用關聯式資料庫儲存多版本 XML 文件
Storing Multi-version XML Documents in a Relational Database System

陳耀輝　蔡政霖　余美伶　何銘啟
國立嘉義大學資訊工程學系
{ychen, s0910253, j3603011, s0920206}@mail.ncyu.edu.tw

摘要

多版本 XML 文件是將原始版本及其後的版本差異整合成一個 XML 文件，如此可節省儲存的空間，並可從其中回復任一版本。在實作上，我們給予 XML 樹狀結構的每個節點兩種編碼：時間上的版本戳記和空間上的結構數值範圍。版本戳記可以被用來擷取之前的 XML 文件版本，而數值範圍則可被用來重建節點之間的關係。我們使用關聯式資料庫儲存多版本 XML 文件，應用關聯式資料庫儲存大量資料的能力來協助 XML 文件版本的管理。我們提出維護數值範圍的方法以方便增加新版本，並且提出以版本為主的資料叢集技術以加速擷取單一版本的時間。

關鍵詞

XML、版本管理、版本戳記、數值範圍、資料叢集

1. 簡介

XML（eXtensible Markup Language）是一種容許使用者定義標籤的標識語言，它提供了一種介於鬆散的 HTML 文件與結構嚴謹的資料庫間的一個半結構化資料結構。XML 近年來已經成為資料儲存的標準格式，也逐漸的被視為資料交換的標準格式。而現今已有數種不同儲存 XML 文件的方法可提供選擇，概略地分為檔案系統、物件導向資料庫[1]、關聯式資料庫[7][9][13][14]和特別為儲存 XML 文件所開發的系統[8][10][11][19]。

雖然已有各種形式儲存 XML 文件的方法被提出，但仍就無法完整的保留使用者對單一文件連續修改的痕跡，我們把這個問題稱為 XML 文件版本管理。為了解決這個問題我們必須要考慮：(1)文件的時間價值性、(2)文件的持續使用性、(3)文件內容不因為時間流逝而改變。所以我們必須完整且持續地去記錄不同時間所修改的 XML 文件內容。最直覺的方法是把每次修改後的 XML 文件後都另存成一份新的文件，並對檔案名稱作系統化的規劃。然而，這將會造成在日後會有大量的 XML 文件產生，而這些 XML 文件的管理複雜度勢必會提升，且檔案容量會越來越大。因此，我們將針對上述問題提出一套以前序式走訪（preorder）的方法，給予 XML 樹狀結構上的每個節點版本戳記（version number）和數值範圍，並且對數值範圍也提出一套維護的機制，而數值範圍包括了階層值（level）、最小權重值（min-weight）與最大權重值（max-weight）[20]。

另外，我們的資料叢集技術不同於以拷貝資料為主的方法，而是以資料的版本密度為基礎發展出 XML 文件版本的叢集方法，並且提出新的儲存方式，結合 XML 資料模式與關聯式資料模式，將 XML 文件版本中的節點、數值範圍和版本戳記存入關聯式資料庫。如此一來，我們便能在不增加檔案空間的條件下，以資料的時間排列密度方式來取得速度與儲存空間的平衡點。

而數值範圍結合版本戳記主要目的是藉由判別節點的版本戳記，可以直接地擷取特定時間的 XML 文件版本，並且利用數值範圍來維護節點之間的關係。因為在關聯式資料庫中記錄之間是沒有順序性的，額外地數值範圍也能成為 XML 文件的索引資訊，作為查詢路徑比對的依據進而加快查詢效率。最後以結合關聯

式資料庫現有處理大量資料的優勢，來解決目前原生 XML 資料庫無法處理檔案容量太大及速度緩慢的問題。

本文的文章結構如下，第二節介紹相關研究，第三節簡單說明我們所使用的 XML 文件編碼方法，第四節則說明 XML 文件編碼的維護方式，第五節介紹版本儲存與擷取的方法，第六節為實驗評估結果，最後第七節是我們的結論。

