透過多層級之混合式快取架構來加速大量影像資料庫查詢

蔡明德	鄭毓融	林昀德	蕭一豪
國家高速網路與計算中心 助理工程師	國家高速網路與計算中心 助理工程師	國家高速網路與計算中心 研究員	國家高速網路與計算中心 研究員
a00mat00@nchc.narl.org.tw	1103902@nchc.narl.org.tw	lsi@nchc.narl.org.tw	lhow@nchc.narl.org.tw

摘要

系統發展與資料庫技術對提高資料查詢的執行效率有重大的顯著效果，尤其目前所面臨到的巨量資料處理，從龐大的資料中有效率地取出有用的資訊成為重要的議題，由於近幾年資料產生的快速成長，並且變化性與複雜性越來越龐大，傳統式的存取資料庫的資料的關鍵技術與方法面臨著極大的衝擊，而轉移使用NoSQL資料儲存方法也需付出龐大的系統與程式的變動與轉換[4]，相較於此，本研究在既有系統與程式架構下使用作業系統與計算機中的快取理論再進而提升傳統式資料庫的查詢效率[1,2,3,5,6]，並可廣泛應用於網際網路之傳統資料庫查詢上。

關鍵詞：資料庫、快取、共享記憶體、通用唯一識別碼、鎖。

1.導論與介紹

近年來熱門的IT主題除了「雲端」之外，另一項就是「巨量資料」(Big Data)。目前資料的趨勢為數位資料增長的速度越來越快，巨量資料所衍生的相關資料儲存、查詢、分析等問題都是需處理的重要課題，這些大量與多種類型的資料如何經過資料挖掘(Data Mining)及適當處理後轉化成為有用的資訊，更是「Big Data」受到全世界關注的重要原因。以往只有少數的產業會面臨巨量資料處理的需求，例如氣象預測、基因解碼或是金融交易等等，但隨著科技的進步，有更多的產業也面臨巨量資料分析與處理的難題。網路購物平台 eBay 每天有數百萬次的商品查詢，資料庫每日增加1.5兆筆記錄，而資料庫的總容量則已達到10PB。所以不僅每天新增的資料量眾多，連資料庫也是很龐大，而要從中分析顧客的瀏覽、消費行為就是一件困難的事情；目前大家熟悉的Facebook 社群網站，每天都有數億人使用並留下龐大的資料，其中有很大的比例是圖片、影片等傳統資料庫系統較不常處理的，這不僅挑戰社群網站業者如何管理，對於想利用社群網站來掌握消費者動向的企業而言，也面臨前所未見的挑戰。美國最大的超市 Wal-Mart 過去透過結帳資料分析，將啤酒與紙尿布擺在一起促長了啤酒的銷售量，就是一個很成功的案例，這些 Big Data 的「處理」及「應用」都成為企業經營非常重要的課題。

為了解決 Big Data 上資料庫大量資料存取的問題，很多公司近年來紛紛捨棄了關聯式資料庫技術，改以 NoSQL 資料庫來提升效能與擴充彈性，NoSQL 資料庫是一個統稱的名詞，泛指非關聯式資料庫的資料庫技術，包括了數種不同類型的資料庫系統，因為這些資料庫大多沒有支援標準的 SQL，例如知名的 Google BigTable、或是微軟 Azure 平臺儲存資料的方

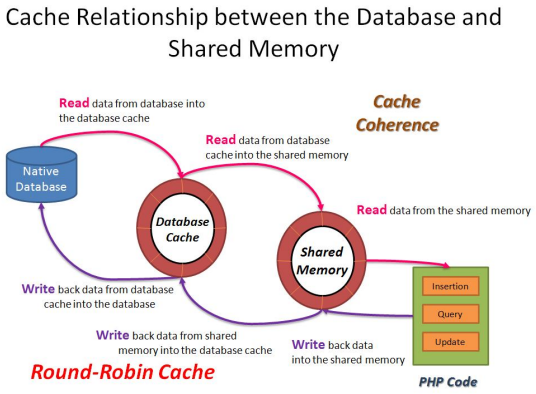
式。關聯式資料庫必須透過資料庫的結構(Schema)來確立資料表之間的欄位關聯，資料庫結構通常是事先設計好的資料表與欄位，上線之後要進行欄位變更非常麻煩，尤其資料量龐大時要變更資料庫結構的難度很高，例如 Twitter 為了調整資料欄位，光是執行 Alter Table 指令來改變資料表的定義就耗費了一個星期完成。NoSQL 資料庫則是改用 Key-Value 的資料模式來解決龐大資料的異動困難。Key-Value 模式是將一筆資料的結構簡化到只有一個 Key 值對應到一個 Value 值，每一筆資料之間沒有關連性，所以可以任意切割或調整，也可以分散到不同的伺服器中建立。

但是要從 SQL 轉換成 NoSQL 的架構，在人力與成本上的耗費都非常龐大，本研究即針對原本的關聯性資料庫加以改良，在不影響原本的 SQL 查詢架構下，使用關鍵的索引值與階層式快取等方式來加快資料庫查詢的速度，進而提升資料庫系統的查詢效率。

