다중 XML 문서 인덱싱을 위한 전역 인코딩 기법[1)]

배진욱[01], 문봉기[2], 이석호[1]
서울대학교 전기컴퓨터공학부[1], 아리조나대학교 컴퓨터공학과[2]
jinuk@db.snu.ac.kr, bkmoon@cs.arizona.edu, shlee@snu.ac.kr

Global Encoding Technique for Indexing Multiple XML Documents

Jinuk Bae[01], Bongki Moon[2], Sukho Lee[1]
School of Electrical Eng. and Computer Science, Seoul National University[1]
Dept. of Computer Science, University of Arizona[2]

요 약

지금까지 제안된 구조조인 알고리즘들은 하나의 XML 문서에 대해 복잡한 질의를 빠르게 처리할 수 있다는 장점이 있다. 하지만, 다중 문서를 처리할 때 각 문서에 부여된 문서식별자에 의해 문서별 질의 처리를 하기 때문에, 문서의 수가 증가한다면 질의 처리 시간도 길어진다는 문제점이 발생한다. 이 논문에서는 이 문제를 해결하기 위해 XML 문서를 XMAS 트리로 병합한 뒤 전역적으로 인코딩을 하는 기법을 제안한다. XMAS 트리는 각 문서의 구조 정보를 유지한 채 공통된 부분을 공유하는 트리이다. 이 공유에 의해서 질의 처리시에 성능 향상을 얻을 수 있다. 실험 결과, 선형 질의에 대해 수백 배, 가지모양 질의에 대해 수십 배 빠르게 질의를 처리할 수 있었다.

1. 서론

XML이 정보 표현과 교환의 표준으로 자리잡음에 따라 다양한 응용을 위한 XML 스키마가 제안되고 있으며, 제안된 XML 스키마를 기반으로 하는 수많은 XML 문서들이 작성되어 데이타베이스에 축적되고 있다.

넘버링 스킴을 이용한 구조 조인 알고리즘들[1,2,3,4,5]은 하나의 XML 문서에 대해 선형 또는 가지모양 질의(path or twig query) 같은 복잡한 질의 처리에 우수한 성능을 보여주었다. 이 방법은 다수의 XML 문서를 처리하기 위해 각 문서마다 문서 식별자 docid를 부여하고 독립적으로 넘버링 스킴에 의해 숫자를 할당한다. 그런 후, 문서 식별자에 의해 각 문서별로 질의를 처리한다. 그림 1은 예를 보여준다. 그림 1의 두 트리는 각각 독립적으로 넘버링 스킴에 의해 각 노드마다 한 쌍의 번호가 부여되었다. 하지만, 질의 처리 시간이 문서의 수에 비례하여 증가한다는 문제점이 있다.

이 논문에서는 이 문제를 해결하기 위해 XMAS 트리(XML document Merging And Storing tree)에 의한 XML 문서 병합 기법을 제안한다. 문서 병합 기법이란 루트 노드가 동일한 문서들을 각 문서 내부의 구조를 유지하면서, 하나의 논리적인 XMAS 트리로 병합하는 방법을 말한다. 그 후, XMAS 트리에 넘버링을 하면 문서들 전체에 전역적인 숫자를 부여할 수 있다. 이 병합 과정 중 트리들의 공통 노드들이 공유되면서 질의를 처리할 때 디스크 접근 회수를 줄여준다.

논문의 구성은 2장에서 연구 배경을 살펴본다. 3장에서는 XMAS 트리에 의한 병합 기법을 제안한다. 4장에서 실험을 통해 성능을 평가하고, 마지막으로 5장에서 결론을 맺는다.

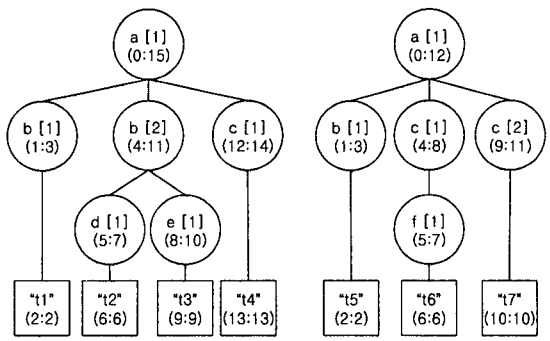

그림 1 1.xml과 2.xml에 대한 지역 인코딩 기법

2. 배경

2.1. 데이타 모델

XML은 데이타를 표현하기 위해 트리 구조를 채택한다. XML 데이타 트리에는 엘리먼트 노드와 텍스트 노드라는 두 종류의 노드가 있다. 엘리먼트 노드는 엘리먼트와 애트리뷰트

1) 본 연구는 2005년도 두뇌한국21사업과 정보통신부의 대학 IT연구센터(ITRC) 지원을 받아 수행되었습니다.

