嶺東科技大學
LING TUNG UNIVERSITY

資訊管理系

基於 Hadoop 之商品排行與程式效能最佳化研究

指導教授：　陳明華　教授

組員名單：　黃家煜　A18C069
余家志　A18C138
陳致瑜　A18C147

中　華　民　國　一　百　零　五　年　五　月

嶺東科技大學
資訊管理系

基於 Hadoop 之商品排行與程式效能最佳化研究

中華民國一零五年五月

嶺東科技大學
LING TUNG UNIVERSITY

資訊管理系

基於 Hadoop 之商品排行與程式效能最佳化研究

指導教授：　陳明華　教授

組員名單：　黃家煜　A18C069
余家志　A18C138
陳致瑜　A18C147

中　華　民　國　一　百　零　五　年　五　月

嶺東科技大學
LING TUNG UNIVERSITY

資訊管理系專題口試委員審定書

基於 Hadoop 之商品排行與程式效能最佳化研究

指導教授：　陳明華　教授

組員名單：　黃家煜　A18C069
余家志　A18C138
陳致瑜　A18C147

指導教授：＿＿＿＿＿＿＿＿

口試委員：＿＿＿＿＿＿＿＿

＿＿＿＿＿＿＿＿

中　華　民　國　一　百　零　五　年　五　月

誌　　謝

本專題報告得以順利完成，首先要感謝恩師陳明華老師細心引導我們，耐心的協助我們，克服研究過程中所面臨的困難，給予我們最大的協助，使本專題得以順利完成。

研究報告口試期間，感謝萬絢老師、陳建興老師不辭辛勞細心審閱，不僅給予我們指導，並且提供寶貴的建議，使我們的專題內容以更加完善，在此由衷的感謝。

最後，感謝系上諸位老師在各學科領域的熱心指導，增進商業管理知識範疇，在此一併致上最高謝意。

黃家煜
余家志
陳致瑜　謹誌
中華民國105年5月於嶺東

摘　　要

鑑於現在電子商務快速發展的趨勢，有更多的人因為方便與快速而以上網購物代替出門採買，使我們能夠以更快速更便利的方式購買到想要與喜歡的商品，讓自己有更多的時間去做其他的事情，而正因為現今社會快速方便的網路傳遞，對於以資料分析也因為資料量越來越龐大而顯得困難。

我們對這樣趨勢感到好奇，因而參加了 2015 Etu Hadoop 競賽，而在比賽結果出爐之後我們對於第一名 11 秒左右的運算時間，產生了挑戰的興趣，因此我們決定以此為目標來做為我們專題的研究。

本研究使用OracleVMVirtualBox建立的Hadoop環境配以自行撰寫的MapReduce來建立商品銷售額前二十名與程式效能優化。將資料以Map程式取出其中進行訂購的資料並計算其銷售額，再以Reduce程式對所有Split做銷售總額的加總，最後使用另外的MapReduce程式進行前二十名的排序。而最後的成果也將效能提升達到 9 秒左右的運算時間。

關鍵字：*Hadoop*、*MapReduce*、*Etu Hadoop競賽*、*VirtualBox*

目　　錄

摘　　要 I

目　　錄 II

表目錄 IV

圖目錄 V

第一章　　緒論 1

1.1　研究動機與目的 1

1.2　研究架構與流程 2

1.2.1 研究架構 2

1.2.2 研究流程 3

第二章　　文獻回顧與研究方法 4

2.1　Hadoop 4

2.2　MapReduce 5

第三章　　研究設計 7

3.1　研究工具 - Oracle VM VirtualBox 7

3.2　實驗樣本 8

3.3　Hadoop 8

3.4　MapReduce 軟體架構 9

3.4.1　MapReduce 軟體架構 9

3.4.2　MapReduce 的執行過程 10

第四章　　結果與討論 15

4.1　VirtualBox 下分散式的 Hadoop 環境 15

4.2 Main 程式 16
4.3 Map 階段 17
4.4 Reduce 階段 18
4.5 第二次 Map 階段 19
4.6 第二次 Reduce 階段 20
4.7 輸出結果 21
第五章 結論與未來展望 23
5.1 結論 23
5.2 未來展望 23
參考文獻 24
附錄：程式原始碼 25
附錄 A:Main 25
附錄 B:Map 26
附錄 C:Reduce 26
附錄 D:第二階段 MAP 27
附錄 E:第二階段 Reduce 27

表目錄

表 3.1 VirtualBox 的特點 7

表 3.2 Hadoop 之特性 8

表 4.1 使用不同節點及演算法下的運算時間.....22

圖目錄

圖 1.1 研究流程圖 3
圖 3.1 樣本格式 8
圖 3.2 MapReduce 2.0 架構 10
圖 3.3 MapReduce 執行過程 11
圖 3.4 MapReduce 執行過程一 11
圖 3.5 MapReduce 執行過程二 12
圖 3.6 MapReduce 執行過程三 12
圖 3.7 MapReduce 執行過程四 13
圖 3.8 MapReduce 執行過程五 13
圖 3.9 MapReduce 執行過程六 14
圖 4.1 VirtualBox 管理介面 15
圖 4.2 Hadoop 環境 15
圖 4.3 Datanodes 列表 16
圖 4.4 Main 程式的執行過程 16
圖 4.5 Map 程式的執行過程 17
圖 4.6 Reduce 程式的執行過程 18
圖 4.7 第二次 Map 程式的執行過程 19
圖 4.8 第二次 Reduce 程式的執行過程 20
圖 4.9 執行時間 20
圖 4.10 取前二十名之輸出結果 21

第一章　　緒論

1.1 研究動機與目的

在2013年11月中國阿里巴巴旗下天貓商城，創下單日銷售人民幣350億元的世界紀錄，總成交筆數量一口氣衝破1.71億筆。如此巨大的資料若以過去傳統檔案系統與關聯式資料庫的方式來進行處理是無法應付的。

隨著電子商務的普及，加上現代幾乎人手一台的智慧型手機而不需再桌電與筆電前煩惱購物，由於網際網路的發達與便捷人們開始利用電子商務而賣家們也開始著重于使用這些便捷的網路場所進行線上商品拍賣。

電子商務的快速發展，也讓關於商品的資料量跟著爆發式的增長，傳統檔案系統與關聯式資料庫逐漸無法承載過度膨脹的資料量，透過硬體升級、採用特殊的演算機制，仍然無法有效解決資料持續增長所造成的系統效能不彰問題，因應這項趨勢Google及Apache發展出Hadoop及MapReduce來處理巨量的資料。

我們對這樣趨勢感到好奇，因而參加了 2015 Etu Hadoop 競賽，而在比賽結果出爐之後我們對於第一名 11 秒左右的運算時間，產生了挑戰的興趣，因此我們決定以此為目標來做為我們專題的研究。

本研究將運用 Hadoop 生態系的程式，將純文字原始資料經過運算，得出電子商務網站的商品銷售排行。求取正確產生商業價值及執行效率，以降低企業營運成本。

1.2 研究架構與流程

1.2.1 研究架構

將研究架構分別為以下伍章：

第壹章 緒論

敘述本研究之動機、目的、架構與流程。

第貳章 參考文獻

探討與本研究相關之國內外學者的論文及期刊。

第參章 研究設計

說明本研究使用之工具與實驗樣本及本研究所撰寫之程式。

第肆章 成果與討論

說明本研究之成果與討論。

第伍章 結論與未來展望

對於本研究之結論與未來展望。

1.2.2 研究流程

本研究之流程圖如圖1.1所示。首先確定研究方向與目標，先利用Oracle VM VirtualBox建立單機式Hadoop環境來撰寫MapReduce程式，再以完成之後的程式為基礎進行修改或增加節點來盡可能的縮短時間並保持結果正確率。