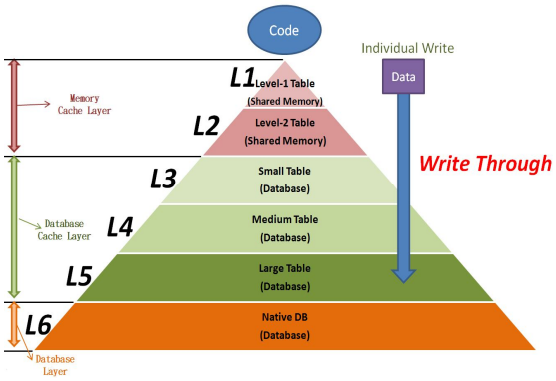
2.相關研究與系統設計

資料庫的使用隨著資料筆數的遞增，查詢所花費的時間會相對的加長，本研究為針對如何提昇資料庫查詢的效能所提出之方法，包含 1.使用 UUID 做為查詢的關鍵索引來簡化查詢，加速多個資料表之間的關連與多個欄位的比對。 2. 使用多層級(5 層)快取方法來提高查詢效能，快取中包含使用共享記憶體(shared memory)來減少對硬碟的 I/O 進而提升整體 I/O 速度。3. 除了增加多層快取儲存裝置外，並進一步使用客製的快取資料取代的演算法來提高快取查詢的命中率。

本研究改善資料庫查詢效能的方法之一，就是對查詢時所使用的關鍵資料產生通用唯一識別碼 Universally Unique Identifier (UUID) 做為資料庫查詢中唯一識別的資訊或索引。傳統上，跨多資料表與欄位的複雜查詢提高了對資料庫的載荷，並加重對 I/O 的使用，大幅影響資料庫查詢效能。使用索引可以幫助資料庫快速查詢資料，一般傳統的資料庫的索引使用二元樹排序索引值，加快對索引值的搜尋。本研究中，對最常用到的欄位資料做 MD5 加密產生固定長度的 UUID，並在資料表中儲存 UUID 並作為索引以減少資料表查詢中對多個欄位的比對，進而提升查詢效率，快速返回查詢結果。



圖<1>. 階層式快取關係圖



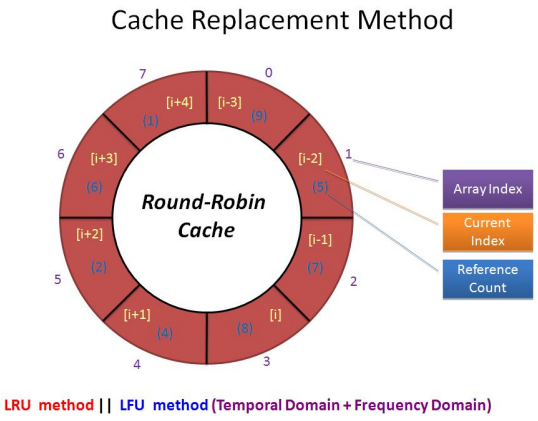
圖<2>. 資料庫五層式快取架構圖

除了使用 UUID 作為索引的方法外，本研究也將計算機中階層式記憶體的概念應用在資料庫查詢，並採用五層的快取的架構，[8] 如圖<1>與圖<2>所示。在程式對資料庫進行資料插入(insert)時，除了在資料庫寫入資料

外，同時也將資料依序寫入五層快取的 Table 中，對於資料的查詢(select)時，也依序將查詢到的資料寫回快取中。在快取中的資料替換機制(Replacement Algorithms)，採用巡迴式佇列(Round Robin Queue)的方法，更新五層快取內的資料。五層式快取中同時使用記憶體作為 Layer 1 與 Layer 2 快取的儲存媒介，而 Layer 3、Layer 4 與 Layer 5 快取則使用資料庫的資料表，各層的快取大小依序從 Layer 1 至 Layer 5 遞增。因記憶體的 I/O 速度遠大於存放在硬碟的資料庫 I/O，可以大幅提升對資料的搜尋與讀取時間。而使用資料庫的三層快取由於各自最大的快取資料筆數固定，並且資料筆數會比儲存大量資料的實際(Native)資料庫少上許多，所以也大幅減少搜尋所有資料的時間，並且資料庫快取大小大於記憶體中的快取大小，提高了在資料庫快取中搜尋資料的命中率。[10,11,12,13]五層式快取是一種混合(Hybrid)快取，應用不同的儲存媒介的優點與設計，縮短查詢時間，提高資料庫存取效率。

當對資料庫查詢資料時，程式先由五層式快取中依序從最上層快取(Layer 1)開始搜尋至最下層快取(Layer 5)，當資料於快取區中被找到或稱快取命中(Cache Hit)，程式直接從快取區中讀取資料並回傳至客戶端使用，如此可避免對實際資料庫查詢時的大量 I/O 次數，加快查詢效率。由於設計了多層的快取區，當資料沒有再上層的快取中被找到，程式將對下一層的快取區中搜尋。當如果資料不在快取區中，無法從快取中搜尋到資料時則稱為快取失誤(Cache Miss)，當快取失誤發生時，需要付出失誤代價(Miss Penalty)，付出額外對下層快取的搜尋時間及上層快取的資料替換時間，但實際失誤代價的時間會因儲存裝置的設計與快取資料替換方法有所差異。

但就對資料的查詢加速而言，無論是使用記憶體或是資料庫做為快取，其實僅在「查詢」層面上，加快了對資料的取得時間，然而一旦涉及到將資料做「寫入」或「更新」的行為時，需要對快取進行「回寫」動作，將新的或更新的資料寫回至各層快取，以保持快取與實際資料庫中的資料一致姓(Consistency)，也因此影響對資料的查詢效能。



