以 API 為導向之 NoSQL 聚合資料模型之效能分析研究

陳筱婷, 丁建文*
國立高雄應用科技大學資訊管理系
E-mail: m1101345101@kuas.edu.tw, *jwding@kuas.edu.tw

摘要

隨著網際網路及雲端運算的發展，現今企業將應用程式及資料存放在資料中心，可動態提供服務及共用運算，使用者可以隨時隨地透過各式連線裝置存取應用程式，但隨著資料量的成長，資料的種類日漸複雜，以往傳統的關聯式資料庫已無法負荷，而海量資料(Big Data)成為目前全球所重視的議題，為了解決巨量資料的問題，許多企業紛紛開始投入研發建置成本較低的分散式開源資料庫，也就是 NoSQL 資料庫，因 NoSQL 具有 free-Schema、以及在叢集上良好的執行特性等，這項技術可使用較低階等級的伺服器，來撐起巨量資料需要的儲存規模和分析處理能力。

過去的研究主要多探討 NoSQL 與 SQL、OODB 的效能比較，至今少有學者提出有關 NoSQL 聚合資料模型評估效能之方法，本研究將根據應用程式為導向，將探討不同的聚合資料模型，在不同的 API(Application Performace Interface)及不同資料量中的效能差異，以本研究所提出的效能評估模式，進行資料模型間的效能優劣比較，透過比較模擬與實驗結果，以驗證了此評估方法的有效性。希望將來可作為使用者在設計 NoSQL 資料庫時的參考。

關鍵詞：BigData、雲端計算、NoSQL

Abstract

With the development of Internet and cloud computing, today's enterprise will store applications and data in a data center. A data center can provide dynamic services and shared computing, and users can use applications through a variety of devices anytime anywhere. However, with the the dramatic growth of data and the increasing number of data types, traditional relational databases are unable to afford the loading, while big data computing is currently an impartial research topic. In order to solve the problem of big data computing, many enterprises have started research and development open-source distributed databases, also called NoSQL, for it has lower implementation costs than traditional relational database, free-schema characteristics, and good execution efficiency on a cluster. The NoSQL technology enables the big data computing to support large-scale computing and huge-volume storage.

While the past research mostly studies the performance comparison of NoSQL, SQL, and OODB, little research has been devoted to the methods for evaluating the performance of NoSQL databases. This thesis proposes an application-oriented evaluating method. We assume that a better data model for a NoSQL database should be designed according to the behavior of a application. We will explore the performance of different data models using the same API (Application Performance Interface) in different volume of data. We studied the performance of different data models using our proposed evaluating method. By comparing the simulation and experimental results, we validate the effectiveness of the proposed evaluating method. The proposed method provide NoSQL database designers a way to evaluating the performance of a data model.

Keywords: Big Data、Clooud Computing、NoSQL

1. 前言

資料庫主要是將資料數據收集並組織成模型的方式儲存，並透過雲端運算的方式，將資訊被多個使用者共享使用。但隨著日漸俱增的資料量成為現成重要的研究議題，為了解決大量資料問題，許多企業紛紛轉而採用 NoSQL，而 NoSQL 泛指目前市面上的非關聯式資料庫，因這些資料庫大多不支援 SQL。目前已有許多企業採用 NoSQL 技術開發應用服務，相當成功，並且確實能夠成功解決巨量資料運算問題。NoSQL 資料庫的特性為(Schema-less)，因此缺乏類似正規化的技術，導致不同的設計者會設計出不同的聚合資料模型(Aggregate Data Models)，而不同的聚合資料模型往往在效能上有很大的差異。本研究將以文件式資料庫中的 MongoDB 為主要實驗對象，以觀察不同聚合資料模型的效能。

以往設計者只能在資料庫實際運作一段時間後，才能觀察出其效能優劣，本研究希望建構一套效能評估模式，作為將來資料庫設計者在不需實作的情況下，從多個聚合資料模型中選出效能相對較佳的聚合資料模型。