2. 相關研究

在索引機制維護方面，[14]將編碼方式整理歸類成三種：（1）編碼方法由深度優先的走訪方式依序給定一數值。（2）編碼方法由每個節點與相關的兄弟節點依序給定數值。（3）與方法二相同，都是由每個節點與相關的兄弟節點依序給定數值，不同的是每個節點必須繼承父親節點的數值。而最近[9]提出將[6][18]兩種編碼方法對 XML 文件內的節點編碼，並且將 XML 文件儲存至關聯式資料庫，期望能夠提供穩定、有彈性和高效能的儲存方式以支援 Xpath 索引結構，使得 Xpath 索引結構結合關聯式資料庫已建立的查詢處理技術，例如 B-Tree 和 R-Tree。[9]給予此編碼維護的建議，給定非連續的數值範圍，以滿足未來節點的新增及重新編碼的方法。對於關聯式資料庫來說 XML 文件的編碼維護方式及文件的查詢與擷取，是一個相當重要的課題。

而目前被提出的 XML 文件版本管理技術有 diff-based [3][4][12][16]、timestamps-based [2][15][17]等方式。Diff-based 方法的主要概念都是利用 edit-script 語法(insert, delete, update, move)來記錄同一份的 XML 文件不同版本之間的改變。[12]的做法是儲存最後一次修改的 XML 文件版本，利用 forward complete deltas 來記錄兩個 XML 文件版本的改變。[16]儲存第一次和最後一次修改的 XML 文件版本加上 forward complete deltas 的特性，改進了擷取 XML 文件版本的速度。[3]首先會完整地儲存最原始的 XML 文件版本，當 XML 文件版本被連續的修改時，只會儲存新插入文件的部份，而未改變的部分則是使用 reference record 來指向在前一版相同資料的位置。以 timestamp-based 方法為基礎架構的[2]提出版本戳記與數值範圍為 XML 文件版本建立索引的機制。但是卻無談論當新的 XML 文件版本加入時要如何的去調整與維護，其索引值的數值範圍也沒有的明確的定義如何給定間距值，只強調節點的祖孫及父子之間要有包含的關係，且也沒說明當一份新的 XML 文件版本被加入時，數值範圍的新增、刪除和修改應如何做調整。

儲存方面，[2][3][4]使用 page 的方式將 XML 文件版本儲存其中，並且使用 page usefulness 概念將資料叢集在一起。其缺點是需要複製原有的資料來達成叢集資料的目的，以至於會增加多餘的儲存成本。優點是版本之間的變動量大時，資料散亂的程度高，故拷貝資料較能將資料集中在一起，提高資料叢集的效能。[15]進一步的提出結合 snapshot、Diff 與 timestamp 優點的 XML 多版本的分支，但未說明分支版本的切割方式與新版本被新增時如何地去維護現有的 XML 多版本分支。[5][17]討論 XML 文件版本可查詢的內容指令，例如查詢兩版本之間的不同之處及路徑查詢的比對方式[5]。

3. 索引機制

為了在關聯式資料庫中儲存和查詢 XML 文件，我們必須使用特殊的編碼方法來維持 XML 文件的順序性，並且要有良好的機制去維護編碼結構。選擇 XML 文件編碼的方法，主要考量的有三個因素，分別是維護的成本、重建文件的效率和查詢的速度。

3.1 數值範圍索引

目前已有各種形式的編碼方式被提出，這些方法主要的問題在於當編碼方法擁有低成本的維護機制時，相對地也會有較高的查詢時間。我們發現[20]的數值範圍表示法，能在這些問題上取得了一個平衡點。但[20]在編碼維護時必須使用相加減的方式處理，且一次只能

處理一個元素，這對於 XML 文件版本管理時常會遇到子樹型態的資料來說，將花費太多的 CPU 計算時間。因此我們藉由[18]所提及的方式和改進[20]的編碼維護方式，來完成 XML 文件版本管理問題的需求。

數值範圍給定的方式是給予樹狀節構上的每個葉節點及非葉節點一組數值範圍，包含了最小權重值和最大權重值並且加上階層值，表示成(最小權重值：最大權重值, 階層值)，例如(1:18, 2)。葉節點只有最小權重值，所以葉節點的最大權重值則設為 Null 值，例如(1:Null, 2)，這是為了未來在關聯式資料中方便區分葉節點及非葉節點的方式。而我們可藉由包含關係的數值比對快速的決定節點間的關係。以下是一些基本原則：（1）數值範圍配置原則：對樹上的所有節點的走訪方式為前序式走訪(preorder)，並且給予最小權重值、最大權重值和階層值。（2）節點包含關係：父節點的最小權重值＜子節點的最小權重值且子節點的最大權重值＜父節點的最大權重值。階層值差 1 為父子關係，差 2 以上為祖孫關係。（3）非葉節點對等關係：左節點的最大權重值＜右節點的最小權重值。