를 포괄하는 용어이다. 엘리먼트 노드에는 암시적으로 컨텍스트 포지션(context position)이 정해진다. 컨텍스트 포지션이란 XML 질의 언어인 XPath[6]에서 정의한 개념으로, 하나의 노드가 가지고 있는 동일한 이름의 자식 노드들에 대해, 문서에 나타나는 순서에 따라 증가하는 양의 정수 값을 부여한 것이다. 한편, 텍스트 노드는 한 쌍의 태그에 의해 둘러싸이거나 애트리뷰트의 값으로 쓰여진 문자열을 말한다.

2.2. 넘버링 스킴

XML 데이타 트리의 노드들은 각 여는 태그와 닫는 태그가 문서에서 나타나는 순서에 따라 오름차순에 의한 숫자를 부여받는다. 예를 들어, XML 문서 <a>t1<c>t2</c>에서 <a>에는 1, 에는 2, t1에는 3 등으로 숫자를 부여한다. 그 결과, <a>와 에 의한 루트 엘리먼트 노드는 (1:8)을, 와 에 의한 엘리먼트 노드는 (2:4)를, 텍스트 노드 t1은 (3:3)이 할당된다. 여기에서 한 노드에 부여된 한 쌍의 숫자를 각각 begin과 end로 나타내는데, 만약 두 노드 n1과 n2가 조상-자손 관계라면 다음 식을 만족하며, 역도 성립한다.

n1.begin < n2.begin and n1.end > n2.end

3. XMAS 트리에 의한 병합 기법

3.1. XMAS 트리

이 절에서는 여러 문서들을 병합하기 위한 XMAS 트리를 정의한다. 이를 위해 먼저 두 가지 형태의 경로를 정의한다.

[정의 1] 엘리먼트-위치 경로란 XML 트리에서 엘리먼트 노드(e_i)와 해당하는 컨텍스트 포지션(p_i)을 덧붙여 나타내는 경로 $e_1[p_1]/e_2[p_2]/.../e_n[p_n]$를 말한다. ■

그림 1에서 a[1]/b[1]와 a[1]/b[2]/d[1]은 엘리먼트-위치 경로의 예이다.

[정의 2] 인스턴스 경로란 루트에서 시작하는 엘리먼트-위치 경로와 이 경로에 의해 도달하는 텍스트 노드 t의 결합으로, $/e_1[p_1]/e_2[p_2]/.../e_n[p_n]/t$를 말한다. ■

그림 1에서 a[1]/b[1]/"t1"와 a[1]/b[2]/d[1]/"t2"는 인스턴스 경로의 예이다.

이제, 정의된 경로들을 기반으로 XML 데이타 트리를 병합하기 위한 XMAS 트리를 정의한다.

[정의 3] 루트 노드의 태그가 동일한 데이타 트리들 $T_1, T_2, ..., T_n$이 주어졌을 때 각 문서들에 존재하는 인스턴스 경로들을 프리픽스(prefix)를 공유하도록 합친 트리를 XMAS 트리라 한다. 이 때, 노드들은 넘버링 스킴에 의해 한 쌍의 숫자 (begin:end)가 주어지며, 각 텍스트 노드에는 해당 인스턴스 경로가 어느 문서에서 왔는지를 나타내는 문서 식별자 docid가 부여된다. ■

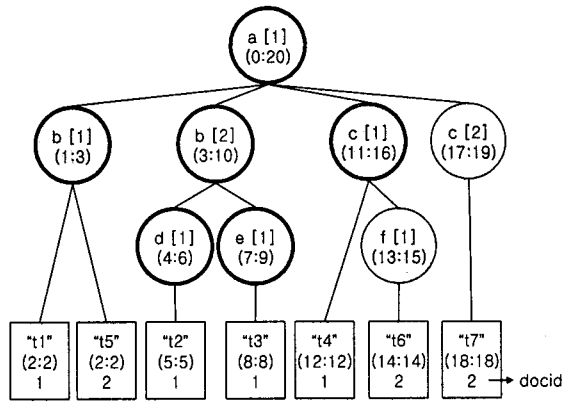

그림 2 XMAS 트리에 의한 전역 인코딩 기법

그림 2는 그림 1의 두 트리를 병합한 XMAS 트리이다. 그림 2의 다음 인스턴스 경로들을 통해 어떻게 XMAS 트리가 생성되었는지를 살펴본다.

