植基於 MapReduce 的可延展性 XML 索引方法

施曉貞　　許雯絞　　廖宜恩
國立中興大學資訊科學與工程學系
{nic3p1217, wenchiaoh}@gmail.com　　ieliao@nchu.edu.tw

摘要

隨著巨量資料時代的來臨，雲端運算的技術也越來越被廣泛的討論與應用。許多原本設計應用於單機的理論與方法，必須重新檢視其上雲端的適用性。例如文獻所探討的 XML 索引方法，大多適用於單機處理較小的 XML 檔案，若是要應用在大型的 XML 文件，容易造成記憶體不足的困境。本文提出了一個延伸 CIS-X 的設計，並實作於 Hadoop MapReduce 中，讓建立索引可以透過雲端平行運算來處理，解決單機記憶體不足以應付大型 XML 檔案的問題，這個方法可以適用於任何完整的 XML 文件，沒有固定格式限制。同時本研究也提供了相對應的查詢方法，讓大型 XML 文件的讀取更加便利。

關鍵詞：XML 索引、CIS-X、MapReduce

Abstract

With the advent of the era of "big data", cloud computing technology is being widely discussed and applied. Many theories and methods, which are originally designed for stand-alone, must be re-examined the applicability in the cloud. For example, most of the XML indexing methods, discussed in the literature, are suitable for processing small XML files by stand-alone. When they deal with a large XML document, memory shortage problem will be encountered. In this paper, we present an extension design of CIS-X, which is implemented in Hadoop MapReduce, to handle the XML indexing through the cloud parallel computing. The proposed method can be applied to any XML file, no matter what structure they have.

Keywords: XML Indexing, CIS-X, MapReduce.

1. 前言

隨著巨量資料時代的來臨，雲端運算的技術也越來越被廣泛的討論與應用。根據美國國家標準局(National Institude of Standards and Technology)的定義[20]：「雲端運算是一種模式，能方便且隨需求應變地透過連網存取廣大的共享運算資源（如網路、伺服器、儲存、應用程式、服務等），並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。」現今雲端運算相關技術越來越成熟，尤其分散式運算與平行化處理的設計，成為運用多處理機系統與雲端運算平台所必備的知識與技能，透過網際網路將龐大的運算程序，拆解成數個較小的程序，再由多個伺服器組成的龐大系統，分別執行較小的程序後，再將結果匯整後傳回給使用者，可以解決大量資料處理的問題。

XML (eXtensible Markup Language) 是網際網路資料交換與儲存的主要文件格式之一。在 XML 文件相關操作中，最常見的就是 XML 查詢。但是，由於 XML 本身是階層式的巢狀結構，且多是以文字型態的檔案儲存在硬碟中，若要在較大的檔案執行多次查詢，易造成讀取速度緩慢的問題。文獻上有許多加速 XML 查詢的理論與方法被提出，最常見的作法是採用「索引技術」(Indexing) 協助判斷目標節點的位置。先前文獻 [1-6, 8, 9, 10] 所探討的索引方法大多是設計在單機上執行，僅適合於處理較小的 XML 檔案，若是要應用在大型的 XML 文件，容易造成記憶體不足，無法完成建置索引與查詢。近期內也有一些文獻開始探討在雲端上存取大量或大型 XML 檔案的議題 [7, 11]。目前本研究所蒐集到的文獻，對於大型 XML 檔做索引的研究相當少，而且大多有使用上的限制，例如，XML 文件被使用在記錄物理實驗或生物資訊的資料，這些文件往往有一個共通點，就是有一定的格式，對此所提出的索引方法，若 XML 文件的格式不固定且變異性較大時，就無法應用了。

因此本研究進一步延伸 CIS-X (A Compressed Index Scheme for Efficient Query Evaluation of XML Documents) [9] 的設計，使建立索引可以透過雲端平行運算來處理，解決單機記憶體不足以應付大型 XML 檔案的問

題，這個方法可以適用於任何完整的 XML 文件，沒有固定格式限制。同時本研究也提供了相對應的查詢方法，讓大型 XML 文件的讀取更加便利。本研究主要的貢獻包括以下幾點：

- 本研究針對 CIS-X 索引方法的特性，提出一個適用於 MapReduce 系統的 XML 文件分割方式，可以分批並平行化地進行索引建置，用以處理大型的 XML 文件。
- 由於目錄及檔案架構類似樹狀結構，因此本研究將 CIS-X 原本放置於記憶體的索引，以相似的架構存放於 HDFS (Hadoop Distribute File System)中。
- 透過索引可以加快處理查詢的速度，本研究針對較費時的產生完整查詢結果步驟，設計水平式的分工方法，改善直覺式的垂直分工，會有比率很高的檔案被多台機器垂疊讀取，且讀取的檔案僅使用極小部分資料，效率較差的缺點。

2. 相關研究

本研究將 CIS-X 的延伸設計實作於 Hadoop MapReduce 中，因此本節將介紹這兩個主題，2.1 節介紹 Hadoop MapReduce，2.2 節介紹 CIS-X。

2.1 Hadoop MapReduce

Hadoop MapReduce [13, 15-17]是一個以 MapReduce 為基礎的 software framework，能夠應用在大型叢集上，並且以一種可容錯的方式平行處理資料，此技術會被發展主要是因為資料量越來越大，也就是所謂的 big data 時代的來臨，單機分析處理已經越來越不堪使用，Hadoop MapReduce 則是一個可以在大型叢集上做大量資料的分析及處理的平台。

MapReduce 分為主要四個步驟，Splitting、Mapping、Shuffling 和 Reducing。首先 Splitting 是先將資料集切割分散給各個節點，Mapping 則是依據切割出去的資料集做處理，在這個步驟使用者須定義輸入的 key 和 value 及輸出的 key 和 value，Shuffling 是把 map 輸出 key 相同的結合到同一個節點，最後 Reducing 是把 Mapping 輸出的資料中，key 相同的 value 做合併處理，也是以 (key, value) 的方式輸出。

如圖 1 [13] 是一個利用 MapReduce 計算文字出現頻率的範例，第一步 Splitting 把 input 的資料依據一行一行做分割，共分為三個小片段並分配給三個 DataNodes 執行， Mapping 階段時依據每一個單字當作 Key，計算單字出現次數(Value)，到 Shuffling 階段則把相同 key 的放在一起再到 Reducing 做計算後輸出，可以看到此方式達到平行運算的效果。

