레벨 정보를 이용한 효과적인 구조 조인 기법
(Effective Structural Joins using Level Information)

김 종 익 †
(Jongik Kim)

요 약 구조 조인은 XML 경로 질의를 처리하기 위한 대표적인 방법 중의 하나이다. 최근의 연구에서는 B+트리 등에 색인된 가로 방향의 엘리먼트 분포 정보를 이용하여 조인 결과에 포함되지 않는 엘리먼트들을 조인 연산에서 배제하는 방법에 초점을 맞추고 있다. 하지만, 이러한 방법은 조인 연산 자체가 매우 복잡해지며 분포 정보를 위한 색인의 부담으로 항상 좋은 성능을 보장하지는 못한다. 본 논문에서는 레벨 정보를 이용하여 조인 결과에 참여하지 못하는 엘리먼트들을 효과적으로 걸러내는 구조 조인 기법을 제안한다. 제안하는 기법의 레벨 정보는 엘리먼트의 세로방향의 분포정보라는 점에서 기존의 기법과 차별화되며 조인 연산을 위한 별도의 색인을 필요로 하지 않는다는 장점을 가진다. 본 논문에서는 실험을 통하여 제안된 기법의 효율성을 보인다.

키워드 : XML, 경로질의, 구조 조인, 레벨 조인

Abstract Structural join is one of the most typical techniques for evaluating XML path queries. Recent researches for structural joins focus on techniques of skipping unnecessary elements using the horizontal distribution information of elements that is indexed on a structure like B+ tree. However, those techniques make the structural join complicated and cannot guarantee efficient join processing due to the overhead of an index structure. In this paper, we propose a new structural join technique that exploits the level information of XML elements. Our technique can skip unnecessary elements using level information, which is vertical distribution information of elements. Through the experimental results, we show that our technique can evaluate structural joins efficiently.

Key words : XML, path query, structural join, level join

1. 서 론

XML(Extensible Markup Language)[1]은 인터넷을 위한 문서의 표준으로 만들어졌지만, 사용자가 태그를 정의할 수 있고 임의의 구조로 태그를 중첩할 수 있는 유연성으로 인하여 인터넷 문서뿐만 아니라 일반적인 데이타의 표현과 교환에 널리 사용되고 있다.

일반적으로 XML 데이타는 트리 형태의 모델로 표현된다. 이때, XML 데이타 내의 엘리먼트는 트리의 노드(node)에 대응되며, 엘리먼트들 사이의 포함관계는 트리의 에지(egde)에 대응된다. 그림 1(a)는 예제 XML 데이타를 보여주며, 그림 1(b)는 해당 XML 데이타를 트리로 표현한 것을 보여준다.

XML 데이타로부터 원하는 정보를 추출하기 위해서 XML을 위한 질의 언어[2]는 데이타 트리 내의 경로를 이용하여 질의를 표현한다. 예를 들어, 그림 1에서 "데이타베이스 내에 있는 모든 바의 이름을 찾아라" 라는 질의는 foodDB//bar/name과 같이 표현된다. 이때, foodDB//bar의 //는 데이타 트리 내에서 foodDB 노드와 bar 노드 사이에 조상-후손 관계가 성립된다는 것을 의미하며, bar/name의 /는 bar 노드와 name 노드 사이에 부모-자식 관계가 성립하는 것을 의미한다.

구조 조인은 XML 질의 내에 있는 조상-후손 관계 또는 부모-자식관계 등의 이진 구조 관계를 만족하는 엘리먼트 쌍을 추출하는 연산을 의미한다. 일련의 구조 조인 연산을 통하여 질의의 결과를 도출할 수 있으므로

· 이 논문은 2007년도 전북대학교 지원 신임교수 연구비에 의해 연구되었음

† 정 회 원 : 전북대학교 전자정보공학부 교수
jongik@chonbuk.ac.kr

논문접수 : 2008년 1월 31일
심사완료 : 2008년 5월 15일

정보과학회논문지: 컴퓨팅의 실제 및 레터 제14권 제6호(2008.8)

