多重K_i-ary Tree為架構之XML結構化索引

葉介山　　林彥昌
靜宜大學資訊管理學系
jsyeh@pu.edu.tw　　g9234083@pu.edu.tw

摘要

近年來可延伸標記語言XML(Extensible Markup Language)已成為非常熱門的研究主題，許多文件格式都以XML為傳遞交換的基礎,XML可說是現今網際網路中儲存和傳送資訊最有發展之標準。而XML文件之中隱含了極多的結構化資訊，為了能使這些結構化資訊能提供結構化檢索(Structured Query)，本論文以K-ary 樹狀結構(K-ary Tree)為理論基礎，延伸發展出多重K_i-ary樹狀結構(Multiple Ki-ary Tree)，並以此理論建立XML文件之結構化索引(Structured Index)。本論文所提出的Multiple Ki-ary Tree的資料結構所佔的空間較K-ary Tree Split Method少。此外，由實驗得知：在不分裂的節點配置情況下,Multiple Ki-ary Tree在資料轉換的效能上比K-ary Tree為佳，且Multiple Ki-ary Tree能處理K-ary Tree所不能處理的某些特殊類型之XML文件。

關鍵詞：XML、K-ary樹狀結構、多重K_i-ary樹狀結構、結構化檢索、結構化索引

Abstract

In recent years XML(Extensible Markup Language) has become a very popular research topic and it has been used to be a basis of transmission and exchange for many document formats. For example, Taiwanese government has endeavored to promote electronic form of official documents and exchanges of electronic official documents in recent years, whose transmission standard are based on XML. XML could be said to be the most developed standard for storing and transmitting information on the Internet at present time. To create structured index and to provide structured query on XML document, this thesis thus utilizes characteristics of K-ary tree structure as theoretical basis to extend and develop Multiple K_i-ary Tree Structure and to establish a Structured Index of XML document through this theory.

Keywords: XML, K-ary Tree structure, Multiple K_i-ary Tree structure, structured Index.

1. 前言

資訊學術界以及業者在過去十數年間，分別提出許多適用於整合電子文件的國際標準或者業界標準，為電子文件整合管理建立良好的基礎。例如：可擴充標示語言 XML(Extensible Markup Language)等。這些標準和技術使得文件或表格的處理，以及與工作流程的整合變成可以實現的實體系統。

近年來可延伸標記語言(XML)已成為非常熱門的研究主題，不論是學界或是業界在理論和實作上皆有許多的研究。

現今，為了儲存和管理 XML 的文件，有兩種主要的處理方式，一是儲存在關聯式資料庫中，再者就是儲存在新發展的原生型 XML 資料庫(Native XML database) 中。但是不論是利用哪一種處理方式來儲存和管理 XML 文件，資料索引與資料檢索都是重點項目，而資料的索引重建更是影響效能的重要因素之一[1]。

本論文係以K-ary樹狀結構的理論為研究基礎，延伸發展出多重K_i-ary樹狀結構(Multiple K_i-ary Tree Structure)，去儘可能的避免索引重建。

在記憶空間節省方面，本論文所提出的Multiple K_i-ary Tree的資料結構所佔的空間較K-ary Tree Split Method為少。此外，在資料轉換的效能方面，由實驗得知：在不分裂的節點配置情況下，Multiple K_i-ary Tree比K-ary Tree為佳，且Multiple K_i-ary Tree能處理K-ary Tree所不能處理的某些特殊類型之XML文件。例如：深度較深的XML文件等。

2. 文獻探討

2.1 XML 背景

XML 是近年來相當熱門的一個話題，XML 由全球資訊網協會(World Wide Web Consortium:W3C)的 XML 工作小組所定義的可延伸標記語言。該小組將 XML 描述如下：

The Extensible Markup Language (XML) is a subset of SGML that is completely described in this document. Its goal is to enable generic SGML to be served, received, and processed on the Web in the way that is now possible with HTML. XML has been designed for ease of implementation and for interoperability with both SGML and HTML.