2. 資料庫相關文獻討論

2.1 關聯式資料庫

關聯式資料庫是建立在關聯模型基礎上的資料庫，將現實世界中的各種實體與實體之間的聯繫均用關聯模型來表示，而關聯式資料庫的資料儲存

結構為一個資料庫可以抱含一個或多個表(Table)，表是以行和列的形式組織而成的資料集合，每張資料表是由許多筆記錄(Record)所組成的，每筆記錄又以許多欄位(Field)組合而成的，每個欄位則存放一筆資料(Data)，且只能存放一筆資料。關聯表中的屬性可分為鍵值(Key)與非鍵值(Non-key)[1]。

正規化是在關聯式資料庫中組織資料的程序，包含建立資料表、資料表間的關聯關係，其目的是為了透過刪除重複性和不一致的相依性，以減少資料庫中資料冗餘，增進資料的一致性，使資料庫更具有彈性且能有效率的管理。關聯式資料模型的發明者 Edgar Frank Codd 提出了正規化的概念，並於 1970 年代初定義了第一正規化、第二正規化和第三正規化的概念 [2]。

一般資料庫管理系統為在寫入或異動資料過程中，確保資料是正確的，這個過程稱為交易(Transaction)，交易是指透過資料庫操作所組成的一個完整的邏輯過程，常見的例子像是：銀行轉帳，需從原帳戶扣款，並至入帳至目標帳戶，這兩項操作是不能拆開的，這樣的過程就稱為交易，然而交易必須具備 ACID 四個特性：原子性(Atomicity)、一致性(Consistency)、隔離性(Isolation Behavior)、持續性(Durability)。

2.2 NoSQL 資料庫

NoSQL 意指 Not Only SQL(不只是 SQL)，是對非關聯式資料庫的通稱，相較於關聯式資料庫，NoSQL 資料庫的設計不需經過正規化的過程，又稱為 No-schema 的設計，沒有固定表格(Table)結構，也無連接(Join)操作，而是採用聚集(Aggregate Data Model)導向，此項技術可使用較低成本而大量的設備來支撐巨量資料所 需要的儲存規模和分析處理能力，因此相較於關聯式資料庫擴充性較佳。

NoSQL 資料庫根據其儲存方式分成四類：文件(Document-oriented)、欄位(Column-oriented)、鍵-值(Key-value)、圖像(Graph)等。

一般而言，NoSQL 資料庫通常具有下面特性：

- 採用 Schema-less 的設計

使用關聯式資料庫時，必須先定義資料表的欄位，像是欄位大小、名稱、型態等，而 NoSQL 資料庫，不須先定義 Schema，更具有彈性 [3]。

- 採用 Aggregate Data Model 的設計

NoSQL 中的鍵值資料庫、文件式資料庫、列儲存資料庫這三種類型都具有聚合(Aggregate)的特性，在聚合中允許多值並且可以將任意數據存放在一個集合裡，當需要存取這些相關資料時，就能更快速的操作[3]。

- 資料分區、副本

NoSQL 資料庫具有一個重要的特性就是水平擴充的能力，可自動將以分片的方式來分割資料集至各節點，並在不同的節點間進行複製。因此只要增加新的伺服器節點，就可以不斷擴充資料庫[4]。

- Map-Reduce

Map-Reduce 是由 Google 提出的一個軟體架構，利用平行運算來處理問題。概念是使用大量機器執行「Map」和「Reduce」，在 Map-Reduce 中分為兩階段，第一階段執行工作是 map，輸入單一聚集，輸出多個鍵值對(key-value pair)，第二階段則是 reduce，主要是將第一階段輸出的結果中，相同鍵的值結合在一起[5]。

- CAP 定理

相較於關聯式資料庫所遵循的 ACID 交易原則，以確保資料的正確性，NoSQL 取而代之的是 CAP 定理，CAP 定理包含資料一致性、可用性、分區容錯性，但三者無法同時共存，僅能同時選擇兩者[6]。