3.2 版本戳記

我們發覺到在 XML 文件版本管理的問題裡每個節點都具有空間性和時間性，所以為 XML 文件版本分配數值範圍來定位每個節點的順序時，也能同時地為節點記錄生命週期。也就是說當 XML 文件經過連續性地修改後，有些資料是不變的，有些則是被新增、刪除或是更改。因此我們可以為節點設立一個版本戳記來表示資料的時間範圍。要特別注意的是在 XML 文件版本的問題裡，刪除只是邏輯上的形式。綜合了時間性與空間性的觀念後，我們只要將節點原有的數值範圍的資訊，再加上版本戳記後，就能充份地了解資料在全部的 XML 文件版本裡所在的位置。我們將數值範圍加上版本戳記表示為（最小權重值：最大權重值, 階層值, 版本起始值：版本終止值）。Figure 1 指出版本一的 XML 文件，每個節點上除了原本的數值範圍外，還加上了版本戳記，例如根節點 people 的權重數值範圍是從 1 到 180，階層為 1，版本起始值是 1，版本終止值是 now，表示為(1:180, 1, 1:now) ，now 代表的是最新的版本。

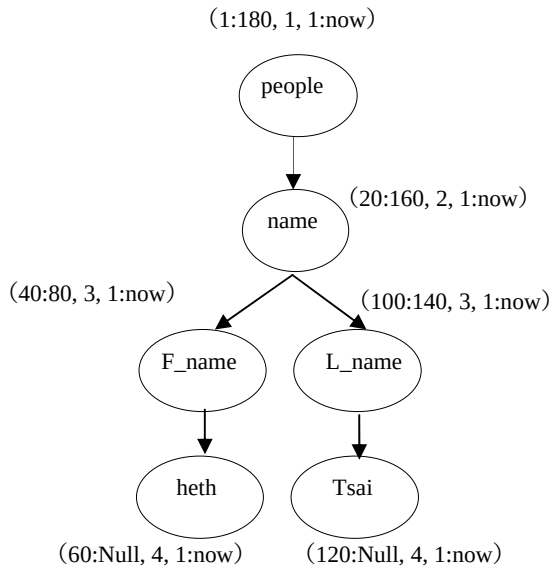

Figure 1: 版本 1-1

4. 索引維護

一個好的編碼方法，必須擁有低成本的維護。對於新增、刪除 XML 文件樹時，只須更動一小部份的資料，且能很方便地利用其它節點來決定被新增、刪除節點的數值範圍。如此，使得新增及刪除的問題更容易處理。XML 文件版本管理的問題中，元素的新增和修改是我們所最關心的問題，刪除則否，因為資料都必須被完整的保留下來，修改則繼承原本的數值範圍。新增的維護方式，主要可分為四種情形。從 Figure 2 可以觀察出，數值範圍的調整，關係到 3.1 節所提的包含關係和對等關係，我們必須了解新資料所處位置的父節點及兄弟節點的數值範圍，如此才能決定出新的數值範圍，文件版本的內容時，能在不需要恢復版本的情況下取得結果。再依據前序式走訪的方式給予子樹數值範圍，Figure 3 為維護的演算法。