圖<3>. 使用 Round-Robin Cache

在五層式快取中，由於各層快取容量大小固定，當快取空間已滿時有新的資料要在加入時，必須使用一套資料替換的機制，如圖<3>所表示的巡迴式佇列(Round Robin)是最基本的取代機制，對資料做依序性的取代。此方式會記憶一計數用的數值(Current Index)用來對應快取中資料的記錄位置，當寫入資料時，會依據此數值寫入資料至對應的紀錄位置，並計數值加一，當最後計數值達到限定的最大值(Maximum Index)時將計數值歸零，下筆資料將從第一個資料記錄位置寫入，覆蓋原有的資料，並再增加計數值。

根據局部性原則(Principle of Locality)，分成時間局部性與空間局部性。時間局部性使用表示快取位址剛被存取後不久可能還會再被存取，而空間局部性使用表示剛被存取的快取位址的附近位址可能很快將

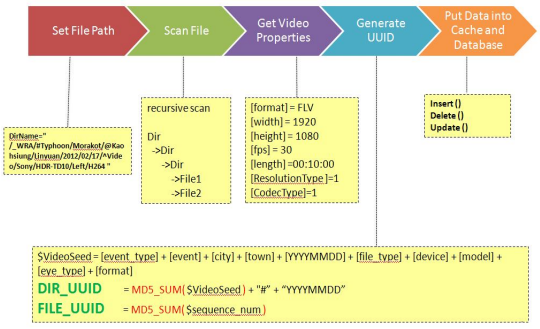
會被存取。對於查詢資料時的快取使用，LRU 應用了時間局部性原則。

針對時間局部性原則，進一步研究 LRU 取代的方法，LRU (Least Recently Used)是將最近最少使用(或最近最久沒有使用)的資料進行取代動作，快取區將會保留最常使用的資料而置換不常使用的資料。一般實現 LRU 的方法是追蹤每筆資料最近的參照(使用)時間，時間最久(最早)的資料將被取代。決定最後資料使用時間的排列順序可以使用多種方式來實現，如計數(Counter)、堆疊(Stack)或年齡位元(age bit)。另一個替換機制 LRU 與 LFU (Least Frequently Used) 相似。LFU 表示使用頻率最低的快取資料會優先被取代。實現 LFU 的方法是紀錄快取資料的使用頻率(次數)。但 LFU 的缺點是缺少時間性的判斷資料的使用先後[7,14]。

我們將同時應用時間性與空間性兩者兼具的快取替換原則來實現最佳的替換法則。資料替換將優先以使用頻率來決定替換資料，如果發生相同使用頻率於多筆資料中，再以時間最早(最舊)的資料做替換，並且可分別在不同層次的快取中套用不同的快取替換機制，提升快取的使用效率以達到最佳的快取加速效果。

階層式快取可以加快資料庫搜尋的速度，但是當使用 Shared Memory 當成快取裝置時，我們需要注意到多人同時進行資料庫查詢時，就會產生競賽情形，Shared Memory 的內容更新同步與快取的一致性就需要進行驗證的動作。本研究將 Lock 的機制應用到 Shared Memory 快取上，當有資料寫入時，對該寫入位置或快取進行 Lock 的動作，暫停其他相同位置的資料寫入，以防止多人同時寫入造成資料混亂或錯誤的狀況發生。對於快取資料的 Lock 機制，將進一步討論 Lock 機制的改良方式，使得使用多層快取來加速資料庫搜尋的研究方法可以更加的完善。[9]

3. 系統實作與方法



圖<5>. 資料插入與更新流程圖

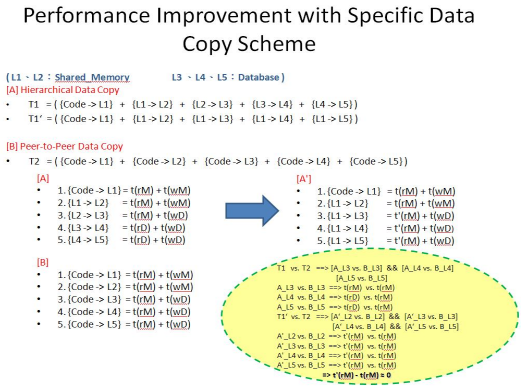
以水利署計畫為例，針對水利署計畫所建立的資料庫，本研究實作加速資料庫查詢的應用。水利署資料庫儲存多種檔案資訊，並且檔案系統中以檔案的相關資訊分層建立檔案路徑，檔案名稱則再以部分檔案資訊結合序號(Sequence Number)命名。圖<5>表示產生 UUID 的方法與資料插入與更新的流程，首先以檔案路徑中的的事件類型(event_type)、事件(event)、縣市(city)、鄉鎮(town)、年月日(YYYYMMDD)、檔案類型(file_type)、裝置(device)、型號(model)、拍攝方式(eye_type)、格式(format)資訊組成 Seed 後做 MD5 演算，並再結合年月日產生路徑 UUID (DIR_UUID)，另外再根據檔案名稱的編號(Sequence Number)做 MD5 演算產生檔案 UUID (FILE_UUID)，以兩個 UUID 組合做資料庫查詢的關鍵索引。

快取的更新有兩種模式，一種是直寫(write-through)，另一種是回寫(write-back)。直寫的方式是當資料寫入實體儲存(實際資料庫)時或更新至快取時，資料同時一併寫入快取與實體儲存中，而回寫則是當