2.2 CIS-X 的索引方法

CIS-X [9]是一個壓縮索引結構的方法，主要是為了提升 XML 文件查詢的效率，以下分兩個小節介紹 CIS-X 的方法，第一小節介紹如何建立壓縮的索引結構，第二小節介紹如何進行查詢。

2.2.1 建立索引結構

CIS-X 採用類似 DataGuide [8] 的合併方法，將相同的 Label path 合併，因此合併之後的索引結構不會有 Label path 重複出現的情形，因此可以有效的壓縮所需的空間，並減少查詢相同 Label path 的時間。如圖 2 (a)為一個 XML 文件的樹狀圖，a_i 代表節點標籤 a 在文件中出現第 i 次，標籤名稱上加上「@」代表屬性節點，方形代表節點相對應的內文，圖 2 (b)則為合併 Label path 後的 XML 索引樹。

CIS-X 採用如圖 3 hash table 來群聚相同標籤名稱的節點，達到節省空間及快速取得資料的優點，圖 3(a) 在 hash table 中記錄了以下的資訊：

- **Label** 是索引節點的 Dewey ID 編碼，代表該節點在索引樹結構中的位置。
- **Vid** 用來記錄索引節點的相關內容編號，如果 Vid 為 0 則表示這個節點沒有相關的內容。
- **Extent** 是索引節點有相同 Label path 的 id 編碼。
- **Children** 是索引節點的子節點。

圖 3(b)則記錄節點所對應的內文，以標籤和 Vid 當做 key，使用 linked list 的方式將內文串連在一起。

2.2.2 文件的查詢

CIS-X 的查詢分為兩階段處理，第一階段使用 TwigList [10] 的方法進行查詢，找出符合查詢結構的所有索引節點，第二階段比對符合節點中的 Extent，過濾出最終的查詢結果。例如圖 4 查詢單一路徑 “A//G” (G 為目標節點)，先執行 TwigList 找出符合的節點為(A,G)={(1.0, 1.0.0.0.1.1)}，透過 G 的 label 前置詞 “1.0” 為 A 的 label，得知 A 是 G 的祖先節點，然後比對 Extent，還原各節點的關係。

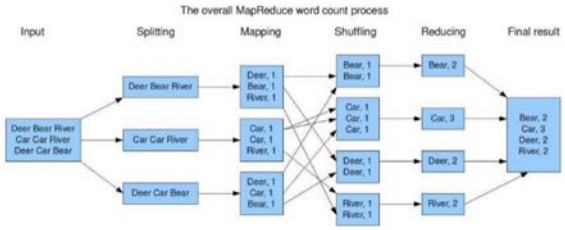


圖 1 MapReduce word count

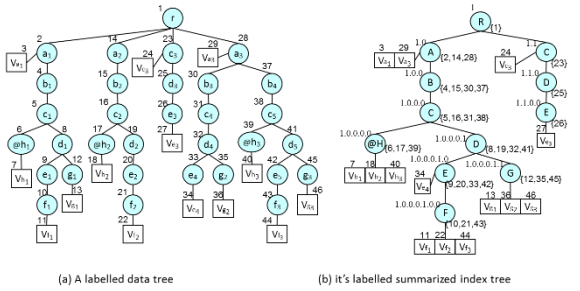


圖 1 (a)A Labeled Data Tree (b)The Labeled Summarized Index Tree

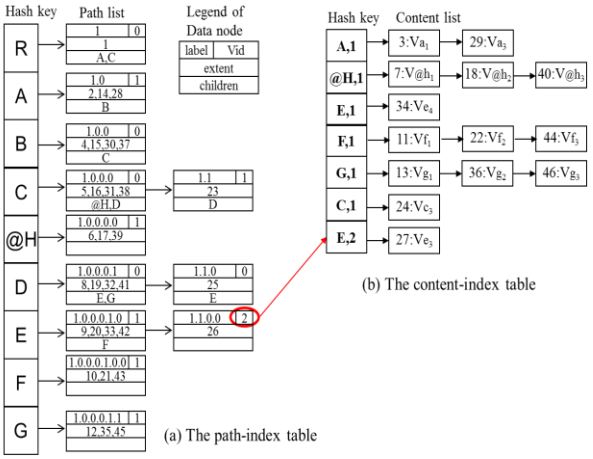


圖 3 CIS-X (a)The path-index table (b) The content-index table

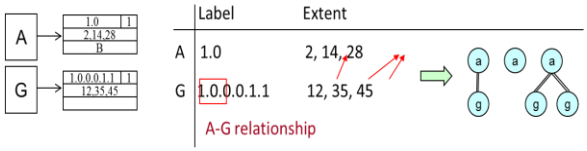


圖 4 查詢 "A//G" 的比對過程

對於分支路徑的查詢，例如圖 5 查詢 "D[//F]//G"，執行 TwigList 找出符合的節點為 (D,F,G)={(1.0.0.0.1,1.0.0.0.1,1.0.0.0.1.1)}，同樣透過比對各節點的 Extent，找出 D 的 Extent 中，在 F 和 G 有對應的子孫節點，即可得到查詢結果的目標節點，本例的查詢結果為 G{12, 45}。

3. 系統架構

本研究主要是利用 CIS-X 所提出的 XML 文件索引建構方法，進一步延伸設計，使適用於 MapReduce 系統，目的在於能夠處理大型的 XML 文件。本研究系統架構如圖 6，第一部分是建立 XML Index，另外則是查詢的處理。

3.1 建立 XML Index

建立 XML Index 的過程，是將載入的 XML 文件，透過 XML SAX Parser 產生編碼後的檔案，再將這些檔案載入到 Hadoop MapReduce 的平台，使用 CIS-X 所提出的方法建構索引，最後將這些索引存入 HDFS 中。