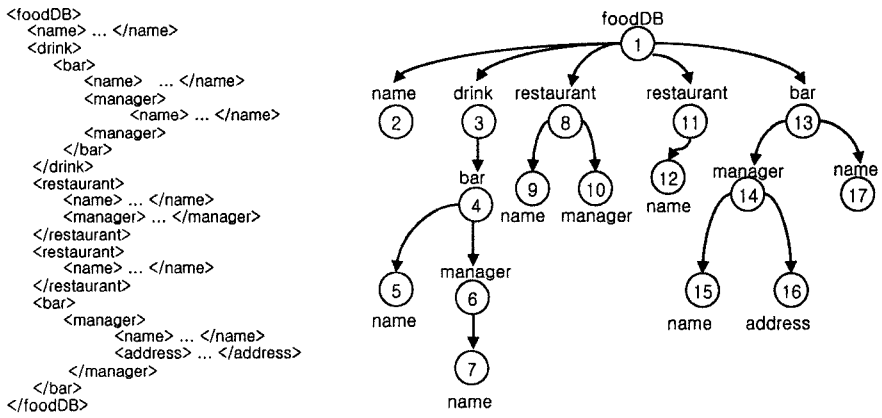


그림 1 예제 XML 문서 및 데이타 트리

구조 조인은 XML 질의 처리의 기본 연산이라고 볼 수 있다. 예를 들어 위의 질의는 foodDB//bar의 구조 조인을 수행한 결과에서 bar엘리먼트들을 추출하고, 추출된 bar엘리먼트들을 이용하여 bar/name의 구조 조인을 수행함으로써 처리될 수 있다.

구조 조인은 XML 질의 처리의 기본 연산이므로 구조 조인을 효과적으로 처리하기 위한 많은 연구들이 진행되었다[3-8]. [3]에서는 기존의 병합 조인과 유사하게 구조 조인을 수행하는 MPMGJN(Multi-Predicate Merge Join) 기법을 제안하였고 [4]에서도 동일한 방식을 사용하는 구조 조인 기법을 제안하였다. [5]에서는 MPMGJN을 일반화한 트리-합병 조인 기법과 구조 조인의 I/O를 최적화한 스택-트리 조인 기법을 제안하였다. 최근에는 조인의 성능을 더욱 향상시키기 위해 조인의 결과에 영향을 미치지 못하는 엘리먼트들을 구조 조인 연산에서 제외시키기 위한 기법들이 제안되었다[6,7]. 이러한 기법들은 내부 또는 외부 색인을 사용하여 불필요한 엘리먼트들을 찾아내지만, 색인의 구축, 탐색 및 유지의 부담이 크며 조인 연산이 복잡해진다는 단점을 가지고 있다. [8]에서는 엘리먼트의 레벨 정보를 이용하는 구조 조인 기법을 제안하여 부모-자식 관계에 대하여 조인의 성능을 향상시켰다. 하지만 조상-후손 관계의 경우 엘리먼트 리스트 내의 서로 다른 레벨의 숫자만큼 엘리먼트 리스트를 반복하여 스캔하는 문제점을 가지고 있어 기존의 구조 조인 기법보다 좋지 못한 성능을 보인다.

본 논문에서는 구조 조인의 결과에 참여하지 못하는 엘리먼트들을 레벨 정보를 이용하여 효과적으로 걸러내는 기법을 제안한다. 제안하는 기법은 부모-자식 관계는 물론이고 조상-후손 관계에 대해서도 엘리먼트 리스트를 한번만 스캔하여 구조 조인을 수행하게 함으로써 기존의 구조 조인 기법보다 항상 향상된 성능을 보장한다.

2. 배경 정보

부모-자식 관계 또는 조상-후손관계에 있는 두 개의 태그 A와 D에 대해, 구조 조인을 수행하기 위해서는 태그가 A인 모든 엘리먼트를 포함하는 리스트인 AList와 태그가 D인 모든 엘리먼트를 포함하는 리스트인 DList의 추출이 선행되어야 한다. 엘리먼트 리스트의 추출은 데이타 내의 각 태그 이름에 대해 해당 엘리먼트 리스트를 B+트리를 이용해 색인해 놓은 역색인을 이용하여 간단하게 이루어진다. 이때, 구조 조인은 정의 1과 같이 정의될 수 있다.

