國立臺灣海洋大學

資訊工程學系

碩士學位論文

指導教授：張雅惠博士

以 HBase 支援生產流程查詢

Supporting the Querying on Manufacturing Processes Using HBase

研究生：陳怡淵　撰

中華民國 105 年 7 月

以 HBase 支援生產流程查詢

研究生 ： 陳怡淵　　Student：Yi-Yuan Tan

指導教授：張雅惠　　Advisor：Ya-Hui Chang

國立臺灣海洋大學

資訊工程學系

碩士論文

A Thesis

Submitted to Department of Computer Science and Engineering

College of Electrical Engineering and Computer Scienece

National Taiwan Ocean University

In Partial Fulfillment of the Requirements

for the Degree of

Master of Science

in

Computer Science and Engineering

July, 2016

Keelung, Taiwan, Republic of China

中華民國 105 年

摘要

製造執行系統主要用來收集製造現場物品的生產資訊，讓使用者能即時得到相關資訊進行分析，以提高生產效率。但是隨著生產資訊與日俱增，達到巨量資料，本論文討論如何以具有 NoSQL 資料庫特性的 HBase 搭配 Java 程式，實作出具有生產查詢功能的系統。本論文提出了兩種定義資料庫 schema 的方式。第一種方法是將物品型號和序號合併為 Row Key，共同儲存在同一表格，此方法稱為『mixed 方法』。我們提出的第二種方法，是將個別型號獨立建成新的表格，並以物品序號為 Row Key，此方法稱為『assorted 方法』。我們實作此二方法，透過一系列的實驗以及不同數量的資料集，比較其查詢效率。實驗結果顯示，assorted 方法在 point query 中的查詢效率遠比 mixed 方法好，而在 scan query 中，兩種方法並沒有很大差別，且隨著資料量呈現線性成長。綜合實驗發現，HBase 對於單一資料以列為查詢條件的處理非常快速且穩定，對於掃瞄整個表格的查詢句則會隨著資料量呈現線性成長。 整體而言，assorted 方法的執行效率比 mixed 方法來得好。

Abstract

The Manufacturing Execution Systems (MES) are used in factories to provide the information about objects during the production process in real time. Traditionally, the MES is constructed using relational databases, but the efficiency might decay when dealing with big data. In this thesis, we study how to use HBase and Java to build a system for supporting the querying on manufacturing process. We propose two methods of defining schemas in HBase. In the “mixed” method, the information of all objects are stored in the same table where the product number and the serial number of an object are combined to form the Row Key. We then propose another method, called the “assorted” method. It will create a new table for each product number and the serial number of each object is the Row Key. We have performed a series of experiments based on a variety of datasets to compare the efficiency of the two proposed methods. Experimental results show that HBase performs very well if it only needs to retrieve a single row, but the execution time will increase linearly if it needs to examine a set of data. Overall, the assorted method is better than the mixed method.

誌謝

首先，感謝指導教授張雅惠博士，對於本論文給予許多幫助，且在研究論文期間不時地共同討論，解決論文的諸多疑點，使學生順利的完成論文，同時也讓學生獲益匪淺，在此向敬愛的老師致上衷心謝忱。

除此之外，感謝口試委員林川傑博士和劉傳銘博士，細心審稿以及修正論文，並對本論文提供寶貴的建議，使本論文更趨近於嚴謹完善，在此深表謝意。

最後，特別感謝我太太很辛苦的為我生下了可愛的寶寶，在家中一直替我加油，望我能早日畢業，你們一直都是我的精神支柱。我知道你一個人帶著小孩，壓力真的很大，常常都在想我能在你身邊。媽媽是最偉大的，這句話是真的，你真的很偉大，你真的很棒，謝謝你為我付出的一切。感謝親愛的爸爸、媽媽、哥哥、姐姐，謝謝你們的包容和鼓勵，讓我在生活中有堅強的依靠。感謝實驗室的同學文瀚、詩盈與學弟妹陪我度過多采多姿的研究所生活。謝謝大學的室友博仁、印印對我的幫助。謝謝竣圍、小貓、貓貓、潔安、...還有很多很多的朋友，謝謝你們在台灣對我的照顧。這兩年內，我發生的事情真的太多、太多了，很多時候我已經快沒辦法一個人面對了，非常感謝曾經協助我的人。再一次謝謝張雅惠老師，這兩年對我的包容和教導，給我很多的空間去處理自己的私事，也提供了很多的幫助給我，真的謝謝您，老師。

目錄

摘要 I

Abstract II

誌謝 III

圖目錄 VI

表目錄 VII

第 1 章　序論 1

1.1　研究動機與目的 1

1.2　研究方法與貢獻 2

1.3　相關研究 3

1.4　論文架構 5

第 2 章　系統功能介紹 6

2.1　HBase 資料庫特性 6

2.2　系統架構與功能 9

第 3 章　以型號與序號合併之方法 12

3.1　資料定義 12

3.2　功能實作 13

第 4 章　以型號分類之方法 24

4.1　資料定義 24

4.2　功能實作 25

第 5 章　實驗 29

5.1　資料集和查詢句設計 29

5.2　不同查詢句的效率評估 31

5.3　Scan Query 和 Column Family 的分析 34

5.4　Point Query 的分析 37

5.5 將型號建立成獨立一個 Column Family 之實驗 38
5.6 改變 Hadoop 中 Block Size 對執行效率的影響之分析 40
第 6 章 結論與未來方向.....41
參考文獻.....42

圖目錄

圖 2-1 HBase 架構圖.....7
圖 3-1 mixed 方法之功能 1 流程圖.....15
圖 3-2 mixed 方法之功能 2 流程圖.....18
圖 3-3 mixed 方法之功能 3 流程圖.....19
圖 3-4 mixed 方法之功能 4 流程圖.....21
圖 3-5 mixed 方法之功能 5 流程圖.....22
圖 4-1 assorted 方法之功能 1 流程圖.....25
圖 4-2 assorted 方法之功能 2 流程圖.....26
圖 4-3 assorted 方法之功能 3 流程圖.....28
圖 4-4 assorted 方法之功能 4 流程圖.....28
圖 4-5 assorted 方法之功能 5 流程圖.....28
圖 5-1 五種類型查詢句的執行時間 (資料集 1).....32
圖 5-2 五種類型查詢句的執行時間 (資料集 4).....33
圖 5-3 Q1 & Q5 之執行時間(s).....34
圖 5-4 HBase 資料庫表格 split 的數量.....36
圖 5-5 all_object 表格之 Meta Data.....36
圖 5-6 Q2 之執行時間.....37
圖 5-7 Q3 & Q4 之執行時間.....38
圖 5-8 將型號資料獨立建立於一個 Column Family 時 Q1 之執行時間.....39
圖 5-9 將型號資料獨立建立於一個 Column Family 時 Q5 之執行時間.....40

表目錄

表 2-1 member 表格 8
表 2-2 功能一 9
表 2-3 功能二 10
表 2-4 功能三 10
表 2-5 功能四 11
表 2-6 功能五 11
表 3-1 prod_flow 表格 13
表 3-2 all_object 表格 13
表 3-3 mixed 方法之功能 1 實作 17
表 3-4 mixed 方法之功能 2 實作 19
表 3-5 mixed 方法之功能 3 實作 20
表 3-6 mixed 方法之功能 4 實作 21
表 3-7 mixed 方法之功能 5 實作 23
表 4-1 fxc 表格 24
表 4-2 prod_desc 表格 25
表 4-3 assorted 方法之功能 1 實作 25
表 4-4 assorted 方法之功能 2 實作 27
表 4-5 assorted 方法之功能 3 至功能 5 實作 27
表 5-1 人工資料集的物品工單 30
表 5-2 查詢句 31
表 5-3 資料集 1 Q1～Q5 之執行時間 (s) 32
表 5-4 資料集 4 Q1-1～Q5-1 之執行時間 (s) 33
表 5-5 Q1 & Q5 之執行時間 (s) 34
表 5-6 Q5 在 Column Family 數量不相同時的執行效率 (s) 37

表 5-7 all_object_plus 表格.....39
表 5-8 Q1 在不同 Block Size 時的執行時間(s).....40
表 5-9 Q2 在不同 Block Size 時的執行時間(s).....40

第1章 序論

在此章，我們先敘述本論文的研究動機與目的，同時提出本論文的研究方法與貢獻，並針對相關研究進行討論，最後說明本論文各章節的內容規劃。

1.1 研究動機與目的

在全球競逐智慧科技發展和就業人口遞減的趨勢下，德國和中國大陸分別提出了“工業 4.0”和“中國製造 2025”政策，而台灣相繼推出“生產力 4.0”計劃[TP4.0I] 。在生產力 4.0 的發展規劃中，主要希望能應用智慧機械、物聯網、巨量資料、雲端計算等技術來引領製造業、商業服務、農業產品等領域提升其服務價值。

在生產力 4.0 計劃中的製造業領域，製造執行系統(Manufacturing Execution System, MES)是其中一個被探討的議題。MES 主要是從工廠接到訂單之後從事生產到產品完成間，收集製造現場的各種資訊並且提供管理者各種的即時資訊，讓管理者可以隨時監控機台狀況並且有效的追蹤工單的製造流程以及預定的生產狀況。建置製造執行系統，能有效的縮短製造週期、確保產品品質、以及提供產線和管理者之間一個整合的溝通平台。

目前市面上常見的 MES 系統使用的資料庫多數為關聯式資料庫[OMES]，基於製造業中資料量會持續增加，達到巨量資料，本論文研究如何把這些生產資訊存入 NoSQL 資料庫。在眾多 NoSQL 資料庫中我們選擇了 HBase 資料庫，該資料庫是種以欄為導向的資料庫(Column-Oriented Database) ，選用的考量如下：

- HBase 資料庫能輕易處理大量的讀寫操作。
- HBase 資料庫 Schema 的定義較為彈性。
- HBase 資料庫對各種程式語言提供了簡單的 API，達到開發者親和性。

- 建立在 Hadoop 的 HDFS 環境下的 HBase 資料庫提供了資料複製，單一節點的毀損不會影響資料的可用性。

- HBase 資料庫不需要專用的高效能伺服器，它們可以輕易運行在一般硬體組成的叢集上。

因此，本論文的目的就在考量生產資訊量龐大、資料的備份及查詢的速度下，研究如何將這些生產資訊儲存至 HBase 資料庫，並提供生產流程查詢與控管功能。

1.2 研究方法與貢獻

本論文所欲建立的系統，希望能夠追蹤一個物品或者是相同型號的物品，其經過站台的資訊，主要是進、出站的時間。我們假設每個物品上貼有條碼,而且每個站台皆有條碼讀碼器。我們的做法是將各個站台所讀取的物品進出站資訊存到資料庫表格中，並且設計對應的程式達到生產流程查詢的功能。同時本論文的系統也支援流程控管，也就是是當使用者在查詢某物品時，若該物品沒有經過其所先行規劃的正確流程，系統將顯示其錯誤資訊。

在實作方面，我們將所有資訊儲存在 HBase 資料庫，功能方面則採用 Java 程式語言和其提供的 HBase Java API 讀寫資料並完成查詢的動作。由於 HBase 是一種 NoSQL 資料庫，它並不像關聯式資料庫中提供了 SQL 查詢語法、Aggregate Function 等等的功能，所以我們主要使用其 get 或 scan 函式抓出資料表的資料後，再搭配 Java 程式完成查詢功能。本論文提出兩種定義資料庫 schema 的方法，其貢獻簡述如下：

- 在 mixed 方法中，我們將型號和序號合併，共同儲存在同一表格下。在查詢同一型號的所有物品或某一序號時，我們利用 scan 功能提供的 filter 針對 row key 做 substring 的特殊過濾即可得到物品資訊。

- 在 assort 方法中是將每個型號獨立建成一個表格，記錄同一型號下所有物品的資訊。在查詢同一型號的所有物品時只需要 scan 以型號命名的表格即可得到物品資訊，而在查詢某一序號的物品資訊時則需先查詢前置資料表中此一序號所屬的型號才能完成 get 此筆物品資訊的動作。

- 我們進行實驗比較上述兩種方法，實驗結果顯示 mixed 方法和 assorted 方法若可以使用 Row Key 對表格進行查詢，抓出單一筆資料時執行效率不會受資料量影響。而上述兩種方法在抓出表格內多筆資料時，即是使用 Scan 函式，則會隨著資料量變大而影響執行效率。也就是 HBase 對於單一資料以列為查詢條件的處理非常快速且穩定，對於掃瞄整個表格的查詢句則會隨著資料量呈現線性成長。