資料在新增到或更新到快取時，不會同時寫入或更新至實體儲存，在快取中的資料視為不完全的或不良的資料(Dirty Data)，當達到一定的門檻(threshold)時再將快取的資料更新至實體儲存中。目前水利署資料庫的資料插入與查詢採用直寫的方式來更新快取。

為了能夠將過去查詢或新增過的資料在快取中找到，需將最新的資料更新或寫入至快取中。在新增資料至資料庫時，資料由第一層快取開始依序寫入至第五層快取，最後再寫入至實際資料庫中，反之，當查詢資料時如果資料最後在實際資料庫中被找到，資料再從第五層依序寫回第一層快取，以保持快取資料的一致性與正確性。由於快取從第一層至第五層遞增，在更新快取資料時，上層快取資料可被其下層快取中找到，當資料於快取中被找到，資料亦將依序寫回至最上層快取(L1)。如果資料即可在最上層快取中找到，則沒有資料寫回上層快取發生。

五層式快取以階層的觀念依序使用各層快取，但是因為記憶體與資料庫快取的儲存媒體不同(RAM 與 HDD)，在資料寫入或讀取時速度的差異性會非常的明顯，資料在各層快取中有四種 I/O 方式：(1). Memory to Memory (2). Memory to Disk (3). Disk to Disk (4). Disk to Memory。由於 Memory 與 Disk 兩種儲存裝置的 I/O 速度差異極大，因此我們可以透過減少 Disk 的 I/O 來提升整體的速度。



圖<7>. 進階的快取複製方式

圖<7>表示兩種資料複製的模式，以下為兩種模式的說明：

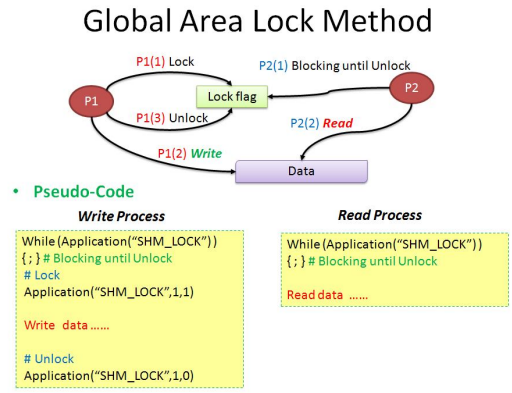
[A]. 階層式資料複製(Hierarchical Data Copy) – T1，其資料的複製過程包含 1. 程式記憶體複製到 L1 快取；2. L1 快取複製到 L2 快取；3. L2 快取複製到 L3 快取；4. L3 快取複製到 L4 快取；5. L4 快取複製到 L5 快取。其中的第 4 和第 5 步驟為 Disk to Disk 的複製動作，所需耗費的 I/O 時間較久，因此我們進行了改善的方法 T1’，把 Disk to Disk 的方式改為 Memory to Disk，即 L1 快取複製到 L4 和 L5 快取。

[B]. 點對點式資料複製(Peer-to-Peer Data Copy)– T2，其資料的複製過程包含 1. 程式記憶體複製到 L1 快取；2. 程式記憶體複製到 L2 快取；3. 程式記憶體複製到 L3 快取；4. 程式記憶體複製到 L4 快取；5. 程式記憶體複製到 L5 快取。在這兩種複製的模式中，T1’ 與 T2 所進行的儲存裝置間的複製是相同的，但是 T2 少了對 L1 階層資料讀取的程式動作，相對來說速度會比 T1’ 快。

在共享記憶體快取中，可能發生資料的競賽狀況，本研究採用鎖定(Lock)的機制來解決資料的正確問題。使用鎖定的原理是當程式在使用共享記憶體快取的資料時，其他的程式

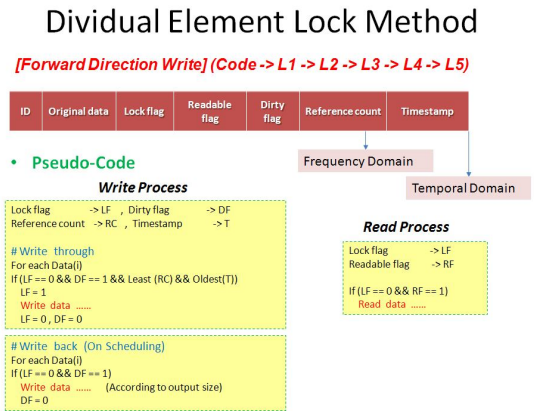
皆暫時無法使用共享記憶體快取內的資料，直到程式完成使用後，其他程式才可進行使用，行成一種有互斥行為的資料鎖定(Mutually Exclusive Lock)。實行鎖定的方法為使用旗標(flag)，旗標只有兩種狀態:鎖定(locked)和非鎖定(unlocked)，如位元(bit)的 0 和 1。當對快取資料進行存取時，如果 flag 狀態為非鎖定時，則將狀態設為鎖定並開始存取，當完成存取時，設回非鎖定狀態。

資料的讀取和寫入優先不同會有不同的快取效果影響。當使用寫入優先時，程式解除鎖定後，其他做寫入的程式會優先取得鎖定，讀取的程式則至最後才能取得鎖定，這樣確保程式所讀取到的資料是最新的，但缺點是寫入的動作很頻繁時，做讀取的程式必須等待的機會將增多，相反的若設定為讀取優先，則讀取時的回應性會增高，但資料更新的速率將會下降，使用讀取優先或是寫入優先視應用場合而定。另外對資料或快取的鎖定有兩種，一種是全域鎖定(Global Lock)，另一種是區域鎖定(Local Lock)，亦可稱分開鎖定(Dividual Lock)，下面分別說明兩種鎖定。



