NoSQL 雲端資料庫資料聚合模型設計之研究

王致超[12], 丁建文[1]*
[1]國立高雄應用科技大學資訊管理系
[2]資訊工業策進會
E-mail: jichaowang27@gmail.com, *jwding@kuas.edu.tw

摘要

隨著巨量資料以及雲端運算的盛行，傳統的關聯式資料庫在運算上的效能負荷越來越重，對企業來說，為了避免影響到其他企業應用程式的運行，只能額外再新增資料庫去減少原資料庫的負擔；關聯式資料庫在定義 Data Model 時，需要進行資料庫的正規化，幫助開發者確認 Data Model 的正確性以及將 Schema 定義出來。關聯性資料庫藉由 SQL 可以幫助資料聚合以及有效的控制資料的完整性。

NoSQL 的 Data Model 是聚合導向的資料模型。在設計時，開發人員可自由定義 Schema，也造成同一個系統的 Data Model 也會因為開發者不同的觀點而產生差異。而現在在 NoSQL 的開發上，尚未有人提出完整 Data Model 的設計方法，因此本研究欲探討哪種 Data Model 的設計方式適用於 NoSQL 的資料查詢，當資料與資料之間具有複雜的關係時，該怎麼設計才能使執行的效能會有所提升。

關鍵字:巨量資料、資料庫、雲端運算、資料模型、聚合模型

Abstract

With the growing popularity of big data and cloud computing, the loading on traditional relational database is heavy. To avoid affecting others applications in an enterprise, the common approach is to add additional databases to reduce the loading of the original databases. A relational database needs normalization in defining a data model (i.e., schema), which helps developers confirm the correctness of the defined data model. Relational database can effectively control the integrity and aggregation of the queried data by using SQL(Structured Query Language).

NoSQL database is an aggregate-oriented data model. During the design phase, developers must know what is the needed information for the system, and then they design the data model according to the needed information. It is still not clear how to define a well-performed data model for a NoSQL database, especially for query commands. In view of this, this thesis investigates how to design a well-performed data model for NoSQL databases.

Keywords： NoSQL、Big Data、Data Model、Aggregate Model、Cloud computing

1. 前言

過去，使用者在使用關聯式資料庫進行開發時，除了定義資料的 Schema 外，還會設計 ERD，除了幫助開發人員進行需求的分析外，也可以釐清資料與資料之間的關係；也因為關聯式資料庫有正規化的步驟，可確保資料庫的設計正確，讓資料庫運作的效能提高，並且避免資料 Schema 的設計錯誤。

NoSQL 資料庫由於是 no-Schema(不必預先定義資料格式、欄位)，不同的使用者在設計 Data Model 時，也容易因為沒有標準再而設計出不同的 Data Model，不一樣的 Data Model 甚至會影響到資料庫對資料的存取效能。

NoSQL 資料庫由於發展的時間較晚，因此不像關聯式資料庫一樣，有一套專門設計、規劃資料庫 Data Model(ERD) 的方法；本研究將提出 Aggregate Model 的設計準則，將會根據所提出的設計準則去設計一個 Model 並與其他設計出的 Model 作比較，證明本研究所提出的設計方法。

2. 相關文獻探討

2.1 NoSQL 資料庫

1998 年，Carlo Strozzi 就已經提出 NoSQL 資料庫的概念[1]，但那時候並沒有被廣泛的使用。2009 年，Eric Evans 重新提出了 NoSQL 的概念[1]。NoSQL 也是「Not Only SQL」的縮寫，與傳統的關聯式資料庫有很大的差別，其中就是沒有了 SQL 查詢語言，也避免了 join 的操作，每一筆資料之間是沒有關聯性的，不像關聯式資料庫一樣需要固定的 schema，因此在擴充上是很方便的，直接增加伺服器節點就可以不斷擴充資料庫容量，讓使用者依照自身的需要而隨時增加或減少所需要的儲存空間，企業也能夠根據自身的需求來對資料庫進行擴充。

