i-2,j} > w_{i-1,j}$。我們使用下面的圖 11. 來幫助說明

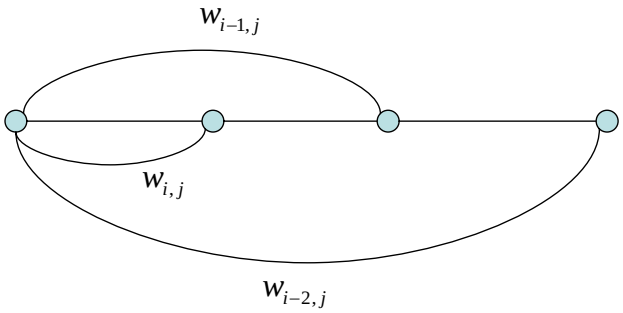


圖 11. $MRST_i$, $MRST_{i-1}, MRST_{i-2}$ 的元件 j 的工作時間

完成 $MRST_i$，$MRST_{i-1}$ 和 $MRST_{i-2}$ 運作的元件 j 的個別工作時間分別為 $w_{i,j}, w_{i-1,j}$ 和 $w_{i-2,j}$。因為 $MRST_i$ 是成功運作的，因此在 $w_{i,j}$ 這段工作時間內，元件 j 是成功運作的。而在 $(w_{i-1,j} - w_{i,j})$ 這段時間內，元件 j 有可能運作失敗而導致 $MRST_{i-1}$ 和 $MRST_{i-2}$ 運作失敗。因此起始時間 $T_b = w_{i,j}$，終止時間 $T_e = w_{i-1,j}$，條件元件的可靠度 $R(m) = e^{-\lambda(j)(T_e - T_b)}$。接下來 m=m+1，而在 $(w_{i-2,j} - w_{i-1,j})$ 這段時間內，

元件 j 有可能運作失敗而導致 $MRST_{i-2}$ 運作失敗。 $T_b = w_{i-1,j}$，$T_e = w_{i-2,j}$，依此類推，直到求出所有 $R(m)$。

使用二元搜尋樹來搜尋這些條件元件可以使 $MRST_1,\cdots,MRST_{i-1}$ 失敗而 $MRST_i$ 成功的所有可能組合，將所求得的機率累加起來，即為 $\Pr(\overline{E}_1 \Lambda \overline{E}_2 \Lambda \cdots \overline{E}_{i-1} | E_i)$。

III. 執行步驟

Procedure $\Pr(\overline{E}_1 \Lambda \overline{E}_2 \Lambda \cdots \overline{E}_{i-1} | E_i)$

Step1. 記錄 MRSTE[j,k]和 MRSTW[j,k] ($j = 1,2,\cdots,i$；$k = 1,2,,\cdots,K$)，其中 K 是在網格系統中，元件(節點和連結)的總數。

Step2. m=0。 $k \in [1,K]$ //K 是在 CEV 裡，條件元件的總數

OS=(MRSTW[j, k] | ($j = 1,2,\cdots,i$))

//OS 是記錄 $MRST_j$ ($j = 1,2,\cdots,i$)，第 k 個元件的工作時間。

Step3. n=0。 $t \in OS$ 且 $t > MRSTW[i,k]$

Step4. m=m+1， n=n+1

$T_b(m) = t(m-1)$ // $t(0) = MRSTW[i,k]$

$T_e(m) = t(m)$

$R(m) = e^{-\lambda(k)(T_e - T_b)}$

$CEV[m] \leftarrow \{k, T_b(m), T_e(m), R(m)\}$

If $t \in OS$ 且 $t > MRSTW[i,k]$， go to ***Step3***。

Else， go to ***Step5***。

Step5. 將 Y,CV,OV 初始化

Y=0

$CV[1,\cdots,m] = [dd\cdots d]$

$OV[1,\cdots,i-1] = [11\cdots 1]$

$\Pr(\overline{E}_1 \Lambda \overline{E}_2 \Lambda \cdots \overline{E}_{i-1} | E_1) = Y$