IP1: /a[1]/b[1]/"t1"
IP2: /a[1]/b[1]/"t5"
IP3: /a[1]/b[2]/d[1]/"t2"

IP1과 IP3은 1.xml에 존재하며, IP2는 2.xml에 존재하는 인스턴스 경로이다. IP1과 IP3은 /a[1]/b[1]을 공통적으로 가지고 있다. 그러므로, 그림 2에서는 이 부분을 공유한 채 b[1]의 자식으로 텍스트 노드 "t1"과 "t5"가 나타난다. 그리고, 각 텍스트 노드에는 문서 식별자 docid가 표시되었다. IP1와 IP2는 /a[1]/b[1]이 공통되지만, IP3는 이 두 경로와 /a[1]만이 동일하다. 그래서, XMAS 트리에서 a[1] 노드는 b[1]과 b[2]를 자식 노드로 가진다.

한 데이타 트리에 있는 인스턴스 경로의 수와, XMAS 트리에서 이 데이타 트리로부터 온 인스턴스 경로의 수는 동일하다. 또한, XMAS 트리에 존재하는 인스턴스 경로들의 수는 각 문서들에 존재하는 인스턴스 경로들의 합과 같다. 예를 들어, 그림 1에는 총 7개의 인스턴스 경로가 존재하는데, 그림 2의 XMAS 트리에도 동일한 수의 인스턴스 경로가 존재한다. 반면에, 루트로부터 시작하는 엘리먼트-위치 경로의 경우 그림 1에는 7개가 존재하나, XMAS 트리에는 6개만 있다. 이렇게 줄어든 이유는 서로 다른 문서에서 온 동일한 엘리먼트-위치 경로가 XMAS 트리를 생성할 때 병합되었기 때문이다.

3.2. XMAS 트리를 저장하기 위한 관계 스키마

XMAS 트리는 XMAS 테이블이라는 이름의 테이블에 저장된다. XMAS 테이블은 크게 엘리먼트 테이블과 텍스트 테이블로 나뉜다. XML 문서에 존재하는 각 태그 tag에 대해, 하나의 엘리먼트 테이블 E_{tag}와 텍스트 테이블 T_{tag}이 생성된다. 엘리먼트 테이블의 스키마는 $E_{tag}(begin,end,level)$이고, 텍스트 테이블의 스키마는 $T_{tag}(begin,level,docid,value)$이다. 구조 조인 알

고리즘이 요구에 따라, 모든 테이블들은 *begin* 애트리뷰트에 의해 오름차순으로 정렬된다.

3.3. 질의 처리

선형 질의를 처리하는 구조조인 알고리즘[1,2,3,4,5]을 함수 getPathSolution에 의해 나타난다. 이 때, 지역 인코딩 기법에 의해 다중 XML 문서들이 인덱싱되어 있다면, 각 문서 식별자마다 getPathSolution을 호출함으로서 질의를 수행한다. 그에 반하여, 이 논문에서 제안하는 전역 인코딩 기법에 의하면 모든 테이블들이 begin에 의해 정렬되어 있음으로 한 번의 getPathSolution 호출에 의해 질의를 처리할 수 있다.

```
입력: 가지모양질의 Q
입력: XML 문서 T1, T2, ..., Tn를 위한 XMAS 트리 X
begin
        {ps1,ps2,...,psk} = getPathSolution(X, Q);
        foreach docid d (1<= d <= n) do
                mergePathSolution(psd);
        end
end
```

그림 3 가지모양 질의 처리 알고리즘

가지모양 질의를 처리하는 구조조인 알고리즘[1,2,3,4,5]은 두 개의 함수 getPathSolution과 mergePathSolution에 의해 이루어진다. 즉, 먼저 가지모양 질의의 각 경로에 대해 답을 구한 뒤, 이 답들을 가지모양으로 병합함에 의해 질의가 수행된다. 지역 인코딩 기법에서는 각 문서에 대해 두 함수를 호출하지만, 전역 인코딩 기법에서는 한 번의 getPathSolution을 호출하면서 텍스트 노드의 docid에 의해 경로 답을 문서별로 분류한다. 그 후에 mergeSolution을 문서별로 호출함으로써 질의를 수행한다. 이 과정은 그림 3에 보여진다.