정의 1. 두 개의 엘리먼트 리스트 AList와 DList에 대해, 구조 조인은 a ∈ AList, d ∈ DList이고 a가 d의 부모이거나 조상인 (a, d)의 쌍들을 찾아내는 과정이다.

구조 조인의 수행 시에 두 엘리먼트 사이의 부모-자식 관계 또는 조상-후손 관계를 파악하기 위해서 데이타 내의 각 엘리먼트를 (StartPos, EndPos, Level)의 세 개의 숫자로 표현한다[3]. 이때, StartPos는 데이타의 시작 위치부터 해당 엘리먼트의 시작 태그까지의 단어 수를 나타내며, EndPos는 데이타의 시작 위치부터 해당 엘리먼트의 끝 태그까지의 단어의 수를 나타내고, Level은 해당 엘리먼트의 포함된 깊이(nesting depth)를 나타낸다. AList의 엘리먼트 a와 DList의 엘리먼트 d에 대해, a.StartPos < d.StartPos 이고 a.endPos > d.endPos인 경우 a는 d의 조상 엘리먼트가 되며, a.Level = d.Level - 1인 경우 a는 d의 부모 엘리먼트가 된다. 예를 들어, <A> <B> <C> ··· </C> </B> </A>인 XML 데이타에서 A는 (0,5,1) 값을 가지고 B는 (1,4,2), C는 (2,3,3)의 값을 가지므로 A는 B의 부모이고 C의

조상이며 B는 C의 부모임을 쉽게 알 수 있다.

3. 레벨 조인

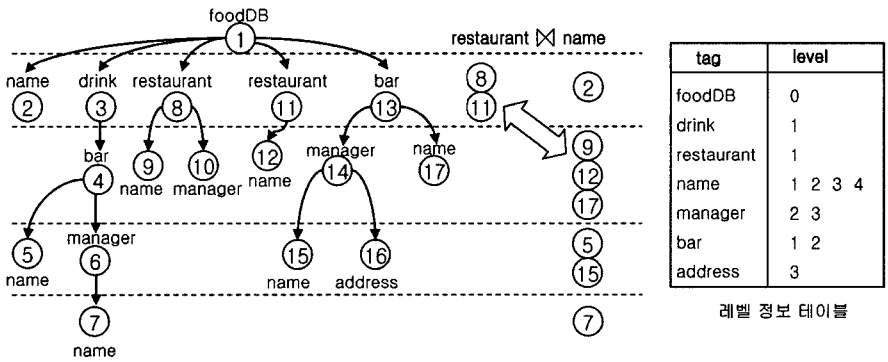
구조 조인 연산의 속도를 향상시키기 위해서는 엘리먼트 리스트에서 결과에 포함되지 않는 일련의 엘리먼트들을 찾아내어 이들을 스캔하지 않고 조인을 계산할 수 있어야 한다. 이러한 일련의 엘리먼트들을 찾아내기 위해 기존의 연구에서는 엘리먼트들의 가로방향의 분포 정보만을 이용하고 있다. 예를 들어, 그림 1에서 bar//name의 구조 조인을 생각해 보자. 4번과 13번 bar 엘리먼트 사이에 있는 9번과 12번 name 엘리먼트는 조인의 대상이 아님을 알 수 있다. 즉, 구조 조인 연산 시에 7번 name 엘리먼트를 읽은 후 15번 name 엘리먼트로 바로 이동할 수 있어야 한다. 이를 위해 기존의 연구에서는 엘리먼트 리스트에 구축된 외부 색인을 사용하지만 색인의 탐색 및 유지 비용으로 인하여 항상 좋은 성능을 보장하지는 못한다.