1.3 相關研究

首先，我們討論製造執行系統的相關研究。近年來已經有相當多的模組和技術在處理生產資訊並儲存在資料庫，在[TCCY13]中，提出了以 RFID 技術來實現零延遲和零庫存為目標，進而實現自動化過程、監測以及生產狀況，並改善良率。[CSDS11] 則介紹用分散式系統來處理大規模的 RFID 資料，並提出在單一架構下，結合容錯的推論和站台位置，來處理複雜的查詢句，藉由分散式的推論與查詢處理，使得效率和延展性都有較佳的效率。在[李 15]中，提出了在 NAS（Network Attached Storage）上面開發 MES 系統，論文中提出了兩種方法，『Baseline 方法』是利用原始的條碼資料，並根據使用者下達的條件組成合適的 SQL 查詢句，『FTT（Flow-Time Tree）方法』是將原始條碼資料轉成追蹤資料，並根據物品經過的站台流程與時間來建立流程時間樹，再利用此樹設計相對的 SQL 查詢句。

其次，我們討論 HBase 資料庫的相關研究。在[MNV11]中，介紹了 Hadoop、HDFS 和 HBase 的架構，並對 HBase 和關聯式資料庫的隨機讀寫做出比較，結果顯示 HBase 資料庫在隨機讀寫的效率比較好。HBase 被稱為 NoSql 資料庫，其資料的表示是一種類資料表的資料結構，因此在定義 schema 時與一般關聯式

資料庫有很非常大的差距性。在[AK12]介紹了 HBase 資料的模型與特性，譬如：HBase 資料庫只以 Row Key 作為索引，並以其作為排序的依據；在 HBase 中所有資料都是以 Byte[]形態被儲存；Column Family 必須在建立表格的時候事先定義，而 Column Qualifier 可以隨時在寫入時新增。此外，在定義 schema 時可以考量以下因素: (1)每一列的 Row Key 應該用怎樣的結構？(2)要有多少的 Column Family？(3)哪些資料應該存入哪些 Column Family 裡？在[ABCD*12]中，提出在資料量龐大的因素下，Facebook Messenger 為什麼選擇 HBase 作為資料庫，並提出 HBase 資料庫的優點和可改進的地方。

最後，我們討論基於雲端環境的研究成果。Hadoop 是目前在利用雲端運算進行大量資料分析的領域中，常被使用的系統，雖然它可以很有彈性的擴充所使用的節點，但是之前的研究顯示其效率比傳統的平行資料庫系統差。論文[JOSW10]討論 5 個影響 Hadoop 效能的因素，包含 (1)架構上的問題(2)資料從 HDFS 取出時的 I/O 負擔(3)parsing 資料時使用 immutable decoder 造成的負擔 (4) sort 時比對 key 的方法(5)建立 index 的方式。該論文在改善上述問題之後，Hadoop 的效率可以達到和平行資料庫差不多的效率。

另外雲端環境也用來解決不同的問題。論文[SWL13]發現在圖形的應用方面，不論是及時的查詢需求，或是離線的分析需求，都需要大量而隨機的資料存取，而目前大量資料必須存放硬碟，但是記憶體才能提供更有效率的存取速度。因此，[SWL13]提出一個架構在分散的記憶體雲（distributed memory cloud）的查詢引擎，稱作 Trinity，該系統可以達到 general-purpose 的圖形管理和運算。其次，論文[ZCTW13] 特別針對 RDF 圖形資料提出 EAGRE（Entity-Aware Graph coompREssion）做法。首先將格式為 (subject , predicated, object)的 RDF 資料轉換成 RDF Entity Graph 有向圖，再進一步將其轉換成 Compressed RDF Entity Graph。該篇論文並定義 Consulting 的概念，以便在進行雲端平行處理必須將資訊送給其他 stage 時，能找出 cost 最低的排程以達到降低 I/O 和網路傳輸的時間。該篇

論文所提出的做法，對於有特別順序要求，如具有 ORDER BY 子句的查詢句，特別有效率。另外[LWST13]則探討大量資料的索引建立與維護機制。該索引的建置可以基於 MapReduce 的 processing engine 上，所以並不會明顯需要額外的成本。在索引的做法，該論文提出 compact bitmap indexing，利用目前 bitmap 的壓縮技術，像是 WAH encoding 和 bit-sliced encoding，來壓縮索引。至於[AMC15]則提出了一個 MapReduce 的框架 HadoopViz，以便處理任意形態且龐大的空間資料，並將其視覺化。

1.4 論文架構

本論文其餘各章節的架構如下。第二章敘述本論文的相關系統架構，其中包含系統的主要功能與資料介面，藉以對本論文的研究有基礎的認識。第三章和第四章分別介紹 mixed 方法和 assorted 方法的資料庫定義及其查詢句實作方式。第五章中以實驗比較兩種方法的查詢句效率。最後於第六章提出結論並討論未來可能的研究方向。

第2章 系統功能介紹

在此章我們先介紹 HBase 雲端資料庫的特性與及基本概念，接著再介紹本系統運作的環境與所提供的功能。

2.1 HBase 資料庫特性 [AHadoop, AHbase]

Hbase 是一個為了方便存取在 Hadoop 環境下的資料而發展出來的資料庫。Hbase 資料庫與關聯式資料庫架構上有極大的不同，這類型的資料庫通常稱為 NoSql 資料庫。Hbase 資料庫系統主要的特點如下：

- 分散式 ： HBase 是架構在 Hadoop HDFS(Hadoop Distributed File System)上的分散式資料庫，由 Master、ZooKeeper 和 Region Servers 三類 server 組成。Master Server 負責管理所有的 Region Servers 並記錄 MetaData，ZooKeeper 負責協調各個節點之間的聯繫，Region Servers 則管理和存取實際存放的資料內容。參考 Block 圖 2-1，在 HBase 中，一個 table 首先由一個 HRegion 組成，當表格不斷增大，為達到負載平衡，會 split 為多個 HRegions，不同的 HRegion 會被 Master 分配給對應的 Region Server 進行管理。在進行 split 的動作時，Master 的 MetaData 記錄了每個 region 中開始和結束的 Row Key。在 Hadoop 中，主要考慮分區儲存（Partitioning/Sharding）和資料複製（Replication）兩方面。其設計的概念是在 cluster（叢集）中有數個節點(DataNode)用來存放資料。一個檔案會被分割成數個區塊（Block，一般 size 為 128M），並將區塊複製數個複本，再分散儲存於不同的節點上。

- 高可靠性、高效能性：HBase Cluster（叢集）中存在多個 Region Server，任意一台 Region Server 中斷服務，將自動由另一台繼續提供服務。每一個分散的資料區塊各自擁有獨立的運算能力，當使用者對資料庫進行操作的時候，分散的資料各自獨立運算處理，同步使用者的要求，達到提高效能的作用。

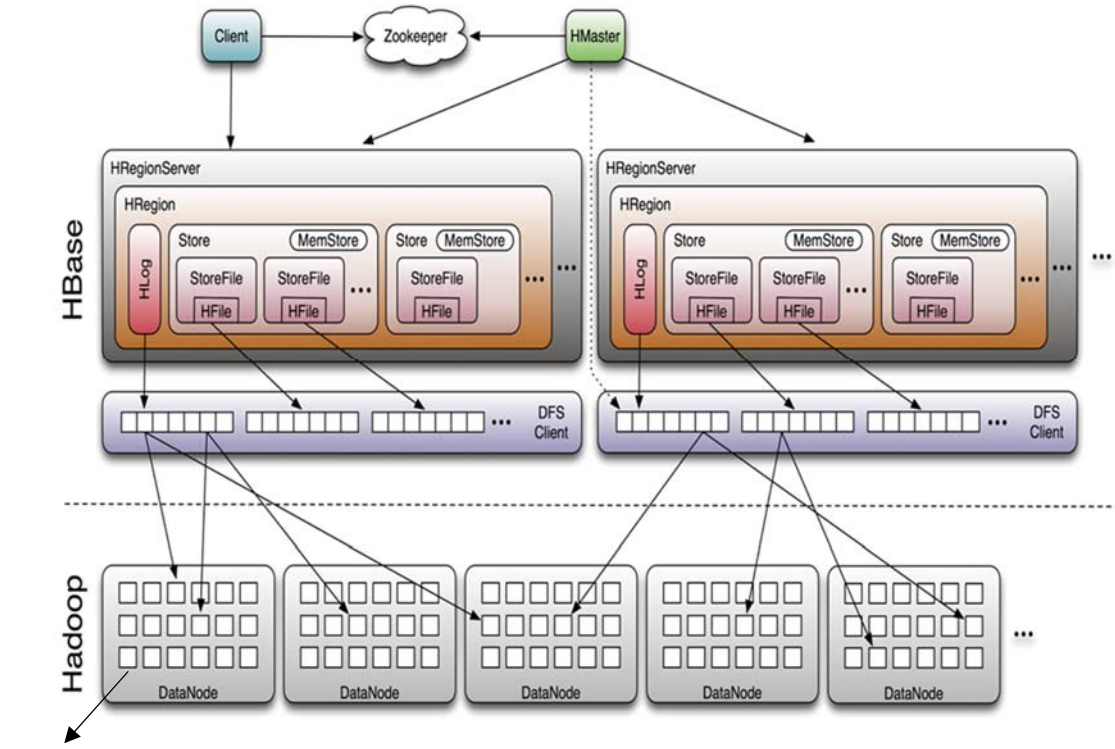


圖 2-1 HBase 架構圖

另外，HBase 提供類似表格的資料結構，但是以 Key-Value 的概念儲存資料，並透過 Row key、Column Families、Column Qualifier 指定唯一的資料內容，這些結構的關係詳述如下：

- Table ：HBase 中將資料組織成為 table，資料表以字串命名。
- Row ：在 HBase 中，每筆資料是以一欄作為儲存依據，並以 Row Key 為索引。Row Key 的資料形態會轉為 Byte[]儲存。
- Column Family：一列中會由數個 column family 組成，其需要在定義 schema 時就先行定義好。
- Column Qualifier: 一個 Column Family 是由數個 Column Qualifier 組成的集合。資料在儲存的時候，需指定某一 Row Key 的 Column Family 內的某 Column Qualifier 為依據，類似 field 的概念。
- Cell ：由 Row Key、Column Family 及其 Column Qualifier 定義，一個 Cell 的表示如：(RowKey, Column Family:Column Qualifier, value)。

以下我們利用 HBase 的 shell command 建立一個名為 member 的表格(表 2-1)以說明上述結構的關係。表格主要記錄會員的姓名，個人與公司的住址和聯絡電話。表格中 Row Key 為會員姓名, Column Family 為 personal 和 office，每個 Column Family 都有 address 和 phone 兩個 Column Qualifier，在表 2-1 中以斜體字表示。

1. >create “member”, “personal”, “office”

#create 命令建立一個表格，接受的參數為 “表格名稱” 、“Column Family1”、“Column Family2” 、 … 。為不影響效率，推薦不使用超過 3 個 Column Family。

2. >put “member”, “John”, “personal：address” , “ 高雄市中正區一號”

   >put “member”, “John”, “personal：phone” , “ 06-162332”

   >put “member”, “John”, “office：address” , “ 新北市北投區一號”

   >put “member”, “John”, “office：phone” , “ 02-365232”

#put 命令針對 Row Key 的某個 Column 做新增的動作，接受的參數為 “表格名稱” ， “Row Key” 、“Column Family：Column Qualifier” 、“value” ，以上範例為新增 John 的私人和公司電話及地址。

表 2-1 member 表格

Row Key	Column Family—personal		Column Family—office	
John	***address*** 高雄市中正區 1 號	***phone*** 06-162332	***address*** 新北市北投區 1 號	***phone*** 02-365232

2.2 系統架構與功能

在本節中首先說明本論文假設的系統運作環境。我們假設每個站台都配置有進站(in)和出站(out)兩個條碼讀碼器。一個物品的條碼內容為此物品型號(product number)及序號(serial number)。在此我們假設型號和序號以特殊符號“-”隔開，而序號的格式為一組由英文或數字所組成的字串然後以“-”符號接上流水號，例如[FXC-8A00529-00001]。在工廠中物品在生產線上的狀況，是由每個站台所附的條碼讀碼器讀取條碼資訊後回傳存入資料庫中，然後我們將所接收到的條碼經過處理後提供使用者特定的查詢功能。我們提供不同類型的查詢句，其中『scan query』表示取出大筆資料，『point query』表示取出特定資料，各個查詢句的功能如下：

- 【功能一：scan query】給定一個型號，查詢該型號所有物品何時經過哪些站台並依照時間先後順序排序。

此功能的操作範例如表 2-2。假設輸入參數為型號“FXC”，則會輸出對應序號進出特定站台的時間，如(S-00,stationA,in,2)代表序號 S-00 物件進入站台 stationA 的時間為“2”。在此，in 代表該物品進站的時間， out 則為該物品出站的時間。

表 2-2 功能一

[功能一]	輸入參數：{FXC}	
	輸出內容：(S-00,stationA,in,2)	(S-01,stationA,in,2)
	(S-00,stationA,out,3)	(S-01,stationA,out,3)
	(S-00,stationB,in,5)	(S-01,stationB,in,7)
	(S-00,stationB,out,6)	(S-01,stationB,out,8)
	(S-00,stationD,in,9)	(S-01,stationD,in,10)
	(S-00,stationD,out,11)	(S-01,stationD,out,13)