圖<8>. Global-Lock Method

圖<8>表示全域鎖定的流程與虛擬程式碼(Pseudo-Code)，使用全域鎖定時，全部快取資料將同時被鎖定，同一時間只有一個程式可以讀取或寫入快取資料，這樣對多個程式同時對快取的使用非常沒有效率，上圖為全域鎖定的設計，程式 1(P1)在執行鎖定時，將鎖定旗標設為 1(鎖定狀態)，當程式 2(P2)進行資料的讀寫時，檢查鎖定旗標，如果檢查結果其值為 1，則進入空迴圈無限循環，直到程式 1 將鎖定旗標設為 0(非鎖定狀態)，使程式 2 得到鎖定旗標檢查結果為 0(非鎖定狀態)，跳出迴圈，程式 2 繼續往下執行。



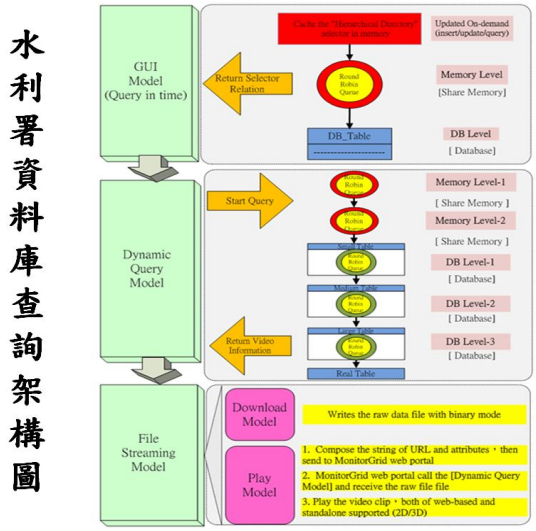
圖<9>. Dividual-Lock Method

區域鎖定則縮小了鎖定範圍以解決效率問題，對快取中個別獨立的資料做鎖定，其他資料不受影響，使其他程式同時可對其他資料進行讀寫，減少了競賽狀況的發生。藉由區域鎖定不但可以解決並行處理中的資料正確性和完整性，也使共享記憶體的使用率不會大幅下降。圖<9>表示區域鎖定的實作方法與虛擬程式碼，在一筆資料(Original data)中堆疊幾個欄位：Lock flag(LF)、Readable flag(RF)、Dirty flag(DF)、Reference count(RC)及 Timestamp(T)。當程式使用直寫模式寫入資料時，需判斷 LF 是否為 0（非鎖定）、DF 是否為 1（資料未與實體儲存同步）、RC 為最小(使用頻率最低)、T 是否為最早(時間點最早)，當符合上述條件時，才可進行寫入。當使用回寫模式時，則只需判斷 LF 是否為 0、DF 是否為 1，即可進行寫入。另外在讀

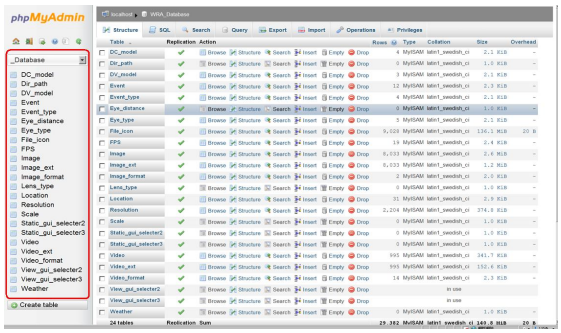
取資料時，則判斷 LF 是否為 0、RF 是否為 1，來決定是否可進行讀取動作。

4. 系統實務應用與範例

應用方面在台灣每年的五、六月梅雨季與七到九月的颱風季除了帶來豐沛的雨量外，有時也帶來嚴重的災情，本研究將 2009 莫拉克颱風後歷年拍攝的災前、災中與災後的圖片以及影片資訊蒐集存放入資料庫內，除記錄相關的歷史影像資料外，並發展防救災資訊的多元性發展多重影像處理技術，利用視覺立體化的影像與可大範圍觀看的全景影像來辨識並判讀資訊，輔助支援防救災與勘災工作。也有拍攝立體的圖片與影片以供進行學術研究試驗，建置多重影像儲存系統，整合過往所蒐集之資訊，可有效保存大量資料，並做為珍貴的歷史記錄。相關人員可經由上述改善資料庫查詢方式所建立之系統來查詢下載所需之檔案資料。同時結合多重影像技術，開發立體動畫影片，進行防災教育之宣導與推廣，加強民眾的防救災意識觀念，進而了解政府相關單位在防救災上所做的努力。復考量此系統技術運用發展迅速，並非單純學術研究，未來應用範疇將可推廣至各項防救災工作。