這段引言從官方的 XML 規格書 1.0 版[4]中節錄下來的，文中的重點就是 XML 是 SGML 的一個

子集它的目標是讓一般的 SGML 能夠在網站上被服務、接收與處理，如同 HTML 一樣而且能和 SGML 以及 HTML 共同合作應用。為何會說 XML 是 SGML 的子集，而 HTML 不算是呢?因為 XML 和 HTML 雖然一樣都是從 SGML 演變而來的，只不過 HTML 是 SGML 的一個應用語言(Application Language)，而 XML 卻是 SGML 的一個精簡子集(Subset)。XML 將 SGML 去蕪存菁，捨棄約百分之二十複雜罕用的部分，承襲了其他百分之八十的特點，是以具備了 SGML 所沒有的簡易性與靈活性，又有著 HTML 所欠缺的擴展性與結構性[3]。

XML 除繼承了 SGML (Standard Generalized Markup Language) 的可擴充能力之外，另外提供使用者自訂標籤、自訂文件結構，並以純文字方式表示各式各樣的文件資訊。此外，XML 還儘可能地降低 SGML 的複雜性與困難度，使其本身簡單且易於使用。XML 對於使用者自行定義的標籤以及文件結構部份，是定義在所謂的 DTD (Document Type Definition 或 Document Type Declaration)中。根據 DTD 是否包含於 XML 文件內，可將 DTD 分成兩種：若 DTD 另屬一個文件則稱此 DTD 為外部 DTD (External DTD；若包含在 XML 文件中則稱之為內部 DTD (Internal DTD))[2]

2.2 資料索引結構 — K-ary 完整樹狀結構

為使得能在結構化的文件中做有效的索引，則必須在結構化的文件中的結構資訊作有系統的索引。如果要快速得知結構化文件中節點和節點之間的連帶關係，則要取決於文件索引結構。Lee 等[5]所提出的 K-ary Tree 結構既可針對結構化文件做索引。

Lee 等所提出的 K-ary Tree 索引結構基本上可以透過基本而簡易之數學方程式來運算，進一步取得節點值。而 K-ary Tree 本身乃是一個完整的樹狀結構(Complete Tree Structure)，而 K 本身代表結構化文件樹狀結構中的最大分支數。接著依序配置結構化文件樹狀結構中的每個節點，可將結構化文件中每一個節點皆給予一個「唯一元素識別符號(Unique Element Identifier, UID)」，其值則代表該節點在階層式架構中的位置，透過節點的 UID 簡易而快速計算之後即可得知其父節點或者是子節點的 UID，進一步擷取所需的資料，請看下列說明。

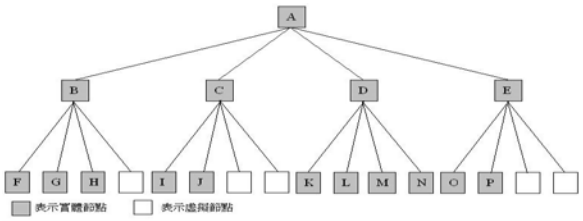

圖 1 4-ary 樹狀結構

為了能達到快速索引，Lee 等以 UID 來代表各個節點在結構化文件樹狀結構中的位置。理論上每一個結構化文件都可以被展開成一個階層式的樹狀結構，所以理論基礎在於能將每一個結構化文件對應到一個完整的樹狀結構(Complete K-ary Tree)，並且以該份結構化文件的最大分支數做為樹狀結構的原始分支數 K，以原始結構化文件的角度來看，假設資料節點有存在於原始結構化文件則該節點被稱為實體節點(Real Node)，相反的，若該資料節點不存在於該原始結構化文件時，則該節點被稱為虛擬節點(Virtual Node)。如圖 1 所示，為一 4-ary Tree，其意思即代表該樹的最大分支數為 4，每一節點都有四個分支，由實線黑底方塊代表的就是實體節點(Real Node)，其意義說表示該節點的資料是實際存在於原始結構化文件之中，而實線白底方塊所代表的就是虛擬節點(Virtual Node)，其意義表示說該節點是不存在於原始結構化文件之中的。

為了保持樹狀結構的完整性，必須將虛擬節點(Virtual Node)一起加入 UID 的配置。依據樹狀結構的順序由左至右配置每一個節點單一識別號，每個節點都要有一個 UID，最後節點的 UID 配置結果如表一所示。

表 1 實體節點 UID 配置表

Element	UID	Element	UID
A	1	I	10
B	2	J	11
C	3	K	14
D	4	L	15
E	5	M	16
F	6	N	17
G	7	O	18
H	8	P	19

