國立政治大學資訊管理研究所

碩士學位論文

指導教授：劉文卿博士

利用 JavaScript Application Framework 與 CouchDB 實作協作雲機制

——以盈餘預測系統為例

研究生：楊凭哲撰

中華民國一百零一年七月

致謝

在碩士班的兩年終於順利完成論文，首先誠摯地感謝指導教授劉文卿老師，讓我在這兩年的時間學習到很多，不論是知識學問或是做人做事的道理。老師的耐心細心使我印象深刻，每當有新的技術老師更是願意用更多的時間陪著我們學習，真的很榮幸在碩士班兩年能有這麼用心的老師帶領。

再來要感謝我的家人，父母親與兩位哥哥，有你們的支持與鼓勵我才能無後顧之憂完成我的學業，從出生以來就享受著你們無微不至的照顧。如今我已經完成學業，往後必定付出最大的心力與努力給我親愛的家人。

感謝研究所的好朋友們，在學業方面互相討論幫助，一起玩樂一起奮鬥，碰到挫折時還能互相鼓勵，很高興在碩士班兩年能交到如此知心的好朋友們。

還要感謝知識管理實驗室的所有人，尤其是勁超與柏翰，碰到問題時互相討論，感謝一路走來能有你們一起陪伴，更高興的是我們都能順利完成論文。另外還要感謝知識管理實驗室的學弟妹，你們常常能讓我從緊張的氣氛中緩和情緒，有你們在的時候實驗室就充滿著歡笑聲，還會邀情我一起參與紓壓的活動，真的很開心能有你們這群可愛的學弟妹。

最後要感謝另外兩個口試委員老師，楊建民老師與陳奕光老師，謝謝兩位老師抽空來參加論文學位口試，並且給我許多關於研究方面的建議，這些都使我獲益良多。除此之外，還要感謝所有關心我、陪伴我的人，這兩年來碰到挫折、心情不好時多虧你們在我身邊加油打氣，這些都給我很大的力量，謝謝。

摘要

本研究主要是想要基於協作雲概念下實作一個系統，並且有離線儲存、同步技術、版本控制等功能，因為 CouchDB 的特性有自動同步與離線儲存技術，符合協同合作的要求，因此選用 CouchDB 當作系統實作資料庫的選擇，另外也針對 CouchDB 的同步與轉換機制做了詳細的討論。而在系統程式語言方面，為了達到離線儲存的需求，簡單前後端語言溝通的負擔，選用了 JavaScript Application Framework，採用的是純粹的 JavaScript 語言，客戶端用了 jQuery、YUI3 等 JavaScript，伺服器端使用的是 Node.js，達成了一種程式語言，兩個執行時間的條件。確定系統框架與資料庫之後，最後使用盈餘預測系統來做為此實作之範例。

關鍵字：協作雲、CouchDB、JavaScript、Node.js、離線儲存、盈餘預測

表目錄

圖目錄

目錄

壹、 緒論

一、 研究背景與動機

資料同步一直是很重要的課題，如果資料庫裡面的資料無法正常的同步，會造成怎樣的後果是可想而知的，今天不論我更新了多少資料，下次又得從頭開始，更遑論其他使用者能得到我的資料。早在 1981 年，Philip A. Bernstein 與 Nathan Goodman 就做過這樣的研究，如何在分散式的資料庫系統中達到同步控制的機制，此研究設計了許多關於同步控制機制的演算法框架，來達到同步控制。

而現今非常熱門的話題即雲端，強調 everything as a service，因此就出現了新的困難點，如果加上雲端，那麼同步控制機制是不是又會更加困難，抑或是可以達到什麼更高的層次呢？

隨著網路上面的資料量越來越大，在 2002 年 Raymond K. Wong 跟 Niocle Lam 認為使用者會想要查詢到之前版本的資料，因此提出針對 XML 資料做的版本管理系統，可以有效地管理及查詢到這些有意義的資料。此外，版本控制可以讓你電腦系統維持最新的狀態，2010 年一份研究由 Ruowen Wang 跟 Vasanth Bala 提出了一個新穎的工具叫做 Niiwa，他分析版本並儲存起來，並且使得程式在離線時能夠繼續改寫，而此工具 Niiwa 也實際應用在 IBM Research Compute Cloud(RC2) 上面。

而近三年雲端的概念更是沸沸揚揚，在協作雲這個部分更是容易被提出來討論，如何在雲端上面建立協作平台、平台如何管理以及資料如何同步都是非常困難的問題，學校建立 e 化平台不遺餘力，使用網路服務的技術來建立一個使用者介面提供使用者整合以及溝通，所以 e 化社群是一個高度協同合作的環境，不論是 E-Learning Computational Cloud(eLC2)：Web Services Platform to Enhance

Task Collaboration(Sidhant Rajam , Ruth Cortez , Alexander Vazheninm , Subhash Bhalla，2010)或是 Private cloud for collaboration and e-learning services：from Iaas to Saas(Frank Doelitzscher , Anthony Sulistio , Christoph Reich , Hendrik kuijs , David Wolf，2010)都是強調建立一個協作雲能達到怎樣的目的。然而，網路世界並不是無所不在的，總是會有斷線或是沒有網路的時候，因此能夠離線作業也變得相當重要，如何離線工作並且能在上面時同步也變成相當重要的議題。

在這方面，現在廣泛應用的技術工具有 Google 雲端硬碟以及 Dropbox。Google 雲端硬碟其實是很新穎的技術，結合了網路硬碟以及 Google 文件，以往 Google 文件只能線上編輯，因此如何協作與同步就變成相當重要，2006 年，Stijn Dekeyser 和 Richard Watson 研究了 Google 文件與其他協作軟體在學者撰寫文獻輔助軟體的表現，點出其他軟體在同步版本的控制方面偏弱，因此 Google 文件是比較好的選擇，再之後，Google 文件的使用率節節攀升。今年 Google 文件結合網路硬碟，Google 推出雲端硬碟的服務，加入了離線儲存與作業的功能，使得 Google 文件自動與本機資源同步，也點出 Google 重視離線儲存這個技術；另一方面，Dropbox 是早於 Google 雲端硬碟的技術，Dropbox 沒有華麗的介面，卻能在手機 App 下載排行高居不下，因為 Dropbox 是推出離線儲存的先驅，建立 Dropbox 帳號就會在電腦裡面自動新增 Dropbox 資料夾，往後本機資料與雲端資料會自動同步更新，也可以設定某些檔案需要分享的對象為何。

2010 年 Hector Gonzalez 等人研究了一個以雲端為基礎資料管理與整合的服務 Google Fusion，最重要的特點是他提供多個不同的使用者一起整合資料，創造出協同合作的環境，使用者可以擁有私人的資料也可以分享給特定的使用者一起協作，或是公開讓所有使用者從搜尋引擎裡面爬到。而 2011 年 Meixing Le , Krishna Kant 以及 Sushil Jajodia 討論了在不同的雲端環境下要如何透過協作來

共享資料以及安全性相關的探討。

基於以上理由，此研究主要是想要建立一個協作雲上面的實作應用，找到一個完整個並行機制讓資料可以在不同的資料庫中同步，除了同步機制確認以外，還能做到離線作業的功能。

二、 研究現況與目的

在資料庫方面，此研究拿 CouchDB 作為底層資料庫，因為 CouchDB 特性很符合這份研究，提供了很強大的備份機制、離線儲存以及上面同步等功能，其他更詳盡內容將會在往後的章節做介紹。選定 CouchDB 作為底層儲存的資料庫之後，接下來便要決定客戶端以及伺服器端要使用哪種技術。

(一) 研究現況

在系統面主要遭遇一些困難，第一：前端介面複雜度高，很常碰到的情況是前端直接呼叫後端系統，因此前端毫無架構可言，這樣會造成前端系統維護成本高，維護不易；第二：前端後端程式語言不同，造成學習太過困難，必須學習多種語言才能順利實作出系統，而且不同系統彼此之間溝通速度慢。故本研究除了使用 CouchDB 作為資料庫以外，還希望能針對以上幾點困難做調整。

(二) 目的與特色

為了解決同步版本控制的問題，本研究最主要的目的是要基於協作雲的機制下，實作出協作雲概念的系統，以盈餘預測系統為例，能達到離線儲存上線同步並且定義出同步與協作機制，討論這個實作系統概念的可行性。研究主要特色有三個。第一：為解決前端介面複雜度高，維護不易的問題，此研究使用兩層式 MVC 應用架構(2-level MVC application framework)，此架構更嚴謹的定義前端介面，因此能更容易做前端介面的維護；第二：使用 JavaScript 應用層框架(JavaScript application framework)，這是一項技術上新的突破，在客戶端與伺服器端皆使用 JavaScript 語言，能達到純粹 JavaScript 語言的環境，所以可以有效解決前後端程式語言不同的問題；最後，本研究使用 CouchDB 作為底層儲存資料庫，能順利達到離線儲存以及同步的機制，系統與資料庫溝通也是透過 JavaScript，在前端的輸出可以更順

利。

(三) 預期貢獻

預期貢獻有五大方向,第一:本研究想要實作協作雲概念的系統;第二::因為 JavaScript 應用層架構是新的技術，所以希望能做出 JavaScript 應用層架構的實作應用，不論在客戶端或是伺服器端都使用 JavaScript 語言，並且定義出明確的使用方法；第三：因為底層儲存資料庫使用的是 CouchDB，因此希望能找出使用 JavaScript 與 CouchDB 互相連接的方式，並且針對關聯式資料庫語文檔型資料庫的轉換機制作討論說明；第四，希望能完整說明 CouchDB 的同步技術,如何利用 CouchDB 做複製以及從傳統的關聯式資料庫到文件型資料庫(CouchDB)的轉換機制為何，明確定義出資料要如何重新設計；最後：本研究提出兩層式架構解決前端複雜度高的問題。

三、 研究途徑

決定系統使用 JavaScript 應用框架、底層儲存使用 CouchDB 之後，接著必須決定要在此架構下建立什麼功能的系統當做實作範例。

由於近年來投資理財已經成為許多人生活中不可分割的一部分，而台灣的投資市場中最活躍的係屬於股票市場。在股票市場中，所有人都想要當贏家沒有人想要輸，但偏偏股票市場是一個極複雜的賽局，無法從中準確預估對手以及賽局如何流動。如果所有投資者都是不知曉賽局情況，那麼做為進入賽局的參賽者來說，這個行為跟賭是沒有什麼不一樣的；但如果部分參賽者了解些許賽局中情況，形成資訊不對稱，那麼對於不知曉的一方來說勝算更是降低，更糟糕的情況可能是不知情的參賽者甚至連其他參賽者知情與否也不可得，因此不論是新聞訊息或是投資理財的部落格上常見到「95%的散戶會賠錢」這句話(散戶泛指參與投資理財一般大眾)，這篇論文並不是要探討這個數據正確與否，只是在這句俗語底下有個問題可以思考，就是該如何增加投資效益呢？

為了增加投資效益，方法之一了解更多賽局中的資訊，資訊獲得愈多獲勝機率也就愈大。為了更加了解賽局中資訊，參考過去文獻，將影響股價的因素分成技術面、基本面、籌碼面以及消息面。技術面是指反應變化的技術指標、走勢型態以及 K 線組合等；基本面則是從公司經營業績、營利能力和成長性著手；籌碼面則是市場上主力進出場情況；而消息面是指新聞或是公司釋出利多利空消息等。

參考先前資料，本論文採用基本面分析。傳統上，股價與盈餘之關係是投資人相當感興趣的問題，因為每股盈餘(Earnings per share，簡稱 EPS)是指公司普通股在一會計期間所賺得之盈餘或發生的損失，每股盈餘常被用來當作評估公司獲利能力的趨勢標準，因此在財務報表分析上，佔有重要的地位。而股價上升或

是下跌以代表了投資人對於公司前景看好與否，。再者，盈餘是公開且容易取得的資訊，因此不只是投資人，這也是會計從業人員以及會計學者所關心的問題。許多學者研究發現，在成熟的股票市場中，每股盈餘與普通股股價有著相當大程度的關聯，基於以上理由，本研究以盈餘資料為基礎做為研究標的。

簡銘宏(1990)的研究指出財務比率對於每股盈餘確實有預測能力，但因產業別不同而有所差異，另外像是邱維正(1991)或是楊慧怡(2007)的研究也都指出產業別對盈餘預測、本益比(股價／每股盈餘)都會有不同的影響，因此本研究使用本益比推測合理股價時也會根據產業別使用產業平均本益比。

