DOI: 10.3745/KIPSTD.2010.17D.2.103

XML 문서 검색을 위한 경로 역 색인 기법

문 경 원[†] · 황 병 연[††]

요 약

최근에 관계형 데이터베이스 관리 시스템(RDBMS)의 장점을 이용하여 XML로 표현된 문서를 효과적으로 저장, 관리, 검색하는 XML 문서 관리 시스템에 대한 많은 연구들이 활발하게 진행되어 왔다. 그러나 경로 질의 중에서 *LIKE* 연산과 같은 부분 패턴 매칭 질의의 경우는 비효율적인 비교 연산으로 인해 검색 성능이 현저하게 떨어지기 때문에 RDBMS의 색인의 효과를 볼 수 없다.

본 논문에서는 XML 문서를 RDBMS에 효율적으로 저장하는 계층형 XML 저장 기법과 경로 역 색인 기법을 제안한다. 제안하는 기법은 XML문서의 엘리먼트를 키워드로 보고 해당 키워드가 속한 경로의 식별자와 시퀀스로 포스팅 파일을 구성하여 경로 기반 질의의 검색 속도를 향상하는데 주안점을 둔다. 검색 성능에 관한 실험을 통해서 제안된 기법이 기존의 RDBMS를 이용한 기법보다 약 60% 우수한 성능을 가지고 있음을 입증한다.

키워드 : XML, 관계형 데이터베이스관리시스템, 문서검색, 역 색인기법, 경로질의

The Path Inverted Index Technique for XML Document Retrieval

Kyung-Won Moon[†] · Byung-Yeon Hwang[††]

ABSTRACT

Recently, many XML document management systems using the advantage of RDBMS have been actively developed for the storage, processing and retrieval of XML documents. However, fractional pattern-matching query such as the *LIKE* operations cannot take the advantage of the index of RDBMS because these operations have deteriorated retrieval performance through its inefficient comparison processing.

The hierarchical XML storage technique which stores XML documents in RDBMS efficiently, and the path inverted index technique are proposed in this paper. It regards the element of an XML document as a keyword, and focuses on organizing a posting file with path identifiers and sequences to reduce the retrieval time of path based query. Through simulations, our methods have shown about 60% better performance than the conventional method using RDBMS in searching.

Keywords : XML, RDBMS, Document Retrieval, Inverted Index Method, Path Query

1. 서 론

1996년 W3C에 의해 XML 표준이 제정된 이래로, XML을 사용하는 문서가 급증하였다. XML의 사용이 증가함에 따라서 XML과 관련된 다양한 분야에 대한 연구의 필요성이 증대되고 있는데, 특히 XML로 표현된 문서를 효과적으로 저장, 관리, 검색하기 위한 XML 문서 관리 시스템에 대한 많은 연구들이 활발하게 진행되고 왔다. 최근에는 주로 관계형 데이터베이스 관리시스템(RDBMS)의 장점을 이용한 XML 문서의 저장과 검색에 대한 연구가 이루어지고 있다. RDBMS의 XML 파서를 이용하여 문서 내용을 관계형 테이블에 매핑(mapping)하면 안정적이고 효율적인 XML 문서 관리 시스템을 구축할 수 있다.

본 논문에서 제안하는 검색 기법은 RDBMS를 기반으로 한다. 그러나 제안된 기법은 기존의 전통적인 RDBMS 기반의 기법에서 발생하는 문제점인 저장된 경로 전체에 대한 스트링 매칭을 통해 결과를 얻어내는 것이 아니고, 역 색인 모듈을 사용하여 경로 비교 연산을 현저히 줄인다. 또한, 본 논문에서는 질의 수행 속도를 개선하고 불필요한 테이블의 저장 모델로 인한 조인 오버헤드를 줄이기 위해 계층적 저장 구조를 제안한다. 제안된 기법은 RDBMS를 기반으로 하는 대표적인 문서 관리 시스템인 XRel 시스템과 검색수행 시간에 대한 성능비교를 통해서 제안된 기법의 성능이 우수하다는 것을 보인다.

본 논문의 구성은 다음과 같다. 2장에서는 기존의 XML

※ 본 연구는 2009년도 가톨릭대학교 교비연구비의 지원으로 이루어졌음.
† 정 회 원 : (주)NHN 검색개발센터 과장
†† 종신회원 : 가톨릭대학교 컴퓨터정보공학부 교수(교신저자)
논문접수 : 2008년 10월 27일
수 정 일 : 1차 2010년 3월 22일, 2차 2010년 4월 12일
심사완료 : 2010년 4월 12일

문서 검색 기법들을 모델별로 분류하고 이러한 기법들이 대용량 XML 문서 검색에 있어 발생되는 문제점에 대해 기술한다. 3장에서는 본 논문에서 제안하는 XML의 계층적 저장 구조와 질의 처리 방법을 살펴보고 4장에서는 경로 역 색인 구조 및 검색 알고리즘을 제안한다. 5장에서는 기존 연구인 XRel 시스템과의 성능비교를 통해 제안된 기법이 검색 수행 시간의 우수성을 입증한다. 끝으로, 6장에서는 결론 및 향후 연구 과제를 기술한다.

2. 관련 연구

XML 저장 및 검색 연구를 인덱싱 모델별로 분류를 해보면 그래프 기반 방법[1-5], 비트맵 기반 방법[6-8], RDBMS 기반 방법[9-14]으로 나누어 볼 수 있다.

스텐포드 대학에서 제안한 Lore[1]는 그래프 기반 방법으로서 구조가 고정되지 않고 변동될 수 있는 반구조적 데이터를 관리하기 위해 개발된 시스템이었으나 XML이 등장하면서 XML 관리 시스템으로 발전되었다. 데이터 색인은 그래프 기반 OEM(Object Exchange Model) 데이터 모델을 사용하고, XML 문서 스키마는 Dataguides[2]를 통해 유지한다. Dataguides는 XML 문서의 중복 경로를 제거하여 요약된 형태의 문서 구조를 제공하는 기능을 한다. 그래프 기반의 다른 방법으로 APEX[3]는 XML 데이터의 구조정보를 표현하고, 검색에 자주 사용되는 경로를 해쉬 트리에 저장해 놓고 검색 속도를 향상시키는 방법이다. 하지만, 그래프 기반 방법은 새로운 경로가 등장할 때마다 그래프의 수정이 일어나게 되고, 이와 같은 상황은 인터넷 검색 환경에서 빈번하게 발생한다.