2.2.1 MongoDB

MongoDB 是由 10gen 團隊於 2007 年創立，是基於分佈式文件存儲格式的資料庫，並介於關聯式資料庫與非關聯式資料庫之間的資料庫，並且是由 C++語言編寫[4]。主要用於解決大量查詢效率問題，根據官方文件顯示，它的資料結構非常鬆散且較具有彈性，是一種於 json 的 bson 格式，因此可以儲存較複雜的資料類型，並且可以實現大多數關聯式資料庫查詢的功能 [7]：

在資料的儲存模式為文件式儲存，一個 MongoDB 資料庫中可以有多個集合(Collection)，相當於關聯式資料庫的表格(Table) [7]，一個集合可能由多個文件(Document)，相當於關聯式資料庫中的記錄(Record)，請參考下圖：

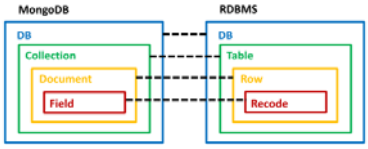


圖 1. MongoDB 與 RDBMS 對照圖

MongoDB 在資料的存儲是依照命名空間來劃分，一個 Collection 是一個命名空間，一個索引，也是一個命名空間，請參考下圖[8]。

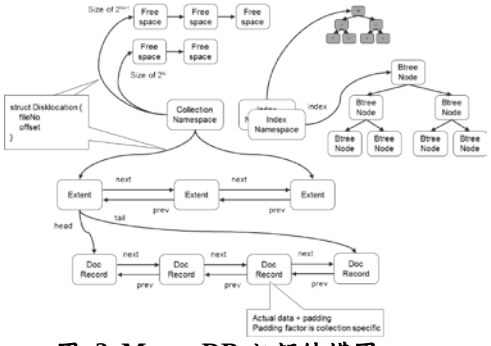


圖 2. MongoDB 內部結構圖

- 在同一個命名空間的資料被分成多個區間，區間之間使用雙向鏈節來連接。
- 每一個區間中，保存了具體每一行的資料，這些資料也是透過雙向鏈節來連接。

- 每一行資料存儲空間不僅包括資料占用空間，還包含一部分附加空間，使資料在更新後可以不更動儲存位置。
- 索引以 BTree 的結構實作

3. 建構 NoSQLAggregate Data Model 的效能評估方法

3.1 效能評估方法介紹

同一個案例(Application) 不管使用哪一種聚合資料模型(Aggregate Data Model)，都是透過同一組 Data Access API 去存取資料。Data Access API 根據 Application 所需要的功能來決定實際上需要存取哪些資料，但是存取資料的效能卻會因為採用不同的 Aggregate Data Models 而有所不同。

因此本研究將結合 API 的使用率，建立一套量化效能評估，相關參數定義如下：TE_j 為最終由 P_i 及 T(〖n)〗_(i,j)推導出的資料模型存取複雜度，請參考下列式子。

- Pi : APIi 被呼叫的機率
- T(n)i,j ： APIi 存取 Data Modelj 的時間複雜度
- TEj : Data Modelj 的存取複雜度

$$TE_j = \sum_{j=1}^{n} T(n)_{i,j} * P_i \quad (1)$$

$$AE_j = \frac{1}{n}\sum_{i=1}^{n} T(n)_{i,j} * P_i \quad (2)$$

$$SDE_j = \sqrt{\frac{1}{n-1}\sum_{j=1}^{n}(TE_j - AE_j)^2} \quad (3)$$

由於一般情況而言，資料庫設計者很難在系統實際運行前，估算 API 的使用頻率，然而有一套方式可用來估算 API 的使用頻率，就是 1949 年由哈佛大學 George Kingsley Zipf 提出的齊普夫定律(Zipf's Law)，齊普夫定律是一個實驗定律，是利用數理統計的實證法來制定的，公式如下：