2.2 關聯式資料庫不適用於巨量資料

在過去，關聯式資料庫是企業的首選，關聯式資料庫除了提供規範來幫助開發者在初步規劃及設計資料庫外，也提供 ACID 原則，確保資料的完整性以及資料庫的 Transaction 能夠正確的執行[7]。由於嚴格的規範使關聯式資料庫不適合應用在 Big Data 的儲存上，對於 Big Data 來講，一個高可擴充性(水平擴充)、靈活性高的儲存架構也是非常重要的；而關聯式資料庫在擴充能力上由於受到 Schema 的影響，也無法馬上進行資料庫的調整；相較而言，NoSQL 資料庫就更適用在 Big Data 上，NoSQL 資料庫原本的設計就是分散式的儲存架構，因此當需要增加記憶體，可以直接新增伺服器節點，原本的資料庫也可以正常的運作；除此之外，對於 Big Data 最重要的就是資料儲存的空間及大量資料處裡的運行速度，而 NoSQL 資料庫由於沒有了 SQL 查詢，在執行效能上比起關聯式資料庫來講也提高了許多。

3. 資料模型建置

3.1 資料的聚合

NoSQL 資料庫由於是 no-schema，也就是不用預先定義資料的格式，讓使用者可依照自身需求去做更改，並且允許使用者同時存取“多值”，甚至是 list(document)，這種方式也被稱作“Aggregate Orientation“。

3.2 案例介紹

每次舉辦會展，會展公司就需要去找來參展的參展商，每位參展商又提供多樣參展商品;參展商的商品種類又是非常多變的，會展開始前，無法將商品欄位記錄完全，需求是非常不明確的，甚至在之後會一直對於資料庫做更正、修改，這樣對於初步資料庫的規劃會產生問題；另一方面，每次舉辦會展參加的人數也都是非常大量的，需要收集的資料量也很大，對於伺服器端的流量控管也是需要考量的問題。底下也將使用此案例來發展 NoSQL 雲端資料庫的 Aggregate Model 設計法則。

3.3 Aggregate Model 設計

在 NoSQL Data Model 中，Aggregate Model 是由 Aggregate 組成，也就是說，一個 Aggregate Model 是由多個 Aggregate 組成的。將不同的 Aggregate 放置在最上層，對於查詢以及寫入的效能的影響都會有明顯的差別，而本研究將建立 NoSQL Data Model 的過程分成了 4 個步驟:

圖 1. 建立 Aggregate Model 步驟

3.3.1 定義資料的聚合

本研究以會展產業來當例子，將功能整理後，此案例所使用到的資料物件統整如下表:

表 1. 案例資料物件定義表

物件名稱	物件描述
會員	紀錄有安裝 App 的民眾個人資訊
電話	會員電話
商品訂單	訂購商品的紀錄
商品訂購項目	訂購商品的品項
會展	會展的相關資訊
參展商	參加會展的參展商資訊
商品	會展及參展商商品
購票明細	購買門票的紀錄
門票	會展門票

3.3.2 找出物件之間的關係

由於 NoSQL 資料庫較常被使用在巨量資料的儲存，因此對於每個物件之間的關係需要謹慎的評估，以免造成查詢資料時對效能的影響。NoSQL 資料庫中，物件與物件之間的關係，會使用”繼承”來表示，而在設計 Aggregate 時，裡面會有多個物件，物件之間的關係表達會用繼承來表示，因此一個 Aggregate 底下會包含一個物件，甚至是包含多個物件。

3.3.3 Aggregate 定義

在 NoSQL 資料庫裡，Aggregate Model 裡面會存在許多的 Aggregate，而在 NoSQL 資料庫裏面允許儲存多值的型態，所以找出 Aggregate 之間的關係是非常重要的，這會影響使用者在讀取資料時的速度。

在 3.3.1 中，有敘述本論文所使用案例的物件，而本案例系統主要分成會員、會展、商品訂單、購票紀錄這四個角度去使用資料，因此分成了四個群組:

- 會員、電話
- 商品訂單、商品
- 會展、參展商、商品、門票
- 購票紀錄、門票

本研究主要有 4 個 Aggregate:會員、會展、商品訂單、購票紀錄。

3.3.4 Aggregate Model 設計

由於在 NoSQL 資料庫沒有一個固定的方法讓使用者設計 Aggregate Model，而在一個系統中，不同的 Model 會有不同的效能影響。而本案例是藉由 API 向後台傳接收資料，因此經由後端提供的 Web Service API 的規格去導出 Aggregate Model，可以減少不必要的資料欄位設計。

