東海大學資訊管理研究所

碩士學位論文

MapReduce 架構輕量級粒子群演算法之設計與實作

Design and Implementation of a Light Particle Swarm Optimization Algorithm Using MapReduce

指導教授：林正偉 博士

研 究 生：羅允妍 撰

中華民國 105 年 12 月

東海大學資訊管理學系碩士學位

考試委員審定書

資訊管理學系研究所 羅允妍 君所提之論文

Map Reduce 架構轉臺級粒子群演算法之設計與實作

經本考試委員會審查，符合碩士資格標準。

學位考試委員會 召集人：謝錫源 （簽章）

委 員：陳牧言

陳伯根

朱正偉

中 華 民 國 105 年 12 月 29 日

誌 謝

首先最要感謝我的指導教授林正偉老師，在研究所的期間給予我細心的指導和關心，遇到問題的時候給予我幫助、建議與鼓勵，在口試前的準備時，感謝老師仔細的給予提點和建言，這一段期間讓我有了許多不同的經驗，在各個方面都感謝老師的照顧！以及感謝各位口試委員細心的校閱與指正，給予我許多寶貴的意見能使論文更為完整謹慎。

在碩士班的這段期間，特別感謝研究室的祐陞、凱元以及侑侑學長，在學業上的指導，還有感謝炯文協助我設置實驗所需的虛擬設備。也特別感謝研究所的同學們（文隆、嘉翔、致中、哲籍、展毅、伯儒、鈞弘），讓這段時間過得非常充實，跟大家聊天，一起吃飯、出遊或是打拚課業，有非常多愉快的回憶；特別感謝最後幾個月一起在 LAB 奮鬥的同學們，尤其是伯儒、哲籍跟展毅的陪伴，真的很感謝他們這段期間給予我很多的幫助！另外要感謝系辦助教們幫助我處理各種不同的事務，並且在我有任何問題的時候給予我許多建議，謝謝！

最後當然是要感謝我的家人，感謝他們對我的付出與照顧，讓我無後顧之憂，並經常給予關心鼓勵，家人是我求學過程中心靈最大的支柱，謝謝我的爸爸、媽媽和姐姐！十分感謝這段期間給予我幫助和支持的家人、老師們、助教們、學長們和同學們！

羅允妍 謹誌

中華民國一百零六年一月

論文名稱：MapReduce 架構輕量級粒子群演算法之設計與實作

校所名稱：東海大學資訊管理學系研究所

畢業時間：105 年 12 月

研究生： 羅允妍 指導教授：林正偉

論文摘要：粒子群優化(Particle Swarm Optimization, PSO)是一個被廣泛採用的優化演算法。在PSO中，每一個粒子代表一個可能的解，這些粒子可以透過一代一代的反覆運算，在解空間中逐漸移動到最佳解附近。隨著巨量資料(Big Data)受到關注，MapReduce平行處理架構引起很多學者的注意。該架構是一個相當簡單的平行運算模型，已經有很好的實現，具有高延展性與容錯性，已經在許多大型資訊服務系統中得證實。由於PSO進行優化求解時會進行多代反覆計算，每一代啟動一次MapReduce進行平行處理的高昂啟動成本時常會抵銷平行處理的效益。我們提出一個基於MapReduce架構的平行化PSO演算法，透過減少啟動MapReduce的次數，讓在同一運算節點的粒子可以進行多代的反覆運算，使得PSO的總運算時間可以有效地的降低。

本研究實驗結果顯示，透過減少啟動Reduce的次數來看可以明顯地降低運算所耗用的時間成本，但隨著減少Reduce的次數越多，其最佳適應值的變化卻越差，而在減少Reduce的2、3次數上，則有較佳的結果。

關鍵詞：粒子群優化演算法、Hadoop、MapReduce

Title of Thesis : Design and Implementation of A Light Particle Swarm Optimization Algorithm Using MapReduce

Name of Institute: Tunghai University, Graduate Institute of Information Management

Graduation Time : (12/2016)

Student Name : Yun-Yen Lo Advisor Name : Jeng-Wei Lin

Abstract : In PSO, each particle represents a possible solution in the solution space. In an iteration, all particles move by simulating the behaviors of birds. After a number of iterations, particles move expectedly more and more close to the global best solution. With the emergence of Big Data, MapReduce, a parallel processing architecture proposed by Google, has attracted the eyes of many researchers. MapReduce is highly scalable and fault tolerant. It has been successfully adopted in many large information services. In the literature, researches had tried to implement PSO using MapReduce. In each iteration, the utility function of particles is computed in parallel by invoking a MapReduce job. However, the cost of MapReduce invocations is high. As a result, the benefit of parallel processing is significant only when the utility function is complex enough. We propose a light parallel PSO algorithm based on MapReduce architecture. In a MapReduce invocation, particles move for several iterations independently. Thus, the number of MapReduce invocation is reduced. Experiment results shows that the time for PSO to convergence is reduced. However, particles will not move close to the global best solution if they compute the utility function independently for a long time. In our experiments, when particles move for two or three iterations in a MapReduce invocation, the performance of the proposed method is high and the cost is significantly reduced.

Keywords : Particle Swarm Optimization (PSO), Hadoop, MapReduce

目錄

圖次

表次

第一章 緒論

1.1 研究背景與動機

近年來隨著網際網路的普及與網路頻寬的速度漸漸增加的情況下，使得使用者越來越習慣使用各種網路服務，如Google Search、Gmail、Facebook等。在這樣的情況下，使得該服務產生了許多巨量的資料(Big Data)，因此如何更好的來處理這些資料，成為了網路業者要思考和解決的難題。

對Google這個主要以網路服務來獲利的業者來說，同樣地也無法避免這個難題，針對這些巨量的資料，Google的處理方法提出了分散式檔案系統架構Google File System(GFS)(Ghemawat, Gobioff and Leung, 2003)，Google提出MapReduce架構用來處理大量的資料(Dean and Ghemawat, 2004)，Google基於Google File System提出了分散式檔案系統BigTable(Dean and Ghemawat, 2008)，這篇論文詳細的說明如何運用其架構並談到用該架構而使得Google增加的重大效益。

由於巨量資料的盛行，MapReduce平行處理架構受到許多學者的注意。許多研究顯示，MapReduce平行處理架構能有效的解決巨量資料處理的問題(McKenna, et al. , 2010; Taylor, 2010; Dittrich and Quiané-Ruiz, 2012; Krishnan, Baru and Crosby, 2010; Lin, Shen, Sung, Lam, Lin and Lai, 2013) 如Apache Hadoop。

Hadoop 雲端運算平台是由Apache(APACHE.ORG)基金會參考Google所發佈的GFS論文，使用Java程式語言所開發出的一套自由軟體，Hadoop平台可由上千萬個節點的電腦所組成，其能夠處理巨量的資料，如Petabyte(PB)規模的資料量，藉由Hadoop平行分散式檔案的處理方式能夠快速獲得處理結果。其具有高延展性(scalability)、可靠性(availability)與容錯性(fault tolerance)，其效能已經在許多大型資訊服務系統中得到證實。

在許多優化應用中，Kennedy and Eberhart (1995)提出的粒子群優化(Particle Swarm Optimization，PSO)是一個被廣泛應用的優化演算法，PSO透過模擬鳥類覓食的行為(Reynolds, 1987; Heppner and Grenander, 1990)，讓許多粒子透過一代一代

的反覆運算，在解空間中漸漸移動到最佳解附近。PSO演算法的概念簡單，實作容易，因此在短期內就能得到很大的發展，成功地解決許多領域中的最佳化問題(Hu and Eberhart, 2002; Van den Bergh and Engelbrecht, 2000; Messerschmidt and Engelbrecht, 2004; Clerc, 2004)。許多針對PSO的研究與改良也相繼被提出來(Eberhart and Shi, 1998; Eberhart and Shi, 2000; Shi and Eberhart, 1998, a ; Shi and Eberhart, 1998, b ;Trelea, 2003)。由於上述所提出的觀點，故有學者將PSO演算法與MapReduce架構相互結合(McNabb, Monson and Seppi, 2007a; McNabb, Monson and Seppi, 2007b)，爾後也有許多以此為基礎的研究論文(Aljarah and Ludwig, 2012; Jin, Vecchiola and Buyya, 2008; Sadasivam and Selvaraj, 2010)，本研究是參考此架構為參考進行PSO演算法與MapReduce的結合而延伸的實驗(McNabb, Monson and Seppi, 2007a)。

1.2 研究目的

根據上述的研究背景與動機，人們面對巨量資料時，大部分都以平行化運算來處理，利用多台的機器同時運算來分攤運算時間，以達減少整體的總運算時間，而PSO演算法由於概念簡單且在平行化處理上較其他仿生演算法來得單純，也就是粒子在部分運算上適合獨立運作而不需與其他粒子互動，因此本研究選擇PSO演算法來進行實驗，故本研究將深入的探討在MapReduce上實作PSO演算法，並試著減少MapReduce架構中Reduce階段，期望透過減少Reduce的次數，來縮短計算收斂所需時間。本研究之主要目的如下：

1. 參考學者在2007年提出的架構MRPSO(McNabb, Monson and Seppi, 2007a)來進行改良。
2. 藉由減少PSO演算法在MapReduce架構上的Reduce階段來縮短計算收斂所需時間。
3. 觀察減少Reduce階段多寡的適應值變化。

1.3 論文架構

本論文的架構分為五章，第一章為緒論，介紹研究背景與動機；第二章為文獻探討，針對論文中使用的平台以及演算法進行介紹；第三章為研究方法，對本研究的架構設計進行詳細的介紹；第四章為實驗結果，將實驗的結果做一個清楚的呈現；最後為本研究的結論與未來展望，對這次的研究過程和結果進行總結，並提出未來持續改進的方向。

第二章 文獻探討

由於網路的盛行，導致現今網路數據資料量大幅提升，巨量資料(Big Data) 這個名詞就在這樣的環境下誕生了，Big Data 主要是指資料量規模巨大到無法透過人工或著電腦，在合理的時間內達到處理或整理成為人們容易解讀的資訊。在Douglas(2001)的研究和演講中以三種特性來描述Big Data，分別是大量（Volume，資料大小）、速度（Velocity，資料輸入輸出的速度）與多變（Variety，多樣性，如企業的銷售資料、庫存資料；網站的使用者動態，社交媒體的文字、影像等。），合稱「3V」。

因為巨量資料的興起，讓許多企業紛紛開始投入巨量資料分析的應用研究中，像是Google發表了一篇論文The Google File System(Ghemawat, Gobioff and Leung, 2003)，文中敘述GFS(Google File System)，由數百個叢集(Cluster)組合而成的，儲存在GFS的檔案會被切割成64MB左右的資料塊。

Google發表了MapReduce論文，MapReduce: Simplified Data Processing on Large Clusters(Dean and Ghemawat, 2004)；Google發表了Bigtable論文，Bigtable:A Distributed Storage System for Structured Data(Dean and Ghemawat, 2008)，此篇論文的發表帶領了NoSQL資料庫的技術應用發展，如HBase等。

2.1 Hadoop

Apache Hadoop 是一個支援資料密集型的分佈式應用，以Apache2.0 協議發佈的一個開源軟體框架，Hadoop是根據Google所發表的相關研究論文所開發而成。

Hadoop是一個叢集系統(cluster system)，由單一伺服器擴充至數以千台的機器，整合成一台超級電腦，Hadoop叢集的資料存放方式是採用HDFS分散式檔案系統（Hadoop Distributed File System），在叢集系統中有數以千計的節點來存放資料，在整個叢集中分為主節點(master node)與其他次要節點(worker node)，資料分佈的方式是先將整個資料分割成數個小塊(block)，每一塊的大小通常是128MB，再將

每小塊的資料拷貝3份，個別分散在次要節點上，我們可以在主節點上監視其他節點存放資料的狀態。

Hadoop 這個軟體框架能夠處理大量的數據並進行分散式處理是因為能夠把應用程式分割成許多小的工作單元，再將這些工作單元放置叢集中任一節點上執行。在一個配置完整的叢集中，要讓 Hadoop 整個運作起來，需要叢集運行一系列的後台程序，不同的後台程序扮演不同角色，而這些角色分別由 NameNode、DataNode、 Secondary NameNode、JobTracker、TaskTracker 所組成。NameNode、Secondary NameNode 以及 JobTracker 運行在 Master 節點上；DataNode 和 TaskTracker 則運行在 Worker 節點上。圖 2-1 為 Hadoop 的簡易架構圖。

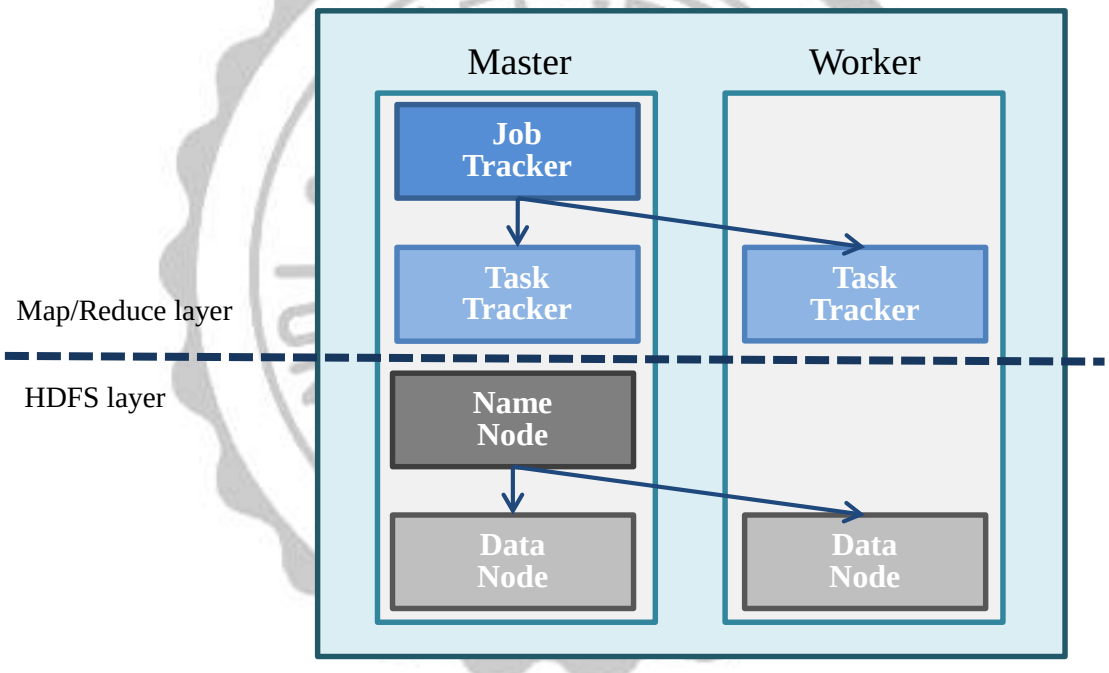

圖 2- 1、Hadoop 簡易架構

NameNode、Secondary NameNode、DataNode在HDFS部分中作介紹；JobTracker、TaskTracker在MapReduce部分作介紹。