整個建構流程大至上可分為兩個大步驟，如圖 7 所示，第一個步驟先解析 XML 檔案並進行分割，一個 XML 文件在解析的過程中，依照拆解規則分割為 n 個 txt 檔案，同時依 CIS-X 的編碼方法和索引架構，記錄相關資訊。分割的檔案大小以 HDFS 的 block size 為基礎，分割為 block size 的倍數，主要目的是可以平均分配每個 DataNode，使執行效能達到最佳狀況。第二個步驟將這些較小且編碼的 txt 檔案依序載入 Hadoop MapReduce 中進行索引檔的建置，分工的方式是依照 block size 的大小分給多個 Map 處理，然後依照 Map 所輸出的 key 送給 Reduce，最後將建立索引檔的結果存入 HDFS(Hadoop Distribute File System)中。

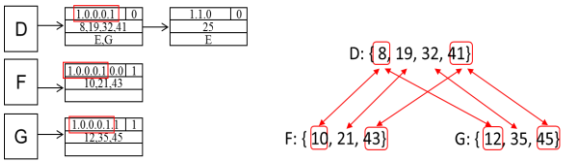


圖 5 查詢 "D[//F]//G" 的比對過程

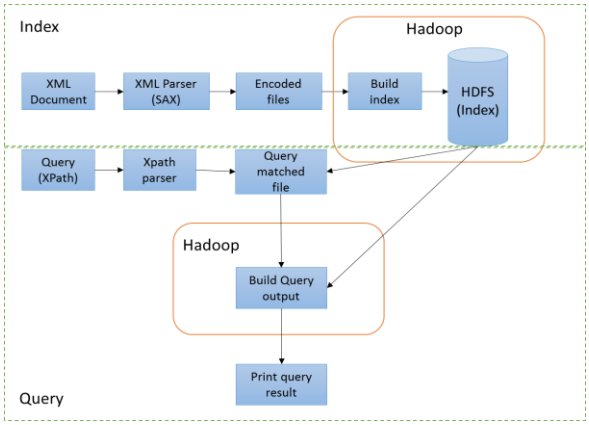


圖 6 系統架構

由於 CIS-X 是將索引建置在記憶體中，若 XML 檔案較大，記憶體不足以應付時，CIS-X 即無法適用。例如 32 位元的 CPU 最多只能支援最大到 4GBytes 的記憶體，而 64 位元的作業系統雖然可支援一倍以上的記憶體 [19]，但其限制仍然在。且一般情況之下，電腦還有其他程式在運作，不可能所有記憶體完全用以記錄索引資料，因此如何利用多台電腦平行處理索引建置工作，便是本研究最主要的目的。

XML 文件具有巢狀階層結構，往往要將 XML 文件轉換成 MapReduce 的 <Key, Value> 設計方式並不容易，例如為 XML 文件建立索引時，當讀取一個標籤的 open tag 時，必須等到相對應的 close tag 被讀取，該標籤在文件中的結構才算完整，更重要的是，下一個被讀取的節點位置才容易判斷，因此經常要將文件全部解讀完畢，才能完整的建立索引。例如圖 8 中，如果文件分割點在</author>，下一個區段由<title>開始，並不易判斷<title>在文件中的位置為何。所以本研究第一個要克服的困難是如何將單一 XML 文件進行分割，以便讓多台機器平行處理。另外一個問題是，如果一次將一個 XML 文件送出進行 MapReduce，雖然可以分工進行索引建置工作，但在最後 Reduce 階段彙整時，太多的資料量仍然會造成堵塞，因此第二個要克服的困難是，如何將 XML 文件分割為多個單位，再依序進入 MapReduce，且每次進行後所產生的索引資料，最後必須能夠正確的合併。

本研究所採用的索引方法是依據 CIS-X 所提出的索引架構，相較於其他文獻常用的區間編碼 [1, 3, 6]，使用深度優先搜尋（Depth-First Search）去程的「開始編號」和回程的「結束編號」，記錄節點在文件中的位置，CIS-X 只使用開始編號，因此不需要等到某節點的結束標籤被讀取後，該節點的編碼才完整，這對於文件分割有很大的助益。根據 CIS-X 建立索引的方式是先將 XML 文件中具有相同路徑的節點合併，形成一個索引結構，並記錄節點在文件中的資訊，以便查詢或還原文件時使用。本研究為了符合 hadoop MapReduce 的 (key, value) 架構，同時充分利用 Reduce 會依 key 將資料群聚，並提供將 value 排序的功能，將節點的類別、標籤、結構，定義為 key，id [-值]為 value 值。說明如下：

- **類別**：分為兩個類別，n 代表一般節點，v 代表內文節點。
- **標籤**：指 XML 文件中的標籤名稱。
- **結構**：是索引節點的 Dewey ID 編碼，代表該節點在索引結構中的位置。
- **id [-值]**：id 為該節點在文件中的編碼，依據深度優先搜尋（Depth-First Search）的順序編碼。若該節點為內文節點，則 [-值]記錄該節點的內文，也就是說類別為 v 的節點才需要記錄 [-值]。

當 XML 文件經 SAX 解析器時，一邊被讀取，一邊記錄相關的資訊（類別、標籤、結構、id[-值]），經過圖 37 的拆解演算法產生如圖 36 中間的字串群，每一行稱之為「一筆索引字串」。例如 "n,a,1-1;2" 這筆索引字串代表 "a" 這個節點為一般節點(n)，其在索引結構（合併相同路徑後的結構）中位置的 Dewey ID 編碼為 "1-1"，且該節點在原文件的深度優先順序為 "2"。從圖 36 中間的字串群中可以看出，"n,a,1-1;2"、"n,a,1-1;14"、"n,a,1-1;28" 都具有相同的 key "n,a,1-1"，表示它們的路徑相同，因此在 MapReduce 時，這三筆索引字串將會被合併在一起。

如同前述，若 XML 文件經解析編碼後所形成的索引字串群儲存於一個檔案，再一次執行 MapReduce 做節點的合併，在 Reduce 階段將會是一個瓶頸。因此在上述的解析編碼階段，當索引字串群達到預設的大小時，就先將索引字串儲存為一個文字檔，如圖 36 右側 Part0.txt，再繼續解析，達到預設大小時，儲存為 Part1.txt，依此類推。而每一個 Part 的大小建議為 HDFS 的 block size 的倍數，可以平均分配每個 DataNode 的工作量，使執行效能達到最佳狀況。