Step6. BinaryTree(CV,OV,z)

z=0

$CV_l[z]=0,\ \ CV_r[z]=1,\ \ OV_l[Fail(z)]=0$

If $OV[1,\cdots,i-1]=[00\cdots0]$

Y=Y+PR(CV)

Else if $CV[1,\cdots,m]=[11\cdots1]$

Stop

Else z=z+1

Function PR(CV)

$x=1$

$k\in[1,m]$

If $CV[k]=1$

$x=x\times R(k)$

Else if $CV[k]=0$

$x=x\times(1-R(k))$

PR=x

IV. 範例說明

考慮一個網格系統如圖 12.所示,當中包含了四個節點和五條連結，有四種資源配置在節點上，收集所需的資源來執行程式。表 10-12 是網格運算系統所需的資訊。

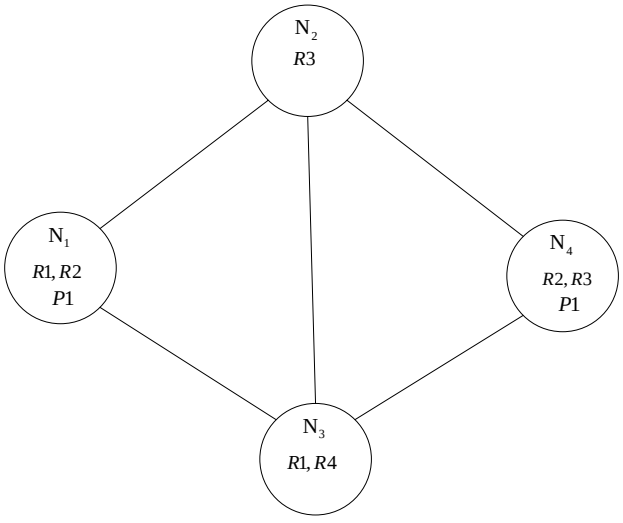


圖 12. 四個節點的網格系統

表 10. 每個連結的速度與失敗率

Links	L(1, 2)	L(1, 3)	L(2, 3)	L(2, 4)	L(3, 4)
Speed	30	20	40	50	45
$\lambda_{i,j}$	0.001	0.002	0.003	0.004	0.005

表 11. 執行程式所需的資料

程式	執行時間 (Sec)	所需資源	資源量
P1	30	R1, R2, R3	500, 400, 300

表 12. 節點的失敗率

節點	N1	N2	N3	N4
失敗率	0.001	0.002	0.003	0.004

表 13. 網格系統可靠度的估算

$MRST_i$	元件	工作時間	$\Pr(E_i)$	第二項
$MRST_1$	N1, N2, L(1, 2)	40, 10, 10	0.9324	-------
$MRST_2$	N1, N2, N3, L(1, 3), L(2, 3)	45, 7.5, 22.5, 15, 7.5	0.8353	0.0149
$MRST_3$	N1, N3, N4, L(1, 3), L(3, 4)	45, 21.7, 6.7, 15, 6.7	0.8184	0.0153
$MRST_4$	N3, N4, L(3, 4)	11.1, 41.1, 11.1	0.7763	0.0411
$MRST_5$	N2, N3, N4, L(2, 4), L(2, 3)	22.5, 12.5, 40, 10, 12.5	0.7324	0.0023
$MRST_6$	N1, N2, N4, L(1, 2), L(2, 4)	16.7, 26.7, 40, 16.7, 10	0.7507	0.0008

表 13.的資料是能使程式 P1 完成執行的所有 MRST(經由演算法 1)和每一個元件的工作時間，和經由方程式(8)所算出每條 MRST 的可靠度 $\Pr(E_i)$，還有經由演算法 2 計算的條件機率 $\Pr(\overline{E}_1 \Lambda \overline{E}_2 \Lambda \cdots \overline{E}_{i-1} | E_i)$。因此將表 4 的 $\Pr(E_i)$ 值和第二項也就是 $\Pr(\overline{E}_1 \Lambda \overline{E}_2 \Lambda \cdots \overline{E}_{i-1} | E_i)$ 的值帶入方程式(10)，P1 的網格服務可靠度