- 【功能二：point query】給定一個序號，查詢該物品何時經過某些站台並依照時間先後順序排序。

此功能的操作範例如表 2-3。假設輸入參數為序號“S-OO”，則會輸出此物品的資訊，如(stationA, in, 2)，即 S-00 這個物品在時間 2 進入站台 stationA。

表 2-3 功能二

[功能二]	輸入參數：｛S-00｝ 輸出內容：(stationA,in,2)　(stationA,out,3) (stationB,in,5)　(stationB,out,6) (stationD,in,9)　(stationD,out,11)

- 【功能三：point query】給定一個序號，查詢該物品經過前、後站台(所有組合)所需要花的時間。

此功能的操作範例如表 2-4。假設輸入參數為“S-00”，則輸出結果為該物品經過站台的所有組合的時間，本系統使用兩個站台進站（in）的時間相減作為兩站所需要花的時間。從表 2-3 得知 S-00 依序經過 3 個站台 stationA，stationB，stationD，所以其所有組合的時間為：(stationA, stationB, 3)、(stationA, stationD, 7)、(stationB, stationD, 4)。

表 2-4 功能三

[功能三]	輸入參數：｛S-00｝ 輸出內容：(stationA,stationB,3) (stationA,stationD,7) (stationB,stationD,4)

- [功能四：point query] 給定一個序號，查詢該物品經過前、後站台(相鄰站台)所需要花的時間。

表 2-5 功能四

[功能四]	輸入參數：{S-00} 輸出內容：(stationA,stationB,3) (stationB,stationD,4)

此功能的操作範例如表 2-5。假設輸入參數為“S-00”，則輸出結果為該物品經過相鄰站台的時間：(stationA, stationB, 3)、(stationB, stationD, 4)。

- [功能五：scan query]給定一個型號，查詢該型號所有物品經過相鄰站台所需的平均時間。

此功能的操作範例如表 2-6。假設輸入的參數為“FXC”，則輸出結果為該型號的所有序號經過相鄰所需的平均時間，計算的方式為將所有序號相鄰的兩個站台的時間差相加再除以序號個數。根據表 2-2，型號 FXC 對應到兩個序號 S-00 和 S-01。由表 2-3 已知序號 S-00 從 stationA 到 stationB 的時間差為 3 秒，stationB 到 stationD 的時間差為 4 秒；另外計算序號 S-01 從 stationA 到 stationB 的時間差為 5 秒，stationB 到 stationD 的時間差為 3 秒，平均所得結果如所示表 2-6。

表 2-6 功能五

[功能四]	輸入參數：{FXC} 輸出內容：(stationA,stationB,4) (stationB,stationD,3.5)

第3章 以型號與序號合併之方法

在本章中介紹第一個做法，稱作以型號與序號合併之方法，簡稱 mixed 方法，也就是將所有物品資料存放在同一個表格。我們先說明資料定義，接續再討論如何實作第 2 章所提之四項功能。另外，接下來所使用的範例資料，我們假設有“bp”和“fxc”兩個型號，而每個型號個別對應到兩個序號。

3.1 資料定義

為了達到流程控管，本系統會先建立一個前置資料的表格取名為 prod_flow，此表格記錄每個型號正確的流程，以檢查物品是否經由事先定義好的流程站台經過。表格的範例如表 3-1，我們將型號作為 Row Key，flow 作為 Column Family，f 作為 Column Qualifier，可看到型號“fxc”的物品必須依序經過 stationA、stationB、stationD。

線上資料的方面，如前所述，每個站台其進出站的條碼讀碼機掃瞄到物品上的條碼後，都會將其回傳至 server 端儲存在資料庫。本作法的想法即是在掃瞄到物品的條碼後，將條碼中的型號和序號合併當作表格中的 Row Key，in 和 out 作為 Column Family 代表進站和出站，其對應的 Column Qualifier 則為條碼讀碼機所在的站台位置，表格的範例如表 3-2。注意到 HBase 允許彈性定義 Column Qualifier，所以在同樣的 Column Family 中我們可看到不同型號的物品其 Column Qualifier 可以不一樣。另外如 1.3 節所述，HBase 會以 Row Key 作為依據自動排序，由於我們的 Row Key 是型號+序號，所以在表 3-2 中，物品都會自動先依型號再依其序號進行排序，這項特點讓我們能夠確保相同型號的物品會連續擺放在表格中，以便於接下來的查詢句設計。

在此我們不將型號獨立建成一個 Column Family 的原因在於太多的 Column Family 會影響執行效率。而我們不將型號建立在 Column Family(in 或 out)中的其中一個 Column Qualifier 的原因在於這樣的方法在查詢型號下的所有物品時需要

針對 Column Qualifier 進行 filter 的特殊過濾，進而影響執行效率，我們將在實驗的部分做後續的說明。

表 3-1 prod_flow 表格

Row Key	ColumnFamily——flow
bp	***f*** stationA，stationC，stationD
fxc	***f*** stationA，stationB，stationD

表 3-2 all_object 表格

Row Key	ColumnFamily——in			ColumnFamily——out		
bp-SC-001	***stationA*** 16:03:05	***stationC*** 16:03:10	***stationD*** 16:03:15	***stationA*** 16:03:08	***stationC*** 16:03:14	***st***ationD 16:03:20
bp-SC-009	***stationA*** 17:00:01	***stationC*** 17:00:07	***stationD*** 17:00:10	***stationA*** 17:00:05	***stationC*** 17:00:09	***stationD*** 17:00:15
fxc-S-00	***stationA*** 13:59:00	***stationB*** 13:59:32	***stationD*** 13:59:38	***st***ationA 13:59:30	***stationB*** 13:59:35	***stationD*** 13:59:59
fxc-S-01	***stationA*** 14:00:00	***stationB*** 14:00:32	***stationD*** 14:00:38	***stationA*** 14:00:30	***stationB*** 14:00:37	***stationD*** 14:00:41

3.2 功能實作

在本節中，我們說明如何利用 Java API 配合 3.1 節所定義的表格，實作所欲達到的功能。為了方便後續說明，我們先提供 API 中所使用的幾個 class 和 method 的基本定義。

- Connection

此物件接受的參數為 HBase 主要的兩個設定檔，hbase-site.xml 和 core-site.xml。這兩個設定檔的路徑需記錄在 Java API 提供的 configuration 這個 class

內。此物件也提供 getTable()這個 method 以表格名稱作為參數，得到對特定表格的操作。

- Get

此物件以 row key 為參數轉換成 Byte[]形態，提供給 Table class 中的 get() method 使用。

- Result

此物件儲存 get 操作後或 ResultScanner 內每筆的 value。提供 getValue()和 getRow()兩個 method。getValue()可以獲取一列當中某 cell 的 value，接收的參數為 column family 和 column qualifier。getRow()則回傳一列中的 row key。

- Scan

掃瞄整個表格時必須宣告的物件。提供 setFilter() method 以指定在掃瞄整個表格的時候所要做的特殊過濾，而 addFamily() method 則指定所需的 Column Family。至於 setStartRow() 和 setStopRow() 這兩個 method 則可以指定某 Row Key 為掃瞄的開始和結束，接受的參數為 Row Key。

- Result Scanner

一個 Result Scanner 物件記錄 scan 操作後回傳的值，與 Result 物件不同的地方在於可接收超過一筆的 value。

- Table

一個 Table 物件對應到一個表格，提供 get()和 getScanner()兩個 method。Get()接受的的參數為 Get 物件，其回傳值為表格中某一筆 Row Key 的資料。getScanner()是用來掃瞄整個表格，接受的參數為 Scan 物件。在操作 getScanner 時需要將抓到的 value 存放在 ResultScanner 這個物件中。

- Filter

可以依據 row key 或 column family 進行特殊的過濾查詢，其中包括 row filter 對 rowkey 做條件篩選。其提供了各類的比較器，包括 regular expression 比較、二進制比較器、一般數學類比較（等於，不等於，大於，小於）等等。

- Cell

如第 2 章所介紹，cell 是由 row key、column family:column qualifier、value 所組成，當我們對表格進行單一個 get 或 scan 可以將 result 存放在 cell 中，以得到每一列中所有的 value。提供 cloneQualifier()和 cloneValue()，用來取得某 cell 中 column qualifier 和 value 的值。

利用以上的 API，當我們對型號下所有物品做查詢的時候可以利用 setStartRow 和 setStopRow 對掃瞄做限制條件抓取資料。在對某筆序號做查詢時則使用 row filter 搭配 substring comparator 抓出對應的資料。接下來，我們針對各個功能說明確切的實作方式。

- 【功能 1】給定一個型號，查詢該型號([product])所有物品何時經過哪些站台並依照時間先後順序排序。

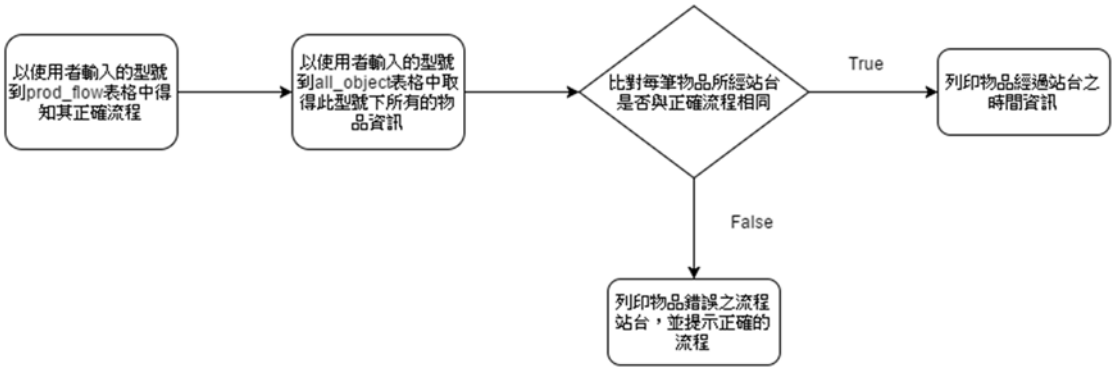


圖 3-1 mixed 方法之功能 1 流程圖

實作功能 1 的流程圖請參考圖 3-1，對應此功能的程式碼則如表 3-3 所示。L01 至 L04 是對 HBase 進行設定檔的設置以及連線，並宣告了 conn 這個 Connection 的物件。之後所有功能實作中皆會引用 L01 至 L04，論文在之後將會將其省略。L05 系統先取得 prod_flow 表格，L06 利用參數[product]宣告一個 Get

物件，L07 利用 API 中的 get()將此型號正確的流程抓出來後並存在 result 物件中。 L08 使用 getValue()指定需要的 column family 為 flow，column qualifier 為 f，取得值後利用 split 把以『，』隔開的站台列表儲存在 flow 陣列中。此段的功能為取得此型號的正確流程。

接下來，我們要取得對應此型號的所有物件。L09 將 table 指定為 all_object 這個表格。L10 宣告一個 scan 物件。L11 和 L12 利用 setStartRow()和 setStopRow()限制 scan 的範圍。舉例說明，在 HBase 中型號[product]下的所有物品為 bp-SC-001 至 bp-SC-009，setStartRow()接受了『"bp"』為參數後會找到有 bp 這個 substring 的第一列，而 setStopRow()接受了『 "bp"+"-end"』為參數後的動作是讀到 substring 與『 "bp"+"-end"』不相同後就會結束，因為 bp-SC-009 下一筆為 fxc-S-00，讀到這裡時 scan 動作就會停止，而回傳的是 bp-SC-001 至 bp-S-009。setStartRow()為 inclusive 而 setStopRow()為 exclusive，而這裡所使用的"-end"字串為一個假設不會出現於序號中的字串。

最後，我們要將物件的進出站時間抓出來，並指出有錯誤流程的物件。L13 利用 getScanner()掃瞄 all_object 表格並將結果存在 ResultScanner 物件。L14 將 ResultScanner 物件中每一列抓出放在 res 中。L16 利用 getRow()抓出此列 Row Key 的值。L17 利用 cell 抓出此列(res)的所有 Column Family：ColumnQualifier 的 value。L18 和 L19 是將 cell 中的 qualifier 和 value 的值抓出。假設 bp-SC-001 此物品經由 stationA、stationC、stationD，根據 L08 取出的 flow 物件及其 flow.legth 我們得知為 3。而根據表 3-2 每一列都有兩個 column family 分別為 in 和 out，所以一列中我們取出來的值會有六筆，所以 L17 中迴圈會執行(flow.length*2)=6 次，前 3 次分別為進站，後 3 次為出站。L20 至 L29 的 if 判斷式中，首先是比對站台是否與正確流程相同，若出現走入錯誤流程，則印出訊息並跳出迴圈，接下來根據 flow.length 判斷當 i 小於 length 時，則為進站，反之為出站。L24 和 L27，當 i=0 和 i=3 時輸出的範例分別為(bp-SC-001,stationA,in,14:00:00)和(bp-SC-

001,stationA,out,14:00:30)。若該物品第一站走到 stationC，則輸出範例如下：(bp-SC-001 in error flow! NO.1 correct station: stationA, incorrect station: stationC)。