비트맵 기반 연구는 BitCube[6-7]와 X-Planeb[8]가 대표적인 연구이다. BitCube는 XML 검색에서 뛰어난 성능을 입증한 3차원 비트맵 인덱스 기법이다. 하지만 BitCube는 문서의 증가에 따라 급격하게 인덱스의 크기가 증가하게 되고, 이로 인해 연산 수행 속도가 저하되는 문제점을 보인다. X-Planeb는 이러한 BitCube의 단점을 보완하기 위해 기존의 3차원 배열구조를 연결 리스트로 변형하여 저장 공간을 축소하였고, 경로 축약 기법을 추가하여 검색 성능을 향상시켰다. 그러나 X-Planeb는 사이즈가 큰 노드를 사용하여 초기에 적은 양의 문서가 적재되었을 때 BitCube보다 메모리 사용량이 더 크다는 단점이 있다.

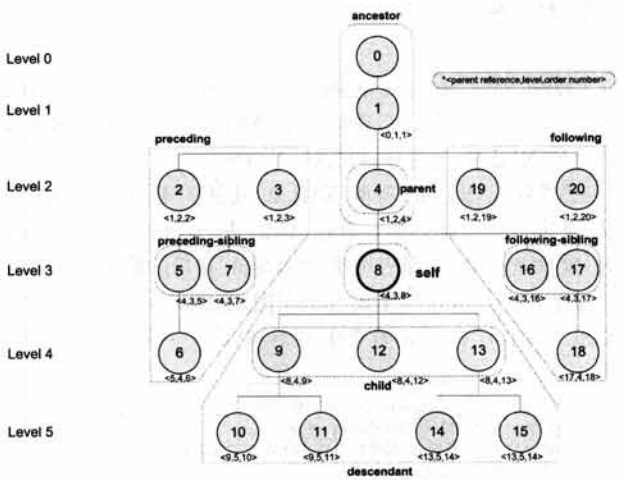
RDBMS 기반 방법은 XML문서를 테이블 스키마로 적절히 표현하고 테이블에 효율적으로 저장하여 SQL질의로 효율적인 검색을 지원하는 시스템에 관한 연구이다. 이러한 방법은 RDBMS라는 검증된 시스템을 기반으로 빠른 검색 성능과 안정적인 시스템을 구축할 수 있는 장점이 있다. XRel[10]은 RDBMS 기반 방법의 대표적 연구이며, 본 연구의 성능 비교 대상이 되는 연구이다. XRel의 저장 구조는 Element, Attribute, Text, Path로 구성되는 네 개의 테이블에 XML 문서를 저장한다. XRel은 경로 질의를 SQL 질의로 변환하여 처리하는데, 경로의 매칭과정에서 *LIKE* 연산과 같은 부분 패턴 매칭 질의를 사용한다. 부분 패턴 매칭 질의는 정확한 검색 조건을 주는 것이 아니라 부분적인 검색 조건을 주고 경로 전체에 대한 비교를 통해서 결과를 추출하기 때문에 문서의 수가 증가할수록 그 검색 성능은 현저히 떨어지게 된다. 검색의 결과도 4개 테이블의 동등 조인(equi-join)을 통해 얻기 때문에 문서의 증가에 따른 조인 오버헤드가 발생한다. 이러한 문제점은 역 색인 모듈을 사용해서 해결할 수 있다. 역 색인 모듈은 인덱스(index) 파일, 포스팅(posting) 파일, 데이터(data) 파일로 구성된다. 인덱스 파일은 사전(dictionary) 또는 컬렉션(collection)이라고도 하고, 데이터 파일은 문서(document) 파일이라고도 한다.

3. XML 문서 저장구조

3.1 계층적 저장구조

본 논문에서는 XML 데이터를 저장하기 위해 계층적 저장 구조를 사용한다. 계층적 저장 구조는 XML의 계층적 구조를 반영하여 RDBMS에 저장된다. XML 문서는 순서가 있는 트리의 형태로 표현되며, 엘리먼트(노드)의 위치에 의해 부모-자식, 조상-후손 관계가 있다. 그러나 이들 관계 외에도 노드들 사이에는 다양한 관계가 존재하는데, 그림 1은 XML 문서에서 노드 사이의 관계 범위를 보여준다.

본 연구에서는 XML 문서의 엘리먼트 노드간 관계를 구분하는데 사용되는 3가지 정보를 선정하였다. 각 엘리먼트 노드들은 부모 참조(parent reference) 값과 트리 레벨(level), 순서번호(order number)등의 정보를 가질 수 있다. 먼저, (그림 1)에서 보는 것과 같이 부모 참조 값은 부모-자식 관계를 표현하는 정보이다. 부모에 대한 참조 값으로 부모 노드의 고유 식별자를 사용한다. 또한 부모 참조 값은 형제(sibling) 관계를 찾는 방법을 제공한다. 두 번째로 트리 레벨은 XML 문서 트리내의 레벨정보를 나타낸다. 끝으로 순서번호는 XML 문서 트리의 전위순회 순서정보이며, 형제 관계 중에서 선행 형제(preceding sibling)와 후행 형제(following sibling) 엘리먼트를 찾는 방법을 제공한다.