- Pi : APIi 被呼叫的機率
- N：元件(API)的個數
- k ：元件(API)間的排序
- s：特徵分佈值

$$P_i = f(k; s, N) = \frac{1/k^s}{\sum_{n=1}^{N}(\frac{1}{n^s})} \quad (4)$$

$$\sum_{n=1}^{\infty} \frac{1}{n} = \infty \quad (5)$$

3.2 資料存取時間複雜度分析

具體來說，API 的實作會因為採用不同的聚合模型，導致程式碼產生不同的物件存取次數與物件存取時間。為得到物件存取次數 T(n)，須先根據應用程式訂定 API，接著從數支 API 中整理出使用到的物件，將這些物件組合成聚合資料模型，最後再針對各 API 去計算各模型的物件存取次數。假設一支 API 分別讀取 ABC 物件，可能依不同的排列順序形成下圖三種不同的聚合資料模型。

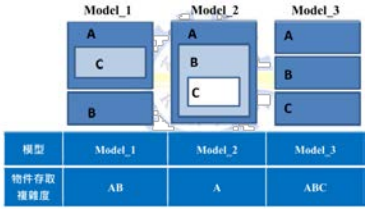


圖 2. 不同的 Aggregate Data Model 存取次數

3.3 資料存取時間複雜度模擬

由於市面上各種 NoSQL 資料庫都有各自不同的存儲資料、解析資料的方式，使資料庫的效能優化，前一節(3.2)的方法無法考慮實際使用 NoSQL 資料庫的實作方式，使得與資料庫實際執行結果有落差，本研究主要將探討文件式資料庫，並以其中的 MongoDB 為例。

由 2.2.1 可得知 MongoDB 中內部資料實際儲存方式，本研究將以 Pseudo code 來模擬資料庫中不同的聚合資料模型存取物件的方式， Pseudo cade(虛擬碼)是一種高層次描述演算法的一種方式，介於自然語言和高階程式語言之間，但比自然語言還簡潔，又比高階程式語言有較多可讀性，

- TTC_i：資料模型的總時間複雜度
- f_i：使用的資料模型
- API_j ：使用的 API
- n_i：使用的的資料物件

$$TTC = f_i, API_j(n_1, n_2 \dots) \quad (6)$$

4. 實驗設計與實作說明

4.1 資料存取時間複雜度模擬

在實驗設計的部分，主要分為三個階段，第一階段為系統功能、聚合資料模型設計的前置作業部分，第二階段為物件聚合時間複雜度分析法、Data Access Pseudo code 分析法兩種分析法的效能評估部分，第三階段則是透過實際程式撰寫實測模型的效能，最後再與前面的分析法作驗證與分析。

4.2 案例介紹

本研究的實驗案例將以民宿業業主所使用的 APP 為例，主要提供的功能以業主即使人在外，也能透過手機得知民宿的狀況，並以下列幾項功能作為實驗主要實作部分：

- 取得該民宿當月收入
- 取得該民宿當月住房率
- 取得該民宿銷售最佳的房間
- 該客戶的訂房記錄

根據上述案例，可歸納出下面會使用到的物件：Host、Hotel、Room、Customer、Reservation 等。依照這幾個物件，我們可以設計出不同的聚合資料模型，在本研究中以下面的三個模型為例，並將以此進行實驗。