본 절에서는 엘리먼트의 레벨 정보를 이용하여 조인 결과에 참여하지 못하는 엘리먼트를 효과적으로 제거하는 구조 조인 방법을 제안한다. 엘리먼트의 레벨 정보는 세로 방향의 분포 정보라고 볼 수 있다. 예를 들어, 그림 1에서 manager//name의 구조 조인을 생각해 보자. name 엘리먼트는 manager 엘리먼트의 자식 또는 후손이어야 하므로 반드시 manager 엘리먼트의 레벨이 name 엘리먼트의 레벨보다 작은 값이어야 한다. 따라서 2번, 9번, 12번, 17번의 name 엘리먼트를 조인 연산에서 바로 제외시킬 수 있다. 조인에서 제외되는 엘리먼트를 찾아내기 위해 본 논문에서는 간단한 레벨 정보 테이블만을 사용한다.

3.1 부모-자식 관계의 레벨 조인 기법

본 논문에서는 레벨 조인을 수행하기 위해 조인에 참여하는 두 개의 엘리먼트 리스트들에 대하여, 각 엘리먼트 리스트 내의 엘리먼트들을 엘리먼트의 Level값을 이용하여 정렬한다. 같은 레벨인 엘리먼트는 StartPos 값에 따라 정렬한다. 예를 들어 그림 3의 name 태그에 대한 엘리먼트 리스트는 {2,9,12,17,5,15,7}과 같이 앞서 언급한 방식대로 정렬된다. 이와 같이 정렬된 엘리먼트 리스트 외에 그림 3(b)와 같이 각 태그의 레벨 정보를 저장하는 레벨 정보 테이블을 유지한다. 레벨 정보 테이블의 크기는 XML 데이타 내의 서로 다른 이름의 태그 크기와 동일하며 따라서 대부분의 경우 메모리에서 유지할 수 있는 매우 작은 크기의 테이블임을 알 수 있다.

위와 같이 정렬된 두 개의 엘리먼트 리스트 AList와 DList에 대해 레벨 정보 테이블을 이용하여 다음과 같이 조인을 수행할 수 있다. AList의 레벨 값이 L인 엘리먼트들에 대해 DList의 레벨 값이 L+1인 엘리먼트들과 기존의 구조 조인을 수행한다. 이때, DList의 레벨 값이 L+1인 첫 번째 엘리먼트를 찾기 위해 레벨 정보



tag	level
foodDB	0
drink	1
restaurant	1
name	1 2 3 4
manager	2 3
bar	1 2
address	3

그림 2 레벨 조인 예제

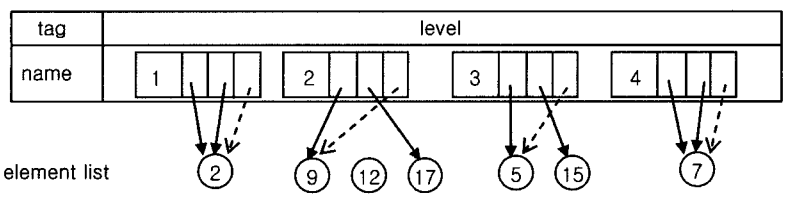


그림 3 레벨 정보 테이블의 name 항목

테이블을 참조한다. 그림 3은 레벨 정보 테이블의 name 태그에 대한 자세한 구조를 보여준다. 레벨 정보 테이블은 각 레벨에 대해 첫 번째 엘리먼트와 마지막 엘리먼트 그리고 현재 처리되고 있는 엘리먼트를 가리키는 세 개의 포인터를 유지한다.

예를 들어 그림 3에서 restaurant/name의 구조 조인을 수행하는 경우, restaurant 엘리먼트들은 레벨 1에만 존재하므로 name 엘리먼트 리스트의 레벨 2인 엘리먼트들만 스캔하여 구조 조인을 수행할 수 있다. 각 엘리먼트 리스트에서 레벨 별로 조인을 수행하기 위해 그림 3과 같은 구조를 가지는 레벨 정보 테이블을 참조한다.