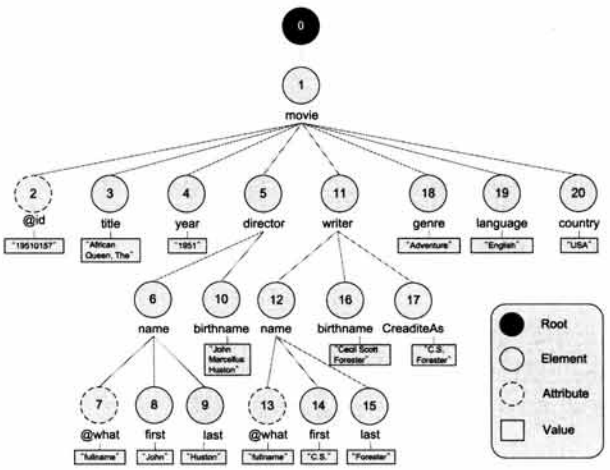


(그림 1) XML 문서의 관계 표현 범위

[정의 1] XML 문서내의 임의의 엘리먼트 노드 P_i와 P_j가 존재할 때 P_i와 P_j 사이에는 다음과 같은 관계를 가질 수 있다. 여기서 *ord*는 노드의 순서 번호(order number)를 의미하고 *ref*는 부모 참조(parent reference)를 의미한다. 예를 들면, (그림 1)에서 *ref*를 1로 가진 노드(노드 번호가 2, 3, 4, 19, 20)는 *ord*가 1인 노드(노드 번호가 1)의 자식노드이다.

1) $P_i.ord = P_j.ref$이면 P_i가 P_j의 부모이고, P_j는 P_i의 자식이다.
2) $P_i.ref = P_j.ref$이면 노드 P_i와 P_j는 형제(sibling) 관계이다.
3) P_i와 P_j는 형제이고, 노드 $P_i.ord < P_j.ord$이면 노드 P_i가 P_j의 선행 형제(preceding sibling)이고, P_j는 P_i의 후행 형제(following sibling)이다.

그 외에도 조상-후손 관계의 노드들은 부모-자식에 대한 재귀적인 탐색을 통해 얻을 수 있고, 선행 노드관계는 자신의 노드의 선행 형제와 그 후손노드, 조상 노드들의 선행 형제와 그 후손들의 합집합으로 표현할 수 있다. 후행 노드 관계 역시 선행 노드와 같은 방식으로 찾을 수 있다. (그림 2)는 트리 기반 모델로 표현된 영화 정보를 나타내는 XML 문서의 예를 나타내고 (그림 3)은 이 XML 문서를 (그림 1)을 기반으로 XMLDocuments 테이블에 저장을 한 모습을 나타낸다.



(그림 2) XML 문서 모델

XMLDocuments

did	pid	name	ref	level	order	value
1	1	movie	R	1	1	null
1	2	@id	1	2	2	19510157
1	3	title	1	2	3	African Queen, The
1	4	year	1	2	4	1951
1	5	director	1	2	5	null
1	6	name	5	3	6	null
1	7	@what	6	4	7	fullname
1	8	first	6	4	8	John
1	9	last	6	4	9	Huston
1	10	birthname	5	3	10	John Marcellus Huston
1	11	writer	1	2	11	null
1	12	name	11	3	12	null
1	13	@what	12	4	13	fullname
1	14	first	12	4	14	C.S.
1	15	last	12	4	15	Forester
1	16	birthname	11	3	16	Cecil Scott Forester
1	17	CreaditeAs	11	3	17	C.S. Forester
1	18	genre	1	2	18	Adventure
1	19	Language	1	2	19	English
1	20	country	1	2	20	USA

(그림 3) XMLDocuments 테이블

3.2 질의 처리

본 논문에서 제안된 기법은 Oracle RDBMS 기반으로 구현되었고, 사용되는 모든 SQL 질의는 Oracle PL/SQL 블록으로 사전에 미리 작성된 프로시저나 함수를 통해서 처리된다. XMLDocuments 테이블과 관련한 질의는 엘리먼트 간의 관계 질의가 주류를 이룬다. 본 절에서는 계층적 저장구조의 엘리먼트 간 관계를 찾아주는 기본 질의인 부모-자식, 선행-후행 형제 질의의 프로시저와 함수의 구성에 대해 살펴보고자 한다.

부모-자식관계의 엘리먼트를 찾는 것은 조상-후손 관계를 찾기 위한 기반이 되는 처리과정이다. (그림 4)는 부모-자식 질의 처리를 수행하는 PL/SQL 블록이다. 엘리먼트 노드와 노드 리스트를 표현하기 위해 사용자 정의형 타입 element_node 객체와 element_node_array 객체를 정의하였다(1~10줄). element_node는 XMLDocuments에서 엘리먼트 노드를 표현하기 위한 필드 값들을 멤버 필드로 포함한다. element_node_array는 element_node 타입의 배열로 다수의 엘리먼트를 반환할 필요가 있을 때 사용되며, 가변 배열이다. parent 함수와 child 함수는 pid와 ref값을 참조하여 해당 노드를 반환하는 기능을 한다(19, 28~30줄).