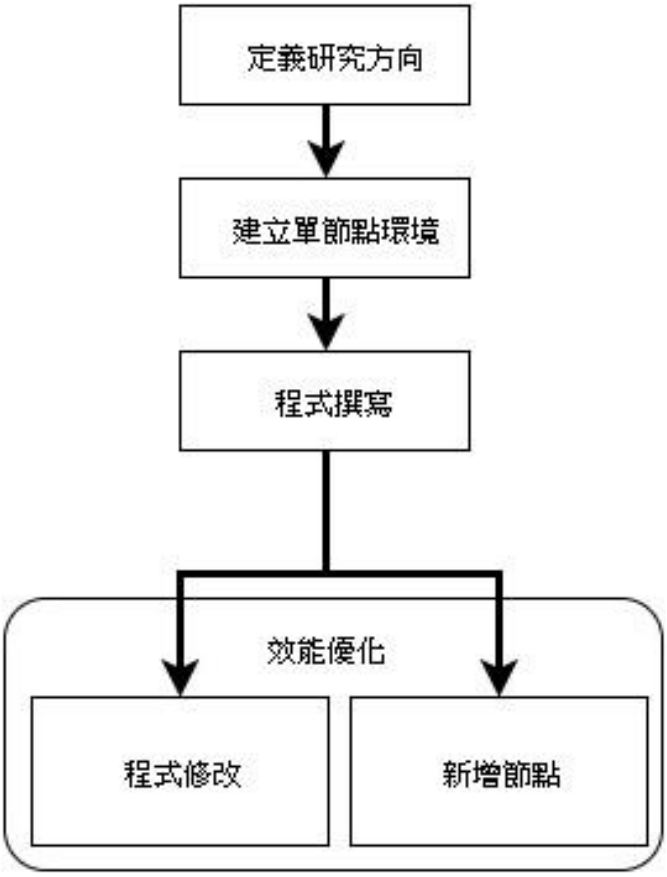

圖 1.1 研究流程圖

第二章　文獻回顧與研究方法

由於Hadoop同時擁有高可靠性、高擴展性、高效性、高容錯性及低成本，同時也可以處理非結構化、非關聯式的資料等優點，而讓其受到許多企業及研究機構關注，因此，在近年來在應用與改進方面皆有許多研究工作。

2.1 Hadoop

由於Hadoop受到許多企業及研究機構關注，因此，在近年來在應用與改進方面皆有許多研究工作。在應用方面，Hadoop主要應用如下：基序與循序樣式探勘[1]、雲端儲存系統[2]、軌跡序列探勘系統[3]、雲端運算系統[4]等方面。

Hadoop相關文獻如下：

(1) 陳仲毅其由於 Hadoop 預設的任務排程有高可能性挑選到一項不再此工作節點上的任務，造成工作節點需要向其他工作節點取得資料導致拖長工作時間的缺點，而提出以針對任務地域性或全域性考量並且平衡工作量的排程演算法[5]。

(2) 陳奕銘利用 OpenStack 透過將 Hadoop 主機叢集虛擬化並增加節點的方式提升 Hadoop 系統的擴充性與安全性[6]。

(3) 林芳毅使用 Hadoop 的平行分散式處理技術與 R 語言的統計軟體演算進行 TH 企業產品銷售預測，同時利用 RHadoop 的倒傳遞類神經網路、C-SVC 支援向量機與指數平滑法三種模型進行預測研究，其研究結果顯示三者之中以 C-SVC 支援向量機所建立的模型最為準確，可作為研究人員及企業決策人員的參考依據及決策指標[7]。

(4) 王辰豪基於 Hadoop 架構利用 HDFS(Hadoop Distributed File System)分散式檔案系統、MapReduce 分散式處理軟體框架以及 Pig 程式語言設計並實作一套網站日誌分析系統，其結果藉由分析網站日誌而獲得

Application Server 流量趨勢及程式效能統計報表和依需求提供不同區間和程式功能的效能統計報告，可協助管理人員快速地分析資料中的潛在價值，並協助企業進行決策[8]。

2.2 MapReduce

MapReduce是Google提出的一個軟體架構，用於大規模數據集（大於1TB）的並行運算。概念「Map（映射）」和「Reduce（歸納）」，及他們的主要思想，都是從函數式程式語言借來的，還有從矢量程式語言借來的特性[9]。通過把對數據集的大規模操作分發給網絡上的每個節點實現可靠性；每個節點會周期性的把完成的工作和狀態的更新報告回來。如果一個節點保持沉默超過一個預設的時間間隔，主節點（類同Google檔案系統中的主伺服器）記錄下這個節點狀態為死亡，並把分配給這個節點的數據發到別的節點。MapReduce相關文獻如下：

(1) 王崇力嘗試在 Reduce 端使用不同的 LZO 壓縮編碼，使其輸出支援分割成 block，讓 MapReduce 可以直接讀取，減少 CPU 的計算負載，其實驗結果證明，MapReduce 執行的總耗時將比沒有做修改的方法減少 10%~20%的資料傳輸時間。[10]

(2) 廖唯志藉由結合 GPU 運算減少 CPU 運算工作的負擔，並加快 Map 工作的運算速度，再經由 MPTCP 減少 MapReduce 運算過程中所產生資料的傳輸時間。經由常見的 Terasort、WordCount 及 PiEstimate 等 Hadoop 效能比較程式來進行實驗與測試以驗證所提方法的效能。所獲致的實驗與測試結果得知：GPU 運算以及 MPTCP 協定的加入確實能夠有效提升 Hadoop MapReduce 的運算效能，以及有效提高網路傳輸的頻寬，並減少了分散式運算工作所需的時間。[11]

(3) 彭聖儒提出了負載平衡及優化技術，稱為 Load Balancing+與 Load Balancing++。在 Map 階段結束後，根據各個節點的運算能力，並且考慮 Communication 來決定各個節點容量(Capacity)，達到平均完成時間相近。同時，利用 Load Balancing++優化技術，在分配資料過程中，更新各個節點的預測完成時間，根據各個節點的預測完成時間，動態的分配資料，進而達到 Communication 的優化與負載平衡。此外，經由性能分析結果顯示，經過優化後，可以進一步使 Load Balancing+的完成時間縮短[12]。

第三章　研究設計

本研究主要使用2015 Etu Hadoop競賽的模擬網站日誌，將資料集上傳至以Oracle VM VirtualBox建立的Hadoop之HDFS系統後以MapReduce進行資料擷取與加總和排行，以下將介紹研究工具Oracle VM VirtualBox、Hadoop及MapReduce。

3.1　研究工具 - Oracle VM VirtualBox

Oracle VM VirtualBox是一款由Oracle公司進行開發的虛擬機器軟體，最初的VirtualBox是在於2007年由InnoTek軟體公司出品，而在於2008年時InnoTek軟體公司由Sun Microsystems公司所併購而Sun Microsystems公司則在2010年時由Oracle所收購，因此現在VirtualBox是Oracle公司進行開發。

VirtualBox是一款功能強大虛擬機器軟體它實現了在於一台實體電腦上安裝多個作業系統的功能，且每個系統皆可獨立開啟、暫停與停止，而實體主機的作業系統與虛擬端作業系統之間皆能夠相互通訊。相較於其他同類型的虛擬機器軟體如VMware，VirtualBox所創建的映像檔相對較小，其圖形化管理介面也較方便直觀適合初次接觸虛擬機器的新手，且VirtualBox本身即有中文介面可以選擇。

表 3.1 VirtualBox 的特點

多系統運行	支援多作業系統同時運行
相容性高	支援 VMware VMDK 及 Virtual PC VH 等格式。
跨平台	支援 Windows 及其他的一些主流平台，如 Linux，Unix 等等。
多 CPU 模擬	最多虛擬 32 顆 CPU

3.2 實驗樣本

實驗樣本利用2015 Etu Hadoop競賽[13]的模擬網站日誌，檔案大小為1.6GB共計130萬筆資料其格式如圖 3.1所示，其中以action將資料分為view(瀏覽)、cart(購物車)及本研究所需的order(訂購)；本研究對原始資料進行處理將所有屬於order的資料個別取出後以Map程式取出其中所訂購的商品資料並計算其銷售額，再以Reduce程式對所有Split做銷售總額的加總，最後使用另外的MapReduce程式進行前二十名的排序。