至於為何要利用 UID 來表示結構化文件的結構資訊呢?由於 K-ary Tree 是一個完整的樹狀結構，所以我們可以利用下列的方程式一以及方程式二來得知結構化文件中所以父節點和子節點的對應關係。簡單來說係利用方程式的計算來得知父節點以及子節的 UID，進一步的快速存取我們所要的節點。

在下列的方程式中，方程式一為父節點 UID 的計算方程式，方程式二為子節點 UID 的計算方程式，而在方程式一中，i 所代表的是本身節點的 UID，而 k 代表的則是此 K-ary 樹狀結構的分支數。而在方程式二中，i 和 k 的定義不變，而 j($1 \leqq j \leqq k$)則代表表該節點的第幾個子分支。

$$\text{parent}(i) = \left\lfloor \frac{i-2}{k} + 1 \right\rfloor$$

方程式 1 計算父節點 UID 之公式

$$\text{child}(i, j) = k(i-1) + j + 1$$

方程式 2 計算子節點 UID 之公式

在此我們利用圖 1 和表 1 為範例來說明。在表

1 中，我們可以得知節點 C 為節點 I 之父節點，而節點 I 的 UID 為 10，代入方程式 1 中可得知其父節點的 UID 為 parent(10) = $\left\lfloor \frac{10-2}{4}+1 \right\rfloor$ = 3，其 UID 值為 3，符合表 1 節點 C 的 UID 值。

相反的，以子節點來說，我們可以得知節點 I 為節點 C 第一個子節點，代入方程式 2 中得到其 UID 值為 child(3,1)=4(3-1)+1+1=10，同樣地也符合表 1 的節點 I 的 UID 值。

2.3 Split Method 說明

新增節點可以說是樹狀結構變動中最為複雜的一種，因為新增節點有可能會造成樹的最大分支變動，以致整個樹狀結構變動，造成整個的樹狀結狀的配置必須重新建置。

針對新增節點後造成整個樹狀結構變動之情形，在 2001 年洪健哲[2]提出了名稱為 K-ary Tree Split Method 的方法。

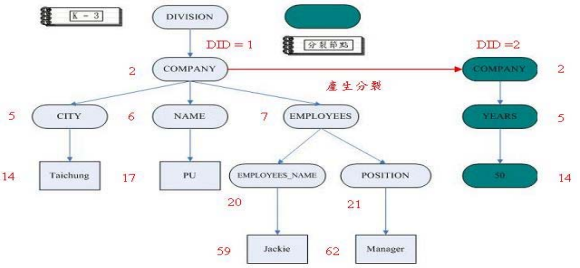

圖 2 應用 Split Method 在 K-ary Tree，在 TEAM 節點下新增一筆 YEARS 資料時所產生的分裂結果

在此我們以 K-ary Tree 的例子來說明 Split Method 的方法。顧名思義，Split Method 就是指新增節點後，若造成 K-ary Tree 的最大分支數變動時，則從被插入節點 A 處分裂出一個新的節點 A’。定義一個對等關係(Equivalent Relation)來記錄兩個節點間的關聯性，並且利用 DID 分屬不同文件的概念將分裂出來的節點 A’ 以及其底下的所有子節點為另一個新的 XML 文件並且對應到原來的 k-ary Tree。

為表示對等關係，Split Method 在索引結構中新增 EQF (Equivalent Forward)與 EQB (Equivalent Back)兩個欄位資訊。以 A 和 A’ 而言，A 的 EQF 為 A’ ，表示節點 A 分裂出來的節點為 A’ ，A’ 的 EQB 為 A，表示節點 A’ 乃從節點 A 分裂出來。在此即以圖 9 的 XML 範例來說明。

在節點「COMPANY」下若新增一名「YEARS」其「50」時，因原有最大分支數只有 3，如此便會導致節點「COMPANY」下的分支數超過最大分支數 K = 3，則經過 Split Method 分裂後的結果如圖 9 所示。

圖 2 中深色部份的節點代表新增之節點，由於插入「COMPANY」後造成此節點的分支數超過 K，因此被插入的節點「COMPANY」隨即分裂出一個對等的節點，即圖中右手邊的『COMPANY』節點。在此，被插入的「COMPANY」節點所屬的 DID = 1，分裂出來的『COMPANY』節點所屬的 DID = 2，至於二個的「COMPANY」節點的 UID 皆為 2。對於新增的「YEARS」節點其父節點即為分裂出來的『COMPANY』節點，該節點的 DID 與其父節點同為 2，而 UID 則依照方程式 2 透過其父節點之 UID 與其所在之分支位置與最大分支數計算得知為 5。以此類推可得知文字節點「50」之 UID 為 14，而兩子節點之 DID 和父節點一樣同是為 2。