1. 設計準則：本研究提出以下的設計準則。

表 2. 設計準則名詞定義

agg	資料聚合
agg_{main}	Model 最上層的aggregate
R	Aggregate 之間的關係(Relation)
obj	Aggregate 的資料物件(object)
topobj	Aggregate 的頂層物件(top object)
level n	Aggregate 在 Model 內的資料查詢層級，當 n 越小則表示與越接近 $aggregate_{main}$(level 0=top)

API 是透過多個 Aggregate 以及這些 Aggregate 的關係所組成的，因此本研究將 API 轉換成的公式，再透過每個 API 去找 Aggregate 間的關係:

$$api = [agg_x + agg_y + agg_z \dots] + [R_{topobjx-topobjy} + R_{objy-objz} + \cdots]$$

$R_{topobjx-topobjy}$ 代表此關係是上層物件 x 與上層物件 y 所產生的

$R_{objy-objz}$ 代表此關係是透過資料物件 y 與資料物件 z 產生

2. 設計準則一：

先針對每個 api 探討所使用到的 Aggregate 以及這些 Aggregate 之間的關係(R)，將與其他 Aggregate 產生最多 $R_{topobjx-topobjy}$ 關係數量的 Aggregate 放置在 Aggregate Model 最上層(agg_{main})。

3. 設計準則二：

根據第一準則所找出的 api，比較 api 內的 Aggregate 以及 R，若 api 內的 R 包含有 $R_{objy-objz}$，比較agg_y 與agg_z的 topobj:

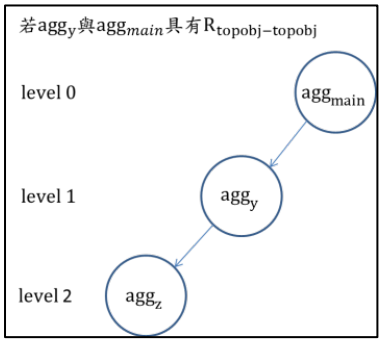

圖 2. 設計準則二(2-1)

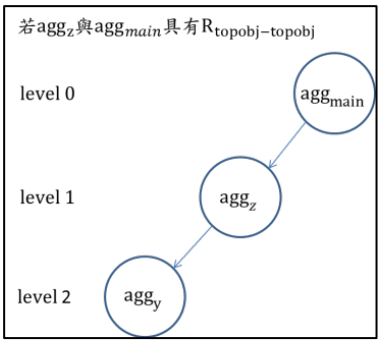

圖 3. 設計準則二(2-2)

假如與agg_{main}的 topobj 沒有$R_{topobj-topobj}$關係，比較與agg_y或agg_z的 topobj 有產生$R_{topobj-topobj}$關係的agg:

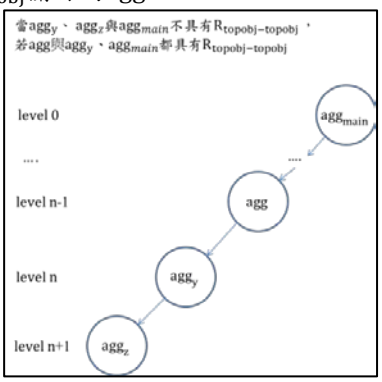

圖 4. 設計準則二(2-3)

反之則:

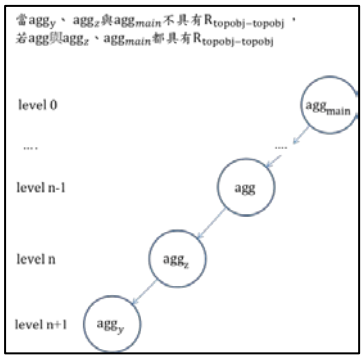

圖 5. 設計準則二(2-4)

4. 設計準則三：

“物件 A 與物件 B 是一對多關係，讓物件 B 記錄物件 A 的 ID(參照 ID)，不要在物件 A 紀錄物件 B 的 ID”(減少 Aggregate A 的使用頻率)：