所以在 JavaScript 應用框架之下，決定選定盈餘預測來當做此應用框架下的實作範例，建立盈餘預測系統以及利用產業平均本益比來推測合理股價。

四、 論文架構

本文共分為五章，其內容摘述如下：

壹、緒論

說明研究動機、研究現況與目的、研究途徑以及論文架構。

貳、文獻探討

從雲端運算、Collaboration Cloud、CouchDB、JavaScript 應用框架以及盈餘預測模型等五個方面進行文獻回顧。

參、研究方法與架構

說明資料來源、盈餘預測模型以及系統架構。

肆、系統建置與研究結果

說明系統實作需要的技術、需要定義的概念以及建置結果呈現。

伍、結論與未來展望

根據實證結果提出本研究結論、限制與未來展望。

圖一係本研究之論文架構。

論文題目：在 JavaScript Application Framework 下，使用盈餘預測方法以推測合理股價——以台灣股市為例。

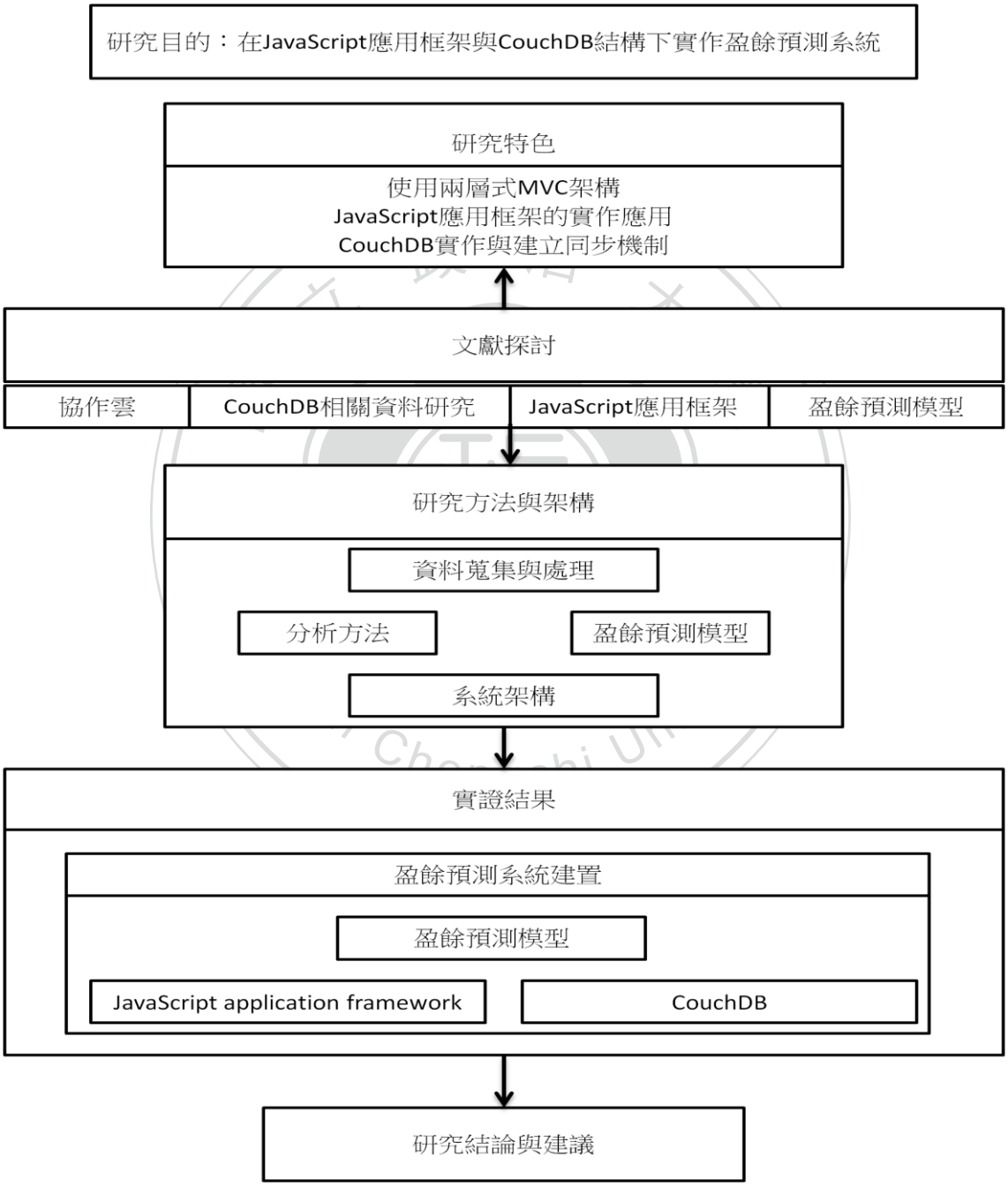

圖 一 論文架構

貳、 文獻探討

本章從五個方面來進行文獻探討，分別是雲端運算、協作雲、CouchDB、JavaScript application framework 以及盈餘預測模型。前四個部分是系統概念與架構，而盈餘預測模型則是主要功能。

一、 雲端運算

根據 NIST 定義，雲端運算為使用無所不在、便利、隨需應變的網路，共享廣大的運算資源，如網絡、伺服器、儲存、應用程式以及服務等，可透過最少的管理工作及服務供應者互動，快速提供各項服務。

雲端運算是一種模式，其依照需求能夠方便地存取網路上所提供的電腦資源，這些電腦資源包括網路、伺服器、儲存空間、應用程式、以及服務等可以快速地被供應，同時減少管理的工作，可降低成本並提昇效能。

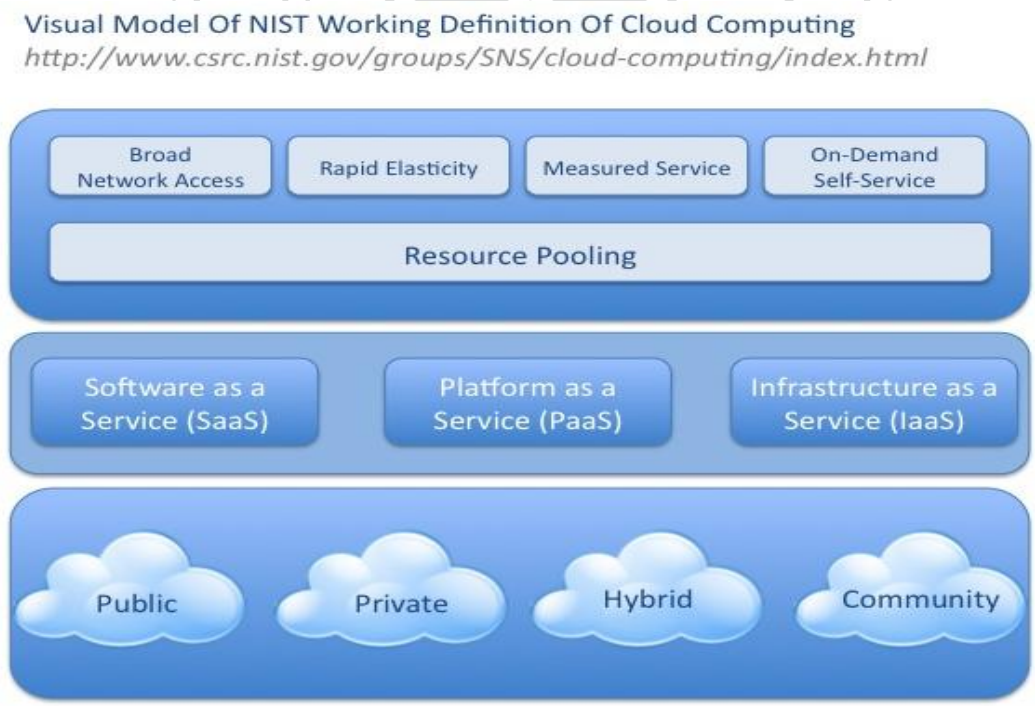

圖 二 NIST 美國國家標準局對雲端運算的定義圖

雲端運算包含五大基本特徵(Essential Characteristic) 及四個佈署模型(Deployment Models)與三種服務模式(Service Model)。

五項基本特徵分別為：

(一) On-demand self-service：

使用者可以依自己的需求直接於網路上取得所需之雲端服務，如網路硬碟或者虛擬伺服器等等，而不需經過人工的機制。

(二) Broad network access：

使用者可以使用電腦、手機或者是更小的部件以標準的溝通機制透過網路取得服務。

(三) Resource Pooling：

多人共享資源，如頻寬、儲存空間、運算資源以及記憶體。

(四) Rapid elasticity：

使用者能夠彈性且快速地重新佈署他們所需要的服務。

(五) Measured Service：

服務是能夠被監控與測量其狀態的。

四種佈署模型分別為：

(一) Private Cloud：

意旨企業自行建置雲端運算平台，其建置成本較為昂貴，但因為企業擁有伺服器控管的權限，所以在安全性以及隱私上的防護較佳，大型企業通常會建置企業本身的私有雲。

(二) Public Cloud：

意旨建置於遠方租賃的伺服器或者是虛擬服務平台，甚至是服務本身，使得企業不用去做資源或者是伺服器的控管，並且可以彈性的調整租用量。但因為企業所有的資訊應用資料皆放置於遠端的公有雲上，因此在安全性與隱私權的威脅相對來說較大。

(三) Community Cloud：

意旨多個組織間互相友善，合作建置共有的社群雲，使得組織間可以共享其他組織所釋出的資源以及分攤雲端的維護費用。

(四) Hybrid Cloud：

將以上三種雲混和即為混和雲，是較為複雜的結構，會出現這種現象通常為私有雲加公有雲。因為某些大型企業會有極大的資源處理需求，私有雲的建置費用極其昂貴，因此會動態條用遠方的服務幫助其運算。

而三種服務模式則分別為：

(一) Infrastructure as a service, IaaS：

也就是提供運算、儲存以及網路等基礎設備的服務，以提供內外部使用者存取之用。為了幫助內部或外部使用者來存取使用，IaaS 通常透過虛擬化技術(Virtualization)來完成伺服器整合的基本作業。目前市面上的 IaaS 以 Amazon EC2(Amazon Elastic Cloud 2), Google Compute Engine 以及 IBM Smart Cloud 最廣為人知。

(二) Platform as a Service, PaaS：

服務提供商提供運算平台給外部開發人員或者使用者，並提供整合的 API 以及相關的管理套件來方便開發人員來構建、開發以及佈署他們的系統，但平台的管理成本相對較為昂貴。目前最有名的為 Google 所推出的 Google App

Engine(GAE), Windows 推出的 Azure 以及 Amazon 的 S3。

(三) Software as a Service：

用戶向服務提供商租用雲端應用服務，使用者透過多種溝通協定對其所租用的軟體進行操作或者取得運算結果。所有軟體的管理以及運轉皆由服務提供商負責，對於使用者管理負擔以及成本的降低有不小的助益。

二、 協作雲相關討論

(一) Concurrency Control in Distributed Database Systems (Philip A. Bernstein and Nathan Goodman ,1981)

同步一直是很重要的問題。在 1981 年這篇論文中提到關於 concurrency control 的方法，這份研究使用了很多的同步技術並且轉換成演算法來解決 concurrency control 的問題,最後這份研究設計了關於 concurrency control 的演算法框架。

(二) Managing and Querying Multi-Version XML Data with Update Logging(Raymond K.Wong and Nicole Lam , 2002)

因為網路上面的資料量越來越大，因此需要控制管理這些資料就變成越來越重要。作者認為使用者會想要查詢到之前版本的資料以及這些文件的變動記錄，以及有效得查詢到某一特定的文件版本，這份研究提出了 XML 的版本管理系統，可以有效地管理及查詢到這些有意義的資料。

(三) Extending Google Docs to Collaborate on Research Papers (Stijn Dekeyser and Richard Watson , 2006)

學者在撰寫研究相關文獻的時候有許多軟體可以選擇，但是很少軟體可以提供他們與其他人協同合作，像是同步版本的系統或是共同合作的系統在當時使用情況還遠不急用電子信箱互相交換文件，主要原因出在系統可用性以及協作環境存在衝突的問題。Google 文件在這方面表現的非常好，特別是學者想要與其他人協同合作一個共同的題目時。此研究比較了 google 文件以及其他解決方案發現用 google 文件可以充分解決這些問題。

(四) Google Fusion Tables: Data Management, Integration and Collaboration in the Cloud(Hector Gonzalez , Alon Halevy, Christian

S. Jensen , Anno Langen , Jayant Madhavan , Rebecca Shapley and Warren Shen , 2010)

Google Fusion 是一個以雲端為基礎資料管理與整合的服務，讓大家上傳表格式的資料,像是 spreadsheet,CSV 等。最重要的特點是他提供多個不同的使用者一起整合資料，創造出協同合作的環境，使用者可以擁有私人的資料也可以分享給特定的使用者一起協作，或是公開讓所有使用者從搜尋引擎裡面爬到，這份研究主要是對 Fusion Table 的使用做討論，討論這些協作者在資料使用上面的細節，還有資料在系統中如何儲存。

(五) Private cloud for collaboration and e-Learning services: from IaaS to SaaS(Frank Doelitzscher , Anthony Sulistio , Christoph Reich , Hendrik kuijs , David Wolf,2010)

Hochschule Futwangen University(HFU)建立了一個私有雲基礎設施，叫做 Cloud Infrastructure and Application CloudIA，CloudIA 主要的目標使用者是 HFU 的教職員工與學生，以及讓外部的人進行協同合作的目的。因此這份研究在介紹 HFU 是如何建立這個私有雲，以及此私有雲是用怎樣的模式讓這個數位平台能在整個大學的環境裡面達到協同合作的作用。

(六) E-Learning Computational Cloud (eLC 2): Web Services Platform to Enhance Task Collaboration(Sidhant Rajam,Ruth Cortez,Alexander VazheninmSubhash Bhalla,2010)

e 化學習平台的建立是很仰賴雲端技術的。這是使用網路服務的技術建立一個使用者介面提供使用者整合以及溝通，所以 e 化社群是一個高度協同合作的環境。此研究主要是要建立一個 e 化學習的服務，來提升協同合作的效率，透過建立這樣的平台，可以讓客戶透過雲端技術去分享資料以達到協作。

(七) Always Up-to-date – Scalable Offline Patching of VM Images in a

Compute Cloud(Wu Zhou; Peng Ning; Xiaolan Zhang; Glenn Ammons; Ruowen Wang and Vasanth Bala , 2010)

版本控制可以讓你電腦系統維持最新的狀態，這份研究主要是提出了一個新穎的工具叫做 Niiwa，他分析版本並儲存起來，並且使得程式在離線時能夠繼續改寫，而此工具 Niiwa 也實際應用在 IBM Research Compute Cloud(RC2)上面。

(八) Cooperative Data Access in Multi-cloud Environments(Meixing Le, Krishna Kant, and Sushil Jajodia , 2011)

這份研究討論在不同的雲端環境下，要怎麼透過協作來共享資料，像是不同企業間擁有自己的資料，彼此共享但又存在某些規則不是完全公開分享，這裡提出了使用權限規則去決定哪些企業可以得到哪些資料，以達到協作環境同時又能確保資料安全性。

三、 CouchDB

CouchDB 是近期新開發的資料庫技術，由 IBM 的 Damien Katz 開發，開發者同時也是 Lotus Notes 的開發者之一，於 2005 年初次發佈，而在 2008 年成為 Apache 的專案。

與傳統的關聯式資料庫不同，CouchDB 是以文件為基礎的資料庫。所謂文件型資料庫，是只 CouchDB 是以文件為儲存單位，因此不像關聯式資料庫一樣是有架構的。在關聯式資料庫當中，每個數據表的字段都要定義為一種類型，像是 int、char 或是 datetime。但在 CouchDB 裡面的字段只有三個：文件編號、文件版本號以及內容，內容字段可以看成是一個 text 檔案的文本，裡面可以隨意定義數據而不用定義數據類型，但此數據必須要以 JSON 格式儲存，也就是 key/value 的格式，但是在 value 方面限制是自由的，可以是字串、陣列或是物件。CouchDB 底層是 ERLANG 語言，以 Restful API 的格式提供服務，所以 CouchDB 所有的讀寫能力都可以通過簡單的 HTTP 請求來實現，正因為採用那麼統一的服務接口，因此可以很方便開發各種語言的個戶端，以方便不同開發者使用。

在 CouchDB 中，Database 表示一個資料庫，每個 Database 對應一個 Storage 以及多個 View Index(用來支持查詢以及儲存結果)。Database Storage 中可以儲存任意的 Document，用戶可以在 Database 中自定義 View，方便對資料庫進行查詢，View 是使用 JavaScript 進行定義，定義好的相關函數保存在 design document 中，而 View 對應的具體數據則是存在 View Index 文件中，我們可以透過上述所說的 HTTP API 請求 Database，Document，View，可以進行簡單的 Query 以及其他各種系統相關的信息。CouchDB 的架構如下圖：