由於Hadoop架構相當穩定，因此採用此架構的應用相當多，在2008年2月Yahoo將Hadoop架構應用在Webmap專案上，該專案為Yahoo Search 的核心搜尋計畫，在同年9月Yahoo 公布了架構平台的處理能力，結果表現相當驚人，HDFS檔案系統的資料儲存能力達到了16PB(Baldeschwieler, 2008)。

王宏仁(2012)的報導指出Wal-Mart利用Hadoop來分析顧客行為，顧客利用搜尋引擎下關鍵字的方式找到Wal-Mart網站，而Wal-Mart則利用這些關鍵字來分析並發掘顧客需求以及規畫下一季的商品促銷策略。在2015年學者的研究論文(Harsoor and Patil, 2015)中，其根據Wal-Mart的過去的銷售紀錄、地理位置、不同時段僱用的員工人數等，並利用MapReduce對顧客行為進行研究調查，好讓零售業者能根據過去的資料改變營運策略，並在不同時期預測員工人數是否滿足商場需求以及市場需求的趨勢。

經營網路拍賣的eBay利用Hadoop來分析買賣雙方在網站上的行為，eBay每日增加的資料量有50TB，有結構化資料也有非結構化資料，如照片，影片，電子郵件等。因此eBay 採用Hadoop來處理非結構化資料，透過Hadoop進行資料的預處理，來加快分析速度同時也減輕了資料倉儲系統的分析負載量(Zicari, 2014)。

在Big Data: How to Turn Big Data Into Great Information(Celio Di Cellio Dias, , Henrique Outi Kauffmann and Fernandes, 2014)一書中指出Visa公司－全球最大的付費網路系統VisaNet，用來作為信用卡付款驗證。2009年，每天要處理1.3億次授權交易和140萬臺的ATM的連線存取。為了降低信用卡各種詐騙和盜領事件而導致的損失，Visa公司透過分析每一筆交易資料來找出可疑的交易，由於使用者眾多，2年累積的資料量就多達36TB，單單分析5億個用戶帳號之間的關聯性必須至少花上1個月的時間來等待結果，Visa在2009年時導入Hadoop，讓分析所耗用1個月的時間降低到只需13分鐘即可快速的找出可疑交易，更快的對銀行提出預警及時地阻止詐騙交易。

2.2 HDFS架構

在Hadoop平台中，儲存空間的資源管理、記憶體空間以及程式排程的安排都是Hadoop的主要核心用途。Hadoop透過HDFS分散式架構的檔案系統將多台一般商用等級的電腦或伺服器組合成一個叢集，來提供巨量資料的儲存。

在HDFS上，NameNode負責管理和維護HDFS的名稱空間，控制檔案的讀寫動

作，同時將正待處理的資料切割成一塊塊的檔案區塊(Block)，每個區塊是128MB，在Hadoop上每一塊的預設是128MB，可依據使用者的需求來做更改，資料切割完後NameNode還會將每一份檔案區塊做備份，一般來說預設是3份，可依使用者需求做更改，備份後的資料會分散儲存在3個不同的DataNode上，只要其中的一份檔案區塊遺失或損壞，NameNode會從其他DataNode上的副本來進行修復的動作，一直將該檔案維持在3份的狀態。

接著介紹 HDFS 的相關程序角色：

(1)、 NameNode

NameNode 是 HDFS 的控制中心，在 HDFS 中負責掌管叢集中所有檔案系統的儲存位置，並監督 DataNode 的存活狀態，NameNode 本身不儲存檔案數據，但知道某檔案是由哪些 block 所組成，而這些 block 的位置又是存放在哪個 DataNode。因此 NameNode 的功用即是引導 Client 要到哪個 DataNode 來索取檔案數據，監督整個叢集儲存空間的容量和 DataNode 的存活狀態，確保每個 block 依循著系統的設定。

(2)、 Secondary NameNode

Secondary NameNode 是用來監控 HDFS 狀態的後台輔助程序，與 NameNode 相同，每個叢集都只有一個 Secondary NameNode，其部屬在一個單獨的伺服器上，Secondary NameNode 不記錄任何實時的數據變化，但會定期的與 NameNode 進行溝通，定期的保存 HDFS 的快照，如果 NameNode 發生問題，Secondary NameNode 則作為備用的 NameNode 使用。

(3)、 DataNode

DataNode 部屬在 Worker 節點上，其根據 Client 端或是 NameNode 的調度儲存、檢索數據，主要功能是負責把 HDFS 數據塊讀寫到本地端。當 Client 端要讀/寫某個數據時，NameNode 會告訴 Client 端該到哪個 DataNode 上來進行具體的讀/寫操作，之後，Client 端會直接與該 DataNode 進行溝通和操作。

2.3 MapReduce

MapReduce是由Google提出的平行運算架構(Dean and Ghemawat, 2008)。Google發表此架構後並沒有開放其原始碼，Apache Hadoop專案下的MapReduce平台是現今許多從事相關研究的人員採用的平台，也是我們這個研究所採用的平台。

MapReduce架構為主從式分散架構，由多節點組成。它採用標準divide-and-conquer策略，由master節點將工作(job)拆解成一個一個的任務(task)，並分配給其他worker節點進行分散式運算，最後再進行總結。

worker節點須定時回報工作進度，master藉此掌握整體運算的執行狀況。當有worker節點逾時沒有回報進度，其所負責的工作會交給其他空閒的worker節點負責，藉此容錯機制實現高可靠性。

MapReduce 四個組成部分，分別為Client、JobTracker、TaskTracker和Task

(1)、Client

使用者的每一個 Job 通過 Client 將應用程式的配置參數 Configuration 打包成 JAR 文件存儲在 HDFS，並把路徑提交到 JobTracker 的 Master 服務，由 Master 創建每一個 Task(Map Task 和 Reduce Task)將他們分發到各個 TaskTracker 服務中。

(2)、JobTracker

JobTracker 負責資源的監控和作業的調度。JobTracker 監控所有 TaskTracker 與 job 的狀況，一旦狀況不佳即將該任務轉移至其他節點；同時，JobTracker 會追蹤任務的執行進度以及資源使用量等相關訊息，並將這些訊息告知任務調度器，讓調度器將空閒的資源分配給需要的任務進行使用。

(3)、TaskTracker

TaskTracker 會規律地通過 Heartbeat 將本地節點上資源的配置情況和任

務運作進度彙報給 JobTracker，並接收 JobTracker 發送過來的命令和執行相對應的操作，(如啟動新的任務等)。

(4)、Task

Task 分為 Map Task 和 Reduce Task 兩種，皆由 TaskTracker 啟動。

Map Task 執行過程如圖 2-2 所示，Map Task 將對應的輸入格式解析成一對對的<key/value>值，依次調用使用者自定義的 map 函數進行處理，最終將 Map Task 的輸出結果(又稱中間結果)分成若干個 partition 並存放在本地磁碟上。而中間結果將被 Reduce Task 處理。

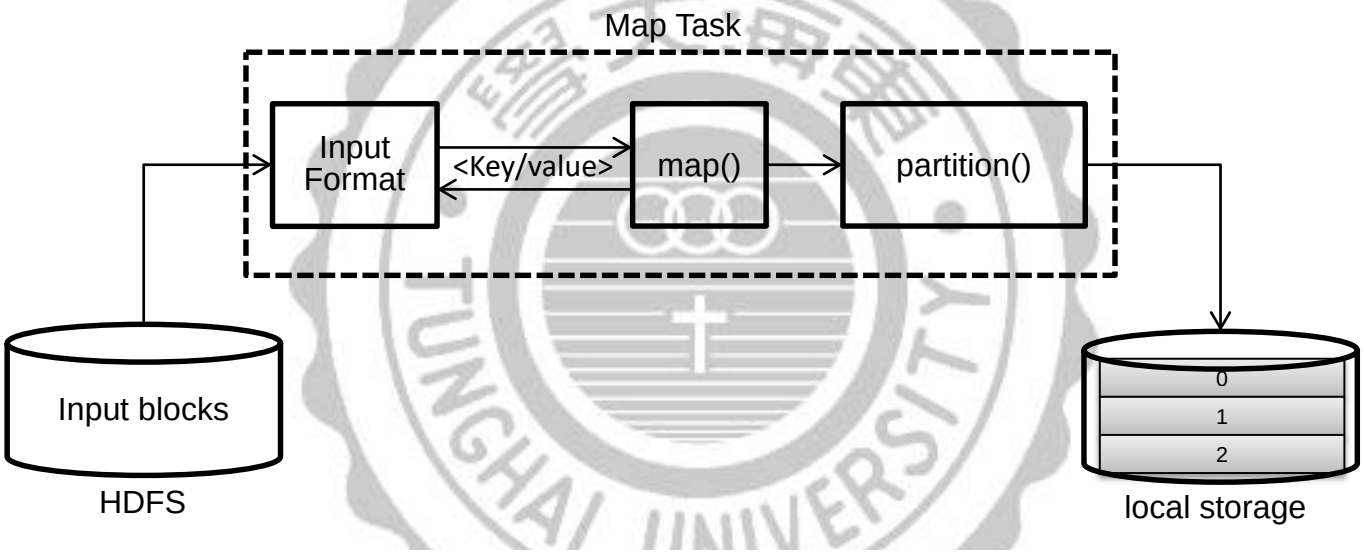

圖 2- 2、Map Task

Reduce Task 執行過程如圖 2-3 所示，首先從各個節點上讀取 Map Task 中間結果，此階段稱之為 Shuffle 階段，再來依 key 值對<key/value>對進行排序，這部份成為 Sort 階段，接著依次讀取<key/value list>，調用使用者定義的 reduce 函數進行處理，並將最終結果存到 HDFS 上，該過程稱為 Reduce 階段。

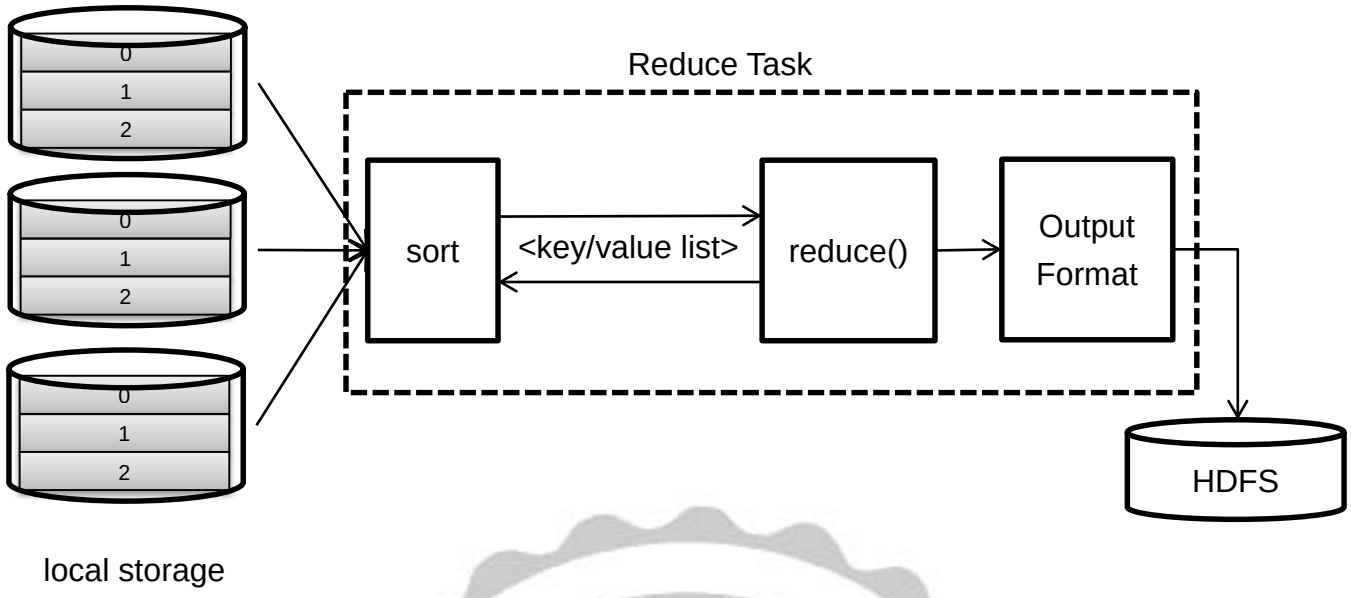

圖 2- 3、Reduce Task

MapReduce 運作可分解為「映射(Map)」和「歸納(Reduce)」兩階段，前者透過多個節點分工進行資料運算，後者將運算結果進行歸納整理。在 Map 階段，master 節點根據設定檔(configuration)，將輸入檔案拆解成數個區塊(block)，分別交給不同的 Worker 節點執行指定的運算，並以 key-value 的方式，將運算的結果交由後續的 Reduce 階段處理。在 Map 階段，各個 Worker 節點獨立執行相同的運算，互不干涉，因此可以得到高度的平行化。進入 Reduce 階段前，會根據 key-value 的設定，將 Map 階段計算得到的結果分散到不同的 Worker 節點上。由於歸納過程一般較不容易平行化，master 節點會盡量把歸納運算分配在一個 Worker 節點或者比較靠近資料的 Worker 節點上。在 Reduce 階段，再由這些 Worker 節點進行整合運算。

相較於傳統的 MPI (Message Passing Interface)平行運算架構，MapReduce 的 programming model 限制較多，彈性較少(Plimpton and Devine, 2011)。然而，正因為其 programming model 相對簡易，使得它能很有效的被實現，特別是在超大型分散式計算環境中。使用者只需專注於 Map 與 Reduce 函數的設計，不須考慮系統面的延展性、可靠性、容錯支援等等。Apache Hadoop 已經在許多應用中實證它能提供很好的效能。