(그림 5)는 선행 형제와 후행 형제를 찾는 PL/SQL 함수들을 보여주고 있는데, 형제는 우선 ref값이 같고 그 중에서 순서(order)가 작은 그룹은 선행 형제들이고 큰 그룹은 나머지 후행 형제들이 된다(6, 21줄).

```
1   TYPE element_node AS OBJECT(
2           did     NUMBER,
3           pid     NUMBER,
4           name    VARCHAR2,
5           ref     NUMBER,
6           level   NUMBER,
7           order   NUMBER,
8           value   VARCHAR2
9   )
10  TYPE element_node_array AS VARRAY(25) of element_node;
11  FUNCTION parent (input_pid VARCHAR2) RETURN element_node IS
12          ref     XMLDocuments.ref%TYPE;      did     XMLDocuments.did%TYPE;
13          pid     XMLDocuments.pid%TYPE;      name    XMLDocuments.name%TYPE;
14          ref     XMLDocuments.ref%TYPE;      level   XMLDocuments.level%TYPE;
15          order   XMLDocuments.order%TYPE;    value   XMLDocuments.value%TYPE;
16  BEGIN
17          SELECT ref INTO ref FROM XMLDocuments WHERE pid=input_pid;
18          SELECT did,pid,name,ref,level,order,value INTO did,pid,name,ref,level,order,value
19          FROM XMLDocuments WHERE pid=ref;
20          RETURN element_node(did,pid,name,ref,level,order,value);
21  END;
22  FUNCTION child(intpu_pid VARCHAR2) RETURN element_node_array IS
23          ref     XMLDocuments.ref%TYPE;      did     XMLDocuments.did%TYPE;
24          pid     XMLDocuments.pid%TYPE;      name    XMLDocuments.name%TYPE;
25          ref     XMLDocuments.ref%TYPE;      level   XMLDocuments.level%TYPE;
26          order   XMLDocuments.order%TYPE;    value   XMLDocuments.value%TYPE;
27          result_elements     element_node_array;     i       INTEGER := 0;
28          CURSOR result IS
29                  SELECT did,pid,name,ref,level,order,value
30                  FROM XMLDocuments WHERE ref=input_pid;
31  BEGIN
32          OPEN result;
33          LOOP
34                  FETCH result INTO r;
35                  result_elements(i) := element_node(r.did,r.pid,r.name,r.ref,r.level,r.order,r.value
36                  i := i + 1;
37                  EXIT WHEN result%NOTFOUND;
38          END LOOP;
39          RETURN result_elements;
40  END;
```

(그림 4) 부모-자식 질의 처리 PL/SQL 블록

```
1   FUNCTION preceding_sibling (input_pid VARCHAR2) RETURN element_node_array IS
2       ...
3   BEGIN
4       SELECT ref,order INTO t_ref,t_order FROM XMLDocuments WHERE pid=input_pid;
5       result IS SELECT did,pid,name,ref,level,order,value
6           FROM XMLDocuments WHERE ref=t_ref AND order<t_order;
7       OPEN result;
8       LOOP
9           FETCH result into r;
10          result_elements(i) := element_node(r.did,r.pid,r.name,r.ref,r.level,r.order,r.value
11          i := i + 1;
12          EXIT WHEN result%NOTFOUND;
13      END LOOP;
14      RETURN result_elements;
15  END;
16  FUNCTION following_sibling(intpu_pid VARCHAR2) RETURN element_node_array IS
17      ...
18  BEGIN
19      SELECT ref,order INTO t_ref,t_order FROM XMLDocuments WHERE pid=input_pid;
20      result IS SELECT did,pid,name,ref,level,order,value
21          FROM XMLDocuments WHERE ref=t_ref AND order>t_order;
22      OPEN result;
23      LOOP
24          FETCH result into r;
25          result_elements(i) := element_node(r.did,r.pid,r.name,r.ref,r.level,r.order,r.value
26          i := i + 1;
27          EXIT WHEN result%NOTFOUND;
28      END LOOP;
29      RETURN result_elements;
30  END;
```

(그림 5) 선행-후행 형제 질의 처리 PL/SQL 블록

4. 경로 역 색인

4.1 역 색인 구조

전통적인 역 색인과 마찬가지로 본 논문에서 사용하는 역 색인은 (그림 6)의 포스팅 테이블에서 보이는 것과 같이 키워드와 포스팅 리스트의 쌍으로 구성된다.

[정의 2] 포스팅 테이블은 *keyword*, <*pid, sequences*> 쌍으로 구성되며, 포스팅 리스트상의 *pid*는 엘리먼트 노드의 경로 식별자이고, *sequences*는 경로가 시작하는 위치부터 해당 키워드가 나타나는 위치까지의 거리들의 집합이다. 임의의 엘리먼트 노드 n_1, n_2 사이에는 다음 관계가 성립한다.

1) $n_1.pid = n_2.pid$이고, $n_1.sequences + 1 = n_2.sequences$이면 n_1은 n_2의 부모이고, n_2는 n_1의 자식이다.
2) $n_1.pid = n_2.pid$이고, $n_1.sequences < n_2.sequences$이면 n_1은 n_2의 조상이고, n_2는 n_1의 후손이다.

포스팅 리스트는 <pid, sequences>의 쌍으로 구성되는데, pid는 해당 키워드가 존재하는 경로의 경로 식별자이고, sequences는 경로가 시작하는 위치부터 해당 키워드가 나타나는 위치까지의 거리들의 집합이다. 포스팅 리스트는 비교연산의 효율성을 위해 B-tree 색인으로 구성되며, 바이트 형태의 객체로 RDBMS에 저장된다.

먼저, 경로 식별자 pid는 키워드들이 같은 경로 상에 나타나는지를 찾기 위한 참조 값이다. (그림 6)에서 director 키워드는 5~10번 경로에 나타나고, name 키워드는 6~9번, 12~15번 경로에 나타나는 것을 볼 수 있다. 이를 통해서 director와 name 키워드가 경로 6~9번에 동시에 나타나고 있다는 것을 알 수 있다. sequences는 경로 상에서 해당 키워드가 어느 위치에 나타나는지 알려주는 정보로써, 6~9번 경로에서 director는 경로의 2번째 위치에 존재하고 name은 3번째 위치에 나타난다는 것을 알 수 있다. 이것은 director가 name의 부모 엘리먼트 노드라는 것을 말해준다. 그리고 director와 first 키워드의 경우에는 8번 경로 상에서 동시에 나타나고, 경로 등장순서는 director는 2번째, first는 4번째 위치에 존재한다는 것을 알 수 있다. 이 경우 sequences 정보는 director가 first의 조상 엘리먼트 노드임을 알려준다.

4.2 경로 질의 처리

제안한 기법에서는 경로 전체에 대한 비효율적인 비교연산을 하지 않고, 역 색인구조를 사용하여 기존의 RDBMS 기반 시스템의 경로 비교연산의 비효율성을 개선하고자 한다. (그림 7)은 역 색인을 이용한 경로 질의 처리 알고리즘을 기술한 것이다.