注意到，本論文處理的錯誤流程狀況，只有經過的站台個數相同但站台名稱不同，若是其他情況(如站台個數不同)則暫不處理。另外，根據本論文中 all_object 表格的 schema 設計，物品的資料在表格中是依照序號排序，且我們假設物品經過站台的順序也是根據序號由小到大，所以在程式實作中不需要特別做排序的動作以符合由時間從小排到大的需求。

表 3-3 mixed 方法之功能 1 實作

L01	Configuration conf ← new Configuration()
L02	conf.addResource(“hbase-site.xml”)
L03	conf.addResource(“core-site.xml”)
L04	Connection conn ← new Connection(conf)
L05	Table table ← conn.getTable(“prod_flow”)
L06	Get g ← new Get([product])
L07	Result result ← table.get(g)
L08	String flow[] ← result.getValue(“flow”,“f”).split(“,”)
L09	table ← conn.getTable(“all_object”)
L10	Scan scan ← new Scan()
L11	scan.setStartRow([product])
L12	scan.setStopRow([product]+”-end”)
L13	ResultScanner scanner ←table.getScanner(scan)
L14	**for each** res **in** scanner
L15	int i=0
L16	String rowkey ← res.getRow()
L17	**for each** cell **in** res
L18	String qualifier ← Cell.cloneQualifier(cell)
L19	String value ← Cell.cloneValue(cell)
L20	**if** i < flow.length **&& !**flow[i].equals(qualifier)
L21	print(rowkey +“in error flow! NO.”+ i + “correct station:” + flow[i] + “incorrect station:” + qualifier)

```
L22             else if i < flow.length
L23                 print("("+ rowkey + ","+ qualifier + ",in ," + value + ")")
L24             else
L25                 print("("+ rowkey + ","+ qualifier + ",out ," + value + ")")
L26             end if
L27             i++
L28         end for
L29     end for
L30     scanner.close()
```

- 【功能 2】給定一個序號（[serial_num]），查詢此物品在什麼時間經過了哪些站台。

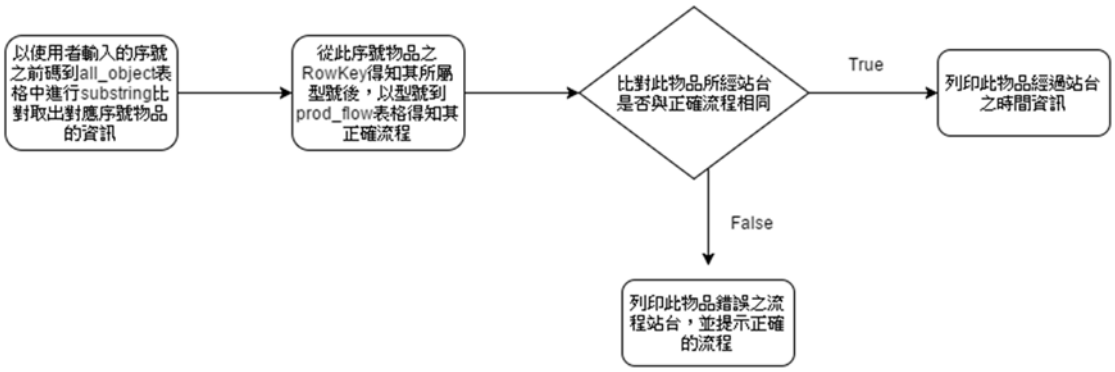


圖 3-2 mixed 方法之功能 2 流程圖

實作功能 2 的流程圖請參考圖 3-2，對應的程式碼則如表 3-4 所示，此功能與功能 1 的差別在於型號可以有很多序號，而序號只有一筆。由於在 all_object 表格中，Row Key 由型號和序號合併，所以在 L03 我們利用[serial_num]為 substring 建立 row filter，比較表格中的 Row Key 以取出對應的物品。為了得到此物品正確的流程站台，我們必須知道此物品的型號，所以 L07 利用 get 取出物品的 row key 中以『—』隔開的第一位置，例如 Row Key 為 fxc-S-00，則會取出 fxc，接下來則可利用其為參數到 prod_flow 表格查詢其正確流程，如 L08 行之後的程式碼所示。

表 3-4 mixed 方法之功能 2 實作

	//同表 3-3 中 L01 至 L04 建立連線
L01	Table table ← conn.getTable(“all_object”)
L02	Scan scan ← new Scan()
L03	filter ← new RowFilter(CompareFilter.CompareOp.EQUAL ,new SubStringComparator(“[serial_num]”))
L04	scan.setFilter(filter)
L05	scanner ←table.getScanner(scan)
L06	Result result ← scanner.next()
L07	Get g ← new Get(result.getRow().split(“-”)[0])
L08	table ← conn.getTable(“prod_flow”)
L09	Result re ← table.get(g)
L10	String flow[] ← re.getValue(“flow”,”f”).split(“,”)
	//同表 3-3 中 L17 至 L30，比對是否經由正確流程並列印出此物品的進出站資訊

- 【功能 3】給定一個序號([serial_num])，查詢該物品經過前、後站台所需要花的時間。

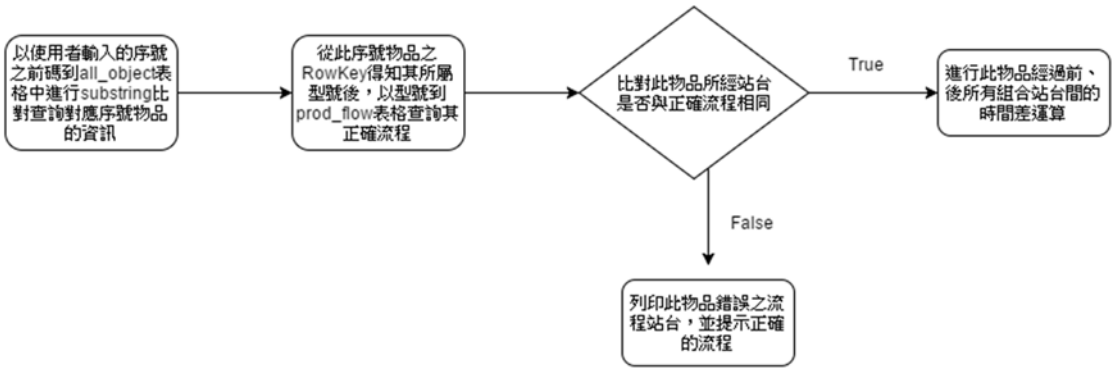


圖 3-3 mixed 方法之功能 3 流程圖

功能 3 的實作請參考圖 3-3 流程圖，對應的程式碼則如表 3-5 所示。此功能與功能 2 的差別是在取出此物品的站台資訊後，我們必須以此物品的進站時間算出此物品經過前、後站台所需的時間。由於只需要用到進站時間所以在程式實作中，我們利用 addFamily(in)讓 scan 時只抓取 in（進站）這個 Column Family。

注意 flow 陣列存放所經過的站台順序，L01 宣告變數 i 代表所經過的站台數以確保每個前、後站台都有做比較。由於經過站台間所需要的時間差是以進站時間做相減，L02 中利用 getValue()，將 column family 設為 in，qualifier 為第 i 個站台，抓出值後以分隔字元[:]分別取出時、分、秒三個值，並存入 time1 的 string 陣列。舉例來說利用 getValue()後抓出的值範例為：『13:59:00』 而 time1 陣列中，第 0 的位置會存放小時『13』，第 1 的位置存放分鐘『59』，第二的位置存放秒數『00』。接下來，由於每小時為 3600 秒，每分鐘為 60 秒，經過 L03 運算後 t1 是此物品進入站台 i 的時間，並換算成秒數作為單位。L04 我們將 j 設定為 i 之後至所經過的站台數 ，確保第 i 位置之後的每個站台都會與 i 做時間差的比較。L05 至 L06 與 L02 至 L03 目的相同。L07 將 t2 和 t1 作相減的動作即可得到兩站間所需的時間差。以序號 S-00 為例，以站台 stationB 的進站時間(13:59:32)減站台 stationA 的進站時間(13:59:00)，輸出的範例為(stationA,stationB,32)。

表 3-5 mixed 方法之功能 3 實作

	// 同表 3-4 中 L01 至 L10，但是 L10 行之前要加入指令『scan.addFamily(in)』
L01	**while** int i < flow.length
L02	String time1[] ← result.getValue(“in”,flow[i]).split(“:”)
L03	int t1 = Integer.valueof(time1[0])*3600 + Integer.valueof(time1[1])*60 + Integer.valueof(time1[2])
L04	**while** int j=i+1 **&&** j < flow.length
L05	String time2[] ← result.getValue(“in”,flow[j]).split(“:”)
L06	int t2 = Integer.valueof(time2[0])*3600 + Integer.valueof(time2[1])*60 + Integer.valueof(time2[2])
L07	**if**(t1==0 || t2==0)
L08	print(rowkey +“in error flow!)
L09	break
L10	**end if**
L11	print(flow[i],flow[j],t2-t1)

L12	j++
L13	**end while**
L14	i++
L15	**end while**

- 【功能 4】給定一個序號，查詢該物品經過相鄰站台所需要花的時間。

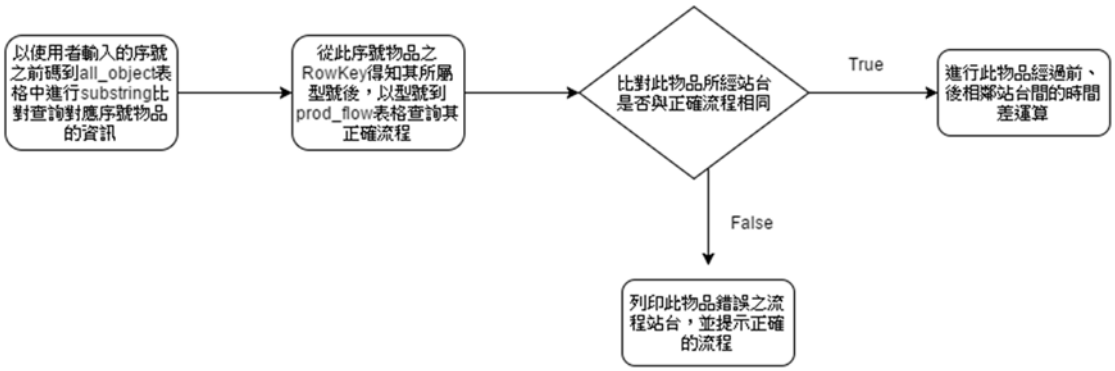


圖 3-4 mixed 方法之功能 4 流程圖

此功能的實作參考圖 3-4 之流程圖，由於此功能只需要抓取相鄰站台，所以表 3-5 的變數 j 直接改成 i+1，對應的程式碼如表 3-6 所示。L04 利用判斷式確保站台 i 的下一站存在，若存在則這兩站為相鄰，我們才會計算兩站之間的時間差。

表 3-6 mixed 方法之功能 4 實作

	// 同表 3-4 中 L01 至 L10，但是 L10 行之前要加入指令『scan.addFamily(in)』
L01	**while** int i < flow.length
L02	String time1[] ← res.getValue(“in”,flow[i]).split(“:”)
L03	int t1 = Integer.valueof(time1[0])*3600 + Integer.valueof(time1[1])*60 + Integer.valueof(time1[2])
L04	**if** i+1 < flow.length
L05	String time2[] ← res.getValue(“in”,flow[i+1]).split(“:”)
L06	int t2 = Integer.valueof(time2[0])*3600 + Integer.valueof(time2[1])*60 + Integer.valueof(time2[2])

L07	**if**(t1==0 || t2==0)
L08	print(rowkey +"in error flow!)
L09	break
L10	**end if**
L11	print(flow[i],flow[i+1],t2-t1)
L12	**end if**
L09	i++
L10	**end while**

- 【功能 5】給定一個型號，查詢該型號([product])所有物品經過相鄰站台平均所需要花的時間。

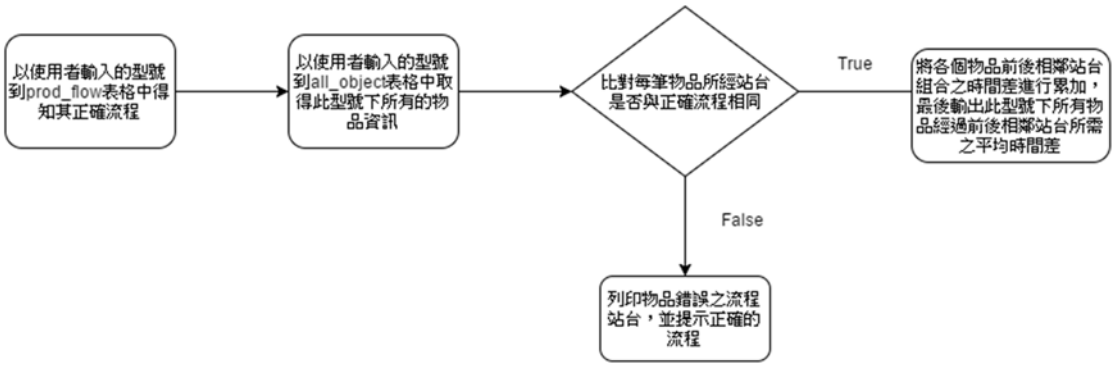


圖 3-5 mixed 方法之功能 5 流程圖

參考圖 3-5 之流程圖，相較於功能 4，我們必須將同一型號的所有物品的時間累加到一個暫存變數以便計算平均值。首先我們同表 3-3 抓出對應此型號的所有物件，然後存放於物件 scanner 中。L01 宣告了一個 int 陣列名為 temp，大小為所經站台數-1。假設我們型號[product]為 fxc，其經過的站台為(stationA,stationB,stationD)，那麼物品經過相鄰站台的時間差會有兩筆，分別為 A 到 B 和 B 到 D，由此我們可推論一個物品所經相鄰站台的時間差個數為站台數-1。此例子中 temp 的大小為 2，temp[0]記錄的值為 A 至 B 的時間差，temp[1]記錄的值為 B 至 C 的時間差。

