國立高雄第一科技大學

資訊管理系

碩士論文

基於Hadoop平台的雲端基因架構

The structure of the CloudGene

based on Hadoop platform

研究生：蔡碧展

指導教授：黃文楨 博士

中華民國 九十九 年 七 月

基於Hadoop平台的雲端基因架構

The structure of the CloudGene based on Hadoop platform

研究生：蔡碧展 Pi-Chan Tsai

指導教授：黃文楨 Wen-Chen Huang

國立高雄第一科技大學

資訊管理系

碩士論文

A Thesis Submitted to
Department of Information Management
National Kaohsiung First University of Science and Technology
In Partial fulfillment of the Requirements
For the Degree of Master
In
Information Management

July 2010
Yenchao, Kaohsiung, Taiwan, Republic of China

中華民國 九十九 年 七 月

國立高雄第一科技大學學位論文考試審定書

■碩士班
□博士班

資訊管理 系(所)

研究生 蔡碧展 所提之論文

基於Hadoop平台的雲端基因架構

The structure of the CloudGene based on Hadoop platform

■碩士
經本委員會審查，符合 學位論文標準。
□博士

學位考試委員會

召集人 [signature]

委員 [signature]

張弘毅

黃文楨

指導教授 黃文楨

主任（所長） [signature]

中華民國 99 年 7 月 26 日

（院系留存）

保存期限：永久
表單編號：ACAA-3-08-3703

摘要

雲端運算是 2007 年所提出的一個新興概念，其名稱源自於技術人員當在討論網際網路時皆會畫出一朵雲用來代表網路網路，因此雲朵也就變成的網際網路的代名詞了。而雲端運算簡單的說就是在網際網路上進行資料的儲存與運算。至今，Google、Amazon、Microsoft、IBM 等 IT 大廠，都已經推出自己的雲端計算平台，並且把雲端運算當作是未來最重要的發展趨勢。因此，其重要性可見一斑。

MapReduce 近來已經成為大規模平行系統的重要開發模型。同時，Hadoop 是一個開放原始碼的 MapReduce 架構，並且被廣泛地應用在以大規模資料運算為主的雲端計算。而在雲端架構下，通常會採用虛擬機器(VM)來做為實作平台，因為他們比較易於管理與維護。

本研究目的希望可以藉由雲端架構的便利性以及龐大的計算能力，而率先提出一種基於 Hadoop 平台的基因創新架構，用以加快基因演化速度。因此本研究提出一個稱之為雲端基因(CloudGene)架構。並且透過演化複雜度比較高的 MonaLisa 圖形演化實驗，結果顯示出比傳統的分散式架構演化速度快上 5 倍。而需要大量資料的換硬幣實驗，則是快上 7 倍。

關鍵字：MapReduce，雲端計算，Hadoop，虛擬機器，基因演算法。

ABSTRACT

Cloud computing is an emerging concept from the year of 2007. Its name is derived from the technical staff while discussing the Internet, by drawing a cloud to represent the Internet network. Therefore it became the Internet cloud synonymous. The cloud computing simply means that the data storage and computing on the Internet. So far, Google、Amazon、Microsoft、IBM and other IT manufacturers, have launched their own platforms for cloud computing, and made the cloud computing is the most important trend in the future.

MapReduce has recently become an important development model on large-scale parallel systems. Meanwhile, Hadoop is an open-source MapReduce framework, and is widely used in large-scale information calculation cloud-computing. In the structure of cloud-computing, it usually uses virtual machines (VM) as the implementation platform, because they are easy to manage and maintain.

The purpose of this study is to take advantage of the convenience of cloud structure and its computing power. In this thesis we will propose an innovative gene-based Hadoop framework to accelerate the pace of genetic evolution. The proposed architecture is called “CloudGene”. The experimental results of Mona-Lisa graphic evolution show that and the coin experiment results show that CloudGene is 7 times faster than the traditional distributed architecture.

Keyword：MapReduce, Cloud computing, Hadoop, Virtual machine, Genetic Algorithm.

致謝

本論文得以順利完成，首先要誠摯感謝我的指導教授－黃文楨教授。在研究的期間，提供了相當寶貴的建議，協助我將原本凌散的思緒逐步的整合成一篇完整的論文，讓我受益良多。同時也要感謝兩位口試委員：曾守正及張弘毅博士抽空前來口試，並提供許多寶貴的建議與指教，使論文得以更加完善。

此外，也要感謝這兩年裡一起打拼奮鬥的 C220 伙伴們。感謝與我一起參加第一屆趨勢雲端競賽的成員：志偉、兆師、恩先雖然比賽沒有得獎，但還是有滿滿的回憶在心中；也感謝參加第二屆趨勢雲端競賽的成員：侑鑫、奕中、淮旻有了你們的努力我們才得以進入決賽；實驗室的同學與學弟們：嘉苑、延璉、智誠當然也不能忘記，謝謝大家的幫忙及搞笑，讓實驗室活潑了不少。

還要感謝陪伴在我身邊的女友秋香，一路走來有歡笑也有淚水，也因為妳的支持讓我即使面對眾多難題，還是能夠關關難過關關過。最後，謹以此拙作獻給所有關心我的家人、師長與朋友們，願將此喜悅與你們一同分享。

目錄

圖目錄

表目錄

第一章 緒論

1.1 前言

近年來隨著網際網路愈來愈普及與網路頻寬速度的增加，使得使用者越來愈習慣使用由網路服務業者所提供的服務，如Gmail、Google Docs、Google Calendar、無名小站、YouTube 等。相對的因應該服務也產生了大量的資料，而這些資料應該如何處理，也著實成為各網路服務業者所困擾的問題。

當然美國網路搜尋的龍頭Google也遇到了相同的難題，因此開始著手研究應該如何處理以及儲存如此龐大的資料量。之後便分別在2003年提出分散式檔案系統架構 Google File System(GFS)[22]、2004 年提出大規模資料處理的模型MapReduce 架構[8]以及2006年提出基於Google File System針對結構化資料儲存的分散式檔案系統 BigTable[9]，並在其論文中提出如何運用其架構以對於Google的各項服務增加了重大的效益。

而雲端運算(Cloud Computing)此一新興名詞也就產生了，該名稱源自於技術人員當在討論網際網路時皆會畫出一朵雲用來代表網路網路，因此雲朵也就變成的網際網路的代名詞了。而雲端運算簡單的說就是在網際網路上進行資料的儲存與運算。其所運用的技術，本質上來看也算是分散式運算的一種，其最基本的概念，就是希望能夠透過網際網路將大規模的資料量放置在雲端，並且將龐大的單一處理程序自動分拆成無數個較小的子程序，再交給雲端中的為數眾多的電腦來處理，以加快處理速度，最後再將處理的結果回傳給使用者。透過這樣的技術，企業可以在極短的時間內，處理數以千萬計的資料，達到類似超級電腦的處理能

力，但其價格又不像超級電腦那般的昂貴以及難以維護。

1.2 研究背景

雲端計算(Cloud computing)約略是在2007年所提出來的全新資料處理概念，其最具代表性的應用，即是Google將其MapReduce與GFS概念運用在搜尋引擎索引的建立。其最早的由來是因為大家發現Google的搜尋引擎非常的有效率，因此開始有人請Google站在教育的方向與學子們分享其經驗。

因此Google便於2007年10月在全球宣布雲端計畫，同時與IBM合作，將美國多所大學納入雲端計算中，包含麻省理工學院、史丹福大學、加州大學柏克萊分校、馬里蘭大學等。希望可以降低分散式運算技術在各個大學中學術研究的成本，並且為這些大學提供相關的軟硬體設備與技術的支援。而學生可以透過這項計畫開發各種以大規模運算為基礎的研究計畫。

Amazon也於2007年提供Elastic Compute Cloud(EC2)[1]服務平台，讓一般的企業或是使用者在面臨大量的運算或是儲存需求時，不需要自行購買機器並且架設平台，而是當需要時，就立即向Amazon購買其所需要的運算能力或是儲存的容量。而且計費的方式相當的多樣化，不僅可以用小時計費也可以採用CPU計費。當使用完畢即可立即停止服務與收費，完全非常的彈性。如今，Amazon已經提供多種平台以供選擇[2]，並且持續增加中。

在2008年1月Google也宣布在台灣啟動「雲端運算學術計畫」，提供Google的雲端計算軟體與技術，並且請自家的工程師與台大、交大資工系的師生分享雲端運算技術，這是Google在美國以外，首次在海外進行雲端運算的學術合作計畫。

而國際知名的防毒大廠趨勢科技也從2009年開始推出「騰雲駕霧程式競賽」，希望透過競賽吸引更多的學生投入開發與研究。亦與台大合作開發雲端趨勢學程，將這種先進的大規模、快速計算技術開始推廣到校園。

2009 年 4 月，Google App Engine(GAE)也開始提供 Java 的執行環境，並且加入一系列專門提供給企業使用的功能，還有針對使用 Eclipse 的開發者提供 GAE 的 plug-in 套件，讓使用者可以更加方便地透過開放原始碼的 Eclipse 開發工具來開發在 GAE 上面執行的應用程，由此可見 Google 已經為雲端運算做好了準備。

1.3 研究動機與目的

由於基因演算法是採用達爾文的物競天擇理念，也就是一種模仿在自然界中適者生存，不適者淘汰的概念，因此就像大自然的演化一樣，在面臨大量資料時通常會需要花費較長時間的運算與等待，才能跳脫區域解而得到比較好的運算結果。但在這樣的情況下常常會造成無法在有限的時間下全部演化完成。

因此本研究希望可以藉由雲端架構的便利性以及龐大的計算能力，而率先提出一種基於 Hadoop 平台的雲端基因創新架構，用以加速基因演化的過程。並且透過實驗顯示出本研究所提出之架構的演化速度以及能力的確是優於其他架構。

1.4 重要性及貢獻

本篇論文主要的貢獻可歸納如下幾點：

- 率先提出基於雲端架構的基因平台，在此平台下能更容易地處理大量的運算資料。
- 透過雲端的眾多電腦，能夠大幅地提升基因演算法的運算能力。
- 在傳統架構下需耗時許久的作業，採行此架構能在有效時間內得到結果。

1.5 論文架構

本研究主要探討基因演算法在雲端架構下的應用，希望透過雲端運算龐大運算能力加速基因演算法的執行速度以及增加能夠處理的量。

本論文章節介紹如下：

第一章緒論，簡述雲端運算的研究背景及研究動機與目的，為以後的章節打下了一個基礎。

第二章文獻探討，深入介紹 Hadoop 平台的技術架構，以及相關論文探討。

第三章研究方法，針對本研究所提出的雲端基因架構做詳細的介紹。

第四章為實驗結果與分析，開始介紹 Hadoop 平台環境的建立，並且將實驗結果詳細的分析與比較。

最後第五章為結論及探討未來可繼續研究的方向與議題。在本研究論文的最後，一併附上研究過程中所引用與參考的文獻資料。

第二章　文獻探討

由於目前雲端運算的主流架構皆是以Google所提出的GFS(Google File System)與MapReduce概念下去發展，第一節與第二節將會詳細的說明GFS與MapReduce的理論架構，第三節會介紹基因演算法，第四節則會介紹分散式基因演算法目前的發展。

2.1 Google File System 分散式檔案系統

GFS分散式檔案系統對於分散式計算非常的重要，因為當計算單元分散到各台電腦上時，通常都會需要讀取或是儲存資料，不管是讀取或是儲存的動作都需要有一個檔案系統來將資料作適當的存放與備份。

Google所提供的GFS是一種相當容易擴大檔案系統容量的架構[22]，主要就是用在大規模、分散式以及需要對大量資料進行運算的應用。它能夠運行在像你我所使用的一般電腦上，並不需要企業等級的高階伺服器才能夠執行，雖然一般的電腦故障率較高，但是透過GFS內部預設的備份機制，能提供強大的容錯功能。而GFS與以往的檔案系統最大的不同之處在於：

a. 硬體會經常性故障，因為系統是建立在大量的廉價電腦上，而這些電腦常常需要被大量的存取，因此硬體的錯誤不應該被視為異常，而是應該視為必定會發生的情況，系統需要能夠自動處理。也由於硬體會時常故障，系統必須這些電腦持續監測，當有問題發生，系統需要從故障中進行恢復。

b. 由於系統需要存放大量的超大型檔案。可能會有數百萬個超過100MB的檔案。也有可能是數GB的檔案，因此應該要對這些大型檔案做有效的管理。但同時

也必須要支援小型的檔案，但系統不必為小檔案進行特別的效能調校。

c. 大部分對於存在 GFS 的處理模式為在檔案結尾處增加資料，比較少的情況是修改既有的資料。因為存在 GFS 檔案系統的資料通常都是很大的單一檔案，所以對一個檔案的隨機寫入操作通常是不存在的。取而代之的是大量的程式對單一檔案進行依序的讀取或是寫入動作，因此必須對此動作進行最佳化以及保證大量的程式同時寫入或是讀取時彼此之間不能混淆。

d. GFS 在寫入的資料完成時，通常是很少會對檔案進行更動，因此在依序讀取大量的資料時，必須要非常的有效率。但同時也需增加在隨機位置對寫入少量的資料的支援，只是不需要特別有效率。

如圖 2-1，一個典型的 GFS 叢集是由一個 master 和多個 chunkserver 所組成，並且可能由多個用戶端程式所存取。每一個節點都是一個普通的 Linux 電腦環境，而 GFS 系統則是運行在該 Linux 系統下的程式。當然只要電腦的資源足夠 chunkserver 與 client 也可以運行在同一部機器上。

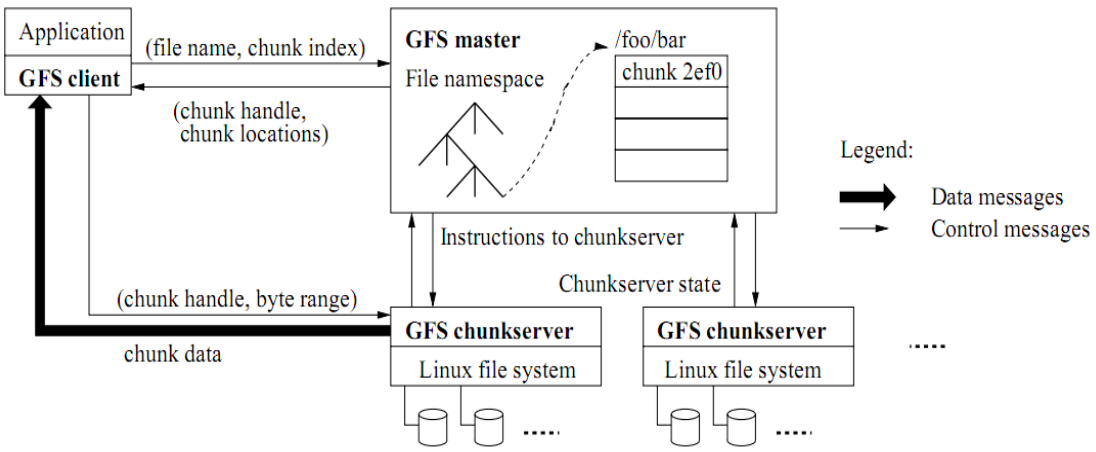

圖 2-1 .典型 GFS 叢集架構[22]

在 GFS 的檔案系統下，每一個存在 GFS 的檔案都會被 master 分割成固定大小的 chunk。每一個 chunk 資料都由 master 根據建立的時間產生一個唯一並且無法改變的 chunk handle 名稱，長度為 64 位元，而該名稱其實就等同於檔名。一個大的檔案被分割成多個 chunk 之後，這些 chunk 就會被存放在多台 chunkserver 上，而且是用 Linux 本身的檔案系統所儲存。當需要存取檔案時，就需要先知道該資料是放在那個 chunkserver 並且透過 chunk handle 以及 byte range 來作讀取。而為了提供容錯機制，因此每個 chunk 檔預設都會有 3 個備份，並且放入不同的 chunkserver 上，以避免單一 chunkserver 毀損。但是使用者可以對不同 namespace 下的檔案設定不同的備份檔案數。

master 會負責所有檔案系統的 metadata，包括 namespace 資訊、檔案與 chunk 的對應關係、chunk 所存放的實際位置以及該檔案的相關存取權限等等。除此之外 master 也負責 GFS 系統的相關活動，像是 chunk 的分割機制、已經失效的 chunk 檔案或是 chunkserver 之間的 chunk 備份管理。因此 master 和這些 chunkserver 彼此之間都會有定期的 HeatBeat 通訊，透過 HeatBeat，master 可以傳遞指令給 chunkserver 並且 master 亦可藉此知道 chunkserver 是否已經掛掉，當每 chunkserver 經過一段時間後沒有傳送 HeartBeat 給 master，那麼 master 會將該 chunkserver 上的所儲存的檔案列出清單，逐一尋找其 chunk 的備份檔，另外複製到其它台 chunkser 上，以維持系統預設的備份個數。

各個應用系統則是透過 GFS 提供的檔案系統操作 API，來與 master 或是 chunkserver 進行讀寫的互動。GFS client 與 master 互動的資訊主要是 metadata，而實際的檔案讀寫動作則是發生在 chunkserver，藉此以避免掉 master 變成效能瓶頸的情況。無論是 GFS client 或是 chunkserver 都不需要 cache 檔案資料。因

為在大部分的情況下所儲存的資料都非常的龐大，大到無法有足夠的容量可以 cache，而也因為沒有 cache 所以不需要做同步的機制，使得整個系統設計也大大的簡化了。但是 GFS client 會 cache metadata 的資料，以減少 master 的負擔。而 chunkserver 當然也不需要 cache 資料，因為 chunk 是以檔案的形式儲存在 Linux 的檔案系統下，所以 Linux 的 buffer cache 就已經會將常用到的資料 cache 到記憶體了。