表 2 利用 Split Method 所產生的 K-ary Tree 索引資料表

SeqId	DID	UID	Level	ElementType	Value	ChildNo	Delete	EQF	EQB
1	1	1	1	Root	DIVISION	1	0	0	0
2	1	2	2	Element	COMPANY	3	0	12	0
3	1	5	3	Element	CITY	1	0	0	0
4	1	14	4	Text	Taichung	0	0	0	0
5	1	6	3	Element	NAME	1	0	0	0
6	1	17	4	Element	PU	0	0	0	0
7	1	7	3	Element	EMPLOYEES	2	0	0	0
8	1	20	4	Element	EMPLOYEES_NAME	1	0	0	0
9	1	59	5	Text	Jackie	0	0	0	0
10	1	21	4	Element	POSITION	1	0	0	0
11	1	62	5	Text	Manager	0	0	0	0
12	2	2	2	Element	COMPANY	1	0	0	2
13	2	5	3	Element	YEARS	1	0	0	0
14	2	14	4	Text	50	0	0	0	0

從表 2 中可看出 SeqId = 14 的『COMPANY』節點，其 EQB = 2，表示此節點是從 SeqId = 2 的「COMPANY」節點分裂出來的。相反地，我們也可以從 SeqId = 2 的「COMPANY」節點之 EQF = 14 得知分裂節點為 SeqId = 14 的節點。

3. 多重K_i-ary Tree架構

3.1 多重K_i-ary Tree架構原理

由於 K-ary Tree 為完整之樹狀結構，若資料節點為不存在於原始文件中，則此節點為虛擬節點，在必須維持樹狀結構完整的前題之下，必須將虛擬節點加入和實體節點一起進行 UID 配置。

但由於結構化文件會有經常性的結構資訊的變更，因此 K-ary Tree 上要經常新增節點、刪除節點以及更新節點。當資料節點更新造成了 K-ary Tree 的最大分支數 K 值改變時，如果假設每次變更的值皆超過 K 值，就必須重建 K-ary Tree 的索引以維護節點 UID 的正確性以及 K-ary Tree 的完整性，但這些動作將會耗費大量的時間，當索引資料量很大時，更顯示出 K-ary Tree 在索引重建時將會耗費極多的時間，這些都是索引重建的缺點。

由於目前的 K 值是依據結構化文件的最大分支數來決定，所以一但決定 K 值就會固定，若如果日後有新增、刪除造成 K 值變化，則必須做 K-ary Tree 索引重建，則會影響系統效能，因此本研究延

伸出一種多重 K 值設定，其定義是在結構化文件的每一階層都給予不同的 K 值(最大分支數)，此舉用意在於每一階層的分支數不用相同，盡量避免或是減少 K 值(最大分支數)的變動而將 K-ary Tree 索引重建的機會降到最低或是變動幅度降至最少。

3.2 Multiple K_i-ary Tree架構說明

Multiple K_i-ary Tree 以 Level 及 LID(Level Identifier)來取代UID(Unique Element Identifier)。UID的概念就是將結構化文件中每個節點都予以一個編號，由左至右依序編碼；而Multiple K_i-ary Tree則是以結構化文件中每一層給予不同的K值(意指給予每一層不同的分支數)，其各節點的Level值代表該節點所位於樹狀結構的階層，LID依每一層的K值進一步將每層中的節點予以個別編碼。

```
<?xml version="1.0"?>
<DIVISION>
  <COMPANY>
    <CITY>Taichung</CITY>
    <NAME>PU</NAME>
    <EMPLOYEES>
      <EMPLOYEES_NAME>Jackie</EMPLOYEES_NAME>
      <POSITION>Manager</POSITION>
    </EMPLOYEES>
  </COMPANY>
</DIVISION>
```