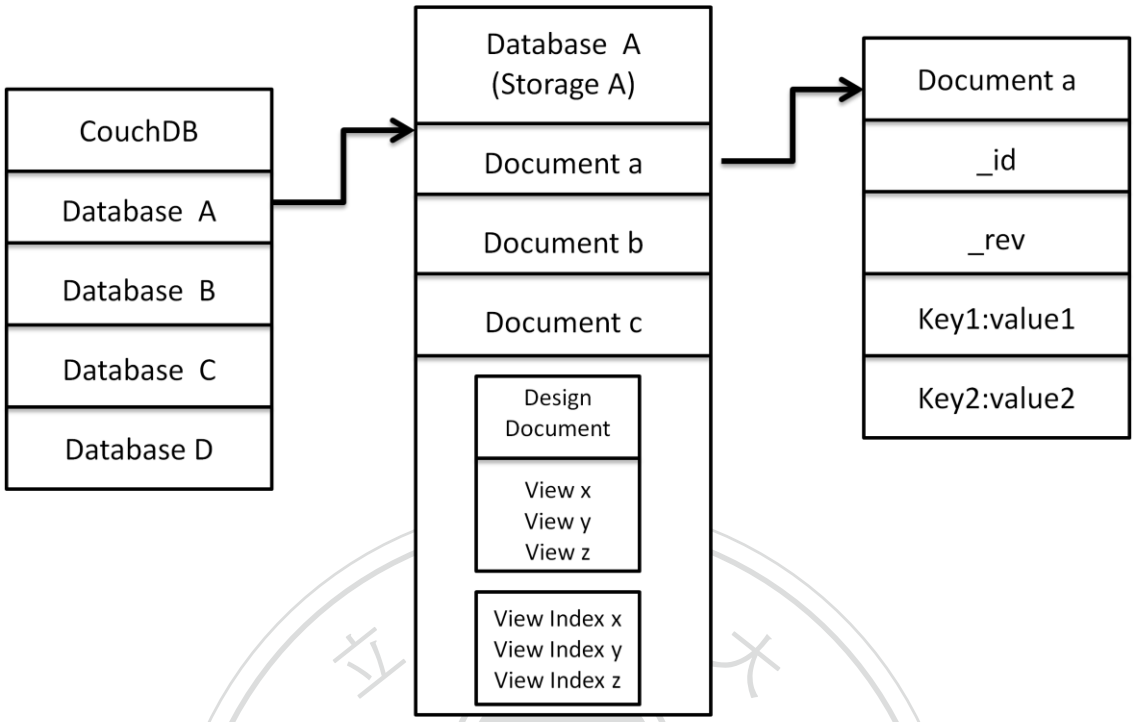

圖 三 CouchDB 架構圖

而底層儲存結構是由 Header 和 Body 組成。以下是關於 Header 與 Body 的規則與特性。Header 包含兩個完全相同的 Header 信息，每個 Header 的 Size 為 2048KB；Header 中前四字節為 magic code：$g、$m、$k、0；隨後為 Header 的 payload，通過 term_to_binary(db_header_record)產生；接下來是填充空格；最後是 16 字節的摘要信息(md5 產生)；Header 總長度為：單個 Header*2 = 2048KB。而 Body 是由兩個 B+Tree 組成，其中一個 B+Tree 根據文件編號進行組織，另一個 B+Tree 以 seqnum(CouchDB 內部使用的序號，用來只是最新版本的文檔)為 key。fulldocinfo_by_id_btree 使用文件編號作為 key，常用來根據文件編號來查找對應的文件，fulldocinfo 中包含對應文件的所有版本信息，通過這些信息，我們可以獲取指定版本的文件；而 docinfo_by_seq_btree 是使用 seq 作為 key，當文件被更新時，對應的 seq 會增加。具體的文件數據(JSON 格式)，以及 B+Tree 混合儲存與這個文件之中。通過 B+Tree，我們可以快速的定位到指定的文件。整個 Storage File 結構如下圖所示：

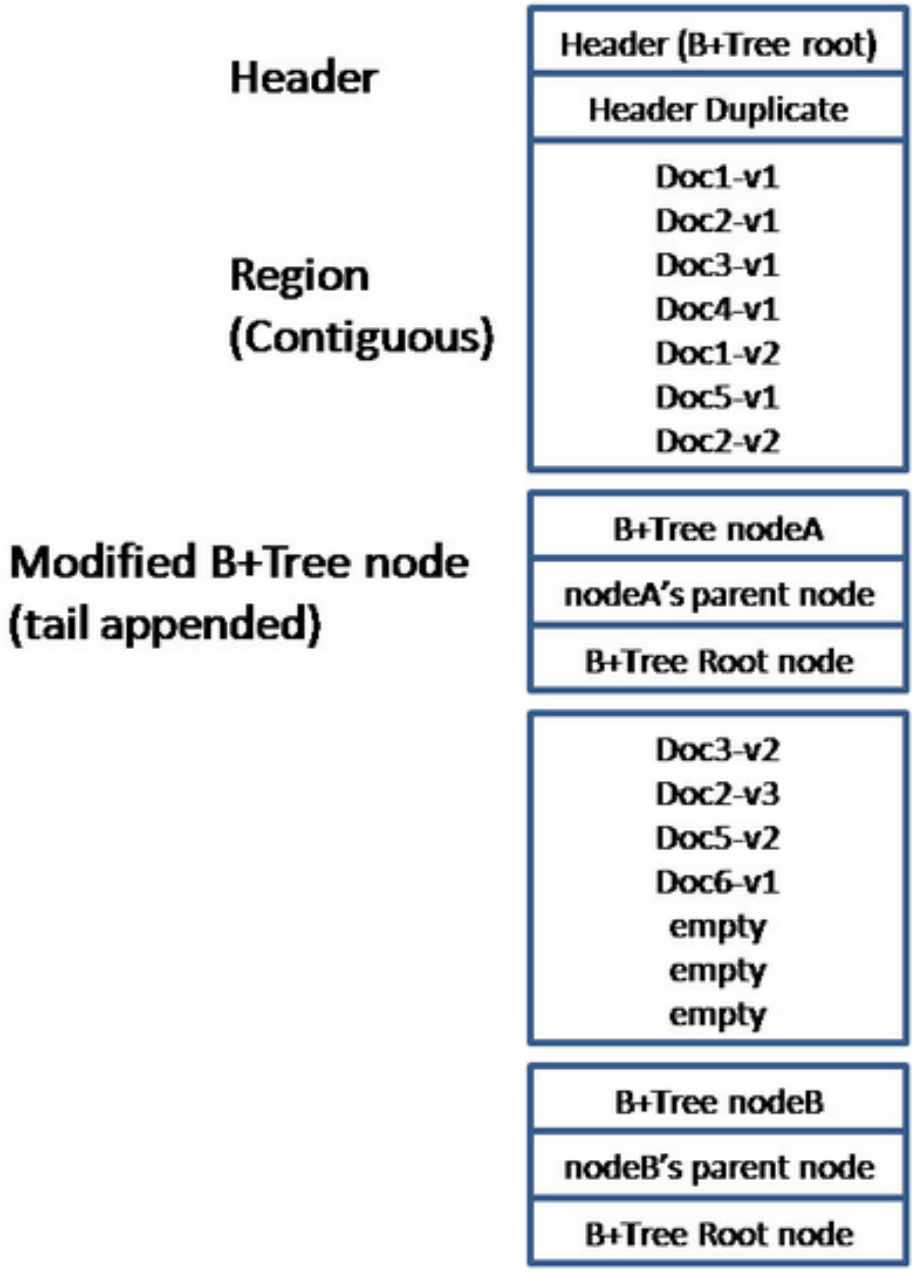

圖 四 底層儲存結構圖

所有的更新操作(包括文件的創建，修改和刪除)都是以在 Couch 文件尾部追加的方式(即 Append 方式)進行。我們進行更新時，首先複製原有的數據信息(僅僅針對修改，如果是創建那麼就沒有複製可言了)，隨後將其追加到文件的結尾，這個時候就激發 B^+Tree 從 leaf 到 root 的更新過程，更新的 Node 信息也是採用 Append 的方式寫入到文件的結尾，到達根節點時，我們將根節點信息寫入到 Header

中。這樣一次更新操作涉及 1 次數據寫入，以及 LogN 次節點更新，所以其複雜度為 O(logN)。

CouchDB 具有以下特性，在此提出來加以說明：

(一) 底層使用 Erlang 編寫

Erlang 主要應用在電信業者，像是北電網路、T-Mobile 等都會用 ERLANG 開發分散式系統，以達到共時(Concurrent)與容錯(Fault-Torrent)的能力，除了共時與容錯，現在多核心與超執行緒(HT)的處理器環境，也提供 ERLANG 語言相當好的發揮環境。而 Erlang 最主要的特色是平行導向程式設計，強調多程序平行運行，並且以訊息對彼此溝通。我們可以想像如果系統要向發一萬筆資料給不同的人，我們可能可以從第一筆發到第一萬筆，但是第一萬個人收到的時間與第一個人收到的時間間隔可能相隔十分鐘，如果使用 Erlnag 就可以輕鬆將一萬筆資料同時送出，這一萬個聯絡人也會幾乎同時收到這些資料。如果在發送的過程中有發生錯誤，這些節點也會回傳錯誤訊息請求重發。當然 Erlang 還有其他特點，但最主要的可以歸為三點：共時(Concurrency)、訊息基礎(Massage Based)與分散式(Distributed)，那麼用 Erlang 開發系統的好處有以下幾點，第一：寫出來的程式，移轉到多核心的環境中執行，速度會自然變快(甚至有可能達到線性加速，n 個核心就提昇 n 倍)；第二：可以寫出容錯的系統，電腦當機之後會重新啟動；第三：寫出來的程式不可思議地精簡。

(二) 資料庫 CAP 特性

資料庫有 CAP 三個特性，分別是資料一致性(Consistent)、可用性(Availability)以及分區容錯性(Partition Tolerance)。一致性是指所有客戶端看到的資料皆是一致的，亦即多台主機間，任何時間點，同一筆資料的一致性；而可用性指的是任何時間點資料都可以讀寫；分區容錯性則是指主

機間出現斷層的容忍性，也就是主機間可不可以有一段時間不能相連(例如網路斷線)。而理論上無法同時兼顧 CAP 這三種特性，如果系統成長到不是一個資料庫可以負擔所有資料的情形，通常會增加更多主機來擴充資料庫，因此分區容錯性在大量資料的使用上更為重要，所以 NoSQL 資料庫通常會選擇 CP 或 AP 這兩種特性來設計。但不管選擇何種特性，最重要的還是主機間能夠同步，當你在這台主機寫了一筆資料，而此筆資料要如何與其他主機間同步便是非常重要的課題。多數 NoSQL 資料庫選擇的是 CP 的設計，像是 CouchDB 也是滿足 CP 兩個條件，CouchDB 談的資料一致性與關聯式資料庫的意義不同。CouchDB 會採取最終一致性(Eventually Consistency)，也就是代表資料遲早會一致的作法，因為 CouchDB 的分散式設計會將資料分散複製到不同節點中，每個節點各自也能異動資料，然後再彼此同步。同步過程就會有時間落差，若同時讀取不同節點上的資料，會發生資料不一致的情況。

以下是 CAP 理論圖：

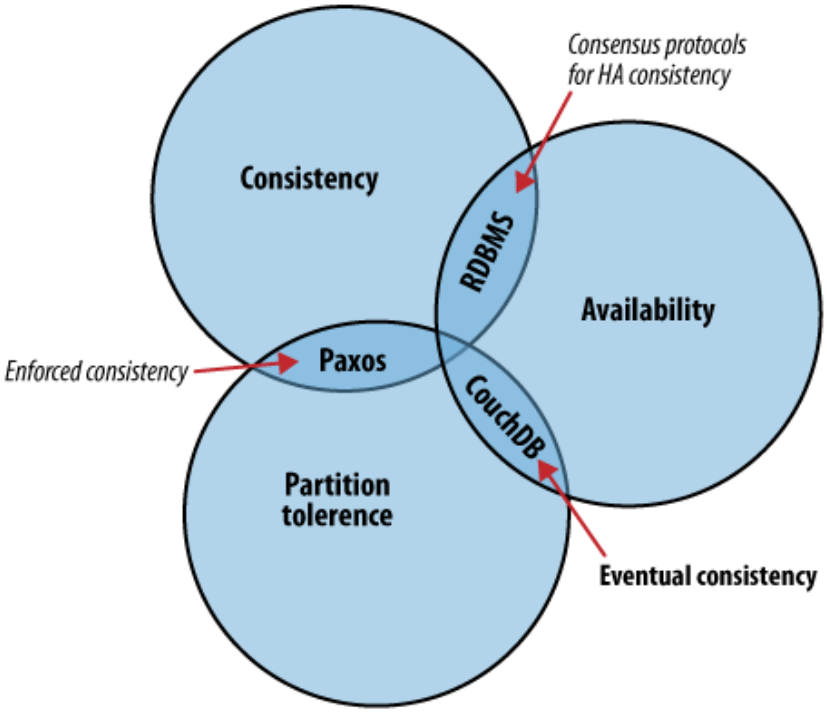

圖 五 CAP 理論圖

(三) 多版本同步控制 Multi-Version Concurrency Control(MVCC)

在關聯式資料庫中，一個表格就是一個資料結構，如果想要更新一個欄位，那麼就必須保證沒有其他使用者正在更新資料，也沒有人可以在資料更新完成前讀取資料，所以最好的解決方法就是鎖定資料庫。如果有多個使用者都想要讀寫資料庫，第一個進入資料庫的使用者就會鎖定資料庫，那麼其他使用者就必須等待，直到第一個使用者使用結束，才輪到下個使用者，以此類推。因此傳統的關聯式資料庫會花很多時間浪費在等待進入資料庫的時間。而 CouchDB 的特性之依舊是 MVCC，在 CAP 理論中，CouchDB 滿足了 AP，其中可用性(Availability)是指隨時都能讀寫資料庫，也就是 CouchDB 讀寫均不鎖定資料庫(Non-Blocking I/O)。

下圖為傳統資料庫與 CouchDB 在讀寫方面之比較：

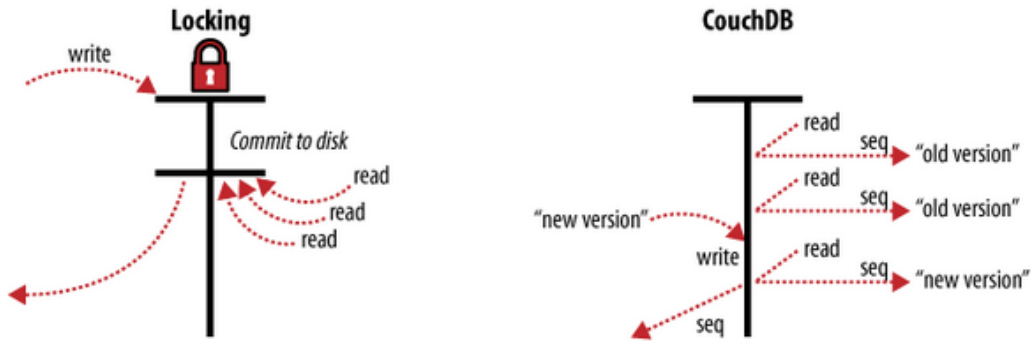

圖 六 傳統資料庫與 CouchDB 在讀寫方面的比較圖

所有 CouchDB 的文件都會有版本號，如果你更新一個文件，其實是在下一個版本號中做所有的更新，而舊版本的資料還是會全部保留。那麼 CouchDB 是如何做到讀寫不鎖定資料庫還是保持正常運作呢？在做任何更新之前，都會先取得文件的版本號，接著在寫入時檢查目前文件版本號是否與取出時相同，如果相同則代表從取出文件修改至寫入的過程中，文件並沒有任何變動，

