XML 문서에서 순수 구조 질의에 대한 인덱싱 및 질의 처리

김성완, 김연희*, 이재호**, 임해철*
삼육의명대학 컴퓨터정보과, *홍익대학교 컴퓨터공학과, **인천교육대학교 컴퓨터교육과
swkim@syu.ac.kr, {kyh, lim}@cs.hongik.ac.kr, jhlee@mail.inue.ac.kr,

Indexing and Query Processing for Pure Structure Query on XML Documents

Sung Wan Kim, Youn Hee Kim[1], Jaeho Lee[2], Hae Chull Lim[1]
Dept. of Computer Information, Sahm Yook College, [1]Dept. of Computer Engineering, Hong Ik Univ.,
[2]Dept. of Computer Education, Inchon Nat'l Univ. of Education,

요 약

XML 문서의 효율적인 저장 및 검색을 위한 많은 연구들이 수행되고 있다. 그러나, 기존의 연구들에서는 주로 내용 검색의 정확도를 높이기 위해 구조적 정보를 이용하는 방법을 제시하고 있으나, 순수하게 구조만을 대상으로 하는 인덱싱 및 질의 처리 방법 특히, 동적인 환경을 고려한 인덱싱 및 질의 처리 방법에 대해서는 많이 언급하고 있지 않다. 본 논문에서는 XML 문서에 대한 순수 구조 질의 처리가 가능하고, 엘리먼트의 삭제 및 삽입 등 동적인 변경을 처리할 수 있는 인덱스 구조를 설계한다. 또한 설계된 인덱스 구조를 기반으로 순수 질의 처리 유형을 분류하고 각각에 대한 질의 처리 방안을 제시한다.

1. 서 론

XML이 웹 상의 문서에 대한 표현 및 교환의 표준 포맷으로 인식되면서, XML 문서의 효율적인 저장 및 검색을 위한 많은 연구들이 수행되고 있다. 이러한 XML 문서에 대한 저장 연구들은 내용 기반 검색 및 구조적 특징을 포함하는 내용 기반 검색이 많은 부분을 차지하고 있으며 이에 대한 연구 결과들이 상당히 나와있다.

특히, 구조 기반 질의의 효율적 처리를 위해서는 문서의 임의 엘리먼트에 빠르게 접근 할 수 있는 인덱싱 기법이 필요하다. 이를 위한 기존의 연구들에서는 전통적인 정보 검색 분야에서 사용되는 역 리스트를 응용한 기법들이 제안되었다[1][2][5]. 이러한 연구들에서는 주로 내용 검색의 정확도를 높이기 위해 구조적 정보를 이용하는 방법을 제시하고 있으나, 순수하게 구조만을 대상으로 하는 질의 처리 방법에 대해서는 언급되지 않고 있다. 또한, [4][6][7] 연구들에서는 XML 문서를 엘리먼트를 노드로 하는 트리 형태로 표현하고, 각 노드에 구조적 정보를 포함하는 특별한 식별자를 부여하여 순수 구조 질의 처리가 가능하도록 설계하였다. 그러나 일부 순수 구조 기반 질의 유형에는 효율적이나 다른 유형의 처리를 위해서 추가적인 연산이 요구된다. 또한, XML 문서에 대한 엘리먼트 삽입 및 삭제와 같은 구조적인 변경이 자주 발생하는 환경에서는 노드에 할당된 식별자에 대해 많은 재구성이 요구되므로 동적인 환경에서는 매우 부적합하다.

본 논문에서는 첫째, XML 문서에 대해 일부 엘리먼트의 삽입 및 삭제 등 동적인 변경을 빠르게 처리할 수 있는 인덱스 구조를 설계한다. 둘째, 설계된 인덱스 구조를 기반으로 순수 구조 질의에 대한 유형을 분류하고 각 유형에 대한 질의 처리 방안을 제시한다.

2장에서는 관련 연구를, 3장에서는 본 논문에서 제안하는 인덱스 구조에 대해 설명한다. 4장에서는 제안된 인덱스 구조를 기반으로 XML 문서에 대한 순수 구조 질의 유형을 분류하고 질의 처리 루틴에 대해 설명한다. 5장에서는 결론과 향후 연구를 설명한다.