當檔案分割為數個 Part 後，Part*.txt 將依序進行 MapReduce，因為每次只執行整個 XML 文件的一部分資料量，因此可以避免在最後 reduce 時，資料量過大造成記憶體不足的情形。

圖 10 表示 Part0.txt 檔案執行建置索引的過程。首先依 HDFS 的 block size 分為多個 Map，然後根據 Map 後的 key 值在 Reduce 做 value 的合併與排序，最後輸出存入 HDFS 中。如圖 38 中，第一個 Map 的 "n,a,1-1;2" 和最後一個 Map 的 "n,a,1-1;14" 在 Reduce 時，合併為"(n,a,1-1)(2,14)"。

為了方便日後查詢時能快速的讀取相關的索引，因此依以下的架構將索引存入 HDFS 中：第一層以原 XML 文件名稱建立同名的資料夾；第二層分為 path 及 value 兩個資料夾，分別存放一般節點索引及有內文節點索引；第三層同樣設多個資料夾，以節點標籤為資料夾

命名；第四層是文字檔，為同名節點之下，再區分不同的 Dewey ID 編碼，以 Dewey ID 編碼為檔名，檔案的內容為同路徑之下的節點編號。以 "(n,a,1-1)(2,14)" 為例，其存放路徑為 "/dblp/path/r/1-1"，"1-1"檔案內記錄的資料為 "2, 14"。如果是屬性節點，則在標籤前面會帶有「@」符號。

當 Part0.txt 執行完 MapReduce 後，再將 Part1.txt 送入 MapReduce 處理，並將 Reduce 後的結果接續儲存在相對應的檔案內。若 key 值已有對應的路徑和檔案存在，則該階段所合併的編碼附加在檔案的後方，若沒有對應的路徑和檔案，則依序新增路徑相關的資料夾和檔案，並儲存合併的編碼。由於 Part1.txt 的節點編碼都大於 Part0.txt，因此將編碼附加在檔案後方，其結果與將所有的索引字串一次進行 MapReduce（即未做 Part 分檔）相同。

3.2 執行查詢

當使用者下達 XPath [18]查詢語句時，可以透過已經建立好的索引加快處理查詢的速度。一般探討 XML Indexing 的文獻，大部分先透過索引找到目標節點，再將 XML 文件解析一次，同時將以目標節點為 root 的子樹輸出。由於 CIS-X 以及本研究方法詳細記錄 XML 文件的資訊，因此不需要進行文件解析步驟，可以透過 Index 將原文件重組。整個查詢的過程可以分為尋找目標節點和輸出結果（包含目標節點及其子孫節點）兩個步驟。由於第一個步驟透過 Index 可以很快地完成，反而比較費時的是輸出結果的步驟，必須找到所有子孫節點，並依原文件順序重新組合，因此本研究設計利用 MapReduce 來執行第二步驟的工作。

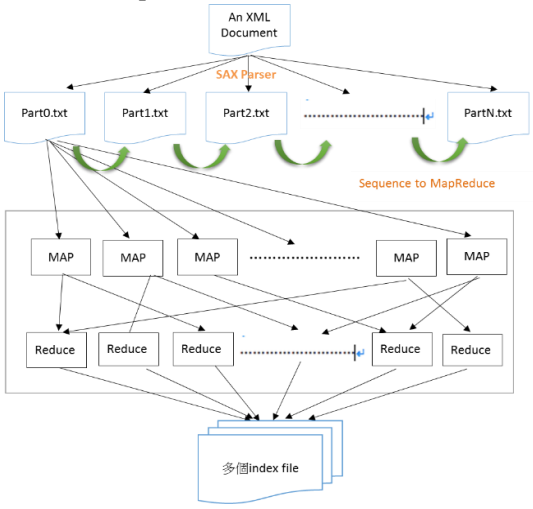


圖 7 XML Index 建構流程

```
<DBLP>
  <book>
    <author>John</author>
    <title>Cloud Computing</title>
    .....
</DBLP>
```

圖 8 一個 XML 文件範例

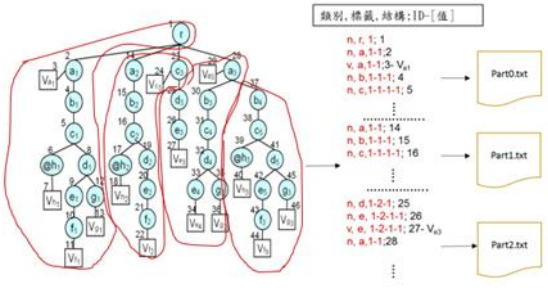


圖 9 XML 文件編碼及拆解示意圖

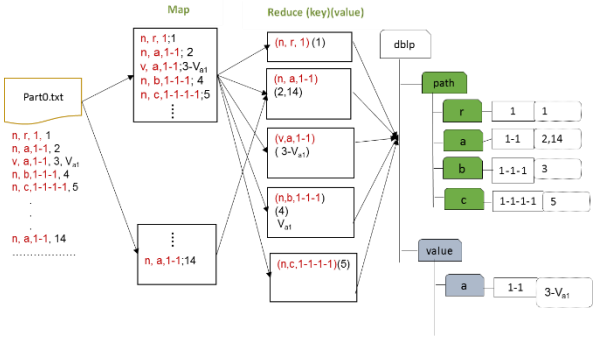


圖 10 MapReduce 索引建置過程

如圖 11 有一個 XPath 的查詢語句 "/r//c//e"，第一個步驟透過類似 2.3.2 節所描述的查詢方法，找到在 HDFS 中符合節點 e 的有 //path/e/1-1-1-1-2-1，目標節點 id「9, 20, 33, 42」，和//path/e/1-2-2-2，目標節點 id 「26」。

至於第二步驟 MapReduce 工作分配的方式，直覺是採用垂直的分工概念，亦即以一個目標節點為一個工作單位，再往下查詢子孫節點，組合成部分結果，最後將所有部分結果合併成最終結果。以圖 11 為例，查詢語句 "/r//c//e" 的目標節點 id 為「9, 20, 33, 42, 26」，若使用垂直分工方式，則分為五個工作單位，由一個 DataNode 執行目標節點 9 及其子孫節點的搜尋和組合工作，一個 DataNode 負責目標節點 20 …依此類推，但這樣的分工方式會造成多數 DataNode 要讀取重疊比率很高的索引檔，例如圖 11 中，有四個 DataNode 都要讀取 //path/e/1-1-1-1-2-1 這個檔案，這四個