此時便可以寫入；但如果寫入時文件版本號超過讀出時的版本號，則代表文件在取出更改的過程中被更新到其他版本，這時 CouchDB 就會發出衝突通知，告知使用者衝突的發生。

```
USER1     ----------->  GET /db/bob
          <-----------  {"_rev":"1-aaa", ...}

USER2     ----------->  GET /db/bob
          <-----------  {"_rev":"1-aaa", ...}

USER1     ----------->  PUT /db/bob?rev=1-aaa
          <-----------  {"_rev":"2-bbb", ...}

USER2     ----------->  PUT /db/bob?rev=1-aaa
          <-----------  409 Conflict  (not saved)
```

圖 七 版本與衝突範例

以上圖為例，USER1 與 USER2 都對 bob 這個資料庫下指令取得版本號為 1-aaa，而後 USER1 與 USER2 按照先後順序分別寫入資料，USER1 寫入資料時順利寫入，但是 USER2 寫入時版本已經由於剛剛 USER1 的寫入所以更新為 2-bbb，因次發生衝突更新失敗。

(四) 以文件為儲存基礎

如上述所說，CouchDB 的資料結構與傳統的關聯式資料庫是不同的。CouchDB 是以文件為儲存單位，文件內容為 JSON 格式的 key/value 鍵對，而文件內容的值可以是字串、數值或是日期格式，這些都是不受限制的，甚至可以是陣列或是物件。與關聯式資料庫最大的不同是數據之間彼此沒有關連性，因此也沒有關聯式資料庫中 join 的觀念。而最重要的每個文件都有自己獨特的文件編號，以方便讀取或是更新資料。

(五) 用戶自定義查詢

CouchDB 使用 View 來做自定義查詢，使用的是 MapReduce 來計算結果，關於 MapReduce 先在以下做簡單的介紹。

關於 MapReduce：

MapReduce 是一個分散式的運算架構，讓開發者可以撰寫程式利用大量的資源做分散式運算，得以處理相大龐大的資料量。圖三為 MapReduce 運作機制。

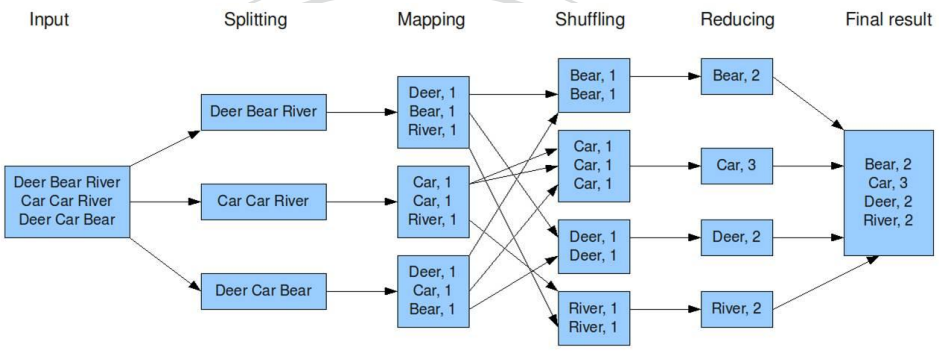

圖 八 MapReduce 運作機制

見圖三，MapReduce 運作可以分成兩個部分，Map 和 Reduce，大量的資料在運算開始的時候，會被系統轉換成一組組(key,value)的序對並做 Spitting 的工作，接下來分別傳給不同的 Mapper 做 Mapping 的動作，Mapper 處理完成之後也要將運算結果整理成一組組(key,value)的序對，最後經由 Reducer 做 Reducing 的動作，整合所有 Mapping 的結果，最後才能將整體的結果輸出。

CouchDB 的儲存核心是很強的 B+Tree 儲存結構，B+Tree 儲存架構提供很優秀的資料搜尋、更新以及刪除的功能。下圖是 CouchDB 使用 B+Tree 儲存資料、文件以及 View，可以從中看出對資料庫發出一個請求並且透過 startkey 和 endkey 建立搜尋條件之後，CouchDB 是如何利用 B+Tree 找到資料

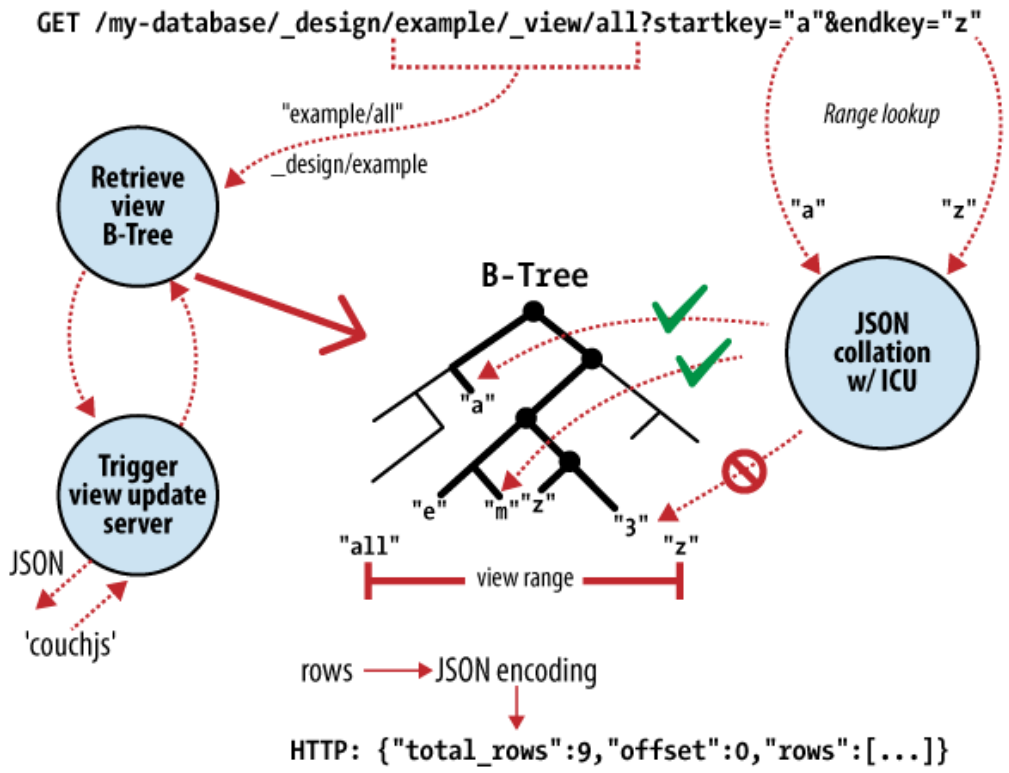

圖 九 View 請求關係圖

通過用戶自定義 View，我們可以匯集，統計數據，採用一個類似 Map/Reduce 的過程。這裡的 Map 將原始的 Document 進行映射處理，Reduce 將 Map 的中間結果進行重新歸併統計，總而生成最終結果。這裡和並行計算中的 Map/Reduce 有些不同。CouchDB 的 View 針對每個 Database，但是其與 Database 關聯性不是很大，View 是一些用戶自定義函數，處理從數據庫的 Document 輸入，產生中間數據（如果沒有 reduce 過程則為最終數據），然後再通過 Reduce 處理中間輸出，產生最終結果。

CouchDB 內部使用 JavaScript 作為 View 的編寫語言，之所以採用 JavaScript，是和 CouchDB 面向 Web 開發相關的。 View 中包含兩個函數，

```
(map函數，必須)
function(doc){
    emit(null , doc);
}
(reduce函數，可選)
function(key , values m rereduce){
    return sum(values);
}
```

圖 十 MapReduce 使用範例

map 與 reduce，圖十是兩個函數的基本用法。Doc 是我們數據酷對應的文件，因為我們採用 JSON 格式儲存數據，所以文件再 JavaScript 中轉化為物件。Emit(null , doc)用來生成 map 的中間結果，其中第一個參數 null 表示結果的 key，第二個參數為結果的 value。

(六) Restful API

CouchDB 所有的讀寫能力都可以通過簡單的調用他的 HTTP 請求來實現。正因為採用那麼一種統一且簡潔的服務接口，可以很方便地開發各種語言的客戶端，像是 C#、Futon、Java、Perl、PHP、Python、Ruby 或是 JavaScript 以及 Node.js 跟 ExtJS 等都有提供支援，方便不同程序員使用。

Restful API 使用 HTTP 的方法 POST，GET，PUT 以及 DELETE 來做基本的 CRUD(Create , Read , Update , Delete)操作。

(七) 內建備份機制

對於大多數資料庫來說，維持資料庫的一致性在單一的資料庫是相對簡單地；真正困難地是如果維持多個資料庫間的一致性。如果今天客戶對資料庫 A 做了操作，要如何同步到資料庫 B、資料庫 C 或是資料庫 D 呢？

而 CouchDB 不用擔心同步的問題，因為 CouchDB 有內建備份機制，在不同資料庫之間存在著最終一致性，也就是資料遲早會同步。因此根據 CouchDB 的備份機制，就可以輕鬆將兩台甚至多台資料庫的資料同步。在備份之後，每個資料庫還是可以獨力作業。而此備份機制也支援離線作業，多台主機如果互相連接，只要連上線後資料便會自動同步。

下圖是 CouchDB 增量複製的示意圖：

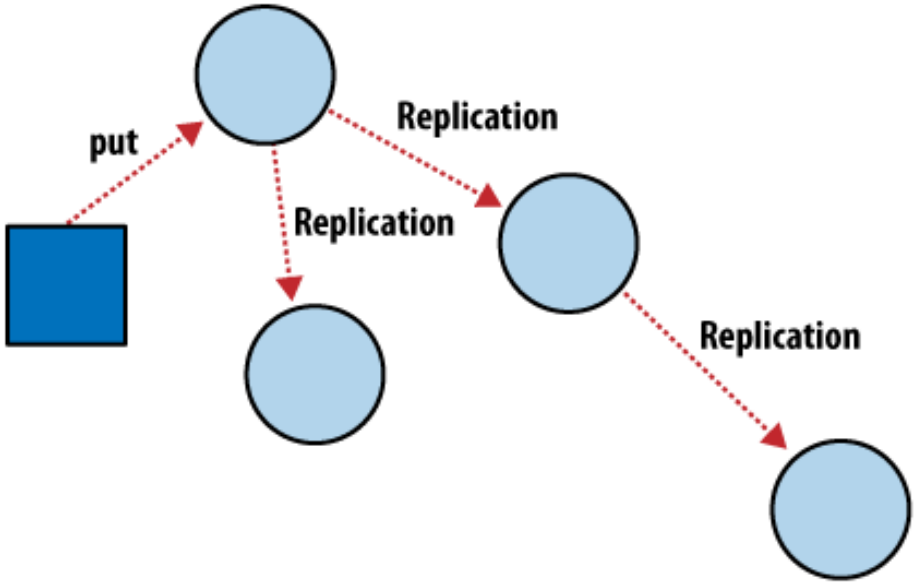

圖 十一 增量複製示意圖

在多個資料庫同步的環境中，一旦文件被更新，其他資料庫的文件也會一同被更新，如果多個不同資料庫裡的同一個文件被更新，那麼更新機制便會參照前面 MVCC 所述，以取得之版本與寫入之版本做比較，最先寫入的會取得寫入權，並且自動同步至其他資料庫，而後寫入的則會發生衝突。其他複製機制將在第四章做更詳盡的討論。

同樣是 NoSQL，CouchDB 很容易被拿來跟 MongoDB 與 Cassandra 做比較，因為當中有一些相似之處，但是也有許多不同之處，以下先對 MongoDB 與 Cassandra 做簡單介紹。

MongoDB 是一個基於分佈式文件儲存的數據庫開源項目。由 C++語言編寫。旨在為 Web 應用提供可護展的高性能數據儲存解決方案。他的特點式高性能、易部屬、容易使用以及儲存數據非常方便。主要特性有以下幾點：面向集合儲存，容易儲存對象類型的數據，所謂面向集合(Collection-Oriented)，意思是數據被分組儲存在數據集中，稱為一個集合。每個集合在數據庫中都有一個唯一的標識名，並且可以包含無限數目的文檔。集合的概念類似關聯式資料庫裡面的表，但不同

的是他不需要定義任何模式(Schema)；模式自由(Schema-free)，意味著對於儲存在 MongoDB 數據庫中的文件，我們不需要知道他的任何結構定義；支持動態查詢；支持複製與故障恢復；使用高效的二進制數據儲存，包含大型對象(如視頻等)；自動處理碎片，以支持雲端計算層次的擴展性；支持 Ruby、Python，Java，C++、PHP 等多種語言；文件儲存格式為 BSON(一種 JSON 的擴展)；可通過網絡訪問。

Cassandra 是基於 Java 的分佈式鍵值資料庫。他不像 MySQL 包含 JOIN 操作，他幫你處理分佈式資料。你可以考慮整個 cluster 是一個很大的 hash table。所有的容錯翰資料分割都是由 cassandra 自己處理。他提供了增量的可擴展性(這意味著你可以增加新的機器來容納更多的處理)。Cassandra 還支持 Column 功能，使他比純粹鍵值的資料庫更方便。在 Cassanddra 中，他可以被認為是一個四、五維的 hash table。從上而下層次結構如下圖所示：

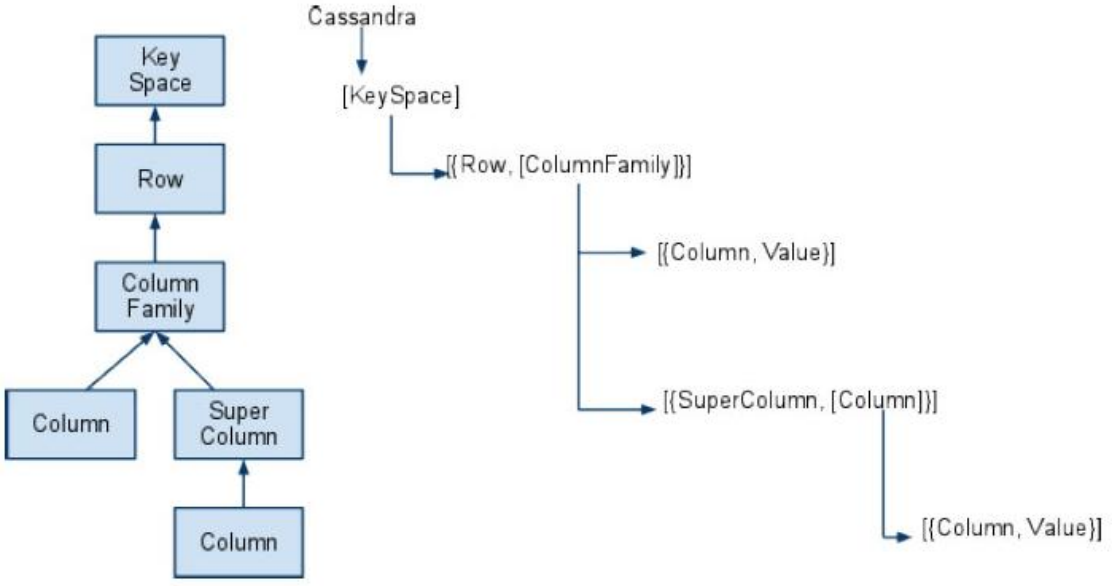

圖 十二 Cassandra 資料結構圖

Cassandra 使用 consistent hash 來做 key partition。每個節點在 Cassandra cluster 中將在 ring 中選取一個 token($0 < token < 2^{32}$)。所以每個節點都會在 ring

