東海大學資訊工程學系研究所

碩士論文

指導教授：林祝興 博士
Dr. Chu-Hsing Lin

植基於雲端技術最佳行車時間之路徑搜尋方法
Finding Shortest-Driving-Time Paths Based on Cloud Technology

研究生：劉明竑
Ming-Hung Liu

中華民國一〇二年六月

ABSTRACT

Intelligent Transportation System (ITS) provides real-time communications and connections; it increases and enhances interactions among people, and vehicles and improves traffic service quality. In this thesis, we propose to use a genetic algorithm, which is boosted by cloud computing infrastructure (MapReduce) to accelerate computation of map information. In addition, we use the concept of map layers to realize the ideas of "partial paths reusability" and "cloud computing results reusability" to enhance the navigation performance and improve the efficiency and accuracy of the overall navigation system. From the experiments, we observed that when the number of nodes of the map increases, the required execution time of the genetic algorithm increases, and the difference between the approximately optimal solution and the optimal solution increases as well. Also, for maps with a large number of nodes, one can use different chromosome initialization settings and different feedback mechanisms in cloud computing, to avoid problems such as chromosome initialization failure or premature convergence of the genetic algorithm that makes it very hard or impossible to find the approximately optimal solution in time. For applications with dramatic number of nodes, the proposed approach using the genetic algorithm and cloud technologies (MapReduce) is a suitable solution since it can offer approximate solutions without consuming lots of resources. The experimental results also showed that the proposed approach can obtain approximately optimal solutions faster and better.

Keywords: Cloud computing, ITS, MapReduce, Genetic algorithm, Partial paths reusability

中文摘要

智慧型運輸系統（Intelligent Transportation Systems, ITS）係透過通訊系統即時的溝通與連結，改善或強化人、車、路之間的互動關係，提升用路人的交通服務品質與績效。本論文以基因演算法為主要演算法，並輔以雲端運算架構(MapReduce)加速地圖資訊的運算，再配合圖層的切割與雲端計算結果再利用概念，實現”部分路徑可再利用性”，以達到節省時間並增進導航效能目的，並有助於提升整體導航之效率與準確度。從實驗結果中，我們觀察到在地圖上節點數目愈多的情況下，基因演算法所需要的執行時間也相對的愈多，而求得的近似最佳解的誤差也愈大。且根據不同染色體初始化的方式，以及使用不同雲端運算架構之回饋運算機制，在節點數目龐大的地圖上可以避免初始化失敗或陷入演算法容易過早收斂而導致無法找到近似最佳解的情形。隨著節點數劇烈增加，基因演算法配合雲端技術(MapReduce)可以在不需耗用太多資源就可以解決這類問題。實驗結果也顯示我們獲得較快及較佳的近似最佳解。

關鍵字：雲端運算，智慧型運輸系統，MapReduce，基因演算法，路徑再利用

致謝

首先要先感謝家人，不管是經濟上或精神上，都給了我最大的鼓勵與支持。雖然碩士短短的兩年，但這段期間，發生了許許多多酸甜苦辣的事情，有歡笑，當然也有淚水，但，我走過來了。在這兩年的期間，林祝興老師是我的授業恩師，林老師的教導並非只侷限於專業領域的知識上，平常更時常與我們分享有關心靈的成長與做人處世的態度。同時感謝劉榮春老師平時在論文上的訂正指導，令我受惠良多。還有賴威伸老師，百忙中依然撥空前來指導我們。在此要感謝百忙中抽空前來參與口試的委員們：詹進科教授、楊中皇教授、賴威伸教授、劉榮春教授，及常常遠從新竹到台中來指導同學們的鎮宇學長，給予學生在論文上的指正與建議，使得論文更加的充實與完整。

實驗室裡總是充滿了歡樂，因為有了學業上的夥伴。感謝我的同學：帥氣搞笑的奇軒、熱心負責的侑廷、安分守己的俊良；同時也感謝陪伴我的好學弟們：帥到不行的紘侑、修圖繪畫超厲害的傑立、穩如泰山的文琛，還有系辦的學姊簡詩蓓，感謝有你們的火力支援以及笑聲陪伴著我畢業。最後再次感謝父母、師長及親友們，感謝你們的支持、鼓勵與包容，在此致上最高的感謝。

劉明竑　謹上

於東海大學資訊工程學系研究所

資訊安全實驗室

2013 年 7 月

目錄

內容

圖表列表

CHAPTER 1　簡介

1.1 前言

地圖是人類重要的發明之一，使用者透過地圖，可以得知某一區域的地理位置概況；當人們旅行時想要前往某個陌生的地區時，可以透過地圖的幫助，導覽出兩地的相關地理位置資訊，然而如何在兩地之間路線的選擇則必須根據旅行者的本身的狀況和經驗下去評估，如此一來，不同的使用者往往會選擇出不同的路徑結果。正因為如此，地理資訊系統(Geographic Information System，GIS)[8]的出現，顛覆了傳統紙本地圖的顯現方式，用方便的縮放方式以及平移模式來操作地圖顯示功能，使得地圖使用者片段閱讀的門檻大大地降低許多；隨著科技的快速發展演進，更因為程式設計的能力提升，地圖不再只是顯示出區域地理位置的資訊工具，漸漸成為了一種決策工具，而出現了最短路徑(Shortest Path)和最佳行車時間(Best Driving Time)[23][24][25]的計算結果決策，其中更牽扯到即時行車路況和行車速率等因素，也因此在多方因素的考量下，決定出最佳行車路徑規劃(Best Driving Route Planning)成為了一個熱門的研究課題。不久前蘋果公司(Apple Inc.)中的地圖功能發布後備受使用者批評，使得 CEO 庫克不得不發聲明道歉。最後傳出負責行動地圖的高層威廉森（Rich Williamson）被迫下台，為蘋果地圖的失敗做出行動上的表示。如此事件顯示出關於地圖和導航的研發重要性和熱門度。

智慧型運輸系統（Intelligent Transportation Systems, ITS）[5][17]係透過通訊系統即時的溝通與連結，改善或強化人、車、路之間的互動關係，提升用路人的交通服務品質與績效，進而增進運輸系統之安全、效率與舒適，同時減少交通環境衝擊。本論文以基因演算法[26][28][29][30]為主要演算法，並輔以雲端運算架構(MapReduce)加速地圖資訊的運算，再配合圖層的切割概念，實現”部分路徑可再利用性”，以達到節省最佳行車時間之路徑搜尋時間

並增進導航效能目的，並有助於提升整體行車時間之效率與準確度。另一方面雲端概念的興起提供 ITS 整合建置的契機，龐大且同性質的交通偵測資料處理程序，可用雲端運算的方式，有效提升即時交通資訊服務效率及服務容量，降低營運成本，並提高用路人行車效率和便利性。

1.2 研究動機與目的

地理資訊系統早期是以單機板(GIS)的型態存在，後來演變成以網路版的地理資訊系統(WebGIS)[22]的樣貌；不過因車載電子的產品興起，車用導航機在路徑規劃方面，仍然是以單機板的軟體為主流，功能也較強大完整，如知名導航公司PAPAGO、GARMIN等導航軟體所提供的路徑規畫決策，可讓使用者選擇以國道為主，避開車多壅擠的特定路段，或是選擇避開收費站等除了導航外之附加功能，不過，雖然單機板導航機功能強大，然而目前常見的應用方式仍然是以GPS搭配地圖資訊系統進行行車導航，但它們大多數屬於封閉式系統，使用者必須不斷地購買新的地圖資訊，才能保有最新版的資料。若無更新至最新的地圖道路資訊，可能會因道路重新規劃或整頓而有所變化，造成行車用路人的導航機與真實道路情況不一致的窘境發生。

由於目前在行車導航方面，屬於開放式、且廣泛地被人採用的系統仍然非常稀少，因此我們的論文將利用現今蓬勃發展之技術，建構一個計算最佳行車路徑的演算法，而演算法或所用之技術皆為開放式的，因此當智慧型行車導航系統使用此演算法時，相對於傳統封閉式的自由度和自定性也能有所提升。另一方面雲端概念的興起提供ITS 整合建置的契機。龐大且同性質的交通偵測資料處理程序，可用雲端運算的方式，有效提升即時交通資訊服務效率及服務容量，降低營運成本。

此外若因雲端服務產生單一資料庫介面，則可加速國內交通資訊產業的加值應用發展。且各地方政府及交通單位以此架構來建置資料及運算中心，可減少硬體成本投資，逐步擴充服務容量，避免重覆投資交通控制軟硬體設備於交通管理中心。

目前的趨勢為在 Web GIS[21]的服務也就是與運算大量的資料的雲端運算結合，例如 Google Map[9] 0~19 的圖層的產生稱為影像金字塔圖層，也有人稱作 Tile，這種服務 OGC 也在草擬 OGC Tile Service，使用 Tile 一片片建成完每一層，各層建完後就成了影像金字塔。真實世界中的埃及的金字塔也是用一個個大型的磚一層一層建立。Google / Microsoft 呈現的這些影像金字塔圖層都已經預先產生好了。若要更新全世界的影像金字塔圖層得花上不少時間,若使用雲端運算,可以減少不少的時間。尤其是地圖繪圖運算 及大量小圖(tiles)IO 存取的時間會大符的減少。雲端運算讓繪圖更快是一定的，比較特別是 IO 存取的時間。以下就加以說明，通常架構簡單的話速度相對而言也是愈快，於是使用 NAS 來存放是不錯的選擇，但是提供 Web GIS 服務時大量檔案 IO 存取會讓 NAS、PC 甚至 Server 無法負荷。透過雲端的大量電腦將產生/存取小圖的 IO 時間大量的分散,達到快速大量服務的目的(high Scalability)。

在 Web GIS 上呈現向量的資料是很消耗頻寬及瀏覽器運算能力的事，例如台灣的鄉鎮行政區的資料大約 7MB，總不能讓每個使用者都下載 7MB 的向量地圖完後再呈現。在網路環境下比較好的作法是將向量資料轉成圖片(tiles)再呈現。這樣多大量的資料都變成固定大小的圖片，速度快了不少。但是少了向量資料的特性，無法動態變更樣式及無段縮放大小(所以 Web GIS 常用固定比例尺分層)。

Google Map 最近也把影像金字塔的雲端運算運用在 Google Map API[20]上，使用 GGeoXML 來呈現向量資料時，Google 會轉這些向量資料轉換成小圖(tiles)再呈現，這樣速度快非常多，尤其是呈現大量的多邊形，例如行政區、地籍圖…等。

在建構一套智慧型行車資訊系統時，勢必需要從地圖資訊、導航演算法等方向著手，試圖建構一個能讓使用者參與、提供地圖資訊、導航演算法為目標，以期能達到開放原始碼之精神。本論文主要集中於智慧型運輸系統ITS產業主要涵蓋資訊、通信、汽車電子與數位內容，提供人、車、路、生活等許多的智慧型服務。目的為利用資通訊科技技術(雲端平行化運算)提升運輸系統的效率、安全與便利性。由於資通訊技術的成熟，讓原本處於閉塞而無法與外界進行單向或雙向資訊傳輸互動的汽車環境，增加與外界溝通的通訊能力，也因此對車載資通訊系統產生了強烈的市場需求。在通訊技術中，車用通訊產品逐漸朝網路化、智慧化及整合化三大趨勢發展。

CHAPTER 2　背景知識與相關技術

2.1 智慧型行車資訊系統

智慧型行車資訊系統(Intelligent Transportation System, ITS)的重點在於能提供地圖資訊及路徑導航等實用功能，並透過使用者的行車情況，利用手持裝置收集行車資訊應用於智慧型車輛運輸系統。使用智慧型行車資訊系統是以手持裝置輸入所設定的導航路徑，傳送至雲端經由雲端平行運算技術實現基因演算法[23]計算出所要走的路徑，然後回傳至手持裝置，逐步指示所需行走的路徑，過程中手持裝置的 GPS 搭配地圖資訊系統進行行車導航，最後抵達目的地。一般沒使用 ITS 的路徑導航系統為封閉式系統，使用者必須不斷地定時購買更新新的地圖資訊，方能保有最新版的資料。而加入 ITS 的使用者因為有與我們所建置的雲端 Server 互動，進而可以隨時隨地獲得新的地圖資訊，以及即時的交通流量，導航的結果更符合實際需求。ITS 所使用的平面道路資訊，是由交通部運輸研究所所提供的，目前為全台最詳盡的地圖，包含國道，省道(含快速道路) ，縣道，鄉道，都市道路，產業道路即無路名道路等既有道路，這些路網的比例都有達到 1/5000。ITS 的即時路況是經由交通部的公路總局所提供的即時路況資料庫來達到即時的路況變更，使用 ITS 功能的使用者都可以快速且安全地抵達目的地。

2.2 雲端運算

雲端運算[2][3]是一種新穎的運算模式並且越來越熱門，它提供了使用者大量的運算能力，儲存能力，和提供軟體資源的需求。雲端運算被分為兩大類，即雲端服務[4]和雲端技術。雲端服務[9][10][11]是透過網路連接到遠端存取服務。這些服務為使用者提供了安裝和使用多種不同的作業系統，例如亞馬遜網路服務（CE2 和 S3），可以看成是這種雲端服務的概念：基礎

設施即為服務（IaaS）和儲存即為服務，軟體即為服務（SaaS）的概念，這兩種服務為雲端服務中需求量最大的!而平台即為服務（PaaS）的概念是雲端計算服務的替代方案。通過這些服務，使用者甚至可以簡單地依靠手機或輕量化的用戶端做很多事情，然而在過去，這些事情只能在個人電腦上完成，這意味著，雲端運算是無所不在的。雲端技術的目的是在建立並且散佈電腦各種運算資源的虛擬化[6][12][13]和使用自動化技術。雲端運算的目的是實現低成本（節省錢），節能（能源效率），以及無所不在（在任何時間，任何地點，任何裝置存取任何服務）。

2.3 Hadoop

2.3.1 HDFS

Hadoop Distributed File System 為 Hadoop 的自由軟體專案，為實現 Google 的 MapReduce 架構，Google File System。其架構為 Namenode 和 Datanode，Namenode 為 Master 負責管理 HDFS 的名稱空間並控制對檔案的讀/寫和配置副本策略以及對名稱空間作檢查及紀錄。而 Datanode 為 Workers 負責執行讀/寫動作和執行 Namenode 的副本策略。其對工作的分配又可分 Jobtracker 和 Tasktrackers。Jobtracker 為 Master 負責對使用者發起工作並指派工作給 Tasktrackers 進行排程決策、工作分配、錯誤處理。而 Tasktrackers 為 Workers 負責運作 Map 與 Reduce 的工作和管理儲存、回覆運算結果。

分散式儲存的方式是為了實現計算向儲存遷移的可能性[32]，計算和儲存整合是雲端運算的一大特性，並可以說是如果沒有實現計算與儲存整合的系統就不能稱作是雲端運算系統；因為在這個資訊發達的時代，我們所要面對的問題類型都不盡相同，有的問題是資料密集型的問題，有的問題是計算密集型的問題，當然也會有資料和計算同時密集型的問題，所以我們為了要因應不同類型的問題，就必須將資料密集和計算密集系統整合在一起[33]。