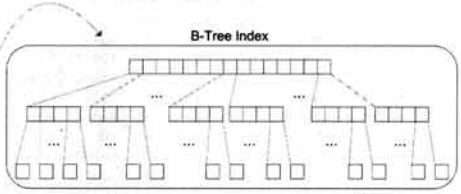
[예제 1] *xpe* =/movie//name/last를 찾는 예를 살펴보자. *xpe*는 XPath 경로 표현식(XPath path expression)을 의미한다. (그림 8)은 검색 과정을 보이고 있다.

path

pid	path
1	movie
2	movie.@id
3	movie.title
4	movie.year
5	movie.director
6	movie.director.name
7	movie.director.name.@what
8	movie.director.name.first
9	movie.director.name.last
10	movie.director.birthname
11	movie.writer
12	movie.writer.name
13	movie.writer.name.what
14	movie.writer.name.first
15	movie.writer.name.last
16	movie.writer.birthname
17	movie.writer.CreditedAs
18	movie.genre
19	movie.language
20	movie.country

posting

keyword	posting list
movie	<1,(1)>, <2,(1)>, <3,(1)>, <4,(1)>,⋯ , <18,(1)>, <19,(1)>, <20,(1)>
@id	<2,(2)>
title	<3,(2)>
year	<4,(2)>
director	<5,(2)>, <6,(2)>, <7,(2)>, <8,(2)>, <9,(2)>, <10,(2)>
name	<6,(3)>, <7,(3)>, <8,(3)>, <9,(3)>, <12,(3)>, <13,(3)>, <14,(3)>, <15,(3)>
@what	<7,(4)>
first	<8,(4)>, <14,(4)>
last	<9,(4)>, <15,(4)>
birthname	<10,(3)>, <16,(3)>
writer	<12,(2)>, <13,(2)>, <14,(2)>, <15,(2)>, <16,(2)>, <17,(2)>
CreaditedAs	<17,(3)>
genre	<18,(2)>
language	<19,(2)>
country	<20,(2)>

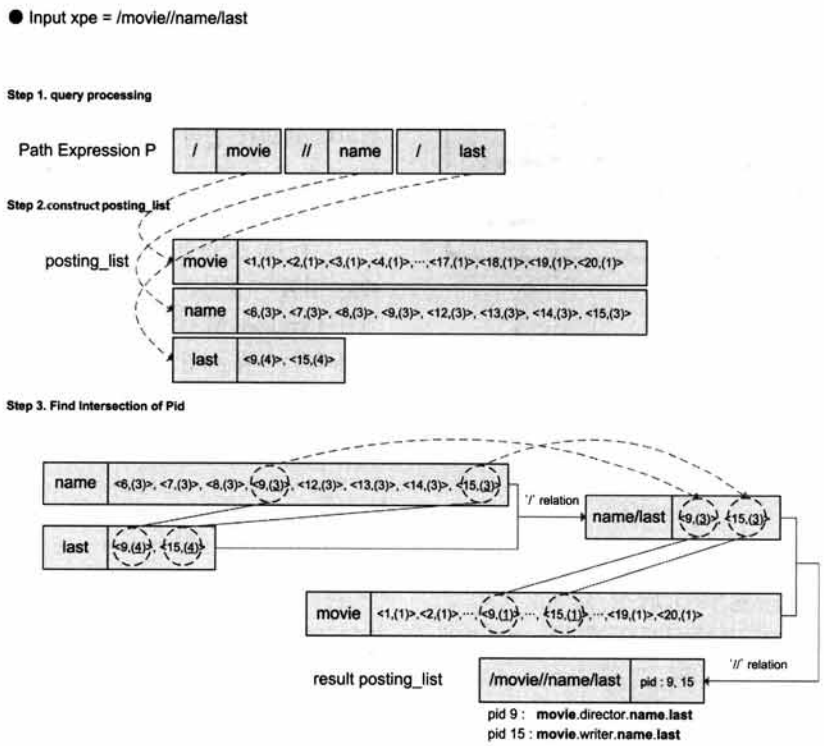


(그림 6) 시스템 역 색인 구조

```
1       Search_Pids(P,L)
2       Input : Path Expression P, Posting L
3       Output : Matched Set of Posting List
4       Algorithms :
5       begin
6
7       (1)construct posting_list : Set keyword and posting_list pair that matched P's #n keyword and L's keyword
8       initialize posting_list[n];
9       (2) find Intersection of Pid
10      for(i=n;i<1;i++){
11              posting_list[i-1] = Intersection_of_Pid(posting[i-1].posting_list,posting[i],posting[i].separator);
12      }
13      return posting_list[0];
14
15      end
```

```
1       Intersection_of_Pid(L1,L2,S)
2       Input : Posting List L1, Posting List L2, Path Separator S
3       Output : Intersection pid Set of Posting List
4       Algorithms :
5       begin
6
7       if(S=="/") {
8               (1) return set of posting_list1 that satisfaction following condition
9               condition : L1's pid equal to L2's pid and L1's sequences+1 equal to L2's sequence
10              return posting_list1;
11      }else if(S=="//"){
12              (2) return set of posting_list2 that satisfaction following condition
13              condition : L1's pid equal to L2's pid and L1's sequences+1 less than L2's sequence
14              return posting_list2;
15      }
16
17      end
```