DataNode 也都會去讀取 //path/f/1-1-1-1-2-1-1 和//value/f/1-1-1-1-2-1-1 兩個檔案，除了可能會造成互相等候 I/O 的狀況外，每一個 DataNode 都只需要檔案的其中一小部分資料，如此資料的擷取費時費工，故垂直分工並不是一個有效率的方式。

因此本研究採用水平的分工方式，前置作業是先以目標節點的 Dewey ID 編碼為基準，搜尋其子孫節點的檔案，即 Dewey ID 編碼的檔名中，含有目標節點 Dewey ID 編碼為前置詞者。如圖 11 中，符合查詢的目標節點檔案有 //path/e/1-1-1-1-2-1 和 //path/e/1-2-2-2，前者有子孫節點 //path/f/1-1-1-1-2-1-1 和 //value/f/1-1-1-1-2-1-1，後者有子孫節點 //value/e/1-2-2-2。之後以一個檔案為一個工作單位，執行 MapReduce。由於最後仍然需要以目標節點 id 為 root，將子樹重組，因此在 Map 階段，必須完成判斷每一個節點 id 屬於哪一個目標節點 id ，以目標節點的資料為 key (即 "DeweyID id")，以節點資料為 value (即" DeweyID id-標籤/值")，在 Reduce 階段便可依相同的 key 做合併，並依 value 的值進行排序，最後即可重組回原文件的片斷。

以圖 11 為例，共計有符合查詢的目標節點檔案//path/e/1-1-1-1-2-1 和 //path/e/1-2-2-2，加上子孫節點及內文節點檔案 //path/f/1-1-1-1-2-1-1、//value/f/1-1-1-1-2-1-1 和 //value/e/1-2-2-2 五個，因此分為五個工作單位，每一個工作單位在 Map 時，整理為<key, value> 輸出，例如 //path/f/1-1-1-1-2-1-1 中有三個 id「10, 21, 43」，如 2.3.2 節的介紹，透過 id 的比對，可以得知 id 10 為 id 9 的子孫節點，id 21 為 id 20 的子孫節點，id 43 為 id 42 的子孫節點，形成三筆 <key, value> 記錄，分別為 <(1-1-1-1-2-1 9), (1-1-1-1-2-1-1 10-f)> 、<(1-1-1-1-2-1 20), (1-1-1-1-2-1-1 21-f)> 、<(1-1-1-1-2-1 42), (1-1-1-1-2-1-1 43-f)>。在 Reduce 階段時，依 key 做合併，再依 value 值的順序重組原文件片斷。例如 key 為 (1-1-1-1-2-1 20)的資料有 id 為 20, 21, 22 三筆，會合併並依順序由小到大排列，再透過 Dewey 判斷節點的關係，即可組合為在原文件的模樣，圖 11 以樹狀結構表示，但實際運作時，是直接輸出符合 xml 格式的文字檔。

4. 系統實作與實驗結果分析

4.1 CIS-X 延展性問題探討

CIS-X 適用於為單機版，由於單機版會受到單一主機環境的限制，例如 32bits 的作業系統或 JAVA 最多只能使用 2^32=4G RAM，故當檔案太大時，就會面臨記憶體不足的問題。為了測試 CIS-X 在單機版的限制，本論文設計了以下的實驗，實驗環境作業系統為 Win7 64 bits，記憶體 8G，JVM 為 Jre6 64 bits，CPU 為四核心 i7 處理器。

本實驗使用 DBLP [14] 為資料集，在本文撰寫時，DBLP 完整下載的檔案為 1.2G，為了測試 CIS-X 在單機環境之下，能夠處理的檔案大小，本實驗將原本的 1.2G 擷取部分資料做為實驗集，分別擷取了 200M、400M、490M 以及 510M 四個不同的檔案大小作為實驗集。其執行結果顯示 200M、400M、490M 處理時間分為 13 秒、32 秒、38 秒，當檔案超過 500M 時即發生記憶不足或當機的現象，由此可知，雖然過去的實驗證明 CIS-X 處理小檔案時是非常有效率的，但當檔案太大，導致記憶體無法承載時，CIS-X 即無法完成 index 的建置。

4.2 開發工具與實驗環境

本研究語言為 JAVA，開發工具為 eclipse plugin hadoop，實驗環境為 6 台虛擬機器，node1~node6，分別建置在 3 台實體機器上，node1 為 NameNode，node4 為 Secondary NameNode，node2, node3, node5 及 node6 為 DataNode。系統開發環境如表 1 所示，分別列出實體機器及虛擬機器的配置。

4.3 實驗資料集

由於本研究方法主要是為了解決 XML 資料集過大時，造成 index 處理過慢或無法執行的問題，因此必須選用較大型的 XML 檔案作為測試資料，但因自行架設的 Hadoop 系統僅有 6 台虛擬機器，因此若資料集太大處理時間會較長，將造成反覆實驗時，要花費太多時間等候結果的不便。因此選擇超過 1G，但不算太大的 DBLP 資料集。DBPL (Digital Bibliography & Library Project，數字書目索引與圖書館項目) 資料集 [14] 由德國特里爾大學負責開發維護，其內容主要為計算機領域的相關書籍及文獻，線上的 DBLP 資料集會不斷的更新，隨著時間而逐漸增加，本論文撰寫期間所下載的檔案大小為 1.2G（下載時間是 2014/1/9），一般節點數為 29,242,336，內容節點數為 26,619,477 ，屬性節點為 7,325,278。