圖<10>. 資料庫查詢架構圖



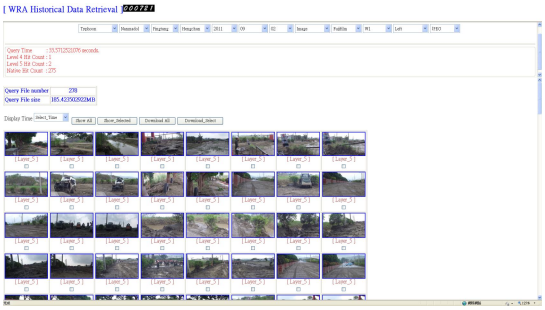
圖<11>. phpMyAdmin 資料庫

整個系統包含資料蒐集、資料庫建置與查詢系統程式撰寫，資料輸出展示平台等三個步驟，將所蒐集之防救災之多重影像資訊以事件、地點、日期等相關資訊儲存於資料庫中，如圖<10>為水利署資料庫查詢架構圖，圖<11>為使用 phpMyAdmin 建立的水利署資料庫。過往之計畫已協助署內蒐集超過數千張之立體影像資料，包括廬山地區、88 水災受災地區、各地重大工程建設(如雲林湖山水庫、屏東縣荖濃溪舊寮一號堤防復建工程、高雄縣旗山溪旗山堤防復建工程與南投陳有蘭溪明德堤防復建工程等)，資料量相當豐富且龐大，新年度仍將繼續進行多重影像資訊之蒐集，此大量資料必須進一步有效管理。規劃將依種類(立體影像、立體影片、全景影像)、事件(88 水災、各工程建設)、地點(高雄、屏東、台東)、日期、內容(水災、堤防、橋樑)等相關資訊進行分類整理，並儲存於資料庫中，有效保存此珍貴之歷史資料，並提供使用者更有效率的使用這些資料。

資料庫查詢部分將所蒐集之多重影像資訊將儲存至資料庫中，並開發一使用者介面提供使用者可即時線上查詢與預覽影像資訊，在查詢方面提供以事件類型(Event_Type)、事件(Event)、縣市(City)、鄉鎮(Town)、年(YYYY)、月(MM)、日(DD)、檔案類型(File_Type)、裝置(Device)、型號(Model)、拍攝方式(Eye_Type)、格式(Format)階層式的

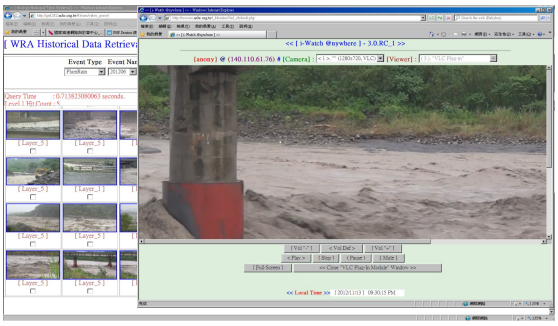
下拉選單讓使用者確認查詢條件，同時將檔案系統中的影像檔案截取縮圖以方便判斷是否為所需影像資訊。

圖<12>. 網頁上進行資料庫查詢畫面 - 檔案再單層快取中擊中



圖<13>. 網頁上進行資料庫查詢畫面 - 檔案再多層快取中擊中

圖圖<13>為實際的網頁查詢頁面之應用，使用者依照所需查詢之條件進行下拉式選單點選後，即可得到所需查詢之檔案縮圖，所查詢的檔案可能皆在單層快取或分別在多層快取中搜尋到。



圖<14>. 查詢得到結果後進行點選播放畫面

圖<14>為撥放檔案的執行結果。使用者依據查詢條件進行資料庫查詢後可先觀看預覽圖示，除檔案縮圖外系統也將縮圖與實際檔案位置進行連結，在網頁上提供檔案播放與下載功能，使用者可進行點選播放來觀看單一檔案或選擇多個檔案內容預覽播放，目前系統總共支援 MJPEG / VLC / SWF / FLV / WMV / MP4/ OGV / WEBM 八種影像格式，使用者可根據適合自己的網路頻寬進行檔案瀏覽，其中 WEBM 格式支援 3D 立體顯示功能，只要有 3D 輸出設備加上支援的瀏覽器即可觀看 WEBM 格式的立體影片。當確認該檔案為所需之資料即可進行選擇下載或全部下載之動作，系統會將選擇檔案用 tar 進行壓縮成 tar 或 zip 檔案，以便將所需資料一次完整傳送給客戶端電腦進行後續處理。

5. 實驗分析與測試結果

針對本研究所導入之 5 層式快取架構，透過一測試程式，組合的四種快取架構進行測試，隨機查詢在各層快取的資料，並計算查詢時間。四種快取架構分別是
(A) 無快取架構：關閉共享記憶體與資料庫快取；(B) 共享記憶體快取架構：只開啟共享記憶體快取；(C) 資料庫快取架構：只開啟資料庫快取；(D)共享記憶體快取+資料庫快取架構：開啟共享記憶體與資料庫快取。

測試程式先隨機取得實際資料庫資料，並持續將不重複的指定資料寫入至快取中，直到使該快取空間使用率達到最大值(滿載)，在開始對該層快取內之資料隨機查詢，計算其查詢時間。查詢時間是以伺服器端執行的五層式快取架構核心的資料查詢處理來計算，不包含客戶端與伺服器端的資料傳送及處理時間。表<1>所示為本實驗測試之資料庫主機硬體與快取架構設定環境。

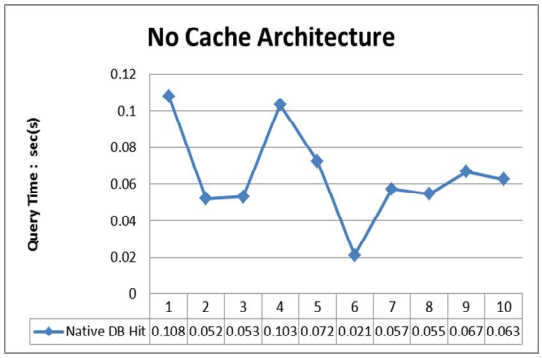
系統環境	CPU: Intel(R) Core(TM) i7-2600@ 3.40GHz
	Memory: 13604864k
	OS: Linux 2.6.25.20 x86_64 GNU/Linux
快取設	L1(memory、Size = 512)
	L2(memory、Size = 1024)
	L3(Database、Size = 2048)