接下來就簡單的介紹圖 2-1 的讀取流程。

a. 首先，GFS client 會將 Application 所要讀取的檔名與 byte offset，依據使用者或是系統內定的 chunk 大小，轉換成檔案的 chunk index，也就是 Application 所要讀取的資料是在第幾個 chunk 中。

b. 接著將 file name 與 chunk index 傳送給 master。

c. master 就會回傳相關的 chunk handle 與實際儲存的 chunkserver location 資訊。然後 GFS client 會將該資訊以檔名+chunk index 的格式當作 cache 的 key，並且儲存起來。實際上，通常 GFS client 會在一次連線中要求多個 chunk 的資訊，master 當然也會回應多個 chunk 資訊給 GFS client。

d. GFS client 接著對 chunkserver 發出連線請求，由於同一份 chunk 會存在三台不同的 chunkserver 上，因此 GFS 會選擇離 GFS client 較近的 chunkserver 做連線。而連線的資料包含了 chunk handle 以及所要取得的 byte rang，後續的讀取動作 GFS client 會透過已經 cache 起來的資訊，再去取的下一筆 chunk 的位置資訊，而不需再經由 master 以減少其負擔。除非 cache 資訊已經過期，那麼就需要再重新向 master 取出連線資訊。

e. chunkserver 傳回 GFS client 所要求的檔案資訊。

2.2 MapReduce 運算模式

MapReduce 對於 Google 來說相當的重要，這是為什麼 Google 的搜尋引擎可以這麼的有效率將我們想要的資料找出來的一個重要關鍵。當然不僅僅應用在改善搜尋引擎的準確度上，MapReduce 已經應用在許多 Google 服務中，如 Google Analytics、Google Earth、Orkut、Google Finance 以及 Personalized Search 等等[9]。因此站在教育與分享的角度上，Google 將 MapReduce 的運算模式公開發表成論文[8]。不過雖然 Google 公開了 MapReduce 的內部運作概念及模式，但是卻沒有公開所開發完成的系統。但是我們依然可以透過其論文一窺 MapReduce 的奧秘。

MapReduce 其實就是一種容易撰寫的分散式與平行處理的程式設計模型，它主要運用在處理大規模的資料集。主要由 Map 與 Reduce 兩個動作組成，開發人員在撰寫程式時，僅需要對所要處理的資料，於 Map 的架構下撰寫商業邏輯(也就是資料運算的部分)，透過 key/value pair 取得輸入資料，運算完畢後一樣將運算結果透過 key/value pair 將資料輸出下一階段處理。而 Reduce 階段則是將上一階段傳送過來的資料依照相同的 key 值做合併簡化處理即可。

開發人員完全不需要有平行處理或是分散式系統的知識，包含了不需要知道程式是如何複製到各個電腦中、資料如何讓在各自電腦中的 Map 存取、機器如果掛掉工作應該如何重新執行以及各台機器中工作如何平行處理等等問題。因此開發人員只需將心力著重在真正要解決的事物上即可，這已經大大的減輕開發人員的負擔。而 MapReduce 強大之處，就是在於你僅須撰寫幾行程式就可以讓成千上萬台電腦為你的工作。

如圖 2-2，是 MapReduce 架構的整個執行流程。當開發人員將其開發完成的程

式丟到 MapReduce 去執行，就會產生如圖 2-2 的操作。流程說明如下：

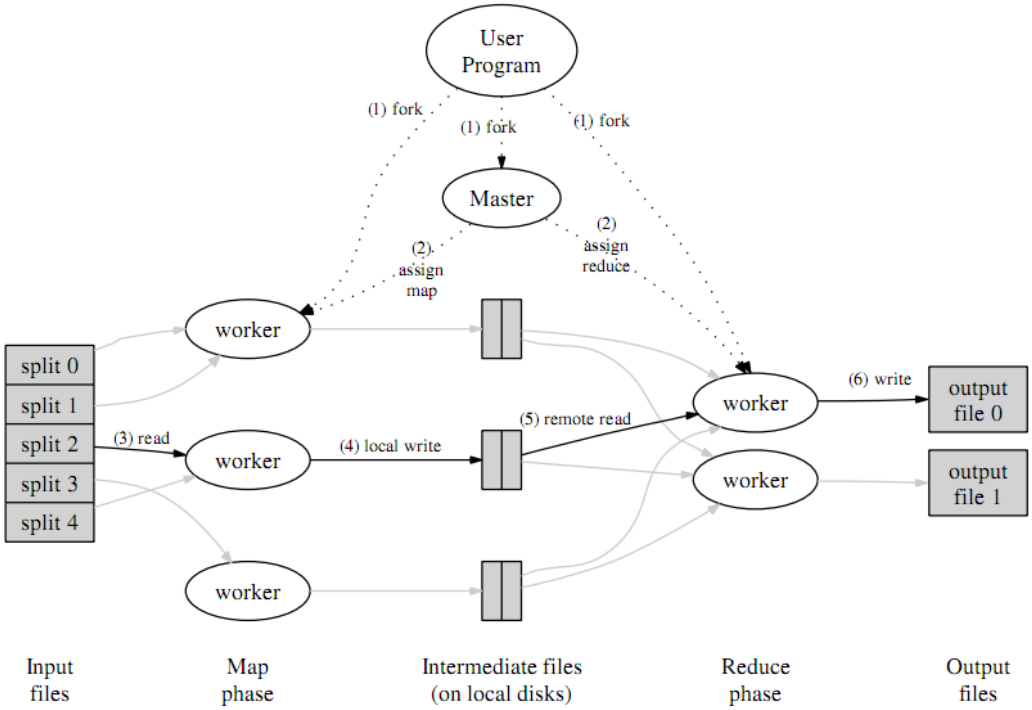

圖 2-2 .MapReduce 架構執行流程[8]

1. MapReduce 架構首先會將該程式所要處理的資料分割成 M 塊，每塊大小多為 16~64MB 之間(每塊的大小可以透過參數來指定)，接著 MapReduce 再將使用者所要執行的程式複製到需要執行的機器上，並且啟動該程式執行。其中會有一台機器負責監控各台機器的執行情況，稱之為 Master。其餘的機器則稱之為 worker。
2. Master 負責將 M 個 Map 任務分派給空閒的 worker，也負責將 R 個 Reduce 任務分配給空閒的 worker。當然兩者的 worker 可以是同一台機器也可以是不同台機器。
3. 其中一個被分配 Map 任務的 worker 從被分割的資料中讀取所需要處理的資料片段，從輸入資料中剖析出 key/value pair 並且傳送給使用者所撰寫的 Map

程式處理。Map 程式再將產生出來的中間結果 key/value pair 存到緩衝記憶體。其他 Map 階段的機器也是如此處理。

4. 這些緩衝到記憶體的中間結果將會被定時地寫入本機硬碟，並且透過 partition 機制分割 R 個區域。而這些中間結果所儲存的位置資訊將會被傳回 Master，並且交由 Master 將此資訊遞送給執行 Reduce 的 worker。
5. 當 Master 通知 Reduce 的 worker 這些位置資訊時，worker 便會使用 RPC 來讀取 Map worker 所產生的中間資料。當 Reduce worker 讀完所有的中間資料，就會使用 key 來進行排序，這樣會使得所有相同 key 的資料群聚在一起。但是若資料太大無法於記憶體中排序，那麼就必須要使用外部排序機制了。
6. Reduce worker 會取出所有相同 key 的 value 傳送給開發人員所撰寫的 Reduce 程式進行處理。每個 Reduce 任務所處理完的結果將會輸出一個檔案，因此有多少個 Reduce 任務就會產生多少個檔案。
7. 當所有的 Map 與 Reduce 任務都處理完了之後，Master 會將執行的權限交給開發人員所撰寫的程式來繼續執行。

當這些任務都結束之後，MapReduce 的執行結果就會存放在開發人員所指定的檔名中，在一般的情況下有 R 個 Reduce 任務就會有 R 個輸出檔。開發人員通常也不需要將 R 個輸出檔合併成一個檔案，而是將此資料在丟給下一個 MapReduce 程式處理。

由於 MapReduce 架構的主要概念是源自於函數語言，我們可以透過下列式子來簡單說明其概念。

map	(k1,v1)	→	list(k2,v2)
reduce	(k2,list(v2))	→	list(v2)

表 2-1 .MapReduce 架構的形態

k1 與 v1 是指在執行 map 階段時的輸入 key/value pair，而 k2 與 v2 則是當 map 階段執行完畢的結果，也是以 key/value pair 的形式輸出。而在 reduce 階段 k2 與 list(v2)則是將 map 階段執行完畢的結果依照相同 key 的 value 集中起來處理，而 list(v2)則是 reduce 階段最後處理完所產生的結果資料。

接下來透過 WordCount 這個常見的例子來做說明，這個例子主要的功能就是統計單一文字出現過的次數。以下為虛擬碼：

Map 階段：

```
map(String key, String value):
    // key: document name
    // value: document contents
    for each word w in value:
        EmitIntermediate(w, "1");
```

表 2-2 .WordCount 之 Map 虛擬碼

Reduce 階段：

```
reduce(String key, Iterator values):
    // key: a word
    // values: a list of counts
    int result = 0;
    for each v in values:
        result += ParseInt(v);
    Emit(AsString(result));
```

Map 階段的 key 為文件名稱，value 為文件的內容。檢查 value 中的每一個單字，並且以每個單字為 key，其 value 皆是數值 1。直到 value 中的所有內容處理

完畢。最後輸出成中間資料。

Reduce 階段的 key 為文件中出現過的單字，values 為同一個單字的所有 value 集合。同一個單字的所有 value 作次數加總，最後輸出該單字的加總結果。

2.3 基因演算法介紹

基因演算法(Genetic Algorithms, GA)是一種模擬自然界中「適者生存，不適者淘汰」的生存進化法則，也就是靠物種不斷的演化進而產生出最能適應環境的後代。最早是由美國密西根大學的 John Holland[14]在 1975 年與其學生所共同提出的概念。目前基因演算法已經被廣泛的應用在許多範疇，如最佳化問題、資料搜尋、機器學習、人工智慧等等領域上。

此演算法是所有演化式演算法(Evolutionary Algorithms, EA)理論模式中應用領域最多、也是最廣為人知的演算法。EA 主要有三中不同的理論模式，也被稱之為 EA 的主流模式[4]，分別是演化式規劃(Evolutionary Programming)、演化策略(Evolution Strategy)與基因演化法。

由於每一個物種在自然環境中會彼此相互的競爭，只有適應力最強的物種才得以存活，並且繁衍出更加優秀的後代。因此 Holland 認為自然界中所謂的演化就是發生在生物的染色體，每一種生物的特徵資訊是來自於上一代的基因排列，因此所謂的適者生存指的就是，在同一代的染色體中較能適應環境的基因會有較大的機率可以延續到下一代的染色體。為了方便理理解基因演化常用的名詞說明如下:

生物名詞	中文譯名	意義說明
population	族群	由 N 個個體所組成的集合，N 則是指族群大小
individual	個體	等同於染色體，由問題的解答所組成
chromosome	染色體	由二進位數值或是實數所組成的串列
gene	基因	由單一或是多個 bit 所組成
genotype	基因型	對個體的資訊做基因編碼
phenotype	表現型	將基因型做解碼後所呈現出來的特徵
allele	對偶基因	將數個基因結合在一起所形成的特性
fitness function	適應函數	用來衡量某個物種對於生存環境的適應程度，通常適應函數愈高的物種會獲得較高的交配機會
selection	選擇	利用適應函數來作為挑選親代交配的依據
crossover	交配	指透過將兩親代染色體內的基因作交叉組合
mutation	突變	指將子代的個別基因做某種程度的變異

表 2-4 .GA 基本名詞說明

2.3.1 運作方式

GA 主要的概念是以操作染色體來當作自然界的演化過程，在反覆的演化過程中，就是在尋找最佳解。

首先需要將問題編碼，可以採用像是二進位制數字、實數或是文字等編碼格式。而這些字元就像是每個物種的遺傳基因 Gene。而將這些字元串接成一個字串，每個字串就代表問題的一組解，就如同自然界中的染色體(chromosome)一樣。當

把所有的染色體集合起來就是所謂的族群(population)，因此一開始需要隨機產生初始族群作為一開始演化的依據。接著將根據這些族群所面對的生存環境，設計出對環境適應能力的評估方式，也就是問題的目標函數或是其限制式，最後轉化成基因演化法的適應函數(fitness function)。適應函數將會評估各物種的適應能力，已決定哪些物種應該留下來，而哪些應該要淘汰，就像是自然界中的「物競天擇，適者生存」。適應函數評估完後，判斷是否已經滿足中止條件，若不滿足則是再透過演化過程中三個主要的運算元：複製(reproduction)、交配(crossover)以及突變(mutation)的重複演化，如此就可以達到演化的目的。

演化的流程圖與演算法如下：

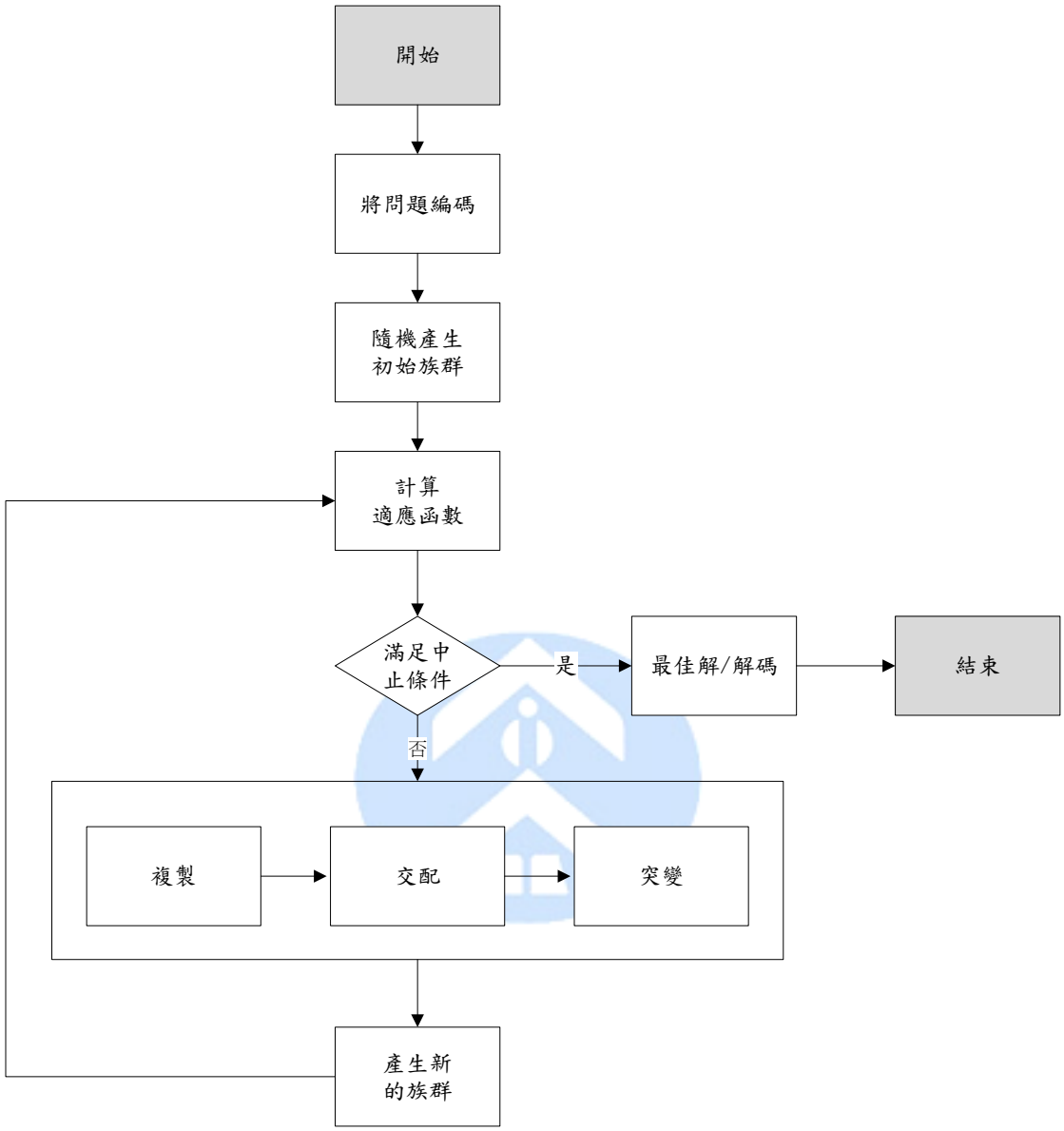

圖 2-3 .GA 的演化流程

```
begin
   n = 0;
   randomly generate initial population P(n);
   while termination not satisfied {
      repeat {
            if crossover condition satisfied {
                  select two parents p1 and p2 from P(n) based on their fitness;
                  perform crossover;
            }

            if mutation condition satisfied {
                  select chromosomes p1, p2 for mutation;
                  perform mutation to produce c1 and c2;
            }

            put c1 and c2 or p1 and p2 into P(n+1);
      } until sufficient offspring created;
      n = n + 1;

   }endwhile

end
```

表 2-5 .基因演算法虛擬碼

由上可知，基因演算法的架構主要可以分成三大部分：問題編碼、適應函數設計、基因運算元，分別於後續小節介紹。

2.3.2 問題編碼

基因編碼工作是此演算法最基礎的部分，基因會組成染色體。而由於解空間中的每一個答案就是使用染色體來代表，因此選用適合的編碼方式，可以大大的降低演化的搜尋時間以及搜尋範圍。編碼方式同時也是基因演算法的一大特點。

常使用的編碼方式有以下三種：二進位編碼、實數編碼以及符號編碼。但無論如何，即使是實數數或是符號的形式，他們底層都是使用二進制編碼。一個染色體會有多個基因，因此基因的數目就是我們所要求解問題的參數個數。如圖 2-4(a)是一個有 5 個基因的染色體，每個基因使用 2 個位元進行編碼。這是一個相當典型的二進位編碼方式。圖 2-4(b)則是使用實數編碼的方式，每一個基因對到每一個實數。圖 2-4(c)是使用符號編碼，每一個基因對照到一個符號，例如當我需要求解像是銷售員旅行問題時，就會採用符號編碼來表示每一個城市的名稱。

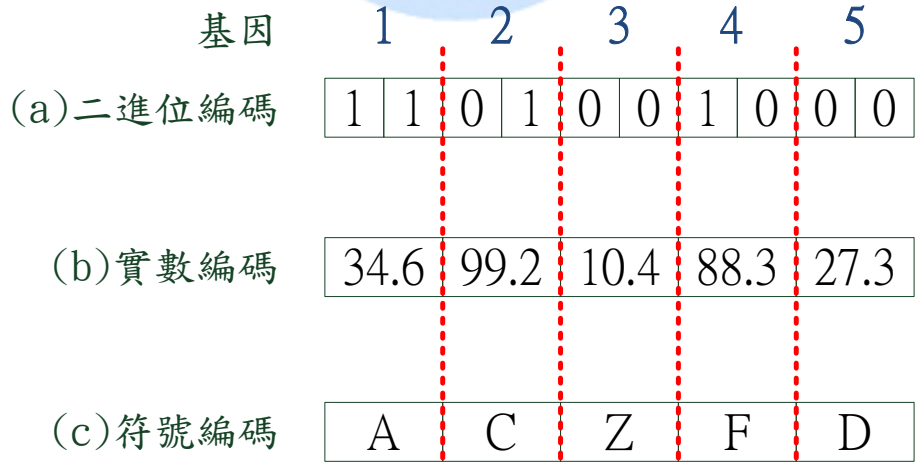

圖 2-4 .染色體的三種編碼方式

2.3.3 適應函數設計

基因演算法是採用適應度(fitness)這個概念來衡量群體中各個個體在演化的計算中最有可能達到或是接近的優良程度。適應度比較高的個體被挑選為親代的機率就會比較高。而度量這個適應度的函數就被稱之為適應函數(fitness function)，由此可知，適應函數的設定不僅直接影響了基因演算法的演化過程與結果，更是會直接影響演算法的收斂速度與能否找到最佳解。若設計的不好，容易造成產生的適應值範圍過於狹窄，而降低了染色體的多樣性，並且容易發生群體過早收斂的情況，當然過早收斂就無法得到全體的最佳解落入區域解的陷阱。