圖 3 XML 文件簡例

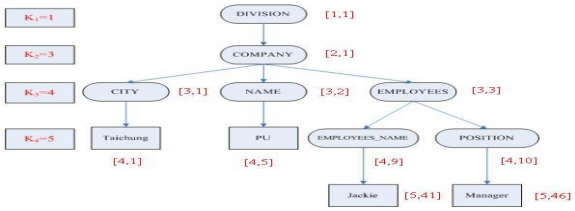

圖 4 XML 文件轉換 DOM 之後的樹狀結構圖

圖 3 的XML結構化文件會先透過XML Parser剖析後再轉換成圖 4 的DOM樹狀結構，在圖中方框的節點表示實際的資料內容，橢圓的節點則為標籤元素的資訊。轉成DOM之後，再根據每一個節點在結構化文件中的階層位置取出其對應的資料並將它轉存到Multiple K_i-ary Tree之中。

在這個範例當中，「DIVISION」為最上層之根節點，而「COMPANY」為其所屬之子節點。而在「COMPANY」節點之下又有其所屬的子節點與資料。我們依據這樣階層關係，XML Parser從根節點開始配置LID。例如：給定K_1= 1，K_2 = 3，K_3 = 4，K_4=5 等，則LID值的範圍如表 3 所示。

表 3 LID 值的範圍

Level	LID
1	1
2	1…K_1= 1
3	1…K_1*K_2 = 1* 3 = 3
4	1…K_1*K_2*K_3 = 1* 3*4 = 12
5	1…K_1*K_2*K_3*K_4 = 1* 3*4*5 = 60

在多重K_i-ary Tree結構中，每一節點將給予一個二維值[Level, LID]，其中Level值代表該節點所位於樹狀結構的階層，LID為該節點在該層之個別編碼。

$$parent([i, j]) = \left[i - 1, \left\lceil \frac{j}{K_{i-1}} \right\rceil \right]$$

方程式 3 計算父節點之 Level 及 LID

$$child([i, j], n) = [i + 1, (j - 1)K_i + n]，其中 n \le K_i$$

方程式 4 計算子節點之 Level 及 LID

在方程式(3)(4) 中，i為該節點中在結構化文件中所代表的所在階層，而j則代表該節點在該階層中的LID值，而n表示該節點的第n個子分支，K_i則代表第i階層中各節點的最大分支數。

在以「COMPANY」節點為例，其 Level 為 2，LID 值為 1，我們欲得知其子節點「EMPLOYEES」的 LID 值，我們可以代入方程式(4)來計算，由於「EMPLOYEES」節點是「COMPANY」節點的第三個分支，所以 child([2,1], 3)= [(2+1), (1-1)3 + 3]，由此方程式(4)得知，「EMPLOYEES」節點的 Level 為 3，LID 值為 3。由此方式即可配置完所有節點的 LID，以圖 3 的 XML 文件簡例為例，則其 LID 配置完的結果如表 4 所示。

表 4 實體節點 LID 配置表

Element	[Level, LID]
DIVISION	[1,1]
COMPANY	[2,1]
CITY	[3,1]
Taichung	[4,1]
NAME	[3,2]
PU	[4,5]

4.實驗分析

此章節本論文將針對K-ary Tree Split Method以及Multiple K_i-ary Tree二者做分析比較。為確保實驗數據有誤差，在每次實驗時，每一單項實驗至少跑五次，然後再取得實驗數據之平均值，如此期望能將實驗的數據誤差值降至最低。

首先我們先針對初始化的UID配置和LID配置

做時間比較。詳見如下：

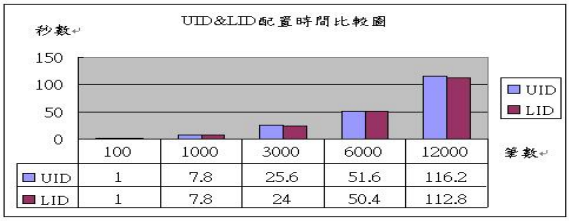

圖 5 實驗 1 之結果

以圖 5 為例，實驗 1 的測試為將二者初始化 XML 文件，也就是配置 UID 以及 LID 的時間做比較分析。以下列出 5 組深度的樹，每組資料以 n(m)表示，其中 n 表示測試筆數 5，m 表示資料節點。測試資料為如表 5：

表 5 實驗 1 之測試資料明細

筆數 / 分支數	K	K_i
100(496)	100	K_1=1,K_2=100,K_3=2,K_4=1,K_5=1
1000(4996)	1000	K_1=1,K_2=1000,K_3=2,K_4=1,K_5=1
3000(14996)	3000	K_1=1,K_2=3000,K_3=2,K_4=1,K_5=1
6000(29996)	6000	K_1=1,K_2=6000,K_3=2,K_4=1,K_5=1
12000(59996)	12000	K_1=1,K_2=12000,K_3=2,K_4=1,K_5=1