3.2 조상-후손 관계의 레벨 조인 기법

조상-후손 관계의 구조 조인을 수행하기 위해서는 [5]에서 제안한 방법과 같이 부모-자식 관계의 레벨 조인을 약간 변형하여, AList의 레벨 L인 엘리먼트들에 대해 DList의 레벨이 L+1보다 큰 모든 레벨의 엘리먼트들과 조인을 수행할 수 있다. 하지만 이 경우 DList의 엘리먼트들이 레벨 별로 반복해서 스캔되는 문제가 생기게 된다.

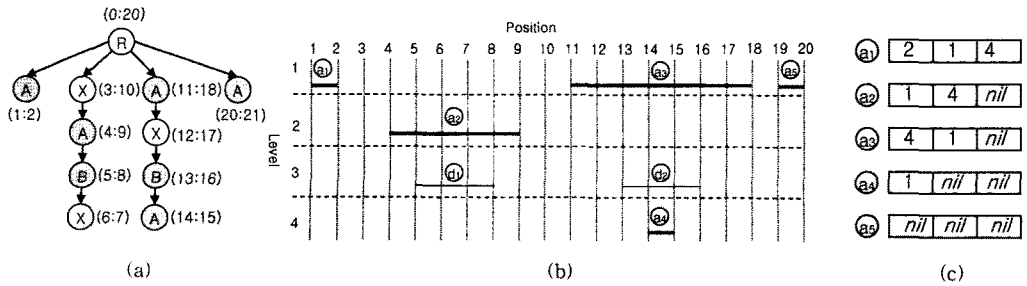
예를 들어, 그림 4(a)에서 A//D인 레벨 조인을 수행하기 위해서는, 그림 4(b)와 같이 AList의 엘리먼트들은 레벨 1, 2, 4로 {[a_1,a_3,a_5], [a_2], [a_4]}와 같이 정렬해야 한다. 앞서 설명한 바와 같이, 레벨 조인 알고리즘은 AList의 레벨 1의 엘리먼트들과 DList의 레벨 3의 엘리먼트들을 조인하고 ([a_1,a_3,a_5]⋈[d_1,d_2]), AList의 레벨 2의 엘리먼트와 DList의 레벨 3의 엘리먼트를 조인하게 ([a_2]⋈[d_1,d_2]) 된다. 즉, DList를 2회 스캔하게 된다. 최악의 경우, AList의 엘리먼트들이 {1, ···, n} 레벨에 흩어져 있고 DList의 엘리먼트들이 {n+1, ···, m} (m > n) 레벨에 흩어져 있다고 할 때 DList는 n회 스캔되며, 따라서 기존의 알고리즘 보다 훨씬 좋지 못한 성능을 보이게 된다.

본 논문에서는 조상-후손 관계의 레벨 조인 시 DList가 반복적으로 스캔되는 문제점을 막기 위하여 레벨 별로 정렬되어 있는 엘리먼트 리스트의 엘리먼트들을 StartPos 순서로 연결할 수 있는 연결 배열을 제안한다. 연결 배열은 엘리먼트 리스트에서 불필요한 레벨의 엘리먼트들을 제거하더라도 엘리먼트들은 여전히 StartPos의 순서를 유지할 수 있도록 만들어져야 한다. 연결 배열을 이용하여 엘리먼트들의 StartPos순서를 유지하는 경우 조인 결과에 참여하지 못하는 불필요한 레벨을 제거한 후 기존의 구조 조인 알고리즘을 수행함으로써 DList가 반복적으로 스캔되는 문제를 피할 수 있다.