系統首先獲得計算的工作任務，工作的檔案讀取為先到達子節點獲得該節點資料區塊的儲存資訊後，會區分為負責儲存或負責計算的單元如下：

負責儲存資料的單元可區分為：

1.Namenode: 擔任 master 的角色，管理檔案的讀寫，以及 Replication 的策略。

2.Datanodes: 擔任 worker 的角色，接受 Namenode 的指揮，實際儲存資料的 Replication，並執行檔案的讀/寫，將資料提供給 Tasktracker 進行運算。

執行運算的單元可區分為：

1.Jobtracker: 擔任 master 的角色，接受使用者發起的工作，並進行工作排程（Job Scheduling），將工作（Job）分派給 Tasktrackers 執行，並彙整 Tasktrackers 的計算結果後，將最終結果回傳給使用者。

2.Tasktrackers: 擔任 worker 的角色，接受 Jobtracker 的工作指派，跟 Datanodes 要求待運算的資料，執行實際的運算後，將結果回傳給 Jobtracker 彙整。

當一個運算任務（Job）要被執行時，JobTracker 會將要執行的程式(相較之下檔案很小)傳送到資料所在的位置(Datanodes)，而後待執行的運算便就地在 Datanode 上由 Tasktracker 完成，這就是以運算就資料，與其將很大量的資料到程式透過網路搬運到程式所在的機器上，不如先搞清楚資料在哪裡，接下來把小小的程式複製到資料所在的位置去執行。在網路上傳輸要執行的程式：Client 到 Jobtracker 再分配到 Datanodes。

在 Datanodes 本機上傳輸等待運算的大量資料時：Datanodes Disks 直接傳輸到 Datanodes memory，完全不需要透過網路，速度當然因而提升。圖 2-1 為 HDFS 架構:

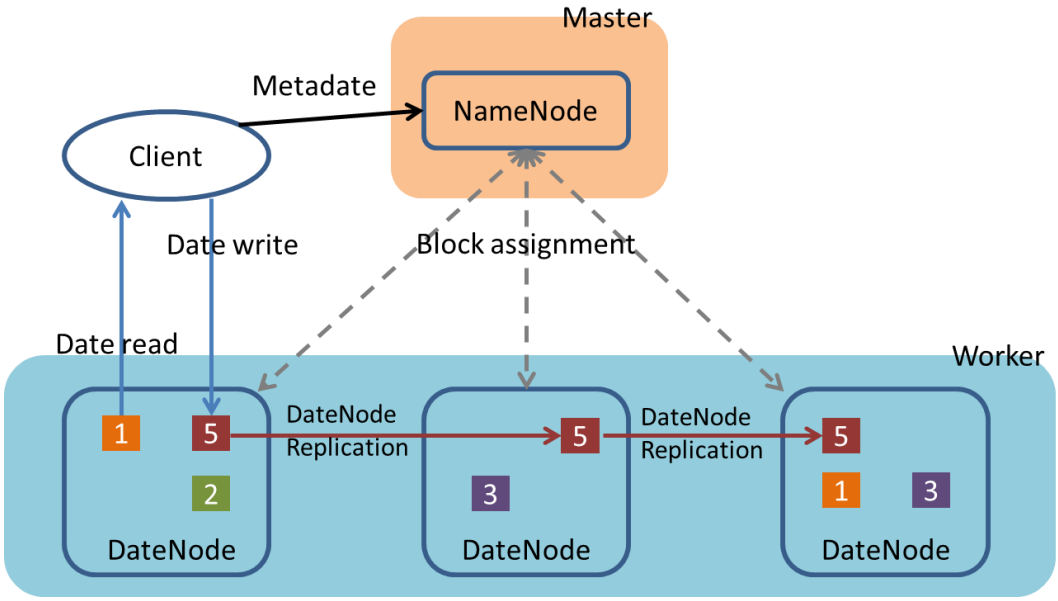


圖 2-1 HDFS 架構

2.3.2 HBase

HBase 為一種類似表格的資料結構，為分散式，高可用性、高效能，很容易擴充容量及效能等等優點。Hbase 適用於利用數以千計的一般伺服器上，來儲存 Petabytes 等級的資料。HBase 以 Hadoop 分散式檔案系統 (HDFS) 為基礎， 提供類 Bigtable 功能，同時也提供了 Hadoop MapReduce 程式設計。它不是關聯式(Relational)資料庫系統[1]，表格(Table)只有一個主要索引(primary index) ，即 row key；它不提供 join 也不提供 SQL 語法。它提供 Java 函式庫，與 REST 與 Thrift 等介面，也提供 getRow()，Scan()存取資料。getRow()可以取得一筆 row range 的資料，同時也可以指定版本(timestamp) ，Scan()可以取得整個表格的資料或是一組 row range (設定 start key, end key) ，而它為有限的單元性(atomicity)與交易(transaction)功能，只有一種資料型態(bytes)。如表 2-1 所示:

表 2-1 HBase 儲存格式範例

name	timestamp	score	phone
ricky	T2	Math: 65	
	T5	Eng: 30	
james	T1		61234
	T3	Math: 90	
	T6	Math: 95	61235
jason	T4		64321

2.3.3 MapReduce

雲端運算的關鍵技術是 MapReduce，最早是由 Google 提出，後來也運用在開放原始碼的雲端技術 Hadoop 中。MapReduce 是雲端運算的關鍵技術，將要執行的問題，拆解成 Map 和 Reduce 的方式來執行，以達到分散運算的效果。使用 MapReduce 模式，開發人員在拆解問題的過程中，就要先對問題進行平行化處理。開發人員需要先分析問題的解決流程，找出可以利用平行運算來處理資料的部分，也就是那些能夠被切成小段分開來處理的資料，再針對可以採用平行處理的部分寫成 Map 程式。

有了 Map 程式之後，就可以使用大量機器用執行 Map 程式，分別同步處理分析每一段資料，再將每一個 Map 程式分析出來的結果，透過 Reduce 程式進行合併，最後則彙整出完整的結果。

MapReduce 程式的執行過程如下：

1.將要執行的 MapReduce 程式複製到 Master 與每一臺 Worker 機器中。

2.Master 決定 Map 程式與 Reduce 程式，分別由哪些 Worker 機器執行。

3.將所有的資料區塊，分配到執行 Map 程式的 Worker 機器中進行 Map。

4.將 Map 後的結果存入 Worker 機器的本地磁碟。

5.執行 Reduce 程式的 Worker 機器，遠端讀取每一份 Map 結果，進行彙整與

排序，同時執行 Reduce 程式。

6.將使用者需要的運算結果輸出。

如圖 2-2 所示:

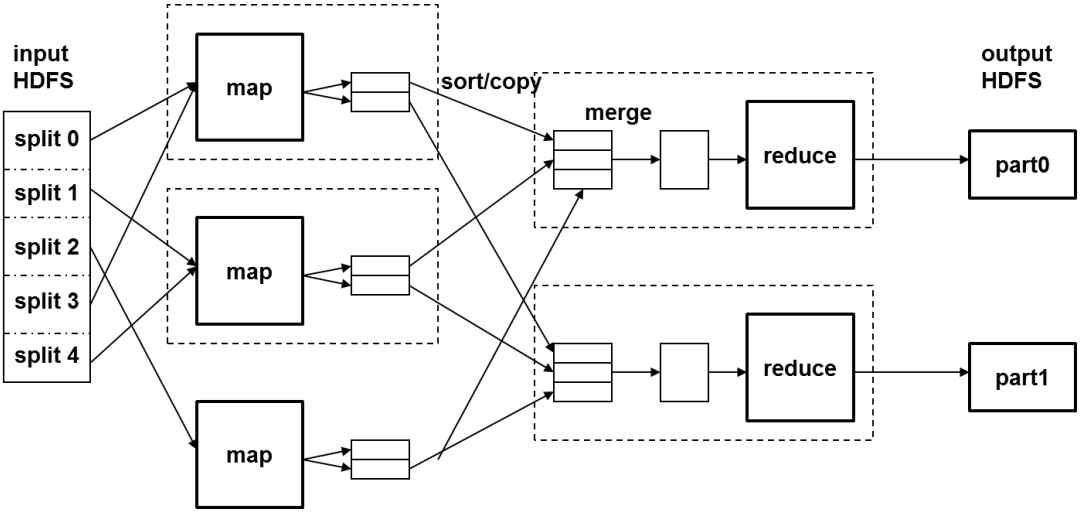


圖 2-2 MapReduce

另外在系統的運作架構上，最簡單的 Hadoop 架構，可以分成上層的 MapReduce 運算層以及下層的 HDFS 資料層。

在 Master 節點的伺服器中會執行兩套程式，一個是負責安排 MapReduce 運算層任務的 JobTracker，以及負責管理 HDFS 資料層的 NameNode 程式。而在 Worker 節點的伺服器中也有兩套程式，接受 JobTracker 指揮，負責執行運算層任務的是 TaskTracker 程式，而與 NameNode 對應的則是 DataNode 程式，負責執行資料讀寫動作，以及執行 NameNode 的副本策略。

在 MapReduce 運算層上，擔任 Master 節點的伺服器負責分配運算任務，Master 節點上的 JobTracker 程式會將 Map 和 Reduce 程式的執行工作，指派給 Worker 伺服器上的 TaskTracker 程式，由 TaskTracker 負責執行 Map 和 Reduce 工作，並將運算結果回覆給 Master 節點上的 JobTracker。

在 HDFS 資料層上，NameNode 負責管理和維護 HDFS 的名稱空間、並且控制檔案的任何讀寫動作，同時 NameNode 會將要處理的資料切割成一個 個檔案區塊（block），每個區塊是 64MB，例如 1GB 的資料就會切割成 16 個檔案區塊。NameNode 還會決定每一份檔案區塊要建立幾個副本，一般來說，一個檔案區塊總共會複製成 3 份，並且會分散儲存到 3 個不同 Worker 伺服器的 DataNode 程式中管理，只要其中任何一份檔案區塊遺失或損 壞，NameNode 會自動尋找位於其他 DataNode 上的副本來回復，維持 3 份的副本策略。 在一套 Hadoop 叢集中，分配 MapReduce 任務的 JobTracker 只有 1 個，而 TaskTracker 可以有很多個。同樣地，負責管理 HDFS 檔案系統的 NameNode 也只有一個，和 JobTracker 同樣位於 Master 節點中，而 DataNode 可以有很多個。

不過，Master 節點中除了有 JobTracker 和 NameNode 以外，也會有 TaskTracker 和 DataNode 程式，也就是說 Master 節點的伺服器，也可以在本地端扮演 Worker 角色的工作。如圖 2-3 所示:

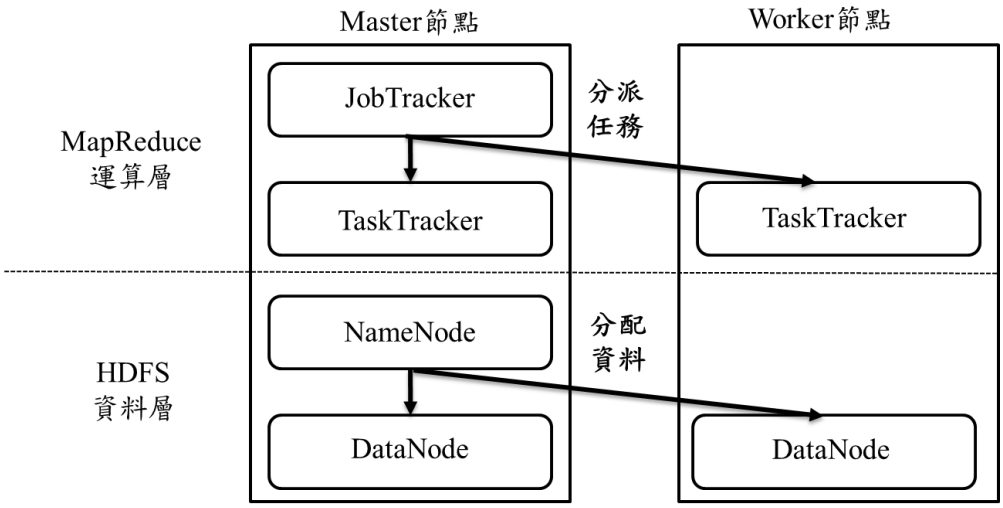


圖 2-3 Hadoop 架構角色分工

2.3.4 Hive

Hive 是建置在 HDFS 上的一套分散式資料倉儲系統，可讓使用者以慣用的 SQL 語法，來存取 Hadoop 檔案中的大型資料集，例如可以使用 Join、Group by、Order by 等，而這個語法稱為 Hive QL。不過，Hive QL 和 SQL 並非完全相同，例如 Hive 就不支援 Store Procedure、Trigger 等功能。

Hive 會將使用者輸入的 Hive QL 指令編譯成 Java 程式，再來存取 HDFS 檔案系統上的資料，所以，執行效率依指令複雜度和處理的資料量而異，可能有數秒鐘，甚至是數分鐘的延遲。和 HBase 相比，Hive 容易使用且彈性高，但執行速度較慢。不少資料庫系統，都是透過先連結到 Hive，才能與 Hadoop 整合。例如微軟就是透過 Hive ODBC 驅動程式，將 SQL 指令轉換成 Hive QL，讓 Excel 可以存取 Hadoop 上的資料。

在同一個 Hadoop 叢集中，Hive 可以存取 HBase 上的資料，將 HBase 上的資料對應成 Hive 內的一個表格。 如圖 2-4 所示:

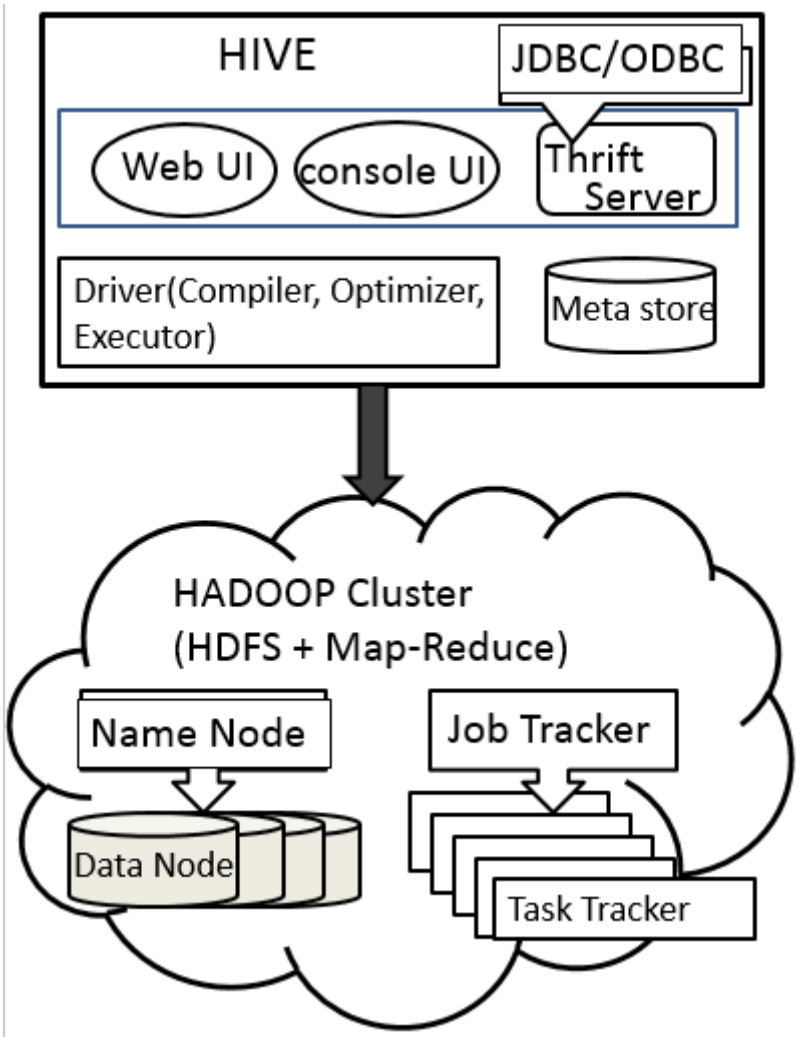


圖 2-4 HIVE 與 HADOOP Cluster

2.4 基因演算法

基因演算法[15]於初始階段隨機產生規模為 N 的群體稱為第 0 代染色體(chromosome)，然後去評估每一個染色體得到適應值(fitness value)。接著根據既定的挑選機制(selection mechanism)來複製個體進行遺傳演化。根據交配機率 Pc （probability of crossover）決定是否要進行交配產生子代，產生後的子代再根據突變機率 Pm（probability of mutation）決定是否要做進一步的突變。交配是主要的遺傳運算，而突變只是次要的動作。產生後的子代或原來的雙親 (沒有進行交配或突變者)共同組成下一世代 (仍然是 N 個)，如此繼

續進行直到滿足終止條件為止。接著要決定每一代究竟有多少個體要進行演化，每一代的個體數目就是群體規模 N。群體進行演化時就相當於同時做 N 組平行找尋，N 太小則演化速度緩慢，也容易陷入區域最佳解；N 太大則計算數量龐大，會耗費較長的計算時間。

事實上，到目前為止 N 值的決定沒有明確的定論，典型的是從 30 到 300 都有，通常要看問題的本質以及執行效率或執行時間等因素，經過多次實驗後才決定之。交配是主要的遺傳運算子，將兩個染色體經過合併的程序來產生子代，讓子代各含有雙親的部分特性。交配的目的是希望子代能夠組合出含有更高適應度的染色體，然而有可能子代只遺傳雙親的缺點，所以交配不保證一定可以製造出更好的子代，不過有了天擇的機制，適者生存，較差(適應度較差)的子代會逐漸被淘汰，而優良(適應度較佳)的子代可以繼續繁衍下去。以下 4 種常用的交配方式：(1)單點、(2)兩點、(3)多點、以及(4)均一交配。許多研究顯示兩點交配較優於單點交配，只有當群體接近收斂時，兩點交配的效果才會遜色於單點交配。突變會導致染色體突然間做隨意的變化，常見的變化是改變染色體上的一個基因。突變的作用會引導基因演算法進入未曾尋找過的基因架構[26][27]，將新基因導入群體中。但是突變次數過多將會導致基因演算法變成隨機演算法，會任意破壞原來的結構，造成子代與親代喪失若干相似的特徵；而過少的突變則會造成有用的基因沒被發覺，可能會陷入區域的最佳解。圖 2-5 為基因演算法之流程圖:

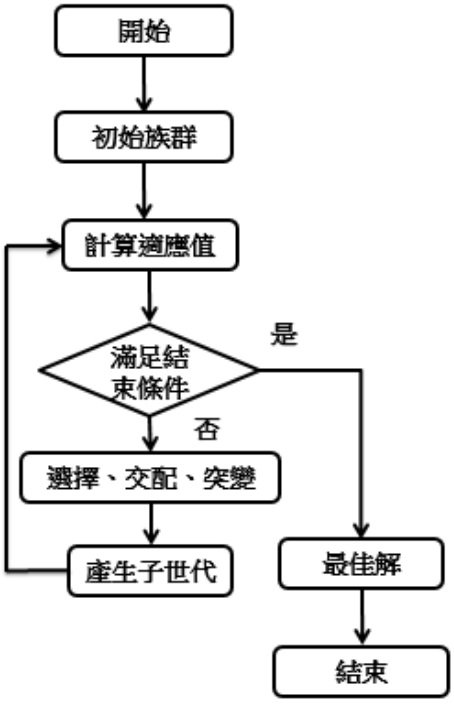


圖 2-5 基因演算法流程

2.5 地圖圖層概念

數值地形圖通常以圖層（layer）模式管理不同主題之資料，網格式資料是指由點或網格所組成的資料，網格式資料的儲存方式則是透過一個個的格子（grid）來進行記錄，而向量式資料則是由點、線、面三種基本元素所組成的資料,經由不同的點可以形成線,而不同的線則可以圍出一塊一塊的面，然後構成了地表的空間資訊。

地理資料的度量上主要分為類別（nominal）、級序（ordinal）、間距（interval）、比例（ratio）等四種資料等級，資料有大小位階，就如同直轄市和縣轄市的位階不同，高速公路與一般平面道路的區別，而彼此的差異是可以度量的，且具有測量的原點，所以可以進行加減乘除等處理。利用圖層切割的觀念與雲端之可用性與部分路徑可再利用性之特性，即計算過之路徑已存在雲端環境中，若有其他使用者還需計算此相同路徑時，即可從雲端環境中立即的使用已計算過之結果，實現”部分路徑可再利用性”，再配合圖層

的分級概念，以達到節省時間並增進導航效能目的。圖 2-6 為地圖圖層概念表示圖:

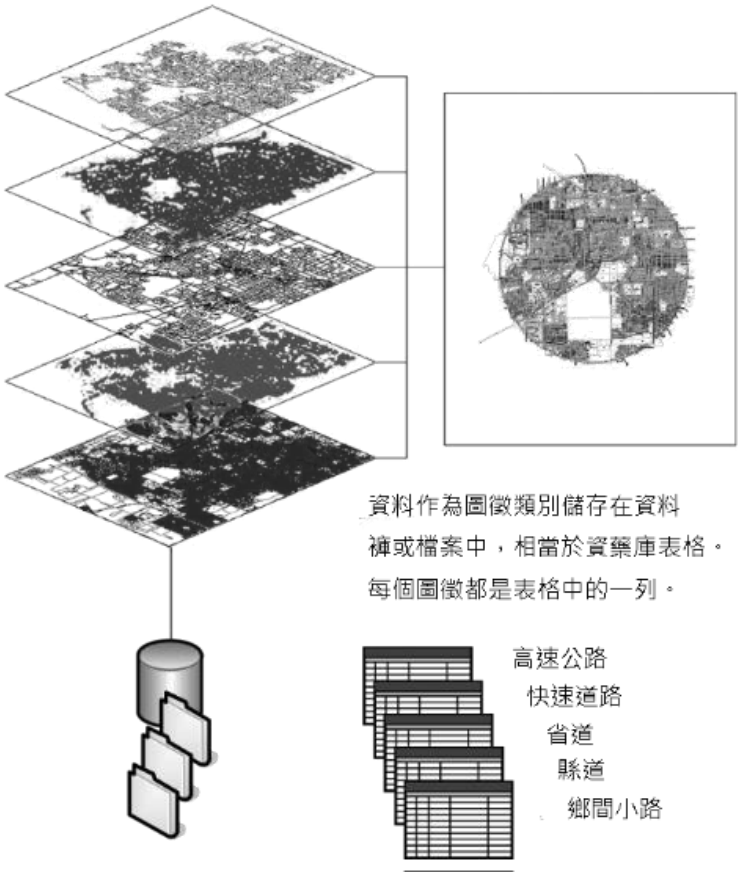


圖 2-6 地圖圖層概念圖

2.6 Google Earth

Google 地球(Google Earth)[31]是一款 Google 公司開發的虛擬地球儀軟體，它把衛星照片、航空照相和 GIS 佈置在一個地球的三維模型上。Google Earth 使用了公共領域的圖片、受許可的航空照相圖片、KeyHole 間諜衛星的圖片和很多其他衛星所拍攝的城鎮照片。甚至連 Google Maps 沒有提供的圖片都有。分為免費版與專業版兩種。

2.7 最短路徑問題

最短路徑問題是圖論研究中的一個經典演算法問題，用於尋找圖（由結點和路徑組成的）中兩結點之間的最短路徑。兩點間最短的距離為直線，但是如果兩點之間並無直接路徑相連，而需要依靠第三點方能到達，最短路徑的問題也就產生了，其演算法具體的形式包括：

(1)確定起點的最短路徑問題 - 即已知起始結點，求最短路徑的問題。適合用 Dijkstra's 演算法。關於使用 Dijkstra's 演算法的時間複雜度:

Dijkstra's 演算法要考慮到找出最小 d 值步驟時所用的資料結構。

如果使用 binary min heap，extract-min 要用 O(log V)，而 decrease-key 要用 O(log V) ，所以總共用 O((V+E)log V)

如果以 Fibonacci heap 實作可以達到 O(VlogV + E) ，|V|為 Vertices 的個數，|E|為 Edges 的個數。

(2)確定終點的最短路徑問題 - 與確定起點的問題相反，該問題是已知終結結點，求最短路徑的問題。在無向圖中該問題與確定起點的問題完全等同，在有向圖中該問題等同於把所有路徑方向反轉的確定起點的問題。

(3)確定起點終點的最短路徑問題 - 即已知起點和終點，求兩結點之間的最短路徑。

(4)全域最短路徑問題 - 求圖中所有的最短路徑。適合使用 Floyd-Warshall 演算法。

最短途徑是兩點之間權重最小的途徑。最短途徑不見得是邊最少、點最少的

途徑。最短途徑也可能不存在。兩點之間不連通、不存在途徑的時候，就沒有最短途徑。

2.8 Quantum GIS

Quantum GIS 由 Gary Sherman 於 2002 年開始開發，並於 2004 年成為開源地理空間基金會的一個孵化項目。版本 1.0 於 2009 年 1 月發布。Quantum GIS 以 C++寫成，它的 GUI 使用了 Qt 庫。Quantum GIS 允許集成使用 C++ 或 Python 寫成的外掛程式。除了 Qt 之外，Quantum GIS 需要的依賴還包括 GEOS 和 SQLite。同時也推薦安裝 GDAL、GRASS GIS、PostGIS 和 PostgreSQL。

Quantum GIS 是一個多平台的應用，可以在多種作業系統上運行，包括 Mac OS X、Linux、UNIX 和 Microsoft Windows。對於 Mac 用戶，Quantum GIS 相對於 GRASS GIS 的優勢在於它不需要 X11 窗口系統，而且界面更簡潔、快速。Quantum GIS 也可以作為 GRASS 的圖形用戶界面使用。相較於商業 GIS，Quantum GIS 的文件體積更小，需要的記憶體和 CPU 處理能力也更少。因此它可以在舊的硬體上或 CPU 運算能力被限制的環境下運行。

Quantum GIS 被一個活躍的志願者開發團體持續維護著，他們定期發布更新和錯誤修正。現在，開發者們已經將 Quantum GIS 翻譯為 31 種語言，它被使用在全世界的學術和專業環境中。圖 2-7 為運行 Quantum GIS 時的截圖:

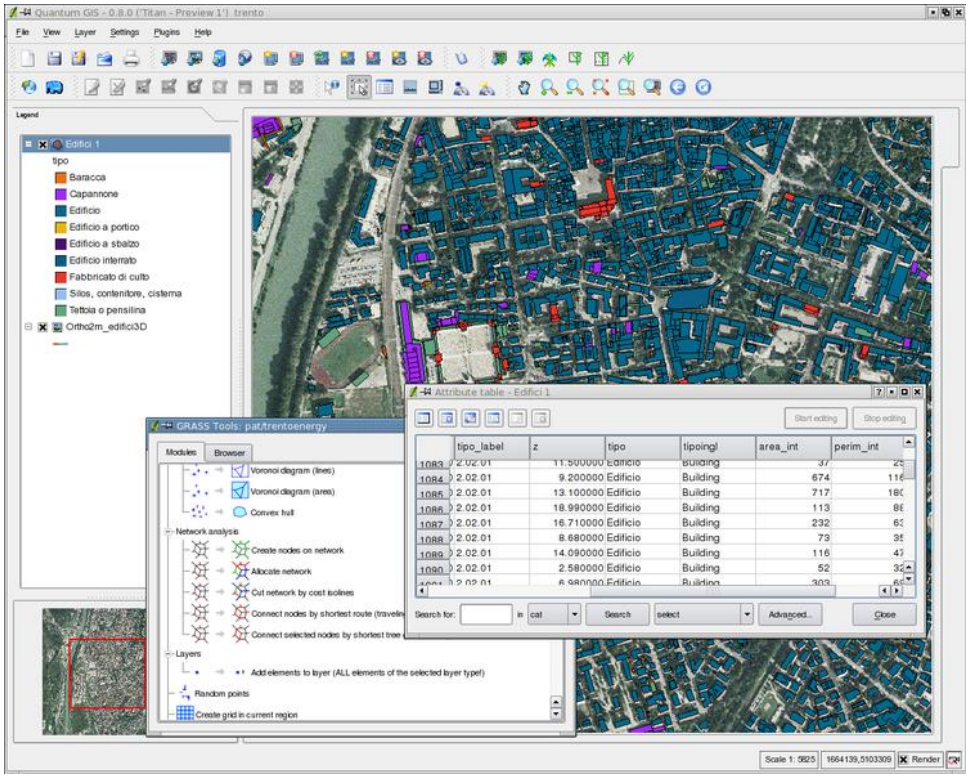


圖 2-7 Quantum GIS

2.9 Quantum Navigator

Quantum Navigator 是基於 Quantum GIS 應用程式的功能，發展出一套具有地圖導航的客製化工具。

此軟體的目標就是讓道路圖上具有基本繪製路線與導航的功能。只要輸入一個正確的道路 shapefile，你就可以選擇路線的起迄點。這個路線會根據你的需求計算適當的路徑(最短、最快或最經濟的路徑)，也會遵守單行道、轉向規則等一些限制。

此應用程式的核心是由 C++撰寫而成。道路圖呈現的是個直觀的圖形，最佳化路徑則以 Dijkstra's algorithm 為運算方法搜尋。使用者介面以 Python 寫成，有地圖瀏覽視窗、增加地圖、道路標誌或選擇起迄點等功能。