因此就需要針對適應函數作調整，而最常看到的基本調整類型有：一是靜態比例調整(static scaling)，另一是動態比率調整(dynamic scaling)[11][20][10]。最簡單的靜態比率調整是把適應函數作線性的轉換，所以有可以稱之為線性調整。而動態比率調整就是動態線性調整，也就是可以透過每一世代動態地調整參數的值。

2.3.4 基因運算子

在基因演化的過程中，最主要包含了三個基因運算子(operators)[5][6]：複製(reproduction)、交配(crossover)、突變(mutation)，分別介紹如下。

a. 複製機制

所謂的複製就是透過選擇的機制挑選適應度較好的染色體當作親代的一個過程。因此採用的選擇機制不同，得到的結果也會不一樣。一般而言，目

前常用的選擇方式有三種，分別是：隨機選擇法(random selection scheme)、分裂選擇法(disrupt selection scheme)、比例選擇法(proportionate selection scheme)。隨機選擇法就是以隨機的方式去挑選親代的染色體到子代，所以每個染色體被挑選到的機率是相同的。分裂選擇法是去除適應值介於中間地帶的染色體，而只挑選所擁有的最大與最小值部分。比例選擇法則是透過適應值的高低不同，而給予不同的選取機率。目前最常用得是輪盤法(roulette wheel selection scheme)，這是屬於比例選擇法中的一種，其作法是每一個染色體代表輪盤上的一個區塊，區塊面積愈大表示其適應值愈高，因此被挑選中的機率也就相對地提高。其概念可參考下圖。f_1 就是染色體 1 號的適應函數值佔總適應值的大小，假如把此輪盤對應成射飛鏢的遊戲，由於 f_1 所佔的面積最大，因此被射中的機率也是最高，而且被射中的次數也可能會超過一次以上。而 f_2 佔的面積最小，被射中的機率也是最低，也有可能完全不會被射中。

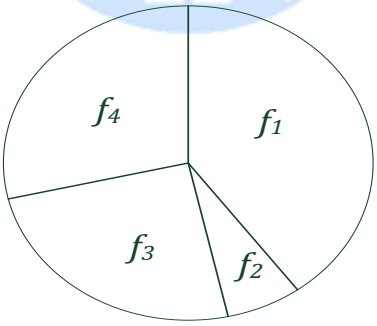

圖 2-5 .輪盤選擇法

其定義如下：

$$P_{(i)} = \frac{f_{(i)}}{\sum_{i=1}^{n} f_{(i)}}$$

其中，$f_{(i)}$代表第 i 條染色體的適應函數值，n 表是群中染色體的總數，而 $P_{(i)}$則是代表第 i 條染色體被選中的機率。

b. 交配機制

所謂的交配(crossover)間單的說是指將選擇機制所挑選出來較優的親代，透過一定的程序做染色體的基因交換動作。主要是根據交配機率(crossover probability)來判斷兩個染色體是否要進行交配。詳細的步驟為：交配過程中，隨機的選取族群中的兩個染色體，稱之為親代(parent)，進行基因的交換動作，而重新組合成兩個新的染色體，稱之為子代(child)。透過這樣的交配過程，可以把親代的優良基因交由子代延續下去，以期望可以演化出更優良的後代(offspring)。

在基因演算法中常見的交配方式有：單點交配(one-point crossover)、雙點交配(two-point crossover)、多點交配(multi-point crossover)以及均勻交配(uniform crossover)[27]。許多的研究都顯示雙點交配是比較優於單點交配，除非是當族群已經接近收斂時，這時雙點交配才會略遜於單點交配[5][6][26]。單點交配如下圖，隨機選擇兩個染色體的交配點，然後進行基因交換。

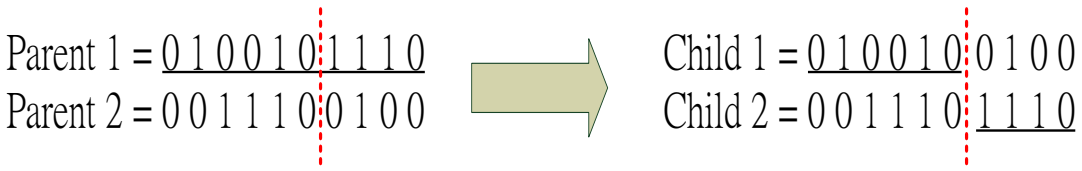

圖 2-6 .單點交配

雙點交配是指選擇染色體的兩個固定位置將串列截斷，將截斷部分進行

對換，產生結果如下圖。由雙點交配的原理可以繼續延伸出多點交配的方式。

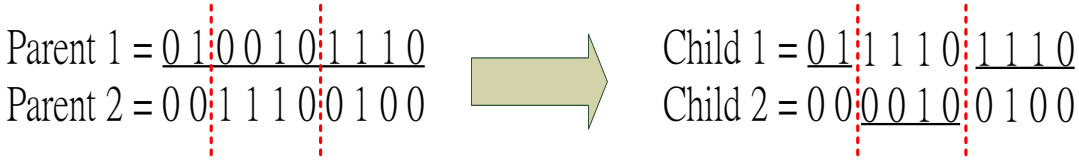

圖 2-7 .雙點交配

均勻交配則是指由亂數產生一組二進位的編碼，稱之為 Mask，以此 Mask 來與親帶比對染色體，與 Mask 編碼中 1 對應的部分就必須要對調。產生的結果如下圖：

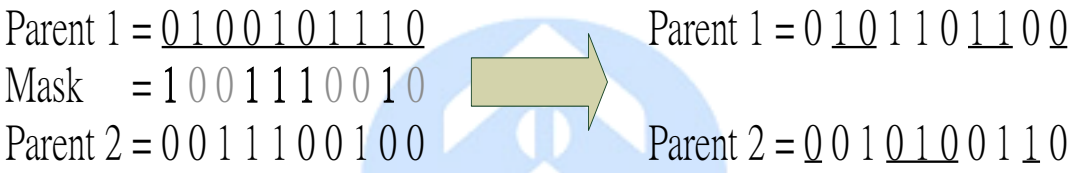

圖 2-8 .均勻交配

c. 突變機制

突變是指將染色體內的基因作隨意的變化，目的是希望可以將新的基因引入染色體中。導引染色體進入未曾尋找過的基因架構，好處是能夠跳離區域解而得到更好的結果。但是過多的突變將會導致 GA 變成隨機的演算法，而破壞了原來的結構，造成子代與親代喪失了若干的相似特徵。然而若是過少的突變則是會變成有用的基因沒有被發現，而造成陷入區域的最佳解。突變的結果如下圖：

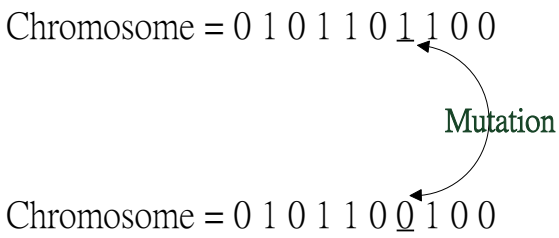

圖 2-9 .突變機制

2.4 分散式基因演算法

雖然基因演算法具有跳脫區域最佳解的能力，但是若其染色體的越長，將會導致演算法所需搜尋的範圍也隨之擴大，也就增加了找出最佳解的困難度與過早收斂等問題[16][30]，並且提高搜尋所需花費的時間。此外，若再加上此問題的複雜度較高，或是有較多的問題需要處理，那麼可想而知其演算時間將會非常的可觀。

因此之後有相當多的學者投入基因演化法的改善研究，例如採用主從式(client/server)的控制方式，由 Server 端先產生一個初始族群，並且將其分割為若干的子族群，之後再分散到各個 client 去執行基因操作，經過一段時間後，Server 端會對各個 client 所回傳回來的結果作紀錄，並且記錄是否為最佳解，若是就結束其演化流程，若不是就繼續演化。

分散式基因演算法演化流程如圖 2-9：

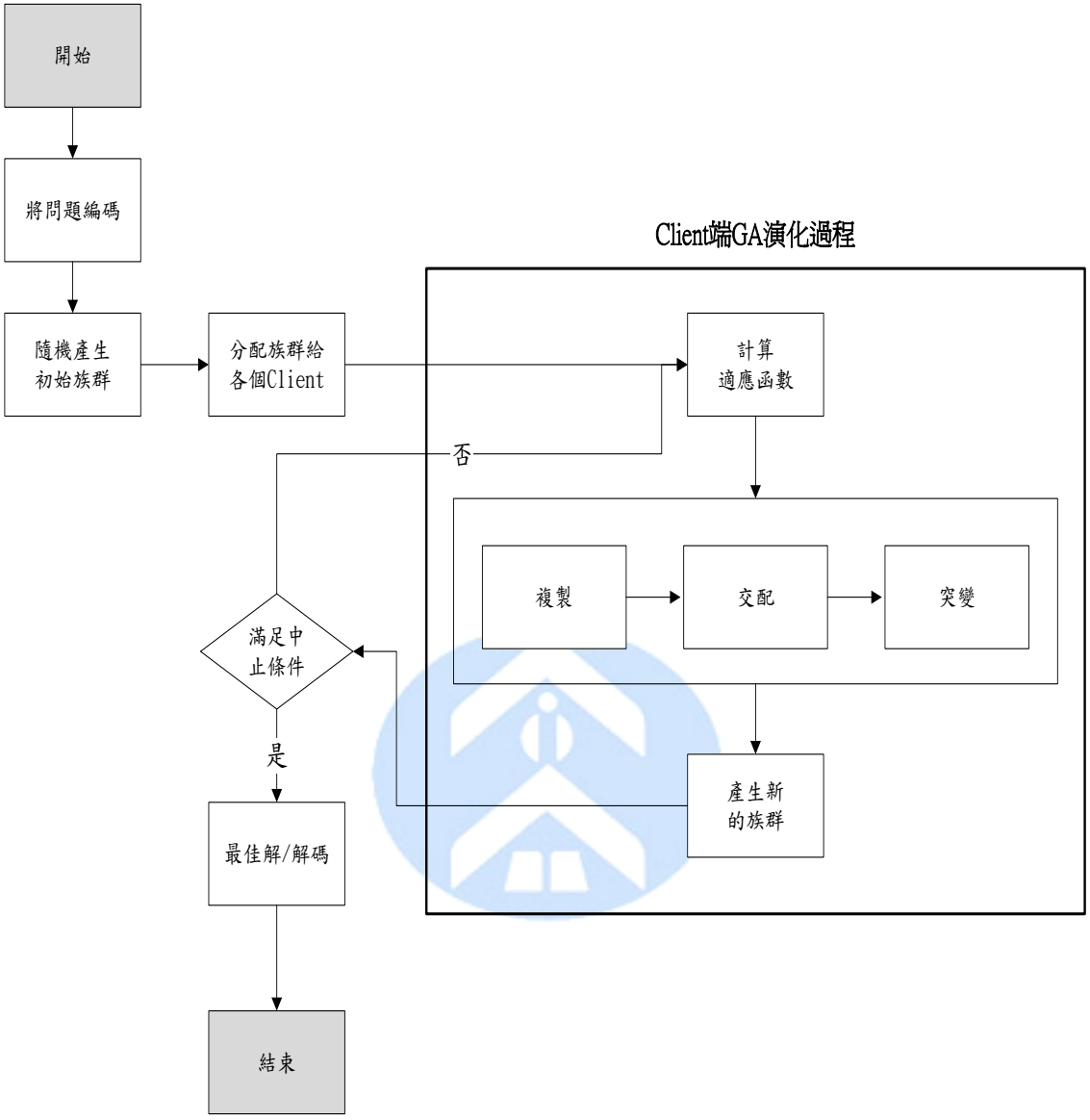

圖 2-10 .分散式基因演算法流程圖

而目前研究分散式基因演算法的研究方向大都著重於下列幾點：

1. 染色體對於演算法的影響

採用不同的遷移拓樸測試對染色體的遷入與遷出的影響，例如採用 ring 與 hypercube，並且測試多次[18][19]。

2. 隨機遷移的方式

採用隨機的遷移率來進行不同個體之間的遷移[13]。

3. 染色體遷移的選擇與取代方式

在染色體的遷出選擇方面，採用隨機選擇與最佳個體選擇方式。遷入的部分則有隨機取代與最差個體取代，並且分別進行四種不同的組合實驗[7]。

4. 動態的分散式基因演算法

採用可調整的策略，動態的調整演化時子群體的大小[30]。

尚未有研究結合雲端技術的分散式基因演算法，因此本研究將針對雲端計算提出一個結合基因演算法的架構，稱之為雲端基因(CloudGene)平台。

第三章　研究理論與方法

雖然 Google 有公開其 MapReduce 與 GFS 的設計架構，但是卻沒有公開他們所開發的 MapReduce 的軟體架構平台。因此本研究採用 Apache 基金會的 Hadoop 平台作為本研究的實驗架構。而基因演算法的部分則是採用 OpenSource 的 JGAP 套件[15]。第一節首先介紹 Hadoop 的歷史背景與所提供的應用架構，第二、三節分別介紹 Hadoop 的 HDFS 檔案系統與 MapReduce 的程式撰寫模式。第四節介紹 JGAP 套件。

3.1 Hadoop 架構概述

Hadoop 目前是 Apache 基金會所維護的一個分散式運算的架構[12]，最早是由 Dong Cutting 所開發。一開始 Dong Cutting 是在 Nutch[21]時發現無法將此搜尋引擎的效能再提升，剛好看到 Google 所提出的 MapReduce[8]與 GFS[22]概念，之後就將此概念加入 Nutch 中。後來又將此功能獨立出來成為一個新的專案，就稱之為 Hadoop。因此 Hadoop 就是以 MapReduce 與 GFS 的概念為基礎的軟體平台。也由於雲端技術相當的熱門，Hadoop 目前已經是 Apache 最 Top 的專案了。

圖 3-1 .Hadoop 的 Logo

Hadoop 的計算部分是採用 MapReduce，而底層的檔案系統也是採用 GFS 的概

念，但是名稱改為 Hadoop Distributed File System，簡稱 HDFS。Hadoop 的 MapReduce 與 HDFS 在下兩節會詳細介紹。

由於 Hadoop 架構非常的穩定，因此採用此架構的應用相當多[3]。其中 Yahoo 是最大的贊助者，也是使用最多的業者。根據 Yahoo 所公布的訊息，在 2008 年 2 月時 Yahoo 就已經建立全球最大的 Hadoop 應用架構，將其應用在 Webmap 的建立之上，而 Webmap 專案其實就是 Yahoo Search 的核心搜尋計畫[29]。詳細資料如表 3-1。

Number of links between pages in the index	roughly 1 trillion links
Size of output	over 300 TB, compressed
Number of cores used to run a single Map-Reduce job	over 10,000
Raw disk used in the production cluster	over 5 Petabytes

表 3-1 .Hadoop 應用於 Webmap

更是在 2008 年 9 月大幅擴編 Hadoop 的架構平台，不僅處理能力大幅增加，HDFS 檔案系統的資料儲存能力也達到了 16PB，這是一般的檔案系統難以達成的目標[23]。詳細資料如表 3-2。

Total Nodes	4,000
Total Cores	30,000
Data Capacity	16PB

RAM	32,000GB
Operation System	RedHat Enterprise Linux AS release4
JDK	Sun Java JDK 1.6.0

表 3-2 .Yahoo 擴大 Hadoop 架構

	500-node cluster		4000-node cluster	
	write	read	write	read
number of files	990	990	14,000	14,000
file size (MB)	320	320	360	360
total MB processes	316,800	316,800	5,040,000	5,040,000
tasks per node	2	2	4	4
avg. throughput (MB/s)	5.8	18	40	66

圖 3-2 .500-nodes vs. 4000-nodes throughput [23]

從圖 3-2 可以看出當 Hadoop 的 node 數從 500 增加到了 4,000，不僅處理的檔案數大幅增加，4,000-node 的寫入的 throughput 增加為原來的將近 7 倍，而讀取也是原來的 3.6 倍。並且每個節點所能處理的任務數也由原來的 2 個增加到 4 個。

本實驗系統開發步驟如下：

a. 定義 Map 處理程序，於程式中取得放在 HDFS 中的圖片檔案，將此圖片檔案當作參數傳給基因運算程式。

b. 定義基因運算功能，

b. 定義 Reduce 處理程序。對於 Map 程式所產生的暫存結果做進一步歸納匯整。之後再產生作最後的輸出結果，當然有 R 個 Reduce 程序就會有 R 個輸出結果。

c. 定義資料來源的 InputFormat 與 OutputFormat 格式。InputFormat 是在定義 Map 程式在接收來自 HDFS 的資料前，要以什麼樣的格式先作編排，最使用的格式為 TextInputFormat，表示輸入的格式為文字檔。OutputFormat 則是定義 Reduce 處理完資料後應該用什麼樣的格式輸出，通常是使用 TextOutputFormat。

d. 最後在 main 主程式中定義 JobConf，用來代表目前正要執行的工作屬性。包含了需要幾個 Map 以及幾個 Reduce 程序，以及輸入輸出的 HDFS 路徑。

只要做好上述幾項工作，就可以完成 Hadoop 的開發工作。而採用 Hadoop 平台架構作為雲端平台具有多項優點：

a. 容易擴增容量：假使現有 Hadoop 系統容量不敷使用，僅需簡單的更改幾項設定，即可將新機器加入有架構。底層的檔案系統資料同步與工作重新分配等問題，Hadoop 會自動幫你處理掉，完全不需要人工調整。

b. 成本較低：由 Hadoop 設計時即考慮到所使用的機器為一般的廉價電腦，因此早已把機器毀損視為正常情況，所以包含資料的毀損或是工作的重新分配等問題都已在 Hadoop 的設計內。

c. 高效率：由於資料會被分割到多台機器上，因此 Hadoop 會在儲存資料的機器上平行處理，所以處理的速度可以相當的快。

d. 可靠度高：HDFS 檔案系統會自動地維護同一份資料的多份副本，因此當有一部機器掛了，其它台機器會有副本資料，因此資料不會遺失。而正在執行的工作若因為硬體而失敗，也會自動地重新在其它台機器上重新執行。