MapReduce 是 Hadoop 分散式運算的關鍵技術，也是一種解決問題的程式開發模式，將要執行的問題拆解成 Map 與 Reduce 的方式來執行，首先程式開發人員必需先分析問題的解決流程，從流程中找出可以平行處理的部分，再將此部分的需求寫成 Map 程式，接著使用大量的機器來執行 Map 程式，最後將每一個 Map 程式丟出的結果透過 Reduce 程式進行合併，彙整出一個完整的結果。MapReduce 程式的執行過程如圖 2- 4 所示：

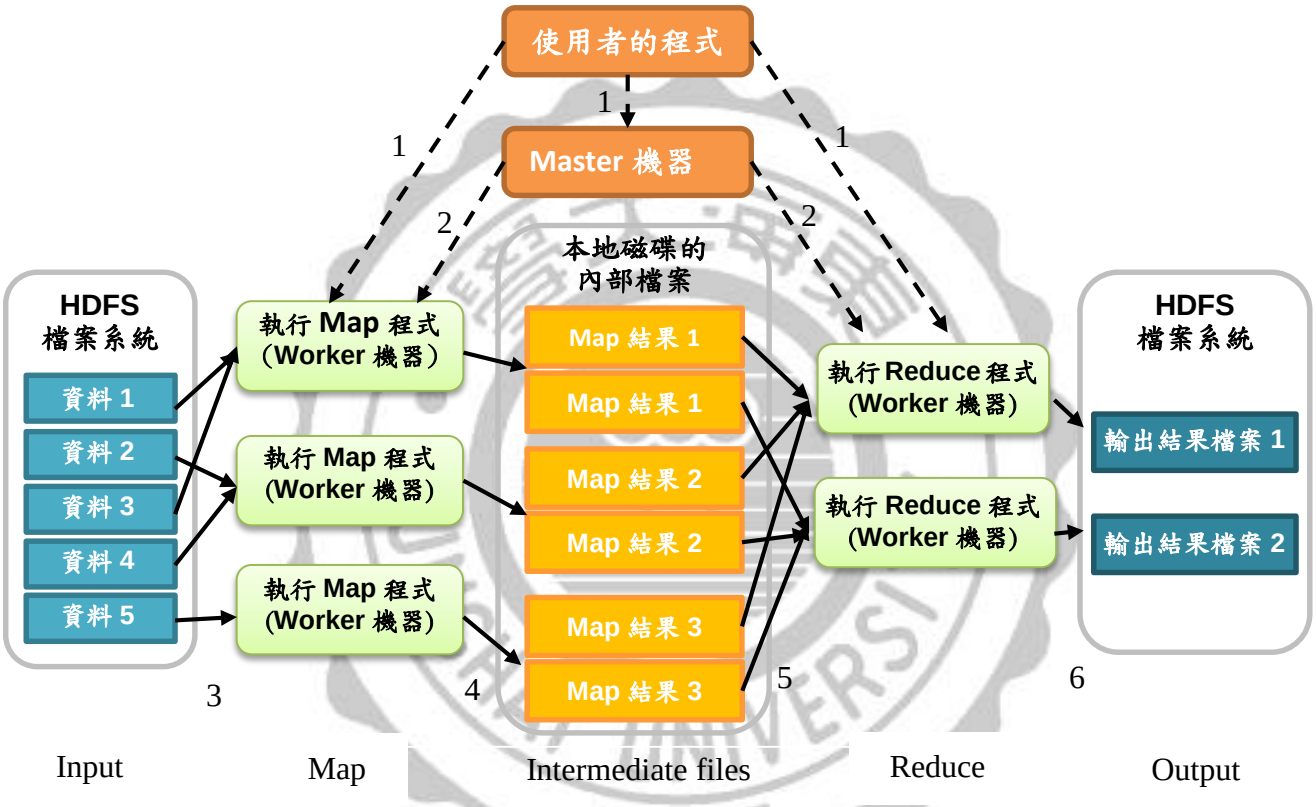

圖 2- 4、HDFS 架構圖

1. 首先將資料切成N塊（blocks），每塊預設的大小為128MB（使用者可透過參數來設定資料塊大小），再來將要執行的MapReduce程式複製到Master與每一台worker機器中，啟動該程式並執行，Master負責監控各台Worker的狀態。

2. Master決定Map程式與Reduce程式，分別由那些worker機器執行。

3. 被分配到Map任務的worker機器讀出需處理的資料片段，進行Map程式的處理，再輸出key/value對儲至緩衝區（cache）。

4. 在緩衝區中的結果被定時的寫入本地磁碟，這些結果稱為中間結果，其儲存位置會再回傳給Master。

5. 要執行Reduce任務的Worker機器，遠端讀取每一份Map的中間結果，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。

因此簡單的來說，MapReduce的運作方式就是先拆解任務，分工處理後再彙總結果，在整個Hadoop叢集架構中，每一台機器會根據不同的用途分成Master節點和Worker節點，Master節點負責分派任務，Worker節點則負責執行任務。

在Master節點的機器上有兩套程式，一個是負責安排MapReduce運算任務的JobTracker，另一個是負責管理HDFS分散式檔案的NameNode程式。在Worker節點的機器上則是另外兩套程式，分別是接受JobTracker指揮，負責執行運算任務的TaskTracker程式，與NameNode對應的DataNode程式，負責執行資料讀寫的動作。

在MapReduce運算上，作為Master節點的機器負責分配運算任務，在Master節點上的JobTracker程式會將Map和Reduce任務分派給Worker機器上的TaskTracker程式，由TaskTracker負責執行Map和Reduce的工作，並將結果回報給Master節點上的JobTracker。

Hadoop叢集中，只有1個JobTracker程式來分配MapReduce任務，有許多TaskTracker程式則會執行分派下來的任務。同樣地，負責管理HDFS的NameNode也只有一個，與JobTracker一同位在Master節點中，相對地，DataNode也有許多個，Worker節點上會有TaskTracker和DataNode。

另外，Master節點除了有JobTracker和NameNode以外，也會有TaskTracker和DataNode程式，簡單的來說就是Master本身也能擔任Worker的角色。

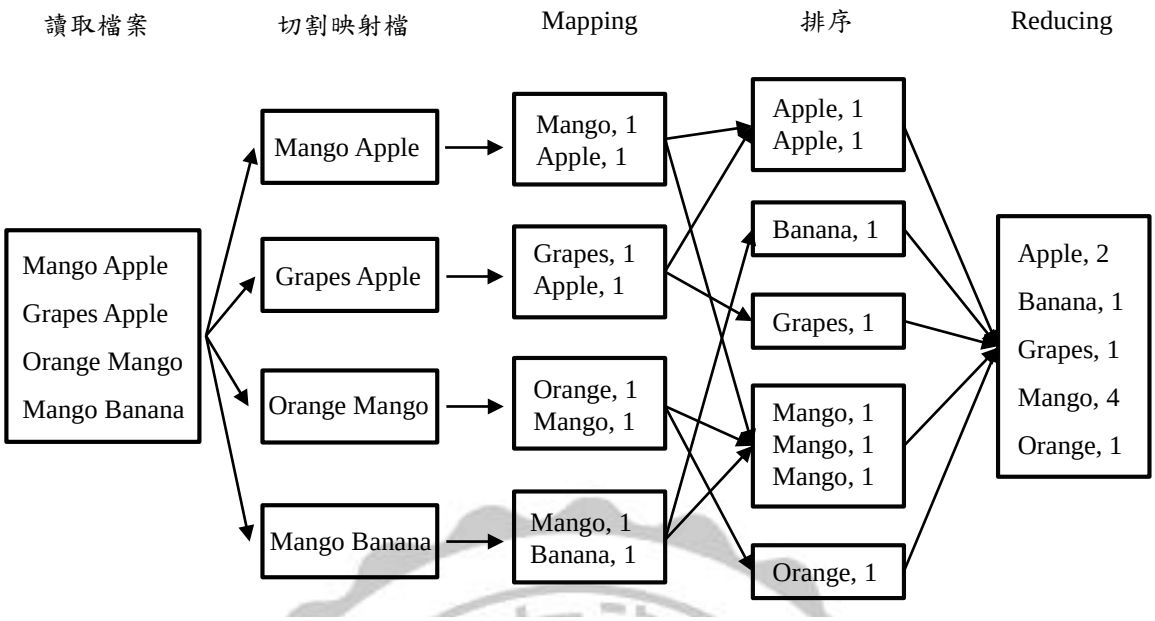

圖 2- 5、MapReduce 案例 Word Count

圖 2-5 為 MapReduce 的典型範例，其運行目的為計算每個單字所出現的次數，首先在 input 檔案中有許多英文單字，input 檔案依照著使用者設定將檔案切割成數個小塊並映射到叢集的節點上，而被分配 Map 任務的節點則將資料根據使用者的 Map 程式部分進行運作，以 key/value 對將 Map 運算結果輸出，在這個例子中，key 為每個英文單字，而 value 為數字 1 代表該單字出現一次，在重新排序的步驟時，以 key 做識別，將相同的 key 整理至同一個節點上，而在執行 Reduce 任務的節點，會將相同 key 值的 value 進行相加，來計算該單字總共出現的次數，最後進行合併整理並輸出總結果。

2.4 粒子群最佳化演算法

PSO演算法是透過模擬鳥類的覓食行為而發展的仿生演算法(Bio-inspired algorithms)(Kennedy and Eberhart, 1995)，此類的演算法是透過模仿生物的生活型態，如覓食、遷移、防衛等。科學家們將這些生活規則透過數學轉換成模型後，用於解決各類實務的問題。像是鳥類群聚飛行時可隱約看出隊伍是有一定規律的集體移動，隊伍中特定位置的成員會比其他成員更早有動作，其他成員則跟隨在後，隊伍偶爾也會出現分散又重新聚集的現象；魚類也會在與同伴爭奪食物的同時，

透過同伴的行為得知食物的位置，來改變自身的移動方向。上述都是很典型的例子，透過分析生物的行為並以各種方式模擬，以數學式表達其行為後，將此現象應用在各領域中。

PSO演算法以「粒子(Particle)」來代替仿生技術的生物，粒子也代表在解空間中的一組候選解，粒子本身具有移動速度，紀錄移動方向和距離，以自身經驗和群體經驗決定下一次移動的方向和速度，透過一次次的移動來尋求最佳解。

演算法初期決定最佳化函數後，將粒子隨機散佈到解空間中，並給予粒子初始速度，在每次的移動中，粒子根據自身最佳經驗、群體最佳經驗，決定自己的移動方向，依據移動速度和方向來更新自己的位置，並計算該位置在此最佳化函數中所得的適應值(fitness value)，來判斷此解是否優劣，基本粒子群演算法流程圖如圖2- 6所示。

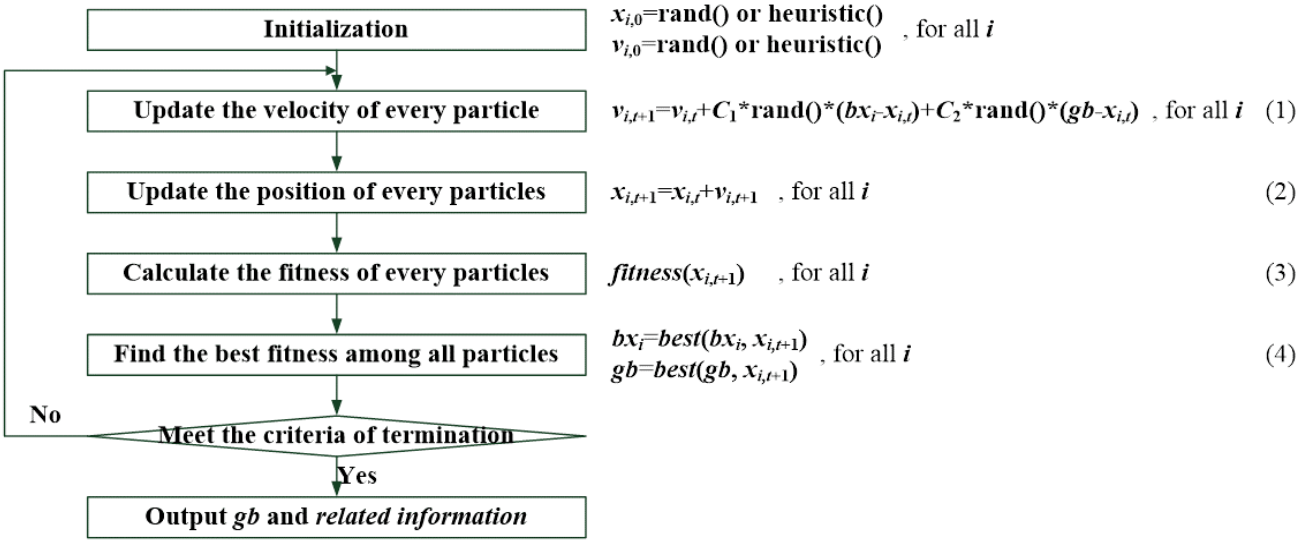

圖 2- 6、PSO 基本架構

粒子在重複的移動行為中，會朝最佳解的方向前進，在前往最佳解的過程中，每個粒子都有機會發現一個更優的解，進而更新群體最佳經驗，在追求最佳解的過程中，同時也對解空間進行探索，直到達到規定的最大化演算代數，依擁有最佳解的粒子判斷出最佳解。

粒子i在某一代t的位置$x_{i,t}$代表了解空間中的一個候選解，它會以慣性速度($v_{i,t}$)、

自身最佳經驗(bx_i)和群體最佳經驗(gb)來決定新的移動速度($v_{i,t+1}$)，在下一代(t+1)群體形成時移動到位置$x_{i,t+1}$，$1 \le i \le N$，N為粒子總數。所有的粒子，透過一代又一代的移動來尋找最佳解。

公式(1)中，C_1、C_2為兩個非負常數，rand()為介於[0, 1]區間的隨機亂數，我們可以看出，粒子i最新的移動速度$v_{i,t+1}$由三項組合而合：(i)慣性速度、(ii)自身經驗與(iii)群體經驗。其中($bx_i - x_{i,t}$)代表粒子往自身最佳經驗(bx_i)調整的過程，屬於自我認知部分，而($gb - x_{i,t}$)則代表粒子追隨其他粒子的最佳經驗(gb)，屬於社會認知部分，透過自己與群體的交互影響，重現一種群體生活的社會型態。C_1、C_2分別代表粒子認可自己與群體經驗的程度，而亂數rand()則扮演經驗吸收過程中的隨機性。

因為概念簡單、實作容易、效果良好，PSO在短期間內就得到很好的發展，得到演化計算研究領域的認可，應用在各方面領域研究，包括多目標最佳化(Hu and Eberhart, 2002)、類神經網路(Van den Bergh and Engelbrecht, 2000)、旅行銷售員問題(Clerc, 2004；Messerschmidt and Engelbrecht, 2004)等。