2. 관련 연구

XML 문서의 구조 정보를 표현하는 방법은 첫째, XML 문서를 트리로 모델링 하여 엘리먼트를 노드로 매핑한 후 각 노드에 대해 구조적 특징을 포함하는 특수한 식별자를 할당하는 방법이다. 노드에 전위순회-후위순회 순서 값[4]을 할당하는 방법은 조상-후손 관계를 상수시간에 결정할 수 있으므로 조상 또는 후손 검색에 용이하나, 특정 레벨 또는 특정 순서를 포함한 구조 질의 처리에는 추가적인 구조 또는 연산이 필요하다. 또한, 근본적으로 새로운 엘리먼트의 추가 등의 변경에 대처할 수 없다. 경로 엘리먼트 ID를 이용하는 방법[6]은 부모, 자식, 형제 및 트리 구조상 특정 위치 엘리먼트를 직접 또는 간단한 처리를 통해 검색할 수 있다. 그러나 문서 크기가 커질수록 ID가 무한대로 늘어나며, 삽입 및 삭제 연산시 경로 ID의 재계산이 요구되는 등 동적 환경에 부적합하다.

둘째, XML 문서 트리를 k-ary 완전 트리로 가정하여 노드에 고유 식별자를 할당하는 방법[1][2][3]이다. 여기서는 XML 문서 트리의 노드에 k-ary 완전 트리의 레벨순 노드 방문 순서 번호를 식별자로 할당한다. 식별자 i를 갖는 노드에 대한 부모 노드의 식별자는 $\lfloor (i-2)/k + 1 \rfloor$ 수식과, j번째 자식에 대한 식별자는 $k(i-1)+j+1$ 의 수식으로 간단히 구해낼 수 있다. 따라서, 특정 레벨의 조상과 특정 레벨의 특정 순서의 자손은 공식에 의해 반복적으로 쉽게 접근할 수 있다. 그러나, 구해진 식별자의 실제 존재 유무 확인 처리과정이 필요되며, 문서의 크기가 커질수록 식별자의 값이 지수에 비례하여 커지게 되므로 식별자 범위 한계를 벗어날 가능성이 있다. 또한, 엘리먼트의 부분 삽입 및 삭제의 경우 식별자 값의 재계산이 요구된다.

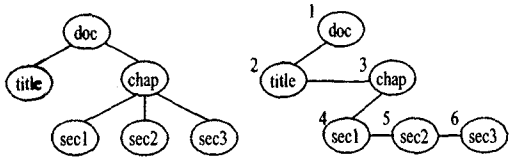
3. 인덱스 구조의 설계

3.1 구조 정보 표현을 위한 XML 문서 모델링

XML 문서의 구조 정보를 정확히 나타내기 위해서는 XML 문서상에서 부모 및 자식과 같은 엘리먼트간의 계층적 포함 관계와 형제와 같은 엘리먼트간의 수평적 관계를 모두 표현할 수 있는 모델링 과정이 필요하다. 일반적으로 XML 문서는 엘리먼트를 노드로 표현하는 트리 형태로 모델링 된다. 여기서는 이

를 XML 문서 트리(XML document tree)라고 하자. 이 문서 트리는 각 노드의 fan out이 각기 다른 일반 트리 형태이다.

본 논문에서는 일반 트리에 대해 동일한 노드 구조를 갖는 트리로 표현하는 방법 중 하나인 왼쪽 자식-오른쪽 형제 표현 방법[7]을 활용하여 XML 문서 트리를 모델링 하였다. 모델상의 노드의 기본적인 논리적 표현 구조는 <엘리먼트, 왼쪽 자식 포인터, 오른쪽 형제 포인터>가 된다. 다음 <그림1>의 오른쪽 표현은 왼쪽의 일반 XML 문서 트리를 왼쪽 자식-오른쪽 형제 표현 방식으로 다시 모델링 한 것이며, 노드에 일련의 식별자를 할당한 모습이다. 그림에서는 실제 텍스트 데이터 등은 생략하고 엘리먼트만으로 구성된 논리적 구조만 표현했다.