Hadoop 的兩大核心功能，就是 HDFS 檔案系統與 MapReduce 程式模型，因此接

著我們分兩節來詳細說明。

3.2 Hadoop 的 HDFS 檔案系統

Hadoop 的 HDFS 是一個在一般電腦上執行的分散式檔案系統，並且提供了需多與一般檔案系統不同的特性。因此接下來我們將從各個角度來探討 HDFS 的優缺點以及架構。

3.2.1 HDFS 的設計目標

a. 硬體錯誤

在 HDFS 的系統中，硬體的錯誤應該要被視為正常的情況，也就是說常常會發生。因為 HDFS 檔案系統是由數百或是數千個機器所組成，而每台機器都存放一部分的檔案系統資料。所以檔案系統不能因為某個機器發生錯誤，就無法提供服務。因此 HDFS 必須要將錯誤偵測與快速且自動地恢復系統錯誤視為系統的核心架構服務。

b. 串流資料存取

必須要提供透過 HDFS 存取資料的應用程式串流式的資料存取，因為該資料量通常是非常的龐大。並且 HDFS 應設計成適合批次處理的應用，而不是用在即時的使用者回應。所以高傳輸量的資料存取是比存取資料的低反應時間，還要來的重要。

c. 大規模的資料集

儲存在 HDFS 的資料量通常都非常龐大，一般來說檔案大小從數 GB 到數

TB都是很常見的大小。因此單一大檔可能需要被分割成數以千萬計個小檔案，而且 HDFS 要能正確地控管這些小檔案。

d. 簡單的資料一致性模型

HDFS 上的檔案通常都是屬於”一次寫入，多次讀取”的模型，也就是說，當檔案一旦建立、寫入、關閉之後通常就不會再修改了。這樣的模型簡化了資料多次寫入後副本間需一致性的困擾，所以才能有比較高的資料傳輸量。

e. 移動計算比移動資料更有效率

由於儲存在 HDFS 檔案系統的資料通常都非常的龐大，因此通常是不會存放在單一機器上。而若是要將存放於各部機器上的資料移到某一台上執行，這會浪費許多的資源，所以不如將計算的功能移到各個存放資料的機器上，就地運算。那麼資料也就不需要移動了。

f. 可移植性

HDFS 的設計應該要能輕易地從一個平台移植到另外一個平台。這樣的特性將使得很多的系統都能夠採用 HDFS 為其檔案系統。

3.2.2 Namenode 與 Datanode

HDFS 是一種主從式架構的檔案系統，因此一個大的 HDFS 叢集會有一個 master 當作 Namenode，而其他的機器則是當作 Datanode。

Namenode 主要是管理檔案系統的命名空間、管理檔案存取的權限或是一個大檔被分割成幾個小檔並且儲存在哪些 Datanode 上等等稱之為 Metadata 的資料。所以 Namenode 實際上並不存放資料，但是當使用者需要存取檔案時第一步都是要先和 Namenode 連線已取得檔案的實際位置，如此一來也降低了 Namenode 變成 HDFS

的效能瓶頸的可能性。

Datanode 則是資料實際存放的地方，Datanode 除了會執行 Namenode 所要求的指令外，還需要應付使用者對資料的讀取與寫入，因為檔案會被分割成多個，並且存放在多台不同的機器上，因此可以加快資料的讀取速度。但是對於寫入則會因為資料副本的關係，而有所延誤。如下圖。

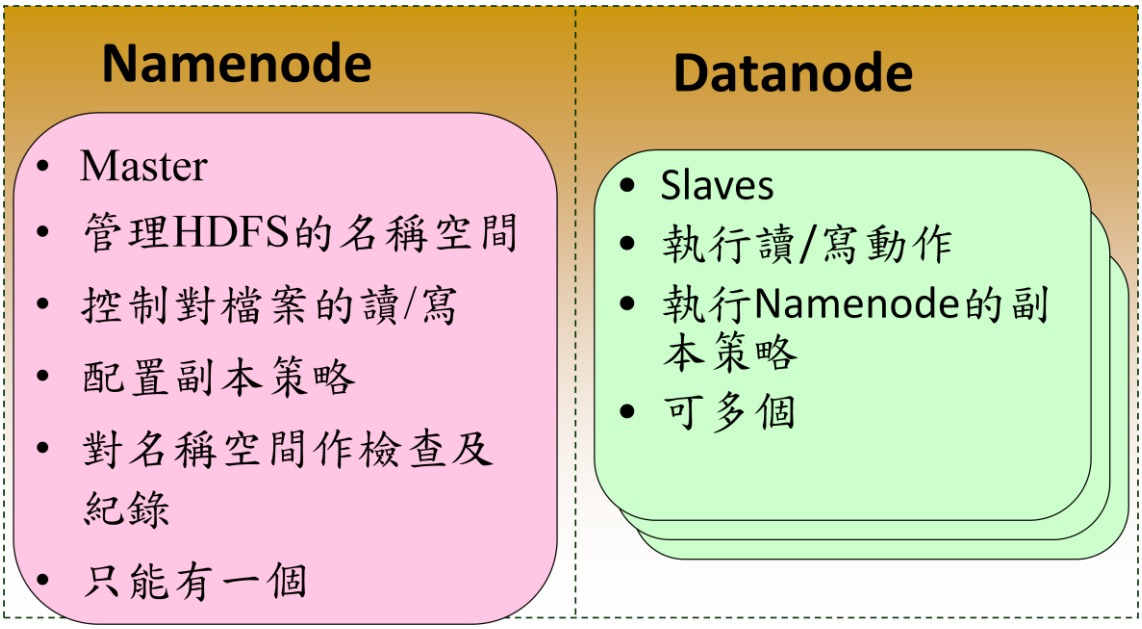

圖 3-3 .Namenode 與 Datanode 的工作分配

通常 Namenode 與 Datanode 都是運行在一般的機器上，並且執行 Linux 的作業系統。但是由於 HDFS 是使用 Java 所開發，所以任何支援 Java 的機器都可以運行 Namenode 與 Datanode，並沒有限定在 Linux 上。

圖 3-4 是 HDFS 的系統架構圖。從圖中可以看到 Client 在讀取資料前，須先與 Namenode 連線以取得檔案實際上所存放的位置，之後再向存放資料的 Datanode 做出讀取的動作，若該檔案比較大也有可能檔案會被存放在多台的 Datanode 中，那麼 Client 就可以同時對多台 Datanode 發出讀取的動作，以加快資料的讀取。

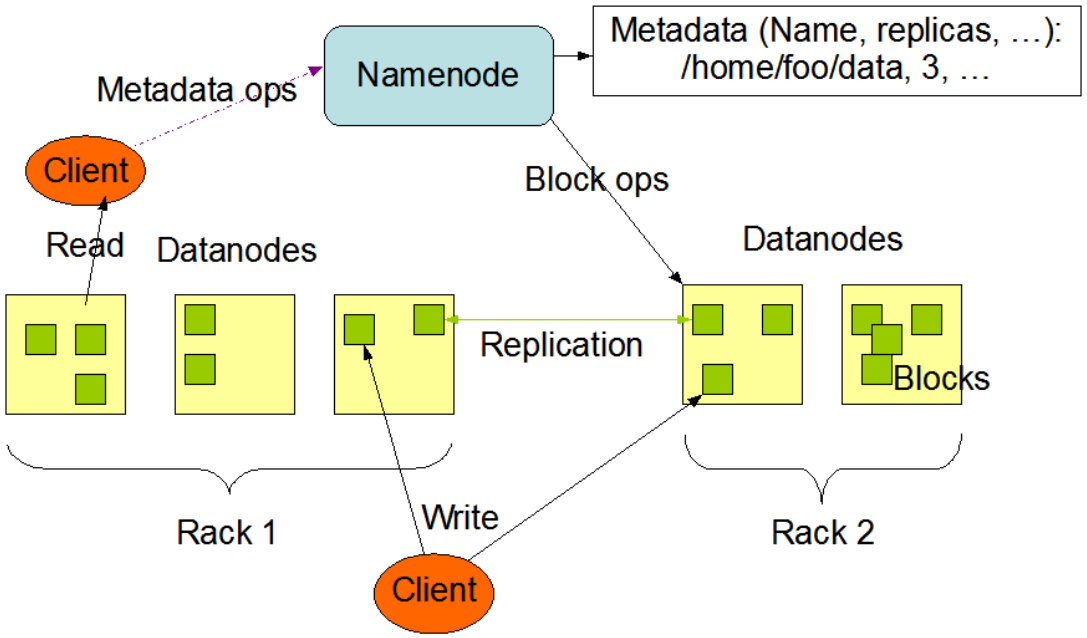

圖 3-4 .HDFS 的架構[12]

3.2.3 資料的複製

HDFS 主要是用來存放大規模的資料集，因此系統預設單一檔案若是超過 64MB 則會被分割成多個一樣大小的區塊(block)。而這些區塊為了提供容錯的機制，因此必須要有副本的策略，也就是說單一區塊會被複製到多台 Datanode 中存放。系統預設的副本數是 3，但是副本數可以由使用者建立檔案時或是之後來做調整。

如圖所示，Namenode 所控制所有區塊的複製動作。它會周期性的收到來自各台 Datanode 的 Heatbeat 與 Blockreport。Heatbeat 是表示該 Datanode 還在正常的運行中，而 Blockreport 則是列出該 Datanode 上的所有區塊檔案。

例如下圖中的 myFile 檔案的副本數為 2，區塊為 1 號與 3 號，從圖中可以看到 1 號區塊存放在 D1 與 D3 的 Datanode 中，而 3 號則是存放在 D5 與 D7 的 Datanode

中。

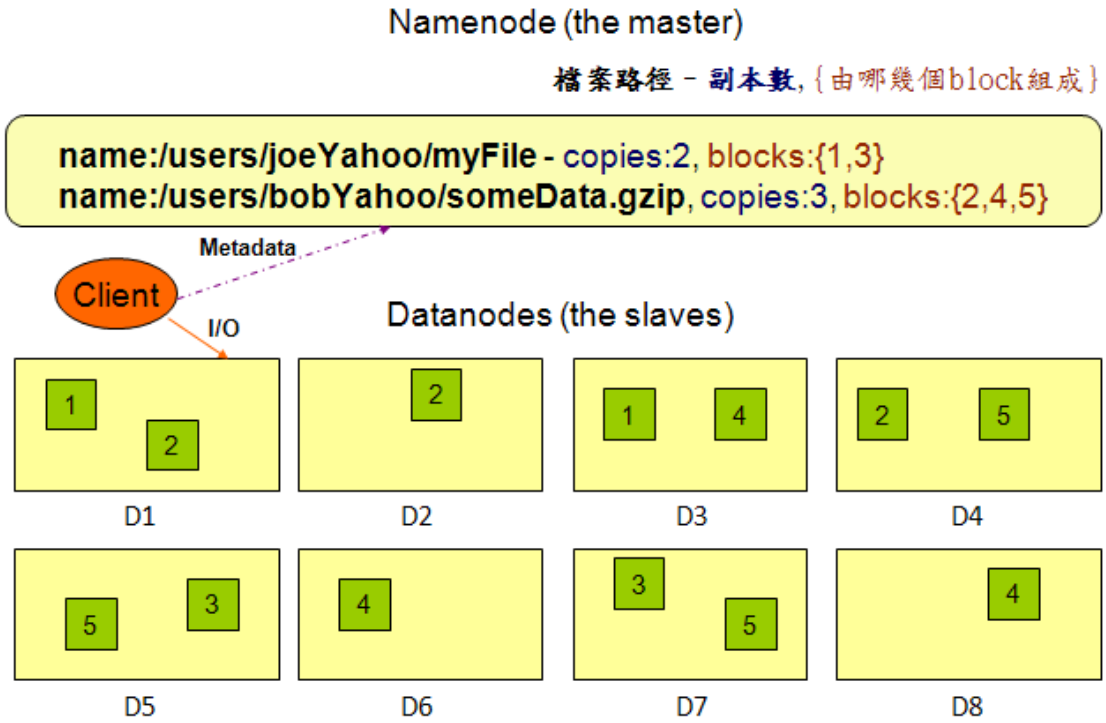

圖 3-5 .資料的複製模式 1[28]

假設 D7 的 Datanode 機器掛掉了，那麼存放在 D7 上的 3 號與 5 號資料就會遺失。這時 HDFS 的 Namenode 就會發現 3 號區塊的副本數不足 2 個，5 號區塊的副本數不足 3 個。因此就會找出區塊數比較少的 Datanode，將其他機器上的 3 號與 5 號區塊各自複製過去，以達到所設定的最低副本數。如下圖。

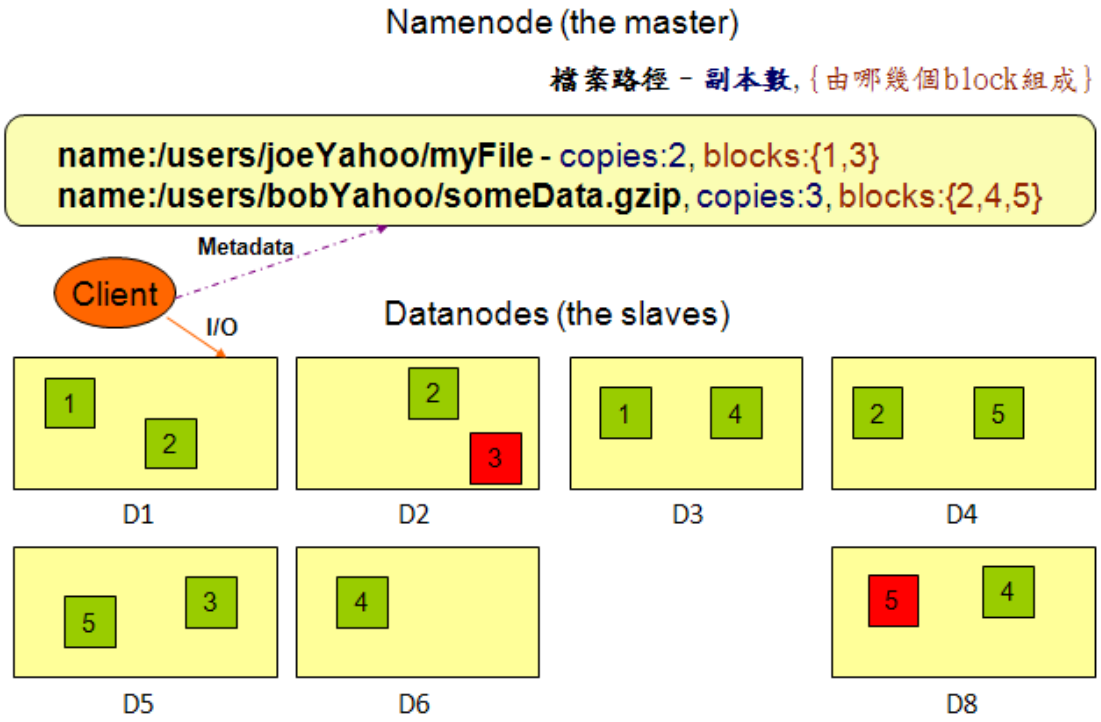

圖 3-6 .資料的複製模式 2

3.3 Hadoop 的 MapReduce 撰寫模式

MapReduce 其實就是一種 Divide 與 Conquer 的過程，適當地將問題 Divide 出來，並且透過 Map 的機制在各個機器上平行執行，最後再將 Map 產生的結果 Reduce 起來(根據某一個 Key)，以得到最終的結果。

Hadoop 中的 MapReduce 是一個實作 Google 的 MapReduce 的分散式運算架構，基於 Hadoop 所撰寫出來的應用程式能夠在上千台的機器上同時執行，並且藉由 Hadoop 平台所提供的容錯機制，開發人員可以完全不需要理會當任務失敗時所產生的錯誤，因為該平台會自動地將發生錯誤的任務重新分配給其他的機器來執

行。

如下圖。一個 MapReduce 工作(job)通常會把需要處理的資料分割成各自獨立的區塊，由 map 任務(task)以平行的方式於各台機器中處理。Hadoop 會對 map 的輸出資料先進行排序，然後再把結果當作 reduce 任務的的輸入資料，進行處理。因此有幾個 reduce 任務就會有幾個輸出結果。而不論是輸入或是輸出的資料，都是存放在 Hadoop 的 HDFS 檔案系統內。

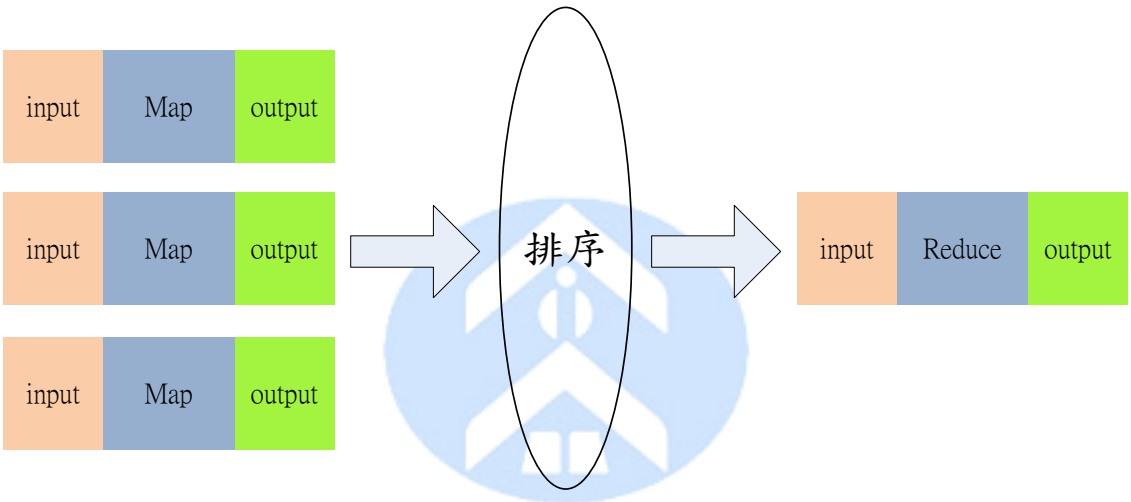

圖 3-7 .MapReduce 的輸入輸出模式

MapReduce 架構由一個獨立的 master 機器與多個 slave 機器所組成。master 又稱之 Jobtracker，主要負責執行開發人員所要執行的工作，當 Jobtracker 接收到此工作(job)時，會將此工作分成好幾個任務(task)交由底下的 slave 機器去執行。因此 JobTracker 本身並不會真正的去執行 Map 與 Reduce 程式，而是負責監控各個任務的執行狀態，假若有某台 slave 機器掛了，則 Jobtracker 就必須負責將該機器上的任務重新分配給其他的 slave 處理。而 slave 又可稱之為