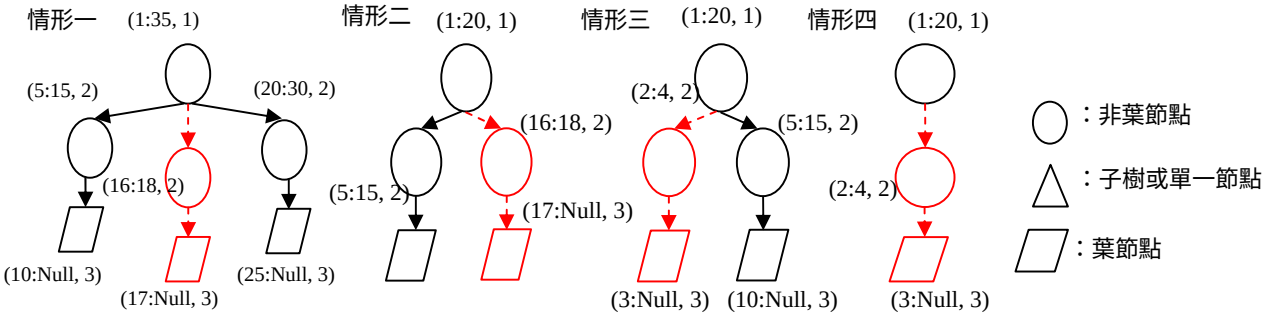

Figure 2: 數值範圍的新增維護方式範例

```
Input:版本差異(Diff_xml)    version:目前版本
Output:關聯表                    xpath:新增、刪除和修改的路徑
operation:新增、刪除和修改的指令

while(operation)
maintain_relation_table(operation, xpath, child, version){
  if(operation=新增){
    找出路徑的父節點與子節點集的數量;
    if(沒有子節點集){
      情形四:決定 parent_node 的範圍值,依前序式走訪給予新增子樹
      的節點數值範圍,並將資料存入關聯式資料庫;
    }
    elseif(只有 1 個子節點集){
      if(要插入的位置為子節點集的第一個){
        情形三:決定 parent_node 與子節點集的範圍值, 依前序式走
        訪給予新增子樹的節點數值範圍和版本戳記,並將資料存入
        關聯式資料庫;
      }
      else{
        情形二:決定 parent_node 與子節點集的範圍值, 依前序式走
        訪給予新增子樹的節點數值範圍和版本戳記,並將資料存入
        關聯式資料庫;
      }
    else{       //兩個以上的子節點集
      計算符合目前版本條件的子節點集並找出其位置;
      if(要插入的位置為子節點集的第一個){
        情形三;
      }
      elseif(要插入的位置>所有子節點集的位置){
        情形二;
      }
      else{
        情形一:找出要插入點的前後子集合數值範圍, 依前序式走訪
        給予新增子樹的節點數值範圍和版本戳記,並將資料存入關
        聯式資料庫;
      }
  }
  elseif(operation=修改){
    找出路徑的父節點與子節點集的數量,並拷貝原節點的數值範圍,
    加上新的版本戳記,存入關聯式資料庫中;
  }
  else{       //刪除
    找出路徑的父節點以及符合刪除條件的子節點集,並給予版本終
    止值;
  }
}
```

Figure 3: 數值範圍維護演算法

在 Figure 2 情形一的例子中，新增的子樹符合演算法情形一的判斷式，因為此子樹擁有左兄弟與右兄弟且數值範圍分別為(5:15, 2)和(20:30, 2)，故子樹的首節點的最小權重值必須大於 5 且最大權重值小於 20，當決定好首節點的數值範圍後依前序式走訪給予子樹的節點數值範圍。情形二中新增的子樹沒有右兄弟，依演算法判斷為情形二，所以我們必須找出左兄弟與父節點的數值範圍，分別為(5:15, 2)和(1:20, 1)，故子樹的首節點的最小權重值必須大於左兄弟最大權重值 5 且最大權重值小於父節點的最大權重值 20，當決定好首節點的數值範圍後依前序式走訪給予子樹的節點數值範圍。情形三與情形二正好相反，必須找出右兄弟與父節點的數值範圍，分別為(5:15, 2)和(1:20, 1)，故子樹的首節點的最小權重值必須大於父節點的最小權重值 1 且最大權重值小於右兄弟節點的最小權重值 5，決定好首節點的數值範圍後依前序式走訪給予子樹的節點數值範圍。情形四沒有左兄弟也沒有右兄弟，故子樹的首節點的最小權重值必須大於父節點最小權重值 1 且最大權重值小於父節點的最大權重值 20，決定好首節點的數值範圍。

經由上述討論的數值範圍索引維護，再配合上版本戳記，使得每份 XML 文件版本可藉由這兩個空間與時間的位置資訊而輕易地從關聯式資料庫中直接擷取。更重要的是，在查詢 XML 文件版本的內容時，能在不需要恢復版本的情況下取得結果。