(그림 7) 경로 질의 처리 알고리즘



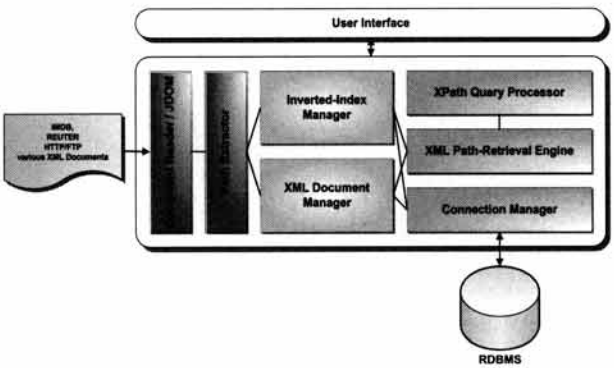
(그림 8) 경로 질의 '/movie//name/last' 검색 과정

검색 과정은 입력으로 들어오는 경로 질의에 대한 사전 처리를 하게 된다. 경로 질의를 경로 구분자와 엘리먼트로 분류하고(Step 1), 각 엘리먼트 별로 매칭되는 키워드를 posting 테이블에서 검색한 후 포스팅 리스트를 구성한다(Step 2). 구성된 포스팅 리스트에서 공통으로 존재하는 pid를 찾고 sequences를 고려하여 pid를 찾는다(Step 3). 앞의 예에서는 먼저 입력 경로 질의 '/movie//name/last'에 대해서 경로 식별자를 기준으로 분리하는 처리과정을 거친 뒤, movie, name, last와 일치하는 키워드의 포스팅 리스트를 구성한다. 그리고 구성된 포스팅 리스트의 비교를 통해서 name 엘리

먼트와 last 엘리먼트가 공통으로 존재하는 9, 15번 pid를 얻을 수 있는데, sequences는 name이 3이고 last는 4이다. 이 결과로 우리는 name과 last 엘리먼트가 부모-자식관계임을 알 수 있다. 끝으로 movie 엘리먼트와 name/last 엘리먼트의 결과 포스팅 리스트의 비교를 통해 9, 15번 경로에 movie 엘리먼트와 name/last 엘리먼트가 존재하고, movie와 name/last 엘리먼트 사이의 sequences의 비교를 통해서 조상-후손관계를 가지고 존재함을 알 수 있다. 검색 과정을 모두 끝마치면 pid 9, 15번이 '/movie//name/last'의 경로 패턴을 만족하는 결과 값이 된다.

4.3 제안 기법의 시스템 구조

(그림 9)는 본 논문에서 제안된 기법의 시스템 구조를 나타낸다. Document Reader/JDOM은 XML 문서를 로딩하고, Path Extractor는 XML 문서 경로를 추출한다. Inverted-Index Manager는 역 색인을 구성 및 유지·관리하고, XML Document Manager는 계층적 저장구조를 갖는 XMLDocuments 테이블에 엘리먼트의 정보를 유지·관리한다. XPath Query Processor는 입력 XPath 질의 변환을 하고, XML Path-Retrieval Engine은 경로검색 질의를 수행한다. Connection Manager는 JDBC/ODBC 드라이버를 통한 RDBMS 접근을 가능하게 한다.



(그림 9) 시스템 구조

5. 성능 평가

본 장에서는 RDBMS 기반의 XRel 시스템과 제안한 기법과의 성능을 비교한다. 성능 평가는 그룹별 XPath 경로 질의를 바탕으로 XRel 시스템과 본 논문에서 제안한 기법간의 검색 속도를 비교한다. 먼저 검색 속도는 경로 수의 증가에 따른 검색 속도 비교와 XPath 경로 질의별 분류에 따른 검색 속도를 비교한다.

5.1 실험 환경

<표 1>은 실험에 사용된 시스템 환경 및 구현도구 등의 실험 환경을 보여주고 있다. 실험에서는 RDBMS로 Oracle 10.1.0.2.0과 MS-SQL 2000 Server를 사용하였다. 2가지 RDBMS를 선정한 이유는 XRel에서 사용하는 기법(*LIKE* 연산)을 Oracle과 MS-SQL에 모두 구현하여 비교하기 위함이다. 구현 언어로는 Java 1.4.2_07 JDK를 사용하였으며, XML 파서는 J-DOM 1.0 API를 사용하였다. 입력으로 사용한 XML 문서는 IMDB[15]와 REUTER[16]에서 제공하는 샘플 XML 데이터와 HTTP와 FTP 상에 존재하는 약 3000개의 XML 문서를 사용하였다. 추출된 총 경로의 수는 119625개 이고, 추출된 유일 키워드(엘리먼트 명)는 2523개이다.

〈표 1〉 실험 환경

구 분	명 칭
CPU & Memory	P4 1.6GHz, RAM 512MB
운영체제	Windows 2000 Server
데이터베이스 서버	Oracle 10.1.0.2.0, MS-SQL 2000
구현언어	Java 1.4.2_07
XML 파서	J-DOM 1.0
입력 XML 문서	약 3000개의 XML 문서 (IMDB, REUTER, HTTP/FTP상에 존재하는 문서)

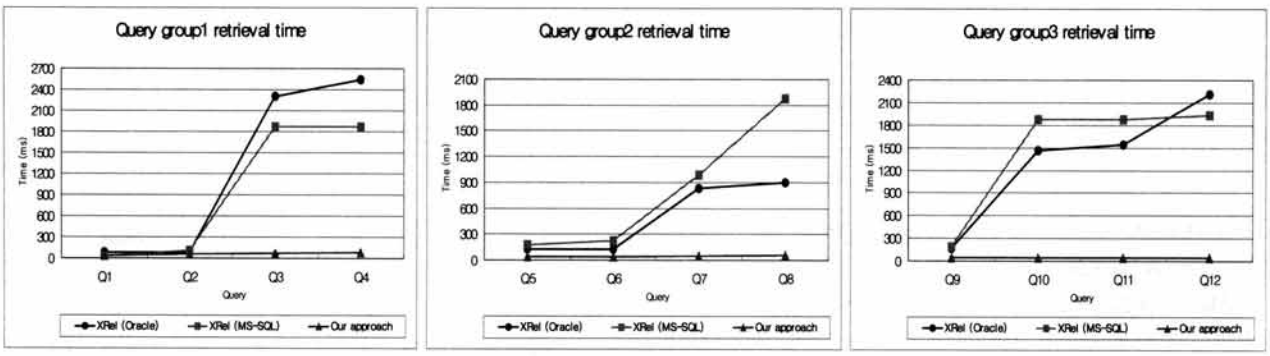
5.2 실험 결과

본 절에서는 RDBMS 기반 XRel 시스템의 검색속도와 제안한 기법간의 검색 속도를 비교하고자 한다. <표 2>는 실험에 사용한 XPath 질의 그룹을 보여주고 있다. 질의 그룹은 3개의 그룹으로 나뉘는데, 각 그룹은 노드의 분포별로 분류를 하였다. Group1은 질의 패턴이 전체 문서 집합에서 다수 등장하는 질의로 구성하였고, Group2는 Group1보다는 적지만 비교적 많이 등장하는 질의 그룹으로, Group3은 3개의 그룹 중 등장하는 분포가 가장 적은 질의 패턴으로 구성하였다. 질의의 선정은 부분 패턴 매칭 질의만을 고려하였다.