$GSR = 0.9915$。

依照演算法 1 的數字說明，程式 P1 在節點 N1 上執行而求得前三條 MRST，其中數據的變化如圖 13. 所示。

$MRST_1 : N_1 \rightarrow N_2$

$MRST_1 : N_1 \rightarrow N_3 \rightarrow N_2$

$MRST_1 : N_1 \rightarrow N_3 \rightarrow N_4$

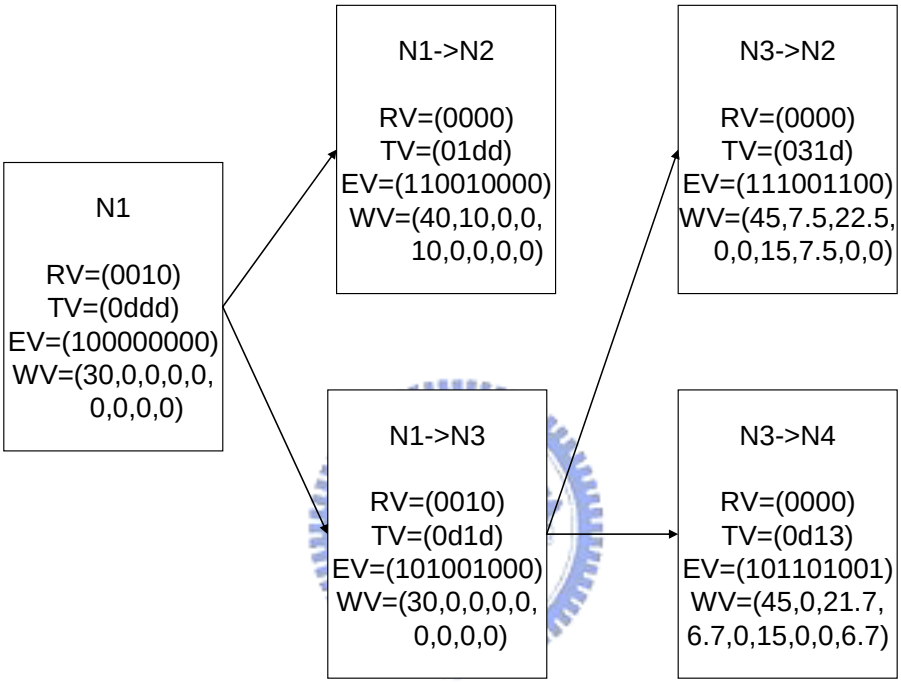


圖 13. MRST 的形成與數據

利用估算 $\Pr(\bar{E}_1 \Lambda \bar{E}_2 \,|\, E_3)$ 這個部份來詳細地說明演算法 2。

Step 1: 識別可以使 $MRST_1$ 或 $MRST_2$ 失敗而使 $MRST_3$ 成功執行的條件元件。

表 14, 表 15 秀出來自演算法 2 的 Step 1 到 Step 4 的結果。

表 14. 演算法 2 的 Step 1, 2

Elements	N1	N2	N3	L12	L13	L23	L34
$MRST_3(T_w)$	45	0	21.7	0	15	0	6.7
$T_w > MRST_3$		7.5	22.5			7.5	
		10		10			

表 15. 經由演算法 2 的 Step 3, 4 得到的 CEV_3

$CEV_3(m)$	$CEV_3(1)$	$CEV_3(2)$	$CEV_3(3)$	$CEV_3(4)$	$CEV_3(5)$
Element	N2	N2	N3	L(1,2)	L(2,3)
$T_b(m)$	0	7.5	21.7	0	0
$T_e(m)$	7.5	10	22.5	10	7.5
$R(m)$	0.9851	0.9950	0.9976	0.9900	0.9778