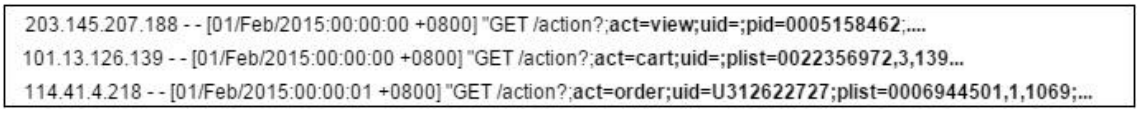

圖 3.1 樣本格式

3.3 Hadoop

Hadoop是一款支援資料密集型分布式應用並以Apache 2.0許可協議發布的開源軟體框架。它支援在商品硬體構建的大型集群上運行的應用程式。Hadoop是根據Google公司發表的MapReduce和Google檔案系統的論文自行實作而成。

Hadoop框架透明地為應用提供可靠性和資料移動。它實現了名為MapReduce的編程範式：應用程式被分割成許多小部分，而每個部分都能在集群中的任意節點上執行或重新執行。此外，Hadoop還提供了分布式檔案系統，用以存儲所有計算節點的資料，這為整個集群帶來了非常高的帶寬。MapReduce和分布式檔案系統的設計，使得整個框架能夠自動處理節點故障。它使應用程式與成千上萬的獨立計算的電腦和PB級的資料。現在普遍認為整個Apache Hadoop「平台」包括Hadoop內核、MapReduce、Hadoop分布式檔案系統（HDFS）以及一些相關項目，有Apache Hive和Apache HBase等等。

表 3.2 Hadoop 的特性

擴容能力	能可靠的（reliably）存儲和處理千兆位元組（PB）數據。
低成本	可以通過普通機器組成的服務器群來分發以及處理數據。這些服務器群總計可達數千個節點。
高效率	通過分發數據，hadoop 可以在數據所在的節點上平行地（parallel）處理它們，這使得處理非常的快速。
可靠性	能自動地維護數據的多份複製，並且在任務失敗後能自動地重新部署（redeploy）計算任務。

3.4 MapReduce軟體架構

MapReduce是一種簡單的程式模型，只要程式設計時使用Map和Reduce函數，系統會幫忙用Map從原始資料上整理分類出中介資料，然後用Reduce程式簡化這些中介資料，MapReduce的目的是對叢集電腦上的大型資料集（dataset）執行分散式運算，替大量資料做平行運算處理。

3.4.1 MapReduce軟體架構

MapReduce 2.0(YARN)的架構如圖 3.2所示，包括：

(1) Client：負責提交 Application 到 RM，它會首先創建一個 Application 上下文件對象，並設置 AM 必需的資源請求信息，然後提交到 RM。 YARN Client 也可以與 RM 通信，獲取到一個已經提交並運行的 Application 的狀態信息等，具體詳見後面 ApplicationClientProtocol 協議的分析說明

(2) ResourceManager（RM）： YARN 集群的 Master，負責管理整個集群的資源和資源分配。 RM 作為集群資源的管理和調度的角色，如果存在單點故障，則整個集群的資源都無法使用。

(3) NodeManager（NM）： YARN 集群的 Slave，是集群中實際擁有實際資源的工作節點。

(4) Container：YARN 集群中资源的抽象，将 NM 上的资源进行量化。

(5) ApplicationMaster（AM）：AM 主要管理和监控部署在 YARN 集群上的 Application。

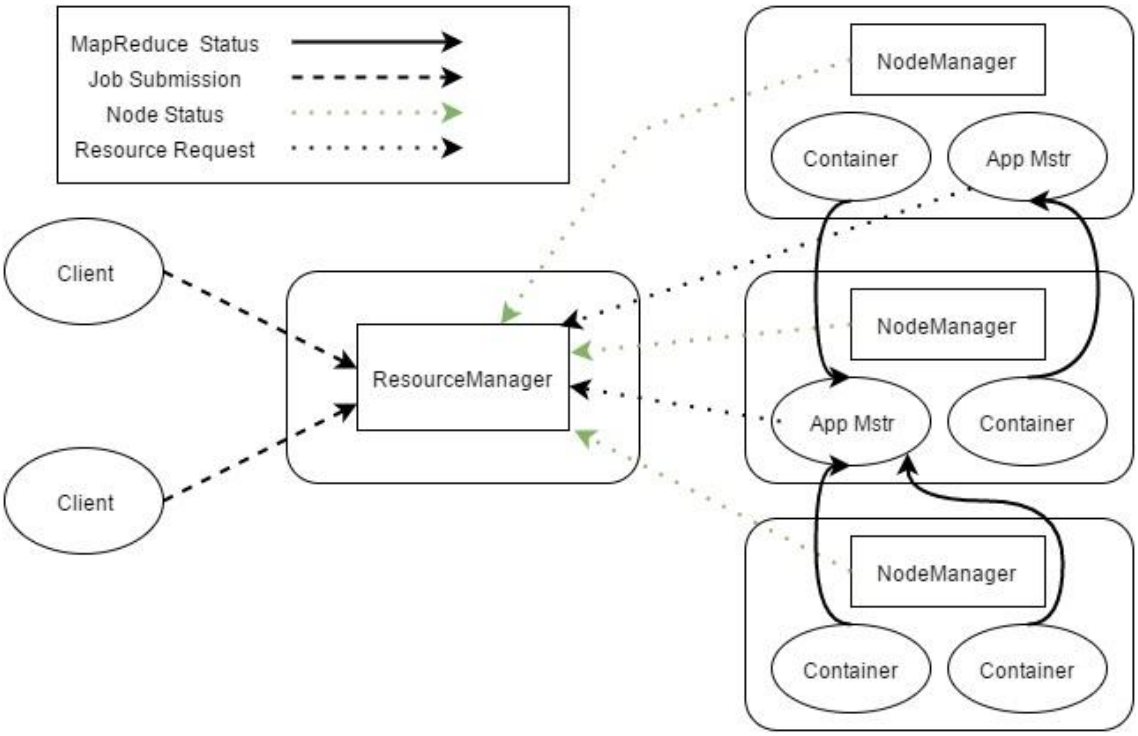

圖 3.2 MapReduce 2.0架構

3.4.2 MapReduce的執行過程

程式執行時首先對輸入的資料進行分割（split）成固定大小的區塊，拆分後每個輸入區塊可以透過Map函數在不同的節點上並行處理，帶節點完成後，再由Reduce函數進行彙總，將執行結果輸出，MapReduce詳細執行過程如圖 3.3所示。