定	L4(Database、Size = 4096) L5(Database、Size = 8192) } 關閉指定快取層時，該層快取資料搜尋及寫入將直接跳過。
資料比數	實際資料庫資料筆數為 50 萬筆。
測試筆數	隨機查詢各層快取之 10 筆資料，亦包含實際資料庫。
測試準備	1. 清空所有 5 層快取資料。 2. 重新啟動 Apache 及 MySQL Database。 3. 執行 sync 指令，同步 Memory 與 Disk 資料。
測試架構分類	(A). 無快取架構 (No Cache Architecture)。 (B). 共享記憶體快取架構(Shared Memory Cache Architecture)。 (C). 資料庫快取架構(Database Cache Architecture)。 (D). 共享記憶體+資料庫快取架構 (Shared Memory + Database Cache Architecture)。

表<1>. 測試主機硬體規格與環境設定

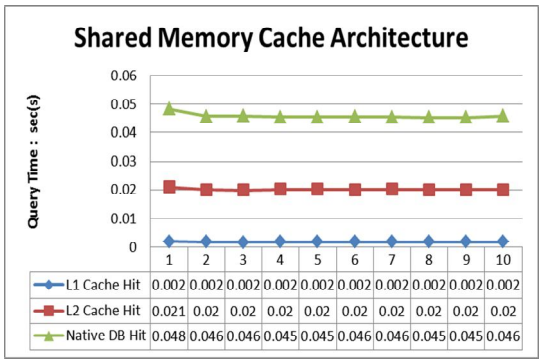
各架構之測試結果如下，各架構中橫軸值表示查詢次數，直軸表示查詢時間。各層快取擊中分別以不同顏色及符號代表，下方顯示其查詢時間結果值。

表<2>為無快取架構設計之查詢測試結果，查詢時間呈現不穩定的狀態。在十次隨機資料的查詢中，查詢時間從十分之一秒到百分之二秒之間，資料的查詢速度時快時慢。



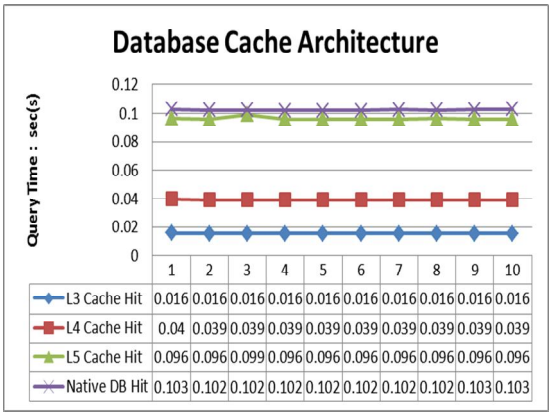
表<2>. - (A). 無快取架構

表<3>為共享記憶體架構的測試結果，共享記憶體L1及L2快取的速度時間平均在千分之二和百分之二秒內。當共享記憶體快取未擊中時，所需的查詢時間平均在百分之五秒以內。



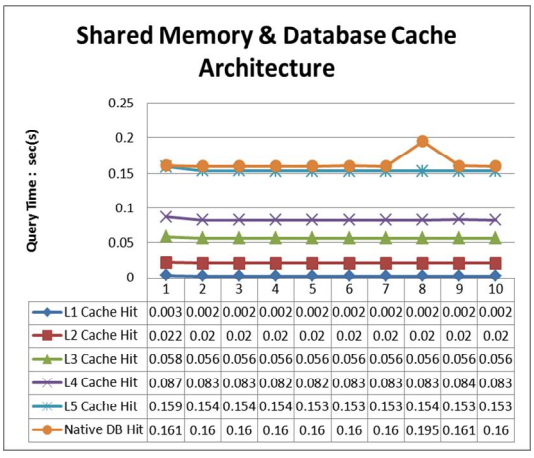
表<3>. - (B). 共享記憶體快取架構

表<4>為資料庫快取架構的測試結果，資料在L3、L4、L5快取的查詢時間平均在百分之二、四、九秒左右，而原始資料庫的平均查詢時間約十分之一秒。



表<4>. – (C). 資料庫快取架構

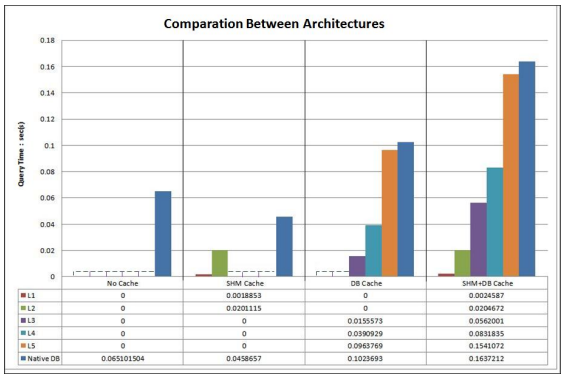
表<5>為共享記憶體+資料庫快取架構的結果，記憶體快取 L1 及 L2 的平均查詢時間約千分之二秒與百分之二秒，兩者速度相差約十倍。資料庫快取 L3、L4 及 L5 的平均查詢時間約百分之五、百分之八與百分之十五秒左右。當全部快取失誤時，大多數的查詢時間約在百分之十六秒左右。