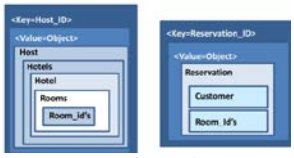


圖 3. A 聚合物件模型圖

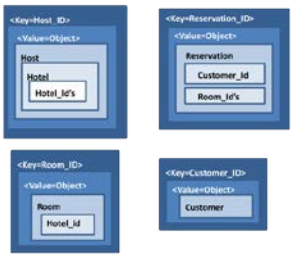


圖 4. B 聚合物件模型圖

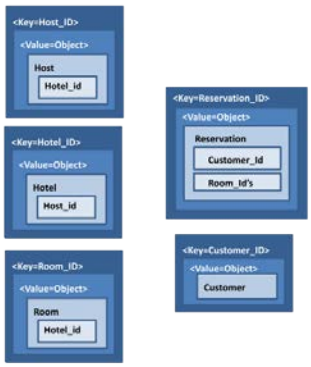


圖 5. C 聚合物件模型圖

4.3 實驗環境與工具

在實驗實作的部分，本研究將根據前述案例規畫數個功能，依據功能設計三種不同的資料模型，並分別寫入五萬 、十萬、一百萬的測試資料，再進行 30 次的測試。本研究的實驗環境如下：

表 1. 實驗環境與工具

作業系統	Windows 7 64-bit
處理器	Intel(R) Core(TM) i5-2415M CPU 2.30GHz
記憶體	4.00GB(DDR2)
資料庫	MongoDB 2.6.1
監控軟體	Robomongo 0.8.4

4.4 實驗結果討論

4.4.1 資料存取時間複雜度分析法

從 4.2 的案例中可得出下列使用到的物件：「Host、Hotel、Room、Customer、Reservation」，在此將這些物件取用英文縮寫的第一字作為符號表示如下：

O：民宿業者 (Number of Owner)
H：民宿 (Number of Hotel)
R：房間 (Number of Room)
A：預約記錄 (Number of Reservation)
C：客戶資料 (Number of Customer)

之後可以根據上述物件，以及採用 3.1 的物件聚合資料時間複雜度分析法整理出各模型於不同的 API 下的物件聚合資料時間複雜度，如下表所示：

表 2. 各模型物件聚合資料時間複雜度

API 名稱 \ Model		A	B	C
API1	該民宿當月收入	$6OA$	$2HRA$	$OHRA$
API2	該民宿住房率	$6OA$	$2HRA$	$OHRA$
API3	當月賣最好房間	$6OA$	$2HRA$	$OHRA$
API4	該客戶的訂房記錄	$2A$	A	A

4.4.2 資料存取時間複雜度模擬法

根據 4.4.1 前述的物件，以及採用 3.2 的 Data Access Pseudo Code 分析法整理出各模型於不同的 API 下的時間複雜度，以第一支 API 為例，欲讀取某民宿的總房間收入，其時間複雜度如下：

API	Model	Data Access Psedocade	Time Complexity	Simplified Time Complexity

API1	Model_A	for (i=0;i<host.length;i++){
		if host[i].host_name = "MARY" {
		if host[i].hotel_name ="BBhotel"
		find_room = host[i].room_id;
		}
		}
		}//讀取 host→room_id
		
		for (i=0;i<reservation.length;i++){
		for (j=0;j<room.length;j++){
		if reservation[i].reservation_room = find_room;
		
		sum_price=sum_price+i.reservation_price;
		}
		}
			O+AR	$N + N^2$

圖 7. 資料存取時間複雜度範例圖

其餘的資料存取時間複雜度結果整理如下表：

表 3. 各模型 Data Access Pseudo Code 分析法表

API	Model	Time Complexity	Simplified Time Complexity
API1	Model_A	O+AR	$N + N^2$
	Model_B	OR+AR	$2N^2$
	Model_C	OHR+AR	$N^2 + N^2$
API2	Model_A	O+AR	$N + N^2$
	Model_B	OR+AR	$2N^2$
	Model_C	OHR+AR	$N^2 + N^2$
API3	Model_A	O+AR	$N + N^2$
	Model_B	OR+AR	$2N^2$
	Model_C	OHR+AR	$N^2 + N^2$
API4	Model_A	A	N
	Model_B	A	N
	Model_C	A	N