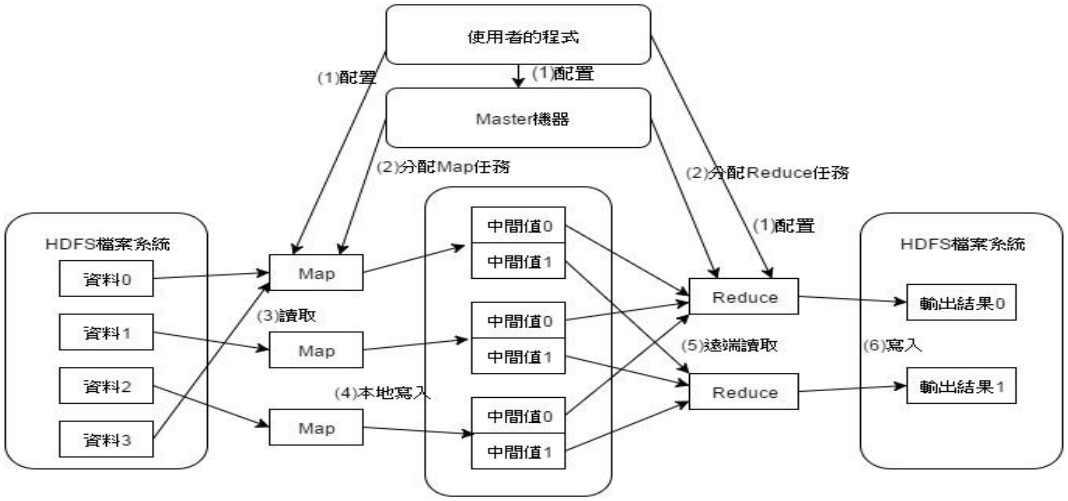

圖 3.3 MapReduce執行過程

如圖 3.4所示輸入檔案分割成16~64MB之間的區塊，大小一般通過參數設定，共分為M塊。然後在不同的節點上執行處理程式的副本。

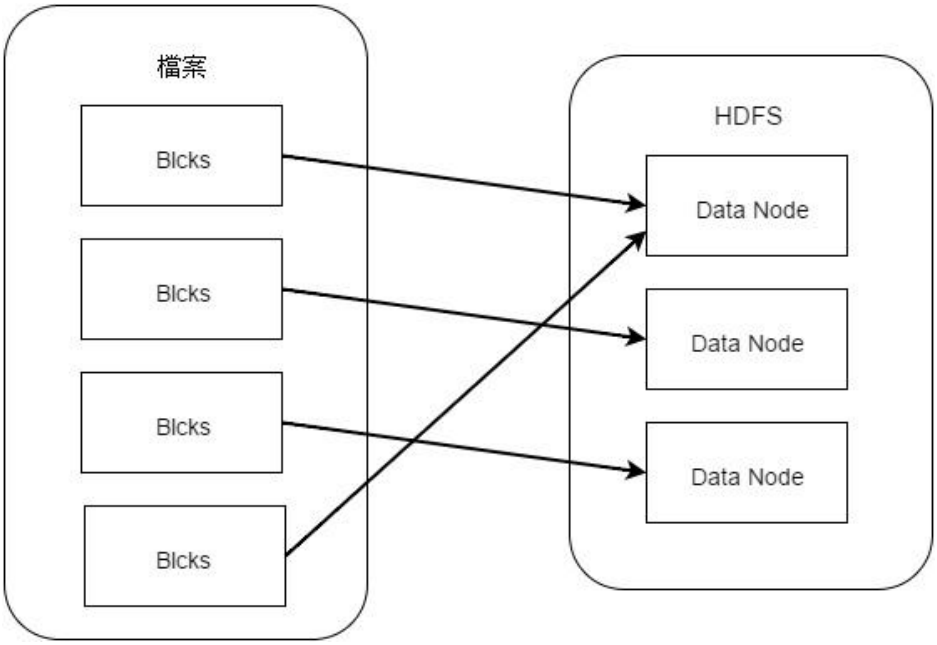

圖 3.4　MapReduce的執行過程一

如圖 3.5所示輸入執行程式中存在一個叫做Master的特殊程式，即

主控程式。其他的都是作為主控程式分類任務的工作程式(Worker)。主控程式需要將M個Map作業和R個Reduce作業分配到空閒的程序上執行。

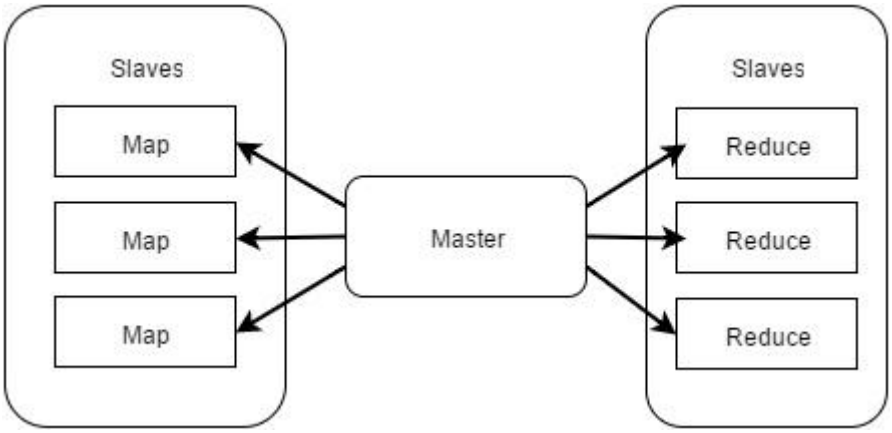

圖 3.5　MapReduce的執行過程二

如圖 3.6所示輸入被分配Map作業的任務已檔案為一個區塊為輸入，並處理輸入資料，分析產生Key/Value對傳遞給Map函數，這些東西會被暫時緩存在記憶體中。

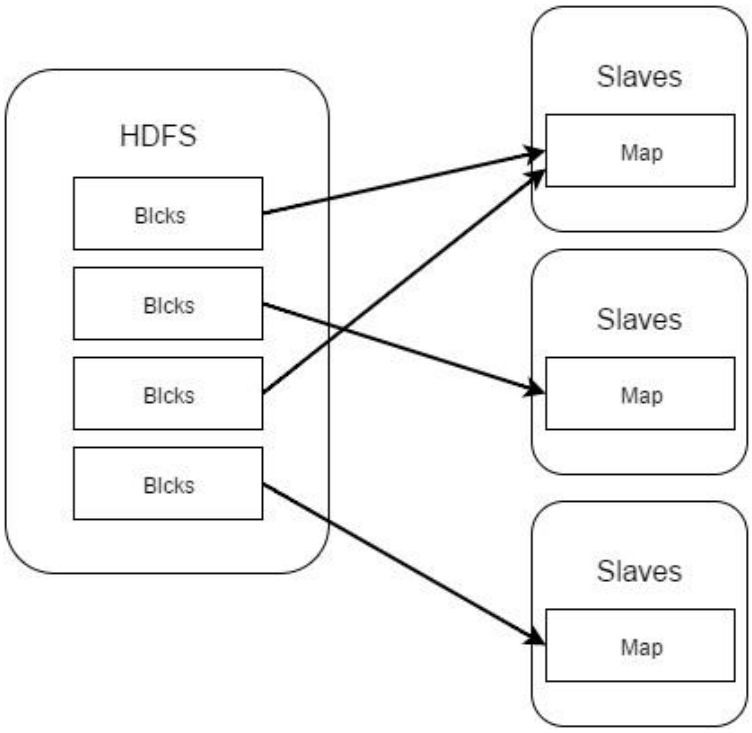

圖 3.6　MapReduce的執行過程三

如圖 3.7所示輸入暫時保存的中間結果集會按照一定時間間隔寫入

工作節點之硬碟，用分區函數把這些中間結果分成R個區。寫入硬碟的資料位置會傳遞給主控制程式，程式將傳遞的資料發給負責Reduce任務的工作節點。

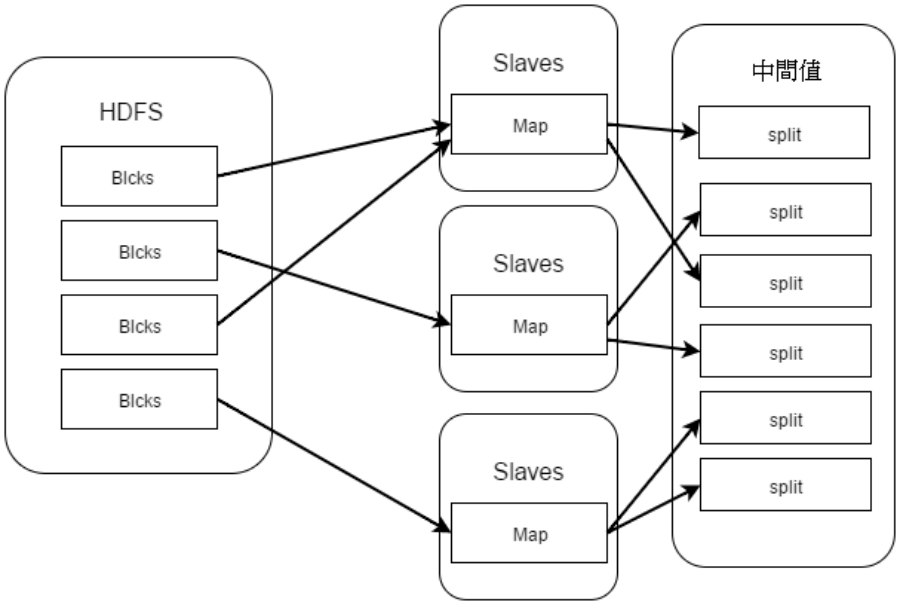

圖 3.7　MapReduce的執行過程四

如圖 3.8所示輸入主控制程式收到資料後，通過遠端調用從Map節點硬碟上讀取中間結果集合。Reduce在獲得全部中間結果集後，按照中間結果集排序。排序是必要的，輸入資料值不同經過Map操作後所得的值可能相同，排序保證中間結果按Key值相同。