연결 배열은 엘리먼트 리스트의 각 엘리먼트마다 한 개씩 만들어 지며, 엘리먼트들이 n개의 레벨 {L_1, L_2, ···, L_n}에 분포되어 있는 엘리먼트 리스트 내의 엘리먼트 e의 연결 배열은 {L_1, L_2, ···, L_n}∪{*nil*, *nil*, ···, *nil*}의 부분집합의 조합을 통해 다음과 같이 만들어진다. 엘리먼트 리스트 내의 각 레벨 L에 대해 e.StartPos 보다 StartPos의 값이 커지는 첫 번째 엘리먼트를 e_L이라고 하자. 이때, 레벨 L에 해당 엘리먼트가 없는 경우 e_L의 값은 nil이 된다. 엘리먼트 e의 연결 배열에는 n개의 레벨 값을 e_L의 StartPos 순으로 정렬한 값이 들어가게 된다. 단, 정렬 시 nil값은 가장 큰 값으로 하여 정렬한다.

그림 4(c)는 AList의 각 엘리먼트에 대한 연결 배열을 보여준다. AList의 엘리먼트들은 레벨 1,2,4에 흩어져 있고, 엘리먼트 a_1의 경우 앞의 설명에 따라 e_1은 a_3, e_2는 a_2, e_4는 a_4가 되며, 이들을 StartPos로 정렬하면 e_2 (a_2), e_1 (a_3), e_3 (a_4)이므로 연결배열에는 2, 1, 3의 값이 들어가게 된다. 동일한 방식으로 모든 엘리먼트의 연결 배열을 만들 수 있다. 하나의 연결 배열을 위해 필요한 공간은 (n+1)*log2(n+1) 비트이며, 한 엘리먼트 리스트의 엘리먼트들이 보통 3개의 레벨에 분포되어 있다고 하면 연결 배열을 위한 공간은 1바이트 정도이다.

본 논문에서는 구조 조인 연산 중에 연결 배열을 사용하여 다음 번 필요한 엘리먼트를 찾기 위해 알고리즘 1과 같은 NextElement 알고리즘을 제안한다. 다음 번 필요한 엘리먼트란 조인에 참여하지 못하는 레벨의 엘리먼트를 제외한 나머지 엘리먼트들 중 현재 엘리먼트 보다 StartPos값이 큰 첫 번째 엘리먼트를 지칭한다. 알



(a) (b) (c)

그림 4 조상-후손 관계의 레벨 조인 예제

고리즘 1의 InvalidLevel 집합에 포함되는 레벨 값은 다음과 같이 계산된다. DList의 경우 AList의 가장 작은 값보다 작은 값을 가지는 레벨 값들을 InvalidLevel집합에 포함시키며, AList의 경우 DList의 가장 큰 값보다 큰 값을 가지는 레벨 값들을 InvalidLevel 집합에 포함시킨다. 예를 들어, 그림 4에서 DList의 가장 큰 레벨 값이 3이므로 AList에 대한 InvalidLevel은 {4}가 된다. 알고리즘 1은 06, 07번째 줄에서 InvalidLevel에 포함되지 않는 첫 번째 레벨 찾아주며, 10번째 줄에서 해당 레벨의 현재 엘리먼트를 돌려주고 해당 레벨의 현재 엘리먼트 포인터를 다음 엘리먼트를 가리키도록 변경한다.

알고리즘 1 Next Element 알고리즘

```
01: /* LIT는 현재 엘리먼트 리스트에 대한 레벨 정보 테이블 항목임*/
02: /* InvalidLevel은 현재 엘리먼트 리스트에 대해 조인에 참여하지 못하는 레벨임*/
03: Element nextElement(Element e)
04: {
05:     conn array = e.connection array;
06:     for(int i = 0; i < conn array.size(); i++)
07:         if(conn array[i] ∈ InvalidLevel) continue;
08:     if(i == conn array.size() or conn array[i] == nil) return nil;
09:     level entry = LIT.lookup(conn array[i]);
10:     return level entry.current element++;
11: }
```

NextElement을 이용함으로써 트리-합병[1], 스택-트리 조인[2]과 같은 기본 구조 조인 알고리즘뿐만 아니라, 가로 방향의 분포를 이용하는 B+트리 조인[3]등의 기존의 알고리즘을 적용할 수 있으며, 각 레벨 별로 하나의 디스크 블록을 저장할 수 있는 메모리를 할당함으로써 AList나 DList를 다시 스캔하지 않고 조인 연산을 수행할 수 있다.