5. 版本儲存與擷取

XML 文件樹裡，一個節點擁有了數值範圍資料和版本戳記，其內容分別為（最小權重值：最大權重值, 階層值）和（版本起始值：版本終止值），並且加上節點所擁有的元素值。我們將這些項目分別對映到資料表的 6 個欄位。主鍵是最小權重值、最大權重值、階層值、版本起始值和版本終止值所共同組成的，以避免造成違反關聯式資料模式唯一性原則。

版本起始值	版本終止值	元素
1	3	people
1	3	name
3	3	age
1	2	city
1	2	zip
3	3	gender
1	3	L_name
3	3	profile.
1	2	Addr.
1	3	F_name

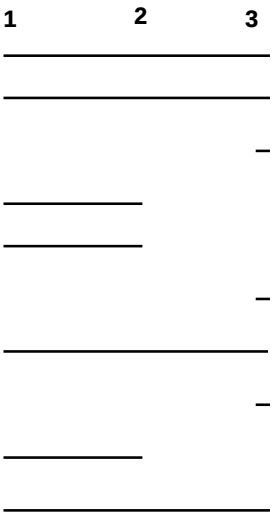

Figure 4: 資料表無叢集狀態

版本起始值	版本終止值	元素
1	3	people
1	3	name
1	3	F_name
1	3	L_name
3	3	age
3	3	gender
3	3	profile
1	2	city
1	2	zip
1	2	Addr.

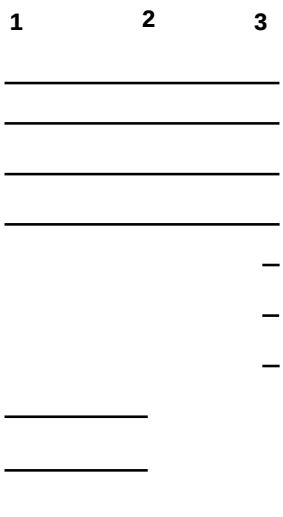

Figure 5: 加速新版本擷取

另外，為了加快 XML 文件的擷取速度，我們採用叢集索引的方式來改善。對版本起始值和版本終止值設定叢集索引，索引的方法則是選用 B-tree Index，因為擷取 XML 文件的查詢式為“ 版本起始值 ≤ 擷取的版本 ≤ 版本終止值 ”，針對特定範圍的查詢 B-tree Index 能獲得最佳的效能。Figure 4 顯示一個未設定叢集索引的資料表，Figure 4 左方資料表的每一筆記錄包含了版本起始值、版本終止值和元素，而右方的數條橫線則是代表每個元素在 XML 文件版本的生命週期。從元素生命週期的觀點來看，可以發現記錄之間的儲存是很鬆散的，由於關聯式資料庫對於資料的存取是以頁(page)為基本的單位，如此一來太過鬆散的資料在取存時將會花費大量的 I/O 時間，緊密的存放則能減少 I/O 的次數進而減少處理時間。

將資料叢集的主要概念來自於元素的生命週期，元素的生命週期有長有短，生命週期長的代表元素在 XML 文件版本裡出現的頻率較高，而生命週期短的代表元素在 XML 文件版本裡出現的頻率較低，如果我們把出現頻率較高的元素排列在一起，而頻率較低也排列在一起，如此一來擷取 XML 文件的時間就能降低許多。

另外當有越來越多的 XML 文件版本被產生時，例如被連續修改 100 次的 XML 文件，則使用者可能希望能快速擷取最新的 XML 文件，例如第 100 份文件，這樣的情況下我們必須將版本終止值較大的集中，綜合上述的觀點，如果使用者希望能加速新版的 XML 文件擷取首先必須將版本起始值由小到大排序；之後再對版本終止值由大到小排序，Table 1 說明了此演算法的步驟，Figure 5 呈現了執行資料叢集演算法一的結果。比較 Figure 4 和 Figure 5 元素生命週期的排列，Figure 4 顯的雜亂無章，Figure 7 則較有系統性的排列。