4.4.2 實驗結果討論

為了得知本研究所提出的兩種評估方法是否與實際資料庫執行結果相同，在此作以同樣的實驗

運行 30 次後所得到的平均值，得到的結果如下：

表 4. 各模型實驗結果表

API	資料筆數	A	B	C
1	5 萬	54.6	141.3333	177.2333
1	10 萬	112.2667	292.9333	314
1	100 萬	871.3667	2703.2	3092.767
2	5 萬	79.1	135.5333	157.6667
2	10 萬	92.16667	274.9667	309.4667
2	100 萬	882.8	2701.267	3130
3	5 萬	46.33333	138.8333	158.5667
3	10 萬	93.36667	275.2667	315.7667
3	100 萬	875.8667	2746.8	3163.067
4	5 萬	1.233333	1.1	1
4	10 萬	1.166667	1.2	1
4	100 萬	1.066667	1.166667	1.033333

單位：ms

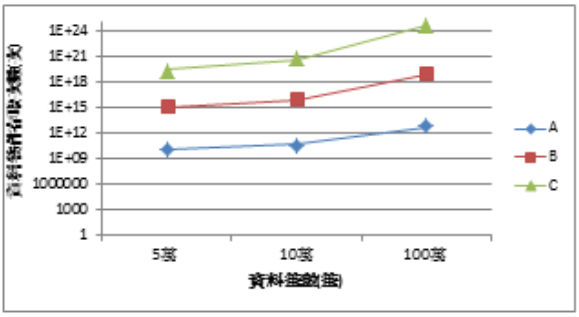
為了探討在不同的 API 使用頻率對資料庫的效能影響，在此本研究假設兩種情況，第一種情況為所有 API 被平均使用，第二種情況為根據齊普夫定律所訂定的使用頻率，如下表所示：

表 5. API 使用頻率表

API		齊普夫定律	100%	平均
API1	檢民宿當月收入	N	48%	25%
API2	檢民宿住房率	N/2	24%	25%
API3	當月負最好房間	N/3	18%	25%
API4	檢客戶的訂房記錄	N/4	12%	25%

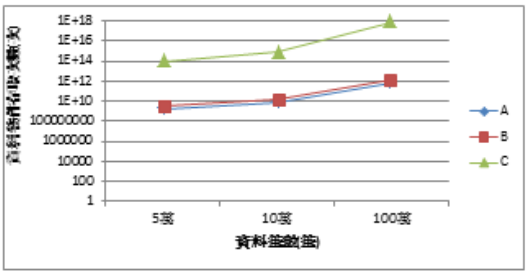
1. 情況一：API 被平均使用

(1). 資料存取時間複雜度分析法



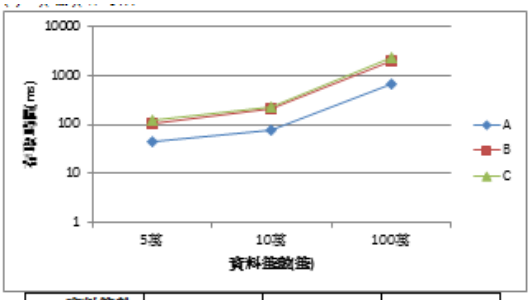
資料筆數	A	B	C
5 萬	1.13×10^{10}	9.38×10^{16}	2.34×10^{19}
10 萬	4.5×10^{10}	7.5×10^{15}	3.75×10^{20}
100 萬	4.5×10^{12}	7.5×10^{18}	3.75×10^{24}

(2). 資料存取時間複雜度模擬法



資料筆數	A	B	C
5 萬	1.88×10^{9}	3.75×10^{9}	9.38×10^{12}
10 萬	7.5×10^{9}	1.5×10^{10}	7.5×10^{14}
100 萬	7.5×10^{11}	1.5×10^{12}	7.5×10^{17}

(3). 實驗實作結果



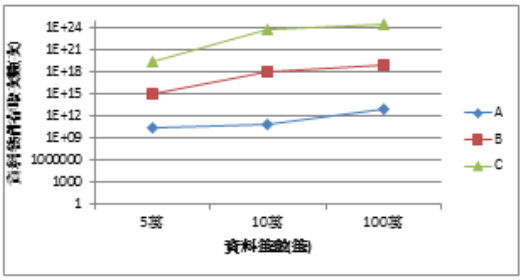
資料筆數	A	B	C
5 萬	45.32	104.20	123.62
10 萬	74.74	211.09	235.06
100 萬	657.78	2038.11	2346.72