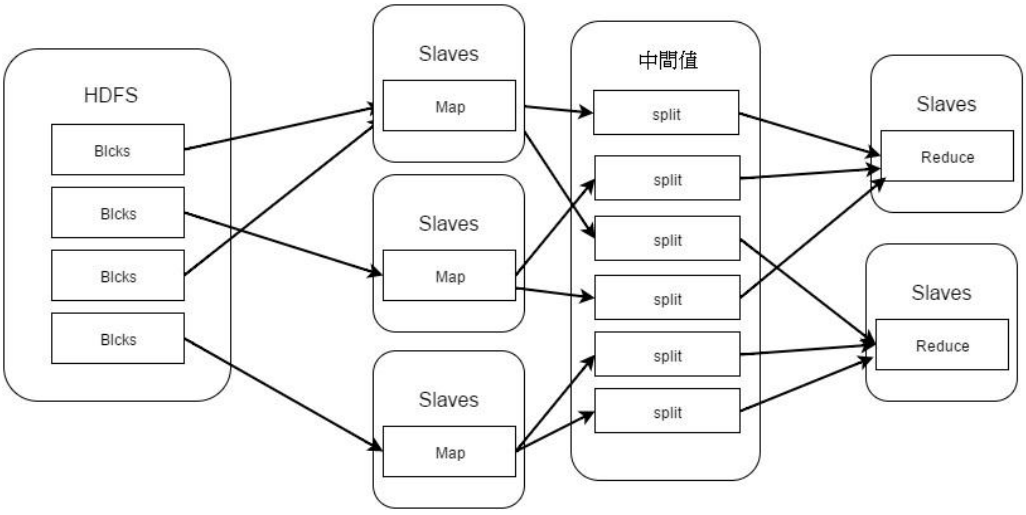

圖 3.8　MapReduce的執行過程五

如圖 3.9所示輸入Reduce處理有序的中間鍵/值對，對中間Key值相同的結果做合併處理，並將資料傳給使用者提供的Reduce函數。Reduce函數將該區塊追加到最終輸出檔中。

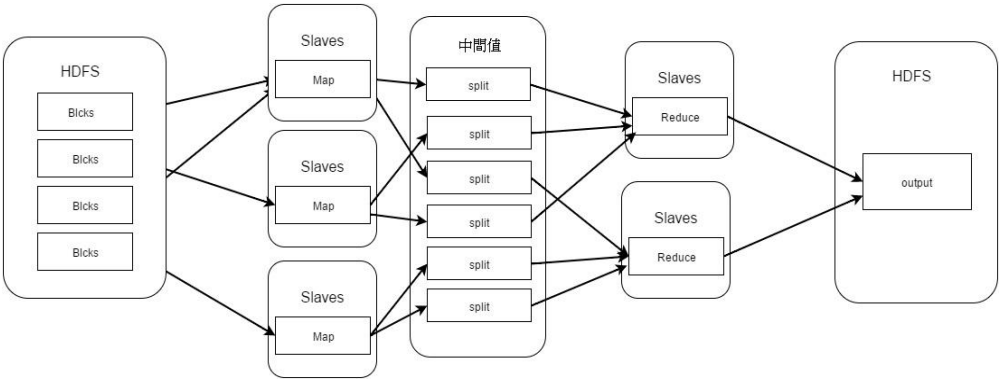

圖 3.9　MapReduce的執行過程六

第四章 結果與討論

目前進度已將三節點的Hadoop環境利用Oracle VM VirtualBox建立完成並且輸出結果正確的MapReduce程式也已撰寫完成，以下介紹建立之環境及本程式之運行步驟，原始程式碼則以附錄方式呈現。

4.1 VirtualBox下分散式的Hadoop環境

如圖4.1及圖4.2與我們使用VirtualBox來建構Hadoop環境，首先我們以單節點的方式建構我們的Hadoop環境，以便我們測試程式，而在基礎程式完成後我們新增了兩個節點以分散的方式來建構Hadoop環境如圖4.3所示，以便測試完成的程式在分散式環境下所能夠達成的最短運算時間。