Table 1: 資料叢集演算法－加速新版本擷取

R : Table name

- Step 1：R.版本起始值由小到大排序
- Step 2：R.版本終止由大到小排序

相反的，使用者希望加速的是舊版本的擷取，則版本終止值由大到小排序，之後對版本起始值由小到大排序。Table 2 演算法說明了此一步驟，Figure 6 為執行結果。

Table 2: 資料叢集演算法－加速舊版本擷取

R : Table name

- Step 1: R.版本終止值 由大到小排序
- Step 2: R.版本起始值 由小到大排序

版本起始值	版本終止值	元素
1	3	peopl
1	3	name
1	3	F_name
1	3	L_name
1	2	city
1	2	zip
1	2	Addr.
3	3	age
3	3	gender
3	3	profile.

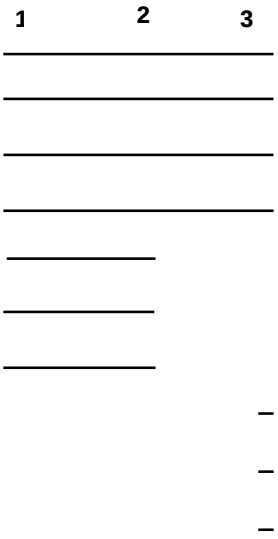

Figure 6: 加速舊版本擷取

我們的叢集演算法不需要每次在新版本被加入時都執行一次，因為排列資料必須花費多餘的時間，故新版本的資料被加入時，可直接排列在原來資料的底部，等待一段時間後資料量變大或者存取速度變慢時再執行重整。而[2]的叢集演算法必須每次新版本被加入時都必須執行頁的 usefulness 偵測，進行拷貝資料及頁的版本戳記維護否則資料的正確性將會出錯。

6. 評估結果

此章節我們分別對檔案空間大小儲存、XML 文件版本的擷取速度，以及讀取 I/O 做了實驗與分析[21]。我們實驗評估對象可分為原始文件（ALL）、保留第一份文件與記錄文件之間差異的文件（Diff）、使用時間戳記及數值範圍並且利用 usefulness 叢集參數來達到資料叢集的方法[2]（SPaR）、及我們提出的方法以 XML 形態儲存（VBVM）、以關聯表形態儲存（RT）、和以關聯表形態儲存並加上叢集（RTC）。Figure 7 和 Figure 8 表示在不同檔案大小變化率的儲存狀態下，我們的方法 VBVM 和 RT 依然能比 Diff 獲得更好的儲存空間。在 Figure 9 中，VBVM 以接近 100% 的改進率，其原因是 Diff 需要以逐步還原的方法，而 VBVM 則能直接存取。而由 Figure 10 可知檔案空間越大，變化率越高對改進率的影響越多，並不意外的是關聯式資料庫對於