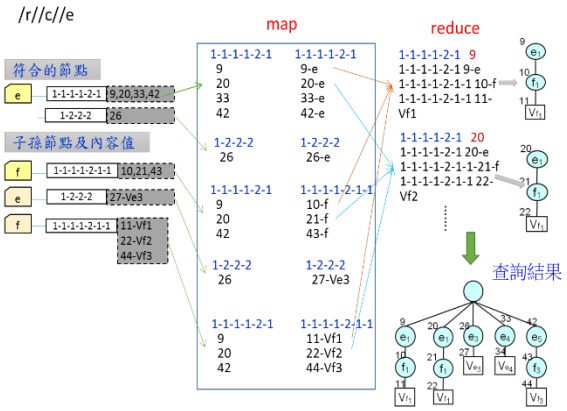


圖 11 MapReduce 查詢處理過程

4.4 索引方法

第一階段將 xml 分割成多個 txt 檔，檔名依順序編號為 Part*.txt，依照 DataNode 數以及 block size 作為分割的依據。本實驗環境有 4 個 DataNode，每個 DataNode 的 block size 64MB，故分割檔案大小為每一個 64MBx4=256MB。因此每個檔案進入 MapReduce 時，一個 block 會分別被分配給一個 DataNode，如此可以讓四個 DataNode 的工作平均分配，以達到最大的效益。實驗資料集為 1.2G 的 DBLP，完成解析、編碼、分割 XML 文件的時間約為 4 分 30 秒，此時間包含寫檔時間。

第二階段將每個 Part*.txt 檔依序利用 Hadoop MapReduce 產生索引檔，索引檔案結構第一層分為兩個資料夾，path 及 value，path 是存一般節點(含屬性節點)的索引，value 則是內文節點的索引，Path 及 value 的下一層為依節點的名稱再建立資料夾，檔案名稱帶有 “@” 代表此節點為屬性節點，1.2G 的 DBLP 建立完 index 後，此路徑下共有 46 個資料夾，代表著此 XML 有 46 個不同的節點名稱。節點名稱的下一層為存放索引內容的檔案，如圖 12 為路徑 “/XBRL/path/author/” 的內容，以 Dewey ID 編碼為檔案名稱（Dewey ID 編碼代表節點在索引結構的位置），檔案內容依據 path 或 value 而有不同的值。path 之下的 Dewey ID 編碼檔案內記錄相同結構的一般節點 id 編碼 (圖 13a)；value 之下的 Dewey ID 編碼檔案內，記錄相同結構的內文節點 id 編碼和內文(圖 13b)。

此實驗共有四個 DataNode，MapReduce 執行時，每個 job 共有 4 個 map 及 1 個 reduce，共執行了 2 分 55 秒，事實上此時間大部分都是 I/O 所花費的時間，Map 的時間 48~51 秒不等，最後 I/O 時間的時間就需要花上 1 分多鐘。

由於本研究提出的方法是將 XML Index 存放在 HDFS 中，並將此 XML 檔案預先拆解，所以不同於 CIS-X，在執行中會有 I/O 的時間及預處理的時間。MapReduce 運算適用於大檔案的處理，對於較小的檔案並沒有比較優勢，故本研究方法對於小檔案的處理效能並沒有比較好，但是可以解決 CIS-X 無法處理大檔案的問題，如 4.1 節的實驗顯示，當檔案大小為 510M 時，CIS-X 已經無法處理，但本研究方法可以處理 1G 以上的資料集。

4.5 索引方法延展性探討

本實驗資料集為 DBLP(1.2G)，因為本實驗只有四個 DataNode 數，故設定拆解後每一個 Part 檔案為 256M，如果 DataNode 數越多，單一 Part 檔案就可以設越大，故一個 XML 拆解後的 Part 檔案數就越少。由於每一個 Part 檔案會執行一次的 MapReduce，而每一次的 MapReduce 都需要有開啟 Map 的時間，約 20~30 秒，所以當拆解後的 Part 檔案越少時，執行速率會加快。

表 2 系統開發環境

	實體機器	虛擬機器
作業系統	lunux	lunux
CPU	AMD Opteron™ Processor 6128 * 2	2 core
RAM	32G	4G
硬碟大小	8 顆 500G 做 raid0	50G

Name	Type	Size
0-0-2	file	20.92 MB
0-1-2	file	11 MB
0-2-2	file	31.23 MB
0-3-2	file	126.1 KB
0-4-2	file	0.04 KB
0-5-2	file	45.38 KB

圖 12 /DBLP/path/author/

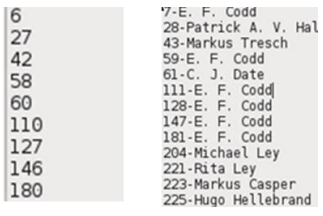


圖 13 a: /DBLP/path/author/0-0-2 ; b: /DBLP/value/author/0-0-2

例如本實驗更改設定，將每一個 Part 檔案定為 256*2=512M（設定為 256 的倍數也是為了讓每個 node 的工作可以平均分配），此拆解後 Part 檔案個數只剩 5 個檔案，但因為每個 Part 檔案太大，所以在 Ruduce 時出現記憶體不足。所以本實驗改為將 Part 檔案縮小為 256M 的一半，128M，共拆成 18 個 Part 檔案，實驗結果如圖 14。雖然檔案較小每一個 Part 執行的時間較短，但因為檔案數量較多，需要執行比較多個 job，所以執行的總時間會比較長。另外當 Part 檔案為 128M 的時候，只有兩個 block 大小，所以只需使用到兩個 DataNode，因此 DataNode 數為 4 個和 2 個的執行時間差不多，因為即使使用 4 個 DataNode，有 2 個 DataNode 是呈現閒置狀態。

為了顯示在 Hadoop 上 DataNode 數量會影響其效能，本研究使用了 1 個、2 個及 4 個 DataNode 進行實驗。實驗資料集是利用原始的 DBLP (1.2G)再節取部分資料附加在檔案後面，產生 1.5G、1.9G 及 2.4G 的資料集進行測試，實驗結果如圖 15 所示，使用 1 個 DataNode 時，會因為 memory 的不足發生中斷，使用 2 個 DataNode 之後就可以順利完成，2 個 DataNode 及 4 個 DataNode 執行時間差約 3 分鐘，由此可知 DataNode 數量越多，可以存取越大的資料量，因此執行速度越快。

由於每一個 Part 檔案都是固定大小，所以每次 MapReduce 執行的時間也相近，如圖 15 所示，當 DataNode 數一樣的時候，執行時間隨 XML 檔案大小（或 Part 檔案個數）而增加。由此可知，借由將 XML 檔案分割為多個 Part 檔，分批且平行處理，無論原 XML 檔案有多大，只需要視執行環境調整 Part 檔案大小，以及 DataNode 個數，本研究所提出的方法都可以完成索引的建置，不受單機記憶體容量的限制。因此本方法具有良好的延展性。