裡面有一個相對應的位置，當新的 key 輸入時，他將使用 MD5 hash，找到在 ring 裡面的下一個節點。資料將被儲存在相對應的節點。

以下是 CouchDB、MongoDB 與 Cassandra 的比較表：

	CouchDB	MongoDB	Cassandra
編寫語言	Erlang	C++	Java
主要特點	資料庫一致性 版本控制機制	保留一些友善的 SQL 屬性	三者之中績效最好
許可	Apache	Apache	Apache
協定	HTTP/Restful API	Custom (BSON)	Custom (Thrift)
優點	有大量數據，但更新量不大，需要預先定義查詢	動態查詢、方便數據經常更動，定義索引而非 MapReduce	大量數據、所有系統裡的元件都是用 Java 寫成
舉例	CRM、CMS 系統，主機與主機的複製機制	大部分想要用 MySQL 當資料庫的情形	銀行、財務產業的系統，寫入速度大於讀取

表 一 CouchDB、MongoDB 與 Cassandra 比較表

四、 JavaScript Application Framework 介紹

JavaScript Application Framework 是一個網路應用框架，結合了多項技術而形成整體框架，像是 CSS3、JavaScript、HTML5、Node.js、YQL、YUI，而此應用框架有下列幾點特色：第一：它在客戶端及伺服器端皆 p 使用 JavaScript，以往 JavaScript 只能在客戶端運作，而伺服器端往往需要使用 PHP 或是 JSP 等其他語言；第二：整個運作環境是純粹的 JavaScript，不再需要不同語言之間的溝通；第三：它是一個程式語言，兩個執行期(Runtime)，分別在客戶端與伺服器端各跑一次。以下分別針對 JavaScript 應用架構、YUI 以及 Node.js 做進一步說明。

(一) JavaScript Application Architecture

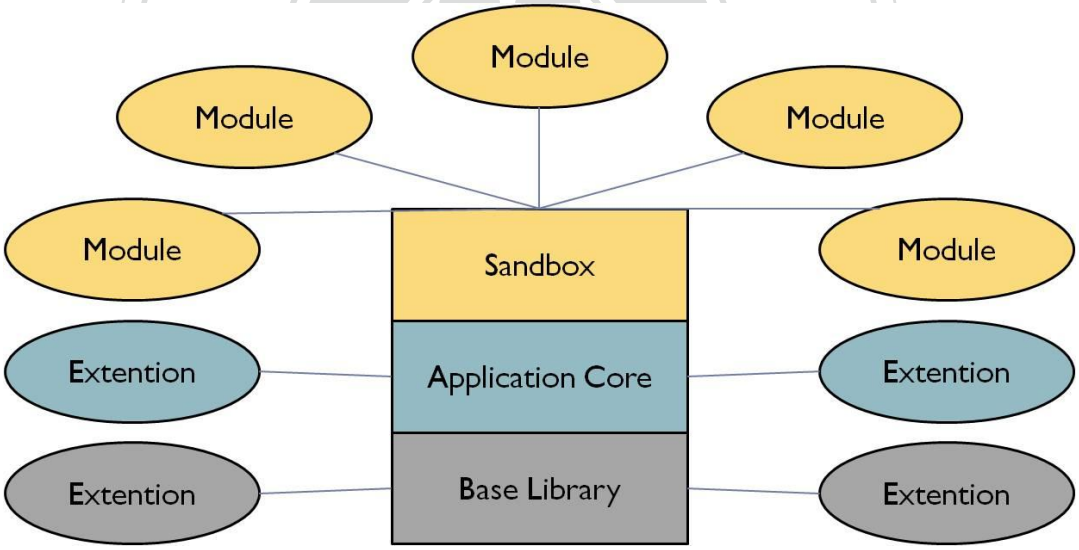

圖 十三 JavaScript 應用架構

上圖六為 JavaScript 的應用架構，主要分成 Module、SandBox、Application Core 以及 Base Library 這四個部分。

Module 最重要的特性是每個 Module 在整個網路應用架構下都是獨立的，而網路應用模組是包含 HTML、CSS 以及 JavaScript，Module 具有以下特色：每個 Module 彼此沒有關係；Module 不能直接使用其他 Module；不能創造全

域變數；所有操作都是透過 SandBox；不能在 Box 之外操作 DOM 物件。Module 主要功能就是創造有意義的使用者經驗，像是天氣的 Module 就是在告知使用者天氣狀況而股票的 Module 就是提供使用者股票市場相關訊息。

Sandbox 主要特點有三個，第一是一致性，他必須建立 Module 標準的呼叫格式，以提供 Module 對 Sandbox 使用的方法；第二是安全性，所有 Module 運用全部都要通過 Sandbox，不得互相呼叫也不得沒透過 Sandbox 直接使用；最後是溝通性，Sandbox 是 Module 之間要互相溝通的重要角色，正因為 Module 之間不能直接互相呼叫，因此更需要透過 Sandbox 來達到互相溝通的效果。

Application core 就是一個應用層的控制者，這個部分必須告訴 Module 何時必須初始，何時必須終止。此層主要的特點有四個，第一個是管理 Module 的生命週期，如同上述，必須告訴 Module 何時初始以及終止；第二個部分是讓 Module 之間可以互相溝通，與上面 Sandbox 不同的是，Sandbox 是 Module 呼叫與使用的管道，而真正的管理是由 Application core 控制；第三點是一般的錯誤處理；第四點是此層必須是可以擴充的，不管是在錯誤處理、Ajax 通訊、新的 Module 建立、一般使用等等，都是要可以繼續擴張的。

最後 Base Library 提供了基本的運作，在一般使用上有幾個目的，可以轉換 XML、JSON 等格式、還有對物件的操作、對 DOM 的控制以及 Ajax 的通訊。

只有 Base Library 知道現在的運行環境是哪個瀏覽器，其他部分不需要知道，而只有 Application core 知道是哪個 Base Library 被使用，只有 Sandbox 知道是哪個 Application core 在使用，但 Module 只知道 Sandbox 的存在，其他的東西 Module 全然不知，這就是整個 JavaScript 的運作架構。好處就是不同的應用都可以建立在這個框架下，在這個定義嚴謹的架構下每個部分是分開獨立的，不管是管理或是維護上都可以達到更高效率的成果。

(二) YUI

YUI 是 Yahoo!的使用者介面資料庫(Yahoo! User Interface Library)，是一個包含了 JavaScript 以及 CSS 的框架,YUI 的特色是可以跨瀏覽器支援，不像有些前端語言需要針對不同瀏覽器寫不同的語法，而 YUI 主要的功能有 DOM(Document Object Model)的操作，發送 Ajax 的請求，建立許多類型的套件等。而且因為 Dav Glass 的開發，目前已經可以使用 YUI 在伺服器端的 Node.js 執行了。

(三) jQuery

jQuery 是一套跨瀏覽器的 JavaScript 函式庫，強化 HTML 與 JavaScript 之間的操作。由 John Resig 在 2006 年 1 月的 BarCamp NYC 上釋出第一個版本。目前全球有 28%的網站使用 jQuery，是目前最受歡迎的 JavaScript 函式庫。jQuery 免費且為開放原始碼，使用 GPL 和 MIT 許可證雙協議[3]。jQuery 的語法設計使得許多操作變容易，如操作文件（document）、選擇 DOM 元素、動畫效果、事件處理、發展 Ajax 以及其他功能，用 jQuery 撰寫可以輕量化程式碼，將原本需要寫到 50 行的程式碼用 10 行處理完。除此之外，jQuery 提供 API 讓開發者將自己所寫的功能融入 jQuery 內。jQuery 有以下特點：跨瀏覽器的 DOM 選擇、事件驅動、CSS 操控、特效及動畫、Ajax、延伸性以及輕量級等。

(四) Node.js

Node.js 是 Ryan Dahl 於 2009 年所創造的框架，建立在 Chrome 瀏覽器的 JavaScript 引擎上，此引擎的名稱為 V8，受益於 V8 的即時(Just In Time)編譯功能,Node.js 應用的運行速度幾乎可以與本機比擬，對比之下,Node.js

要比 PHP 以及 Ruby 快上許多，相較於 YUI 是在瀏覽器端執行，Node.js 則是在伺服器端執行。Node.js 有以下幾點特色：非同步的 I/O、模組為基底的系統、建立於 HTTP 伺服器的函式庫以及事件驅動的特性。非同步的 I/O 是指在程式底下的函式必須立即回傳值，以此達到更高效率；模組化的系統是指 Node.js 是在需要的時候才將要用到的模組載入，這讓 Node.js 的程式碼相較之下比較輕量化；而且也擁有內建的 HTTP 伺服器端的函式庫；最後事件驅動的特性與傳統的執行緒驅動的作法不同，相較於執行緒驅動會耗費大量的記憶體，事件驅動的方法是可以耗費較少資源，缺點是互動性比較差。

五、 盈餘預測模型

根據張嘉玲(2007)的文獻，可將盈餘預測模型劃分與歸類。盈餘預測資訊的來源，按照使用者不同可以分為時間序列模型以及分析師盈餘預測模型。按照是否針對盈餘進行拆解又可劃分成盈餘彙總數字為基礎的盈餘預測模型以及以盈餘組成要素為基礎的盈餘預測模型，而後者按照拆解方式又可分為盈餘功能性解構的盈餘預測模型以及資金基礎解構的盈餘預測模型(洪慧娟，2002)。除了以盈餘為基礎的盈餘預測模型以外還有以成本習性為基礎所解構的盈餘預測模型，像是 Banker and Chen(2006)以成本變動性與成本僵固性等成本習性所提出的 CVCS 模型。各個盈餘預測模型按照上述分類彙整成圖七。

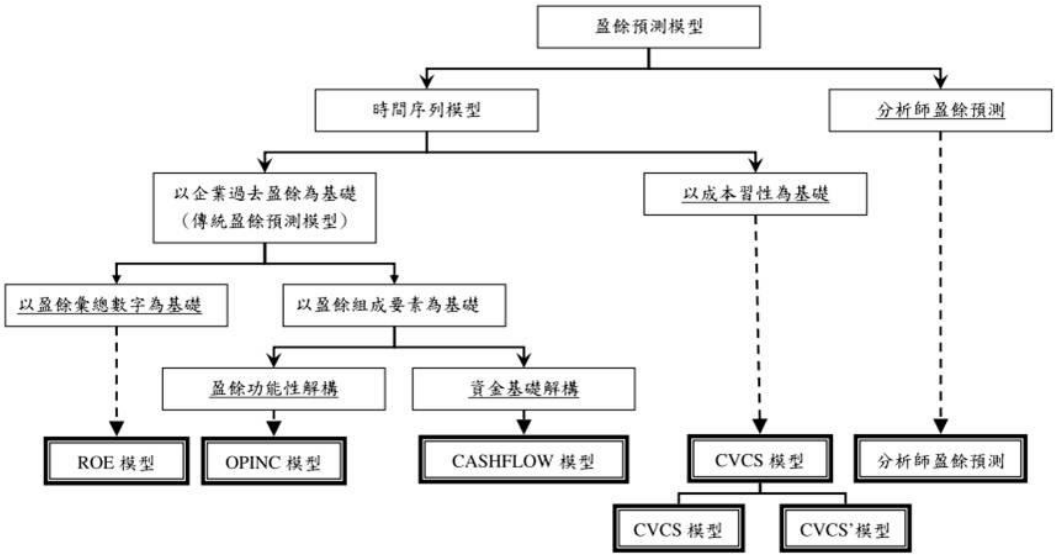

圖 十四 盈餘預測模型之劃分與歸類

以下，引用相關文獻對上圖時間序列模型中各個盈餘預測模型作更深入的解釋。

(一) ROE 模型

ROE 模型屬於以盈餘彙總數字為基礎的傳統盈餘預測模型，Ruland(1978)曾述及，盈餘預測資訊之使用者可以自由地使用任何盈餘預測模型以形成對企業未來盈餘之預期，而在選擇盈餘預測模型時，經驗豐富、判斷準確與成本較高之盈餘預測模型較容易受到資訊使用者之青睞。因此僅以企業過去盈餘為基礎所建構之盈餘預測模型，如 ROE 模型，所預測之企業未來盈餘，理論上並不應該視為市場對企業未來盈餘預期之替代變數。然而，Ruland(1978)強調，以企業過去盈餘為基礎所建構之盈餘預測模型實為判定特定盈餘預測模型是否具有盈餘預測能力之最低門檻。

(二) OPINC 模型

OPINC 模型屬於盈餘功能性解構之傳統盈餘預測模型，Fairfield et al.(1996)以 1973 年至 1986 年每七年為一樣本期間，建立以盈餘彙總數字為基礎之盈餘預測模型與盈餘功能性解構之盈餘預測模型，據以預測 1981 年至 1988 年平減後稅後淨利(bottom-line ROE)，實證結果顯示：將盈餘彙總數字拆解成平減後營業損益、平減後營業外純益、平減後特殊項目與平減後非重複發生項目，將使估計錯誤數絕對值之平均數從 0.2958 下降至 0.2735，換言之，相較於以盈餘彙總數字為基礎之盈餘預測模型，盈餘功能性解構之盈餘預測模型具有較高之盈餘預測能力。

(三) CASHFLOW 模型

CASHFLOW 模型屬於資金基礎解構之傳統盈餘預測模型，Sloan(1996)以 1962 年至 1991 年為樣本期間，將盈餘拆解成應計數與現金流量，以探討應計數與現金流量之盈餘預測能力。實證結果顯示：現金流量與應計數之迴歸係數皆顯著異於零，換言之、現金流量與應計數在盈餘預測上皆具有解釋能力。

Banker and Chen(2006)以 1988 年至 2001 年每四年為一樣本期間，建立以盈餘彙總數字為基礎之盈餘預測模型與資金基礎解構之盈餘預測模型，據

以預測 1992 年至 2002 年平減後營業部門淨利。實證結果顯示：將盈餘彙總數字拆解成應計數與現金流量，將使估計錯誤數絕對值之平均數從 0.1427(0.1215)下降至 0.1420(0.1206)，換言之、相較於以盈餘彙總數字為基礎的盈餘預測模型，資金基礎解構的盈餘預測模型皆具有較高之盈餘預測能力。

(四) CVCS 模型

Banker and Chen(2006)以理論推導之方式，利用成本變動性與成本僵固性等成本習性建構 CVCS 模型，並從實證的角度，探討 CVCS 模型盈餘預測之盈餘預測能力。實證結果顯示：不論相較於以盈餘彙總數字為基礎之 ROE 模型、盈餘功能性解構之 OPINC 模型或是資金基礎解構之 CASHFLOW 模型，CVCS 模型皆具有較高之盈餘預測準確度與較豐富之資訊內涵，換言之，CVCS 模型在理論上與實證上皆具有一定程度之合理性。