圖 4.1 VirtualBox管理介面

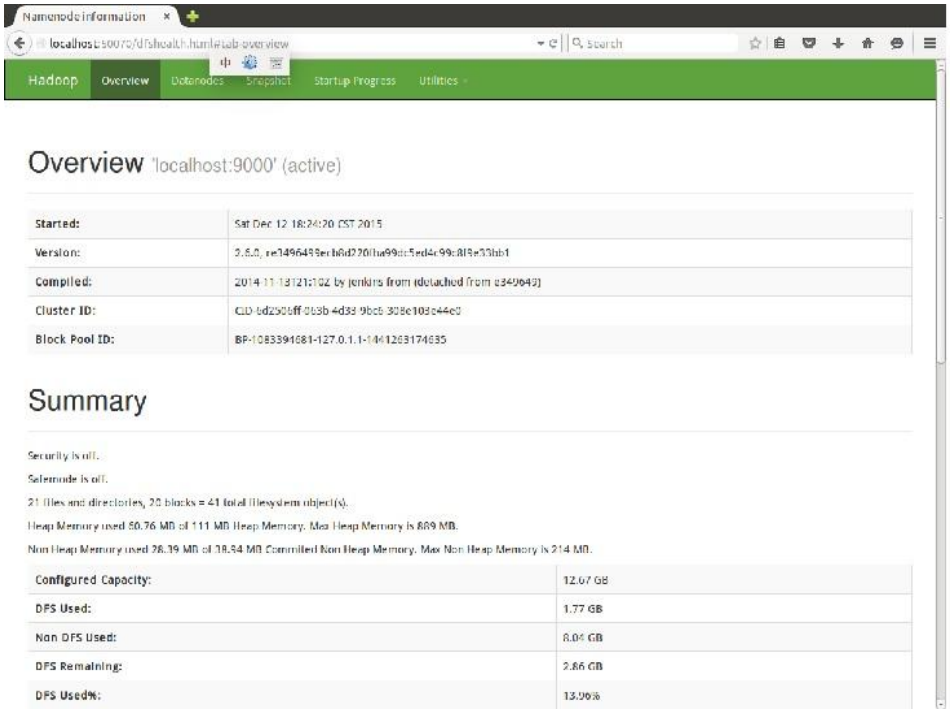

圖 4.2 Hadoop環境

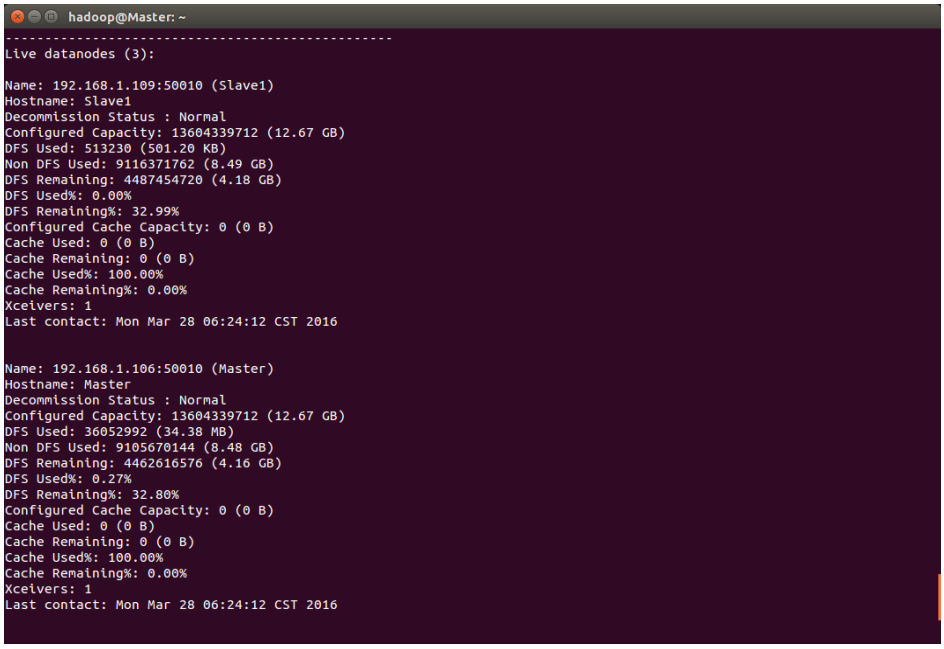

圖 4.3 Datanodes列表

4.2 Main程式

執行過程如圖 4.4所示，對Hadoop輸出指令後Main程式將呼叫Map程式對指令中指定的路徑進行資料讀取，而程式本身已寫入預設

之路徑，因此即使指令無輸入指定路徑也可以對預設之路徑檔案進讀取。

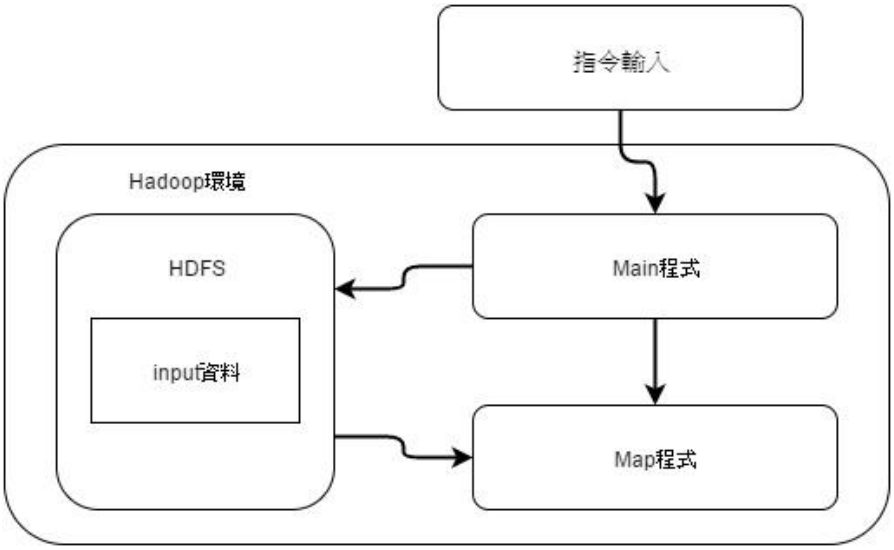

圖 4.4　Main程式的執行過程

4.3 Map階段

如圖 4.5所示輸入Main程式呼叫Map程式後，Map程式會對資料進行讀取及切割，而本研究主要目標只有資料行中有action=order的資料，因此在本程式階段會對讀入的資料行進行判斷，若action≠order則讀入下一行資料，如action=order則將資料存入k/v對並且將其金額加總後給Reduce程式。