4. 실험결과

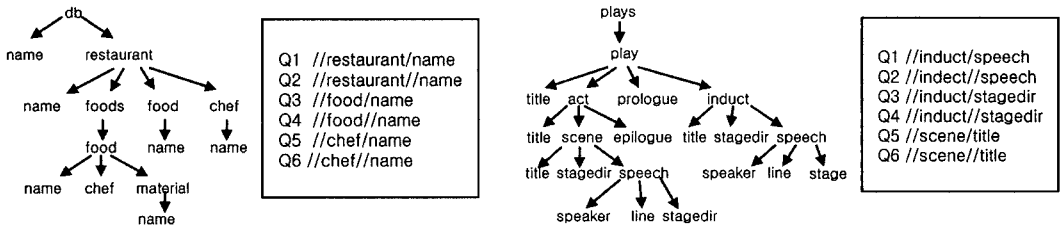
본 절에서는 본 논문에서 제한하는 기법의 성능을 실험을 통해 보여준다. 실험을 위해 4Kbyte메모리 페이지 20개를 가지는 실험용 XML 데이타베이스 시스템을 사용하였다. [8]과 동일하게 그림 5(a)의 DTD를 XML 데이타 생성기[9]를 이용하여 생성한 20Mbyte의 데이타와 그림 5(b)의 DTD를 가지는 7.5MByte의 Shakespeare 데이타를 사용하였고, 조상-후손 관계를 실험하기 위하여 질의집합을 약간 변경하여 실험하였다.

실험에서는 본 논문에서 제안한 방법(LevelJoin2)과 [8]에서 제안한 방법(LevelJoin1) 그리고 스택-트리 조인(StackTree)[5] 방법을 비교하였다. 그림 6은 실험의 결과를 보여준다. 그림 6(a)의 Q2, Q4, Q6와 그림 6(b)의 Q2, Q4에서 볼 수 있듯이, [8]에서 제안한 방법의 경우 조상-후손 관계의 구조 조인 시에 DList를 반복하여 스캔하기 때문에 많은 경우 스택-트리 조인 기법보다 좋지 못한 성능을 보이는 것을 알 수 있다. 하지만 본 논문에서 제안한 방법은 해당 질의 집합에 대해 불필요한 엘리먼트들을 제외하고 엘리먼트 리스트를 한번만 스캔하므로 항상 스택-트리 조인 기법보다 우수한 성능을 보인다는 것을 실험을 통하여 알 수 있다.

5. 결 론

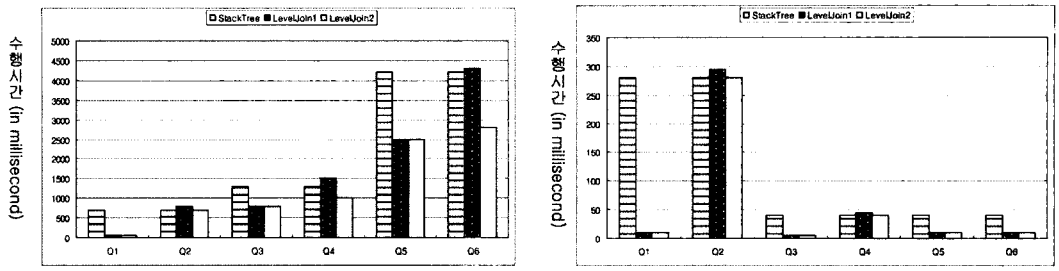
구조 조인은 XML 질의 처리의 기본 연산이다. 따라서 XML질의 처리의 성능 향상을 위해서는 구조 조인의 성능향상이 필수적이라 할 수 있다. 하지만, 구조 조인에서 사용하는 엘리먼트 리스트는 특정 태그 이름을 만족하는 모든 엘리먼트를 포함하고 있기 때문에 조인 결과에 영향을 주지 못하는 엘리먼트들이 많이 존재할 수 있다. 따라서 구조 조인의 성능 향상을 위해서는 엘리먼트 리스트 내의 조인 결과에 참여하지 못하는 엘리먼트들을 효과적으로 걸러내는 것이 필요하다. 현재 제안된 방법들은 불필요한 엘리먼트를 걸러내기 위해 복잡한 외부 색인을 사용하고 있으나 색인의 사용으로 인해 조인 기법이 매우 복잡해지고 색인의 부담으로 인하여 성능이 저하되는 경우가 발생할 수 있는 단점을 가지고 있다.