Tasktracker，僅負責執行 Jobtracker 所交辦下來的任務，當任務執行完畢後再回報給 Jobtracker。Jobtracker 與 Tasktracker 的工作分配可參考下圖。

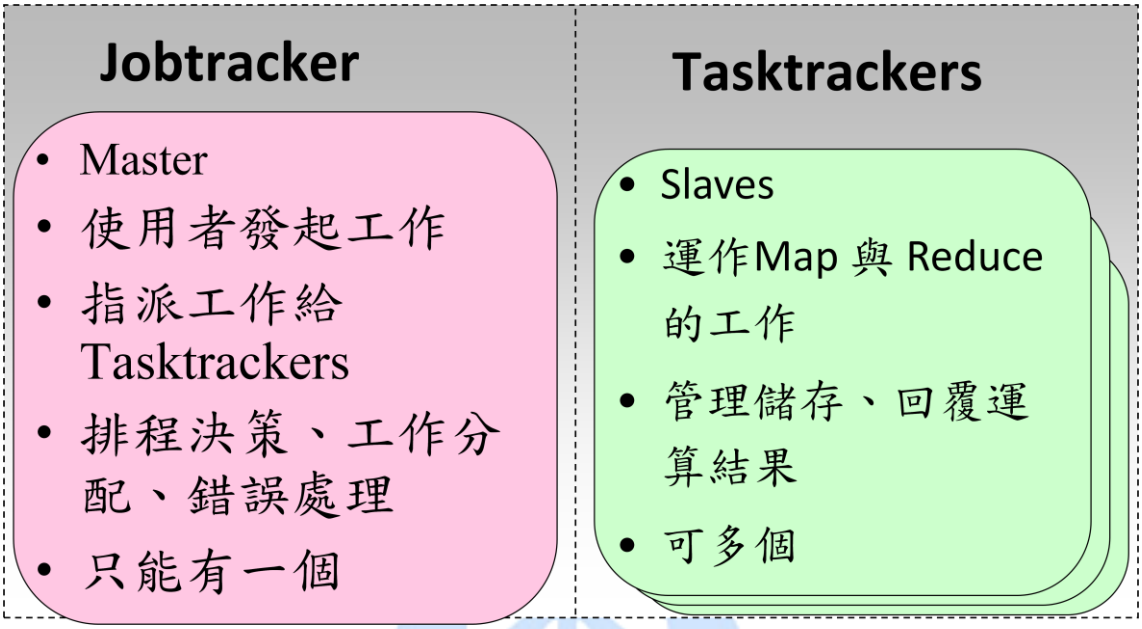

圖 3-8 .JobTracker 與 TaskTracker 的工作分配

開發人員撰寫程式時，一般都會指定程式資料在 HDFS 的輸入與輸出路徑。也會設定真正執行 Map 與 Reduce 的程式名稱。再加上其他的作業參數，就組成了工作配置(Job Configuration)。之後，Hadoop 的 job client 再送出包含程式的 jar 檔與工作配置給 Jobtracker，Jobtracker 就會負責將程式與工作配置資訊送給 slave，並且監控他們的執行狀況。

3.3.1 執行流程介紹

MapReduce 的執行流程圖如下。

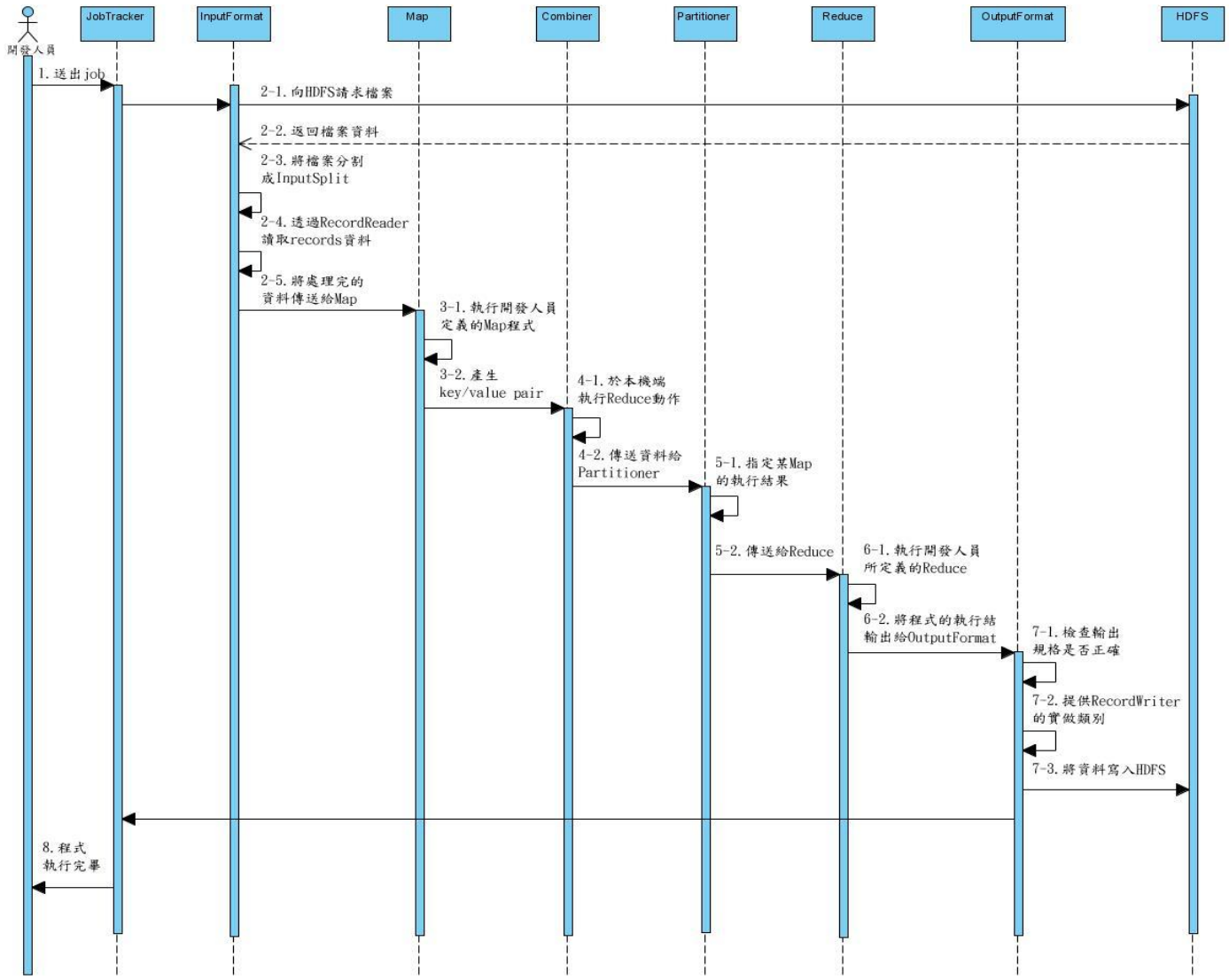

圖 3-9 .MapReduce 的執行流程圖

1. 開發人員透過命令列指令送出含有 MapReduce 程式的 jar 檔給 JobClient。
2. InputFormat 負責做 Map 的前處理，主要包含下列幾項工作：
   - 驗證輸入的格式是否符合 JobConfig 的定義，本格式可以自行定義或是繼承自 Writable 介面。
   - 將程式所要處理的資料分割成 InputSplit，每一個 map 都會有自己的 InputSplit。這是因為在 HDFS 檔案系統中，系統預設檔案大小只要超過

64MB 便會自動分割。如此分割之後所要處理的資料便可在各個機器上同時處哩，以增進執行的效能。

- 透過 RecordReader 再將經過 InputSplit 分割後的資料分成一組組的 records，用以輸出給 map 真正進行處理。

3. 將 RecordReader 處理後的資訊做為 Map 的輸入資料。然後執行開發人員所定義的程式，執行完畢後再產生 key/value pair 的資料。
4. Combiner 是選擇性的動作。主要的作用是在每一個 Map 執行完畢後，先在本機端作 Reduce 的動作，用以減少在後續 Reduce 過程中的資料傳輸量。
5. Partitioner 也是選擇性的動作。主要是在多個 Reduce 任務的情況下，指定某 Map 的執行結果要由特定的 Reduce 處理。
6. Reduce 階段則是將上階段傳過來的資料，進行簡化的動作。因此會去執行開發人員所定義的程式，並且將處理的結果輸出給 OutputFormat，而每一個 Reduce 任務就只會產生一個輸出檔。
7. OutputFormat 主要提供下列作用：
   - 檢查工作的輸出規格是否正確，例如，檢查輸出的目錄是否已經存在，若不存在則需建立。
   - 提供 RecordWriter 的實作類別，並且將 Reduce 的執行結果輸出到 HDFS 中。
8. MapReduce 程式執行完畢。

3.4 JGAP 簡介

JGAP[15]是一個 OpenSource 的基因演算法軟體，它使用 Java 所開發完成。套件裡面提供了一般基因演算法所需的基本功能，並且也包含了一個簡單的分散式基因套件，而此分散式套件則是另外使用一套 OpenSouce 的 JCGrid 軟體所完成。

3.5 系統流程

本研究率先提出了 CloudGene 雲端基因架構，並且進一步地做改良。由於本架構會大量的使用到網路頻寬，因此是屬於內部雲的架構，比較適合應用在彼此機器間具有高速網路連結的情況。下一節將會用實驗數據呈現出改良前與改良後的差異。實驗部分主要區分成兩個區塊：

實驗一： 會透過基因演算法從一個完全空白的圖演化成一張 MonaLisa 的圖片，因為此演化過程相當的耗時，所以主要是要呈現出本架構與傳統的單機與分散式基因演化在處理一個複雜度相當高的問題時，在時間上與品質上的差異點。族群數為 5，採用 LMS 方式計算適應函數，每個染色體有 2,100 個基因，演化代數則是依實驗而不同。

實驗二： 使用基因演算法來演化換零錢的動作，也就是將輸入大量的 99 分零錢去演化。本實驗的目的是顯示本架構在處理為數眾多的個別基因演化時，所程現出來的效能。族群數為 30，採用零錢數越少越好當作適應函數，每個染色體 4 個基因，每次演化 100 代，銅板種類有 25 分、10 分、5 分以及 1 分四種。

3.5.1 實驗一：MonaLisa 的圖形演化

基因演算法的部分：

其主要概念是透過隨機建立 150 個多邊形，並且透過染色體設定顏色基因與點基因。使用 LMS(Least-Mean-Square)計算 fitness 函數，計算原始圖形與新產生圖形的 RGB 差為最小，故 fitness 越小表示此多邊形與原始圖形的相似度越高，也就越適合當作父代。

每個染色體由 150 個多邊形所組成，每個多邊形會有 5 個點，使用 RGB 色彩模型，有 4 個顏色基因，前 3 個為 RGB 的值，後一個為 alpha 值，顏色基因值的範圍為 0～255。每個點有 2 個基因，分別代表寬與高，其範圍為 0～原始圖片的寬或是高。所以每個多邊形有 4(顏色基因)＋5＊2(點基因)＝14 個基因。而每一個染色體總共會有 2,100 個基因。

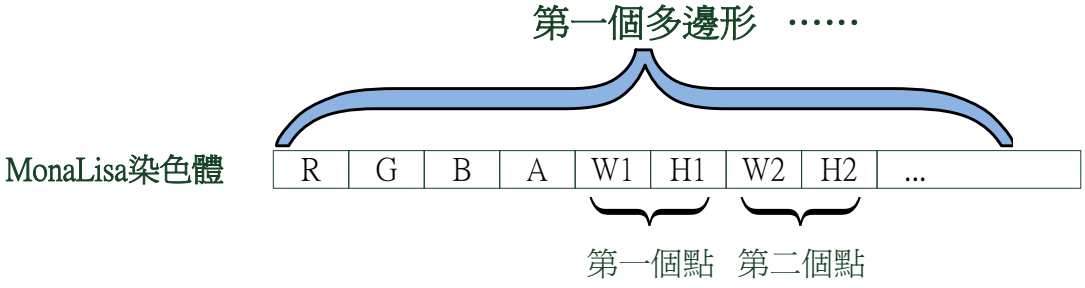

圖 3-10 .MonaLisa 染色體的組成

改良前的 CloudGene 架構：

流程說明如下：

1. 首先將要處理的圖片分割成 N 個，也就是說若要使用 N 個 Map 去執行基因演化，就必須要有 N 個切割後的圖檔。
2. 因為 CouldGene 架構所要處理的資訊都是從 HDFS 中取出，因此必須先將分割後的圖檔置入 HDFS 中，並且使用一個文字檔紀錄各個子圖的檔案名稱，方便程式存取。
3. 接下來就是在 Map 的運算中，透過 RecordReader 取出該 Map 所要處理的子圖，再交由 GA 去進行演化的部分。由於 Map 運算是在 TaskTracker 中執行，因此此部分的運算是在 16 台機器中各自獨立運行。
4. 每的 TaskTracker 所執行完的結果，會各自獨立地放入 HDFS 中。
5. 統一由單個程式取出各個 Map 的執行結果，並且進行合併後在計算其 PSNR 值。

改良前的 CloudGene 架構其執行流程，如圖 3-11。

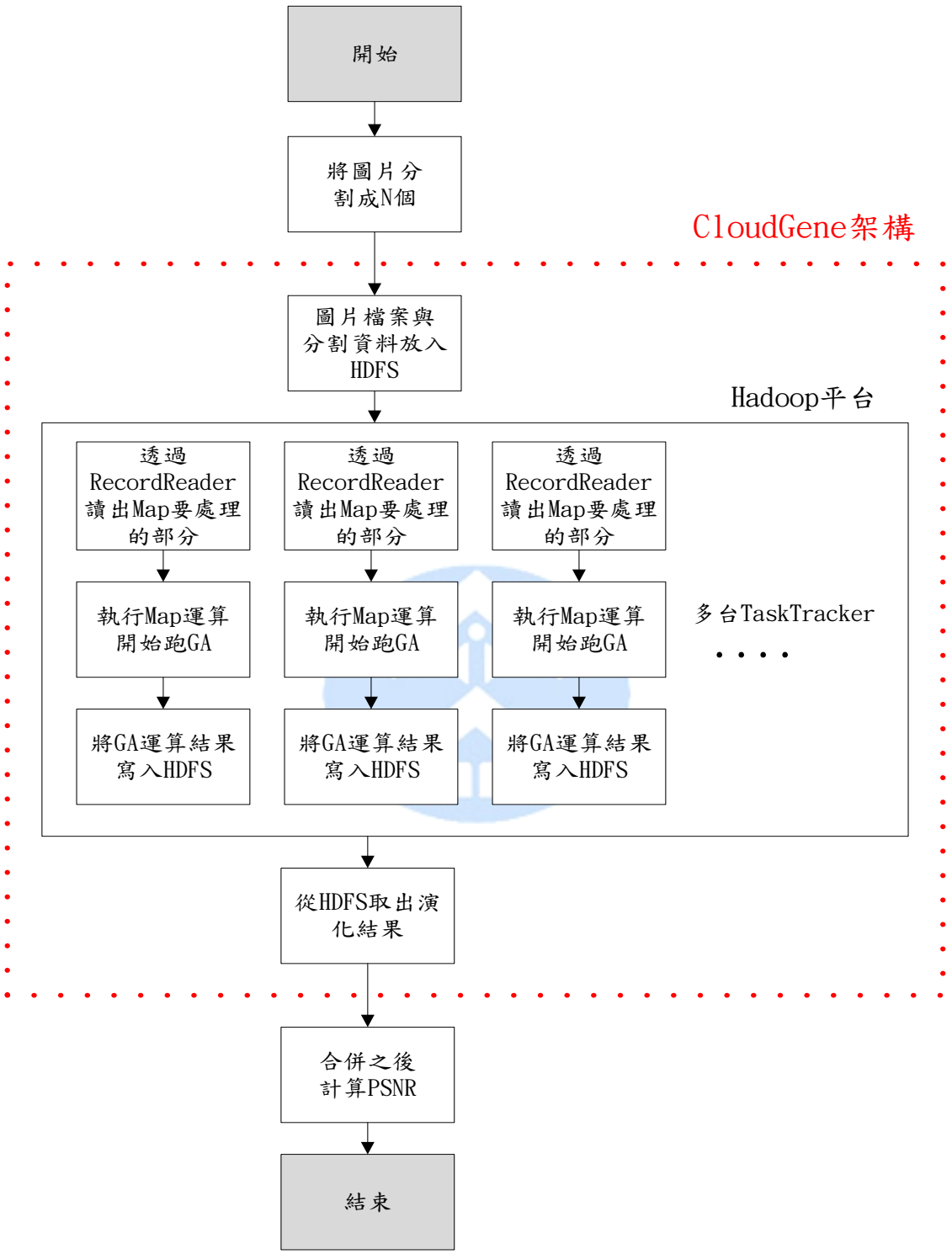

圖 3-11 .改良前 CloudGene 執行流程圖

改良後的 CloudGene 架構：

改良後的 CloudGene 架構與改良前最主要的差異是讓 Hadoop 能夠直接支援圖形的分割，也就是說原本 Hadoop 平台不支援直接將圖分割後再分散給各個 TaskTracker 去處理，而本研究額外提供了四個類別讓 Hadoop 能夠直接處理圖形資料，並且將其分散到各個 TaskTracker 中處理。

四個類別功能說明如下：

a. ImageInputFormat：主要負責定義 Hadoop 應該如何處理輸入的圖檔，像是該圖檔能不能分割、應該要如何分割等。分割出來的子圖會成為 ImageSplit 物件。
b. ImageSplit：存儲分割出來的子圖，並且交由 ImageReader 處理。而每一個 TaskTracker 都會分配到一個 ImageSplit。
c. ImageReader：此類別定義分割後的 ImageSplit 子圖，應該如何於 Map 階段處理。
d. ImageWritable：用來存放序列化(serialization)的圖檔資訊，如圖檔的大小、圖高、圖寬以及分割後圖檔的內容。透過此類別此圖檔才能夠在 CloudGene 平台下分散到各 TaskTracker 處理。

改良後的 CloudGene 其流程說明如下：

1. 不需先將圖片檔案分割，而是直接將整個圖檔放入 HDFS 中，交由程式直接處理。
2. 透過 ImageInputFormat 將使用者所放入 HDFS 中的圖形，分割成多個