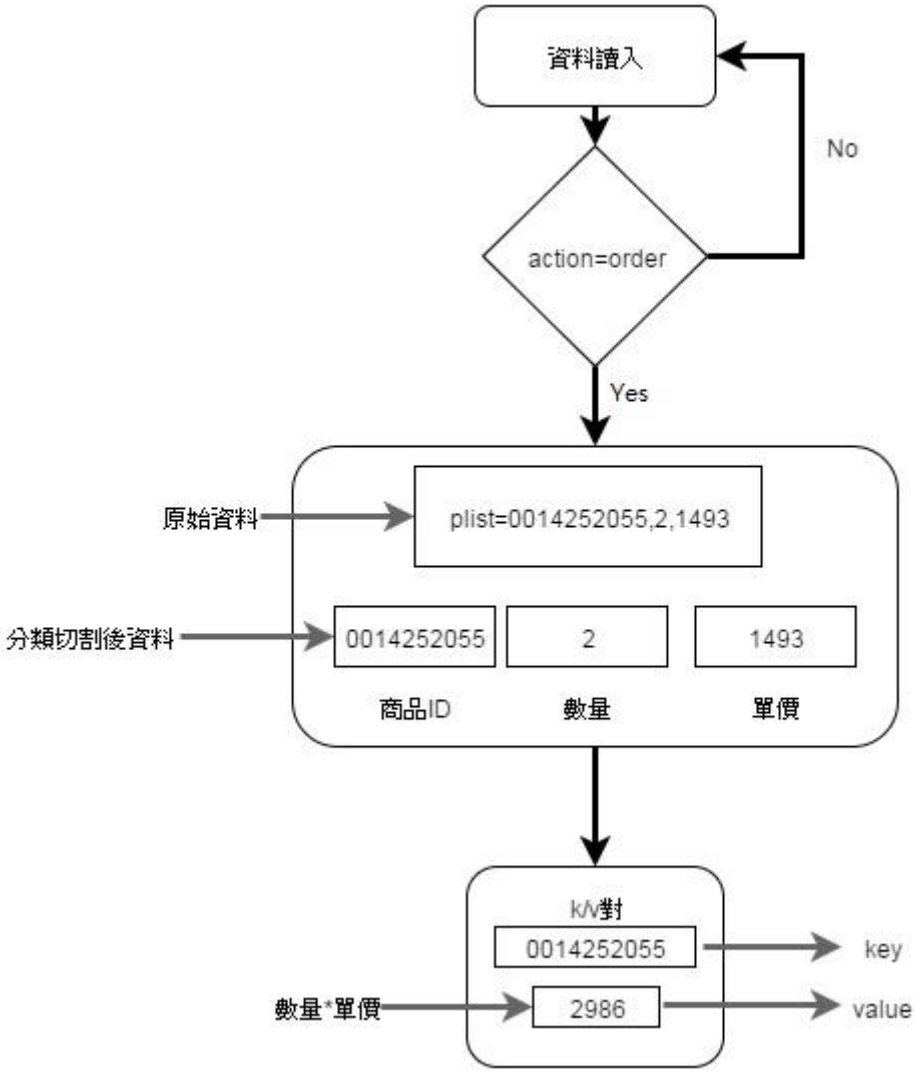

圖 4.5　Map程式的執行過程

4.4 Reduce階段

如圖 4.6所示輸入Reduce程式收到Map程式送達的k/v對後會對同樣key資料的value部分進行加總以計算出各商品的總銷售額並以key的資料進行排序。

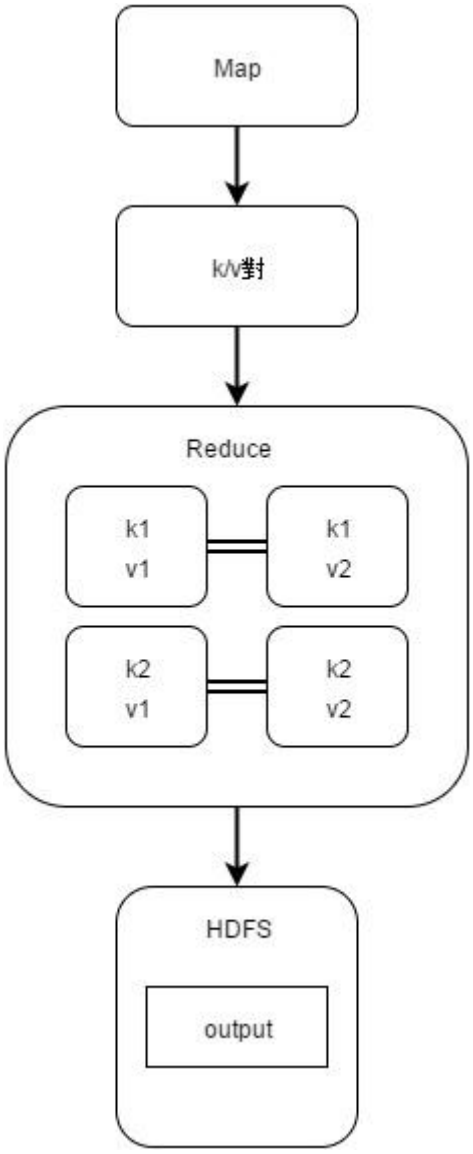

圖 4.6 Reduce程式的執行過程

4.5 第二次Map階段

由於MapReduce在Reduce階段輸出結果之後就不允許對結果做任何的修改，因此我們使用了二次排序的方式來達成我們想要對商品銷售總額進行排序，而在第二次Map階段則如圖4.7所示，以上一個Reduce階段所輸出的結果以value也就是商品銷售總額的部份作為本階段的key利用MapReduce本身以key作為排序依據的特性來進行銷售額的排序。

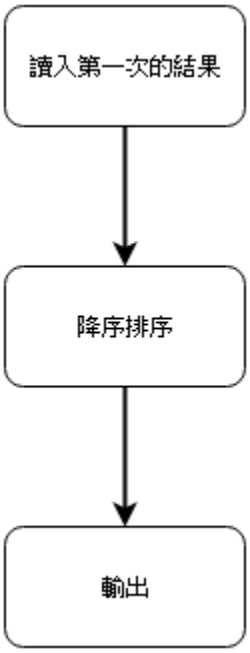

圖 4.7 第二次Map的執行過程

4.6 第二次Reduce階段

如圖4.8所示，第二次Reduce階段則會將第二次Map所輸出的結果將前二十名的資料以名次及商品ID以降序的方式進行最終結果的輸出。

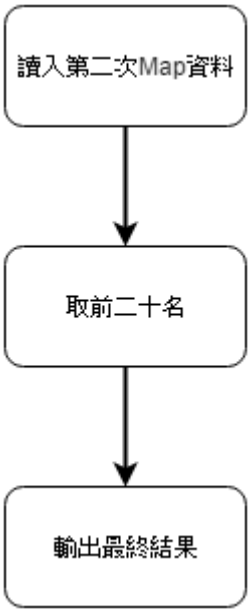

圖 4.8 第二次Reduce的執行過程

4.7 輸出結果

最終完成的程式在以使用HashCode演算法的情況下執行時間如圖 4.9中的實際時間 (real time)所示約在9至10秒之間，而最後取其前二十名的輸出結果則如圖4.10所示輸出格式以Tab分隔，第一個欄位為資料名次而後則是商品ID。

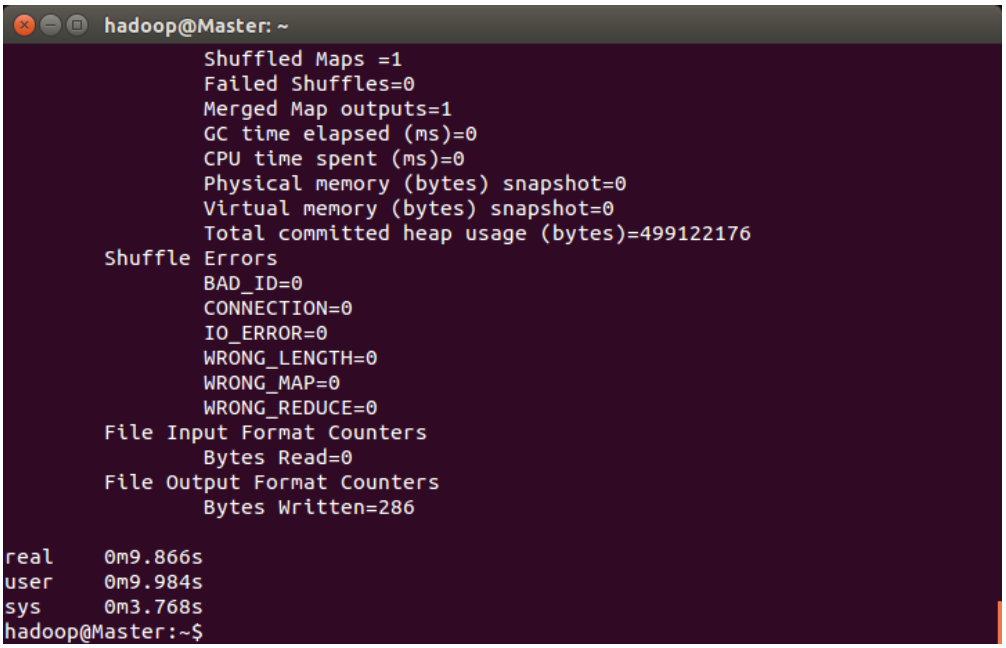

圖 4.9 執行時間

01	0006584093
02	0000143511
03	0007082051
04	0005772981
05	0014252066
06	0006323656
07	0004607050
08	0024239865
09	0003425855
10	0004134266
11	0006993652
12	0004862454
13	0009727250002
14	0006270095
15	0014252055
16	0006993663
17	0009727290016
18	0018504861
19	0000143500
20	0024634260

圖 4.10 取前二十名之輸出結果

在最後完成的程式，我們使用了HashCode演算法來加速程式運算的時間HashCode演算法是在MapReduce中Shuffle階段進行，Shuffle是依據Map輸出的intermediate key/value產生分區。系統將map處理後的intermediate key/value轉移和排序後，輸出至Reducer的過程。

以下為公式用於計算HashCode的方法:

$$\text{HashCode} = W_{\text{n}} \times 31^{n-1} + W_{n-1} \times 31^{n-2} + \cdots + W_1 \times 31^0$$
$$= \sum_{n=1}^{TotalWord} W_n \times 31^{n-1}$$

而HashCode演算法雖可均勻分配運算量，但其輸出結果並未排序完成，若要排序仍將各Reducer輸出的結果，合併再進行排序，因此我們仍然需要使用第二次的MapReduce來進行排序，才能夠達成我們所要求的結果。

而在使用了HashCode演算法之後，我們程式運行的時間相較於未使用的程式大約縮短了3至4秒左右。

我們也嘗試了在較少節點數的情況下，程式運算所需要的時間，在嘗試的過程中我們發現節點數對於運算時間產生的影響非常巨大，雙節點的情況下以同樣的程式進行運算就能夠縮短將近5秒左右的時間。

表4.1使用不同節點及演算法下的運算時間

節點數	未使用HashCode演算法	使用HashCode演算法
單節點	24秒-27秒	20秒-23秒
雙節點	20秒-23秒	14秒-18秒
三節點	15秒-18秒	9秒-12秒

第五章 結論與未來展望

5.1 結論

本研究所建構之MapReduce程式，在以三個節點所建構而成的Hadoop環境下已達成本研究最終十秒內的成果。

5.2 未來展望

本研究之成果未來可以嘗試求取在更多節點的Hadoop環境下程式與節點之間的關係式並加以分析，或可推測出Hadoop的節點與資料處理的最佳化狀態，另可加入對於使用者的瀏覽紀錄等，並使用回歸預測等演算法加以分析並建立模型，或可讓企業使用成果來對銷售進行預測，成為企業的決策參考之一。

參考文獻

[1] 羅振維，"基於 Hadoop 的基序與循序樣式探勘研究─以台灣家用電力時間序列資料為例"，臺灣大學資訊管理學研究所碩士論文，民國一百零三年七月。

[2] 謝岳霖，" 基於 Hadoop 與 HBase 平台之 LCD Flicker 雲端儲存系統之實作 "，東海大學資訊工程學系碩士論文，民國一百零四年一月。