接下來，L02 中 count 變數用以記錄此表格中有多少筆資料。L03 將 scanner 物件中每一列抓出放在 res 中。L06 至 L10 與表 3-6 中 L02 至 L06 相同，用來計

算經過相鄰站台所需之時間，然後於 L11 中將其相加。L16 至 L21 我們將 temp 陣列中記錄的時間差總和除以物品個數(count 變數)，得到此型號所有物品經過相鄰站台的平均所需時間。針對型號“fxc”和表 3-2 的資料，則會輸出 (stationA,stationB,32)， (stationB,stationD,6)。

表 3-7 mixed 方法之功能 5 實作

	// 同表 3-3 中 L01 至 L13，但是 L13 行之前要加入指令『scan.addFamily(in)』
L01	int []temp ← new int[flow.length – 1]
L02	int count ← 0
L03	**for each** res **in** scanner
L04	count++
L05	**while** int i < flow.length
L06	String time1[] ← res.getValue(“in”,flow[i]).split(“:”)
L07	int t1 = Integer.valueof(time1[0])*3600 + Integer.valueof(time1[1])*60 + Integer.valueof(time1[2])
L08	**if** i+1 < flow.length
L09	String time2[] ← res.getValue(“in”,flow[i+1]).split(“:”)
L10	int t2 = Integer.valueof(time2[0])*3600 + Integer.valueof(time2[1])*60 + Integer.valueof(time2[2])
L11	temp[i] += t2 –t1
L12	**end if**
L13	i++
L14	**end while**
L15	**end for**
L16	**while** int i=0 < flow.length
L17	**if** i+1 < flow.length
L18	print(flow[i],flow[i+1],temp[i]/count)
L19	**end if**
L20	i++
L21	**end while**

第4章 以型號分類之方法

本章說明所提出的第二個方法，稱為以型號分類之方法，簡稱 assorted 方法，也就是將每個型號的物品資料分開存放於不同的表格。

4.1 資料定義

第三章所提之 mixed 方法將所有物品資料存放與同一個表格，但是由於我們並不需要進行不同型號物品間的比較，所以本論文中提出的 assorted 方法是將每個型號各自建立成一個表格，然後以物品序號作為該表格的 Row Key。在表 4-1 我們列舉型號 fxc 表格和其所定義的屬性。該表格以序號(serial_num)作為 Row Key，in 和 out 作為 Column Family，in 和 out 的 Column Qualifier 則由物品所經過的站台組成。in 在本論文中代表進站，out 代表出站。

表 4-1 fxc 表格

Row Key	ColumnFamily—in			ColumnFamily—out		
S-00	***stationA*** 13:59:00	***stationB*** 13:59:32	***stationD*** 13:59:38	***stationA*** 13:59:30	***stationB*** 13:59:35	***stationD*** 13:59:59
S-01	***stationA*** 14:00:00	***stationB*** 14:00:32	***stationD*** 14:00:38	***stationA*** 14:00:30	***stationB*** 14:00:37	***stationD*** 14:00:41

但是，利用此方法在對序號做查詢時需要事先知道這個序號隸屬於哪個表格，如 2.2 節中所提序號的格式是由一組由英文或數字組成的字串然後以[“-”]符號接上流水號，所以在前置資料 prod_desc 中，我們記錄每個型號其以[“-”]符號隔開前的這組前置字串，且為了控管物品是否經由正確流程，所以一併記錄了每個型號的正確流程。如表 4-2 所示，該表格是以型號作為 row key，Column Family 為 serial 和 flow。serial 的 Column Qualifier 為 s，記錄序號的前置碼，flow 的 Column Qualifier 為 f，記錄此型號正確的流程。

表 4-2 prod_desc 表格

Row Key	ColumnFamily--serial	ColumnFamily--flow
bp	***s*** S	***f*** stationA,stationC,stationD
fxc	***s*** SC	***f*** stationA,stationB,stationD

4.2 功能實作

- 【功能 1】給定一個型號([product])，查詢該型號號所有物品何時經過某些站台並依時間排序。

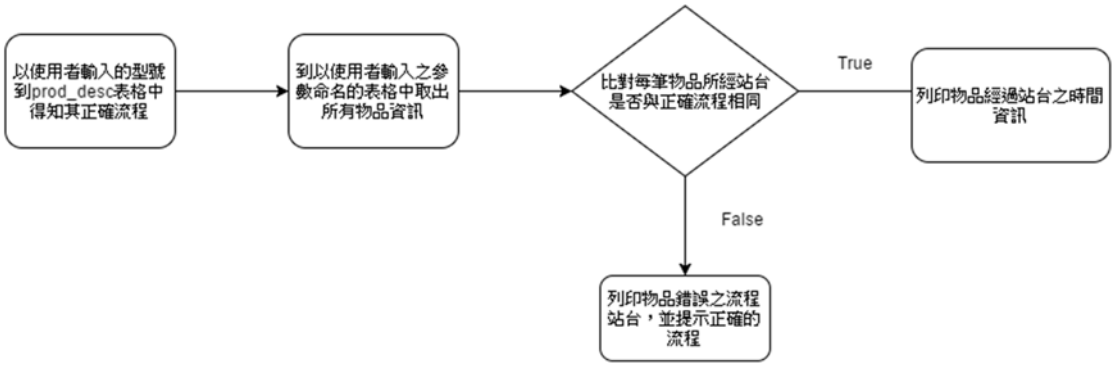


圖 4-1 assorted 方法之功能 1 流程圖

參考圖 4-1 之流程圖，與前一個方法的差別是於 L01 我們直接指定到以[product]命名的表格，而 L02 代表不需要設定任何限制條件。

表 4-3 assorted 方法之功能 1 實作

	//同表 3-3 中 L01 至 L08
L01	table ← conn.getTable([product])
L02	Scan scan ← new Scan()
	//同表 3-3 中 L14 至 L32

- 【功能 2】給定一個序號（[serial_num]），查詢此物品在什麼時間經過了哪些站台。

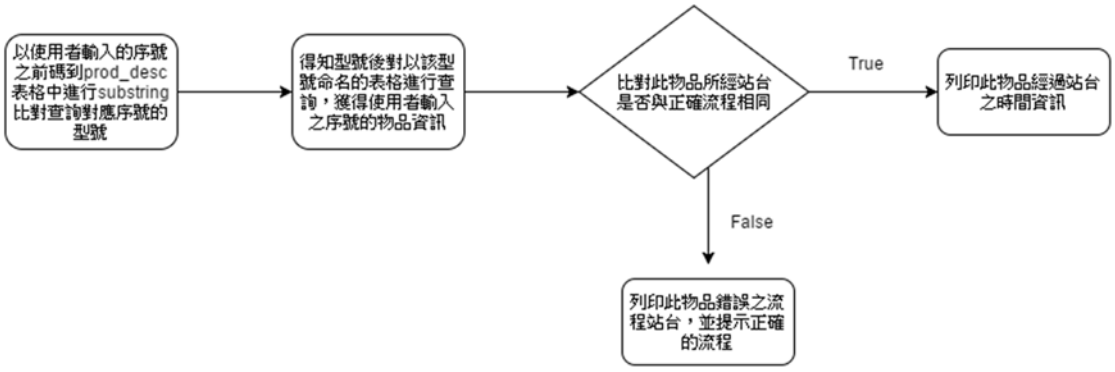


圖 4-2 assorted 方法之功能 2 流程圖

參考圖 4-2 之流程圖，對應的程式碼如表 4-4。如前所述，我們必須用序號的前置字串來決定其型號。舉例說明，在 L01 若使用者輸入的序號為 S-00，我們將其以分隔符號"-"隔開後將 "S"存在變數 s_front。 L03 至 L07 我們利用 filter 的特性，讓 s_front 為 substring 針對 prod_desc 表格中 column family 為 serial，qualifier 為 s 的一欄做特殊過濾，找出 s_front 屬於哪一個 Row Key，從中得知此序號對應到哪個型號，也就是屬於哪個表格。L08 利用 getRow()將抓出的 Row Key 存在 product 變數。L10 我們將 table 指定為名為 product 的 table。L11 將使用者輸入的 serial_num 作為參數建立一個 Get 物件。L12 將在 product 表格抓出的值放入 result 物件中。接下來的執行方式則同第三章的方法。

表 4-4 assorted 方法之功能 2 實作

	//同表 3-3 中 L01 至 L04 相同
L01	String s_front ← [serial_num].split(“-”)[0]
L02	Scan scan ← new Scan()
L03	SingleColumnValueFilter filter ← new SingleColumnValueFilter(“serial”, "s" ,CompareFilter.CompareOp.EQUAL, new BinaryComparator(s_front));
L04	scan.setFilter(filter);
L05	Table table ← conn.getTable(“prod_desc”)
L06	ResultScanner scanner ←table.getScanner(scan)
L07	Result res ← scanner.next()
L08	String product ← res.getRow()
L09	String flow[] ← res.getValue(“flow”,”f”).split(“,”)
L10	table ← conn.getTable(product)
L11	Get g ← new Get([serial_num])
L12	Result res ← table.get(g)
	//同表 3-4 中 L11 至 L25 相同

表 4-5 assorted 方法之功能 3 至功能 5 實作

【功能 3】 給定一個序號([serial_num])，查詢該物品經過前、後站台所需要花的時間。	//同表 4-4 中 L01 至 L12 //同表 3-5 中 L01 至 L11
【功能 4】 給定一個序號([serial_num])，查詢該物品經過相鄰站台所需要花的時間。	//同表 4-4 中 L01 至 L12 //同表 3-6 中 L01 至 L10
【功能 5】 給定一個型號，查詢該型號([product])所有物品經過相鄰站台平均所需要花的時間。	//同表 4-3 中 L01 至 L02 //同表 3-7 中 L01 至 L21

接下來的三個功能，由於其實作方式與之前的方式大致相同，所以一起列舉於表 4-5，對應的流程圖則參考圖 4-3、圖 4-4、圖 4-5。如表 4-5 所列，實作功能 3 和功能 4，主要是運用功能 2 中抓取表格資料的方式後再進行個別的計算，而在功能 5 中則是運用功能 1 中抓取表格資料的方式。