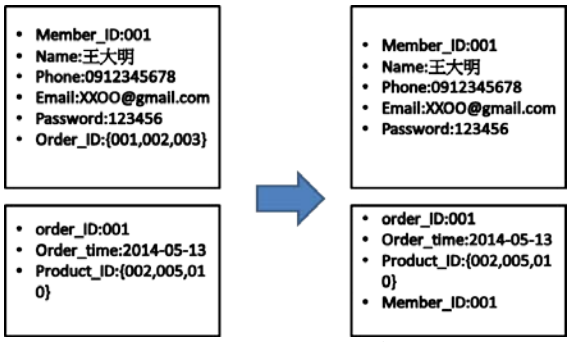

圖 6. 設計準則三 範例

透過上述的設計準則，本研究將根據設計準則，產生出 Aggregate Model A，另外設計出其他二種 Aggregate Model(B、C)進行驗證比較。

5. Aggregate Model：

本研究根據所提出的設計準則進行設計，因而設計出 Model A。Model B、C 則是將其他的 Aggregate 放置不同位置進行驗證比較。

6. Aggregate Model A：

底下將使用設計準則帶入本研究案例，將 Aggregate Model A 設計出來。

根據第一個設計準則去觀察 API，將使用到一個以上 Aggregate 數量的 API 進行比較，而主要進行比較 API 的有 7 支 API:

表 3. 設計準則一 API 列表

api_1:查詢會員參加過的會展
api_2:查詢參加會展的會員
api_3:查詢會展所賣出的商品
api_4:查詢會員所購買過的商品
api_5:各年齡層的會員參加會展類型
api_6:一年以內，會員參加過幾次會展
api_7:參加會展的會員所來自地方的比例

第 1 支 API 是查詢會員所參加過的會展。將 API 轉成公式，如下:

$$api_1 = [agg_{member} + agg_{t\_order} + agg_{exhibition}] + [R_{topobj_{member}-topobj_{t_{order}}} + R_{obj_{ticket}-obj_{ticket}}] \quad (1)$$

會員會直接與購票紀錄的 topobj 產生 $R_{topobj-topobj}$ 關係。

第 2 支 API 一樣是使用到會員、購票紀錄與會展這三個 Aggregate，因此公式為:

$$api_2 = [agg_{member} + agg_{t\_order} + agg_{exhibition}] + [R_{topobj_{member}-topobj_{t_{order}}} + R_{obj_{ticket}-obj_{ticket}}] \quad (2)$$

購票紀錄與會員的產生 $R_{topobj-topobj}$ 關係。

第 3 支 API 則是查詢會展所賣出的商品。將 API 轉成公式，如下:

$$api_3 = [agg_{p\_order} + agg_{exhibition}] + [R_{obj_{product}-obj_{product}}] \quad (3)$$

透過公式可以看出會展並不會與商品訂單的 topobj 產生 $R_{topobj-topobj}$ 關係。

第 4 支 API 是去查詢會員所購買過的商品。將 API 轉成公式，如下:

$$api_4 = [agg_{member} + agg_{p\_order}] + [R_{topobj_{member}-topobj_{p_{order}}}] \quad (4)$$

透過 API4 的公式可以發現到，會員的 topobj 會直接與商品訂單產生 $R_{topobj-topobj}$ 關係。

而第 5~7 支 API 也同樣都使用到會員、會展、購票紀錄這三個 Aggregate，而在前面已經比較過，統計如下表:

表 4. 設計準則一_關係統計表

Aggregate	$R_{topobj-topobj}$ 數量	關係 Aggregate
會員	2	購票紀錄、商品訂單
購票紀錄	1	會員
商品訂單	1	會員
會展	0	無

透過表 5，可以觀察到 Aggregate_會員產生最多的 $R_{topobj-topobj}$ 關係，因此本研究將其擺放在 Model A 的最上層(agg_{main})。

第二個設計準則要去觀察 API 內含有 $R_{obj-obj}$ 關係的 Aggregate，而第一個設計準則的時候，已經將 API 轉成公式了，透過公式所觀察到具有 $R_{objy-objz}$ 關係的 API 有 6 支，會探討到的 API 有:

表 5. 設計準則二_API 列表

api_1:查詢會員參加過的會展
api_2:查詢參加會展的會員
api_3:查詢會展所賣出的商品
~~**api_4**:查詢會員所購買過的商品~~
api_5:各年齡層的會員參加會展類型
api_6:一年以內，會員參加過幾次會展
api_7:參加會展的會員所來自地方的比例

第 1 支與第 2 支 API 的 $R_{obj-obj}$ 關係是由購票

紀錄與會展所產生的，因此比較購票紀錄以及會展的 topobj，將購票紀錄與會展之間的關係看成:

agg_{t_order}為level 1，$agg_{exhibition}$為level 2，則資料的查詢可以表示成:

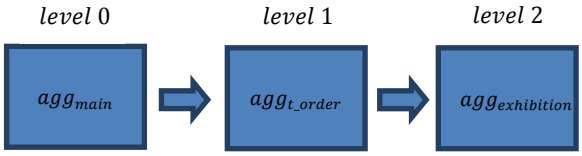

圖 7. 設計準則二 資料層級(1)

第 3 支 API 則是觀察商品訂單與會展的關係，透過公式可以看出這兩個 Aggregate 之間是產生$R_{obj-obj}$關係，因此去觀察它們的 topobj；在api_4會找出商品訂單與agg_{main}有產生$R_{topobj-topobj}$的關係，所以將商品訂單與會展此兩個 Aggregate 之間的關係看成:

agg_{p_order}為$level\ 1$，$agg_{exhibition}$為$level\ 2$，則資料的查詢可以表示成:

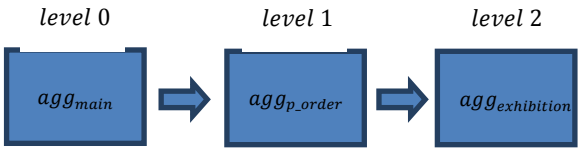

圖 8. 設計準則二 資料層級(2)

也就是說，當需要查詢有關會員在會展所購買的商品時，中間會藉由商品訂單查詢。

最後將前面三個設計準則所描述 4 個 Aggregate 間的關係可以組合成下圖(Model A):

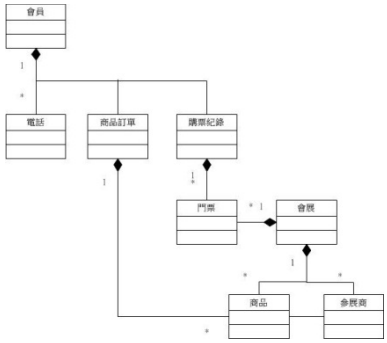

圖 9. Aggregate Model A

Model A 是由會員這個 Aggregate 當作最上層的 Aggregate。

7. Aggregate Model B：

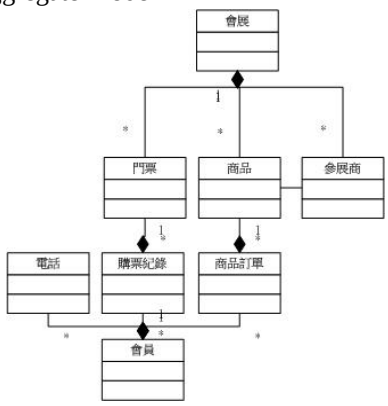

圖 10. Aggregate Model B

Model B 為了與 Model A 做出反差以及驗證本研究，因此採用違反本研究所提出的設計準則去設計 Model；由會展作為最上層的 Aggregate 作為本研究的對照 Model。

8. Aggregate Model C：

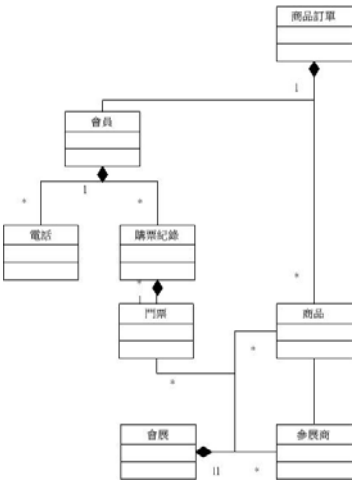

圖 11. Aggregate Model C

在前面章節有探討過，由於現在沒有一定的設計方法，因此不同人可能會設計出不同的 Model，所以本研究提出第三種 Model 作為設計準則之驗證；Model C 是由商品訂單當作最上層。