根據情況一，在 API 分別平均被使用時，我們在資料庫中進行實證，得到上面結果，無論資料筆數多寡，A 資料聚合模型的效能都是最好的，B 模型為次之，而 C 模型為相對效能最差的。並且觀察到，在資料筆數為五萬時，C 模型的查詢時間大約為 A 模型的三倍，當資料筆數增加至一百萬時，C 模型的查詢時間大約為 A 模型的四倍，顯示在資料筆數快速增加時，C 模型所花費的查詢時間增加得的比 A 模型還來得快。

結論：根據上面三張圖表可得知，在 API 平均被使用情況下，無論是本研究所提出的兩種方法與實際實驗結果，在 5 萬、10 萬、100 萬筆資料時，都是 A 模型的效能較佳。

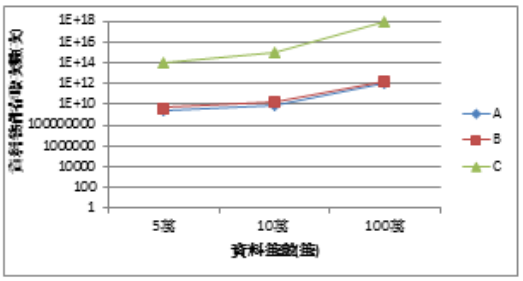
2. 情況二：根據 Zipf's Law 的 API 使用率

(1). 資料存取時間複雜度分析法



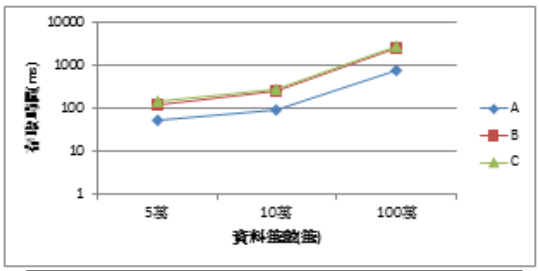
資料筆數	A	B	C
5 萬	2.49×10^{10}	1.13×10^{15}	2.81×10^{19}
10 萬	9.96×10^{10}	1.21×10^{16}	6×10^{23}
100 萬	9.96×10^{12}	9×10^{18}	4.5×10^{26}

(2). 資料存取時間複雜度模擬法



資料筆數	A	B	C
5 萬	2.25×10^{9}	4.5×10^{9}	1.13×10^{16}
10 萬	9×10^{9}	1.8×10^{10}	9×10^{16}
100 萬	9×10^{11}	1.8×10^{12}	9×10^{17}

(3). 實驗實作結果



資料筆數	A	B	C
5 萬	53.68	125.49	151.57
10 萬	92.95	256.29	281.95
100 萬	787.91	2440.40	2805.20

根據情況二，在 API 根據 Zipf's Law 被使用時，我們在資料庫中進行實證，得到上面結果，無論資料筆數多寡，A 資料聚合模型的效能都是最好的，B 模型為次之，而 C 模型為相對效能最差的。並且觀察到，在資料筆數為五萬時，C 模型的查詢時間大約為 A 模型的三倍，當資料筆數增加至一百萬時，C 模型的查詢時間大約為 A 模型的四倍，顯示在資料筆數快速增加時，C 模型所花費的查詢時間增加得的比 A 模型還來得快。

根據上面實驗結果可得知，在 API 的使用率依照齊普夫定律所算出的機率下，無論是本研究所提出的兩種分析法與實際實驗結果，無論資料筆數多寡，都是 A 模型的效能較佳，但物件聚合資料時間複雜度分析法(圖表一)中的 B、C 模型是呈現凸曲線，與實際實驗結果的凹曲線不同，而 Data Access Pseudo Code 分析法的曲線與實驗結果較吻合，因此可得出以下結論：