ImageSplit 物件，再交由 ImageReader 來讀取。

3. 接下來就是在 Map 的運算中，透過 ImageReader 取出該 Map 所要處理的子圖，再交由 GA 去進行演化的部分。由於 Map 運算是在 TaskTracker 中執行，因此此部分的運算是在 16 台機器中各自獨立運行。
4. 每的 TaskTracker 所執行完的結果，會各自獨立地放入 HDFS 中。
5. Reduce 程式取出各個 Map 的執行結果，並且進行合併後儲存於 HDFS 中。
6. 從 HDFS 中取出已經合併完成的圖檔，並且計算其 PSNR 值。

改良後的 CloudGene 架構其執行流程，如圖 3-12。

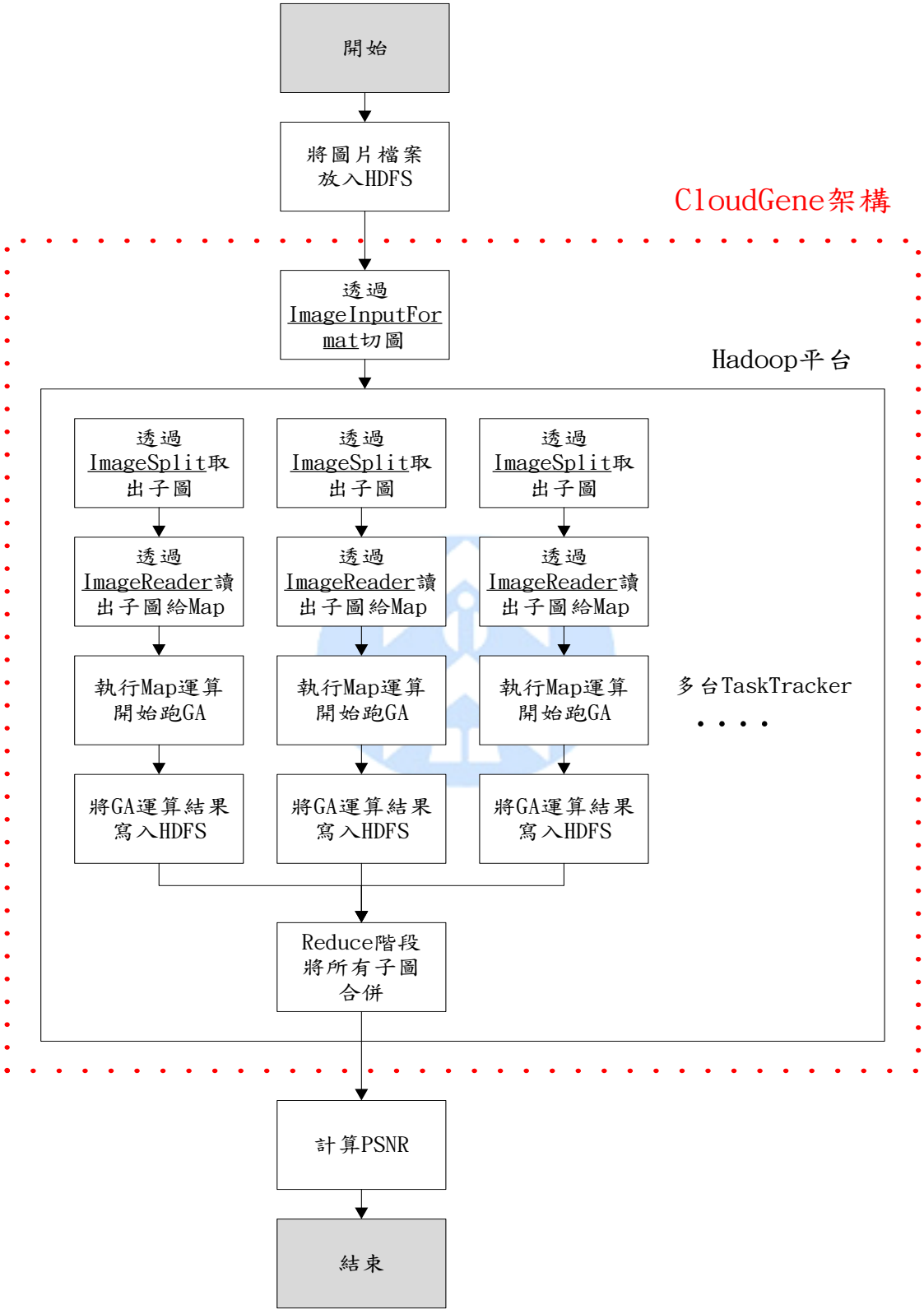
開始
將圖片檔案放入HDFS
CloudGene架構
透過ImageInputFormat切圖
Hadoop平台
透過ImageSplit取出子圖
透過ImageSplit取出子圖
透過ImageSplit取出子圖
透過ImageReader讀出子圖給Map
透過ImageReader讀出子圖給Map
透過ImageReader讀出子圖給Map
執行Map運算開始跑GA
執行Map運算開始跑GA
執行Map運算開始跑GA
多台TaskTracker
. . . .
將GA運算結果寫入HDFS
將GA運算結果寫入HDFS
將GA運算結果寫入HDFS
Reduce階段將所有子圖合併
計算PSNR
結束

圖 3-12 .改良後的 CloudGene 架構

3.5.2 實驗二：換零錢

本實驗會將大量需要換零錢的資訊，存在 HDFS 中。透過基因演算法演化所應該換出來的零錢數。使用的單位有 25 分(Quarter)、10 分(Dime) 、5 分(Nickel)以及 1 分(Penny)。fintess 函數以硬幣數越少者代表基因愈優良。

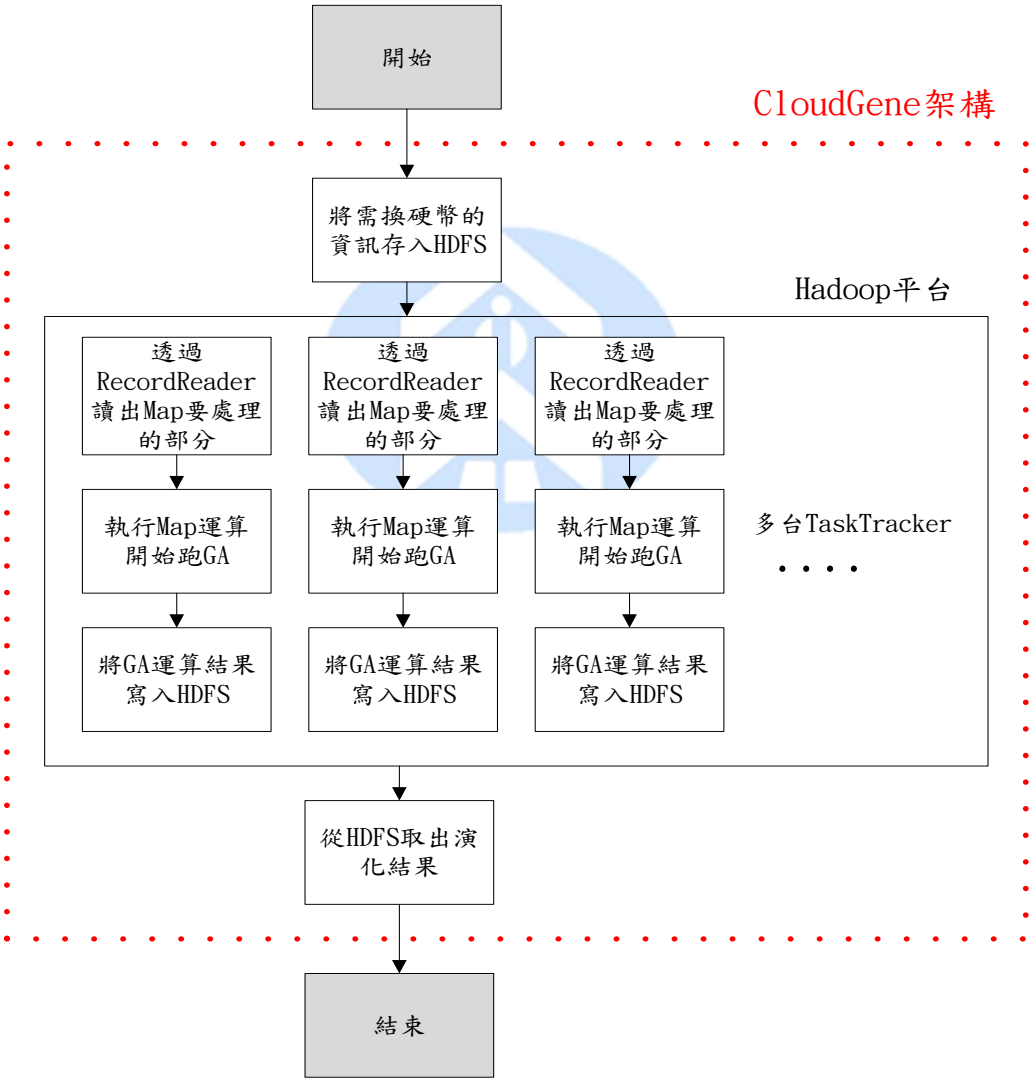

圖 3-13 .換硬幣流程圖

第四章　實驗結果與分析

4.1 系統環境配置

本實驗總共架設 16 台相同硬體設備的電腦，2G 的記憶體，實體機器的環境配置如表 4-1。由於目前雲端架構多採用虛擬機器(Virtual Machine, VM)[24] [25] [17]，而採用 VM 最大的好處便是易於管理。因此本研究也採行 VM 的架構，所使用的虛擬機器軟體是 VMware Workstation 7。每部實體機器上皆會運行一個虛擬機器。虛擬機器的環境配置如表 4-2。由於實體機器為雙核心，因此會分配一個 CPU 給 VM 使用。

名稱	規格
作業系統	Windows XP
CPU	Intel Core 2 Duo E4700　2.6GHz
記憶體	2G
硬碟空間	100G
VM 軟體	VMware Workstation 7

表 4-1 .實體機器環境配置

名稱	規格
作業系統	Ubuntu 8.0.4
分配 CPU	1 CPU
分配記憶體	1G
硬碟空間	100G
JDK	1.6
Hadoop	0.19.1
JGAP	3.4.4

表 4-2 .虛擬機器環境配置

在 16 個 VM 中，全部都會成為 DataNode 與 TaskTracker。其中 hadooper-vm10 除了負責 DataNode 與 TaskTracker 外，另外還要當作 NameNode 與 JobTracker。IP 配置如下：

hadooper-vm10 ： 192.168.1.10

hadooper-vm11 ： 192.168.1.11

hadooper-vm12 ： 192.168.1.12

．．．．

hadooper-vm22 ： 192.168.1.22

hadooper-vm23 ： 192.168.1.23

hadooper-vm24 ： 192.168.1.24

hadooper-vm25 ： 192.168.1.25

本研究所提出的 CloudGene 系統架構圖，如下：

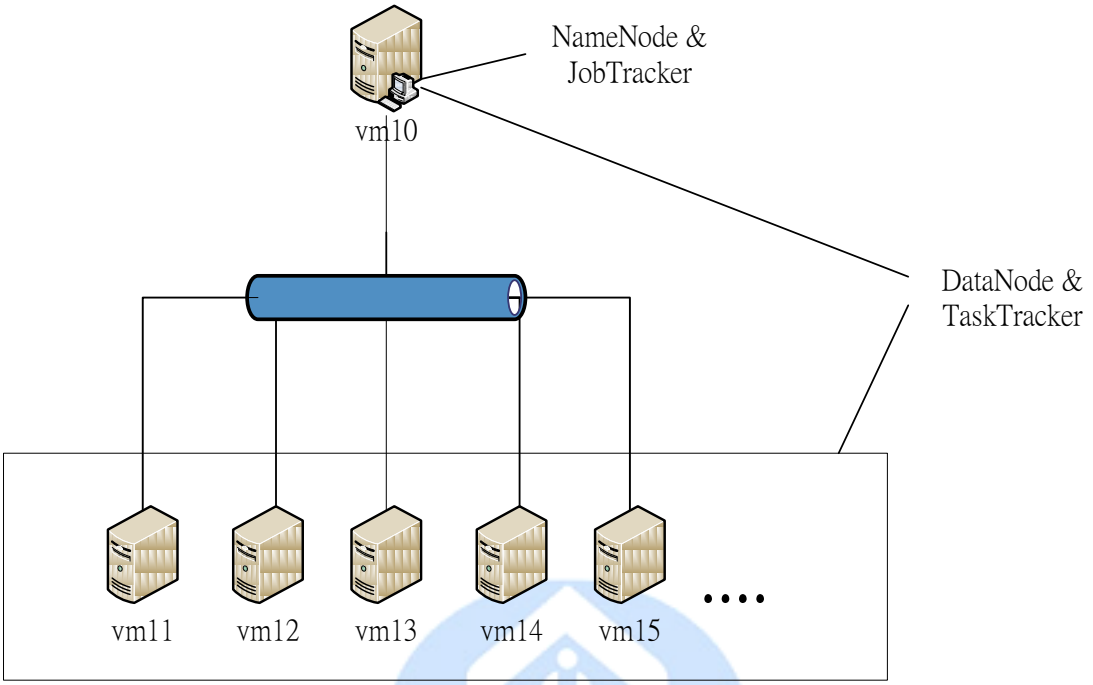

圖 4-1 .CloudGene 系統架構圖

為了讓 Hadoop 平台能夠順利運行，需進行下列設定：

a. 設定 hosts

因為各個 vm 會互相通訊，必須要讓每個 vm 的都能正確地解析主機名稱與 IP 位址，所以必須在/etc/hosts 內設定主機名稱與 IP 位址對照。設定如下：

```
127.0.0.1       localhost           localhost
192.168.1.10    hadooper-vm10       hadooper-vm10
. . . .
192.168.1.21    hadooper-vm22       hadooper-vm22
192.168.1.22    hadooper-vm23       hadooper-vm23
```

```
192.168.1.23    hadooper-vm24    hadooper-vm24
192.168.1.24    hadooper-vm25    hadooper-vm25
```

b. SSH 設置

由於 Hadoop 各個節點彼此之間是透過 SSH(Secure Shell)來做溝通，像是啟動或是關閉 DataNode。而節點之間執行指令是不需要輸入密碼的，因此我們需要先設置 SSH 使用無密碼公鑰認證。設置指令如下：

```
ssh-keygen  -t  rsa  -f  ~/.ssh/id_rsa  -P  ""
cp  ~/.ssh/id_rsa.pub  ~/.ssh/authorized_keys
```

產生完畢後須將/.ssh/目錄下的所有檔案複製其他 15 台 vm 中。完成後可以透過下列指令得知設置是否成功。若成功就可以不需要輸入密碼即可登入其他台 vm。

```
ssh  hadooper-vm11
```

c. 設置 hadoop-env.sh

由於 hadoop 是使用 Java 所開發，因此需再 hadoop-env.sh 中設定 JDK 的安裝路徑。除了 JDK 外，也需設定 hadoop 的安裝目錄，以本實驗來說設定如下：

```
export  JAVA_HOME=/usr/lib/jvm/java-6-sun
export  HADOOP_HOME=/opt/hadoop-0.19.1
export  HADOOP_CONF_DIR=/opt/hadoop-0.19.1/conf
```

d. 設置 hadoop-site.xml

hadoop-site.xml 對於 Hadoop 平台來說相當的重要，主要用來設定 NameNode 與

JobTracker 分別是在哪一台機器上以及所使用的 Port Number。本實驗的 NameNode 與 JobTracker 都是在 hadooper-vm10 上，而 Port Number 分別是 9000 與 9001。除了這兩個主要設定外，其他系統的細部設定也是在此檔，像是 hadoop.tmp.dir 就是在設定 hadoop 的暫存資料夾的位址。

```
<configuration>
  <property>
    <name>fs.default.name</name>
    <value>hdfs://hadooper-vm10:9000/</value>
  </property>
  <property>
    <name>mapred.job.tracker</name>
    <value>hadooper-vm10:9001</value>
  </property>
  <property>
    <name>hadoop.tmp.dir</name>
    <value>/var/hadoop/hadoop-${user.name}</value>
  </property>
</configuration>
```

e. 設置 masters 與 slaves

masters 是用來設定哪台機器要當 NameNode 與 JobTracker，而 slaves 則是設定哪些是 DataNode 與 TaskTracker。

masters 設定如下：

```
hadooper-vm10
```

slaves 設定如下：

```
hadooper-vm10
hadooper-vm11
．．．．
hadooper-vm24
hadooper-vm25
```

f. 格式化 HDFS

安裝完成的 Hadoop 架構，在啟動前必須要先對 HDFS 作格式化，概念上就像我們安裝作業系統前必須要先對硬碟格式化一樣。透過下列指令即可將 Hadoop 格式化。

```
bin/hadoop  namenode  -format
```

g. 啟動 Hadoop

一般情況下，我們只需透過 start-all.sh 即可將我們設定在 masters 與 slaves 檔案中的所有機器啟動，毋須人工逐一啟動。但除了 start-all.sh 外 Hadoop 尚提供其他常用指令，說明如下：

檔案名稱	功能
start-all.sh	啟動所有機器的 Hadoop 環境，包含 NameNode、DataNode、JobTracker 與 TaskTracker。
stop-all.sh	關閉所有機器的 Hadoop 功能。
start-mapred.sh	啟動 MapReduce 執行環境，包含 JobTracker 與 TaskTracker。
stop-mapred.sh	關閉 MapReduce 功能。
start-dfs.sh	啟動 HDFS 執行環境，包含 NameNode 與 DataNode。
stop-dfs.sh	關閉 HDFS 功能。

表 4-3 .Hadoop 常用功能

4.2 實驗結果與討論

由於基因演算法每次演化所得到的結果會有所不同，因此以下的實驗數據為同樣的實驗運行 10 次後所得到的平均值。在 16 台機器的情況下，得到下列結果：

實驗一：MonaLisa 的圖形演化

首先我們先測試改良前的 CloudGene 的架構，分別比較在 2、4、8、16 個 Map 的情況下以及基因演化代數為 1K~10K 的環境下作圖片 PSNR 值與所花費時間(秒)的情況。從圖表中可以看出，不管是 PSNR 值或是耗費時間在 16 個 Map 時，皆有最好的演化品質與耗費時間最少的表現。誠如先前所提，基因演算法每次演化的結果會不一致，因此在表 4-4 可以看到在 2 Map 的情況下 7K 代數所花費的時間比 6K 還要短，或是 4 Map 的情況下 6K 的 PSNR 值比 7K 還要高，皆屬於正常的演化現象。