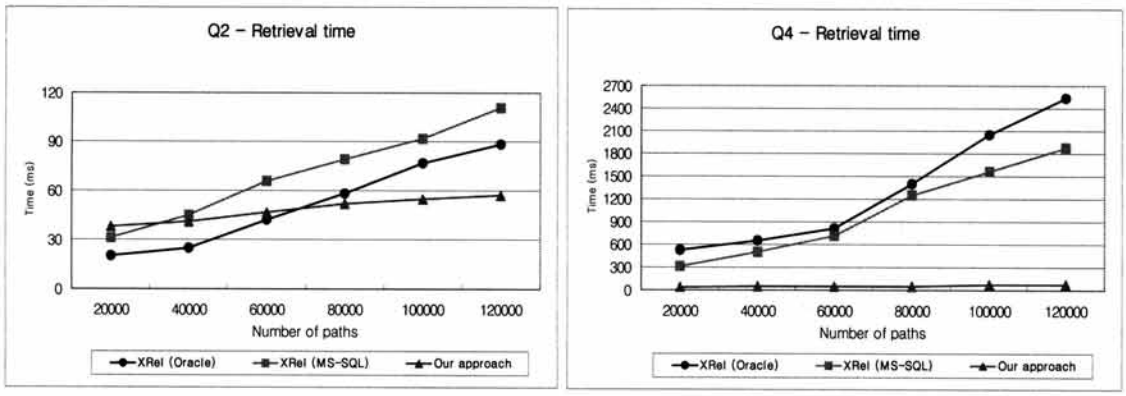
(그림 10)은 실험에 사용된 질의 그룹별 다양한 질의에 대한 검색 수행시간을 비교한 결과를 보여준다. 이 실험 결과는 Oracle과 MS-SQL에서의 XRel 시스템 성능과 제안한 기법과의 수행 시간 비교 결과이다. (그림 10)의 결과를 통해서 XRel 시스템은 검색 질의 패턴과 일치하는 노드의 수보다는 부분 매칭 질의의 길이(경로 구분자를 기준으로 한

〈표 2〉 실험에 사용된 XPath 질의 그룹

Group	Name	XPath Query	Number of node
Group1	Q1	//MOVIE	10114
	Q2	//MOVIE//CAST	6703
	Q3	//MOVIE//CAST//PLAYER	6106
	Q4	//MOVIE//CAST//PLAYER//NAME	965
Group2	Q5	//TEXT	8990
	Q6	//TEXT//BODY	2304
	Q7	//TEXT//BODY//DIV	1094
	Q8	//TEXT//BODY//DIV//HEAD	38
Group3	Q9	//ENTRY	4612
	Q10	//ENTRY//ABOUT	60
	Q11	//ENTRY//ABOUT//AUTHOR	15
	Q12	//ENTRY//ABOUT//AUTHOR//NAME	5



(그림 10) 질의 그룹별 검색 수행 시간



(그림 11) 경로 수 증가에 따른 검색 수행 시간

엘리먼트의 수)가 짧을수록 빠른 성능을 보임을 알 수 있다. 그리고 RDBMS에서 부분 패턴 매칭 질의를 처리하기 위해 사용하는 *LIKE* 연산은 약간씩의 차이는 있지만 Oracle과 MS-SQL에서 거의 비슷한 수행 시간을 보임을 알 수 있다. 결과에서 제안된 기법의 수행 시간은 질의 변화에 따라 큰 차이가 없으며 XRel 시스템보다 월등한 성능향상을 보인다.

(그림 11)은 경로 수 증가에 따른 경로 질의의 수행 시간을 비교한 결과를 보여준다. <표 2>의 XPath 경로 질의들 중 RDBMS에서 비교적으로 우수한 성능을 보인 Q2질의와 성능이 떨어지는 Q4를 대상으로 경로의 수 증가에 따른 검색 수행시간을 비교하였다. 수행 시간의 측정은 경로 20000개, 40000개, 60000개, 80000개, 100000개, 120000개의 그룹에서 각각 측정하였고, 각 그룹에서 Q2, Q4의 등장 빈도는 6703, 965번으로 각 경로 그룹마다 동일하게 설정하여 실험을 하였다.

Q2 결과에서 경로의 수가 약 60000개 정도가 될 때까지는 XRel 시스템이 대체적으로 우수한 성능을 보이지만 그 이상 경로가 증가하면 제안한 기법의 수행 시간이 훨씬 빠르다는 것을 볼 수 있다. 예를 들면, 제안한 기법에서 경로의 수가 10000개 일 때 Oracle과 MS-SQL에 비해 각각 약 40%, 80%의 효율이 증가하였고, 12000개일 때에는 Oracle, MS-SQL에 비해 각각 약 50%, 100%의 효율성을 보였다. 제안한 시스템은 경로가 아무리 적어도 포스팅 리스트 객체에 대한 처리과정 때문에 20ms~30ms의 시간을 소모하게 된다. 따라서 문서의 개수가 얼마 되지 않아 경로의 수가 적고, Q2의 질의와 같이 '//'의 개수가 적은 경우 RDBMS의 *LIKE* 연산을 사용하는 것이 좋다. 하지만 Q4 결과에서 보이는 것과 같이, 질의에 '//'의 개수가 많고 경로가 많아질수록 제안한 기법의 성능이 월등히 앞서는 것을 알 수 있다.