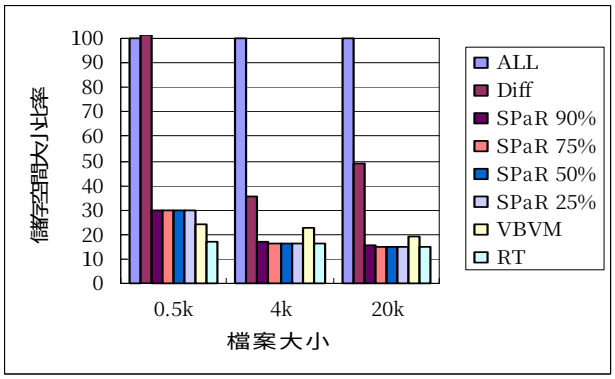

Figure 7: 變化率 15%，檔案大小 0.5K、4K、

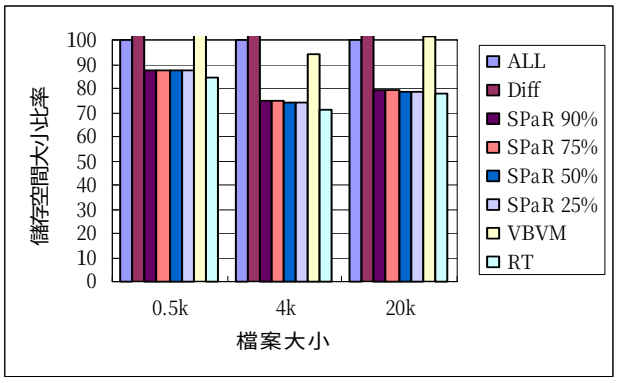

Figure 8: 變化率 120%，檔案大小 0.5K、4K、20K

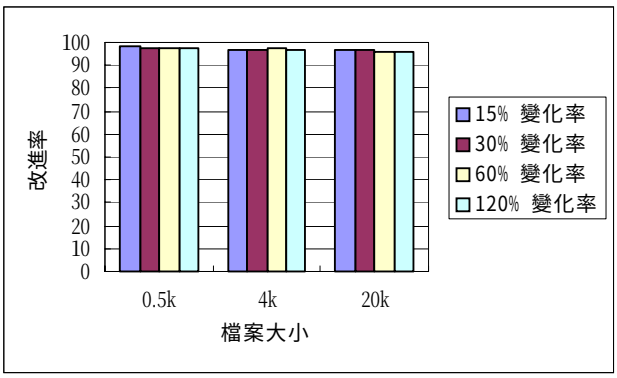

Figure 9: VBVM 對 Diff 擷取速度的改進率

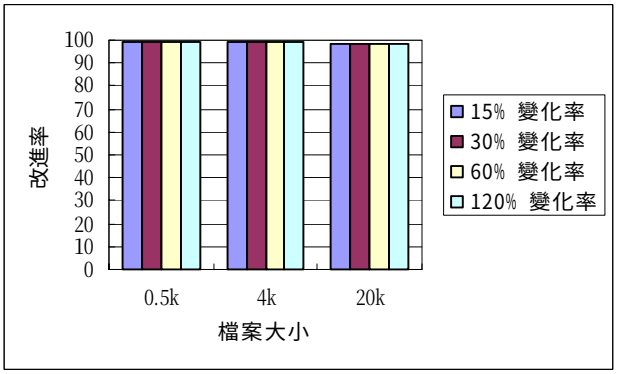

Figure 10: RT 對 VBVM 擷取速度的改進率

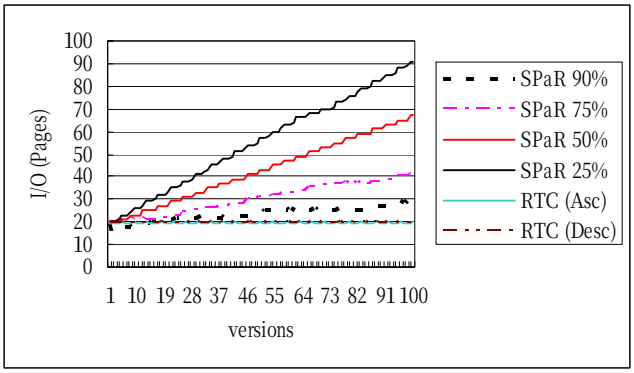

Figure 11: 檔案大小 20K，變化率 15%

處理大量資料的能力是不可否認的。目前的所有方法都沒辦法有降低儲存空間，除了壓縮之外，這點可從儲存空間的數據中獲得答案。所以我們版本密度方法較資料密度的方法[2]趨近於現實環境，因為在變化率低時我們的演算法有極佳的效率，如 Figure 11。

7. 結論

我們所提出的方法是利用前序式走訪 (preorder)，給予 XML 樹狀結構上的每個節點版本戳記和數值範圍，並且也提出數值範圍的維護方式。額外地數值範圍也能成為 XML 文件版本的索引資訊，因為我們 XML 文件版本的索引方法是屬於全域型，是橫跨所有的版本，而不是針對單一版本所作的索引方式，且索引值也是維護關聯表內所有記錄之間順序的重要資訊，藉由版本戳記的判斷則能直接地擷取任意版本文件。另外我們的資料叢集技術不同於以拷貝資料為主的方法，是以資料的版本密度為出發點所設計的叢集方法，是依據元素的生命週期來叢集資料與必須拷貝資料才能提高效率的方式完全不同，其優點在於能不增加檔案空間也能提昇擷取效能。空間與時間的資訊輔助，我們完滿的解決了數值範圍維護成本的問題、版本重建的效率和複雜路徑的查詢，使得 XML 文件的價值性、再利用與完整性得以有效率的儲存與利用。

參考文獻

[1] M. J. Carey, et al. “Shoring Up Persistent Applications,” *Proceedings of the 1994 ACM SIGMOD International Conference on Management of Data*, 1994.