CloudGene	2 Map		4 Map		8 Map		16 Map	
代數	PSNR 值	時間(秒)	PSNR 值	時間(秒)	PSNR 值	時間(秒)	PSNR 值	時間(秒)
1K	14.5591	465	14.5859	283	15.4544	192	14.6303	186
2K	16.2303	946	16.6819	564	17.5808	414	17.6659	399
3K	16.8018	1361	18.7506	854	18.8111	653	19.3284	598
4K	18.5213	1614	19.0945	1115	20.0894	889	20.5417	796
5K	19.3227	1979	20.2069	1405	21.5127	935	22.1102	966
6K	19.4444	2803	21.7961	1710	21.7961	1236	22.8545	1163
7K	20.3132	2723	21.7559	2052	22.3102	1315	23.5520	1403
8K	20.6489	3189	21.3003	2254	23.1477	1614	23.9148	1560
9K	21.2913	3627	22.4254	2582	23.0254	1680	24.2926	1670
10K	22.2537	4075	22.4841	2866	23.4919	2142	24.6090	2043

表 4-4 .改良前 CloudGene 架構下基因代數與 Map 數的關係

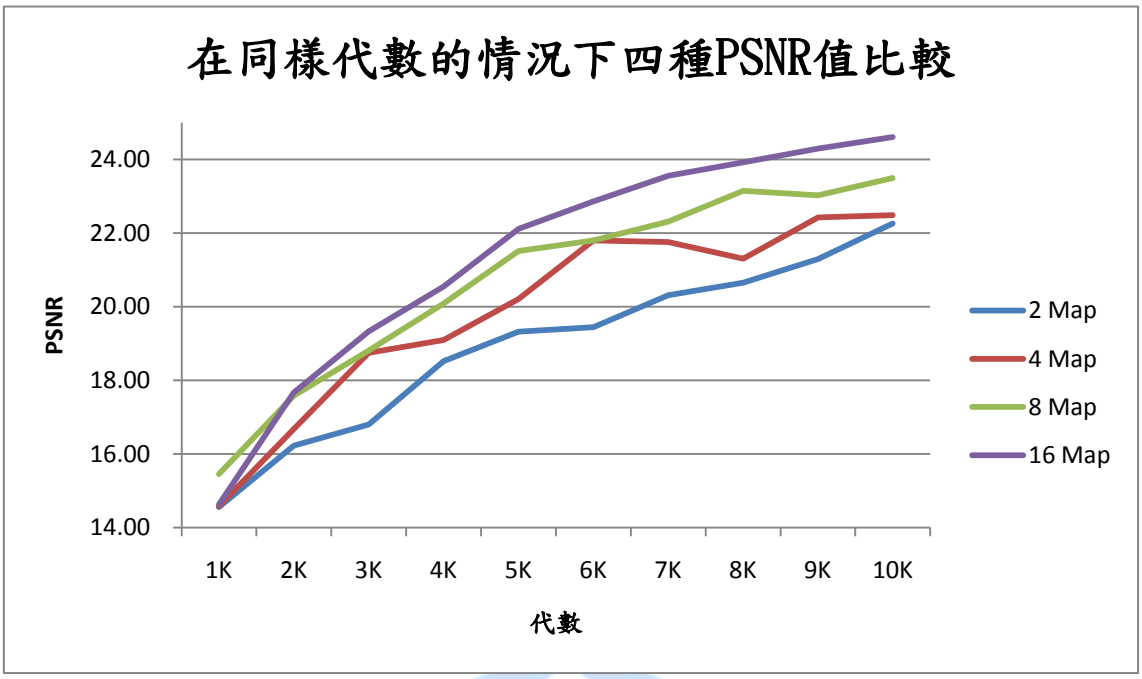

圖 4-2 .改良前 CloudGene 架構下四種 Map 數的 PSNR 值比較

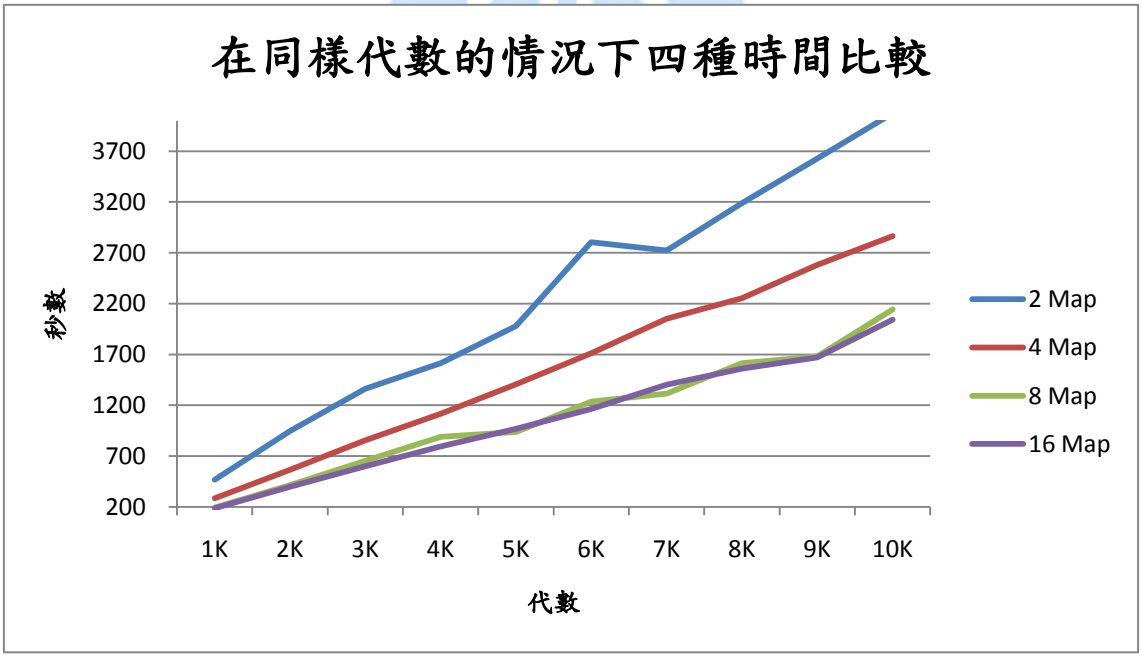

圖 4-3 .改良前 CloudGene 架構下四種 Map 數的耗時比較

接著測試改良後的 CloudGene 的架構，分別比較在 2、4、8、16 Map 以及基因代數為 1K~10K 的環境下作圖片 PSNR 值與所花費的時間(秒)的情況。從圖表中可以看出，不管是 PSNR 值或是耗費時間在 16 個 Map 時，比其他的 Map 數皆有比較好的品質與效能。不過若與改良前的架構相比其 PSNR 值較少一些，但是其效能又比改良前快 50%以上。

CloudGene	2 Map		4 Map		8 Map		16 Map	
代數	PSNR 值	時間(秒)	PSNR 值	時間(秒)	PSNR 值	時間(秒)	PSNR 值	時間(秒)
1K	14.3269	359	14.7762	185	14.8287	188	15.8881	159
2K	15.9665	565	16.5927	355	17.1617	297	18.1308	231
3K	17.1444	865	18.4694	536	18.7707	357	18.8264	307
4K	18.9821	1151	18.6614	739	19.4916	468	19.7297	347
5K	19.5095	1422	19.1162	977	20.1099	682	20.2989	430
6K	19.5035	1985	20.3929	1264	20.3132	803	20.6801	512
7K	20.6645	2009	20.6957	1391	20.6879	953	20.9069	582
8K	20.8962	2356	20.7114	1405	21.1274	1015	21.2624	712
9K	21.0136	2555	20.8191	1598	21.3602	1230	21.4636	792
10K	21.0407	2801	21.0242	2119	21.4659	1246	21.9238	825

表 4-5 .改良後 CloudGene 架構下基因代數與 Map 數的關係

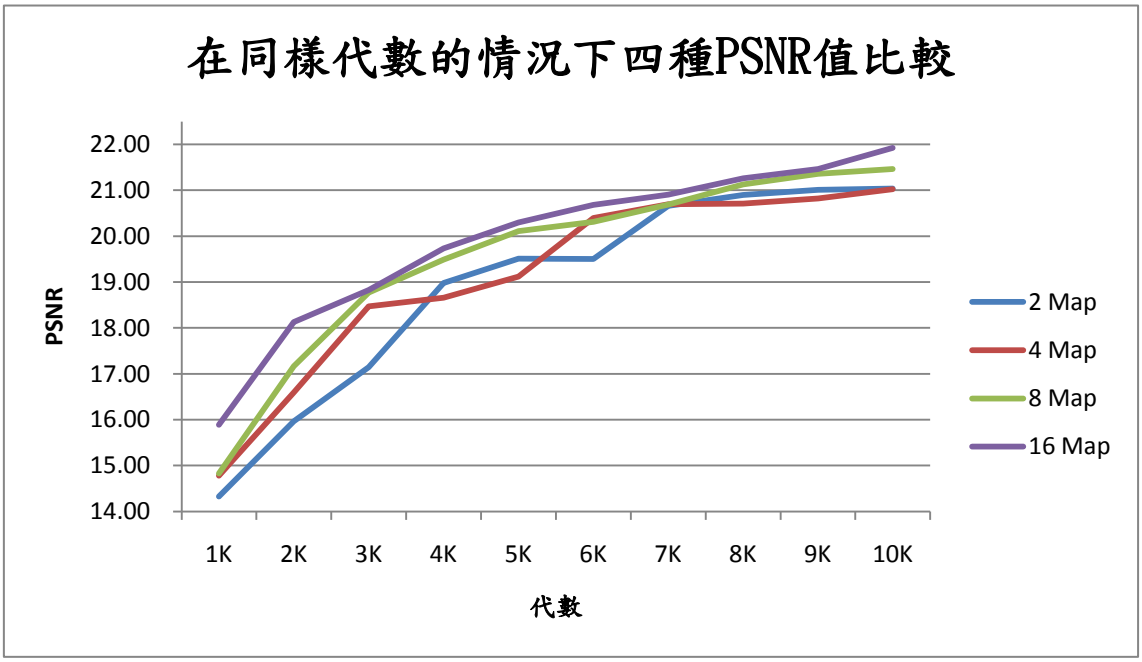

圖 4-4 .改良後 CloudGene 架構下四種 Map 數的 PSNR 值比較

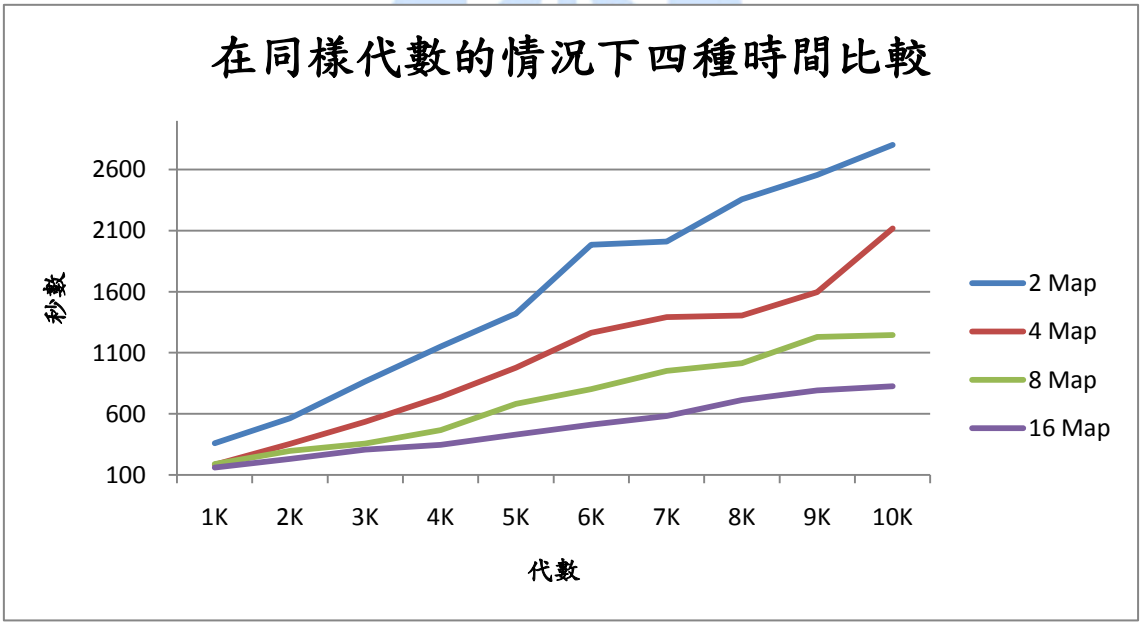

圖 4-5 .改良後 CloudGene 架構下四種 Map 數的耗時比較

接著取出改良前後最優的 CloudGene 數值與傳統的分散式基因架構(JGAP)以及單機版的 GA 作比較，在 PSNR 的品質方面品質差異並不會太大，但是時間上面則是有比較大的效能改善。推測 JGAP 的效能提升不是那麼明顯的主因，應該是因為耗費太多時間於資料的接收與傳輸上面，導致整理效能不彰。

	CloudGene New		CloudGene		單機		JGAP	
代數	PSNR 值	時間(秒)	PSNR 值	時間(秒)	PSNR 值	時間(秒)	PSNR 值	時間(秒)
1K	15.8881	159	14.6303	186	13.2397	641	14.8490	492
2K	18.1308	231	17.6659	399	15.0366	1210	16.3067	974
3K	18.8264	307	19.3284	598	16.7257	1988	17.2815	1462
4K	19.7297	347	20.5417	796	17.7446	2432	17.8451	1957
5K	20.2989	430	22.1102	966	18.8828	3121	18.3719	2417
6K	20.6801	512	22.8545	1163	19.2043	3586	18.5739	2899
7K	20.9069	582	23.5520	1403	18.8520	4124	19.4679	3525
8K	21.2624	712	23.9148	1560	19.7047	4919	19.6736	3939
9K	21.4636	792	24.2926	1670	20.1305	5463	20.1305	4347
10K	21.9238	825	24.6090	2043	20.7509	6101	20.4672	4876

表 4-6 .CloudGene 與其他架構比較

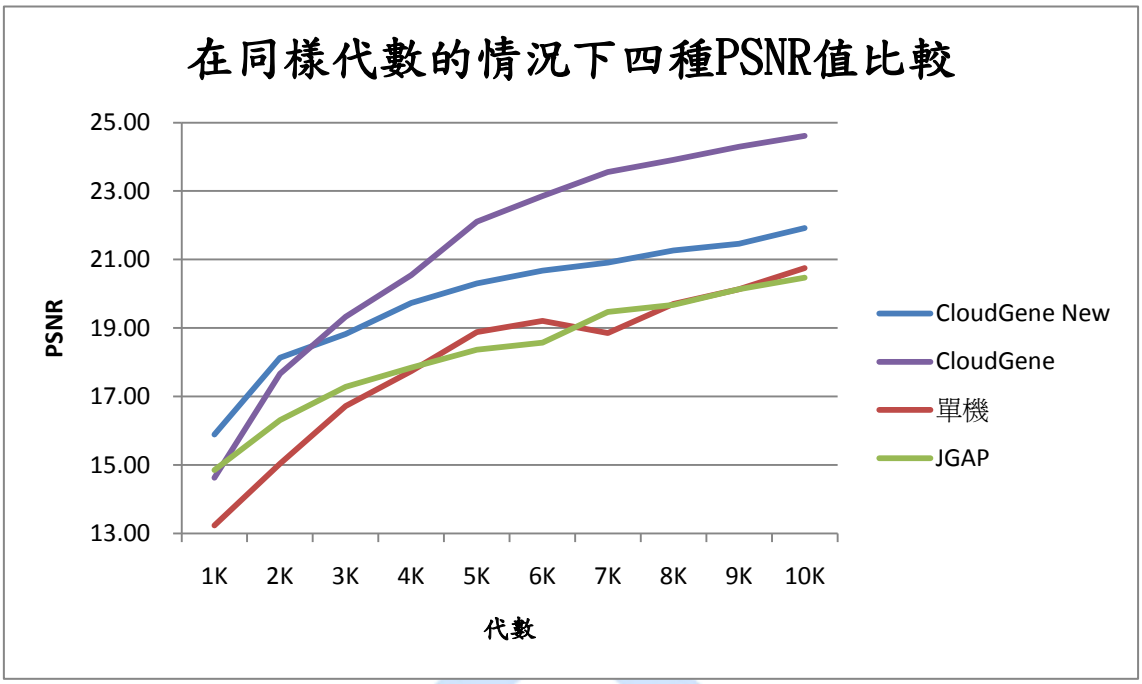

圖 4-6 .CloudGene 與其他架構的 PSNR 值比較

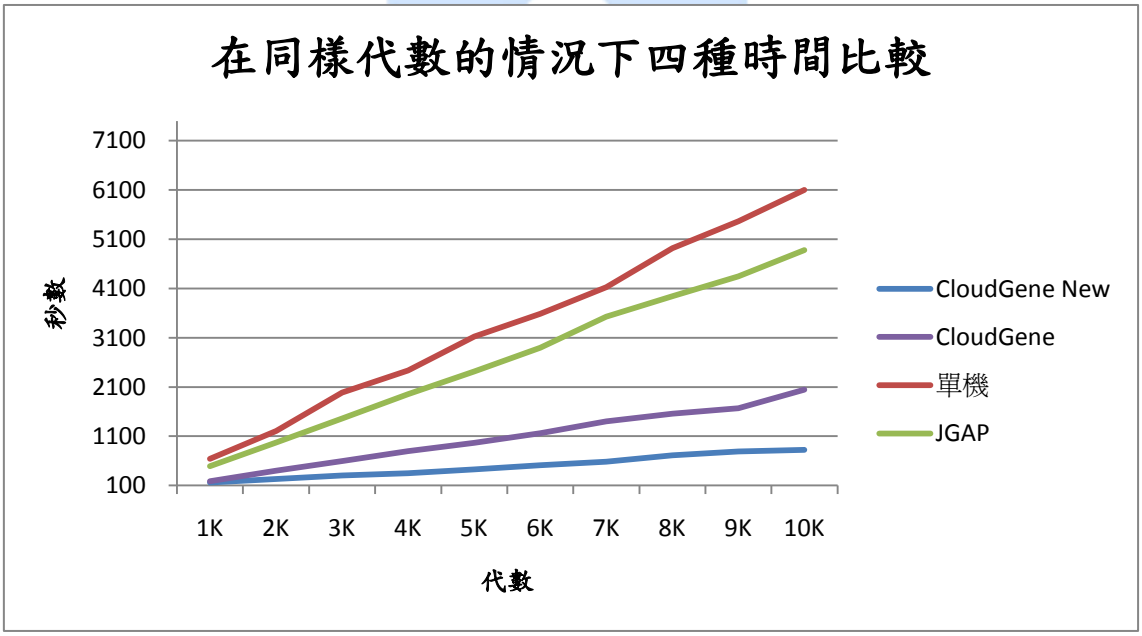

圖 4-7 .CloudGene 與其他架構的的耗時比較

改良前的 CloudGene 架構在一萬代時效能比 JGAP 快上一倍，而改良後的 CloudGene 更是快上將近五倍。若與單機版比較效能更是有顯着的提升。圖 4-12 是原始圖片。