4. 資料模型效能驗證

4.1 時間複雜度

在 NoSQL 資料庫中，由於沒有 SQL 語法來幫助資料查詢，在資料物件的對應查詢上需要使用者自行控制、處理，時間複雜度則可以用來計算程式的執行次數、時間；因此本研究採用時間複雜度來預測 3 個 Model 的資料查詢效能。

時間複雜度函數:

(1) 會員:m (2) 商品訂單:o
(3) 購票紀錄:r (4) 會展:e
(5) 商品:p (6) 門票:t

表 6. API 時間複雜度

API	Model A	Model B	Model C
API 1	to*t	e*t*r	m*r*t
API 2	to*t	e*t*r	m*r*t
API 3	o*p*e	e*p*o	o*p*e
API 4	o	e*p*o*m	o*m
API 5	m*r*t*e	e*t*r*m	o*m*r*t*e
API 6	m*r*t*e	e*t*r*m	o*m*r*t*e
API 7	m*r*t*e	e*t*r*m	o*m*r*t*e

表 6 中，比較起來在不同的 Model 中，在時間複雜度上也有明顯的差別，本研究將時間複雜度用不同的參數帶入，可以看到 API 與各參數間的關係，在 Model A 中，時間複雜度最高的是 4 個參數產生相乘的關係；Model B 中，最高也有到達 4 個參數產生關係；Model C 中，最高則到達五個參數。

4.2 實驗結果

在實驗方面，本研究是採用 Amazon dynamodb 作為 NoSQL 資料庫測試，將每支 API 都測試過 30 次，最後取平均值。

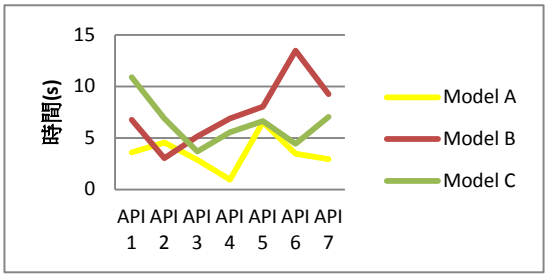

圖 12. 效能驗證統計圖

從上圖可以看出 API 在不同 Model 中的執行時間，在 3 個 Model 中，相較於其他兩個 Model 的 API，根據本研究設計準則所設計出的 Model A，在 API 的執行時間上是最短的；違反本研究設計準則所設計出的 Model B，透過 API 存取資料則是花費最多的時間，若是資料量增加時，Model B 的執行時間也會大大的增加。

5. 結論

本研究根據所提出的設計準則去設計出一個 Aggregate Model A，用違反設計準則的方式去設計出兩個 Model，再以時間複雜度預估效能以及實驗去測量這三個 Aggregate Model 的 API 讀取資料的時間，每個 Model 的每支 API 都實際測量 30 次後再取平均時間，最後比較三個 Model 讀取資料所花費的時間；最後結果則是透過本研究提出的設計準則所設計出的 Model A 比起其他兩個 Model 讀取資料所花費的時間是最少的、效能最佳的，完全違反設計原則所設計出的 Model B 則是花費最多時間的。

參考文獻

[1] Wiki-NoSQL: http://zh.wikipedia.org/wiki/NoSQL

[2] Eric Evans, *Domain-Driven Design* (Boston:Addison-Wesley,2004).

[3] Pramod J.Sadalage and Martin Fowler , *NoSQL Distilled: A Brief Guide to the Emerging World of Polyglot Persistence* (Boston:Addison-Wesley,2012).

[4] Shalom Nati, *No to SQL? Anti-database movement gains steam – My Take* , July 2009.

[5] J.Han, E.Haihong , G.Le. “Survey on NoSQL Database.” , *In IEEE Congress on Pervasive Computing and Applications (ICPCA)*, pages 363–366, 2011.

[6] Robin Hecht and Stefan Jablonski , ”NoSQL Evaluation: A Use Case Oriented Survey.” *In IEEE Cloud and Service Computing (CSC)* , pages 336 -341, 2011.

[7] wiki-ACID: http://zh.wikipedia.org/wiki/ACID