MapReduce架構被提出後，文獻上已經有PSO在MapReduce上實現的討論(McNabb, Monson and Seppi, 2007)，不少應用也取得很好的成果(Sadasivam and Selvaraj, 2010; Aljarah and Ludwig, 2012)。類似的仿生計算演算法(Lin, Shen, Sung, Lam, Lin and Lai, 2013; Shen, Zhou, Lin, Sung, Lam, Chen, Lin, Pan, Chiu and Lai, 2013; Aljarah and Ludwig, 2012)，如基於MapReduce的平行化GA也有當然好的成果。

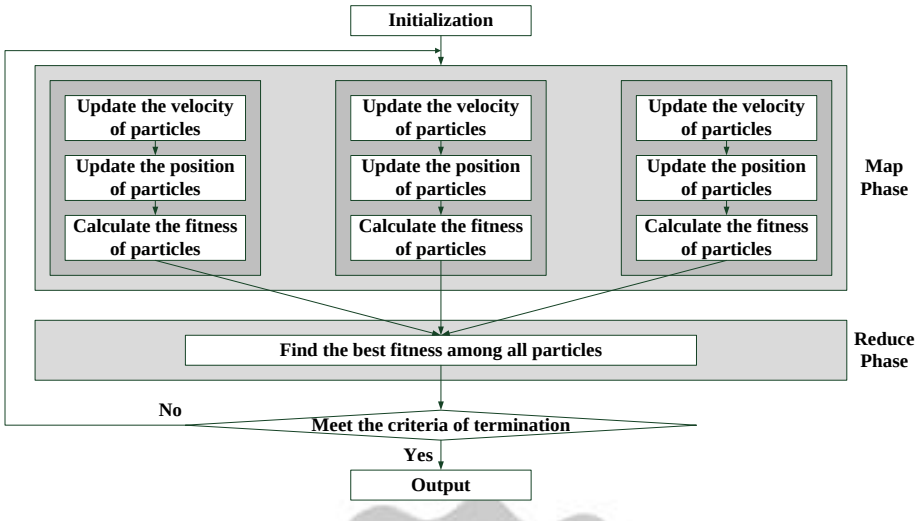

圖 2- 7、PSO 於 MapReduce 架構上實作的一個直覺上的結構

就我們所知，這些實作均是以最直覺的方式將PSO或GA實現在MapReduce架構上，大致結構有如圖2-7所示。如MRPSO: MapReduce particle swarm optimization (McNabb, Monson and Seppi, 2007a)，它將粒子調整速度、更新位置、計算適應值這些運算放在Map階段，再將尋找群體最佳位置的運算放在Reduce階段。由於每個粒子的運算是獨立的，很容易達到高度的平行化，在Map階段透過平行運算實現十分直觀。而在Reduce階段總合所有粒子的運算結果，尋找群體最佳位置也同樣直觀。

在圖2-7中，我們可以注意到，MRPSO進行每一代的計算時，會啟動一次MapReduce運算。由於MapReduce運算的啟動成本相當的高，在McNabb, Monson, and Seppi(2007)提出的論文中，作者提到MRPSO並不適合使用在適應值函數容易計算的應用。

紀玟君(2013)在使用MapReduce之平行化詢問式粒子群演算法的論文中，進行了PSO在MapReduce上一些相關實驗以及MapReduce與機器數量的效能測試實驗。圖2- 8為該篇論文的成效實驗：

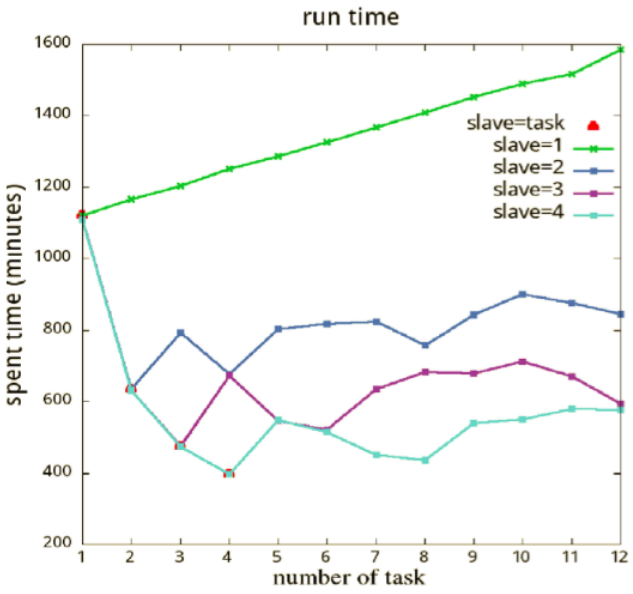

圖 2- 8、運行不同數量 slave 與 task 的運行時間

資料來源：紀玟君(2013)

從圖2-8可以發現，task數量等於機器數量，以及task數量小於機器數量時，可以得到最短的運行時間；當task數量大於機器數量時，運行時間則呈現高低起伏的現象，這表示，當task越多時，運行時間不見得越好，其效能與task數量不成正比，所以從這邊可以發現每啟動一次MapReduce的成本是相當高的；在紀玟君(2013)的這篇論文中，觀察圖2- 8發現並得出當task數量為機器的2 * n (n 為正整數)時為最佳配置，因此本研究在task的數量上的設定為預設值2。

由於從圖2- 8中，了解到啟動MapReduce是要花費一定的時間成本，而在MRPSO圖2- 7可以觀察到一個現象，粒子群每搜索(更新自身最佳解)一次即群聚(尋找群體最佳解)一次，而群聚的動作則是在Reduce階段進行的，在MapReduce的架構中我們可以很清楚的了解到，由於啟動Reduce階段時，會收集各個Map的中間結果來進行整合，故得出啟動Reduce的成本會比啟動Map的成本來得更高，因此本研究希望透過減少粒子群群聚的次數來降低PSO在MapReduce實作的收斂時間成本，簡而言之即是減少Reduce階段來達到降低啟動MapReduce的次數。

第三章 研究方法

3.1 研究架構

在文獻探討中，可以了解到單機版的PSO運作模式，其中能夠觀察到單機版的PSO在計算群體最佳解時成本並不高；在紀玟君(2013)的實驗中，可以發現每次啟動MapReduce的成本並不低；而在MRPSO架構中，可以發現PSO在進行更新群體最佳解的部分時，所花費的成本比單機版的PSO高出許多，因為單機版的PSO不需花費與其他機器溝通的時間，在上述文獻MRPSO: MapReduce particle swarm optimization (McNabb, Monson and Seppi, 2007a)中，它將粒子調整速度、更新位置、計算適應值這些運算放在Map階段，在這邊我們也將此部分稱作粒子獨自探索階段，而在Reduce階段尋找群體最佳適應值的過程我們則稱之為群聚階段。因此在MRPSO的架構圖中就能看出，粒子群每群聚(更新群體最佳解)一次的成本其實相當的高。

在這個研究中，我們也將在MapReduce架構上更有效實現與改進PSO的方法。如上述所言，MRPSO進行每一代的計算時，會啟動一次MapReduce運算。由於啟動時所花費的高成本，因此本研究試著減少PSO在MapReduce架構中群聚的次數，也就是減少Reduce次數來達到降低成本的概念。這個概念像是當山難發生時，搜救人員在搜救的過程中，並不是每個搜救人員每搜索一個點就彙報一次，可能是搜索完了附近這1、2個點才進行一次彙報又或是到規定的時間，才進行彙報。

本研究的方法是將粒子在每一代更新自身最佳解結束後立即與其他粒子比較並找出該代群體最佳解的步驟省略，就是每次Map階段結束後，並不一定會進入Reduce階段，而是讓每一個粒子在Map階段獨自搜索2代、3代後，再與其他粒子做最佳適應值得比較，即是當粒子獨自搜索時，其群體最佳值的經驗並未改變，直到達到我們所設立的標準，才進入Reduce階段與其他粒子做最佳適應值的比較並更新群體最佳適應值。

簡而言之即是每當粒子獨自搜索後並不一定要立即進行群聚的階段，圖3-1為

我們的架構圖。由於本實驗PSO的粒子數目不多，故稱之為輕量級PSO。

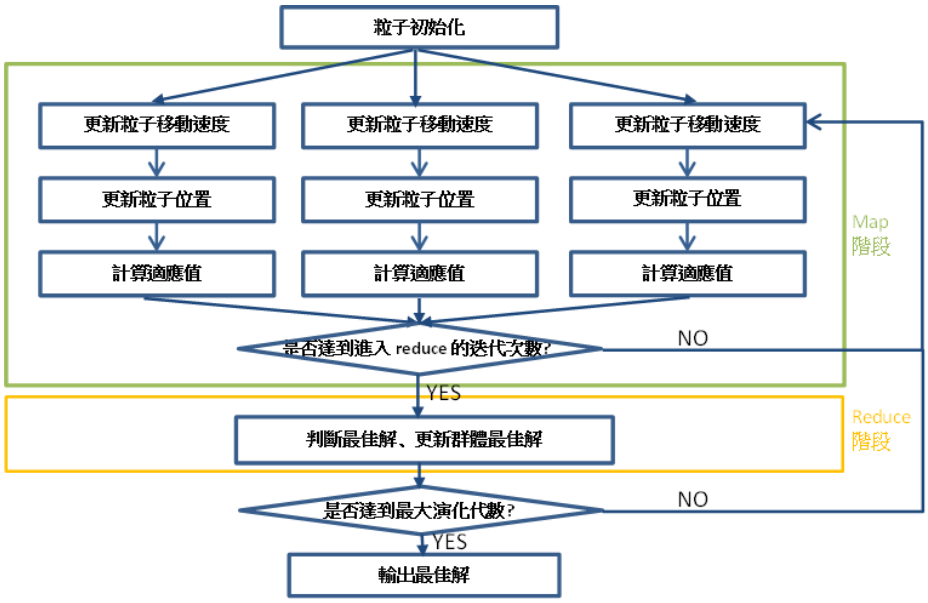

圖 3-1、輕量級 PSO 於 MapReduce 架構上的結構

3.2 實驗設計

在整個實驗中，我們會將粒子在Map階段獨自搜索的次數從1次到10次，簡單的來說就是當粒子在Map階段獨自搜索1次，表示粒子在Map階段每運行1次後則進入Reduce階段，而當粒子在Map階段獨自搜索2次，表示粒子在Map階段每運行2次後則進入Reduce階段，因此粒子在Map階段獨自搜索的次數從3次到10次，是以此類推，其中值得注意的一點是當Map階段獨自搜索次數為1時，即是MRPSO，表示粒子在Map階段每運行1次後則進入Reduce階段，因此當Map階段獨自搜索次數為1時，其概念與MRPSO的概念是一致的。

由於在眾多PSO演算法的改良研究中，尚未發現在粒子獨自搜索階段運行較為多次的研究，因此本研究將粒子獨自搜索的次數範圍定為1至10次，來觀察其最佳群體適應值的比較。

本實驗用3個基準函式來進行實驗，分別是Griewank function、Rastrigin function、Sphere function；選擇Griewank function、Rastrigin function的目的是想了解在擁有

區域最佳解時，其群體最佳解的變化，以及選擇只有單一極值Sphere function在本實驗中與有區域最佳解的function有哪些差異。整個Hadoop叢集數為10台，詳細的叢集配置會在第四章實驗結果作說明。

本實驗大致上分為3個，實驗一為不同初始值進行的實驗，意謂著每個函式進行時，都擁有不同的初始值來執行，也就是PSO演算法有進行初始化的步驟；實驗二為以相同初始值進行的實驗，意謂著每個函式進行時，是按照相同的初始值來執行，也就是PSO演算法沒有進行初始化的步驟；最後一個實驗是根據上面2個實驗結果而延伸出來的。

以下為3個函式的方程式和二維分布圖：

(a) Griewank function

方程式：$f(x) = \frac{1}{4000}\sum_{i=1}^{n} x_i^2 - \prod_{i=1}^{n} cos\left(\frac{x_i}{\sqrt{i}}\right) + 1$，$n$ 為解空間的維度