<그림 1> XML 문서 트리

3.2 구조 정보 표현을 위한 인덱스 구조

왼쪽 형제-오른쪽 자식 표현 기법을 응용하여 XML 문서 트리상의 구조적 정보를 손실 없이 모두 포함하는 인덱스 구조는 아래와 같다. 우선, 왼쪽 형제-오른쪽 자식 표현 기법에 따라 모델링된 XML 문서 트리상의 각 노드를 하나의 엔트리로 매핑한다. 엔트리 구조는 아래와 같이 9개의 필드로 구성된다.

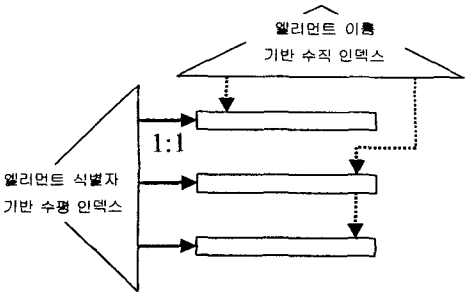
<UID, EName, PUID, LUID, RUID, FUID, NOC, ELEV, FOW>

UID는 고유한 엘리먼트 식별자를 의미하며, 식별자의 값은 특별한 순서나 구조적 특징을 내포하는 특수한 값이 아닌 일반적인 숫자 값을 갖는다. EName은 엘리먼트의 이름을 의미하며, PUID, LUID, RUID, FUID는 각각 부모 엘리먼트의, 바로 왼쪽 형제, 바로 오른쪽 형제, 첫째 자식에 대한 식별자 값을 의미한다. NOC는 자식 엘리먼트의 개수를, ELEV은 XML 문서 트리에서 엘리먼트가 위치한 레벨을 의미한다. 마지막으로 FOW는 동일 엘리먼트 명을 갖은 엔트리를 그룹핑 하기 위해 사용되는 엔트리에 대한 포인터 필드이다. 예를 들어 <그림 1>에서 식별자 3인 엘리먼트 노드를 위한 엔트리는 다음과 같다.

<3, chap, 1, 2, Null, 4, 3, 2, next_ptr>

인덱스 구조는 이렇게 생성된 엔트리들을 포스팅으로 하고, 엔트리의 엘리먼트 식별자 필드의 값을 키로 인덱스 파일을 구성하는 역 리스트 구조이다. 인덱싱 구성 과정은 1차적으로 XML 문서 트리 상의 각 엘리먼트 노드에 대해 위에서 언급한 방식과 같이 엔트리들을 생성한 후, 생성된 엔트리들을 엘리먼트 식별자 값을 키로하여 B+ 트리 등을 이용한 인덱스 파일에 삽입하고 포스팅과 연결한다. 여기서 인덱스에 삽입된 키와 포스팅은 1:1 관계이다. 이러한 XML 문서 트리에 대한 구조 정보를 포함하는 전체적인 인덱스 구조은 <그림 2>와 같다.

한편, 실제 사용자의 질의는 특정 엘리먼트 식별자가 아닌 엘리먼트 이름을 기반으로 주어지므로 주어진 엘리먼트 이름과 엘리먼트 식별자의 사상을 위한 사상 테이블이 별도로 요구된다. 또한, XML 문서 컬렉션에는 동일 이름의 엘리먼트가 많이 존재하므로 하나의 엘리먼트 이름은 다수의 엘리먼트 식별자와 관련되어야 한다.



<그림 2> 구조 정보 인덱스 구조

본 논문에서는 이러한 별도의 사상 테이블을 유지하지 않고, 주어진 이름과 동일한 엘리먼트들에 대해 직접적이고 빠르게 접근하여 질의 처리 시간을 단축하기 위해 엘리먼트 이름을 키로하는 또 다른 차원의 인덱스를 두어 이중 인덱스 기반의 역 리스트 구조로 설계하였다. <그림 2>의 엘리먼트 식별자를 키로하는 인덱스를 '***UID 기반의 수평 인덱스***', 엘리먼트 이름을 키로하는 인덱스를 '***엘리먼트 이름 기반의 수직 인덱스***'라 한다. 특히, 별도의 테이블 등에 대한 접근 과정 없이 엘리먼트 이름 기반의 수직 인덱스를 이용하여 같은 엘리먼트 이름을 가지는 포스팅들 서로 연결하여 특정 이름을 가지는 엘리먼트들에 대한 포스팅들만을 직접적으로 검색할 수 있다. 여기서 가변적인 크기의 엘리먼트 이름 필드의 공간 낭비를 위해서 실제 인덱스 구성에서는 엘리먼트 이름 필드를 제거할 수 있다.