[2] S.-Y. Chien, V. J. Tsotras, C. Zaniolo, D. Zhang, “Storing and Querying Multiversion XML Documents using Durable Node Numbers,” *Proceedings of the 2nd International Conference on Web Information Systems Engineering*, 2001.

[3] S.-Y. Chien, V. J. Tsotras, C. Zaniolo, “Efficient Management of Multiversion Documents by Object Referencing,” *Proceedings of the 27th International Conference on Very Large Data Bases*, 2001.

[4] S.-Y. Chien, V. J. Tsotras, C. Zaniolo, “Version Management of XML Documents,” *The Third International Workshop on The World Wide Web and Databases*, 2000.

[5] S.-Y. Chien, V. J. Tsotras, C. Zaniolo, D. Zhang, “Efficient Complex Query Support for Multiversion XML Documents,” *Proceedings of the 8th Conference on Extending Database Technology*, 2002.

[6] P. F. Dietz, “Maintaining Order in a Linked List,” *Proceedings of the Fourteenth Annual ACM Symposium on Theory of Computing*, 1982.

[7] D. Florescu, D. Kossman, “Storing and Querying XML Data using an RDMBS,” *IEEE Data Engineering Bulletin*, 1999.

[8] R. Goldman and J. Widom, “DataGuides: Enabling Query Formulation and Optimization in Semistructured Databases,” *Proceedings of the Twenty-Third International Conference on Very Large Data Bases*, 1997.

[9] T. Grust, M. van Keulen, J. Teubner, “Accelerating XPath Evaluation in Any

RDBMS," *ACM Transactions on Database Systems, Vol. 29, No. 1*, 2004.
[10] H. V. Jagadish, et al. "TIMBER: A Native XML Database," *The International Journal on Very Large Data Bases, Volume 11, Issue 4*, 2002.
[11] C. Kanne, G. Moerkotte, "Efficient Storage of XML Data," *Proceedings of the 16th International Conference on Data Engineering*, 2000.
[12] A. Marian, S. Abiteboul, G. Cobena, L. Mignet, "Change-Centric Management of Versions in an XML Warehouse," *Proceedings of the 27th International Conference on Very Large Data Bases*, 2001.
[13] J. Shanmugasundaram, K. Tufte, et al., "Relational Databases for Querying XML Documents: Limitations and Opportunities," *Proceedings of the 25th International Conference on Very Large Data Bases*, 1999.
[14] I. Tatarinov, Stratis D. Viglas, K. Beyer, J. Shanmugasundaram, E. Shekita, C. Zhang, "Storing and Querying Ordered XML Using a Relational Database System," *Proceedings of the 2002 ACM SIGMOD International Conference on Management of Data*, 2002.
[15] Z. Vagena, M. M. Moro, V. J. Tsotras, "Supporting Branched Versions on XML Documents," *Proceedings of the 14th International Workshop on Research Issues on Data Engineering*, 2004.
[16] R. K. Wong, N. Lam, "Managing and Querying Multi-version XML Data with Update Logging," *Proceedings of the 2002 ACM symposium on Document engineering*, 2002.
[17] F. Wang, C. Zaniolo, "Temporal Queries in XML Document Archives and Web Warehouses," *Proceedings of the 10th International Symposium on Temporal Representation and Reasoning and Fourth International Conference on Temporal Logic*, 2003.
[18] C. Zhang, J. Naughton, D. DeWitt, et al. "On Supporting Containment Queries in Relational Database Management Systems," *Proceedings of the 2001 ACM SIGMOD International Conference on Management of Data*, 2001.
[19] Berkeley DB, http://www.sleepycat.com/.
[20] 李銘桀, *針對多重路徑查詢建構 XML 文件索引機制*, 碩士論文, 國立嘉義大學資訊工程研究所, 2002.
[21] 蔡政霖, *XML 文件版本管理*, 碩士論文, 國立嘉義大學資訊工程研究所, 2004.