測試解空間範圍：$-600 \leq x_i \leq 600,\ i = 1,2,\dots n$

最小解：$f(x^*) = 0,\ x_i^* = 0,\ i = 1,2,\dots,n$

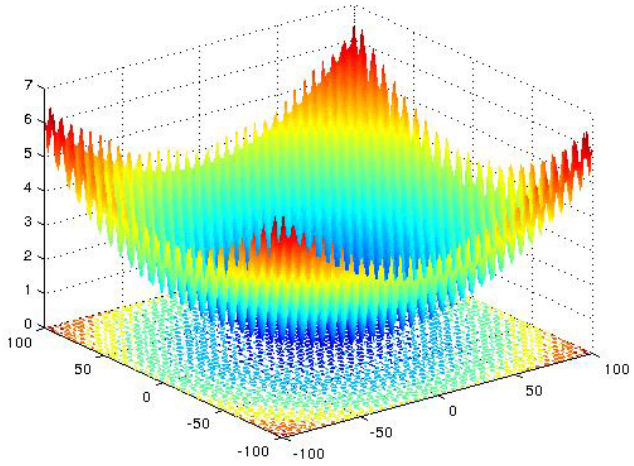

圖 3- 2、Griewank function 的 2 維分佈圖

(b) Rastrigin function

方程式：$f(x) = 10n + \sum_{i=1}^{n}(x_i^2 - 10\cos(2\pi x_i))$，$n$ 為解空間的維度

測試解空間範圍： $-5.12 \leq x_i \leq 5.12,\ i = 1,2,\ldots n$

最小解：$f(x^*) = 0,\ x_i^* = 0,\ i = 1,2,\ldots,n$

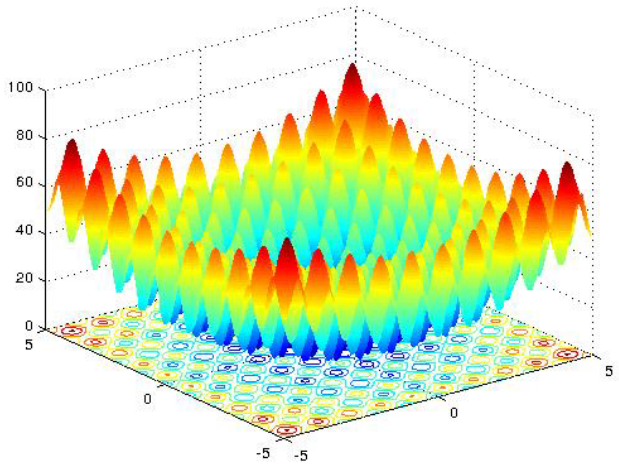

圖 3- 3、Rastrigin function 的 2 維分佈圖

(c) Sphere function

方程式：$f(x) = \sum_{i=1}^{n}(x_i^2)$ ，n 為解空間的維度

測試解空間範圍：$-5.12 \leq x_i \leq 5.12,\ i = 1,2,\ldots n$

最小解：$f(x^*) = 0,\ x_i^* = 0,\ i = 1,2,\ldots,n$

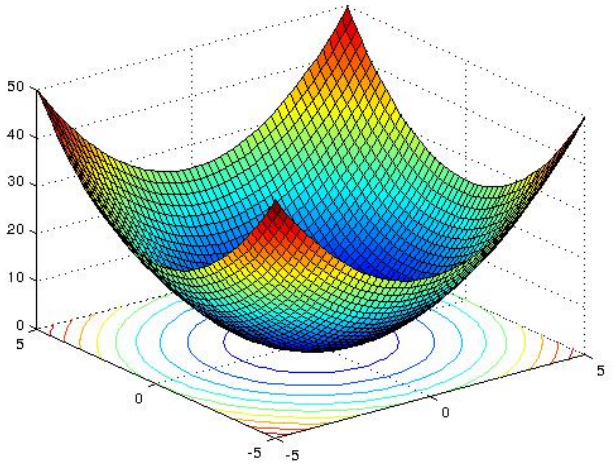

圖 3- 4、Sphere function 的 2 維分佈圖

這邊簡單介紹實驗步驟，詳細部分會在第 4 章作說明：

1. 實驗一為 Griewank function、Rastrigin function、Sphere function 皆進行初始化的步驟並在 Map 階段獨自搜索 1 至 10 次，並觀察三個函式的適應值變化。
2. 實驗二為 Griewank function、Rastrigin function、Sphere function 皆不進行初始化的步驟並在 Map 階段獨自搜索 1 至 10 次，並觀察三個函式的適應值變化。
3. 延伸實驗為比較上述兩者的實驗結果，並觀察三個函式分別在 Map 階段獨自搜索幾次時有較好的表現。

第四章 實驗結果

4.1 系統配置與相關參數設定

本研究的實驗是利用10台虛擬機架設Hadoop環境，其中一台同時作為master和 worker，每台虛擬機器的3.30 GHz CPU、8GB 記憶題、50GB 硬碟空間、ubuntu 12.04 作業系統、Hadoop版本為1.2.1，每台節點的map task數量最多同時執行2個，其中本實驗粒子數皆為8192個，維度為10，迭代最高次數為360次。

由於本研究不深入探討PSO相關參數之設定，而較專注於架構上所帶來之影響，因此本研究PSO之相關係數是參考Eberhart, R. C. and Shi, Y. (2001)所提出的論文，另外本研究在實驗中定義了mSearch的意義，這邊舉例說明，如mSearch_1為在Map階段獨自搜索1次即進入Reduce階段進行群聚的動作，mSearch_2為在Map階段獨自搜索2次即進入Reduce階段進行群聚的動作，以下以此類推，實驗以在Map階段獨自搜索1到10次來進行。

4.2 實驗步驟

1. 實驗一－不同初始值之實驗

每個function各自從粒子初始化開始，因此每個function的第一代初始值皆不同，Map獨自搜索的次數從1次到10次，觀察從初始化開始到執行完第360次迭代所花費的時間，接著觀察Map獨自搜索次數從1次到10次的群體最佳適應值變化，並找出各個function群體最佳適應值所是坐落於第幾次的Map獨自搜索次數。

2. 實驗二－相同初始值之實驗

每個的function以相同初始值的開始，以Griewank function為例，只要是執行Griewank function不管Map獨自搜索的次數是多少，其初始值數值皆是同一份初始檔案，而Rastrigin function、Sphere function皆有各自的一份初始檔案；因此每個function的第一代初始值皆相同，Map獨自搜索的次數從1次到10次，觀察從第1代開始到執行完第360次迭代所花費的時間，接著觀察Map獨自搜索次數從1次到10

次的群體最佳適應值變化，並找出各個function群體最佳適應值所是坐落於第幾次的Map獨自搜索次數。

3. 延伸實驗

根據不同初始值之實驗與相同初始值之實驗，發現Griewank function坐落於Map獨自搜索的次數差異較大而進行的延伸實驗。

4.3 實驗結果

1. 不同初始值之實驗

這邊整理3個基本函式不同初始值的實驗結果，本研究所有函式時間圖為進行本研究架構時該函式在實驗過程中的總運行時間，縱軸為以小時為單位的時間軸，橫軸為Map獨自搜索次數，以圖4-1來看，mSearch_1總運算時間大約花了5個多小時，而mSearch_10總運算時間則是花了約2個小時半，幾乎是節省了一半執行mSearch_1的總運算時間；本研究所有函式之適應值比較圖和適應值差異圖，縱軸皆為最佳群體適應值，而橫軸皆為迭代的次數，觀察每個不同次數的mSearch分別在不同代數時，其最佳群體適應值的好壞。

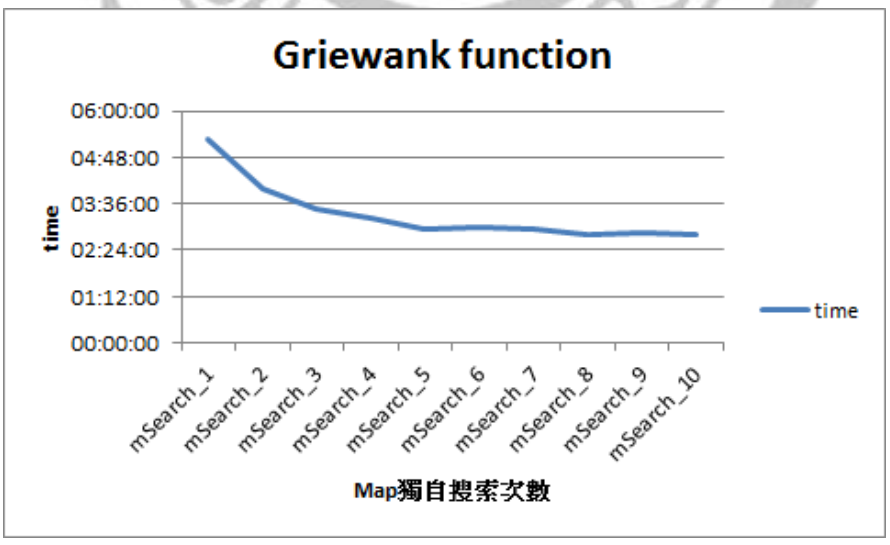

圖 4- 1、不同初始值 Griewank function 時間圖

圖4-1表示，粒子Map獨自搜索1次至10次時，實驗所花費的時間，適應值函式為Griewank function。

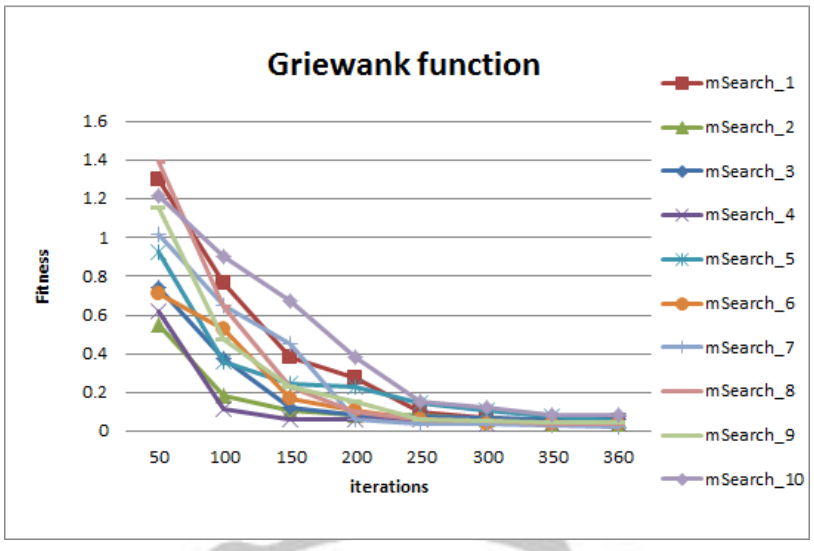

圖 4- 2、不同初始值 Griewank function 適應值比較圖

圖 4-2，Griewank function 在 Map 獨自搜索不同次數時的適應值比較，在不同代數之最佳適應值的差異較小，但以在 Map 獨自搜索 7 次後，其最後一次的適應值是最佳的。

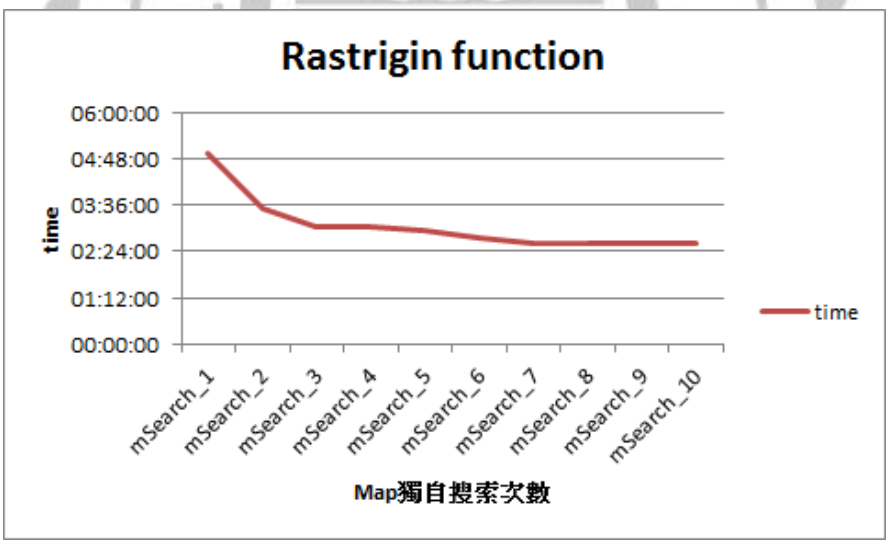

圖 4- 3、不同初始值 Rastrigin function 時間圖

圖 4-3 表示，粒子 Map 獨自搜索 1 次至 10 次時，實驗所花費的時間，適應值函式為 Rastrigin function。

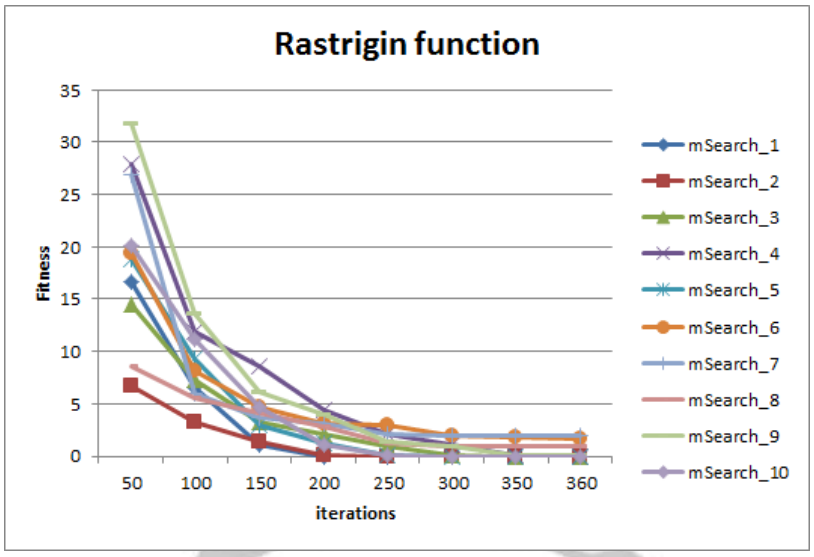

圖 4- 4、不同初始值 Rastrigin function 適應值比較圖

圖 4-4，Rastrigin function 在 Map 獨自搜索不同次數時的適應值比較，在不同代數之最佳適應值的差異較小，但以在 Map 獨自搜索 2 次後，其最後一次的適應值是最佳的。

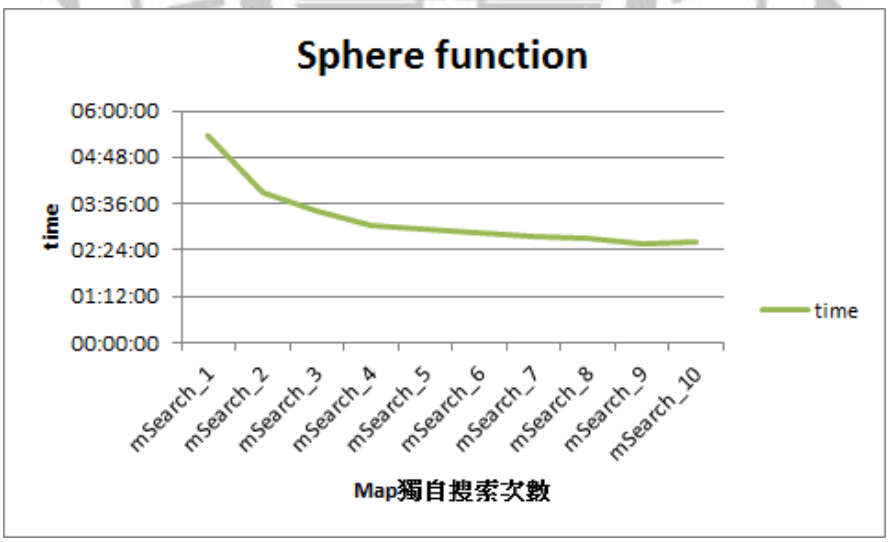

圖 4- 5、不同初始值 Sphere function 時間圖

圖4-5表示，粒子Map獨自搜索1次至10次時，實驗所花費的時間，適應值函式為 Sphere function。

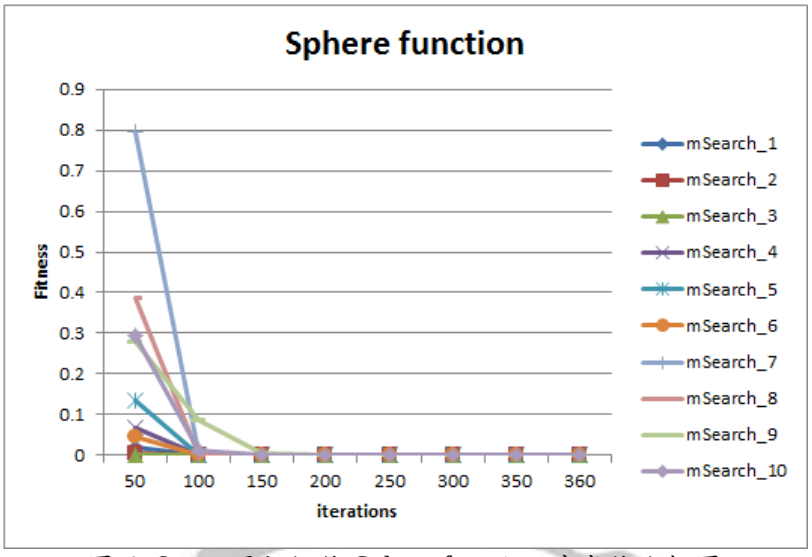

圖 4- 6、不同初始值 Sphere function 適應值比較圖

圖4-6，Sphere function在Map獨自搜索不同次數時的適應值比較，在不同代數之最佳適應值的差異較小，但以在Map獨自搜索3次後，其最後一次的適應值是最佳的。