由圖 5 的結果來看，二者的配置時間上其實並沒有太大的差異，不過隨著筆數的增加，LID 配置法還是比 UID 時間上來得快。

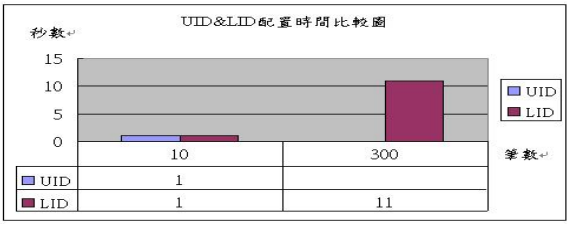

圖 6 實驗 2 之結果

實驗 2 的測試筆數(括號內為資料節點數)分別為 10(222)、300(6022)，測試資料樹的深度為 18，詳見表 6。

表 6 實驗 2 之測試資料明細

筆數 / 分支數	K	K_i
10(222)	10	K_1=1,K_2=10,K_3=3,K_4=1,K_5=1,K_6=1,K_7=1,K_8=2,K_9=1,K_{10}=1, K_{11}=1,K_{12}=1,K_{13}=1,K_{14}=1,K_{15}=1,K_{16}=1,K_{17}=1,K_{18}=1
300(6022)	300	K_1=1,K_2=300,K_3=3,K_4=1,K_5=1,K_6=1,K_7=1,K_8=2,K_9=1,K_{10}=1, K_{11}=1,K_{12}=1,K_{13}=1,K_{14}=1,K_{15}=1,K_{16}=1,K_{17}=1,K_{18}=1

由圖 6 可以發現一件事，就是K-ary Tree Split Method的實驗數據在 300 筆數時是空的，那是因為在配置UID的過程之中發生了溢位的情況，當K-ary Tree Split Method在配置第九層節點UID時，此時第九層的節點值已超過關連式資料庫目前所定義的數值範圍，由實驗 2 的結果來看，K-ary Tree Split Method在配置UID值時會有文件格式上的限制，但因為Multiple K_i-ary Tree是每一層定義的最大分支數不相同，所以並不會有這種類似情形。

再進一步去分析二者的儲存格式的資料結構來看，本論文所提出的Multiple K_i-ary Tree的資料結構所佔的空間比K-ary Tree Split Method還要少的。

表 7 K-ary Tree 空間配置表(Bytes)

SeqId	DID	UID	Level	ElementType	Value	ChildNo	Delete	EQF	EQB
4	4	4	4	10	10	4	1	4	4

表 8 實驗 2 之測試資料明細

SeqId	DID	Level	LID	ElementType	Value	ChildNo	Delete	EQF	EQB
4	4	4	4	10	10	4	1	4	4

由表 7 和表 8 的分析來看，本論文所提出的Multiple K_i-ary Tree每一筆資料所佔的空間K-ary Tree Split Method少了 4 Btyes。因為Multiple K_i-ary Tree所定義的每一層最大分支數是不相同的，所以在資料庫欄位的定義上只需定義int(4Bytes)即可，如果是最大分支數不大的文件，甚至可定義smallint(2Btyes)將會更節省空間。

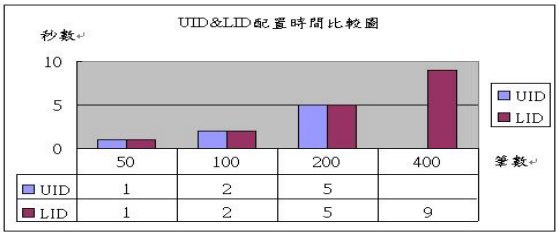

圖 7 實驗 3 之結果

接下來實驗 3 的測試筆數(括號內為資料節點數)分別是 50(742) 、100(1402) 、200(2802)、400(5602)，而測試資料樹的深度為 10，詳見表 9。