[3] 吳柏翰，" SeMiHBase: 運用雲端 Hadoop 與 HBase 於軌跡序列探勘系統之研發 "，中興大學電機工程學系所碩士論文，民國一百零二年七月。

[4] 鄭峰麒，"Hadoop 雲端運算效能評估與行動管理系統"，虎尾科技大學資訊工程研究所碩士論文，民國一百零二年七月。

[5] 陳奕銘，"基於 OpenStack 有效提升 Hadoop 擴充性與安全性的研究與發展"，義守大學資訊工程學系碩士論文，民國一百零四年七月。

[6] 陳仲毅，"使用多重任務佇列以改善 Hadoop 中之資料地域性"，國立成功大學資訊工程學系碩士論文，民國一百零二年七月。

[7] 林芳毅，" Hadoop 結合 R 之資料探勘 – 以 TH 公司產品銷售預測為例"，國立高雄應用科技大學工業工程與管理系碩士論文，民國一百零四年七月。

[8] 王辰豪，"基於 Hadoop 架構之網站日誌分析系統"，國立交通大學資訊學院資訊學程碩士論文，民國一百零二年七月。

[9] 沈炳宏，"初探 Hadoop 開放原始碼平台環境"，RUN! PC 雜誌：民國九十九年十一月。

[10] 王崇力，"Hadoop MapReduce 環境中使用改良的 LZO 演算法配置以提升資料傳輸效能"，大同大學資訊工程學系(所)碩士論文，民國一百零三年七月。

[11] 廖唯志，"結合 GPU 與多重鏈路協定提昇 Hadoop/MapReduce 效能之研究"，銘傳大學資訊工程學系碩士論文，民國一百零三年七月。

[12] 彭聖儒，"改良 MapReduce 運算模型以加速資料密集型應用"，中華大學資訊工程學系碩士論文，民國一百零三年七月。

[13] 2015 Etu Hadoop 競賽，http://ehc.etusolution.com/index.php/tw/。

附錄：程式原始碼

附錄 A:Main

```
public static void main(String[] args) throws Exception{

  Configuration conf = new Configuration();                                    //
  String[] otherArgs = new GenericOptionsParser(conf, args).getRemainingArgs(); //設定可手動指定輸入及輸出路徑
  String in = "input";                                                         //預設之輸入路徑為input
  String out = "out/" + new SimpleDateFormat("mmss").format(new Date());       //預設之輸出路徑為out資料夾下以時間為檔案名稱
  Path dst_path = new Path ("out");                                            //
  FileSystem hdfs = dst_path.getFileSystem(conf);                              //

      If (otherArgs.length >= 1) {                                             //如果使用者有指定輸入路徑
          in = otherArgs[0];                                                   //則將輸入目標改為使用者所指定的路徑
      }
      If (otherArgs.length >= 2) {                                             //如果使用者有指定輸入路徑
          out = otherArgs[1];                                                  //則將輸入目標改為使用者所指定的路徑
          dst_path = new Path (otherArgs[1]);                                  //輸出檢查路徑也該為使用者指定之路徑
      }
      If (hdfs.exists(dst_path)){                                              //檢查檔案系統中是否已有與輸出目標相同的資料夾
        hdfs.delete(dst_path, true);                                           //若有則刪除
      }                                                                        //

String sort = out + "-tmp";                                                    //
Job job = Job.getInstance(conf, "word count");                                 //
job.setJarByClass(WordCount.class);                                            //
job.setMapperClass(WordCountMapper.class);                                     //
job.setCombinerClass(WordCountReducer.class);                                  //
job.setReducerClass(WordCountReducer.class);                                   //
job.setOutputKeyClass(Text.class);                                             //
job.setOutputValueClass(LongWritable.class);                                   //
FileInputFormat.addInputPath(job, new Path(in));                               //
FileOutputFormat.setOutputPath(job, new Path(sort));                           //
job.waitForCompletion(true);                                                   //此為Map及Reducer的設定段落
  If (job.isSuccessful()) {                                                    //
    conf.set("", ",");                                                         //
    Job job2 = Job.getInstance(conf, "sort job");                              //
    job2.setNumReduceTasks(1);                                                 //
    job2.setJarByClass(WordCount.class);                                       //
    job2.setMapperClass(Sortmapper.class);                                     //
    job2.setReducerClass(Sortreducer.class);                                   //
    job2.setSortComparatorClass(KeyDescComparator.class);                      //
    job2.setOutputKeyClass(LongWritable.class);                                //
    job2.setOutputValueClass(Text.class);                                      //
    MultipleInputs.addInputPath(job2, new Path(sort), TextInputFormat.class);  //
    FileOutputFormat.setOutputPath(job2, new Path(out));                       //
```

附錄 B:Map

```
public class WordCountMapper extends Mapper<LongWritable, Text, Text, LongWritable>{

  @Override
  protected void map(LongWritable key,Text value,Context context)
      throws IOException, InterruptedException {
        String line = value.toString();
          if (!line.contains(";act=order;") || line.contains(";plist=;")) {      //判斷讀入資料行是否含有act=order及plist=
             return;                                                             //若不含act=order及plist=則return
          }                                                                      //

          int p = line.indexOf("plist=");                                        //計算從讀入資料的第一個字到plist=的長度
            if (p != -1) {                                                       //
              p += "plist=".length();                                            //
            }                                                                    //
          int e= line.indexOf(";", p);                                           //從plist=開始到第一個分號的長度

        String oData = line.substring(p, e);                                   //取出從plist=開始至第一個分號的資料存至oData
        String[] data = oData.split(",");                                      //將oData 中的資料以分號切割並存至data陣列

          if (data.length % 3 != 0) {                                          //如data陣列長度不是3的倍數即顯示錯誤訊息
              System.err.println("ERR " + line);
          }
          for (int x = 1; x < data.length; x += 3) {

            context.write(new Text(data[x - 1]), new LongWritable(Integer.parseInt(data[x]) * Integer.parseInt(data[x + 1])));
    //將資料寫入output的kv對          商品ID                                        數量          *        單價
          }
      }
  }
```

附錄 C:Reduce

```
public class WordCountReducer  extends Reducer<Text, LongWritable, Text, LongWritable> {

  @Override
  protected void reduce(Text key,  Iterable<LongWritable> values,Context context)
          throws IOException, InterruptedException
  {
          int count = 0;
          for(LongWritable value : values) {      //
            count += value.get();                 //將相同key之value加總
          }                                       //


          context.write(key, new LongWritable(count));
          //output kv對 value值為相同key之加總

  }
}
```

附錄 D:第二階段 MAP

```
public class Sortmapper extends Mapper<LongWritable, Text, LongWritable, Text>{
  protected void map( LongWritable key,Text value,Context context)
      throws IOException, InterruptedException {

    String line = value.toString();                    //資料存入line
    String[] words = line.split("\t");                 //以TAB分隔
    long count = Long.valueOf(words[1]);               //count為讀入時tmp的value部份也就是商品銷售額
    String pid = words[0];                             //pid為tmp的key部份也就是商品id

    context.write(new LongWritable(count), new Text(pid));   //以count為key pid為value輸出
  }
}
```

附錄 E:第二階段 Reduce

```
package sort;


import org.apache.hadoop.io.LongWritable;


  public class Sortreducer  extends Reducer<LongWritable, Text, Text, Text> {
    static final int n = 20;
    static int c = 0;

    @Override
    protected void reduce(LongWritable count, Iterable<Text> values,Context context)
        throws IOException, InterruptedException {
      for (Text value : values) {
        c++;
        if (c <= n) {                                  //
          context.write(new Text(String.format("%02d", c)), value);   //
        } else {                                       //取前二十名
          break;                                       //
        }                                              //
      }
    }
  }
  //mapreduce本身的排序是以key為主作遞增排序 這第二次的mapreduce是將第一次mapreduce所輸出的結果以商品銷售額作為key來進行排序
```