(五) 多變數迴歸模型

此模型之變數為先前文獻中提出，楊慧怡(2001)以各公司的財務比率為資料來探討其與公司每股盈餘之間的關連性。為了提高其解釋能力，楊慧怡(2001)檢視各財務比率與每股盈餘之皮爾森相關係數(Pearson correlation Coefficient)，再以台灣經濟新報資料庫(TEJ)對財務比率的分類，對同一財務屬性選出與每股盈餘相關係數高且具有意義的財務比率為代表。

參、 研究方法與架構

一、 資料蒐集整理

本研究之資料來源為台灣經濟新報資料庫(TEJ)，蒐集台灣 2001 年~2011 年上市公司的財務資料。

資料處理的部份，由於不同產業本益比相差甚鉅，而且產業平均 EPS 也不盡相同，因此將資料按照 TEJ 產業別分類，分別有水泥工業、食品工業、塑膠工業、紡織工業、電機機械、電器電纜、化學生技醫療、玻璃陶瓷、造紙工業、鋼鐵工業、橡膠工業、汽車工業、電子工業、建材營造、航運、觀光、金融保險、貿易百貨、證券、投資信託、油電燃氣、綜合等 22 個產業。

二、 盈餘預測模型

(一) 盈餘預測模型

1. ROE 模型(以盈餘彙總數字為基礎之盈餘預測模型)

$$ROE_t = \Upsilon_{a0} + \Upsilon_{a1} ROE_{t-1} + \varepsilon_{at}$$

ROE_t：t 年期初股東權益帳面價值平減後盈餘；

ROE_{t-1}：t-1 年期初股東權益帳面價值平減後盈餘；

2. OPINC 模型(盈餘功能性解構之盈餘預測模型)

$$ROE_t = \Upsilon_{b0} + \Upsilon_{b1} OPINC_{t-1} + \Upsilon_{b2} NOPTAX_{t-1} + \Upsilon_{b3} SPECIAL_{t-1} + \varepsilon_{bt}$$

ROE_t：t 年期初股東權益帳面價值平減後盈餘；

$OPINC_{t-1}$：t-1 年期初股東權益帳面價值平減後營業損益；

$NOPTAX_{t-1}$：t-1 年期初股東權益帳面價值平減後營業外純益；

$SPECIAL_{t-1}$：t-1 年期初股東權益帳面平減後特殊項目；

3. CASHFLOW 模型(資金基礎解構之盈餘預測模型)

$ROE_t = \Upsilon_{c0}+\Upsilon_{c1}CFO_{t-1}+\Upsilon_{c2}ACCRUAL_{t-1}+\varepsilon_{ct}$

ROE_t：t 年期初股東權益帳面價值平減後盈餘；

CFO_{t-1}：t-1 年期初股東權益帳面價值平減後來自營業活動之現金流量；

$ACCRUAL_{t-1}$：t-1 年期初股東權益帳面價值平減後應計數；

4. CVCS 模型(以成本習性為基礎)

$ROE_t = \Upsilon_{d0}+\Upsilon_{d1}\cdot D_t+\Upsilon_{d2}\cdot ROE_{t-1}+\Upsilon_{d3}\cdot S_{t-1}+\Upsilon_{d4}\cdot S_{t-1}\cdot D_t+\varepsilon_{dt}$

ROE_t：t 年期初股東權益帳面價值平減後盈餘；

ROE_{t-1}：t-1 年期初股東權益帳面價值平減後盈餘；

D_t：t 年銷貨收入減少與否之虛擬變數，減少時為 1，否則為 0；

S_{t-1}：t-1 年期初股東權益帳面價值平減後銷貨收入淨額；

5. 多變數迴歸模型(選取財務屬性相同、相關係數高之財務比率)

根據楊慧怡(2001)文獻中選用之七個財務比率為自變數。

應變數：每股盈餘；

自變數：營業費用率、現金流量比率、負債比率、流動比率、營收成長率、營業利率成長率、營業利益率；

迴歸模型為通用表示式：$Y = \alpha+\beta 0X0+ \beta 1X1+\cdots+ \beta 7X7+\varepsilon$

Y：應變數；

Xi：自變數；

α, β：迴歸估計係數；

ε：誤差值；

(二) 變數定義

1. 平減後盈餘(ROE)【T3920/T2000】

2. 平減後營業損益(OPINC)【(T3925-T3300-T3510)/T2000】

3. 平減後營業外純益(NOPTAX)
   【(T3400-T3500+T3510-T3447-T3489+T3547+T3590-T3910)/T2000】

4. 平減後特殊項目(SPECIAL)【(T3434+T3489-T3547-T3590)/T2000】

5. 平減後來自營業活動之現金流量(CFO)【T7210/T2000】

6. 平減後應計數(ACCRUAL)【(T3920-T7210)/T2000】

7. t 年銷貨收入減少與否之虛擬變數(D_t)

8. 平減後銷貨收入淨額(S)【T3100/T2000】

以下將各研究變數與其相對應資料來源，彙整成表二。

資料來源				變數
台灣經濟新報資料庫	台灣財經資料庫	TEJ Finance DB	上市(櫃)合併財務(累計)一般產業	薪資合計(T330A)
				存貨(T0170)
				固定資產(T0400)
				股東權益總額(T2000)
				營業收入淨額(T3100)
				營業毛利(T3295)
				營業費用(T3300)
				營業外損益(T3400-T3500)
				減損迴轉利益(T3447)
				其他收入(T3489)
				利息支出(T3510)
				減損損失(T3547)
				其他損失(T3590)
				所得稅費用(T3910)
				繼續營業部門純益(T3920)
				來自營運之現金流量(T7210)
				每股淨值(TR308)
				每股盈餘(T3990)

表 二 本研究使用之變數與代碼

三、 系統架構

(一) 傳統 MVC 應用框架

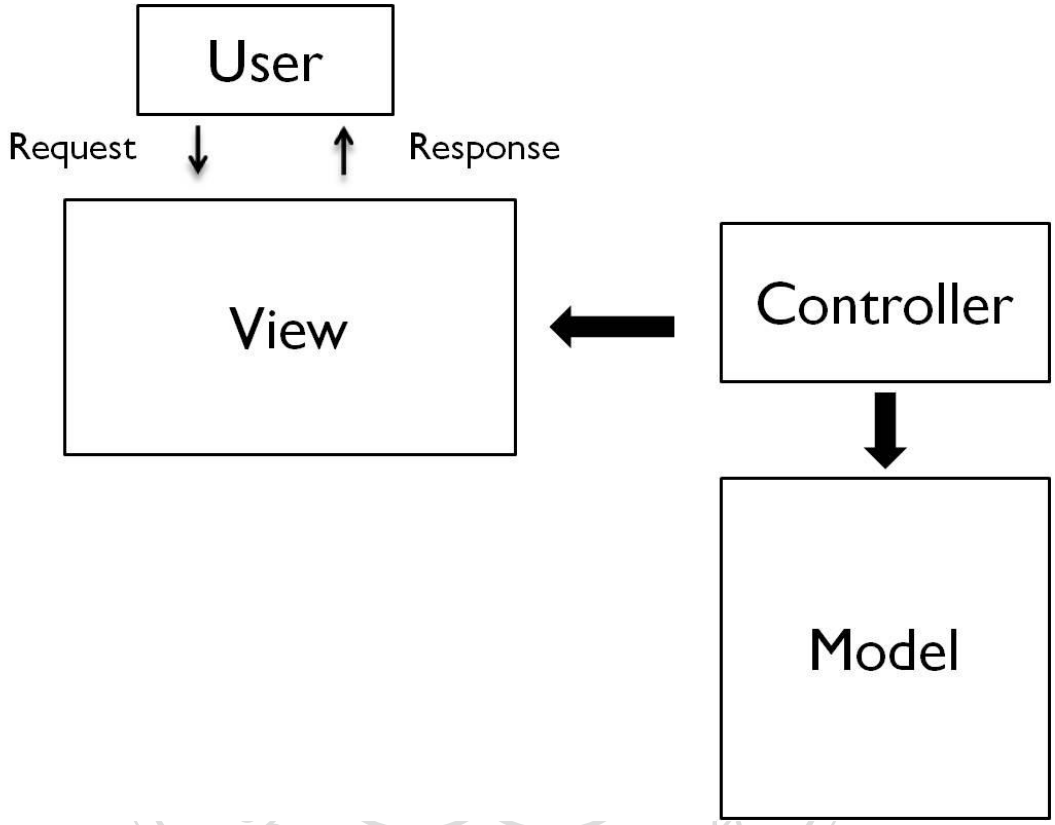

圖 十五 傳統 MVC 應用框架

在傳統框架中使用者的請求經由 View 傳至 Controller，由 Controller 改變 Model 狀態，接著再通知 View，最後由 View 回應給使用者。View 的主要功能為前端使用者介面的呈現；Controller 是負責轉發請求，對請求進行處理；而 Model 是包含邏輯運算、演算法、數據管理和資料庫設計等。

(二) 2-level MVC 應用框架

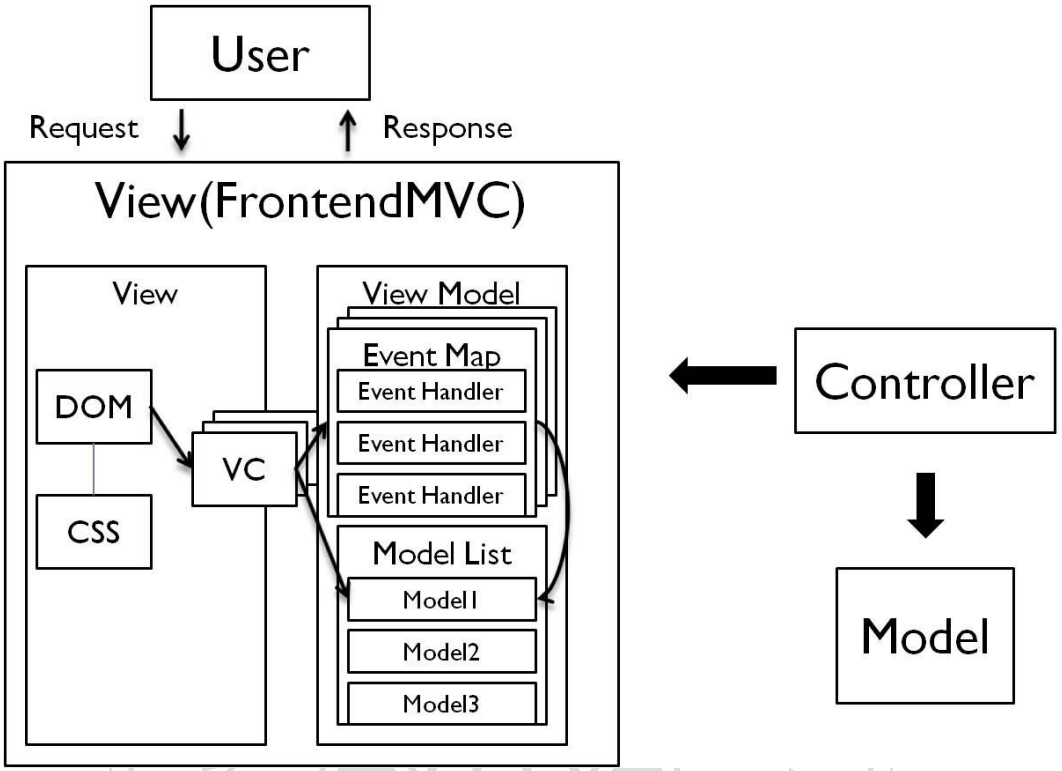

圖 十六 2-level MVC 應用框架

相對於原本傳統 MVC 架構，2-level MVC 應用框架(兩層式 MVC 應用框架)是在傳統 MVC 的 View 裡面在放一層 MVC，如圖九所示，此層 MVC 稱為 Frontend MVC。

Frontend MVC 的組成為 View、ViewController 以及 View Model，使用者在介面操作送出請求，會把請求交給 ViewController，而 ViewController 會連結一組 View Model，此 View Model 包含一組 Event Map 以及一個 Model List 裡面的 Model，因此 ViewController 會在 Event Map 裡面登記許多 Event Handler 並且對應至該 Model。在整個 Frontend MVC 裡面會有多個 View Controller 並對應至多組 Model，這麼做的好處是在客戶端的程式架構可以

定義更清楚，並且把前端程式對於 DOM 的操作用完整的架構儲存起來。

前端架構使用 Frontend MVC 的架構，接著就能將狀態移轉圖(來自使用者案例)轉換成 Frontend MVC 的標準訂定出來，利用 ViewController 對 DOM 的操作能力以及事件處理和模型登記來對應狀態移轉圖中的狀態。詳細轉換標準會在下個章節中呈現。

(三) 系統架構

圖十為建立在 2-level 應用框架、JavaScript 應用框架與 Hadoop 架構下之系統架構。

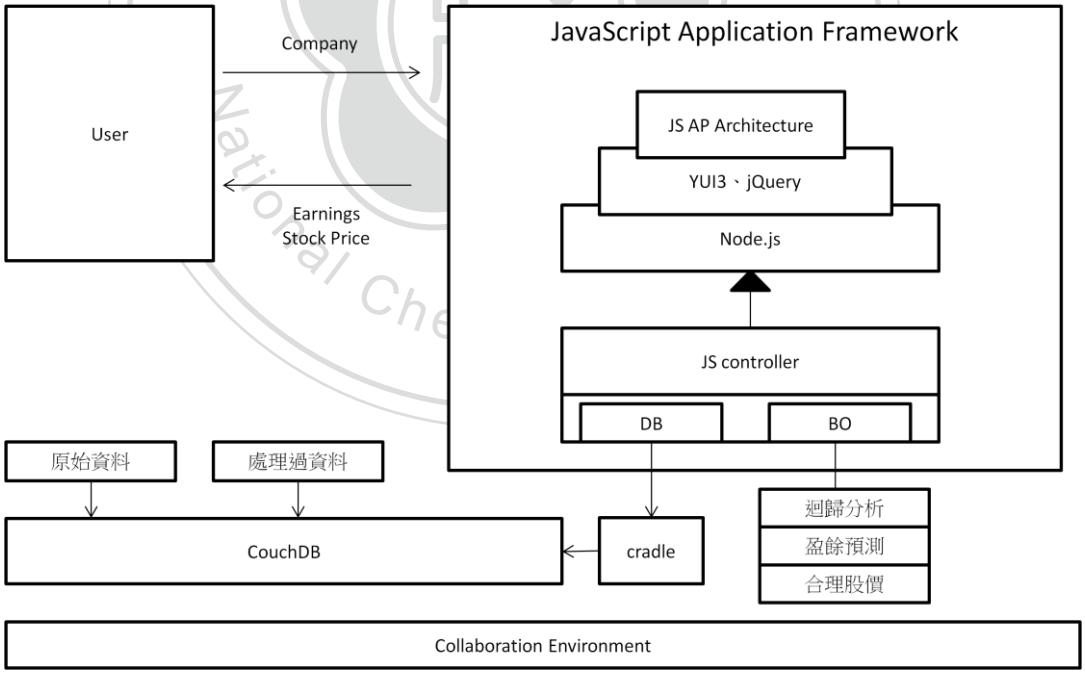

圖 十七 系統架構

由系統架構可以看出，使用者送出公司資訊後，會進入 JavaScript Application Framework(JavaScript 應用框架)，在此框架中客戶端有