4. 실험

지역 인코딩 기법과 전역 인코딩 기법을 비교하기 위해 DBLP 데이타셋[7]과 Chromosome 데이타셋[8]을 이용하였다. 전자는 2002년도 버전으로 얻은 약 27만 개의 화일로, 후자는 128개의 화일로 구성되어있다.

실험을 위해 각 데이타셋 마다 정해진 질의 길이에 대해 데이타셋에 존재하는 모든 질의를 생성하였다. 표 2에 질의셋에 대한 명세가 보여진다. 생성된 질의셋에 대해 표 3에 보여지듯이 지역 인코딩 기법과 전역 인코딩 기법에서의 질의 처리 시간을 측정하였다. 실험 결과, PD 질의셋에 대해 410배에 이르는 성능향상을, TC 질의셋에 대해 23배의 성능향상을 얻었다. 이 성능향상은 3.1절에서 설명하였듯이 엘리먼트 노드의 병합에 의해서 얻어진다. DBLP 데이타셋의 경우 3백만 개의 엘리먼트 노드가 3천 개로, Chromosome 데이타셋의 경우 1.3억 개의 노드가 백만 개로 병합되었다. 다만, 텍스트 노드의 경우 병합이 발생하지 않기 때문에, 선형 질의보다 텍스트 노드를 많이 접근하는 가지모양 질의의 성능향상 정도가 적었다.

표 2 질의셋 명세

	데이타셋	질의	길이	개수
PD	DBLP	선형	3	44
PC	Chromosome	선형	4	214
TD	DBLP	가지모양	2	158
TC	Chromosome	가지모양	3	209

표 3 질의 수행 시간 (초)

	PD	PC	TD	TC
지역	8.2	24999	17	71254
전역	0.02	606	2	3040

5. 결론

이 논문에서는 다중 XML 문서에 대한 질의 처리를 할 때, 지역 인코딩 기법에서 발생하는 문제점을 해결하기 위해 XMAS 트리를 이용하는 전역 인코딩 기법을 제안하였다. XMAS 트리는 XML 데이타 트리들의 구조는 유지한 채 동일한 노드들을 공유함으로써 성능향상을 가져온다. 실험 결과, 선형 질의에 대해 수백 배, 가지 모양 질의에 대해 수십 배의 성능 향상이 있음을 확인하였다.

참고 문헌

[1] C. Zhang, J. Naughton, D. DeWitt, Q. Luo,, and G. Lohman, "On supporting containment queries in relational database management systems", ACM SIGMOD 2001

[2] Q. Li and B. Moon, "Indexing, querying XML data for path expressions", VLDB pp.361-370, 2001

[3] S. Al-Khalifa, H. V. Jagadish, N. Koudas, J. M. Patel, D. Srivastava,, and Y. Wu, "Structural Joins: a primitive for efficient XML query pattern matching", ICDE 2002

[4] N. Bruno, N. Koudas, and D. Srivastava", "Holistic twig joins: optimal XML pattern matching", ACM SIGMOD 2002

[5] H. Jiang, W. Wang, H. Lu, and J. X. Yu, "Holistic twig joins on indexed XML documents", VLDB, pp.273-284, 2003

[6] A. Berglund, S. Boag, D. Chamberlin, M. F. Fernandez, M. Kay, J. Robie, and Jrme Simon, "XML Path Language (XPath) 2.0 W3C Working Draft 16", World Wide Web Consortium, 2002, http://www.w3.org/TR/xpath20/

[7] Michael Ley. DBLP Bibliography. http://www.informatik.unitrier.de/~ley/db/index.html

[8] R. Inpharmatics, "Experimental annotation of the human genome using microarray technology", http://download.rii.com/tech/pubs/nature/chromo22.htm