본 논문에서는 레벨 정보를 이용하여 구조 조인 시에 불필요한 엘리먼트를 효과적으로 제거하는 기법을 제안하였다. 특히, 레벨로 정렬된 정보와 StartPos로 정렬된



(a) 가공 데이타 DTD와 질의 집합 (b) Shakespeare 데이타 DTD와 질의 집합

그림 5 실험 데이타 및 질의 집합



(a) 가공 데이타 실험 결과 (b) Shakespeare 데이타 실험 결과

그림 6 실험 결과

정보를 동시에 유지하기 위한 연결배열을 제안함으로써 조상-후손 관계에 대해서도 엘리먼트 리스트를 한번만 스캔하여 조인을 수행할 수 있음을 보였다. 하지만, 조상-후손 관계의 구조 조인 시 조인에 참여하는 엘리먼트들 중에 가장 작은 레벨 값을 가지는 엘리먼트가 AList에 존재하는 경우 불필요한 엘리먼트를 제거할 수 없다는 단점을 가지고 있다. 이러한 단점을 극복하기 위하여 세로 방향의 분포 정보인 레벨 정보에 가로 방향의 분포 정보를 추가로 이용하는 레벨 조인 기법에 대해 향후에 연구할 예정이다.

참 고 문 헌

[1] T. Bray, J. Paoli, and C.M. Sperberg-McQueen, "Extensible markup language (XML) 1.0," Technical Report, W3C Recommendation, 1998.

[2] D. Chamberlin, D. Florescu, J. Robie, J. Simeon, and M. Stefanescu, "XQuery: A Query Language for XML," Technical report, W3C Working Draft, February 2001.

[3] Chun Zhang, Jeffrey Naughton, David DeWitt, Qiong Luo, and Guy Lohman. On supporting containment queries in relational database management systems. In *Proceedings of the ACM SIGMOD International Conference on the Management of Data*, 2001.

[4] Q. Li and B. Moon, "Indexing and querying XML data for regular path expressions," In Proceedings of the Conference on Very Large Data Bases, 2001.

[5] Shurug Al-Khalifa, H. V. Jagadish, Nick Koudas, Jignesh M. Patel, Divesh Srivastava, and Yuqing Wu. Structural Joins: A Primitive for Efficient XML Query Pattern Matching. In *IEEE International Conference on Data Engineering*, 2002.

[6] Shu-Yao Chien, Zografoula Vagena, Donghui Zhang, Vassilis J. Tsotras, and Carlo Zaniolo. Efficient Structural Joins on Indexed XML Documents. In *Proceedings of the Conference on Very Large Data Bases*, 2002.

[7] Haifeng Jiang, Hongjun Lu, Wei Wang, and Beng Chin Ooi. XR-Tree: Indexing XML Data for Efficient Structural Joins. In *IEEE International Conference on Data Engineering*, 2003.

[8] 이윤호, 최일환, 김종익, 김형주, "색인된 XML 문서에서 레벨 정보를 이용한 효과적인 구조 조인 기법", 정보과학회논문지:데이타베이스, 32권 6호, pp. 641-649, 2005년 12월.

[9] IBM XML Generator. http://www.alphaworks.ibm.com/tech/xmlgenerator, 2004.

김 종 익

1998년 한국과학기술원 전산학과 학사
2000년 한국과학기술원 전산학과 석사
2004년 서울대학교 컴퓨터공학과 박사
2004년 8월~2007년 3월 한국전자통신연구원 선임연구원. 2007년 4월~현재 전북대학교 전자정보공학부 전임강사