JavaScript 架構以及 YUI3、jQuery 來處理，伺服器端有 JavsScript Controller 透過 Node.js 來運作，JavaScript Controller 有兩個組成單元，稱作 DB 單元以及 BO 單元，DB 單元透過 Node.js 的模組 cradle 來跟 CouchDB 做連接，此模組是專門為了 Node.js 連接 CouchDB 而開發的，另外 BO 單元負責處理商業邏輯的部份，像是迴歸分析、盈餘預測以及推測合理股價。因此前端接收到公司資料之後，先把公司資料傳進資料庫中取得相對應的公司會計科目資料，再將許多年度的會計科目經由 BO 單元做迴歸分析，分析完之後將盈餘與股價資料回傳至使用者介面，除了一般傳統電腦的使用者介面以外，還可基於 JavaScript 應用框架將資料在伺服器端執行完成之後以 HTML 形式回傳至前端，如此還能將資料輸出至智慧型手機或是平板電腦等運算功能較差之雲端裝置。

肆、 系統建置與研究結果

一、 系統實作與建置

(一) 客戶端

在客戶端方面，採用了網頁設計基本的 HTML 與 CSS Style 做為使用者介面的排版，功能方面加入了 jQuery 與 YUI3 兩種語言來撰寫。公司搜尋條件使用了 jQuery 中的 autocomplete，最後資料呈現在畫面上則使用 YUI3 來當作填寫工具。

(二) 伺服器端

在伺服器端採用了 Node.js 當作開發工具，使用的是純粹的 JavaScript 語言。在系統裡面會用到以下功能，index 為 Node.js 驅動的 js 檔案，server 主要功能是開啟伺服器端網頁，router 為接收 URL 所傳入之參數，另外還有 requestHandlers 是前端主要呈現的模組，使用者介面以及執行函數接在這個模組裡面執行。

前面有介紹過，Node.js 是以模組為基底的，因此模組的建置與使用就變得相當重要，這裡提出本系統的其中一個模組當範例來說明如何建立與運用模組，首先建立一個 requestHandlers 的 js 檔，由下圖所示能看到此 js 檔案中包含兩個函數，start 與 upload：

```
function start(response, postData) {
  console.log("Request handler 'start' was called.");

  var body = '<html>'+
    '<head>'+
    '<meta http-equiv="Content-Type" content="text/html; '+
    'charset=UTF-8" />'+
    '</head>'+
    '<body>'+
    '<form action="/upload" method="post">'+
    '<input type="text" name="text"></text>'+
    '<input type="submit" value="Submit text" />'+
    '</form>'+
    '</body>'+
    '</html>';

    response.writeHead(200, {"Content-Type": "text/html"});
    response.write(body);
    response.end();
}
```

圖 十八 模組中 start 函數

```
function upload(response, postData) {
  console.log("Request handler 'upload' was called.");
  response.writeHead(200, {"Content-Type": "text/plain"});
  response.write("You've sent the text: "+
  querystring.parse(postData).text);
    db.get(querystring.parse(postData).text, function (err, doc) {
        console.log(doc._id);
        console.log(doc.ACCRUAL_200103);
        console.log(doc.ACCRUAL_200106);
        console.log(doc.ACCRUAL_200109);
        console.log(doc.ACCRUAL_200112);
    });
  response.end();
}
```

圖 十九 模組中 upload 函數

函數內容不是本研究重點，重點是在 requestHandlers 這個模組中建立此二函數之後，要如何讓這個模組與其函數成為真正的模組讓其他 js 檔使用呢？就是利用 export，利用 export 將函數匯出即可，要將 requestHandlers 的 start 與 upload 函數匯出即如下圖所示：

```
exports.start = start;
exports.upload = upload;
```

圖 二十 利用 export 匯出模組函數範例圖

只需要短短一行：exports.函數名稱 = 函數名稱，即可將函數匯出，此模組與其函數此時已經可以供其他 js 檔使用。接下來介紹如何使用模組，使用 require 函數將模組路徑即名稱以參數方式丟入可以引用，如果是 Node.js 內建模組則只需要名稱不需路徑，以下圖為例子說明：

```
var http = require("http");
var url = require("url");
var server = require("./server");
var router = require("./router");
var requestHandlers = require("./requestHandlers");
```

圖 二十一 模組引用說明範例圖

依上圖所示，上圖為 index 首頁的模組引用範例，從圖中可以看出 index 頁面裡用 require 函數引用五個模組，分別是 http、url、server、router 以及 requestHandlers，前兩個 http 以及 url 是 Node.js 內建模組，因此參數不需要加上路徑即可成功引用，而後面三個 server、router 與 requestHandlers 是內建模組，在引用內建模組是必須要有路徑位置與模組名稱，路徑位置可以是絕對位置也可以是相對位置。從上圖中成功引用五個模組，在與前面範例介紹的 requestHandlers 結合說明，接下來只需要下指令 requestHandlers.start()就可以執行 requestHandlers 中剛剛建立的 start 函數。

(三) 資料庫端

因為 CouchDB 的特色有內建備份機制，支援離線儲存與線上同步以及文件為儲存基礎等特性，所以這個小節主要是針對 CouchDB 的這兩個特色做討論。

1. CouchDB 同步機制

在講解同步機制前，先放上 CouchDB 簡易結構圖如下：

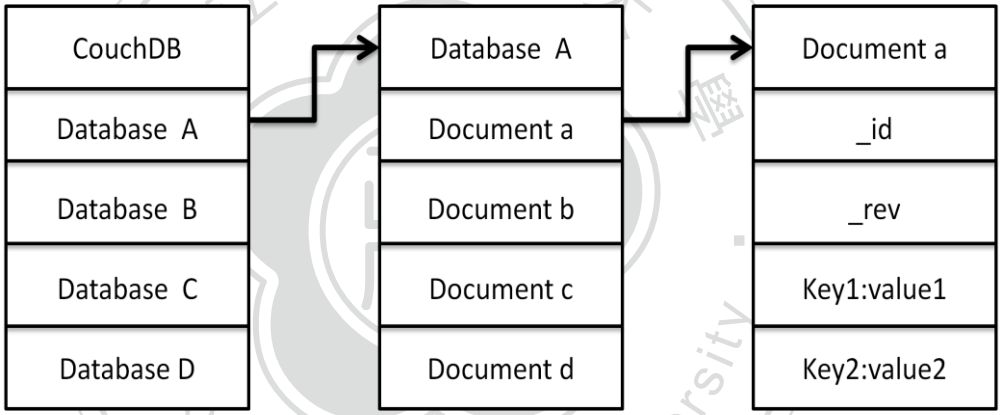

圖 二十二 CouchDB 簡易結構圖

簡易結構圖如上，我們可以看到 CouchDB 是可以存取許多個資料庫，也就是 CouchDB 是多個資料庫所組成的。而一個資料庫 A 是由若干個文件(Document)所組成，一個資料庫可以存放許多的文件，然一個文件 a 裡面組成元件有文件編號、文件版本號以及文件內容包含 key 以及 value 若干組。而了解 CouchDB 結構後，我們來看看許多客戶端主機與一台主機想要將所有資料同步該如何處理。

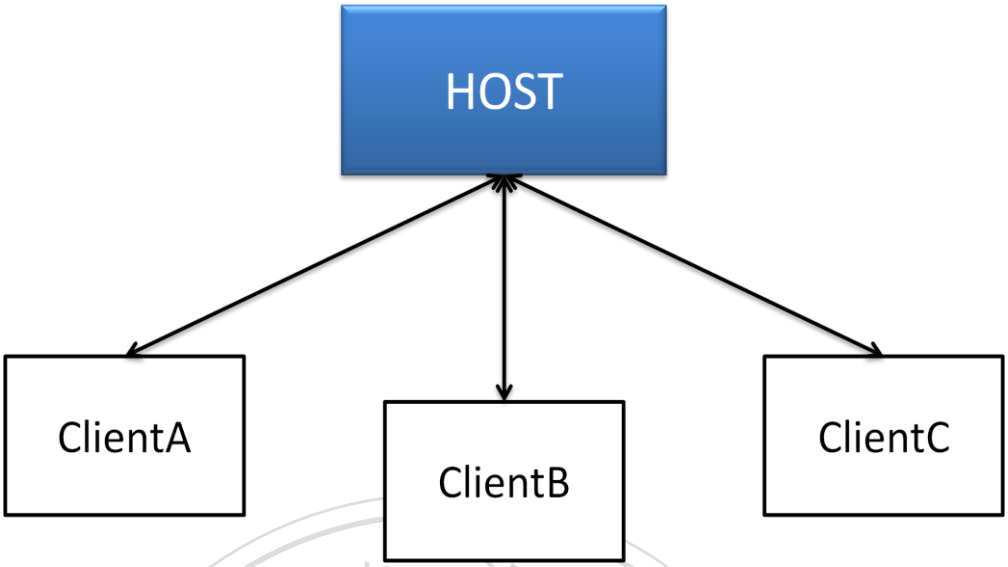

圖 二十三 多台客戶端主機與主機連接示意圖

客戶端 A、B、C 與 HOST 皆為雙向同步，因此可以想像成資料在彼此之間會完全互相傳遞，接下來看這其中是如何運作。假設客戶端 A 更新一筆資料，因為客戶端 A 有箭頭指向 HOST，也就是客戶端資料會與 HOST 同步，同步完成後，因為 HOST 資料發生改變，便會通知客戶端 B 與客戶端 C，接著將資料同步於客戶端 B 與 C，經過這樣的同步程序後，客戶端 A、B、C 與 HOST 就會保持資料一致性，不論是哪個客戶端對資料庫更動一筆資料，所有與其連線的主機將會與該資料庫進行同步。

而事實上同步指的是資料庫間的同步，而且不限定資料庫名稱、使用者資料等條件，更詳盡的同步示意圖如下：

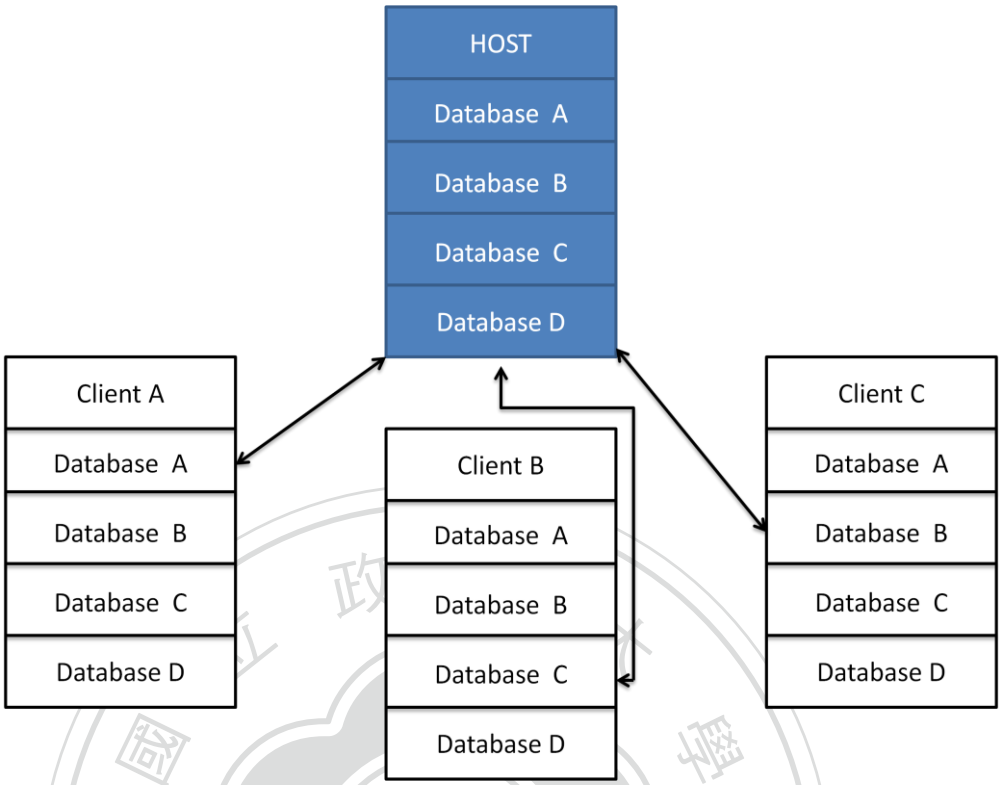

圖 二十四 CouchDB 同步機制詳細示意圖

如上所述，同步不限定資料庫名稱與客戶端用戶，因此從圖二十四中可以看出，其同步可以是客戶端 A 的資料庫 A 與 HOST 端的資料庫 D 以及客戶端 B 的資料庫 C 和客戶端 C 的資料庫 B 互相同步，且不受到其他相同命名資料庫之干擾，也就是只要設定好，任何名稱的資料庫不論幾台都可以互相同步，這也是 CouchDB 同步機制中很強大的一部分。

從剛剛到現在，我們討論的都整個資料庫的同步，但是其實 CouchDB 的同步機制還有類似篩選器(Filter)的功能，資料庫與資料庫間可以只保有一部分文件同步，以下圖說明之：