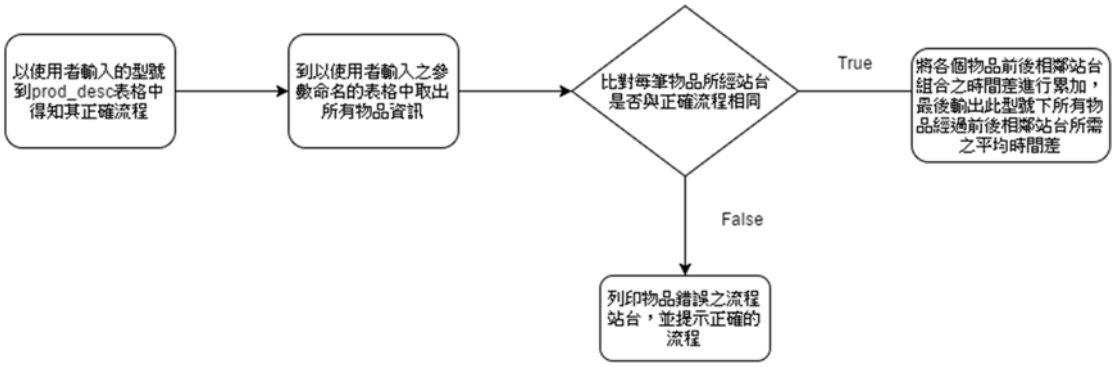


圖 4-3 assorted 方法之功能 3 流程圖

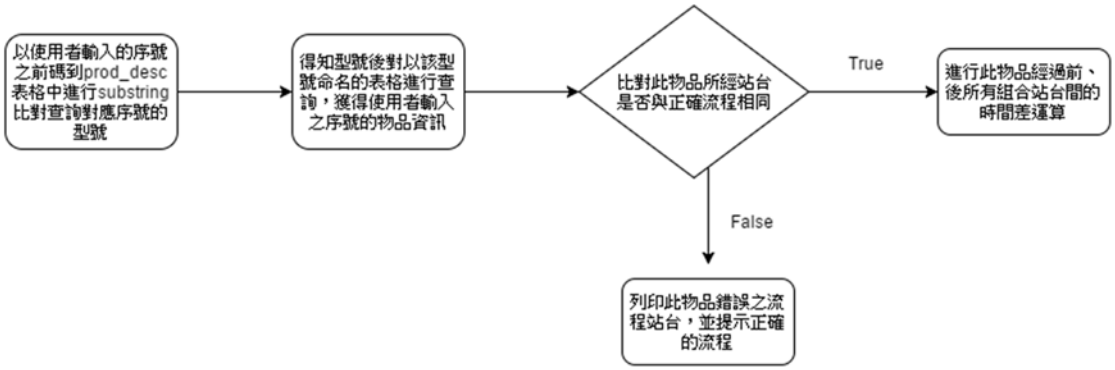


圖 4-4 assorted 方法之功能 4 流程圖

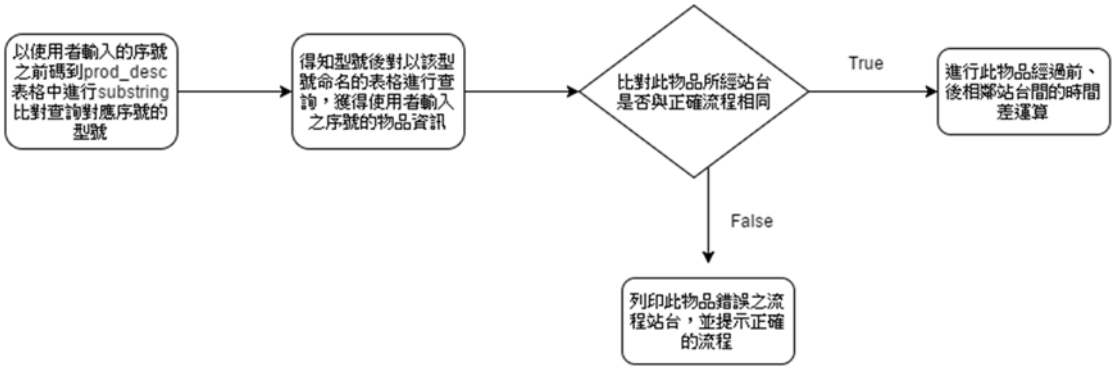


圖 4-5 assorted 方法之功能 5 流程圖

第5章 實驗

在本章節中，針對第 3 章所提出之 mixed 方法，和在第 4 章所提出之 assorted 方法，我們進行實驗來比較其效率。首先說明我們進行的實驗的環境，我們在四台電腦架設 Hadoop 和 HBase 環境。Master Server 的電腦配置為 CPU ：Core 2 Quad 2.2G 和 RAM：DDR2 2G。Region Server 中三台電腦的配置分別為 CPU：Core 2 Quad 2.1G、Core 2 Quad 2.4G、Core 2 Quad2.66G 和 RAM：DDR2 2G、DDR2 2G、DDR2 2G。

在 Hadoop 中以一台為 Master，其他三台為 Slave；HBase 則將在 Hadoop 中當 Master 的主機設為 Master Server，其他三台為 Region Server。我們所採用的作業系統為 Ubuntu 14.04，Hadoop 版本為 2.7.2，HBase 版本為 2.0.0。在實驗環境中，每台電腦透過網路分享器利用內網互相連線。在 Hadoop 的設定中，我們將 data replication factor 設為 3， Block Size 為 128M，HRegion 的 Max File Size 設定為 1G，也就是當表格容量超過設定值後會進行 split。

5.1 資料集和查詢句設計

由於物件的生產資料沒有眾所皆知的真實資料集，因此我們基於部分工廠本身的真實情況加以延伸產生出人造資料集，其中包含物品的型號和序號以及正確流程。如表 5-1 所示，我們有四張工單，型號分別為「FXC」、「GDA」、「BP」和「CS」這四種，其序號的前置字串分別為“8A00529”、“8B00529”、“8A00528”、“8C00528”，正確流程則分別為“A,B,D”、“A,C,D,E”、“A,B,C”、“A,C,D”，其中每個英文字母對應到一個站台。

表 5-1 人工資料集的物品工單

資料集	型號	序號（起）	序號（迄）	正確流程	數量	資料大小(mb)
1	FXC	8A00529-00001	8A00529-30000	A,B,D	30,000	8.15
2	GDA	8B00529-000001	8B00529-300000	A,C,D,E	300,000	111
3	BP	8A00528-0000001	8A00528-3000000	A,B,C	3,000,000	849
4	CS	8C00528-00000001	8C00528-30000000	A,C,D	30,000,000	8460

在表 5-2 中列出我們在實驗所使用的查詢句，我們以分號作為輸出與限制條件的區隔，其中 S/N 和 P/N 代表物品的序號和型號。注意到 Query1 至 Query5 是針對第一個資料集(資料量最小)，而 Query1-1 至 Query5-1 則是針對第 4 個資料集(資料量最大)。另外，Query1、Query1-1 和 Query2、Query2-1 主要是追蹤物品在哪些時間經過了哪些站台，對應到第三章功能 1 和 2。Query3、Query3-1 和 Query4、Query4-1 主要是針對單一物品經過各個站台的時間差，對應到第三章的功能 3 和 4。Q5 和 Q5-1 主要是找出一個型號下所有物品經過相鄰站台平均所需要的時間，對應到第三章功能 5。依據所需要取出的資料列數在原表格所占的比例，我們也將查詢句分做兩類，其中 Query1(Query1-1)和 Query5(Query5-1)需要取出大量筆數，稱做 scan query，而 Query2(Query2-1)至 Query4(Query4-1)因為只需抓取特定的一列，所以稱做 point query。我們的測量執行時間的方式為：執行查詢句五次，並取平均值，執行時間是由開始執行到結束後但不包含輸出訊息的時間，而 I/O 時間指的是從在進行 cloneQualifier()、cloneValue()、getRow()、getValue()這四類 API 的時間。

表 5-2 查詢句

Query Number	Query	類型	Query Number	Query	類型
Query1	<S/N, 站台名稱, 進出站台時間; P/N="FXC">	scan	Query1-1	<S/N, 站台名稱, 進出站台時間; P/N="CS">	scan
Query2	<站台名稱, 進出站台時間; S/N="8A00529-00001">	point	Query2-1	<站台名稱, 進出站台時間; S/N="8C00528-00000001">	point
Query3	<前站 L1, 所有後站 L2,Timediff(L2,L1)); S/N="8A00529-00001">	point	Query3-1	<前站 L1, 所有後站 L2,Timediff(L2,L1)); S/N="8C00528-00000001">	point
Query4	<前站 L1, 相鄰後站 L2,Timediff(L2,L1)); S/N="8A00529-00001">	point	Query4-1	<前站 L1, 相鄰後站 L2,Timediff(L2,L1)); S/N="8C00528-00000001">	point
Query5	<前站 L1, 相鄰後站 L2, AVG(L1,L2)); P/N="FXC">	scan	Query5-1	<前站 L1, 相鄰後站 L2, AVG(L1,L2)); P/N="CS">	scan

5.2 不同查詢句的效率評估

首先我們先用一個比較小的資料集來測量 mixed 方法與 assorted 方法針對不同類型查詢句的執行效率，這次實驗用的資料筆數為 30K 筆（資料集 1）。參考圖 5-1 和表 5-3，雖然 Q2 到 Q4 是屬於 point query，兩個方法的 I/O 時間皆小於 1 毫秒，但是在 mixed 方法中給定一個序號要從 all_object 表格中抓取特定一筆資料時，只能對 Row Key 做 substring 的特殊過濾，造成執行時間比 assorted 方

法長許多，所以我們假設使用者在做查詢時會一起給予型號和序號，我們將其實作並加入此次試驗中，稱為『mixed*方法』。從圖表中可看出，mixed*方法和 assorted 方法在 Q2 至 Q4 都能根據使用者提供的參數直接對 Row Key 做查詢抓取特定的一筆資料，所以執行時間非常接近。

接下來，我們比較 scan query 的表現，mixed 方法和 mixed*方法在 Q1 和 Q5 的實作上沒有差別，都是從表格中抓取特定範圍的資料列，也就是實際上是執行 range query，所以時間約略比 assorted 方法多。另一方面，Q1 至 Q5 都必須抓取 3 萬筆資料，而 Q5 的 I/O 時間略比 Q1 快一些是因為 Q5 只讀取 Column Family 為 in 的資料，但 Q5 需要額外的 CPU 運算時間，所以整體而言，Q5 比 Q1 需要多一點的執行時間。

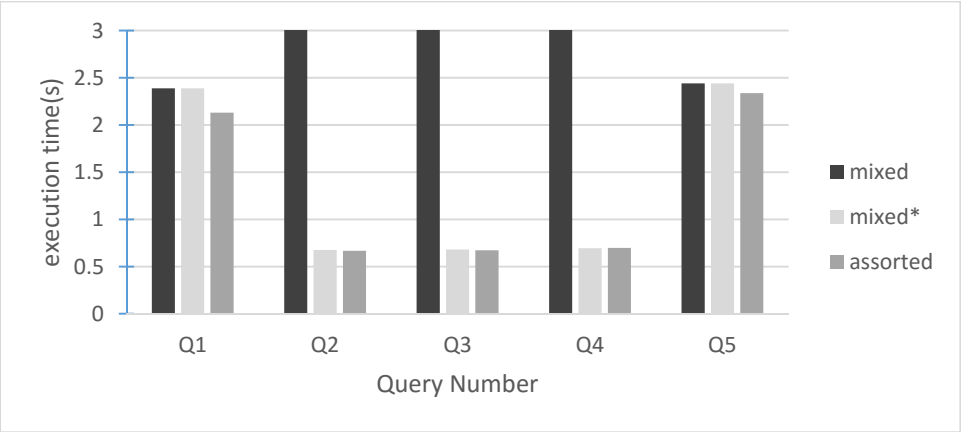


圖 5-1 五種類型查詢句的執行時間 (資料集 1)

表 5-3 資料集 1 Q1～Q5 之執行時間 (s)

	Q1	Q2	Q3	Q4	Q5
mixed	2.388	454.6956	69.7394	71.339	2.441
I/O 時間	0.253	<0.001	< 0.001	< 0.001	0.205
mixed*	2.3888	0.676	0.6816	0.695	2.441
I/O 時間	0.253	< 0.001	< 0.001	< 0.001	0.205
assorted	2.1298	0.6678	0.6722	0.6982	2.3376
I/0 時間	0.24	< 0.001	< 0.001	< 0.001	0.202

其次，我們利用一個比較大的資料集來比較 mixed 方法、mixed*方法和 assorted 方法。這次實驗用的資料筆數為 30M 筆（資料集 4）。參考圖 5-2 和表 5-4，我們可看到，依然是 assorted 最快，mixed 最慢。我們先討論 mixed 方法在五種類型的查詢句的執行效率，我們可以很明顯看到 scan query 都比 point query 需要更多的時間。針對 scan query，Q1 和 Q5，他們都要處理三千萬筆資料，但是 Q5 設定只讀取 Column Family 為 in 的值，所以在 I/O 時間上 Q5 比 Q1 快約 2 倍，但是 Q5 在讀取資料後必須進行運算，所以整體的執行時間比 Q1 快約 1.47 倍。接下來，針對 point query，Q2 至 Q4，從表 5-3 和表 5-4 可看出，兩筆不同的資料集在執行時間非常接近，這是由於 HBase 在執行特殊過濾時需要比對表格中每一列的 Row Key 後才回傳符合的值，根據 all_object 表格的設計所有物品資訊都放在同一表格，所以在 mixed 方法不管查詢任何型號下的序號之資訊，執行時間都十分相近。而 Q3 和 Q4 都比 Q2 快的原因一樣是受到只讀取某一 Column Family 所影響。最後，針對 mixed*方法和 assorted 方法在資料量為 30M 時，Q2 至 Q4 的執行時間與資料量為 30K 的執行時間相近，I/O 時間一樣維持在小於 1 毫秒。至於 Q1 和 Q5 皆因資料量變大以後執行時間相對提高許多。