主要功能說明：

使用介面為地圖視窗、側邊控制資訊工具選項。地圖操作為全覽地圖；基本工具：移動、縮放；經緯度顯示。查詢路線圖為用於查詢道路詮釋資料及標識資料，並在另一視窗標示出最近道路的屬性資料。其他還有道路起迄點標示、前後路段標示。建立路線為可以設定起迄點，並自動計算路徑。提供兩種決定終點的方法：一為使用者在地圖上指定，程式會自動搜尋最鄰近的道路。路線類型為三種可能的路線類型：最短、最快、最經濟。GPS 模擬器為透過增加 waypoint 來模擬，可顯示目前的位置、GPS 時間與對應道路。當程式達到生產階段時，將對真正的 gps 支援(NMEA or gpsd)。導航為設定導航後，會告訴你何時該轉彎，以及距離下一個轉彎或目的多遠的距離。模擬路徑為檢視導航的路徑。圖 2-8 為 Quantum Navigator 之介面:

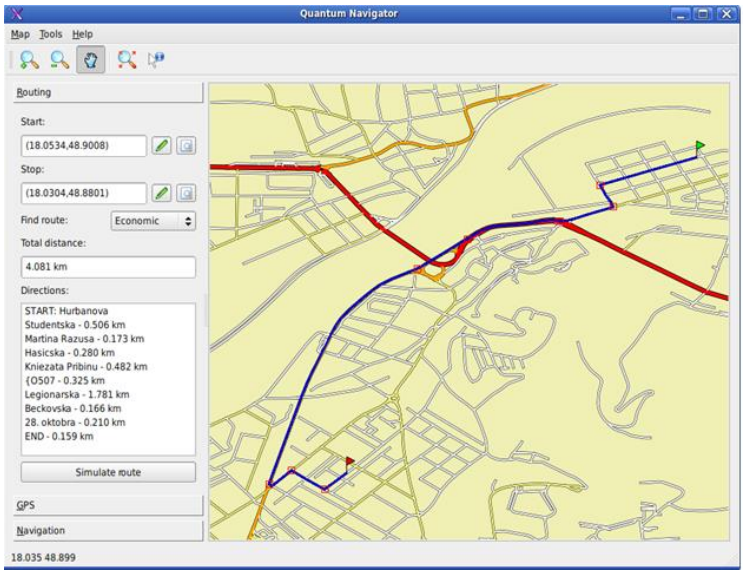


圖 2-8 Quantum Navigator 介面

CHAPTER 3 最短行車時間問題

3.1 最短行車時間問題數學模型

最短行車時間時間（SDT）的問題可被配製成一個數學模型。其數學模型的符號列表，如表 3-1 所示：

表 3-1 最短行車時間時間（SDT）數學符號列表

Problem parameters:	
N	set of all nodes
A	set of all links
S	source node
D	Destination node
i,j	index of node, i,j
<i,j>	node i to node j, directional
E_{ij}	link node i to node j
d_{ij}	distance of node i to node j
v_{ij}	velocity of node i to node j
T_{ij}	cost time of node i to node j
U_{ij}	binary, 1 if the link from node i to node j exists in the routing path, 0 otherwise
t	total drive time

v_{ij} 為行車之變動速率,即 node i and node j 的最高行車速率；而 node i and node j 的距離為 d_{ij}，則每段路的最短行車時間為 $T_{ij} = d_{ij} / v_{ij}$. 然而我們依照公式即可求得最短行車時間 t ，如下列公式(3.1):

$$\text{Minimize } t = \sum_{i=S}^{D}\sum_{j=S}^{D} T_{ij}U_{ij} \text{ subject to}$$

$$\sum_{\substack{j=S \\ j\neq i}}^{D} U_{ij} - \sum_{\substack{j=S \\ j\neq i}}^{D} U_{ji} = \begin{cases} 1, & \text{if } i = S \\ -1, & \text{if } i = D \\ 0 & \text{otherwise} \end{cases} \quad \text{and} \qquad (3.1)$$

$$\sum_{\substack{j=S \\ j\neq i}}^{D} U_{ij} \begin{cases} \leq 1, & \text{if } i \neq D \\ = 0, & \text{if } i = D \end{cases}$$

$$\text{Where } U_{ij} \in \{0,1\}, \text{ for all } i \text{ and } T_{ij} = d_{ij} / v_{ij}$$

3.2 使用雲端技術結合基因演算法解決最短行車時間問題

在傳統的最短路徑問題上，往往只考慮跨越二點所需的代價，稱之為 costs，如無特殊意義時，在道路上則為 node 間距離。假設行車時為固定速率 V，則起點至終點的 Total Time=(Total Distance/Velocity)，所以求得最佳行車時間(T)即為最短距離(D)除以行車的速率(V)。

假設行車時為變動速率 V_{ij}，即 node i 至 node j 的最高行車速率，而 node i 至 node j 的距離為 d_{ij}，則每段最佳行車時間 $T_{ij}=d_{ij}/v_{ij}$，此問題依樣可以使用 Dijlstra's Algorithm 來解，需要將本來的 costs 代表的距離改成花費的時間即可，同時基因演算法也可以解決相同的問題，即求解 minimize($T_{ij}xU_{ij}$)。

以上情況為基本的考量行車速率下的結果，但實際行車狀況有可能因為改變交通工具或道路在不同時段有不同時速的限制而產生變化，但皆可使用基因演算法求解出最佳解。

以下為使用我們所改良過的基因演算法為基礎，用以下的步驟來解決最佳行

車時間問題。

(1) Genetic Representation:使用變動長度的染色體表示，但長度最多為 N。染色體的起點為 S，終點為 D，例如有一路徑(S-N_1-N_2-……-N_{K-1}-N_K-D) 。如圖 3-1 所示:

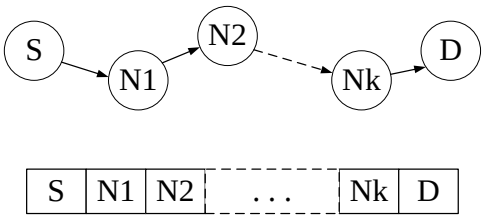


圖 3-1 染色體表示路徑

(2)Population Initialization:產生第一代的染色體，需依照母群體的大小來決定要使用多少個染色體來代表初始狀態。由於一個染色體即代表一個問題解，因此初代族群也意謂著初始解的集合。

一個族群應該要包含多少個染色體呢?一般來說,越複雜的問題需要越大的族群規模來解決，是因為族群的染色體越多，代表參與搜尋的個體越多，有更多的機會得到較佳的搜尋結果!但同時也表示會佔用較大的系統資源，也會花費較長的搜尋時間!若族群越小，則有可能造成提早收斂，或是忽略最佳解的情況產生。

應用平行運算技術實現基因演算法，主要的特色是將整體的族群資料，分割成許多的子族群，再分別對每一個子族群進行基因演算，待子族群持續演化，並滿足結束條件時，再經過族群間的交配，產生新的族群，並再持續演化，以得到最佳解。所以我們朝著解決花系統資源，花時間去搜尋的初始族群來進行 MapReduce 平行運算，以達到雲端運算的最大效益。藉由 Map 平行運算後各個節點得到部分最佳解後，接著 Reduce 運算的處理方式有兩種策略，策略一為結合所有 Map 基因運算的部分最佳解，得到完整的整體最佳解，並在整個雲端基因運算的最外面加上一層判斷式以模擬平行基因演算法的回饋機制，也就是將 Reduce 運算後的結果(output)當作下一次運算

的輸入資料(input)再次進行基因演算法。策略二為比對所有 Map 基因運算的運算結果，並挑選部分最佳解其中之一，當作演算的全體最佳解，與策略一相比的話就是無回饋機制。我們將根據兩種不同的策略架構來進行實驗。圖 3-2 為基因演算法平行部位示意圖:

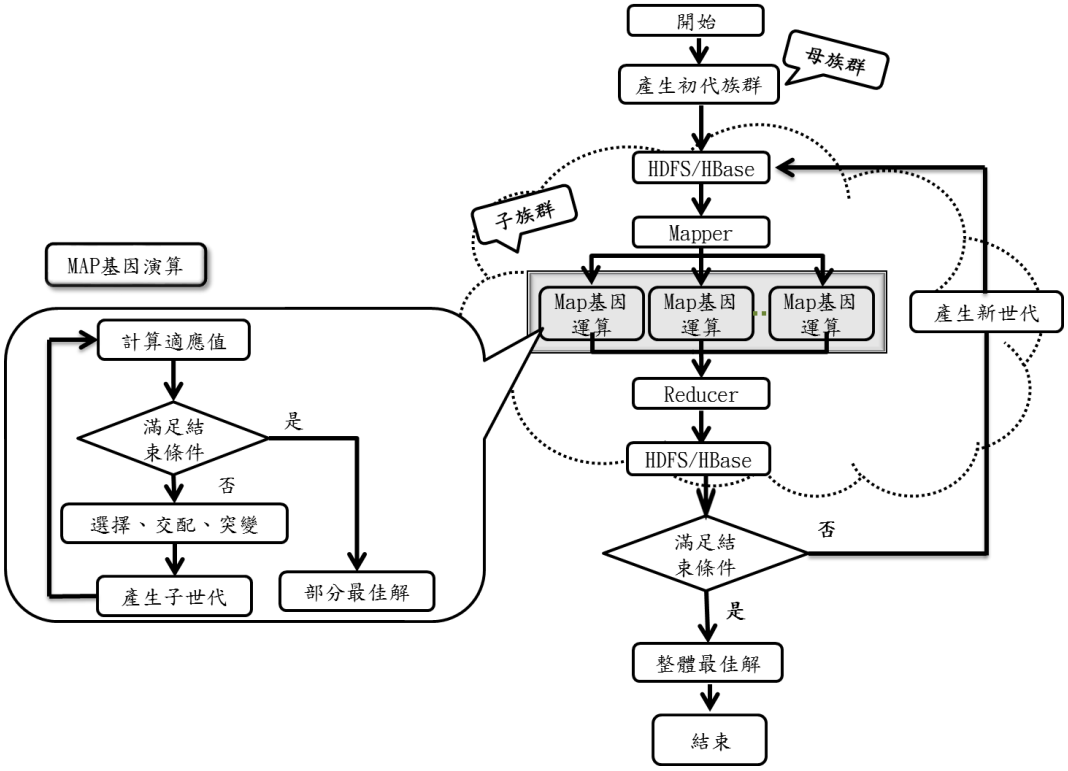


圖 3-2 基因演算法平行部位示意圖

(3)適應函數(Fitness Function):適應函數的選擇很明確的定義為:

$$f = \frac{1}{\sum(d_{ij}/v_{ij})}$$

f 表示為染色體的適應值(fitness value)，d_{ij} 為 node i to node j 的距離，v_{ij} 為 node i to node j 的速率。

(4)Selection:使用 tournament(競爭式選擇法)來進行，將會選擇較好的二個染色體進行交配，然而相同的染色體不應該被使用兩次以上。

(5)Crossover:因 Genetic Representation 是可變動長度的染色體，交配的方式為在二個染色體中找出可以 crossover 的點後，再隨機選擇一點做 crossover point。因其交配後與突變完成後，在染色體上面的表示式可能存在迴圈(loop) ，此時可以進行修復，把這些重複的資訊去掉，在計算適應值的時候才會正確。如圖 3-3 所示:

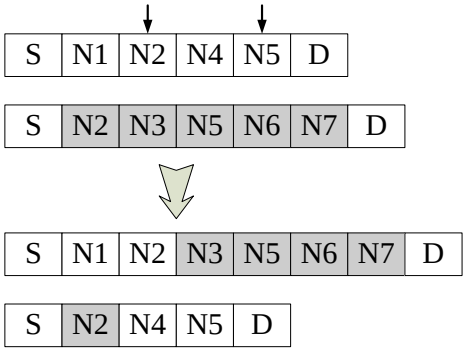


圖 3-3 基因演算法交配階段

(6)Mutation:使用 6-10%的突變機率做為產生新路徑的方法。在原染色體中的其中一點突變出去隨機找尋一條新的路徑到 D 的點來取代原先的路徑，太低的數值不容易變化，很難收斂最佳解，太高的數值又會造成收斂困難，不過設定多少要依問題的情況而定。如圖 3-4 所示:

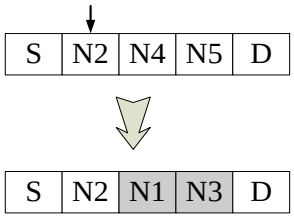


圖 3-4 基因演算法突變階段

3.3 群體初始化問題

從全域解的空間中我們隨機選擇的路徑是由基因演算法中的群體初始化所決定的。在實際場合中，用於車輛導航找到最短的行車時間，主要的挑戰是要從出發地到目的地中找到一條可行的路徑，其地圖節點可能超過 10000 個節點。由於每個節點連接到許多其他節點，有些節點間的鏈接是有方向性的，只能行駛於固定的方向。因此，在一個真正的行車時間計算應用於基因演算法上，初始化染色體是非常複雜的問題。應用平行運算技術實現基因演算法，主要的特色是將整體的族群資料，分割成許多的子族群，再分別對每一個子族群進行基因演算，待子族群持續演化，並滿足結束條件時，再經過族群間的交配，產生新的族群，並再持續演化，以得到最佳解。

為了找到一個巨大的車載導航地圖節點初始群體，有一個簡單的辦法就是從出發點下手，選擇任何可用的路徑連接到中間節點，直到到達目的地。從群體物件之圖形資料推導得物件之空關係如下幾點特性，連接性是路段之連接情形。方向性是路徑方向。相鄰性是區域之相鄰情形。包含性是某點位或線段是否在某區域內等。然而從原圖形資料演化出新圖形如環域(buffer)為將原圖形等距擴張成一區域。最佳路徑為從網路中依某種性質找出最佳路徑。選址為依某些條件選出最佳位置。將同區域的資料分成不同的類型或層級儲存，例如依不同地類、主題、年代或樓層等，各儲存類別稱為”圖層” 。傳統紙圖常依不同的主題，如人口分佈圖、地質圖、地形圖等，來表現不同的人文活動或是地表現象，這些圖稱為主題圖(thematic map)。而目前許多 GIS 數值圖則常以資料項目分層，稱為資料層(data Layer)，但也常被稱為圖層或主題。如圖 3-5 所示:

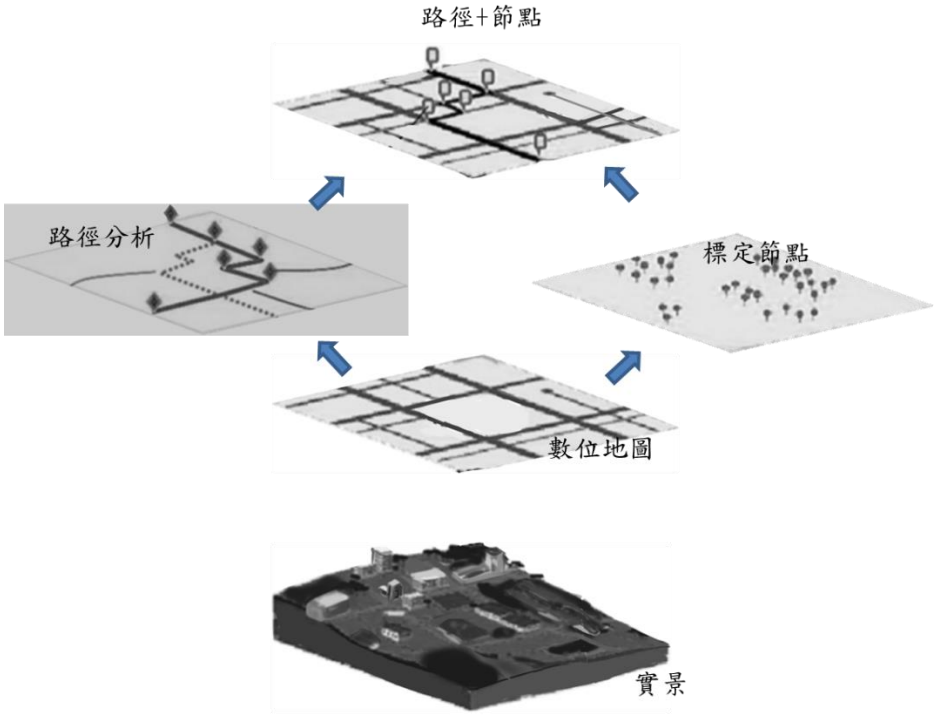


圖 3-5 圖層運算概念

CHAPTER 4 實驗結果與分析

4.1 實驗環境

實驗的環境建置為使用五台電腦來架設 Hadoop 叢集，在每一台電腦上安裝了 Oracle VM VirtualBox，在每一台 VM 上虛擬一部機器，所以一共有五部實體電腦上生成五部虛擬機[7]，一台當 Namenode，而其它 4 部虛擬機上各有 2 個 Worker 節點，所以總共有 8 個 Worker 節點如圖 4-1 所示。Hadoop 實驗叢集架構圖分布，如圖 4-2 所示:

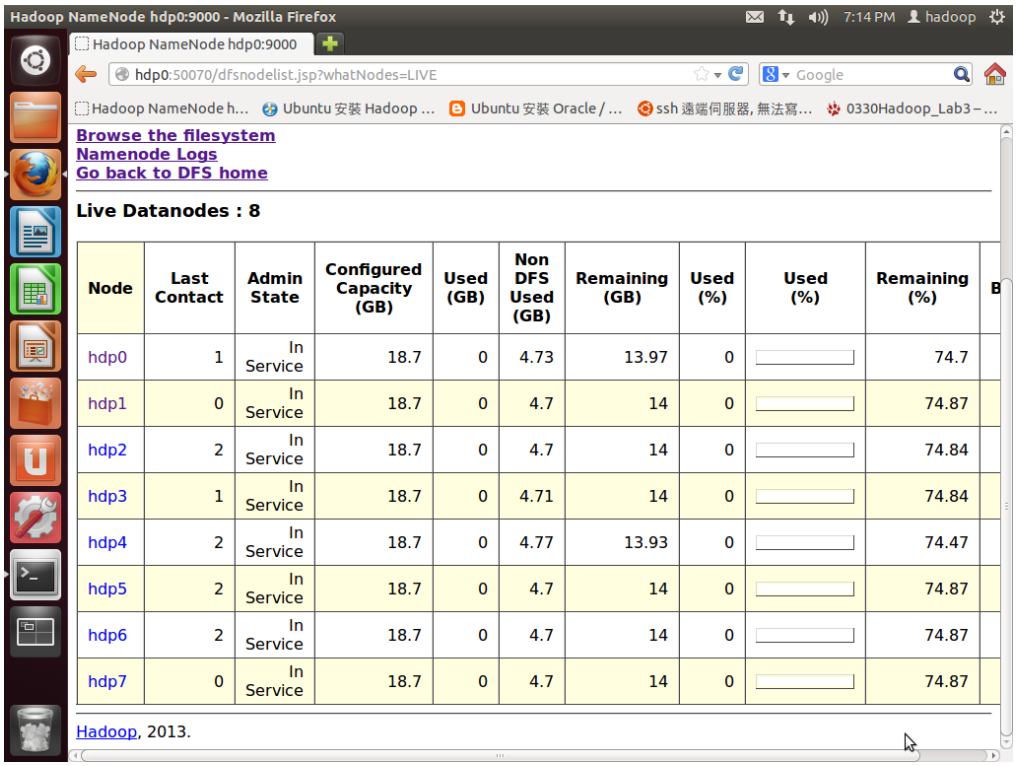


圖 4-1 實驗用之 hadoop 叢集

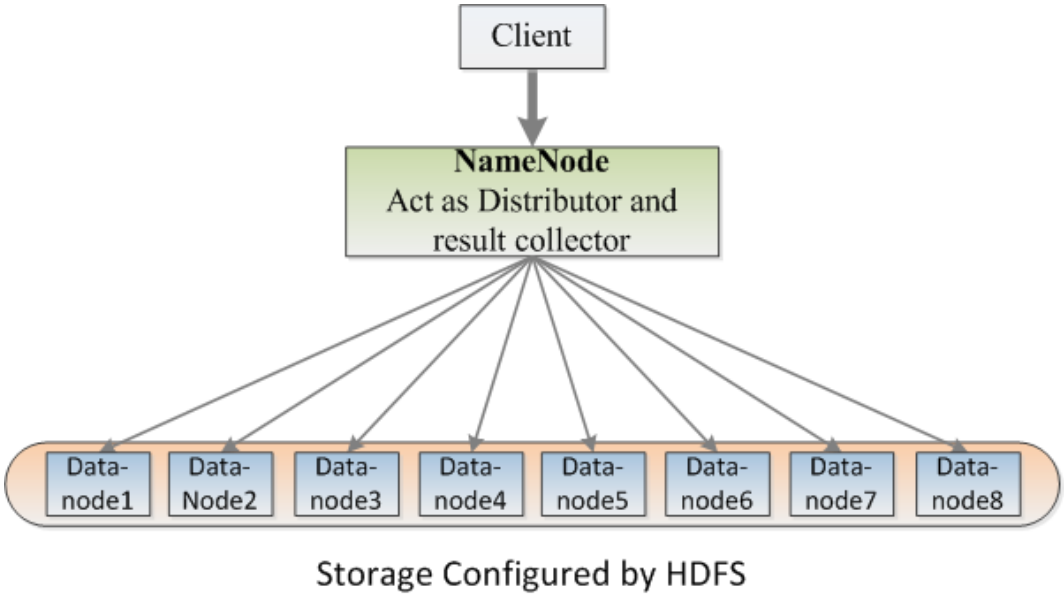


圖 4-2 Hadoop 實驗叢集架構圖

而 HDFS 為 Hadoop 的分散式檔案系統，把一個檔案存入 HDFS 時，HDFS (有時稱 Namenode)會把檔案切割成固定大小的 block，而後將各 block 分散儲存到不同的 Datanodes 上，由於每個檔案的儲存都是跨實體機器的，因此 HDFS 可視為一個虛擬的分散式檔案系統 (傳統的檔案系統一樣會將檔案切割為 block，但都儲存到同一台實體機器的硬碟上)，或者說是一個 Logical File System，而 Namenode 就負責扮演 Linux file system 中 inode 的角色，要知道組成某個檔案的所有 block 被儲存在哪些 Datanodes? 問 Namenode 就對了。Nmaenode 的網頁監控資訊如圖 4-3 所示:

為了盡可能的提昇 HDFS 的存取效能(特別是讀取速度)，HDFS 在儲存資料時必須將資料根據機率平均的分佈在組成 Hadoop cluster 的成員硬碟上(balancer 的工作)。圖 4-4 為 Nmaenode 的網頁監控資訊與實驗之硬體設備:

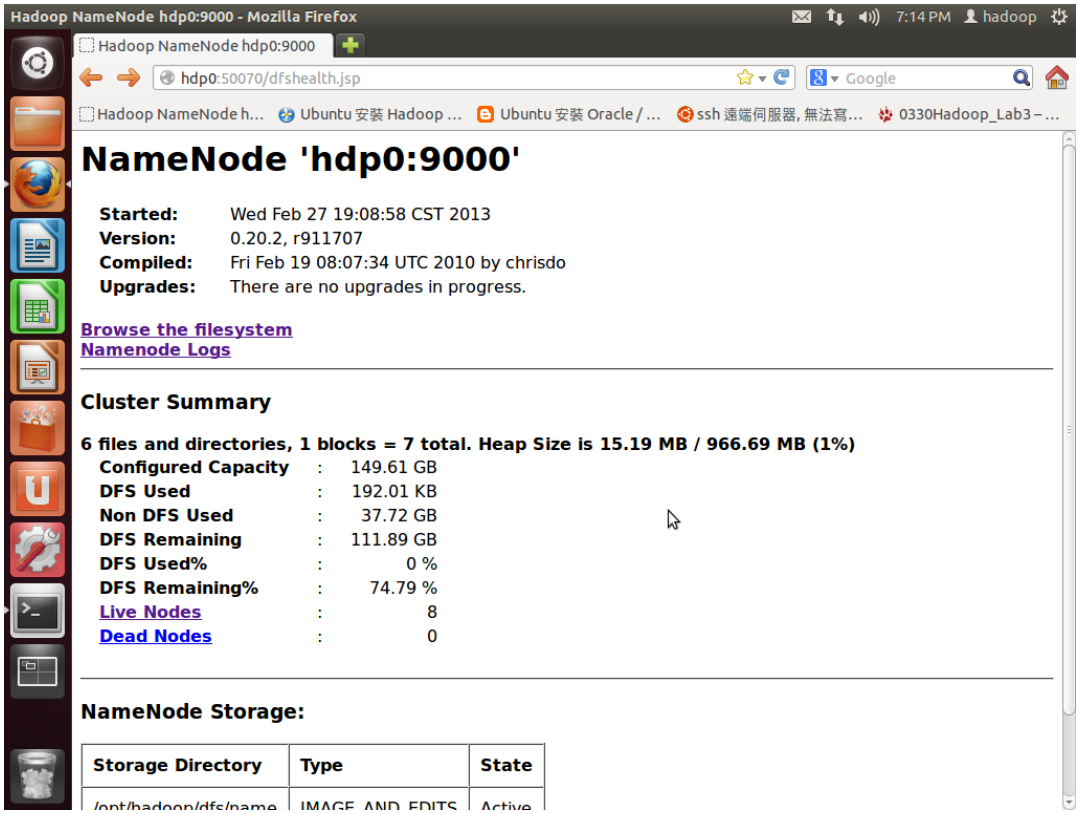


圖 4-3 Nmaenode 的網頁監控資訊

表 4-1 實驗的硬體設備

Node	Processor	Memory	OS	JAVA
Node 1	Intel(R) Core(TM) i5 CPU650 x 2	1G	ubuntu-12.10	jre 1.6.0 _24
Node 2	Intel(R) Core(TM) i5 CPU650 x 2	1G	ubuntu-12.10	jre 1.6.0 _24
Node 3	AMD Phenom II x4 945 3.0Ghz	1G	ubuntu-12.10	jre 1.6.0 _24
Node 4	AMD Phenom II x4 945 3.0GHz	1G	ubuntu-12.10	jre 1.6.0 _24
Node 5	AMD Phenom II x6 1065t 2.9GHz	1G	ubuntu-12.10	jre 1.6.0 _24
Node 6	AMD Phenom II x6 1065t 2.9GHz	1G	ubuntu-12.10	jre 1.6.0 _24
Node 7	Intel(R) Core(TM) i5 CPU750 x 2	1G	ubuntu-12.10	jre 1.6.0 _24
Node 8	Intel(R) Core(TM) i5 CPU750 x 2	1G	ubuntu-12.10	jre 1.6.0 _24

4.2 雲端系統伺服端演算法

以下為雲端系統伺服端演算法:

主程式-呼叫雲端物件部分

雲端運算-步驟 1.建立運算資料

雲端運算-步驟 2.複製運算資料到 HDFS

雲端運算-步驟 3.執行運算

3.1 基因演算-步驟 1.實作運算主程序

3.2 基因演算-步驟 2.設計適應函數與編碼

3.3 基因演算-步驟 3.實作輪盤法

3.4 基因演算-步驟 4.實作複製運算

3.5 基因演算-步驟 5.實作交配運算

3.6 基因演算-步驟 6.實作突變運算

3.7 基因演算-步驟 7.實作演化迭代

雲端運算-步驟 4.取得運算結果

主程式-步驟 4.系統測試與數據分析

由交通部運輸研究所的台灣地區交通路網數值地圖利用 Google earth 所呈現之台灣本島縣市界圖層、臺灣本島市鄉鎮區界圖層、臺灣本島村里參考界圖層，而台灣道路節點如表 4-2 所示:

表 4-2 台灣道路節點

項　目	2000 年版	前版	備註
圖層數	道路(含節點)、水系、鐵路、行政界、地標地物共 5 層	只有道路(含節點)1 層	
涵蓋範圍	台灣本島、澎湖本島	台灣本島	
道路筆數	23 萬 3 千餘筆	14 萬 7 千餘筆	
具道路名稱之道路筆數	15 萬 9 千餘筆	2 萬 3 千餘筆	其餘為"其他道路"
道路節點數	15 萬 6 千餘點	10 萬 8 千餘點	

HBase 的資料表格(table)是由列(row)與行(column)所組合成的類似二維陣列，我們將 QGIS 圖層資料庫儲存格式轉換為 HBase 儲存格示。包含兩個行家族(column family) ，分別為地址(address)和地址詳細資料，最前面第一列為資料的主鍵值(row key) ，在此處我們命名為 ID，HBase 的資料表格是由主鍵值(row key)的順序下去做排序的;而左邊一列為時間戳記(timestamp) ，Hbase 每當一筆新儲存資料進來時，皆會給予每一筆資料一個時間戳記，如此一來，因為儲存的時間不同，也能分辨出就算在列與行相同的情況下，不同的資料。

配合著 HBase 的儲存資料，我們可以將其運算中及運算後的資料，於 HBase 儲存及提取再使用，以達到我們設計之基因演算法回饋機制，達到我們的路徑可再利用之特性目的。圖 4-4 為 QGIS 圖層資料庫儲存格式:

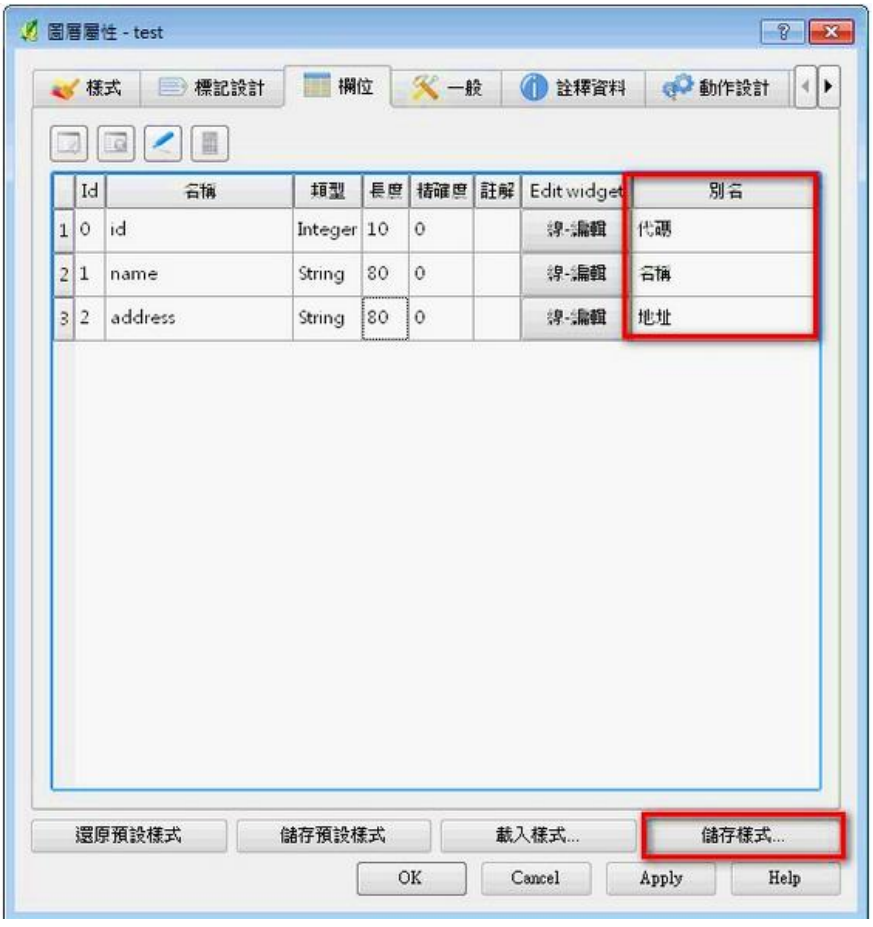


圖 4-4 QGIS 圖層資料庫儲存格式

4.3 使用節點模擬以計算平均尋找最佳路徑所需要的演化代數

我們使用基因演算法模擬節點(square matrix map)來做實驗。

這部分的地圖為使用 8×8、16×16、32×32、64×64、128×128、256×256 的方陣地圖，各有 64、256、1024、4096、16384、65536 個節點，假設最左上角為起點，最右下角為終點，每個節點與每個節點的距離假設為 250 公尺，設定一般道路行駛速率為 30-80，隨機分佈行駛速率於地圖上，如圖 4-5 所表示 4×4 square matrix map。而基因演算法所用的交配機率為 90%，突變機率為 8%，突變率太低會造成數值不容易變化，很難收斂最佳解。太高的話，會造成很難收斂最佳解。而交配率太高的話表示優良物種被取走的速度快過產生新物種的速度，則會造成求解的原意喪失，交配率太低的話會導致搜尋過程停滯而最終無法找到全域最佳解。我們設定最多演化 60 代，每一份實驗分為單次或做 100 次取平均數

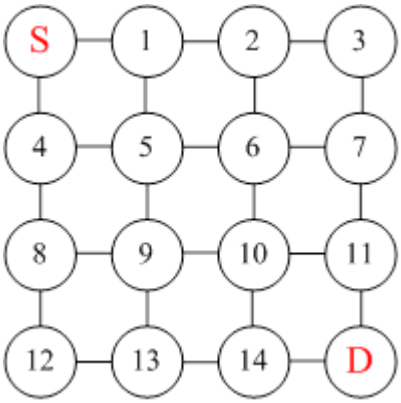


圖 4-5 4×4 square matrix map

我們應用雲端運算技術(MapReduce)方法於觀察群體初始化與在不同的染色體演化代數與地圖節點數目時，其收斂的狀況為何?以模擬地圖節點（64、256、1024、4096、16384、65536）在地圖上的每個節點不超過四個鏈接到其

他節點。在實驗中，我們研究了染色體演化代數和找尋最短行車時間誤差率，其公式為:

$$找尋最短行車時間誤差率 = \frac{染色體演化代數}{地圖節點數} * 100$$

表 4-3 中橫欄表示地圖節點數目，縱欄表示染色體數目，表格中的數值代表未使用雲端架構的基因演算法達到最佳行車時間所需要的演化代數，由表中顯示，65536 個節點，收斂所需的演化代數為 60 代內。圖 4-3 為未使用雲端架構之平均尋找最佳路徑所需要的演化代數圖:

表 4-3 平均尋找最佳路徑所需要的演化代數

染色體數目 \ 地圖節點數目	64	256	1024	4096	16384	65536
40	9	12	21	40	51	55
80	11	12	23	42	51	55
160	16	16	23	44	52	56
320	17	19	24	44	53	56
640	18	20	24	45	53	57
1280	20	24	26	45	54	58
2560	21	25	26	46	55	59

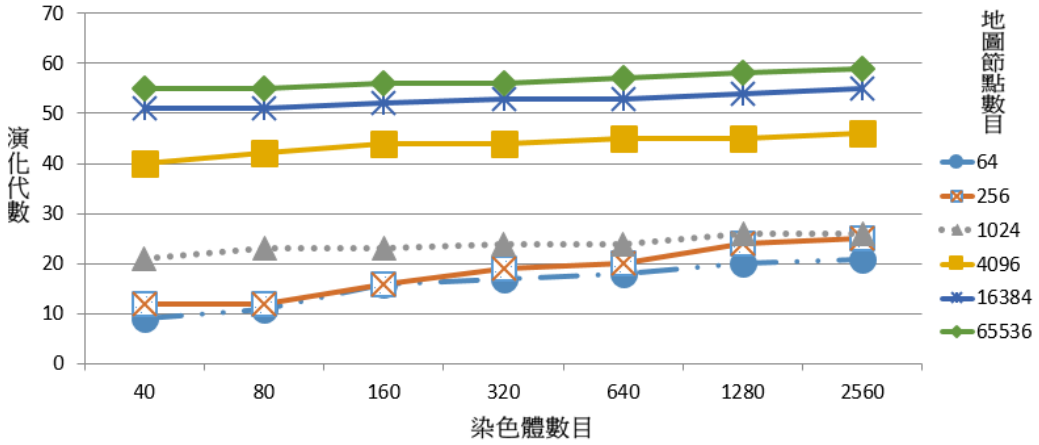


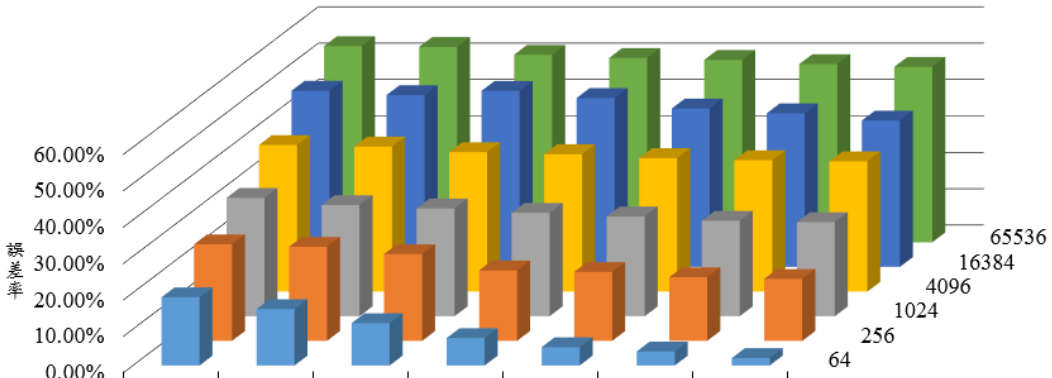
圖 4-6 未使用雲端架構之平均尋找最佳路徑所需要的演化代數

表中得知，最多演化於 60 世代內達到最短行車時間的誤差機率。我們可以發現，隨著染色體數目的增加，可以使得到最短行車時間的誤差機率降低，相對而言，所需要收斂代數也會增加，當然運算的時間也會增加。如表 4-4 所示:

表 4-4 未使用雲端架構的基因演算法達到最佳行車時間誤差百分比

染色體數目 \ 地圖節點數目	64	256	1024	4096	16384	65536
40	18.74%	26.52%	32.53%	40.29%	48.43%	53.94%
80	15.49%	25.83%	30.53%	39.86%	47.23%	53.69%
160	11.63%	23.85%	29.63%	38.32%	48.42%	51.53%
320	7.53%	19.34%	28.40%	37.78%	46.43%	50.68%
640	5.03%	18.93%	27.43%	36.69%	43.53%	50.13%
1280	3.85%	17.49%	26.23%	36.11%	42.23%	48.98%
2560	2.04%	17.03%	25.85%	35.79%	40.23%	48.24%

如圖 4-7 所示，我們發現在較小地圖中，如 64 個節點和 256 個節點時，可以取得較接近最短行車時間的最佳解，誤差值可以接近至 17%；但對於較大地圖而言，如 16384 個節點和 65536 個節點，其誤差值卻達到近 48%，增加了近 3 倍之多!因為受限於基因演算法上的局部限制(local optimal)，原因為沒有探索到全域解的空間而造成的結果誤差。



地圖節點數目 \ 染色體數目	40	80	160	320	640	1280	2560
64	18.74%	15.49%	11.63%	7.53%	5.03%	3.85%	2.04%
256	26.52%	25.83%	23.85%	19.34%	18.93%	17.49%	17.03%
1024	32.53%	30.53%	29.63%	28.40%	27.43%	26.23%	25.85%
4096	40.29%	39.86%	38.32%	37.78%	36.69%	36.11%	35.79%
16384	48.43%	47.23%	48.42%	46.43%	43.53%	42.23%	40.23%
65536	53.94%	53.69%	51.53%	50.68%	50.13%	48.98%	48.24%

圖 4-7 未使用雲端架構的基因演算法得到最短行車時間的誤差機率

接著，我們使用雲端架構結合調整過的基因演算法來做實驗。

表 4-5 中橫欄表示地圖節點數目，縱欄表示染色體數目，表格中的數值代表使用雲端架構的基因演算法達到最佳行車時間所需要的演化代數，由表中顯示，基因演算法的收斂速度相當快，即使有 65536 個節點，在染色體為 2560 條時，收斂所需的演化代數為 40 代。與沒有使用雲端架構相比，達到最佳行車時間所需的演化代數明顯變少。如圖 4-8 所示:

表 4-5 使用雲端架構的基因演算法達到最佳行車時間所需要的演化代數

染色體數目 \ 地圖節點數目	64	256	1024	4096	16384	65536
40	13	18	19	20	22	23
80	15	18	20	21	24	25
160	19	23	24	25	26	27
320	24	25	26	31	33	35
640	27	27	28	33	35	37
1280	28	29	30	34	36	38
2560	29	30	32	35	37	40

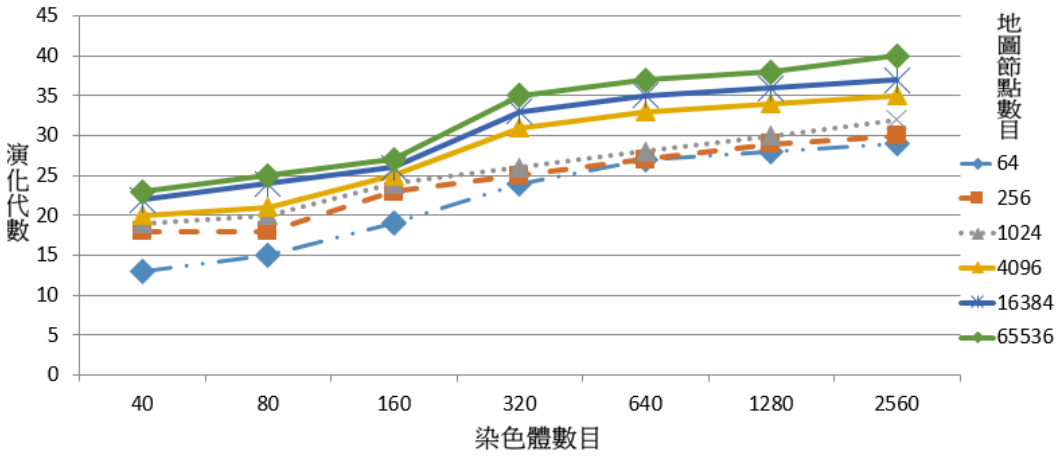
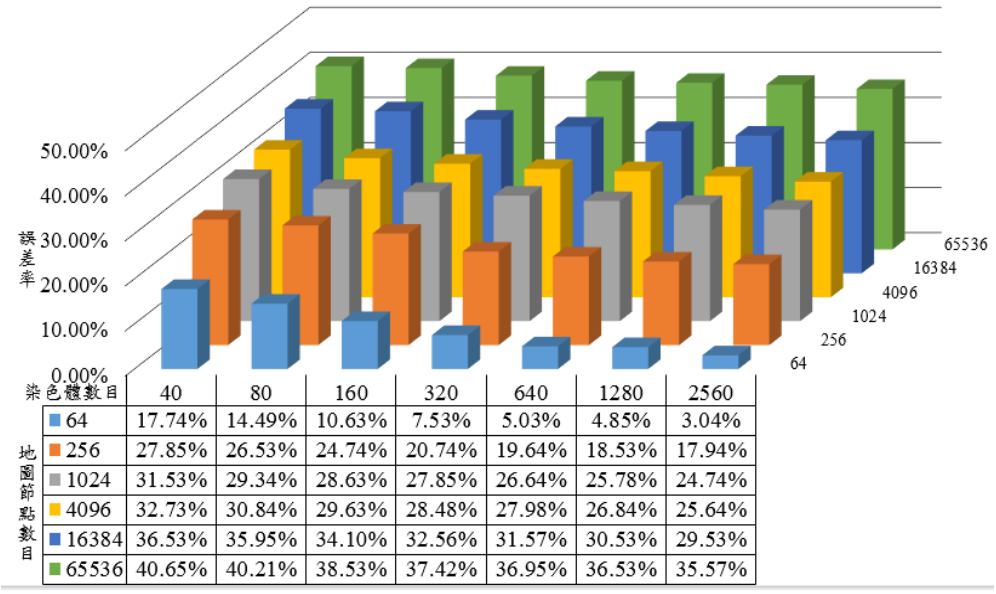


圖 4-8 使用雲端架構之平均尋找最佳路徑所需要的演化代數

表 4-6 中得知，最多演化 40 世代內達到最短行車時間的機率。我們可以發現，隨著染色體數目的增加，可以使得到最短行車時間的誤差機率降低，相對而言，所需要收斂代數也會增加，當然運算的時間也會增加。但無使用雲端架構和使用雲端架構的基因演算法收斂代數相比較下，較大的差異在於多節點時(65536 個節點)，使用雲端架構的誤差機率縮減為 35%，相對於無使用雲端架構的誤差機率 48%，為降低其找到最佳行車時間之誤差率；若再將地圖節點目數放大，其誤差機率將會更為明顯的降低。如圖 4-9 所示:

表 4-6 使用雲端架構的基因演算法達到最佳行車時間誤差百分比

染色體數目 \ 地圖節點數目	64	256	1024	4096	16384	65536
40	17.74%	27.85%	31.53%	32.73%	36.53%	40.65%
80	14.49%	26.53%	29.34%	30.84%	35.95%	40.21%
160	10.63%	24.74%	28.63%	29.63%	34.10%	38.53%
320	7.53%	20.74%	27.85%	28.48%	32.56%	37.42%
640	5.03%	19.64%	26.64%	27.98%	31.57%	36.95%
1280	4.85%	18.53%	25.78%	26.84%	30.53%	36.53%
2560	3.04%	17.94%	24.74%	25.64%	29.53%	35.57%