表 4- 1、不同初始值在第 360 次迭代的適應值

	Griewank	Rastrigin	Sphere
Map獨自搜索1次	0.050875123	0.000000127	7.11E-15
Map獨自搜索2次	0.035773774	0.000000079	1.54E-18
Map獨自搜索3次	0.062806706	0.000016358	9.19E-20
Map獨自搜索4次	0.036956582	0.006616965	6.99E-15
Map獨自搜索5次	0.052458438	0.000011800	4.66E-18
Map獨自搜索6次	0.036971567	1.711970552	2.02E-15
Map獨自搜索7次	0.018530766	2.005487919	2.73E-10
Map獨自搜索8次	0.034430378	0.995450295	2.05E-10
Map獨自搜索9次	0.045201166	0.020321972	8.32E-10
Map獨自搜索10次	0.079455675	0.000288000	3.78E-07

上述表4-1為各個函式在Map中獨自運行後的群體最佳適應值，從上面的表格可以清楚的看出各個函式的群體最佳適應值。

在這個實驗中3個function的時間圖來看，可以發現在Map獨自搜索的次數越多，時間下降的越快，但是到Map獨自搜索7次後時間下降的曲線開始變的平緩，這是因為後期跳過Reduce次數差易變得不大，如Map獨自搜索9次時，在360次的迭代下，共進行40次的Reduce階段，當Map獨自搜索10次時，在360次的迭代下，共進行36次的Reduce階段；兩者進行的Reduce次數差異不大，所以時間曲線開始變的平緩。

而在適應值的比較上，可以看的出來，在Map獨自搜索10次時，其群體最佳適應值的下降程度是較差的，而由於是單一極值Sphere function其收斂的狀況較其他兩個function來的更好。

2. 相同初始值之實驗

這邊整理3個基本函式不同初始值的實驗結果，圖4-7、4-8為Griewank function的實驗結果，圖4-9、4-10為Rastrigin function實驗結果，圖4-11、4-12為Sphere function實驗結果。

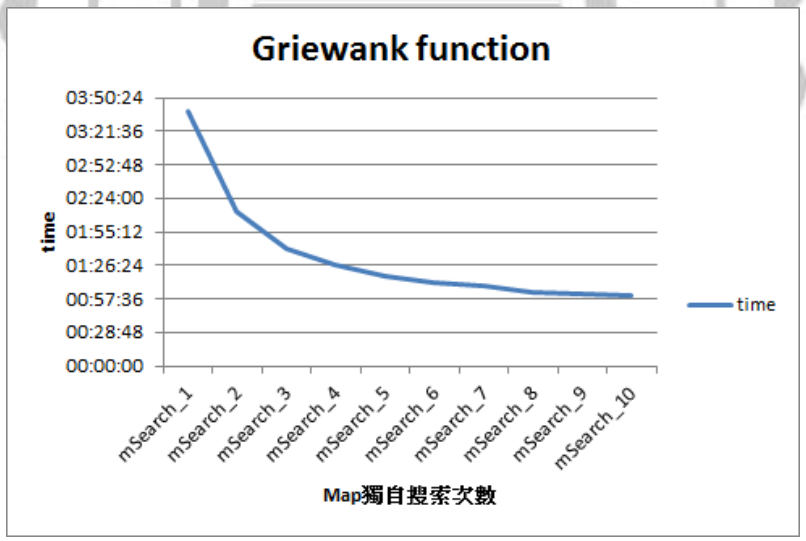

圖 4- 7、相同初始值 Griewank function 時間圖

圖 4-7 表示，粒子 Map 獨自搜索 1 次至 10 次時，實驗所花費的時間，適應值函式為 Griewank function 。

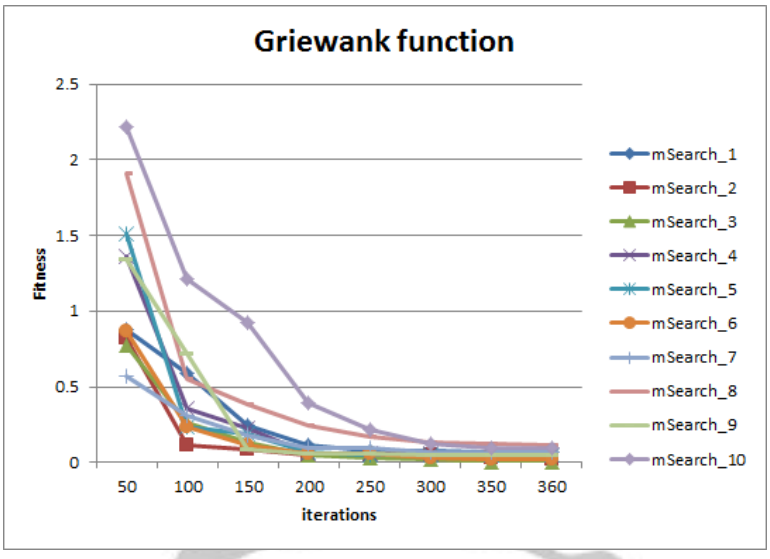

圖 4- 8、相同初始值 Griewank function 適應值比較圖

圖 4-8，Griewank function 在 Map 獨自搜索不同次數時的適應值比較。

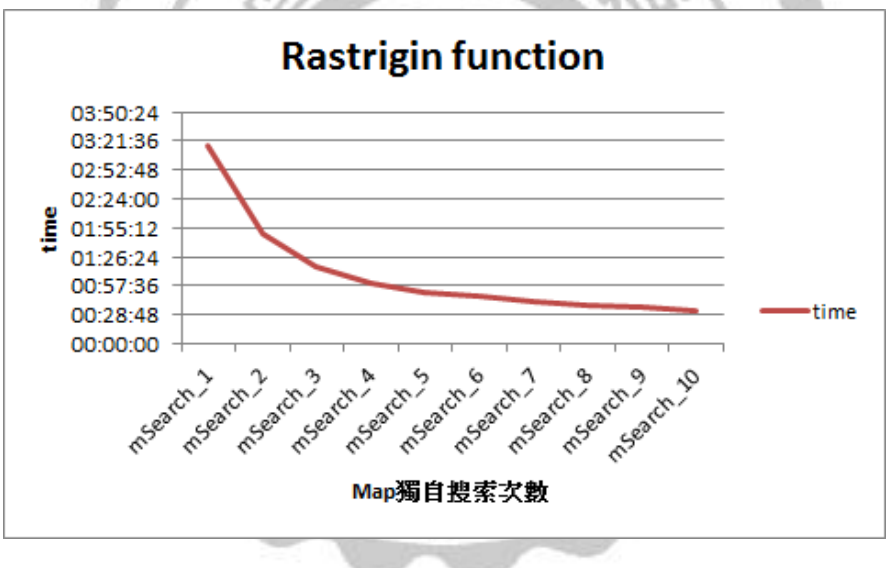

圖 4- 9、相同初始值 Rastrigin function 時間圖

圖 4-9 表示，粒子 Map 獨自搜索 1 次至 10 次時，實驗所花費的時間，適應值函式為 Rastrigin function。

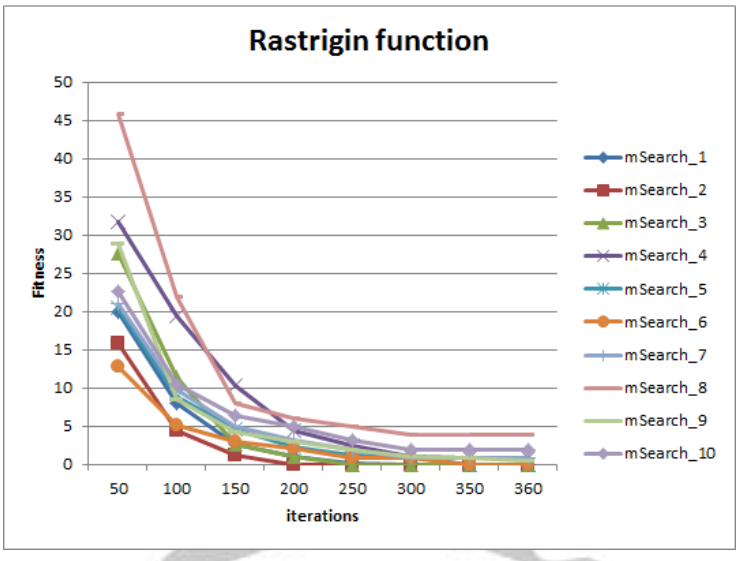

圖 4- 10、相同初始值 Rastrigin function 適應值比較圖

圖 4-10，Rastrigin function 在 Map 獨自搜索不同次數時的適應值比較。

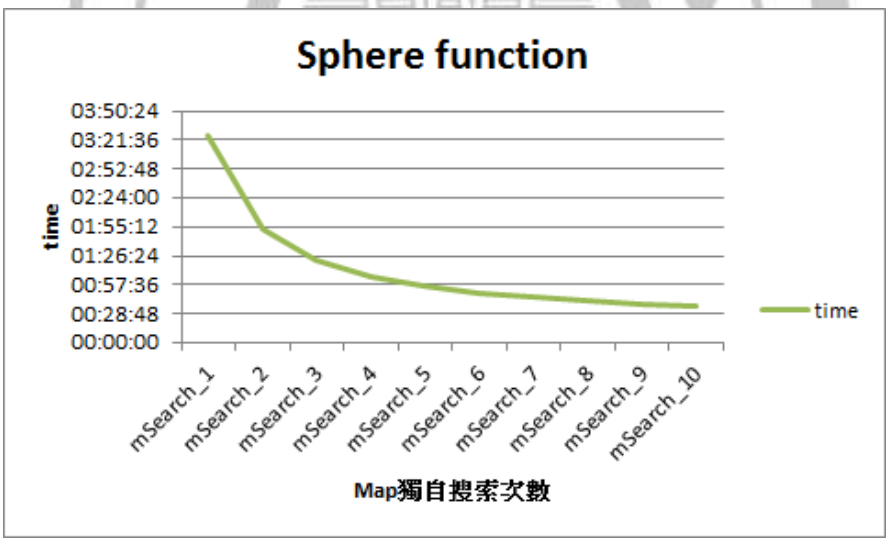

圖 4- 11、相同初始值 Sphere function 時間圖

圖 4-11 表示，粒子 Map 獨自搜索 1 次至 10 次時，實驗所花費的時間，適應值函式為 Sphere function。

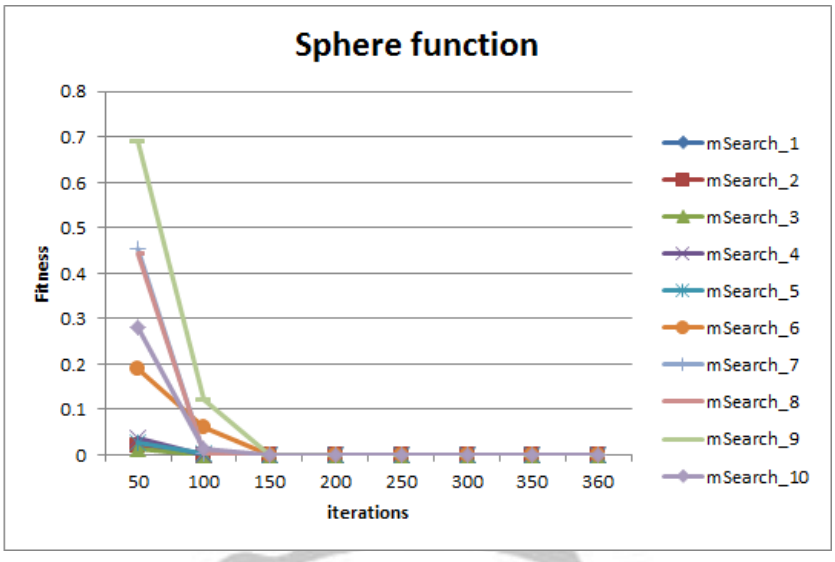

圖 4- 12、相同初始值 Sphere function 適應值比較圖

圖4-12，Sphere function在Map獨自搜索不同次數時的適應值比較。

表 4- 2、相同初始值在第 360 次迭代的適應值

	Griewank	Rastrigin	Sphere
Map獨自搜索1次	0.071326036	0.0000463000	9.31E-15
Map獨自搜索2次	0.024102191	0.0000000004	5.61E-18
Map獨自搜索3次	0.003004954	0.0000003570	2.16E-17
Map獨自搜索4次	0.046731198	0.9954867027	2.24E-18
Map獨自搜索5次	0.046725016	0.0707240433	1.42E-11
Map獨自搜索6次	0.020830018	0.0060508933	1.24E-13
Map獨自搜索7次	0.066403645	0.9953986098	1.31E-12
Map獨自搜索8次	0.112315770	3.8915104170	1.23E-12
Map獨自搜索9次	0.044442551	0.5787578837	5.32E-11
Map獨自搜索10次	0.094591093	2.0195929560	1.15E-10

根據上面的實驗結果，可以發現這兩次的實驗中，Griewank function在擁有最佳適應值的結果上，發現其落在Map獨自搜索次數的結果差異較大。

在這個實驗中3個function的時間圖與上一個實驗曲線相差不大，較大的差異在於不需初始化，因此在整體的時間花費上減少了約2個小時。在適應值的比較上，其變化也與上一個實驗是相似的，比較令人驚訝的地方在於Griewank function，從兩次的實驗中，可以觀察出Griewank function群體最佳適應值坐落的位置差異較大，一個是落在Map獨自搜索7次，另一個則是落在Map獨自搜索3次。