Step 2: 根據演算法 2 中的 Step 5, 6，用來估算 $\Pr(\bar{E}_1 \Lambda \bar{E}_2 \,|\, E_3)$ 的二元樹在圖 14. 中詳細的描述。

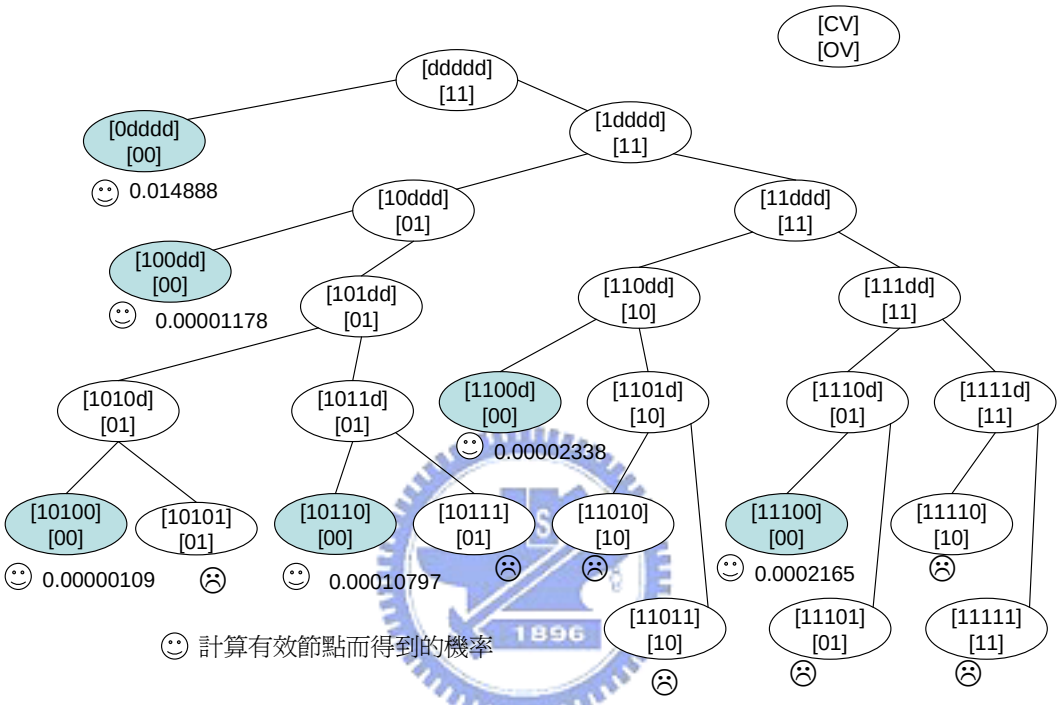


圖 14. 計算 $\Pr(\bar{E}_1 \Lambda \bar{E}_2 \,|\, E_3)$ 的二元樹

在圖 14. 中，☺代表的是當 $MRST_3$ 運作成功時，而 $MRST_1$ 和 $MRST_2$ 均失敗。因此☺的總和就是 $\Pr(\bar{E}_1 \Lambda \bar{E}_2 \,|\, E_3)$=0.015249。

參考文獻

[1] I. Foster, C. Kesselman and S. Tuecke, The anatomy of the grid: enabling scalable virtual organizations, Int J High Perform Comput Appl **15** (2001), pp. 200 – 222.

[2] S.K. Das, D.J. Harvey and R. Biswas, MinEX: a latency-tolerant dynamic partitioner for grid computing applications, Future Generat Comput Syst **18** (2002), pp. 477 – 489.

[3] Dai, YS, Xie M, Poh KL. Reliability analysis of grid computing systems. In: IEEE Pacific Rim international symposium on dependable computing (PRDC2002). 2002. p. 97 – 104.

[4] I. Foster and C. Kesselman, The grid: blueprint for a new computing infrastructure, Morgan-Kaufmann, San Francisco, CA (1998).