1.無論使用哪種方法，皆得出 A 模型優於 B 模型優於 C 模型。

2.結合齊普夫定律使得兩種分析法與實際實驗的結果更明顯。

3.資料存取時間複雜度模擬法評估的準確度是比資料存取時間複雜度分析法來得較高的。

5. 結論與建議

本研究將應用程式導向的概念導入設計資料庫，作為資料庫設計的參考依據，提出一套判斷 NoSQL 資料庫效能的評估方法，並建立評估的步驟，並在 MongoDB 的環境下，分別以五萬、十萬、一百萬的測試資料進行實驗，

藉由前面章節的實驗測試過程，採用了兩種方法以及實際實驗，分別在不同的資料筆數中進行測試，經過比對後，無論情況一：API 平均被使用、或是情況二：API 依齊普夫定律得出的機率使用，發現兩種方法與實驗結果得到的效能為 A 模型優於 B 模型優於 C 模型，這三個模型之間的關鍵差異為聚合物件的個數，由此我們可觀察出幾個結果，"資料存取時間複雜度分析法"雖然較不精準，但仍可得到與實驗結果相當接近的趨勢，而資料存取時間複雜度模擬法"可以讓資料庫設計者於短時間內，正確選擇出幾個模型內效能相對較佳的資料模型，由以上的實驗結果，我們另外可得出下面這項結論，應根據應用程式最常使用的 API 所存取到的多個物件組合成一個聚合物件，如此一來效能較佳。本研究的主要目的為提出 NoSQL 效能評估法，提供為未來資料庫設計者，在不需實際撰寫程式以及資料庫實際運行一段時間的情況下，可供設計 NoSQL 資料庫時的參考，雖然資料存取時間複雜度模擬法的步驟較為繁複，但根據實驗結果，資料存取時間複雜度模擬法的結果較為準確，因此建議設計者可以採用此種方法作為設計參考。

NoSQL 本來應用在分散式環境下，將資料分散至不同節點主機，達到規模經濟，增加運算效能，由於本研究主要探討聚合資料模型的效能分析，因此只在相同的本機端作測試，以免網路速度或 NoSQL 的自動分片、備份機制影響結果。另外因 NoSQL 的出現是為了解決加速回應大量查詢的問題，因此在本研究中案例只使用查詢功能的 API 進行測試。由於各種資料庫都有各自不同設計的索引功能，以優化查詢的速度，為了使實驗數據更明顯具不被干擾，在此不使用官方提供的索引方式改以自行手動查詢。未來的研究方向，可朝向 NoSQL 聚合資料模型的儲存成本計算，因加快了資料庫的執行速度，相對的就是是犧牲了一定的儲存成本，在企業中的應用除了得知資料庫的執行效能，也應當了解可能花費的儲存成本，又或者可以進一步探討 NoSQL 效能最佳化模型的設計準則，建議未來研究相關議題者可參考上述的方向。

參考文獻

[1] 曾守正, 1995, "資料庫系統的回顧與未來研究發展", 中華民國資訊學會會刊, 1 卷, 1 期, 頁 9-31

[2] Wikipedia. "Relational Database", http://en.wikipedia.org/wiki/Relational_database.

[3] Stydytonight. Normalization of Database. 2013, http://www.studytonight.com/dbms/database-normalization.php

[4] Pramod J. Sadalage, Martin Fowler, 2012, NoSQL Distilled:A Brief Guide to the Emerging World of Polyglot Persistence, Addison-Wesley Professional.

[5] NoSQL, http://nosql-database.org/.

[6] Tom White, 2012, *Hadoop: The Definitive Guide*, O'Reilly Media / Yahoo Press

[7] Wikipedia, "CAP theorem", https://en.wikipedia.org/wiki/CAP_theorem.

[8] MongoDB, http://www.mongodb.org/.