3. 延伸實驗

由於上述兩個實驗的結果，我們的延伸實驗只針對Griewank function來進行，以相同初始值之Griewank function，將Map獨自搜索1次至10次，各執行10次，共100次，這次的延伸實驗是為了觀察Griewank function群體最佳適應值較多坐落在第幾次的Map獨自搜索。以下為比較結果。

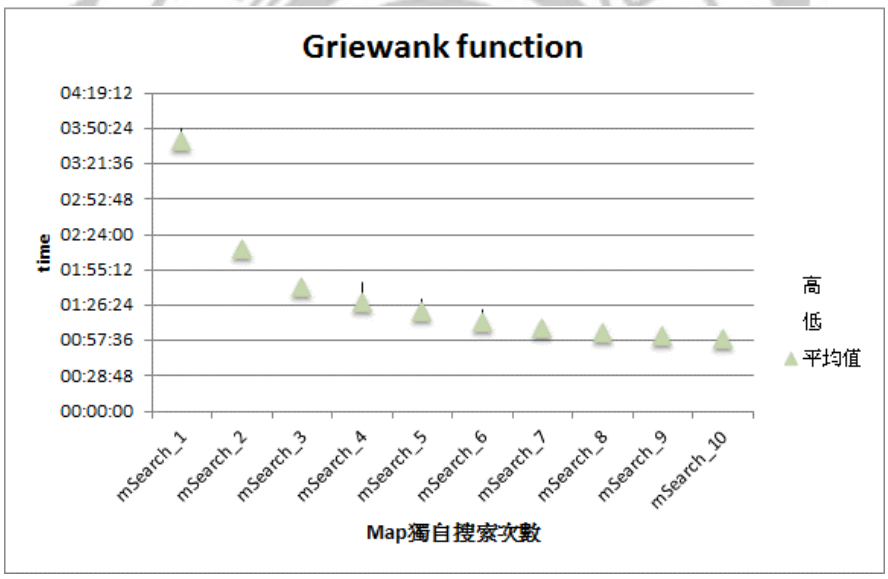

圖 4- 13、10 次 Griewank function 時間比較圖

圖 4-13 為進行 10 次相同實驗所耗用的時間圖。

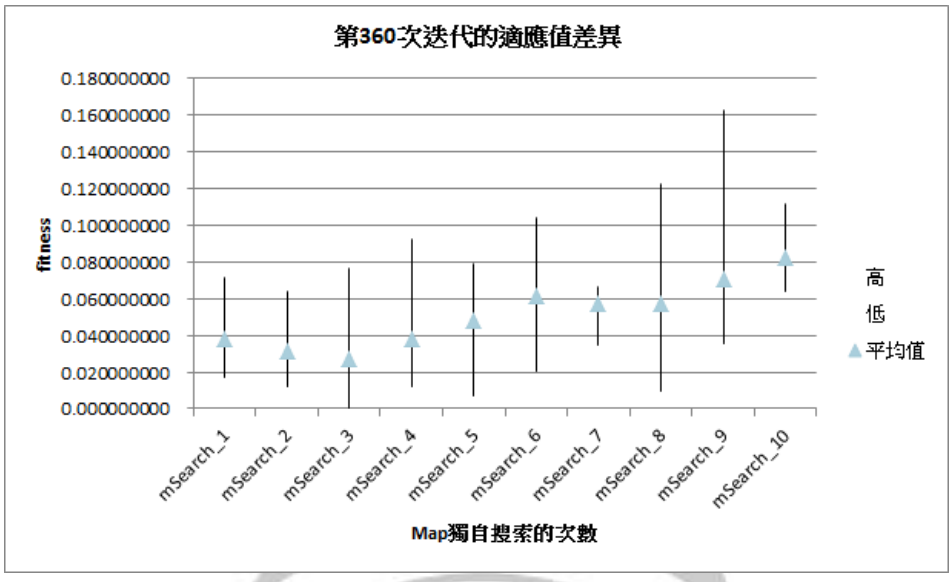

圖 4- 14、第 360 次迭代的適應值差異圖

圖 4-14 為第 360 次迭代 Griewank function 在 Map 獨自搜索 1 次至 10 次的適應值變化，其中包括最差適應值、最佳適應值以及適應值的平均值。

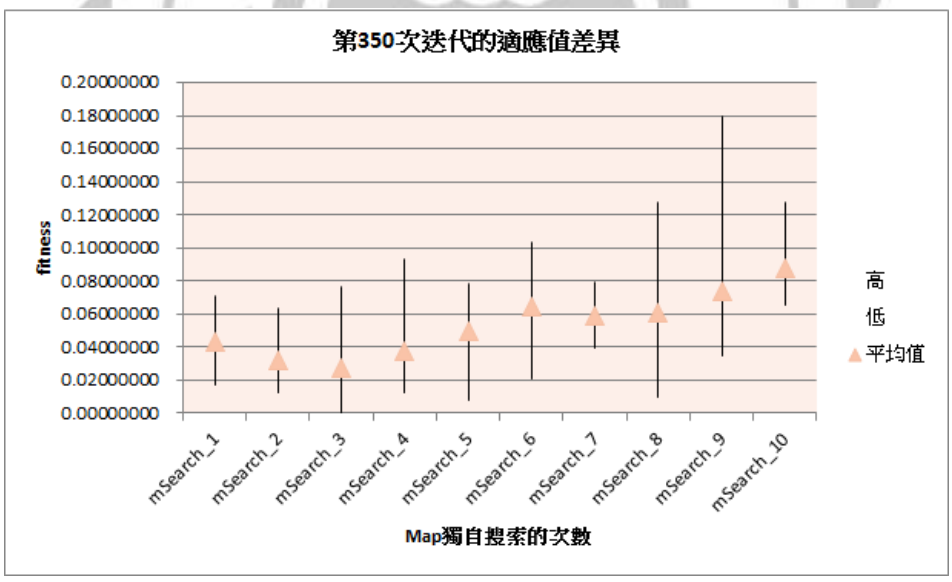

圖 4- 15、第 350 次迭代的適應值差異圖

圖 4-15 為第 350 次迭代 Griewank function 在 Map 獨自搜索 1 次至 10 次的適應值變化，其中包括最差適應值、最佳適應值以及適應值的平均值。

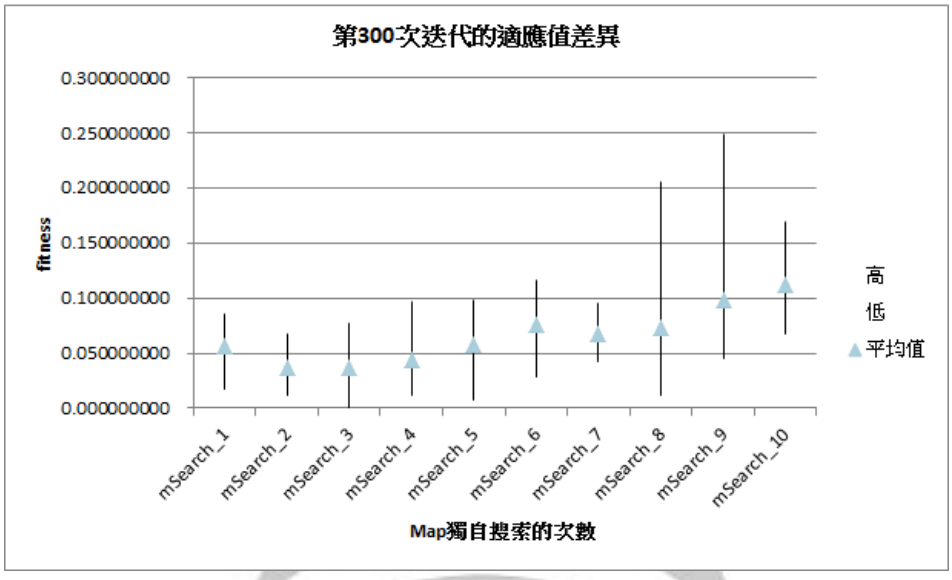

圖 4- 16、第 300 次迭代的適應值差異圖

圖 4-16 為第 300 次迭代 Griewank function 在 Map 獨自搜索 1 次至 10 次的適應值變化，其中包括最差適應值、最佳適應值以及適應值的平均值。

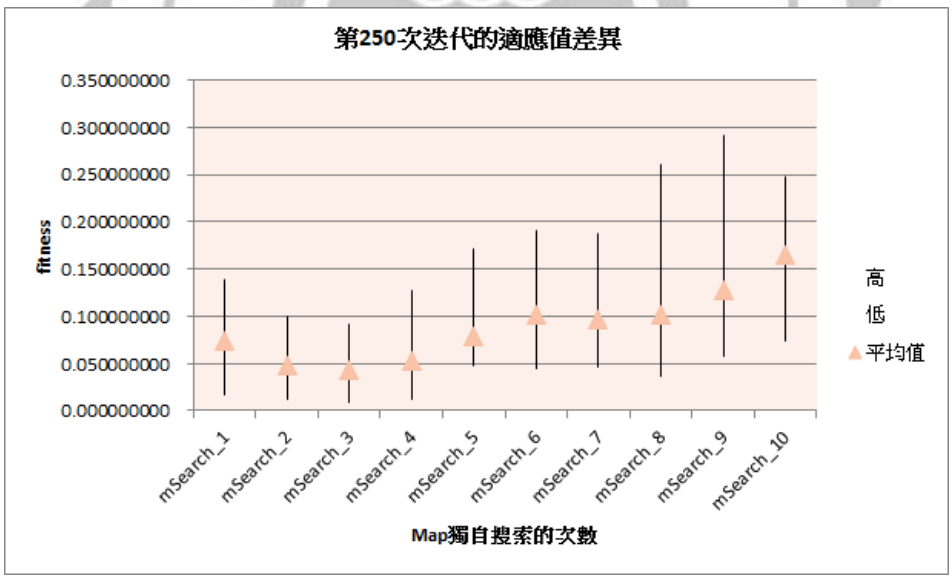

圖 4- 17、第 250 次迭代的適應值差異圖

圖 4-17 為第 250 次迭代 Griewank function 在 Map 獨自搜索 1 次至 10 次的適應值變化，其中包括最差適應值、最佳適應值以及適應值的平均值。

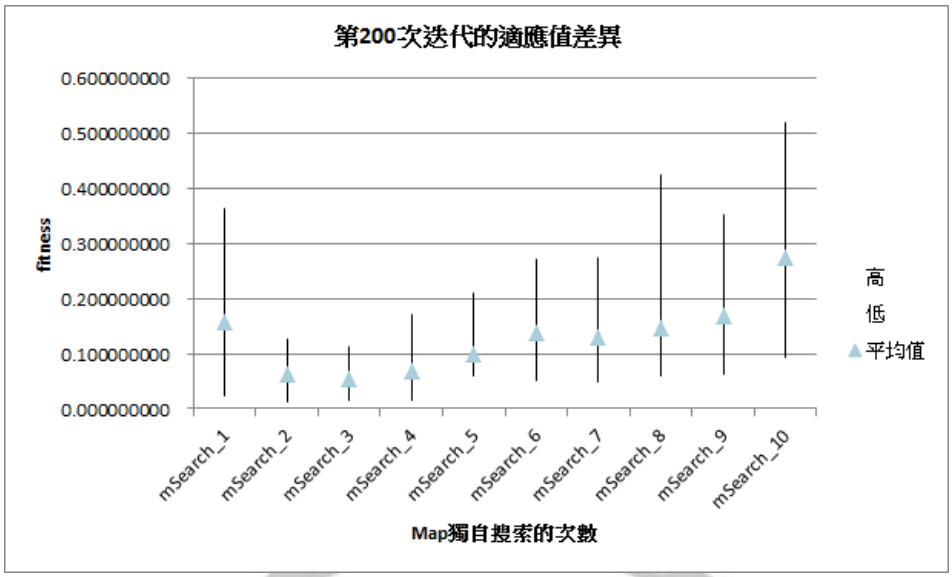

圖 4- 18、第 200 次迭代的適應值差異圖

圖 4-18 為第 200 次迭代 Griewank function 在 Map 獨自搜索 1 次至 10 次的適應值變化，其中包括最差適應值、最佳適應值以及適應值的平均值。

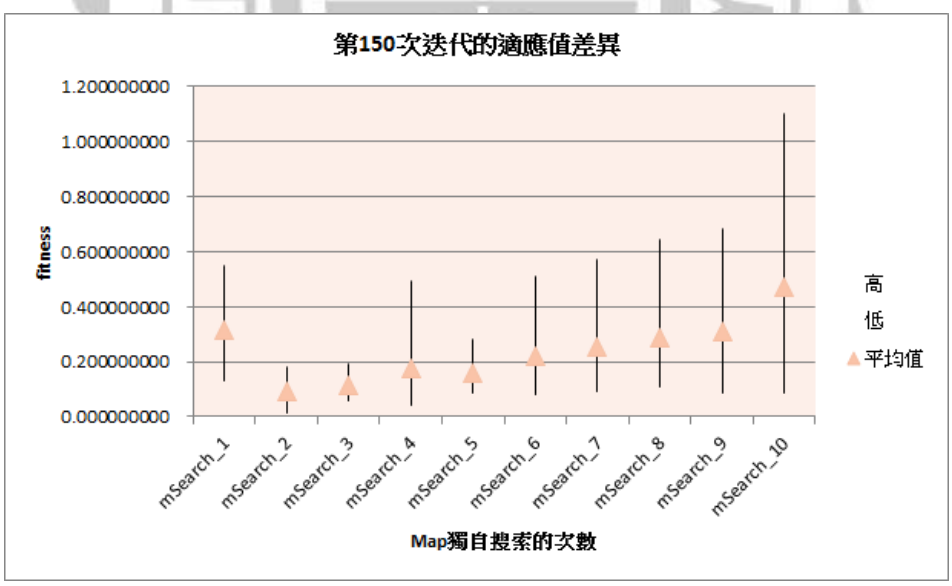

圖 4- 19、第 150 次迭代的適應值差異圖

圖 4-19 為第 150 次迭代 Griewank function 在 Map 獨自搜索 1 次至 10 次的適應值變化，其中包括最差適應值、最佳適應值以及適應值的平均值。