6. 결 론

본 논문에서는 계층적 저장구조를 사용하여 XML 문서 엘리먼트 간에 존재하는 다양한 관계 연산을 제안하였다. 또한, 기존 RDBMS 기반 시스템인 XRel의 비효율적인 경로 비교 연산을 개선한 새로운 검색 기법을 제안하였다. XRel 시스템에서는 부분 패턴 매칭 질의 시, 존재하는 모든 경로에 대해서 스트링 매칭을 통해 결과를 얻어오기 때문에 문서의 수가 증가할수록 검색 수행 시간이 급격히 느려지게 된다. 반면에 XRel과 같이 RDBMS 기반의 제안된 기법은 경로 역 색인을 사용하여 입력 질의에 나타나는 엘리먼트 수만큼의 포스팅 리스트를 비교함으로써 문서의 증가에 크게 영향을 받지 않고 우수한 검색 성능을 보임을 실험을 통해 입증하였다.

그러나 제안된 기법이 우수한 검색성능을 보이지만 경로 역 색인의 구성 및 유지관리를 위해 많은 자원을 소모하게 되는 단점이 있다. 문서의 삽입, 삭제, 수정으로 인한 포스팅 리스트 B-Tree 색인의 갱신은 시스템 자원의 과부하를 초래한다. 향후 연구로는 이러한 경로 색인의 구성 및 유지비

용 절감을 위한 경로 축약 기법이나 효율적인 색인 구조의 연구가 필요하다.

참 고 문 헌

[1] J. McHugh, S. Abiteboul, R. Goldman, D. Quass, and J. Widom, "Lore: A Database Management System for Semistructured Data," ACM SIGMOD Record, Vol.26, No.3, pp.54-66, 1997.

[2] R. Goldman and J. Widom, "Dataguides: Enabling Query Formulation and Optimization in Semistructured Databases," Proc. of the 23rd Int'l Conf. on Very Large Databases, pp.436-445, 1997.

[3] C. Chung, J. Min, and K. Shim, "APEX: An Adaptive Path Index for XML Data," Proc. of the Int'l Conf. on ACM SIGMOD, pp.121-132, Madison, Wisconsin, June, 2002.

[4] R. Kaushik, P. Shenoy, P. Bohannon, and E. Gudes, "Exploiting Local Similarity for Indexing Paths in Graph-Structured Data," Proc. of the 18th IEEE Int'l. Conf. on Data Engineering, pp.129-140, 2002.

[5] B. F. Cooper, N. Sample, M. J. Franklin, G. R. Hjaltason, and M. Shadmon, "A Fast Index for Semistructured Data," Proc. of the 27th Int'l Conf. on Very Large Databases, pp.341-350, Rome, Italy, Sep., 2001.

[6] J. Yoon, V. Raghavan, V. Chakilam, and L. Kerschberg, "BitCube: A Three-Dimensional Bitmap Indexing for XML Documents," J. of Intelligent Information Systems, Vol.17, pp. 241-254, 2001.

[7] J. Yoon, V. Raghavan, and V. Chakilam, "BitCube: Clustering and Statistical Analysis for XML Documents," Proc. of the 13th Int'l Conf. on Scientific and Statistical Database Management, Virginia, 2001.

[8] 이재민, 황병연, "xPlaneb: XML 검색을 위한 3차원 비트맵 인덱스," 정보과학회논문지, 31권, 3호, pp.331-339, 2004.

[9] D. Hong, "On supporting full-text retrievals in XML query," International Journal of Fuzzy Logic and Intelligent Systems Vol.7, No.4, pp.274-278, 2007.

[10] M. Yoshikawa and T. Amagasa, "XRel: A Path-Based Approach to Storage and Retrieval of XML Documents using Relational Databases," ACM Transactions on Internet Technology, Vol.1, No.1, pp.110-141, 2001.

[11] R. Krishnamurthy, R. Kaushik, and J. Naughton. "XML-to-SQL Query Translation Literature: The State of the Art and Open Problems," Proc. of the 1st Int'l XML Database Symposium, pp.1-18, Berlin, Germany, Sep., 2003.

[12] H. Jiang, H. Lu, W. Wang, and J. X. Yu, "Path Materialization Revisited: An Efficient Storage Model for XML Data," Proc. of the 13th Australian Database Conference, pp.85-94, Melbourne, Australia, Jan., 2002.

[13] 정민경, 홍동권, 남재열, "XML을 RDBMS에 저장하기 위한 Analyzer 설계 및 구현," 한국정보과학회 2005 한국컴퓨터종합학술대회 논문집, pp.148-150, 2005.

[14] 김재훈, 여준호, 이규철, "메모리-상주 관계형 DBMS에서 XML 데이터 처리를 위한 효율적인 저장 기법," 한국정보과학회 2008 가을 학술발표논문집 제35권 제2호, pp.55-59, 2008.

[15] http://us.imdb.com/top_250_films

[16] http://about.reuters.com/newsml

문 경 원

e-mail : mmmkiller@catholic.ac.kr

2004년 가톨릭대학교 컴퓨터공학과(공학사)

2006년 가톨릭대학교 컴퓨터공학과(공학석사)

2006년~2008년 ㈜비츠로시스 연구원

2008년~현 재 ㈜NHN 검색개발센터 과장

관심분야 : 대용량 데이터베이스 시스템, XML 데이터베이스

황 병 연

e-mail : byhwang@catholic.ac.kr

1986년 서울대학교 컴퓨터공학과(공학사)

1989년 한국과학기술원 전산학과(공학석사)

1994년 한국과학기술원 전산학과(공학박사)

1994년~현 재 가톨릭대학교 컴퓨터정보공학부 교수

1999년 University of Minnesota Visiting Scholar

2007년 California State University, Sacramento Visiting Scholar

관심분야 : XML 데이터베이스, 데이터마이닝, 정보검색, 지리정보시스템