表<5>. – (D). 共享記憶體+資料庫快取架構（ 5 層式快取架構 ）

由於各層快取的大小以倍數增加，並且使用共享記憶體與資料庫做為快取，因此得到的測試結果符合理論上的查詢效果呈現，使用共想記憶體作為快取比使用資料庫來的快上許多，另外當快取的大小越大，查詢時間也相對越久，資料之回寫速度亦受到影響。

表<6>為各快取架構的比較結果，由於不同架構的快取應用，各層快取分別以不同顏色表示，並由左至右排列，未使用的快取層以虛線表示。



表<6>. 各快取架構之比較（虛線部分為未使用的快取）

由於本研究實驗為測試五層式快取架構之加速效果，因此針對無快取架構與五層式快取架構做比較，以下為五層式快取架構各層快取速度與無快取設計架構之比較：

a. 五層式快取架構之 L1 快取的速度是無快取設計架構的 26.47 倍。
b. 五層式快取架構之 L2 快取的速度是無快取設計架構的 3.18 倍。
c. 五層式快取架構之 L3 快取的速度是無快取設計架構的 1.158 倍。
d. 五層式快取架構之 L4 快取的速度是無快取設計架構的 0.782 倍。
e. 五層式快取架構之 L5 快取的速度是無快取設計架構的 0.422 倍。
f. 五層式快取架構之 All Miss 時的速度是無快取設計架構的 0.397 倍。

根據測試結果與比較，資料在 L1、L2 及 L3 快取的查詢速度皆快於無快取架構設計的查詢速度，而 L4 及 L5 快取，因上層快取(L1、L2、

L3)之 Cache Miss 所付出之 Miss Penalty 增加了查詢的耗費時間，超出無快取架構設計的平均查詢時間，無法達到加速效果。此情況未來可調整快取層數及快取大小來解決，另外本實驗使用資料庫比數 50 萬筆來進行比較測試，未來當資料庫的資料筆數提高數倍時，如 100 萬筆、500 萬筆、1000 萬時，快取加速效果會更加明顯。

6.結論

本研究所提出之五層式快取架構(5 Tier Cache Architecture)、通用唯一識別碼(UUID)及可客製化的快取替換機制(Cache Replacement Algorithm)證明可提升資料庫查詢效能，減少實際資料庫之負載。五層式快取架構中，使用共享記憶體做為快取來提升資料庫查詢速度，最高可加速查詢 2647%，並將資料搜尋改以 UUID 方法來簡化多欄位與資料表之比對，最後加入可客製化的多種快取替換機制，使超過 50%以上之局部特性的查詢在 L1、L2 快取中查詢到，所以證明本實驗能夠有效地加速資料庫查詢。最後，本研究的資料庫查詢加速方法及架構可容易套用至傳統的資料庫查詢架構，無須修改大量的程式碼與程式架構，可快速應用與實現資料庫查詢加速的效果。

參考文獻

[1] Abraham Silberschatz, Greq Gagne and Peter B. Galvin, "*Operating System Concepts 8th, Update Edition*", Wiley, 2011.

[2] Ann McHoes and Ida M. Flynn, "Understanding Operating Systems", Course Technology, 2010.

[3] Irv Englander, "The Architecture of Computer Hardware, Systems Software, & Networking: An Information Technology Approach", Wiley, 2009.

[4] Jim R. Wilson, "Seven Databases in Seven Weeks: A Guide to Modern Database and the NoSQL Movement", Pragmatic Bookshelf, 2012

[5] J. Stanley Warford, "Computer Systems", Jones & Bartlett, 2009.

[6] Jim Handy, "The Cache Memory Book", Morgan Kaufmann, 1998.

[7] John Allspaw and Jesse Robbins, "Web Operations: Keeping the Data On Time", O' Reilly Media, 2010.

[8] Steven A. Przybylski, "Cache and Memory Hierarch Design: A Performance Directed Approach", Morgan Kaufmann; 1990.

[9] Xavier Vera, Bj¨orn Lisper , Jingling Xue, "Data Cache Locking for Higher Program Predictability", IGMETRICS '03 Proceedings of the 2003 ACM SIGMETRICS international conference on Measurement and modeling of computer systems, 272-282

[10] 高易中與鄭芳田(2010)。植基於主記憶體型資料庫技術之新式全自動化虛擬量測架構。(碩士論文，國立成功大學，2010)，台灣博碩士論文知識加值系統。

[11] 高明慶與曾新穆(2002)。延伸式標籤語言資料庫上的預測性快取記憶體管理機制。(碩士論文，國立成功大學，2002)。**台灣博碩士論文知識加值系統**。

[12] 邱士誠與王宗一(2011)。關聯式資料庫之快取置換機制。(碩士論文，國立成功大學，2011)。**台灣博碩士論文知識加值系統**。

[13] 楊志祥與許蒼嶺(2000)。高效率與階層式架構的 WWW 快取設計。(碩士論文，國立中山大學，2000)。**台灣博碩士論文知識加值系統**。

[14] 賴王駿與林志敏(2006)。LRU/LFU 混合置換策略之實作與效能評估。(碩士論文，逢甲大學，2006)。**台灣博碩士論文知識加值系統**。