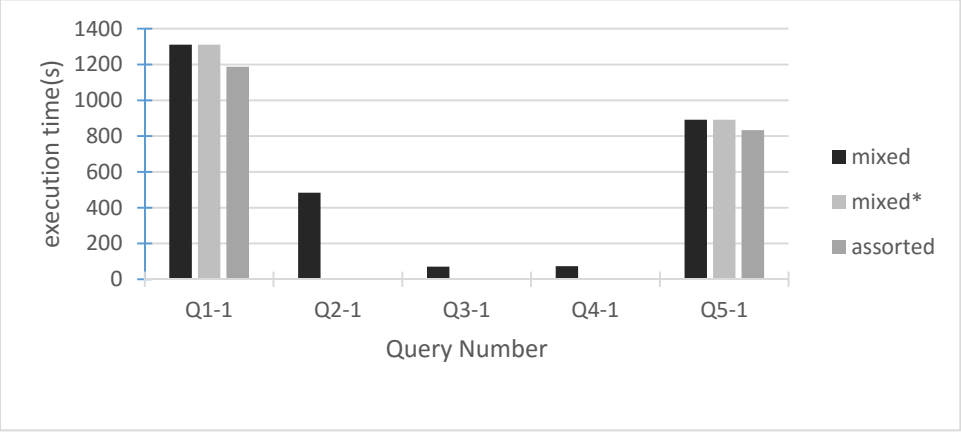


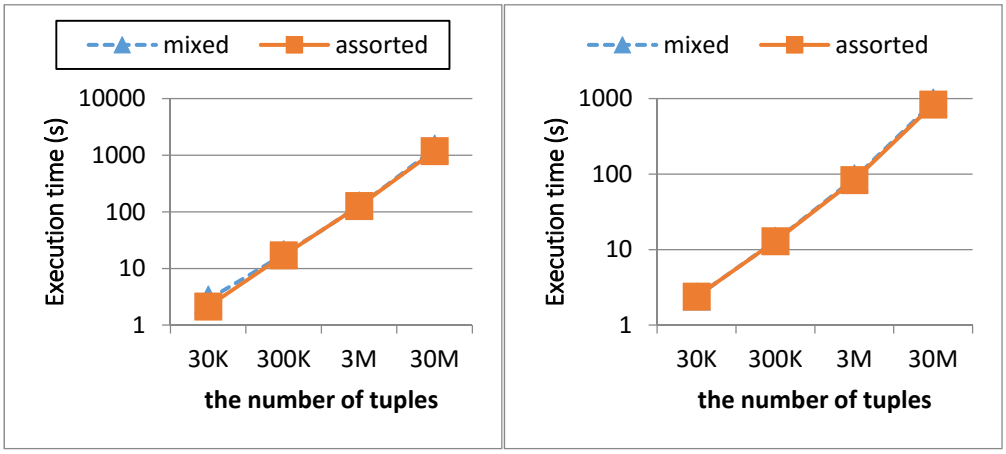
圖 5-2 五種類型查詢句的執行時間 (資料集 4)

表 5-4 資料集 4 Q1-1～Q5-1 之執行時間 (s)

	Q1-1	Q2-1	Q3-1	Q4-1	Q5-1
mixed	1310.617	483.647	70.888	73.265	891.827

I/O 時間	129.99	< 0.001	< 0.001	<0.001	64.789
mixed*	1310.617	0.7436	0.7708	0.778	891.827
I/O 時間	129.99	< 0.001	< 0.001	< 0.001	64.789
assorted	1187.843	0.7342	0.7618	0.75	833.734
I/O 時間	132.89	< 0.001	< 0.001	< 0.001	64.08

5.3 Scan Query 和 Column Family 的分析



(a) Q1 (b) Q5

圖 5-3 Q1 & Q5 之執行時間(s)

表 5-5 Q1 & Q5 之執行時間 (s)

Q1	30K	300K	3M	30M
mixed	2.388	17.9214	130.7798	1310.617
assorted	2.1298	17.1522	126.7016	1187.843
Q5	30K	300K	3M	30M
mixed	2.441	13.106	88.06	891.827
assorted	2.386	12.906	83.517	833.734

在本節中，我們針對四個資料集，設計對應的 Q1 和 Q5，四個資料集是以 10 倍數的數量增加。另外，Q1 中 mixed 方法對 all_object 表格進行 scan 的動作並設定了 startRow 和 endRow，而 assorted 方法則是直接 scan 對應的型號表格。

在此次的實驗中，參考圖 5-3 和表 5-5，針對 Q1 在 mixed 方法，資料集 2 的執行時間是資料集 1 的 7.5 倍，資料集 3 的執行時間是資料集 2 的 7.3 倍，而資料 4 的執行時間是資料集 3 的 10.02 倍。而 assorted 方法，資料集 2 的執行時間是資料集 1 的 8.05 倍，資料集 3 的執行時間是資料集 2 的 7.4 倍，資料 4 的執行時間則是資料集 3 的 9.37 倍。也就是兩個方法所需時間很接近，也大約隨資料量成線性成長。

在 Q5 中，我們只需要使用進站的時間來進行運算，為了提升效率，在兩種方法中 scan 的動作都設定了 addFamily(in)，只讀取 Column Family 為 in 的資料，再對其進行運算。針對 Q5 在 mixed 方法，資料集 2 的執行時間是資料集 1 的 5.37 倍，資料集 3 的執行時間是資料集 2 的 6.71 倍，而資料 4 的執行時間是資料集 3 的 10.1 倍。而 assorted 方法，資料集 2 的執行時間是資料集 1 的 5.4 倍，資料集 3 的執行時間是資料集 2 的 6.47 倍，資料 4 的執行時間則是資料集 3 的 9.98 倍。

若仔細分析，我們發現資料集 1(30K 筆)至資料集 3(3M 筆)中執行時間成長幅度較小，但資料集 4(30M 筆)與資料集 3(3M 筆)的執行時間是以約 10 倍在增長。此外，我們可看出 assorted 方法會比 mixed 方法的執行時間快了一些，我們將討論此原因。我們附上在系統上 HBase 資料庫中，表格 split 的真實情況。參考圖 5-4 我們可發現 mixed 方法中使用的 all_object 表格被分成了 8 個 region。參考圖 5-5，每個 region 的 start key 和 end key 都清楚被記錄，而根據框起來的部分可見 bp 和 cs 兩個型號的序號有部分被分在同一的 region 中，在進行 filter 這種特殊過濾或 setStartRow 和 setStopRow 限制時會需要比較或過濾不同型號的物品，進而影響效率。assorted 方法將每個型號各別建立表格，避免了不同型號的物品被分配在同一 region 中。

最後，我們分析控制 column family 數量對效率的影響。我們利用 Q5 中 assorted 方法的實作，將原來只讀取某一 Column Family 的資料(稱之為 assorted)

與讀取所有 Column Family 做比較(稱之為 assorted+)。參考表 5-6，可看出在資料量超過 300K 後兩者的執行效率相差約 1.7 倍。對應圖 2-1，可看出在資料量變大時 table 需要 split 成多個 Hregion，Hregion 是由 store 組成，一個 store 是由一個 Column Family 組成，所以在設定了 addFamily 指定讀取的 Column Family 時所讀取的資料相對的少了很多。

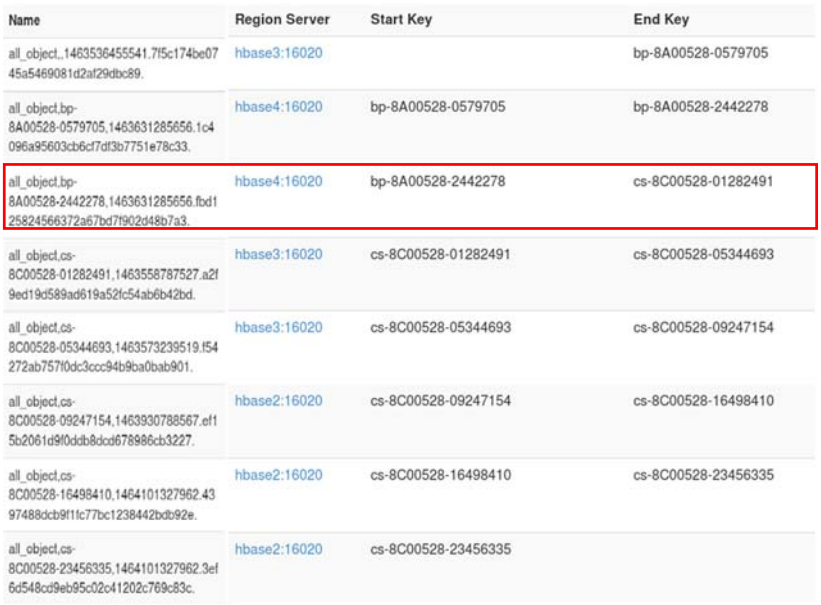
Tables

User Tables | System Tables | Snapshots

7 table(s) in set. [Details]

Namespace	Table Name	Online Regions
default	all_object	8
default	bp	3
default	cs	7
default	fxc	1
default	gda	1
default	prod_desc	1

圖 5-4 HBase 資料庫表格 split 的數量



Name	Region Server	Start Key	End Key
all_object,,1463536455541.7f5c174be0745a5469081d2af29dbc89.	hbase3:16020		bp-8A00528-0579705
all_object,bp-8A00528-0579705,1463631285656.1c4096a95603cb6cf7df3b7751e78c33.	hbase4:16020	bp-8A00528-0579705	bp-8A00528-2442278
all_object,bp-8A00528-2442278,1463631285656.fbd125824566372a67bd7f902d48b7a3.	hbase4:16020	bp-8A00528-2442278	cs-8C00528-01282491
all_object,cs-8C00528-01282491,1463558787527.a2f9ed19d589ad619a52fc54ab6b42bd.	hbase3:16020	cs-8C00528-01282491	cs-8C00528-05344693
all_object,cs-8C00528-05344693,1463573239519.f54272ab757f0dc3ccc94b9ba0bab901.	hbase3:16020	cs-8C00528-05344693	cs-8C00528-09247154
all_object,cs-8C00528-09247154,1463930788567.ef15b2061d9f0ddb8dcd678986cb3227.	hbase2:16020	cs-8C00528-09247154	cs-8C00528-16498410
all_object,cs-8C00528-16498410,1464101327962.4397488dcb9f1fc77bc1238442bdb92e.	hbase2:16020	cs-8C00528-16498410	cs-8C00528-23456335
all_object,cs-8C00528-23456335,1464101327962.3ef6d548cd9eb95c02c41202c769c83c.	hbase2:16020	cs-8C00528-23456335	

圖 5-5 all_object 表格之 Meta Data

表 5-6 Q5 在 Column Family 數量不相同時的執行效率 (s)

Q5	30K	300k	3M	30M
assorted	2.386	12.906	83.517	833.734
assorted+	2.9582	20.7834	144.6967	1447.036
倍數	1.24	1.6	1.73	1.74

5.4 Point Query 的分析

我們首先利用 Q2 針對四個資料集設計對應的查詢句進行實驗。參考圖 5-6，在 assorted 方法中，即使資料量已經達到 10^7 仍然保持穩定效率，主要是因為我們以物品序號作為 Row Key，而 HBase 資料庫會以 Row Key 建立索引，所以在 assorted 方法中以序號做查詢的動作都能快速找到相關物品資訊。這也說明 HBase 資料庫中 Row Key 的重要性，對於 Row Key 的設計會直接影響查詢的效率。在 mixed 方法中，我們使用 filter 進行 substring 比對來獲取相對的物品資訊。HBase 在進行特殊過濾時會把 all_object 表格讀取一遍之後再回傳符合的值，所以我們可看出 mixed 方法在查詢不同資料集的物品中執行時間沒有太大的差別且效率非常差。mixed*方法則與前文所述相同，假設使用者輸入的參數為型號與序號，在實作的時候就能與 assorted 方法實作中相同，直接對 Row Key 做查詢的動作，其執行的效率與 assorted 方法幾乎相同，在之後的 Q3 和 Q4 的試驗中就不將 mixed*方法加入。

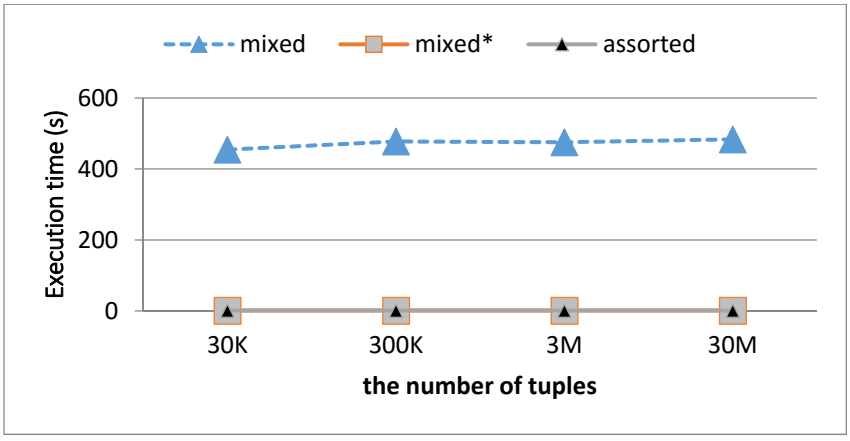
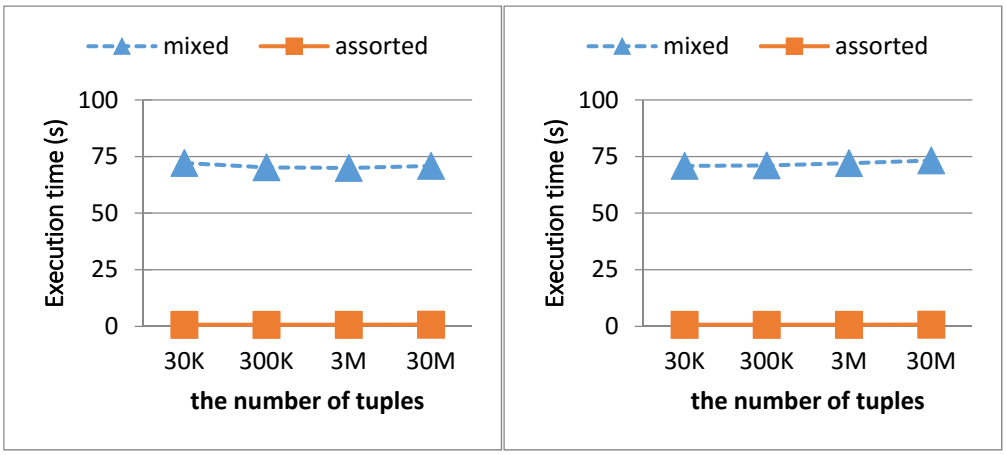