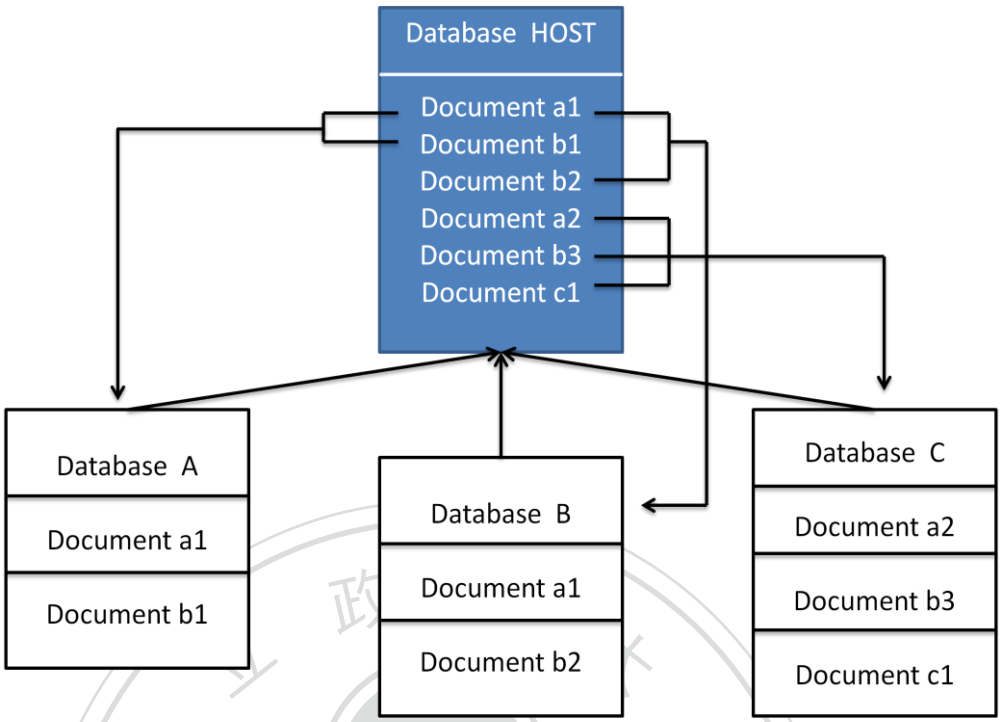

圖 二十五 CouchDB 同步機制加上篩選器

上圖表示資料庫 A 的文件 a1 與 b1 與資料庫 HOST 的文件 a1、b1 進行雙向同步；而資料庫 B 的文件 a1、b2 與資料庫 HOST 的 a1、b2 做雙向同步；資料庫 C 的文件 a2、b2 以及 c1 與資料庫 HOST 的文件 a2、b2、c1 做雙向同步。雖然資料庫 A、B、C 總共有七個文件總數，但是由於資料庫 A 與資料庫 B 中都存有文件 a1，所以扣除重複的部份總共有六種文件，也就是資料庫 HOST 的文件總數。資料庫 C 的文件並未與其他資料庫相同，因此可以看做是整個資料庫同步(即使資料庫 C 還有其他文件也可不進行同步)，資料庫 A 與資料庫 B 有相同的文件 a1，可看做未加入篩選器時的同步機制，不論誰更新都會同步至資料庫 HOST 接著再同步至另外一台資料庫，此外，資料庫 A 與資料庫 B 都分別擁有自己才有的文件，分別是 b1 與 b2，此資料會與資料庫 HOST 進行同步更新但不受其他資料庫干擾。

2. 傳統關聯式資料庫轉換文件導向資料庫之規則

由於 CouchDB 是 schema-less 的資料庫，但所謂 schema-less，其實就是指沒有 schema，當然也沒有傳統的關聯式資料庫裡面合併(join)的概念。然而，儘管 CouchDB 本身沒有提供這樣的支援，真的需要用到關聯式表格的時候還是能使用一些小技巧，像是重新定義 schema，利用新的文件加上 View 就可以做到合併(join)的用法。在本小節會舉簡單的例子稍作說明。

Student

ID	name	city	company
1	Eric	Taipei	AAA
2	Brian	Taiachung	BBB
3	Sogas	Taipei	CCC

表 三 學生資料表

Company

company_name	address	tel
AAA	a Rd.No10	0912345678
BBB	b Rd.No59	0987654321
CCC	c Rd.No133	0911111111

表 四 公司資料表

上面兩張分別是學生資料表以及公司資料表，在關聯式資料庫中，若要查詢該學生所在的公司其電話及地址變可使用 join 的觀念將兩張表格合併，不過在 CouchDB 中並沒有這樣的概念，那麼要如何做這樣的功能呢？先從下圖左邊看起：

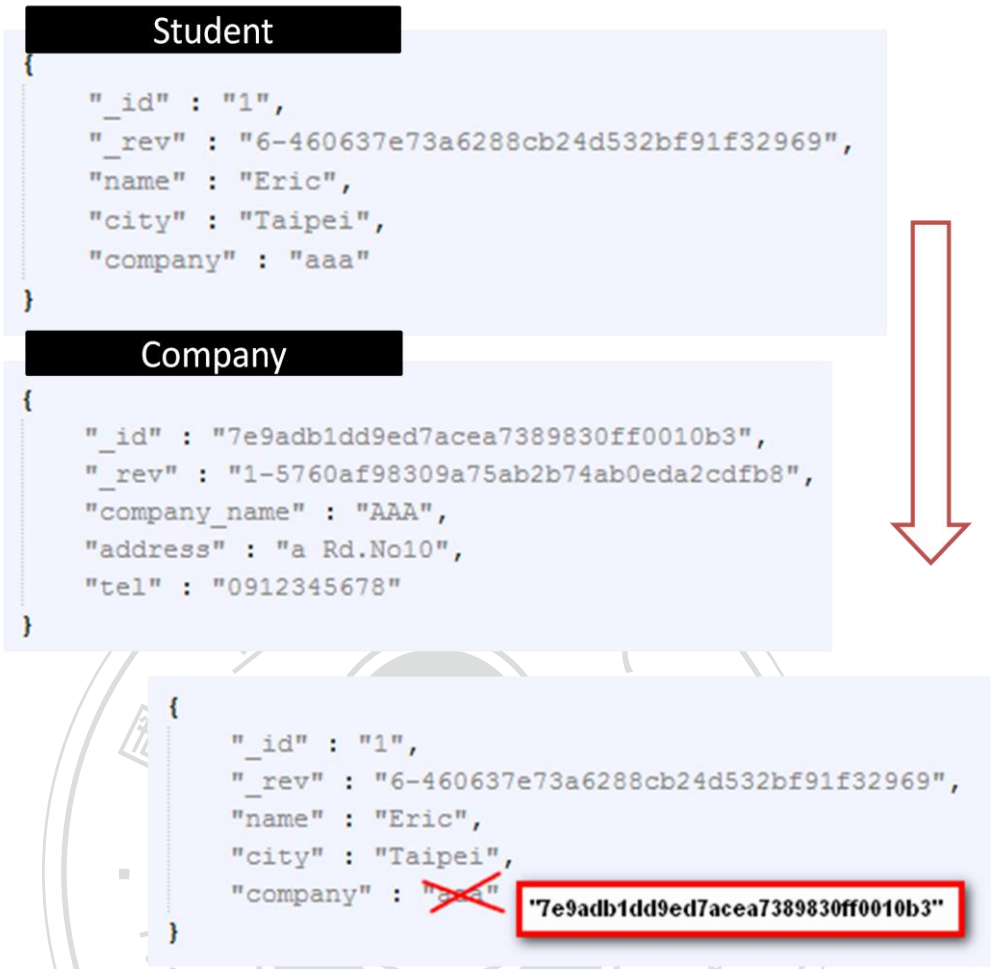

圖 二十六 簡易轉換規則說明圖

上圖可以看到 Student 與 Company 資料表裡的資料格式，以 JSON 格式展示，資料表內有各自的文件編號、文件版本號還有文件自己的屬性。今天如果希望 Student 與 Company 有所連結，就可透過圖二六最下面所示，在 Student 資料格式的 Company 改成儲存 Company 為 AAA 這間公司的文件編號，如此一來只要從 Student 表格中取出 Company 的值(此時為文件編號)，再利用文件編號去找到此間公司的詳細資訊，雖然此方式稍較複雜，但是在沒有 schema 的 CouchDB 中，已經可以利用此方法解決資料關連性的問題。

二、 研究結果

(一) 系統介面呈現

進入系統之後輸入公司名稱或代號，而後經過計算則會輸出公司合理股價。輸入與輸出畫面可在下兩張圖中看到：

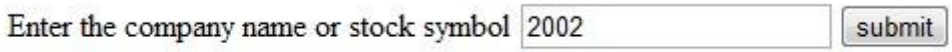

圖 二十七 系統輸入

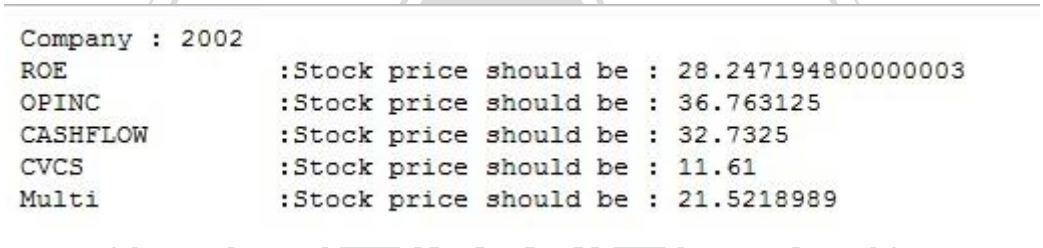

圖 二十八 系統輸出畫面

從圖中可以看到，輸入公司代號之後，系統會輸出根據 ROE、OPINC、CASHFLOW、CVCS 以及多變數迴歸等五種模型之預測盈餘結果，根據五種不同的預測方式產出不同的結果並分別列出。

因為本研究旨在建立 JavaScript Application Framework 與 CouchDB 的連線機制，盈餘預測系統只是拿來實作的範例，因此系統介面並無美化，僅是將連線機制確立並且印出結果。

(二) 系統與研究困難點

本研究有幾個困難點，第一：儘管提供了傳統的關聯式資料庫轉換至文件導向資料庫的規則，但是畢竟 CouchDB 原本就不是設計給複雜資料用的。剛好本研究採用的範例需要的資料是公司的會計科目，並沒有太過複雜的結構，否則複雜架構的資料用 CouchDB 處理會非常困難。第二：由於 CouchDB 特性的關係，資料量只會越來越大，而該如何在版本控制與 Compaction 之間取得平衡，依舊是很困難的。第三：因為本研究採用 JavaScript 應用框架，此為目前較新穎的技術，因此實作上面仍有許多資料搜尋是相對困難許多，但是相信往後技術越來越成熟這方面問題也會逐漸改善。

伍、 結論與未來展望

這份研究主要是利用 JavaScript Application Framework 來實作盈餘預測系統，實作出符合前後端(客戶端、伺服器端)同一程式語言的架構，而底層資料庫技術使用的是 CouchDB，CouchDB 特點有內建強大的備份機制，因而可以做到離線儲存、自動同步等功能。此研究雖然不是建立協同合作的平台，但有特別針對 CouchDB 的同步機制加以說明，也點出 CouchDB 是很適合作為協同合作平台的底層資料庫選擇，但也因為 CouchDB 版本控制機制的緣故，資料量變大會非常占空間也是最大的問題。

本研究最終有實作出以 CouchDB 為資料庫的系統，並且探討 CouchDB 所提供的同步機制與離線儲存技術，能利用這些技術運用到其他地方並且做出實作雲的概念，較可惜的是此次系統並沒有提供完整的協作機制，只是針對這些協作機制做詳細的討論。這次的研究只是一個實作研究，說明以 CouchDB 當做資料庫能符合協作環境的需求，因此若要實作協作雲系統 CouchDB 將是底層資料庫的選擇之一，而盈餘預測系統只是本研究的一個實作範例，並沒有提出更新的架構，往後的研究不論是提出新的盈餘預測方法、利用 JavaScript Application Framework 建置其他系統或是討論更多 CouchDB 的應用這幾個方面，都能有不錯的發展。

參考文獻

1. Apache.(2008).”Hadoop Official Site.” from http://hadoop.apache.org/.
2. NIST.(2010).”Cloud Computing Forum & Workshop” from http://www.amazon.com/Nicholas-C.-Zakas/e/B001IGUTOC .
3. IBM DeveloperWorks.(2009).“Introducing Apache Mahout” from http://www.ibm.com/developerworks/java/library/j-mahout/index.html .
4. Yahoo!Developer Network.(2011.)”Yahoo!Announces Cocktails – Shaken,Not Stirred” from http://developer.yahoo.com/blogs/ydn/posts/2011/11/yahoo-announces-cocktails-%E2%80%93-shaken-not-stirred/.
5. NCHC.(2010).” Hadoop v0.20 + Hbase + thrift + php 教學手冊” from http://trac.nchc.org.tw/cloud/wiki/waue/2010/HbaseThrift.
6. Leslie M. Orchard & Ara Pehlivanian .(2009). Professional JavaScript Frameworks: Prototype,YUI,ExtJS,Dojo and MooTools.
7. Orchard , L. M., Pehlivanian, A., Koon, S., & Jones, H. (2009). Professional JavaScript Frameworks: Prototype, YUI, ExtJS, Dojo and MooTools. Wrox Press Ltd.
8. Zakas, N. C. (2010). High Performance JavaScript. Yahoo Press.
9. Manuel Kiessling.(2011). The Node Beginner Book.
10. Ran Barniv,Mark Myring.(2006). An International Analysis of Historical and Forecast Earnings in Accounting-Based Valuation Models. Journal of

Business Finance & Accounting,33(7) & (8),1087-1109.

11 (Bernstein and Goodman 1981; Wong and Lam 2002; Dekeyser and Watson 2006; Gonzalez, Halevy et al. 2010; Rajam, Cortez et al. 2010; Zhou, Ning et al. 2010; Doelitzscher, Sulistio et al. 2011; Le, Kant et al. 2011)

12 Bernstein, P. A. and N. Goodman (1981). "Concurrency control in distributed database systems." ACM Computing Surveys (CSUR) 13(2): 185-221.

13 Dekeyser, S. and R. Watson (2006). "Extending google docs to collaborate on research papers." Toowoomba, Queensland, AU: The University of Southern Queensland, Australia 23: 2008.

14 Doelitzscher, F., A. Sulistio, et al. (2011). "Private cloud for collaboration and e-Learning services: from IaaS to SaaS." Computing 91(1): 23-42.

15 Gonzalez, H., A. Halevy, et al. (2010). Google fusion tables: data management, integration and collaboration in the cloud, ACM.

16 Le, M., K. Kant, et al. (2011). "Cooperative data access in multi-cloud environments." Data and Applications Security and Privacy XXV: 14-28.

17 Rajam, S., R. Cortez, et al. (2010). E-learning computational cloud (eLC2): web services platform to enhance task collaboration.

18 Wong, R. K. and N. Lam (2002). Managing and querying multi-version XML data with update logging, ACM.

19 Zhou, W., P. Ning, et al. (2010). Always up-to-date: scalable offline patching of vm images in a compute cloud, ACM.

20 Anderson, J. Chris/ Lehnardt, Jan/ Slater, Noah ， Oreilly & Associates Inc(2010).CouchDB: The Definitive Guide.

21 Thompson, Mick ， Oreilly & Associates Inc(2011). Getting Started With GEO, CouchDB, and Node.js.

22. Apache.(2008).”CouchDB Official Site.” from http://couchdb.apache.org/.

23. 張嘉玲，2007，CVCS 模型與 CVCS’模型盈餘預測準確度與資訊內涵之探討，國立政治大學會計學系碩士學位論文。

24. 楊慧怡，2001，長期資料之隨機效果模型分析—公司每股盈餘與財務比率之關連性研究，國立政治大學統計學系研究所碩士論文。

25. 劉錦花、黃耀銜、李仁棻，2009，每股盈餘預測之分析，Journal of Commercial Modernization, Vol. 5, No. 2, 117-131.

26. 張鈺雯，2006，我國上市、櫃公司股東權益報酬率之習性分析，國立雲林科技大學企業管理系碩士班碩士論文。

27. 彭火樹，2005，股價與盈餘關係之研究：盈餘與權益帳面價值定式問題之再考量，The International Journal of Accounting studies，第 40 期，第 60-90 頁。