[5] I. Foster, C. Kesselman, J.M. Nick and S. Tuecke, Grid services for distributed system integration, Computer **35** (2002), pp. 37 – 46.

[6] K. Krauter, R. Buyya and M. Maheswaran, A taxonomy and survey of grid resource management systems for distributed computing, Software—Practice Experience **32** (2002), pp. 135 – 164.

[7] V.K.P. Kumar, S. Hariri and C.S. Raghavendra, Distributed program reliability analysis, IEEE Trans Software Eng **SE-12** (1986), pp. 42 – 50.

[8] A. Kumar, S. Rai and D.P. Agarwal, On computer communication network reliability under program execution constraints, IEEE J Select Area Commun **6** (1988), pp. 1393 – 1400.

[9] D.J. Chen and T.H. Huang, Reliability analysis of distributed systems based on a fast reliability algorithm, IEEE Trans Parallel Distribute Syst **3** (1992), pp. 139 – 154.

[10] A. Kumar and D.P. Agrawal, A generalized algorithm for evaluating distributed-program reliability, IEEE Trans Reliab **42** (1993), pp. 416 – 424.

[11] D.J. Chen, R.S. Chen and T.H. Huang, A heuristic approach to generating file spanning trees for reliability analysis of distributed computing systems, Comput Math Appl 34 (1997), pp. 115 – 131.

[12] M.S. Lin, M.S. Chang and D.J. Chen, Efficient algorithms for reliability analysis of distributed computing systems, Inform Sci 117 (1999), pp. 89 – 106.

[13] M.S. Lin, M.S. Chang, D.J. Chen and K.L. Ku, The distributed program reliability analysis on ring-type topologies, Comput Oper Res 28 (2001), pp. 625 – 635.

[14] Y.S. Dai, M. Xie, K.L. Poh and G.Q. Liu, A study of service reliability and availability for distributed systems, Reliab Eng Syst Saf 79 (2003), pp. 103 – 112.

[15] M. Livny and R. Raman, High-throughput resource management, The grid: blueprint for a new computing infrastructure, Morgan-Kaufmann, San Francisco, CA (1998), pp. 311 – 338.

[16] M. Xie, Y.S. Dai and K.L. Poh, Computing systems reliability: models and analysis, Kluwer Academic Publishers, New York (2004).

[17] B. Yang and M. Xie, A study of operational and testing reliability in software reliability analysis, Reliab Eng Syst Saf 70 (2000), pp. 323 – 329.

[18] C.D. Lai, M. Xie, K.L. Poh, Y.S. Dai and P. Yang, A model for availability analysis of distributed software/hardware systems, Inform Software Technol 44 (2002), pp. 343 – 350.

[19] Y. S. Dai and X. L. Wang, "Optimal resource allocation on grid systems for maximizing service reliability using a genetic algorithm," Reliability Engineering and System Safety, vol. 91, no. 9, pp. 1071–1082,2006.

[20] A. Kumar, R. Pathak and Y. Gupta, Genetic algorithm-based reliability optimization for computer network expansion, IEEE Trans Reliab 44 (1995), pp. 63 – 72.

[21] D. Coit and A. Smith, Reliability optimization of series-parallel systems using genetic algorithm, IEEE Trans Reliab 45 (1996), pp. 254 – 266.

[22] Z. Yangping, Z. Bingquan and W. Dongxin, Application of genetic algorithm to fault diagnosis in nuclear power plants, Reliab Eng Syst Saf **67** (2000), pp. 153 – 160.

[23] M. Marseguerra, E. Zio and M. Cipollone, Designing optimal degradation tests via multi-objective genetic algorithms, Reliab Eng Syst Saf **79** (2003), pp. 87 – 94.

[24] G. Levitin, Y.S. Dai, M. Xie and K.L. Poh, Optimizing survivability of multi-state systems with multi-level protection by multi-processor genetic algorithm, Reliab Eng Syst Saf **82** (2003), pp. 93 – 104.