圖 4-8 .MonaLisa 的原始圖片

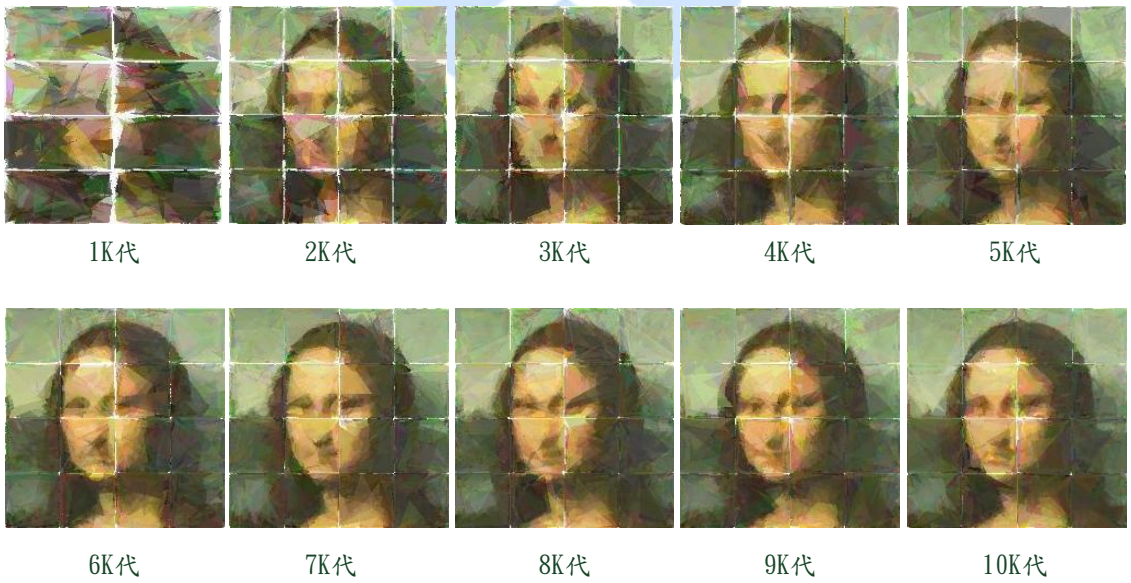

圖 4-9 .改良前的 CloudGene 架構 16Map 於不同代數的演化結果

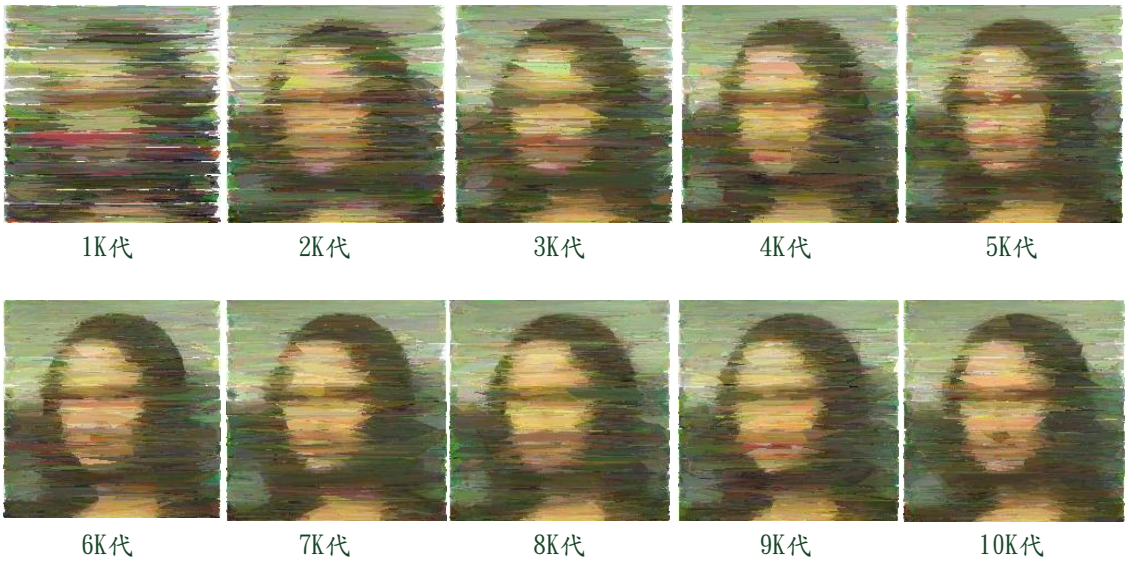

圖 4-10 .改良後的 CloudGene 架構 16Map 於不同代數的演化結果

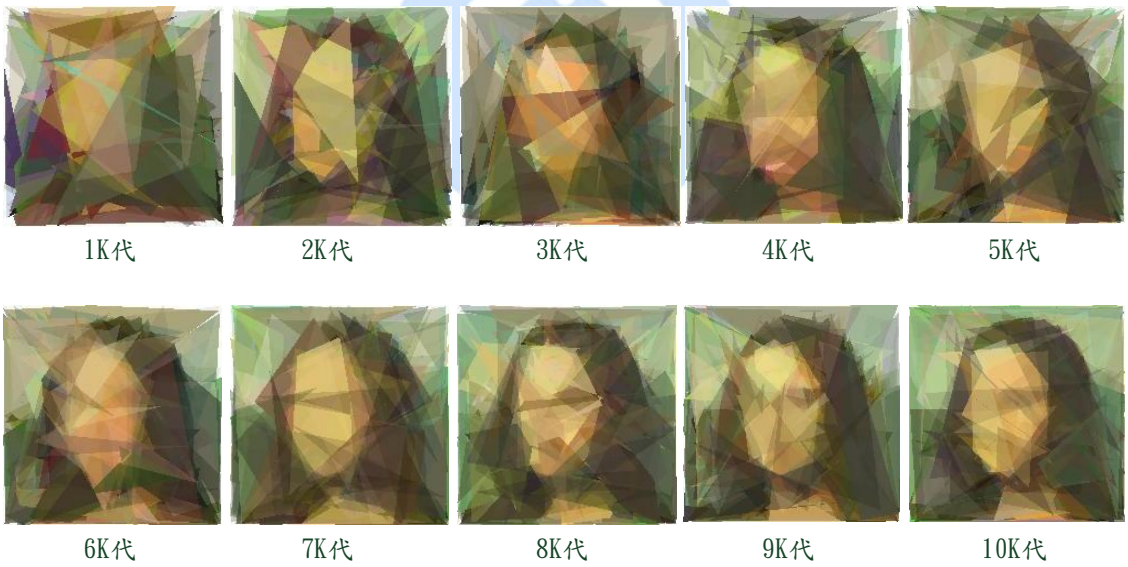

圖 4-11 .JGAP 於不同代數的演化結果

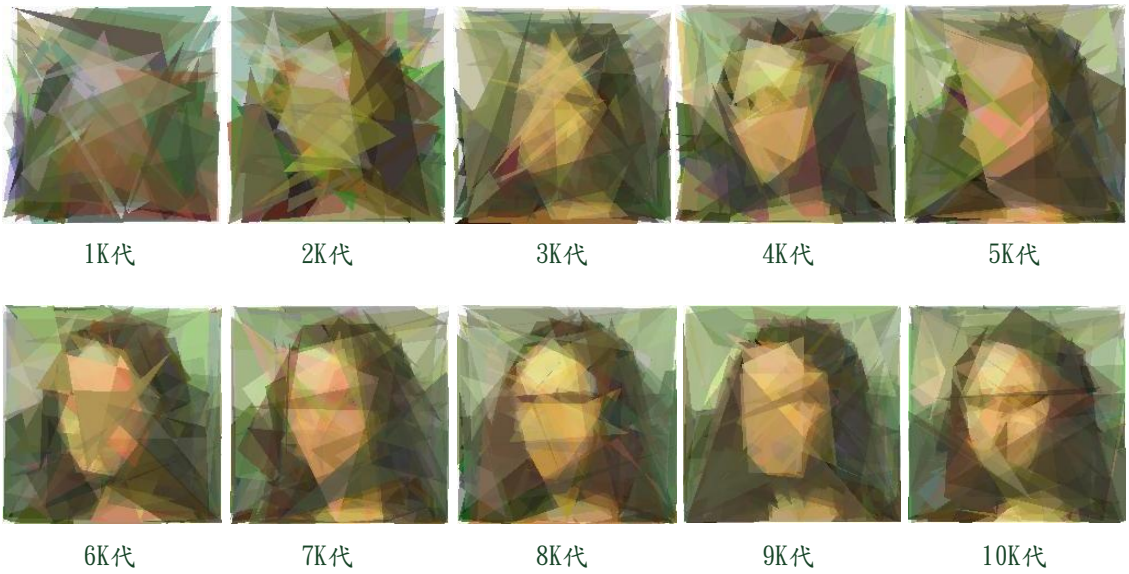

圖 4-12 .單機版於不同代數的演化結果

實驗二：換零錢

CloudGene 架構即使用在需要大量的基因演化資料下，依然可以比 JGAP 快上七倍，而與單機相比更是快上將近 17 倍。

CloudGene	資料筆數	時間(秒)	JGAP	資料筆數	時間(秒)	單機	資料筆數	時間(秒)
	1K	11		1K	20		1K	42
	2K	12		2K	38		2K	81
	3K	14		3K	58		3K	121
	4K	15		4K	78		4K	164
	5K	16		5K	97		5K	205
	6K	19		6K	116		6K	244
	7K	20		7K	137		7K	259
	8K	21		8K	159		8K	283
	9K	23		9K	173		9K	336
	10K	24		10K	192		10K	406

表 4-7 .CouldGene 與其他架構耗時比較表

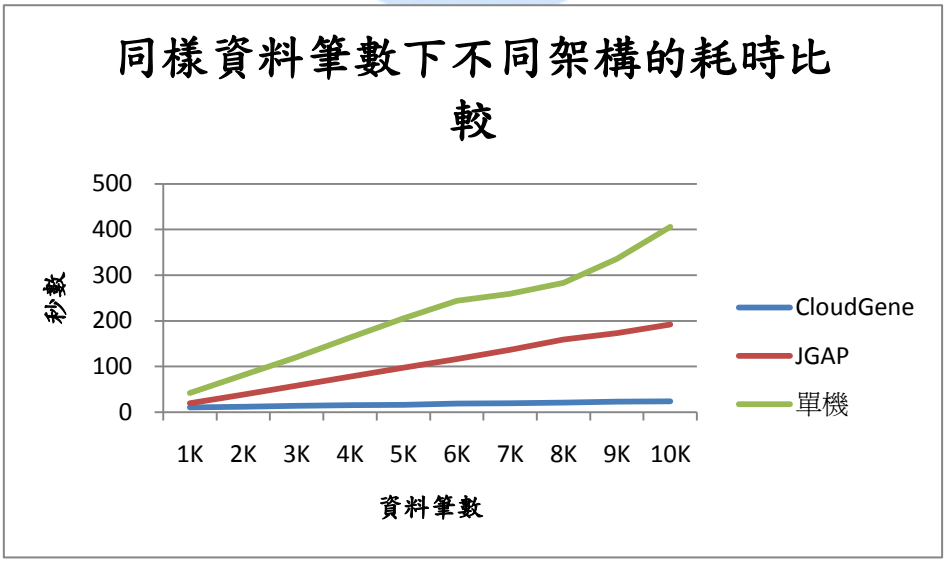

圖 4-13 .CloudGene 與其他架構耗時比較圖

第五章　結論與未來研究方向

本研究提出了一個基於 Hadoop 平台下的雲端基因(CloudGene)架構，其目的是希望可以解決基因演算法在處理較複雜的問題時，能夠在有限的時間內取得演化結果。或是需要大量的基因運算時，能更快速地得到演化結果。欲達到此結果，電腦硬體的資訊與效能佔著極為重要的角色，而網路的穩定性與可靠性也扮演著成敗的關鍵。

雖然本架構可以在一般廉價型的電腦上運行，但是運算效能較佳的中央處理器與較多的記憶體空間，對於演化的速度還是會有舉足輕重的影響。而由於此架構需要依靠大量的網路傳輸，所以對於網路品質的要求也相對性的較高。至於一般電腦常會遇到的電腦當機或是毀損，在此架構下由於有資料的副本機制，是可以容許少量的電腦毀損，而不影響資料的安全性。即使是正在執行演化的電腦發生了毀損的狀況，JobTracker 也能夠自動地將此台電腦的工作，轉移到其它台閒置電腦上運行。雖然可能會耽誤整體運行的時間，但是整個演化還是能夠順利地完成。

經過一系列的實驗證明，本研究所提出雲端基因(CloudGene)架構的確可以在演化品質差異不大的情況下，比傳統的分散式基因或是單機型的基因演算法更加的快速得到運算結果。

在未來的研究方面，此架構下的電腦毀損程度對於整體演化效能的影響以及網路品質的改善與效能提升之間的關係，或是將此架構應用在其他需要大規模運算的領域上，如影像辨識或是影片比對上面等等。皆是未來可以研究的方向。

參考文獻

[1] Amazon Elastic Compute Cloud, http://aws.amazon.com/ec2.

[2] Amazon EC2 Instance Types, http://aws.amazon.com/ec2/instance-types/.

[3] Applications powered by Hadoop: http://wiki.apache.org/hadoop/PoweredBy.

[4] Back, Th. and Schwefel, H. -P., 1993, “An overview of evolutionary algorithms for parameter optimization”, Evolutionary Computation, Vol. 1, No. 1, Pages 1-23.

[5] Beasley, D. Bull, D. R., and Martin, R. R., 1993, "An Overview of Genetic Algorithms, Part1, Fundamentals", University Computing, Vol.15, No.2, pp.58-69.

[6] Beasley, D. Bull, D. R., and Martin, R. R., 1993, "An Overview of Genetic Algorithms, Part2, Research Topics", University Computing, Vol.15, No.4, pp.170-181.

[7] Cantú-Paz, E., 1999 , "Migration Policies and Takeover Times in Parallel Genetic Algorithms," Genetic and Evolutionary Computation Conference, pp. 775.

[8] Dean, J., and Ghemawat, S., 2008, “MapReduce: Simplified Data Processing on Large Clusters”, Communications of the ACM, 51(1): p. 107-113.

[9] Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C. Hsieh, Deborah A. Wallach Mike Burrows, Tushar Chandra, Andrew Fikes, and Robert E. Gruber, 2006, “Bigtable: A Distributed Storage System for Structured Data”, 7th USENIX Symposium on Operating Systems Design and Implementation, pp.205-218.

[10] Gen, M. and Cheng R., 1997, Genetic Algorithms & Engineering Design, New York: John Wiley & Sons, Inc.

[11] Goldberg, D. E., 1989, Genetic algorithms in search, optimization and machine learning, MA: Addison Wesley.

[12] Hadoop, http://hadoop.apache.org/.

[13] Hiroyasu, T., Miki, M., Negami, M., "Distributed Genetic Algorithms with Randomized Migration Rate, 1999," IEEE Proceedings of Systems, Man and Cybernetics Conference SMC '99, Vol. 1, pp. 689-694.

[14] Holland, J. H., 1975, Adaptation in natural and artificial systems, MI: The University of Michigan Press.

[15] JGAP, http://jgap.sourceforge.net/.

[16] Kojima, K., Matsuo, H., Ishigame, M., 2003, "Asynchronous Parallel Distributed GA using Elite Server", Congress on Evolutionary Computation, Vol. 4, pp. 2603-2610.

[17] Matei Zaharia, Andy Konwinski, Anthony D. Joseph, Randy Katz, Ion Stoica, 2008, “Improving MapReduce Performance in Heterogeneous Environments”, 8th USENIX Symposium on Operating Systems Design and Implementation.

[18] Matsumura, T., Nakamura, M., Miyazato, D., Onaga, K., Okech, J., 1997, "Effects of Chromosome migration on a Parallel and Distributed Genetic Algorithm," International Symposium on Parallel Architectures, Algorithms and Networks, pp. 357-361.

[19] Matsumura, T., Nakamura, M., Okech, J., Onaga, K., 1998, "A Parallel and Distributed Genetic Algorithm on Loosely-coupled Multiprocessor Systems," IEICE Trans, Vol. E81-A, No. 4, pp. 540-546.

[20] Michalewicz, Z., 1992, Genetic algorithms + data structures = evolution programs, Artificial Intelligence. Berlin: Springer.

[21] Nutch, http://nutch.apache.org/.

[22] Sanjay, Hgobioff, and Shuntak, 2003, “The Google File System”, 19th ACM Symposium on Operating Systems Principles(SOSP).

[23] Scaling Hadoop to 4000 nodes at Yahoo!, http://developer.yahoo.net/blogs/hadoop/2008/09/scaling_hadoop_to_4000_nodes_a.html.

[24] Shadi Ibrahim, Hai Jin, Bin Cheng, Haijun Cao, Song Wu, 2009, “CLOUDLET: Towards MapReduce Implementation on Virtual Machines”, Proceedings of the 18th ACM international symposium on High performance distributed computing, Pages: 65-66.

[25] Shadi Ibrahim, Hai Jin, Lu Lu, Li Qi, Song Wu, and Xuanhua Shi, 2009, “Evaluating MapReduce on Virtual Machines: The Hadoop Case”, Proceedings of the 1st International Conference on Cloud Computing, Pages: 519 - 528.

[26] Spears, W. M., De Jong, K. A., Back, T., Fogel, D. B., and deGaris, H., 1993, "An overview of evolutionary computation", Proceedings of the 1993 European Conference on Machine Learning.

[27] Syswerda, G., 1989, "Uniform crossover in genetic algorithms", Proceedings of the third international conference on genetic algorithms and their applications, San Mateo, CA: Morgan Kaufmann, 2-9.

[28] Welcome to NCHC Cloud Computing Research Group, http://trac.nchc.org.tw/cloud.
[29] Yahoo! Launches World's Largest Hadoop Production Application, http://developer.yahoo.net/blogs/hadoop/2008/02/yahoo-worlds-largest-production-hadoop.html.
[30] Yi, W., Liu, Q. He, Y., "Dynamic Distributed Genetic Algorithms", Evolutionary Computation, Vol. 2, pp. 1132-1136, 2000.