4.6 查詢方法

整個查詢的過程可以分為尋找目標節點和輸出結果（包含目標節點及其子孫節點）兩個步驟，而第二個步驟是相對比較費時的，因此本研究設計利用 MapReduce 來執行第二步驟的工作。首先利用之前所產生的 index 資料，找出符合的節點以及子孫節點（含一般節點和內文節點）所存放的位置，此部分是單台機器來處理，並沒有透過 MapReduce，原因包括：(1) 透過索引可以很快地找到目標節點的位置，工作量並不大；(2) MapReduce 需要較長的前置時間，除非必要，否則使用 MapReduce 反而效率不佳。(3) 即使 Hadoop 有提供設定檔案複製數量的功能，但除非複製的檔案和 DataNode 數一樣，否則平行處理時，會搶資源的狀況，效率會受到影響。例如當 query 為 "//title/sup"時，圖 16(a)，找到符合的節點檔案為 "0-0-3-0-0"、"0-0-3-1"、…，再蒐尋其子孫節點檔案，整理如圖 4 16(b)，後面為目標節點或其子孫節點，前面是所屬目標節點，用來作為 Reduce 合併時的 key。符合節點以及其子孫節點的存放位置確定後，第二部分將這些符合的檔案利用 MapReduce 判斷各節點之間的關係，找出其之下的子結構樹，然後利用相同的祖先關係合併後，成為一個完整個 XML 樹狀結構輸出。圖 17 顯示"//title/sup" 部分查詢結果，執行時間共 39 秒。

本實驗再測試多個查詢執行的情形，其 XPath 語句如表 2 所示。由圖 18 可以看出，單機找出目標節點所需的時間很短，因為本研究的索引方法是要經過相同路徑的節點合併，所以所需比對的資料量較少。而 MapReduce 產生最終結果的時間主要受到輸出結果的檔案大小以及目標結點數的影響。由於每一個節點會送給一個 Mapper 處理，所以節點數越多所需的 Mapper 就越多，每使用一個 Mapper 就會需要多一個開啟 Map 的時間，所以，雖然“//title/sup” 的輸出結果的檔案雖然比“/dblp/incollection/editor”小，但所需的時間反而比較長，不過這個因素的影響相對較小，影響時間最大的因素在於最後 reduce(寫檔)的時間，對照圖 18 和圖 19，“/dblp/phdthesis/author”產生的檔案有 253.41M，所以所需的時間也會相對較長，需要 4 分 24 秒。圖 18 和圖 19 可以看出查詢結果時間大致和輸出結果的大小成正比。

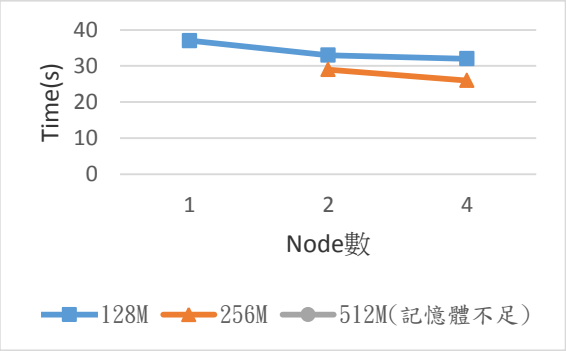


圖 14 檔案大小與執行時間

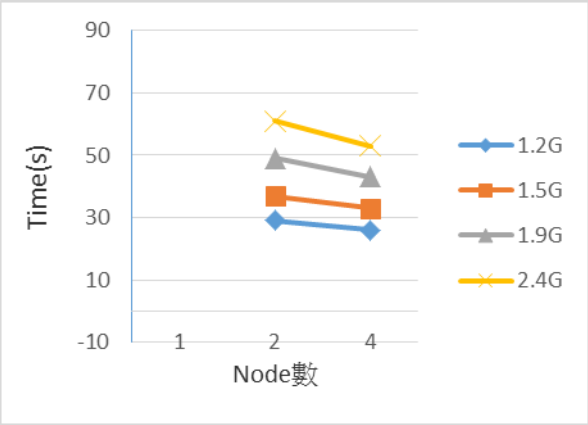
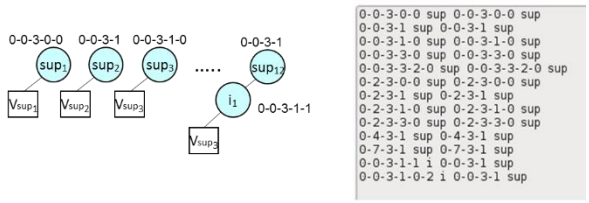


圖 15 DataNode 數與執行時間



a b

圖 16 (a)查詢"//title/sup"符合的索引檔
(b)MapReduce 的<key, value>

```
</sup>
1
-control cost and applications for the placement
<sup>
</sup>
3
+y
<sup>
```

圖 17 "//title/sup"查詢結果

表 2 查詢語句

編號	XPath
1	//title/sup
2	/dblp/phdthesis/author
3	/dblp/incollection/editor
4	/dblp/article/publisher
5	/dblp/incollection/year