染色體數目	40	80	160	320	640	1280	2560
64	17.74%	14.49%	10.63%	7.53%	5.03%	4.85%	3.04%
256	27.85%	26.53%	24.74%	20.74%	19.64%	18.53%	17.94%
1024	31.53%	29.34%	28.63%	27.85%	26.64%	25.78%	24.74%
4096	32.73%	30.84%	29.63%	28.48%	27.98%	26.84%	25.64%
16384	36.53%	35.95%	34.10%	32.56%	31.57%	30.53%	29.53%
65536	40.65%	40.21%	38.53%	37.42%	36.95%	36.53%	35.57%

地圖節點數目

圖 4-9 使用雲端架構的基因演算法得到最短行車時間的誤差機率

所以我們將應用平行運算技術實現基因演算法，主要的特色是將整體的族群資料，分割成許多的子族群，再分別對每一個子族群進行基因演算，待子族群持續演化，並滿足結束條件時，再經過族群間的交配，產生新的族群，並再持續演化，以得到最佳解。所以我們朝著解決花系統資源，花時間去搜尋的初始族群來進行 MapReduce 平行運算，以達到雲端運算的最大效益。

在表中我們發現在較小地圖中，如 64 個節點和 256 個節點時，因為由於 Hadoop 分散運算 Job 初始化時間較長，對於少量運算，分散運算不見得佔有優勢；隨運算量增加，分散運算與單機運算明顯出現差異，最大差異值接近 10%；但對於較大地圖而言，如 1024 個節點和其更多之節點，在 1024 個節點時，誤差值慢慢的收斂而變小，此時分散式運算的效果逐漸明顯！在 65536 個節點時，演化的代數更大大得縮短，40 代內即可完成。由此可顯示出，使用雲端運算之 MapReduce 對我們平均尋找最佳路徑所需要的演化代數是有顯著成效的。

4.4 使用節點模擬以計算平均尋找最佳路徑所需要的收斂代數

雲端架構結合基因演算法的優勢在於我們可以保留上一次的運算結果來加速運算速度，假設我們將上一次運算結果的染色體數目保留一半的話，以 32×32 的 matrix map 來做實驗，在未保留任何染色體的情況下，原先在染色體數目為 2560，收斂的代數為 30 代左右，約莫經過 12 個節點後收斂代數開始下降，如圖 4-10 所示。然而如果保留一半的染色體數目，一開始需要的代數為 33 代，在第 4 次導航後，所需收斂的代數就只要 17 代即可收斂，後續也是趨於緩緩下降，如圖 4-11 所示。由實驗結果顯示出，雲端架構結合基因演算法可以保留上一次的運算結果來加速運算速度的優勢是顯著的。

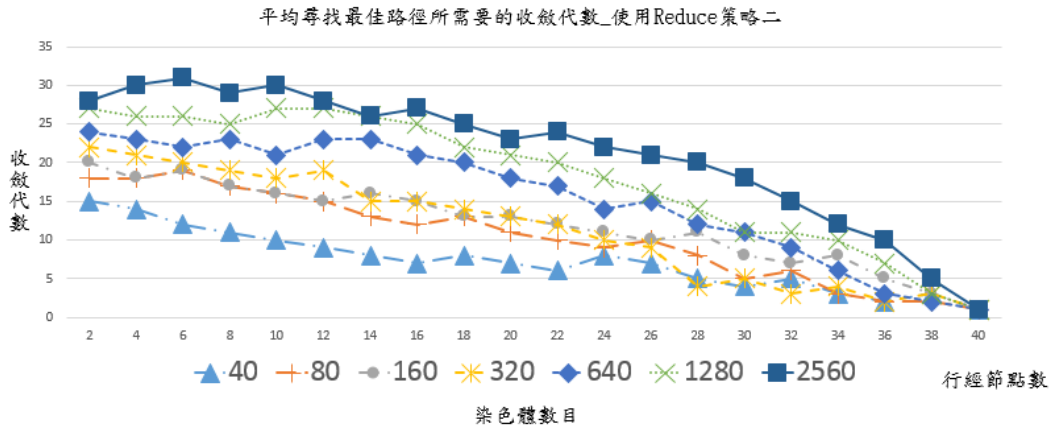


圖 4-10 平均尋找最佳路徑所需要的收斂代數_未使用上次計算結果

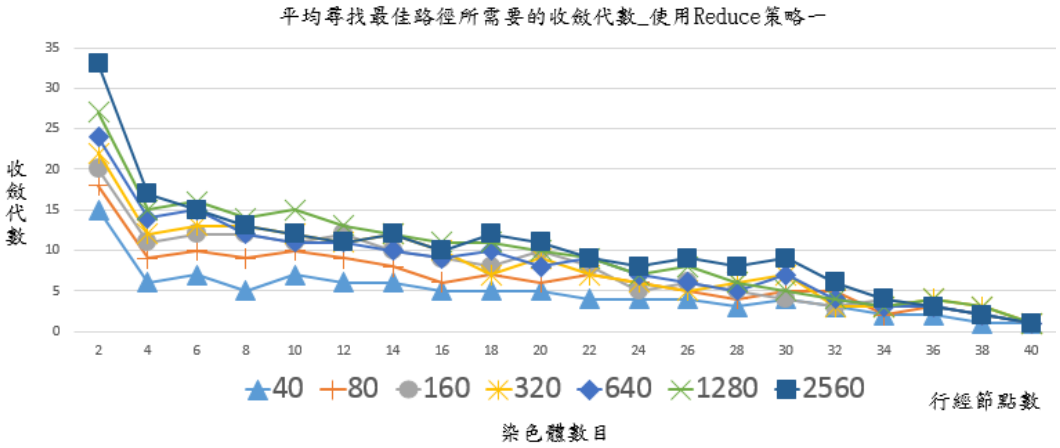


圖 4-11 平均尋找最佳路徑所需要的收斂代數_使用上次計算結果

4.5 基因演算法多次平均與單次的誤差百分比比較

再來，為了能更貼近真實情況，現實生活因為需要用到導航的時機，基本上行車時間是相當長的，所以我們使用多次導航來增加基因演算法的準確程度。我們也使用相同的 square matrix map 為我們的實驗地圖。基因演算法中所使用的交配機率為 80%，突變機率為 7%，因考慮節點數的多寡，最多演化代數設為 60 代，並將實驗次數設定為 1 次和 500 次，也就是說，每一份實驗做 500 次取平均值。所得結果如表 4-7 所示

表 4-7 多次平均與單次的誤差百分比比較

	地圖的NODE數目									
多次平均 / 染色體數目	64		256		1024		4096		16384	
	有	無	有	無	有	無	有	無	有	無
80	16.24%	17.74%	25.94%	27.85%	28.74%	31.53%	28.53%	32.73%	17.74%	24.75%
160	13.35%	14.49%	24.53%	26.53%	26.94%	29.34%	26.34%	30.84%	15.49%	23.43%
320	9.34%	10.63%	22.94%	24.74%	25.23%	28.63%	25.63%	29.63%	15.01%	22.54%
640	6.94%	7.53%	18.23%	20.74%	24.23%	27.85%	24.85%	28.48%	13.53%	21.84%
1280	3.94%	5.03%	17.84%	19.64%	23.12%	26.64%	22.64%	27.98%	12.93%	20.24%
2560	1.94%	4.85%	16.83%	18.53%	22.42%	25.78%	21.78%	26.84%	10.85%	19.93%
5120	0.04%	3.04%	15.84%	17.94%	21.47%	24.74%	19.74%	25.64%	8.04%	18.74%

由表 4-8 中展現出，我們使用多次導航所改良之後的效能比較，我們可以由 64 個地圖節點當中，若使用 5120 個染色體時，使用多次導航時的距離誤差以縮小至 0.04%，然而，由於節點數目的增加，在 16384 個節點和染色體 5120 條時，誤差也由 18.74%降為 8.04%。藉由染色體的數目增加，我們可以發現以雲端技術的輔助將顯現出其平行運算的效能增進，並配合多次導航，顯示出可以明顯的縮減駕駛所需要的行駛時間

接下來以節點模擬實驗環境所得之 HADOOP 各節點的數據:

圖 4-12 為開始實驗之 Node summary of the cloud environment:

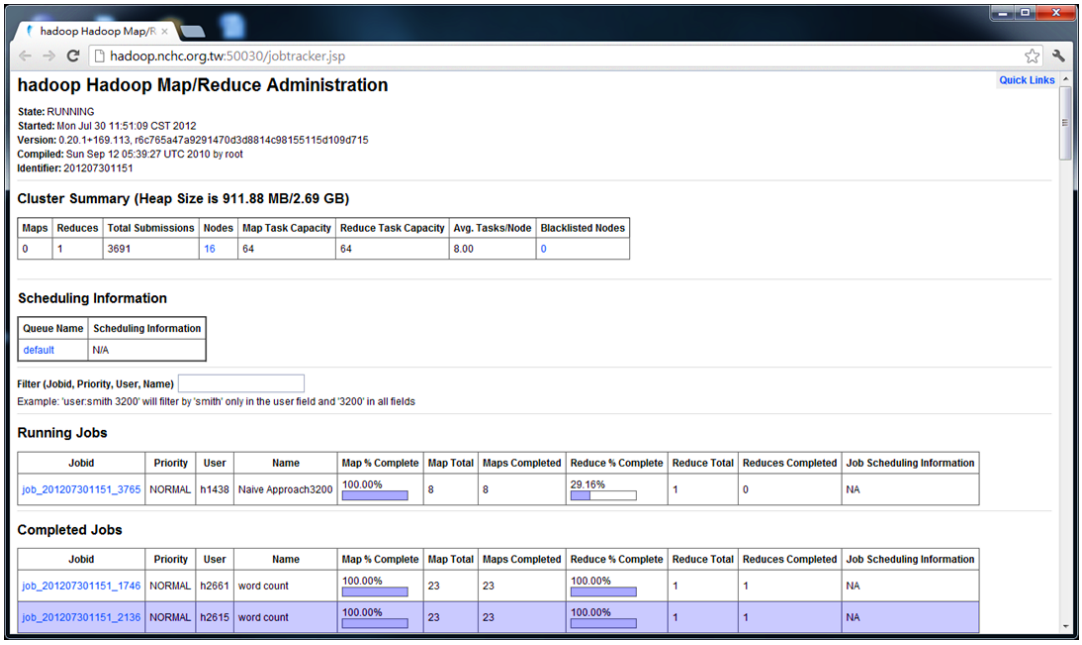


圖 4-12 雲端環境中之節點狀況

藉由 node 數的增加，各個節點的運算時間也隨著節點的增加個數呈等比較低，為了測得收斂的情況，由於 hadoop 易擴充的性質，我們將節點數增加為 16 個，得到了使用 16 個節點，各節點之負載量是平均分布下去做運算的，

故以此演算法來運算節點的平行度是很好的。需運算的節點規模限制，使得運算加速情形開始收斂的結論。但隨著節點內部的運算溝通以及 I/O 時間，產些了些微的誤差，但還是配合平均附載量而產生的運算時間等比下降的現象。

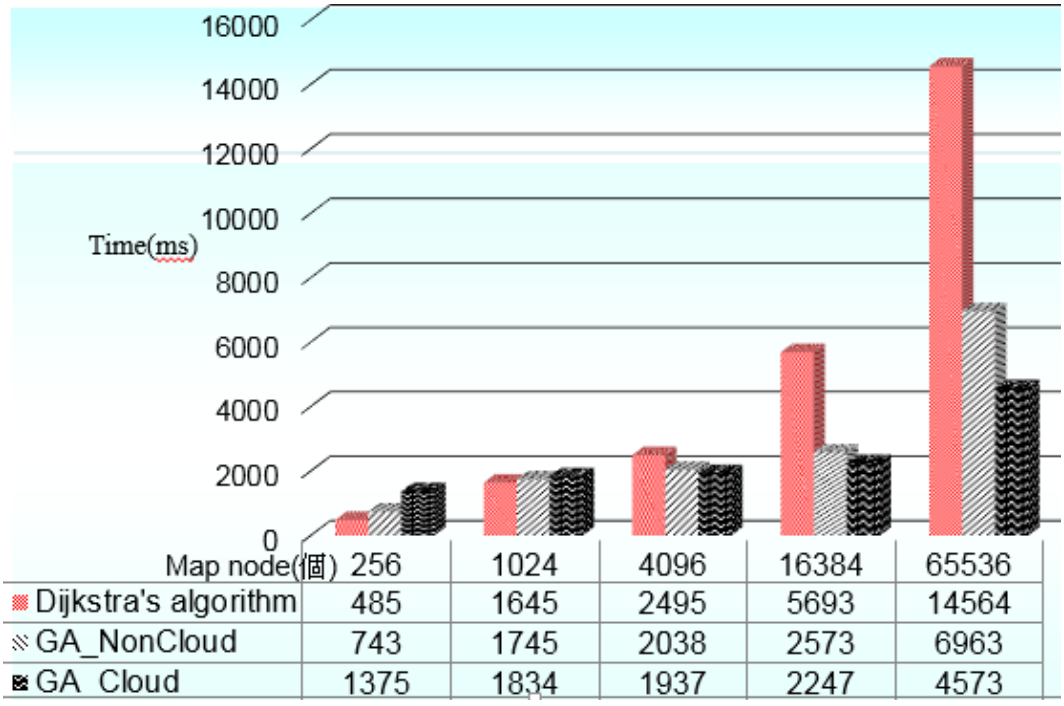
我們再以多點叢集實驗(8 node)，並與 Dijkstra's algorithm、無雲端架構配合基因演算法以及有雲端架構配合基因演算法下去做地圖節點的測試，使用 Dijkstra's Algorithm 來尋找最佳路徑的速度將遠遠不及基因演算法。當節點數量大增時，Dijkstra's Algorithm 的計算時間複雜度(Time Complexity)假設共有 n 個節點，則需要 n*(n+1)/2 次比較，時間複雜度為 $O(n^2)$ ，是相當耗時的！記憶體空間需求也會很大。當連結的網路變大(節點多)，每個 router 的 routing table(節點連結表)就會擴張的很迅速，因此必須要想辦法減少 routing table 所佔的記憶體空間。一般而言，基因演算法所需要的運算時間是以演算法中的收斂代數當做評估指標。同時，也要看染色體的數量，若能有效的使用平行化處理，將可以將染色體的數量所帶給演算法運算時間的負擔降低許多！時間將會下降許多。而基因演算法所需的記憶體空間為染色體的空間和產生下一代的空間，其他的變數所需運算空間相當的少，所以非常適合在記憶體有限的環境中運算，所以適合運算能力及記憶體有限的嵌入式裝置(導航機)運行。