圖 5-6 Q2 之執行時間



(a) Q3 (b) Q4

圖 5-7 Q3 & Q4 之執行時間

最後，我們針對四個資料集設計 Q3 和 Q4 對應的查詢句，並進行實驗。參考圖 5-7，我們可看到此兩個 Query 在兩個方法中執行的時間沒有太大的差異，我們將共同討論其原因。在 mixed 方法中，我們必須先利用 filter 找出相對序號的物品資訊，此動作佔據的時間較多，且在 Q3 和 Q4 我們只讀取 in 這個 Column Family，所以執行時間比 Q2 少約 6.6 倍。在 assorted 方法中，其實作方式與 Q2 的差異在讀取到資料後需再進行時間差的運算，我們可看出這運算非常快速，資料量的增加對執行效率並沒有太大的影響。

5.5 將型號建立成獨立一個 Column Family 之實驗

這次實驗中，我們將利用 mixed 方法中的想法，即是把所有物品資訊都存放在同一個表格中，但是不將型號和序號合併當作 Row Key，而是將型號獨立建立成 Column Family。表格的範例如表 5-7all_object_plus 表格，而對應的實作方法稱作 mixed+。

表 5-7 all_object_plus 表格

Row Key	CF-in			CF-out			CF-p
SC-001	***stationA*** 16:03:05	***stationC*** 16:03:10	***stationD*** 16:03:15	***stationA*** 16:03:08	***stationC*** 16:03:14	***st***ationD 16:03:20	***no*** bp
SC-009	***stationA*** 17:00:01	***stationC*** 17:00:07	***stationD*** 17:00:10	***stationA*** 17:00:05	***stationC*** 17:00:09	***stationD*** 17:00:15	***no*** bp
S-00	***stationA*** 13:59:00	***stationB*** 13:59:32	***stationD*** 13:59:38	***st***ationA 13:59:30	***stationB*** 13:59:35	***stationD*** 13:59:59	***no*** fxc
S-01	***stationA*** 14:00:00	***stationB*** 14:00:32	***stationD*** 14:00:38	***stationA*** 14:00:30	***stationB*** 14:00:37	***stationD*** 14:00:41	***no*** fxc

我們針對四個資料集，設計 Q1(scan query)和 Q2(point query)的查詢句並進行實驗。在 Q1 的查詢句中，我們可以利用 filter 對 Column Family 為 p 的一欄進行字串比對，找出同一型號下所有的物品。參考圖 5-8，在新增了一個 Column Family 後，在資料集 1(最小資料筆數)中的執行時間就比 mixed 方法和 assorted 方法慢了將近 100 倍，而隨著資料筆數的增加，執行時間是呈緩慢成長。另一方面，參考圖 5-9，Q2 可以透過 Row Key 進行查詢，執行時間不受資料筆數的影響。根據此次實驗，針對 scan query，mixed+的執行效率會比 mixed 差，而針對 point query，mixed+是比 mixed 好。

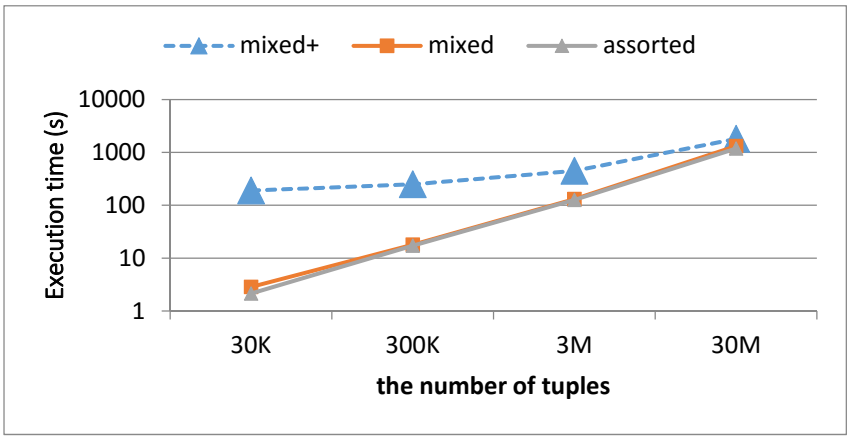


圖 5-8 將型號資料獨立建立於一個 Column Family 時 Q1 之執行時間

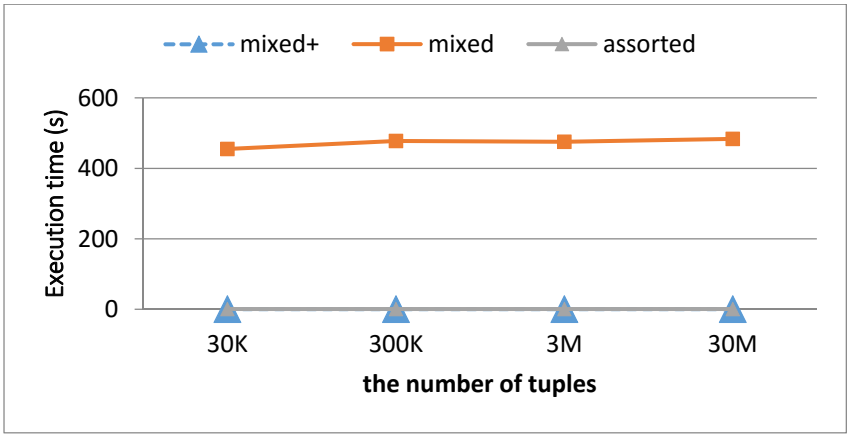


圖 5-9 將型號資料獨立建立於一個 Column Family 時 Q5 之執行時間

5.6 改變 Hadoop 中 Block Size 對執行效率的影響之分析

在此小節中，我們將進行當 block size 為 16Mb、128Mb(預設值)及 1Gb 時執行效率的比較。我們改變 block size 並針對四個資料集對 Q1(scan query)和 Q2(point query)進行實驗。參考表 5-8、表 5-9，在不同的 block size 中 Q1 和 Q2 的表現都是相近的，由此得知 Block Size 並沒有對執行效率造成影響，這是因為 HBase 的平行處理讓在讀取不同數量的 Block 時效率仍然維持穩定。注意表 5-8 和表 5-9 中，一欄內的資料左邊是 mixed 方法的執行時間，右邊為 assorted 方法的執行時間。

表 5-8 Q1 在不同 Block Size 時的執行時間(s)

Q1	30K		300K		3M		30M	
16M	2.447	2.129	19.574	17.152	141.877	126.701	1419.88	1187.843
128M	2.456	2.211	19.612	17.455	143.824	127.63	1416.58	1186.561
1G	2.437	2.153	19.523	17.345	141.966	127.95	1420.06	1178.456

表 5-9 Q2 在不同 Block Size 時的執行時間(s)

Q2	30K		300K		3M		30M	
16M	454.695	0.667	477.563	0.711	475.24	0.7338	483.647	0.7482
128M	461.741	0.678	475.12	0.705	475.872	0.7122	482.89	0.738
1G	458.45	0.668	477.693	0.7131	477.81	0.7156	483.804	0.7453

第6章 結論與未來方向

在本論文中，我們以 HBase 資料庫搭配 Java 程式語言建構出具有查詢生產流程功能的製造執行系統。我們提出了兩種資料庫 schema 的設計方式。第一個 mixed 方法，是以型號和序號合併作為 Row Key，將所有物品都存放在同一個表格中。此方法在對 point query 即是查詢某一序號的物品資訊時實作中使用 filter 來進行特殊過濾，造成查詢效率不佳的問題，而且我們發現在沒需要進行不同型號的物品比較的情況下，把不同型號的物品各別放在不同的表格執行效率會較好一些。所以我們進一步提出了 assorted 方法。此方法中，我們將各別的型號獨立建立成一個新的表格，以物品的序號作為 Row Key，且在前置資料表中記錄了序號的前碼以分辨此序號所屬的型號。

我們進行實驗來評估這兩種方法，分析各個查詢句在不同資料量時的執行效率。結果顯示，assorted 方法在 point query 中的表現比 mixed 方法來的好，在 Q2、Q3、Q4 查詢句中不會因為資料數量增加而影響執行效率。在 scan query 的表現中，兩種方法都隨著資料量變大而執行時間隨著變久，但 assorted 方法在執行效率上仍然較好。綜合而言，HBase 雖然會在資料大時自行分散資料，但是若能根據查詢句的特性事先利用不同的表格將資料分散，針對 scan 類型的查詢效率仍然有很大的改善。除此之外，Column Family 的數量在針對 scan 類型的查詢句上也會造成執行時間的緩慢，而在 point query 類型的查詢句中，只要能夠利用 Row Key 進行查詢就不會對執行時間有所影響。

最後，關於本論文未來的研究方向，由於在處理 scan query 上，執行效率隨著資料量變大而變差。我們希望透過 Apache Spark 這個運算平台，利用其在 main memory 的運算功能改善 scan query 的表現。

參考文獻

[ABCD*12] Amitanand Aiyer, Mikhail Bautin, Guoqiang Jerry Chen, Pritam Damania Prakash Khemani, Kannan Muthukkaruppan, Karthik Ranganathan Nicolas Spiegelberg, Liyin Tang, Madhuwanti Vaidya, “Storage Infrastructure Behind Facebook Messages Using HBase at Scale”, Bulletin of the IEEE Computer Society Technical Committee on Data Engineering, 2012.

[AHadoop] Apache Hadoop project, http://hadoop.apeche.org

[AHBase] Apache HBase Reference Guide, http://hbase.apache.org/book.html

[AK12] Amandeep Khurana, “Introduction to HBase Schema”, The Usenix Magazine, Vol.37, No.5, 2012.

[CSDS11] Zhao Cao, Charles Sutton, Yanlei Diao, Prashant Shenoy, “Distributed inference and query processing for RFID tracking and monitoring”, Proceedings of the VLDB conference, 2011.

[EMJ15] Ahmed Eldawy， Mohamed F. Mokbel， Christopher Jonathan， “HadoopViz: A MapReduce Framework for Extensible Visualization of Big Spatial Data”, Proceedings of the ICDE Conference, 2015.

[JOSW10] Dawei Jiang, Beng Chin Ooi, Lei Shi, Sai Wu, “The Performance of MapReduce: An In-depth Study”, Proceedings of the VLDB Endowment, Vol. 3, No. 1, 2010.

[LWST13] Peng Lu, Sai Wu, Lidan Shou, Kian-Lee Tan, “An Efficient and Compact Indexing Scheme for Large-scale Data Store”, Proceedings of the ICDE Conference, 2013.

[MNV11] Mehul Nalin Vora, “Hadoop-HBase for Large-Scale Data”, Proceedings of the International Conference on Computer Science and Network Technology, 2011.

[OMES] Oracle MES User Guide, https://docs.oracle.com/cd/B34956_01/current/acrobat/120gmomesug.pdf.

[SWL13] Bin Shao, Haixun Wang, Yatao Li, “Trinity: A Distributed Graph Engine on a Memory Cloud”, Proceedings of the ACM SIGMOD International Conference on Management of Data, Pages 505-516, 2013.

[TCCY13] Yung-Shun Tsai, Ruey-Shun Chen, Yeh-Cheng Chen, Chun-Ping Yeh, “An RFID-based Manufacture Process Control and Supply Chain Management in the Semiconductor Industry”, International Journal of Technology Management, Vol.12, No. 1/2, pp. 85-105, 2013.

[TP4.0I] 行政院生產力 4.0 發展方案，http://www.bost.ey.gov.tw/Upload/UserFiles/%E8%A1%8C%E6%94%BF%E9%99%A2%E7%94%9F%E7%94%A2%E5%8A%9B4_0%E7%99%BC%E5%B1%95%E6%96%B9%E6%A1%88.pdf

[ZCTW13] X. Zhang, L. Chen, Y. Tong, and M. Wang, “EAGRE: Towards Scalable I/O Efficient SPARQL Query Evaluation on the Cloud”, Proceedings of the ICDE Conference, 2013.

[李 15] 李韋錫，“快速處理生產流程查詢與控管之研究”，碩士論文，國立台灣海洋大學資工系研究所，2015。