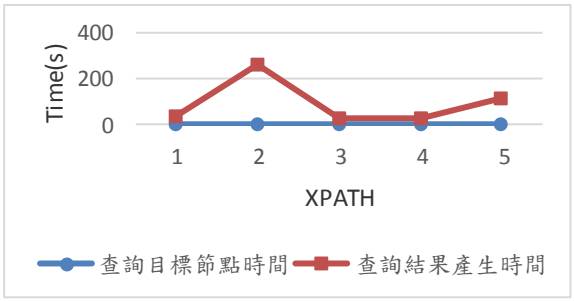


圖 18 各查詢目標節點查詢及結果產出時間

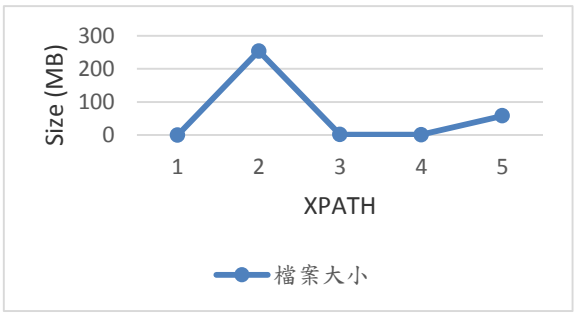


圖 19 各查詢檔案輸出大小

5. 結論與未來研究

本研究為 CIS-X 方法的延伸，提出並改善原方法在處理大量資料時速度過慢及記憶體空間的問題，本研究方法進行兩個方向探討：

第一個方向是針對建立 index 提出適用於 MapReduce 的方法，此方法利用檔案分割的方式將 xml 檔案分割成若干個檔案，此檔案大小依據 Hadoop 環境的變數不同，選擇為可以平均分配工作為原則，因為本研究有分割檔案，所以適用於各種大小以及各種形式的 xml 檔案，使得建立 index 更有延展性。第二個方向是利用本研究提出的 index 方法，提出適用於本索引結構及適用於 MapReduce 環境的查詢方法。此查詢方法利用 Hadoop 平行處理的特性，將單機處理找出符合的節點的結果，平行化的送給 Hadoop DataNodes 個別找出其子結構或內容值，並利用（key, value）的特性將共同的祖先節點合併，成為一個完整的 XML 結構輸出。

XML 資料集如同關聯式資料庫一般，也是會有更新資料內容的時候，如果依照本研究方法，如有資料更新則需再建立索引一次，當檔案大時，每一次建立索引的時間則需要很長，為了減少重新建立索引的時間，未來可以繼續探討有效更新節點的方法，改善本研究無法支援更新索引檔的缺點。

致謝

本研究接受國科會研究計畫經費補助，編號：NSC 102-2221-E-005-070。在此感謝國科會提供相關的資源與協助，使得本研究能夠順利進行與完成。

參考文獻

[1] Al-Khalifa, S., Jagadish, H. V., Koudas, N., Patel, J., Srivastava, M. D., and Wu, Y., "Structural joins: a primitive for efficient XML query pattern matching," in Proceedings of ***the 18th International Conference on Data Engineering*** (ICDE '02), Washington, DC, USA, 2002.

[2] Brenes, S., "Structural summaries for efficient XML query processing," in Proceedings of ***the 2008 EDBT Ph.D. workshop*** (Ph.D. '08), New York, NY, 2008.

[3] Catania, B., Ooi, B. C., Wang, W. and Wang, X., "Lazy XML updates: laziness as a virtue, of update and structural join efficiency," in Proceedings of ***the 2005 ACM SIGMOD international conference on Management of data*** (SIGMOD '05). ACM, New York, NY, USA, 2005.

[4] Chen, Q., Lim, A. and Ong, K. W., "D(k)-index: an adaptive structural summary for graph-structured data," in Proceedings of ***the 2003 ACM SIGMOD international conference on Management of data*** (SIGMOD '03). ACM, New York, NY, USA, 2003.

[5] Chen, Q., Lim, A. and Ong, K. W., "Enabling structural summaries for efficient update and workload adaptation," ***Data & Knowledge Engineering,*** Vol.64, Iss.3 pp. 558-579, 2008.

[6] Chien, S. Y., Vagena, Z., Zhang, D., Tsotras, V. J. and Zaniolo, C., "Efficient structural joins on indexed XML documents," in Proceedings of ***the 28th international conference on Very Large Data Bases*** (VLDB '02), 2002.

[7] Dede, E., Fadika, Z., Gupta, C. and Govindaraju, M., "Scalable and Distributed Processing of Scientic XMLX data," in Proceedings of ***the 2011 IEEE/ACM 12th International Conference on Grid Computing*** (GRID '11), Washington, DC, USA, 2011.

[8] Goldman, R. and Widom, J., "DataGuides: enabling query formulation and optimization in semistructured databases," in Proceedings of ***the 23rd International Conference on Very Large Data Bases***, 1997.

[9] Hsu, W.-C. and Liao, I-E., "CIS-X: A compacted indexing scheme for efficient query evaluation of XML documents," ***Information Sciences***, Vol. 241, pp. 195-211, 2013.

[10] Qin, L., Yu, X. J. and Ding, B., "Twiglist: make twig pattern matching fast," in Proceedings of ***the 12th international conference on Database systems for advanced applications*** (DASFAA'07), Ramamohanarao Kotagiri, P. Radha Krishna, Mukesh Mohania, and Ekawit Nantajeewarawat (Eds.), Springer-Verlag, Berlin, Heidelberg, 2007.

[11] Zinn, D., Kohler, S., Browers, S. and Ludascher, B., "Parallelizing XML data-streaming workflows via MapReduce," ***Journal of Computer and System Sciences***, pp. 447-463, 2010.

[12] 陳雨霖，***NCIM:植基於節點群聚的有效率 XML 索引方法***，中興大學資訊科學與工程研究所，碩士論文，2009。

[13] "A Simple Example to Demonstrate how does the MapReduce work," [Online] http://xiaochongzhang.me/blog/?p=338. [Accessed 11 Jan 2014].

[14] "dblp: DBLP Bibliography," [Online] http://dblp.uni-trier.de/xml/. [Accessed 9 Jan 2012].

[15] "MapReduce Tutorial," [Online] https://hadoop.apache.org/docs/r1.0.4/mapred_tutorial.html. [Accessed 11 Jan 2014].

[16] "Welcome to ApacheTM HadoopTM! ," [Online] http://hadoop.apache.org/. [Accessed 11 Jan 2014].

[17] "Wikipedia, Apach Hadoop," [Online] https://en.wikipedia.org/wiki/Apache_Hadoop. [Accessed 11 Jan 2014].

[18] "XML path language(XPath) verson 3.0," W3C Recommendation, 2013. [Online] http://www.w3.org/TR/xpath-30/. [Accessed 11 Jan 2014].

[19] "記憶體實體容量的支援度," [Online] http://www.synnex.com.tw/asp/fae_qaDetail.asp?from_prg=&topic=FAE&group=&parent=&classifyid=01997&seqno=19240. [Accessed 11 Jan 2014].

[20] "雲端開發測試平台"，財團法人資訊工業策進會，2012，[Online] http://www.cloudopenlab.org.tw/ccipo_industryDefinition.do. [Accessed 11 Jan. 2014].