如下面表 4-8 和圖 4-13 所示，在節點數為 256 個節點時，傳統的 Dijkstra's 演算法運算速度會比基於非雲端環境基因演算法以及基於雲端環境基因演算法的執行時間還要來的少，這是因為在節點數目少時，使用基因演算法和雲端架構時，會因為平行化的步驟，使的整體時間將會被拖慢許多；而在 1024 個節點時，三者的差距逐漸縮小，直到 4096 個節點之後，Dijkstra's 演算法運算速度會比基於非雲端環境基因演算法以及基於雲端環境基因演算法的執行時間就會有明顯的差距了，隨著節點數的上升，三者差距更為顯著，在 65536 個節點時，因隨著節點數暴增後，雲端平行運算將發揮其效果，藉由

MapReduce 的概念，Namenode 分配(Map)每個 Datanode 下去個別運算，之後再將個別運算結果收集(Reduce)回來，如此一來，將有助於整體的運算效能提升。

表 4-8 Dijkstra's 演算法與基於非雲端環境基因演算法以及基於雲端環境基因演算法的執行時間比較表

Computing alogrithm / Map node number	Dijkstra's algorithm	GA_NonCloud	GA_Cloud
256	485ms	743ms	1375ms
1024	1645ms	1745ms	1834ms
4096	2495ms	2038ms	1937ms
16384	5693ms	2573ms	2247ms
65536	14564ms	6963ms	4573ms



Map node(個)	256	1024	4096	16384	65536
Dijkstra's algorithm	485	1645	2495	5693	14564
GA_NonCloud	743	1745	2038	2573	6963
GA_Cloud	1375	1834	1937	2247	4573

圖 4-13 Dijkstra's 演算法與基於非雲端環境基因演算法以及基於雲端環境基因演算法的執行時間比較

藉由本實驗測試，證明在大量資料存取的環境下，利用 Hadoop 的 MapReduce 平行運算架構的效能可以比傳統的資料單點運算表現更加出色，本研究使用的是 8 至 16 個節點的叢集，如擴大叢集規模，可以進一步提昇叢集的效能。而由本研究的實驗測試，也可充份驗證 HDFS 在即時性巨量資料的存取上的效能，也可以充份滿足巨量地圖圖層節點運算和管理的需求。

4.6 使用變動速率以基因演算法結合雲端架構尋找路徑

在本小節以上皆是以固定速率來做為實驗的輸入速率資訊，而在本小節中，我們考慮在真實的 ITS Server 實際運作情況下，由於現實中的交通流量根據不同的路段和不同的道路階層，而不斷的更新及時道路速度，而我們為了更貼近真實的行車情況，所以導航演算法就必須將變動速率的結果給包含

進去。而基因演算法遇到變動速率的問題與 Dijkstra's Algorithm 比較下會有不同的結果；我們設計的基因演算法是可以將上次使用過的演算結果當作回饋值(feedback)，保留一部分的染色體，同時再產生一部份的染色體，而新的運算就會比較快速的收斂運算完成，達到更有效率的結果。然而 Dijkstra's Algorithm 遇到變動速率問題時，每每速率變動更新後，就只能重新設定起點和終點，再重新計算過，相當的費時沒效率。

我們一樣使用基因演算法模擬節點(square matrix map)來做實驗，這部分的地圖為使用 8×8、16×16、32×32、64×64、128×128、256×256 的方陣地圖，各有 64、256、1024、4096、16384、65536 個節點，假設最左上角為起點，最右下角為終點，每個節點與每個節點的距離假設為 250 公尺，設定一般道路行駛速率為 30-120，隨機分佈行駛速率於地圖上，而每經過交叉點時，速率就會隨機更動 10%、15%、30%，例如假使固定行車速率為每小時 100 公里的話,那變動速率 10%就是速率每小時 90-110 公里,時間區間內隨機跳動，以模擬真實行車狀況，如高速公路遭遇塞車或一路暢行無阻等。我們保留上次的計算過的染色體 50%，藉由此回饋機制，以加快其收斂和運算的速度。而基因演算法所用的交配機率為 90%，突變機率為 8%，最多演化 50 代，每一份實驗做 100 次取平均數，結果如表 4-9 和表 4-10 所示：

表 4-9 使用變動速率即時更新與原始最短時間路徑比較 1

地圖節點數目

變動百分比 / 染色體數目	64			256			1024		
	10%	15%	30%	10%	15%	30%	10%	15%	30%
160	5.35%	3.42%	-1.84%	10.39%	4.23%	1.38%	3.12%	2.63%	1.39%
320	2.34%	0.63%	-4.53%	5.32%	3.28%	0.23%	2.49%	1.94%	0.52%
640	0.94%	-1.53%	-6.23%	2.48%	0.23%	-1.42%	1.42%	0.32%	-1.49%
1280	-1.94%	-3.03%	-8.34%	1.32%	-1.42%	-2.54%	0.52%	-1.42%	-2.42%
2560	-4.94%	-4.85%	-9.32%	-1.23%	-2.23%	-4.23%	-1.43%	-2.42%	-3.93%
5120	-5.04%	-6.84%	-10.30%	-2.42%	-4.58%	-7.32%	-2.47%	-4.01%	-5.39%

表 4-10 使用變動速率即時更新與原始最短時間路徑比較 2

地圖節點數目

變動百分比 / 染色體數目	4096			16384		
	10%	15%	30%	10%	15%	30%
160	0.95%	-0.32%	-1.32%	0.43%	-0.85%	-1.42%
320	-0.54%	-1.32%	-3.32%	-0.53%	-1.45%	-2.43%
640	-1.34%	-3.01%	-4.53%	-1.53%	-3.53%	-4.23%
1280	-2.43%	-5.28%	-6.34%	-3.53%	-6.34%	-7.32%
2560	-4.34%	-7.45%	-8.34%	-5.34%	-8.34%	-10.42%
5120	-6.45%	-8.34%	-9.53%	-7.43%	-10.32%	-12.53%

如表 4-9 和表 4-10 可以顯示出在不同的速率更新頻率下，使用基因演算法結合雲端架構來進行多次導航，每次更新便不斷的反覆修正最佳路徑下，所獲得的結果比上未使用速率更新的情況下好上不少。

表 4-9 和表 4-10 中的負數表示比原始路徑所需要的行車時間還要來的少。我們在 16×16 的地圖中，變動速率為 30%，使用的染色體數目為 5120，比使

用 Dijkstra's Algorithm 找出來的最短時間路徑還要好上 4.58%。在變動速率 10%的情況下，從 10.39%到-2.42%，會有此大幅度的變動是因為染色體一開始的時候尚不足，所以地圖節點探索的還不夠多，等到探索到一定數量的時候，效果就慢慢突顯出來了。由 10%到 30%的變動速率更加貼近真實行車狀況，所求得的最短行車時間效果是相當顯著的。

Chapter 5 結論與未來展望

智慧型運輸系統（Intelligent Transportation Systems, ITS）透過通訊系統即時的溝通與連結，改善或強化人、車、路之間的互動關係，提升用路人的交通服務品質與績效，進而增進運輸系統之安全、效率與舒適，同時減少交通環境衝擊。然而目前常見的應用方式乃是以GPS搭配地圖資訊系統進行行車導航，但它們大多數屬於封閉式系統，使用者必須不斷地購買新的地圖資訊，方能保有最新版的資料。由於目前在行車導航方面，屬於開放式、且廣泛地被人採用的系統仍然非常稀少,因此我們利用現今蓬勃發展之雲端技術，試圖建構一個開放式的智慧型行車導航系統。另一方面雲端概念的興起提供 ITS 整合建置的契機。龐大且同性質的交通偵測資料處理程序，可用雲端運算的方式，有效提升即時交通資訊服務效率及服務容量，降低營運成本!

本論文為利用資通訊科技技術(雲端平行化運算)提升運輸系統的效率、安全與便利性。GIS空間運算使用Hadoop實作，再由OGC WPS(Web Processing Servic)方式提供服務。這樣使用QGIS或其它支援WPS的Client就可以使用其功能來作運算。類似這樣的方式就可以集中自己的精神作自己擅長的事，且與整個的GIS作結合，也是一個不錯的方向。由於資通訊技術的成熟，讓原本處於閉塞而無法與外界進行單向或雙向資訊傳輸互動的汽車環境，增加與外界溝通的通訊能力，也因此對車載資通訊系統產生了強烈的市場需求。在通訊技術中，車用通訊產品逐漸朝網路化、智慧化及整合化三大趨勢發展。

參考文獻

[1] IOT 路網數值圖 100 年版使用手冊,交通部運輸研究所,中華民國 100 年 8 月

[2] 智慧型運輸_101 年度 ITS 白皮書,交通部, 中華民國 101 年 7 月

[3] 國科會資訊學門研究發展規劃書,行政院,中華民國 101 年 12 月, http://www.etop.org.tw/jspui/program/cs/

[4] 李昇暾、詹智安(2012)。《Android 雲端實務程式設計：適用 Android 2.x~4.x》。臺北市：碁峰資訊。

[5] 江寬、龔小鵬(2011) 。Google API 開發詳解：Google Maps 與 Google Earth 雙劍合壁(第二版) 。臺北市:松崗資產管理。

[6] GIS 相關技術分享 – Google Map, Google Earth, Virtual Earth, Web GIS, RIA GIS, QGIS (Quantum GIS)…等相關技術與心得分享, 中華民國 102 年 5 月,http://gis-tech.blogspot.tw/

[7] 李介中,地理資訊系統簡介, 中華民國 101 年 3 月 http://lab.geog.ntu.edu.tw/course/gislucc/GIS_introduction.pdf

[8] (GE-7)GPS 的 gpx 檔+照片定位+Google Earth 的 kml 檔,Keep Environment's Blog, http://blog.xuite.net/lwkntu/blog/13586958

[9] 王鵬(2010)。雲端運算的關鍵技術與應用實例。台北市：佳魁資訊。

[10] .NET 碎碎唸-以運算就資料(在地運算) vs. 以資料就運算 (雲端運算核心技術 Hadoop & MapReduce 概念班上課心得)。 http://dotnetmis91.blogspot.tw/2010/04/vs-hadoop-mapreduce.html

[11] Zhifeng Xiao and Yimin Liu, “Remote Sensing Image Database Based on NOSQL Database,” International Conference on Geoinformatics, pp. 1-5 , June, 2011

[12] S. Ramgovind , M.M. Eloff and E. Smith, “The Management of Security in Cloud Computing,” Information Security for South Africa , pp. 1-7 , August, 2010

[13] A.F. Mohammad and H. Mcheick,“Cloud Services Testing: An Understanding,” International Conference on Ambient Systems, Networks and Technologies, Vol. 5, pp.513-520, August, 2011

[14] N.J. King , and V.T. Raja, “Protecting the Privacy and Security of Sensitive Customer Data in the Cloud,” International Journal of Computer Law & Security Review, Vol. 28, Issue 3, pp. 308-319, June, 2012

[15] T. Zelinka , Z. Lokaj , and M. Svitek ,”Service Quality Management for the ITS Mobile Wireless Multipath Telecommunications Subsystems,” International Conference on Telematics and Information Systems, pp. 1 – 8, May, 2012

[16] I. Muttik and C. Barton, “Cloud Security Technologies,” Information Security Technical Report, Volume 14, Issue 1, pp. 1-6, February, 2009

[17] Jianhua Gu, Jinhua Hu, Tianhai Zhao, and Guofei Sun, “A New Resource Scheduling Strategy Based on Genetic Algorithm in Cloud Computing Environment,” Journal of Computers, Vol. 7, No 1 , pp. 42-52, January, 2012

[18] Google App Engine, 2010. http://groups.google.com/group/googleappengine

[19] Amazon Web Services (AWS), 2010. http://aws.amazon.com/

[20] Windows Azure- A Microsoft Solution to Cloud, 2010.

[21] IBM Cloud Computing, 2010. http://www.ibm.com/ibm/cloud/

[22] Open Nebula, 2010. http://www.opennebula.org/

[23] Sin Man Cheang, Kin Hong Lee, and Kwong Sak Leung," Evolving Data Classification Programs Using Genetic Parallel Programming," IEEE Transactions on Evolutionary Computation, Volume:1, pp. 248-255, December, 2003

[24] Kin Hong Lee and Sin Man Cheang," Evolving Parallel Machine Programs for a Multi-ALU Processor," IEEE Transactions on Evolutionary Computation, Volume:2, pp. 1703-1708, May, 2002

[25] Jong Won Park, Chang Ho Yun, Shin-gyu Kim, H.Y. Yeom , and Yong Woo Lee ," Cloud Computing Platform for GIS Image Processing in U-city," International Conference on Advanced Communication Technology , pp. 1151-1155, February, 2011

[26] Chu-Hsing Lin, Chen-Yu Lee, Jung-Chun Liu , and Hao-Tian Zuo," Investigations of Factors Affecting the Genetic Algorithm for Shortest Driving Time," International Conference of Soft Computing and Pattern Recognition, pp. 106-111 , December, 2009

[27] Chu-Hsing Lin, Jui-Ling Yu, Jung-Chun Liu, Wei-Shen Lai, and Chia-Han Ho,"Genetic Algorithm for Shortest Driving Time in Intelligent Transportation Systems," International Journal of Hybrid Information Technology Vol. 2, No. 1, January, 2009

[28] Chu-Hsing Lin, Jui-Ling Yu, Jung-Chun Liu, and Chia-Jen Li, "Genetic Algorithm for Shortest Driving Time in Intelligent Transportation Systems,"

International Conference on Multimedia and Ubiquitous Engineering, pp. 402-406 , April, 2008

[29] D. Srinivasan , Lai Wei Lup , X. German , E. Taylor , S.H. Ong, “Genetic Algorithm Based Route Planner for Large Urban Street Networks,” IEEE Congress on Evolutionary Computation, pp. 4469-4474, September, 2007

[30] P. Borovska ,M. Lazarova , “Migration Policies for Island Genetic Models on Multicomputer Platform,” IEEE International Workshop on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications, pp. 143-148, September, 2007

[31] Claudio Mattiussi and Dario Floreano, “Analog Genetic Encoding for the Evolution of Circuits and Networks,” IEEE Transactions on Evolutionary Computation, Volume:11 , Issue: 5 , pp. 596-607 , October, 2007

[32] S. Xu and J.C. Bean, “A Genetic Algorithm for Scheduling Parallel Non-identical Bath Processing Machines,” IEEE Symposium on Computational Intelligence in Scheduling, pp. 143-150, April, 2007

[33] Ismail Rakip Karas and Umit Atila,” A Genetic Algorithm Approach for Finding the Shortest Driving Time on Mobile Devices,” Journals of Scientific Research and Essays Vol. 6(2), pp. 394-405, January, 2011

[34] Chu-Hsing Lin, Chen-Yu Lee, and Tang-Wei Wu, “A Cloud-aided RSA Signature Scheme for Sealing and Storing the Digital Evidences in Computer Forensics,” International Journal of Security and Its Applications, Vol. 6, No. 2 , pp. 241-244, April, 2012

[35] Chu-Hsing Lin, Chen-Yu Lee, and Shi-Pei Chien, “Digital Video Watermarking on Cloud Computing Environments,” International

Conference on Cyber Security, Cyber Peacefare and Digital Forensic, pp. 49-53, March, 2013