表 9 實驗 3 之測試資料明細

筆數 / 分支數	K	K_i
50(742)	50	K_1=1,K_2=50,K_3=3,K_4=1,K_5=1,K_6=1,K_7=1,K_8=2,K_9=1, K_{10}=1
100(1402)	100	K_1=1,K_2=100,K_3=3,K_4=1,K_5=1,K_6=1,K_7=1,K_8=2,K_9=1, K_{10}=1
200(2802)	200	K_1=1,K_2=200,K_3=3,K_4=1,K_5=1,K_6=1,K_7=1,K_8=2,K_9=1, K_{10}=1
400(5602)	400	K_1=1,K_2=400,K_3=3,K_4=1,K_5=1,K_6=1,K_7=1,K_8=2,K_9=1, K_{10}=1

由圖 7 得知二者在配置節點值上面速度上並沒有什麼太大的差異性，唯一最大的不同在UID筆數

400 的格子是沒有數據的，原因也是發生資料計算時出現了溢位。由實驗 1、2、3 得知，二者在配置節點值的速度上其實是沒有太大差異的，但是在較特殊的XML文件上，例如：樹深度較深情況下，Multiple K_i-ary Tree仍然可以處理節點配置而K-ary Tree Split Method卻不行。

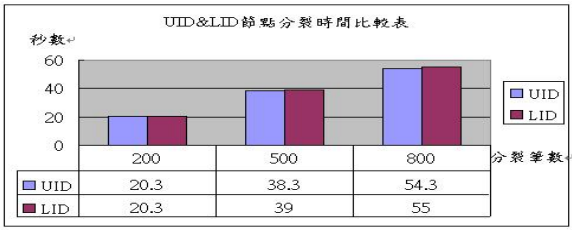

圖 8 實驗 4 之結果

在實驗 4 中，我們假設一開始配置節點值時就發生加入節點數已超過定義之最大分支數，所以一開始便產生分裂節點動作。測試筆數為 1000，詳見表 10。

表 10 實驗 4 之測試資料明細

筆數 / 分支數	K	K_i
1000	800	K_1=1,K_2=800,K_3=3,.....略
1000	500	K_1=1,K_2=500,K_3=3,.....略
1000	200	K_1=1,K_2=200,K_3=3.....略

當 K=800 時，此時加入節點數 1000 已超過所定義最大分支數 800 所以產生 200 個分裂節點，所以 K=500 產生了 500 個分裂節點，而 K=200 產生了 800 個分裂節點，如圖 8 所示，其實二者在效能上相去不遠。

綜合上面實驗，K-ary Tree在分裂節點配置比Multiple K_i-ary Tree為快，而在不分裂的節點配置上，Multiple K_i-ary Tree又比K-ary Tree較佳，然而Multiple K_i-ary Tree能處理K-ary Tree所不能處理某些特殊類型之XML文件，例如：深度較深的XML文件，這點是Multiple K_i-ary Tree的優點所在。

5.結論

- 本論文提出了利用Multiple K_i-ary Tree來將XML文件透過DOM將一份有階層式的結構化文件轉換成資料庫的二維表格，如此任何文件皆能轉換至關連式資料庫中，再透過系統介面可將使用者所需之資料以文件型態格式匯出。
- 本論文針對K-ary Tree Split Method提出另一種可行性方法，且本論文提出的方法能處理原有方法所無法處理之特殊文件，利用Multiple K_i-ary Tree更可以縮小結構化文件節點值計算範圍，避免在計算節點時發生溢位，而且Multiple K_i-ary Tree所使用的空間位置比原有方法更節省。
- 針對特殊的XML文件(例如：深度較深)的文件處理上，經過實驗K-ary Tree Split Method並沒有辦法針對這種類型文件做節點配置，但Multiple K_i-ary Tree卻是可以的，其原因在於Multiple K_i-ary Tree它定義每一層不同的最大分支數，各層的節點分開計數卻又有關連性。

參考文獻

[1] 吳柏璋。1992。XML 文件儲存方式之研究。朝陽科技大學資訊管理研究所碩士論文。台中縣。

[2] 洪健哲。2001。XML 文件漸進式結構化索引更新與路徑表示式檢索之研究。國立交通大學資訊科學研究所碩士論文。新竹市。

[3] 佛教圖書館館訊。1990。第二十三期。http://www.gaya.org.tw/journal/m23/23-main2.htm

[4] W3C。http://www.w3.org/TR/REC-xml/

[5] Y. K. Lee, S. J.Yoo, and K. Yoon, ,“Index Structures for Structured Documents”,Proceedings of the 1st ACM international conference on DigiTal libraries, Pages 91-99, 1996.