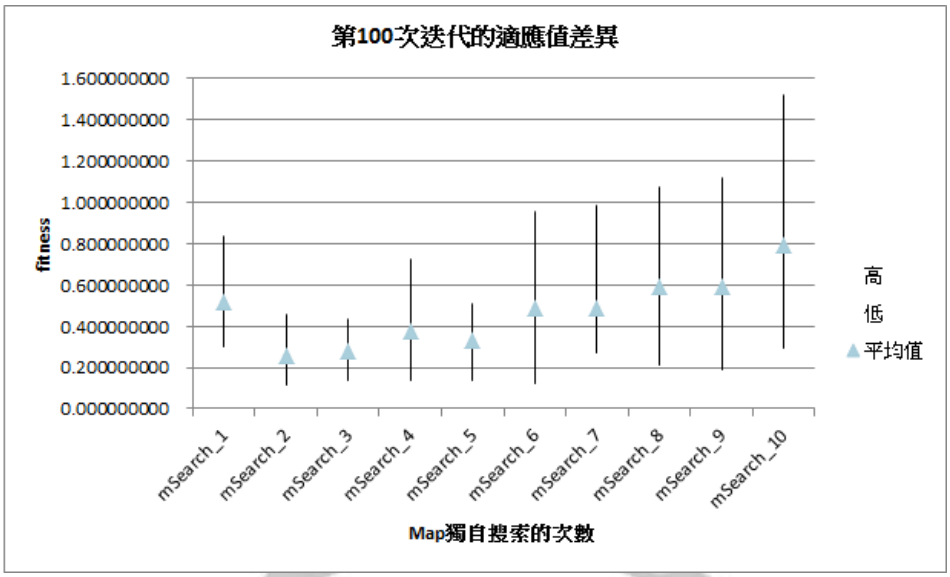

圖 4- 20、第 100 次迭代的適應值差異圖

圖 4-20 為第 100 次迭代 Griewank function 在 Map 獨自搜索 1 次至 10 次的適應值變化，其中包括最差適應值、最佳適應值以及適應值的平均值。

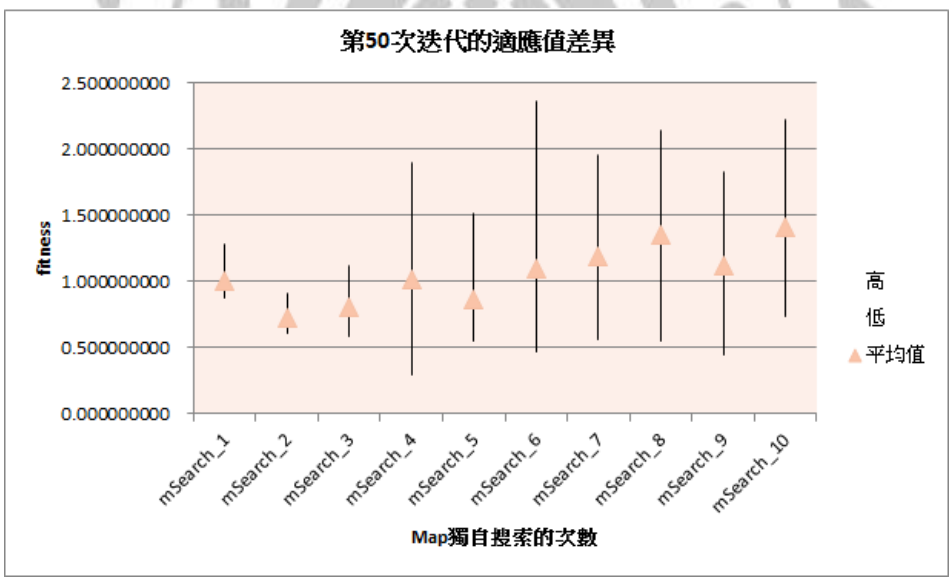

圖 4- 21、第 50 次迭代的適應值差異圖

圖 4-21 為第 50 次迭代 Griewank function 在 Map 獨自搜索 1 次至 10 次的適應值變化，其中包括最差適應值、最佳適應值以及適應值的平均值。

延伸實驗結果顯示Griewank function群體最佳適應值坐落於Map獨自搜索1、2、3和8次的次數各為2次，而坐落於Map獨自搜索4、5次的次數為1次。從適應值差異

圖上來看，可以發現坐落於Map獨自搜索1、2、3次的差異比坐落於Map獨自搜索8次來的小。

根據上述三個實驗來看，可以發現以下幾點：

1. Map獨自搜索的次數越大，時間縮短的越快，而到了Map獨自搜索8、9、10次時,可以發現時間曲線趨於平緩,是由於三者進入Reduce階段的次數差異不大，分別是45、40以及36次。
2. Map獨自搜索的次數越大，其最佳群體適應值的表現不一定最佳，反而比Map獨自搜索2、3次來的差。
3. 每個函式坐落於最佳Map獨自搜索的次數皆不相同。
4. Map獨自搜索2次時，其最佳群體適應值與Map獨自搜索1次也就是所謂的MRPSO相比，差異不大，但在運行的時間上減少了許多。
5. 延伸實驗中，迭代次數為前100代且Map獨自搜索2至5次時，其最佳群體適應值之差異相差不多。而到了迭代次數後期最佳群體適應值之差異則相當明顯。

第五章 結論與未來展望

5.1 結論與未來展望

從上述的實驗結果來看，Map獨自搜索能明顯降低實驗所花費的時間，但Map獨自搜索次數越多，其在最後一代的適應值結果往往不是最佳的，這表示如果粒子一直不與外界作交流，其成效是差的。就像是人們只專心埋頭於自己的工作，而忽略了外界的變化，那麼這樣可能就無法得知有新的事物出現，而且該事物能快速地幫助自己處理目前的工作。

另外在以相同初始值進行的實驗中，可以看出Rastrigin function以及Sphere function 最佳適應值所坐落在減少Reduce次數的位置上並沒有很大的差異，而是Griewank function有較大的不同，經過幾次的實驗，可以發現最佳適應值似乎較常坐落於Map獨自搜索2次、3次以及8次，但就以上面2個function的經驗來看，採用減少Reduce 2次、3次是較為保險的，因為其適應值的差異程度較小。

雖然本實驗有些許的成果，但由於實驗次數不多，故無法確定目前的結果一定是最佳的，因此在未來，一方面仍是希望會進行更多的實驗，而另一方面，希望能夠在剛開始迭代的期間內增加 Map 獨自搜索次數，而在後期迭代期間內減少 Map 獨自搜索次數，來達到節省時間並精準適應值的目標。

5.2 研究限制

對於本研究的成果只適用於在本研究的環境架構下以及PSO演算法之相關參數設定，研究本之相關設定如下：

1. 本研究的系統環境為ubuntu12.04、hadoop版本為1.2.1、叢集數量為10台節點，其中master又兼具worker的功能。
2. 本研究PSO的相關參數設定採用Eberhart, R. C. and Shi, Y. (2001)所提出的相關參數值。
3. 本研究之成果會依不同函式而有不一樣的成果。

參考文獻

一、中文文獻

1. 紀玟君(2013)，使用 MapReduce 之平行化詢問式粒子群演算法，碩士論文，國立臺灣大學工學院工程科學及海洋工程學系。
2. 王宏仁，2012，Hadoop 技術協助企業解決巨量資料難題，http://www.ithome.com.tw/node/73977

二、英文文獻

3. Aljarah, I. and Ludwig, S. A. (2012), “Parallel particle swarm optimization clustering algorithm based on mapreduce methodology,” 2012 Fourth World Congress on Nature and Biologically Inspired Computing (NaBIC). doi: 10.1109/NaBIC.2012.6402247
4. Apache Hadoop. Available at http://hadoop.apache.org/.
5. Baldeschwieler, E., (2008), Yahoo! Launches World’s Largest Hadoop Production Application. http://yahoohadoop.tumblr.com/post/98098649696/yahoo-launches-worlds-largest-hadoop-production
6. Celio Di Cellio Dias, P., Henrique Outi Kauffmann, L. and Fernandes, G. B., (2014). Big Data: How to Turn Big Data Into Great Information, São Paulo :Serasa Experian.
7. Chang, F., Dean, J., Ghemawat, S., Hsieh, W. C., Chandra, T., Fikes, A., and Gruber, R. E. (2006), “Bigtable: ADistributed Storage System for Structured Data,” 7th USENIX Symposium on Operating Systems Design & Implementation, pp.205-218.
8. Clerc, M. (2004), “Discrete particle swarm optimization, illustrated by the traveling salesman problem,” in New optimization techniques in engineering, ed:

Springer, pp. 219-239.

9. Dean, J. and Ghemawat, S. (2004), “MapReduce: Simplified Data Processing on Large Clusters,” 6th Symposium on Operating Systems Design & Implementation, San Francisco, CA.
10. Dean, J. and Ghemawat, S. (2008), “MapReduce: simplified data processing on large clusters,” Communications of the ACM, Vol. 51, pp. 107-113.
11. Dittrich, J. and Quiané-Ruiz, J. A. (2012), “Efficient big data processing in Hadoop MapReduce,” Proceedings of the VLDB Endowment, Vol. 5, pp. 2014-2015.
12. Douglas, L. (2001), 3D Data Management: Controlling Data Volume, Velocity and Variety.
13. Eberhart, R. C. and Shi, Y. (1998), “Comparison between genetic algorithms and particle swarm optimization,” in EVolutionary Programming VII, pp. 611-616.
14. Eberhart, R. C. and Shi, Y. (2000), “Comparing inertia weights and constriction factors in particle swarm optimization,” in EVolutionary Computation, 2000. Proceedings of the 2000 Congress on, pp. 84-88.
15. Eberhart, R. C. and Shi, Y. (2001), “Particle Swarm Optimization: Developments, Applications and Resources,” Proceedings of the IEEE Congress on Evolutionary Computation. Piscataway , NJ : IEEE Service Center , vol. 1, pp. 81-86.
16. Ghemawat, S., Gobioff, H., and Leung, S. T. (2003), “The Google File System”, 19th ACM Symposium on Operating Systems Principles, Lake George, NY.
17. Harsoor, A. S. and Patil, A., (2015), “Forecast of Sales of Walmart Store Using Big Data Applications,” IJRET: International Journal of Research in Engineering and Technology, vol. 4, pp.51-59
18. Heppner, F. and Grenander, U. (1990), “A stochastic nonlinear model for coordinated bird flocks,” American Association for the Advancement of Science,

Washington, DC, USA.

19. Hu, X. and Eberhart, R. (2002), “Multiobjective optimization using dynamic neighborhood particle swarm optimization,” in Computational Intelligence, Proceedings of the World on Congress on, pp. 1677-1681.
20. Jin, C., Vecchiola, C. and Buyya, R. (2008), “MRPGA: an extension of MapReduce for parallelizing genetic algorithms ,” in eScience, 2008. eScience’08. IEEE Fourth International Conference on, pp. 214-221.
21. Kennedy, J. and Eberhart, R. (1995), “Particle swarm optimization,” in Proceedings of IEEE international conference on neural networks, pp. 1942-1948.
22. Krishnan, S., Baru, C., and Crosby, C. (2010), “Evaluation of MapReduce for gridding LIDAR data,” in Cloud Computing Technology and Science (CloudCom), 2010 IEEE Second International Conference on, pp. 33-40.
23. Lin, F. S., Shen, C. P., Sung, H. Y., Lam, Y. Y., Lin, J. W., and Lai, F. (2013),“A High Performance Cloud Computing Platform for mRNA Analysis,” the 35th Annual International Conference of the IEEE Engineering in Medicine and Biology Society, Osaka, Japan, pp. 1510-1513.
24. McKenna, A., Hanna, M., Banks, E., Sivachenko, A., Cibulskis, K., Kernytsky, A. (2010), et al., “The Genome Analysis Toolkit: a MapReduce framework for analyzing next-generation DNA sequencing data,” Genome research, Vol. 20, pp. 1297-1303.
25. McNabb, A. W., Monson, C. K., and Seppi, K. D. (2007a), “MRPSO: MapReduce particle swarm optimization,” in Proceedings of the 9th annual conference on Genetic and eVolutionary computation, pp. 177-177
26. McNabb, A. W., Monson, C. K., and Seppi, K. D. (2007b), “Parallel PSO using mapreduce,” in EVolutionary Computation, 2007. CEC 2007. IEEE Congress on, pp. 7-14.

27. Messerschmidt, L. and Engelbrecht, A. P. (2004), “Learning to play games using a PSO-based competitive learning approach,” EVolutionary Computation, IEEE Transactions on, Vol. 8, pp. 280-288.
28. Plimpton, S. J., Devine, K. D. (2011), “MapReduce in MPI for Large-scale graph algorithms,” Parallel Computing, Vol. 37, No. 9, pp. 610-632.
29. Reynolds, C. W. (1987), “Flocks, herds and schools: A distributed behavioral model,” ACM SIGGRAPH Computer Graphics, Vol. 21, pp. 25-34.
30. Sadasivam, G. S. and Selvaraj, D. (2010), “A novel parallel hybrid PSO-GA using MapReduce to schedule jobs in Hadoop data grids,” in Nature and Biologically Inspired Computing (NaBIC), 2010 Second World Congress on, pp. 377-382.
31. Shen, C. P., Zhou, W., Lin, F. S., Sung, H. Y., Lam, Y. Y., Chen, W., Lin, J. W., Pan, M. K., Chiu, M. J., and Lai, F. (2013), “Epilepsy Analytic System with Cloud Computing,” the 35th Annual International Conference of the IEEE Engineering in Medicine and Biology Society, Osaka, Japan, pp. 1644-1647.
32. Shi, Y. and Eberhart, R. (1998), “A modified particle swarm optimizer,” in EVolutionary Computation Proceedings, 1998. IEEE World Congress on Computational Intelligence., The 1998 IEEE International Conference on, pp. 69-73.
33. Shi, Y. and Eberhart, R. C. (1998), “Parameter selection in particle swarm optimization,” in EVolutionary Programming VII, pp. 591-600.
34. Taylor, R. C. (2010), “An overview of the Hadoop/MapReduce/HBase framework and its current applications in bioinformatics,” BMC bioinformatics 2010 11(Suppl 12):S1, doi: 10.1186/1471-2105-11-S12-S1
35. Trelea, I. C. (2003), “The particle swarm optimization algorithm: convergence analysis and parameter selection,” Information processing letters, Vol. 85, pp. 317-325.

36. Van den Bergh, F. and Engelbrecht, A. P. (2000), “Cooperative learning in neural networks using particle swarm optimizers,” South African Computer Journal, pp. 84-90.

37. Zicari, R. V., (2014), “Big Data: Challenges and Opportunities,” Big Data computing, pp. 103−128.