한편, 이렇게 왼쪽 자식-오른쪽 형제 표현에 대한 용용 방법을 활용한 제안 인덱스 구조는 XML 문서에 대한 새로운 노드의 삽입 및 삭제 등 구조적인 동적 갱신에 대해서 직접 연관된 포스팅외에 다른 포스팅에는 영향을 주지 않게 된다. 예를 들어 <그림 1>의 왼쪽 XML 문서 트리상에서 식별자 2와 3의 사이에 새로운 엘리먼트가 삽입될 경우, 우선, 삽입될 엘리먼트 노드에 대한 엔트리 포스팅을 생성하고, 식별자는 가용한 숫자를 할당(여기서는 7)하고, PUID, LUID, RUID, ELEV 를 각각 1, 2, 3, 2으로 채운다. 그리고 식별자 2에 대한 포스팅 엔트리의 RUID를 7로, 식별자 3에 대한 포스팅 엔트리의 LUID를 7로 변경하고 식별자 1의 NOC를 1증가 시킨다. 그리고 최종적으로 새로 생성된 포스팅 엔트리의 UID와 EName을 키로하여 수평적 인덱스와 수직적 인덱스에 적절하게 추가하면 된다.

4. 순수 구조 질의 유형 분류 및 질의 처리

4.1 순수 구조 질의 유형 분류

먼저 3장에서 설계된 인덱스 구조를 기반으로 순수 구조 질의는 처리 특성에 따라 3가지로 분류된다.

1) 질의에 포함된 최종 목표 대상 엘리먼트의 유무
2) 질의에 포함된 시작 엘리먼트와 직접 연관된 경우 (single step 처리 가능)와 그렇지 않은 경우 (multiple step 처리 요구)
3) 특정 레벨/순서만 한정하는 경우 또는 특정 레벨/순서까지 포함하여 검색하는 경우

또한, 순수 구조 기반 질의는 그 성격에 따라 조상/부모 검색, 형제 검색, 자식 검색, 후손 검색 등 4가지로 유형으로 분류된다. 여기서, 부모의 경우 첫 번째 조상 검색으로 대체할 수 있으므로 조상 검색과 통합하였다. 그러나, 자식 검색의 경우 첫 번째 후손으로 대체할 수 있으나 질의 처리 방법의 효율성 및 복잡성을 고려하여 따로 분류하였다. 한편, 첫 번째 제시된

3가지 분류 사항을 조합하면 6가지의 순수 구조 질의의 조합이 가능하다. 예를 들어, 조상 및 부모(첫번째 조상) 검색의 경우 아래의 6가지 경우가 가능하며, 나머지 3가지 유형에 대해서도 6개의 조합으로 세분할 수 있다.

1) All elements - <a>의 조상 엘리먼트를 검색하라
2) All elements with a specific name - <a>의 조상 엘리먼트 중 <b>인 엘리먼트를 검색하라
3) All elements at a specific level - <a>의 2번째 조상 엘리먼트를 검색하라
4) All elements with a specific name at a specific level - <a>의 2번째 조상 엘리먼트 중 <b>인 엘리먼트를 검색하라
5) All elements within a specific level - <a>의 2번째 까지 포함된 조상 엘리먼트를 모두 검색하라
6) All elements with a specific name within a specific level - <a>의 2번째 까지 포함된 조상 엘리먼트 중 <b>인 엘리먼트를 검색하라

```
Ancestor_Parent(str SN, str TN, int ORD, int FLG)
S1. Name 필드가 SN인 엔트리들 추출하여 후보 집합 A 구성
S2. ForEach 엔트리 E of 후보 집합 A
If (ORD == 0) {   // 조상 끝까지 엘리먼트 검색
  엔트리 E1 <- E의 PUID를 UID로 가지는 새 엔트리 검색
  While (엔트리 E1 != NULL) {
    If (TN == Null) // 모든 조상 검색
      최종결과집합 R <- 엔트리 E1 의 UID 추가
    Else If (TN == E1의 Name 필드) // TN과 같은 조상 검색
      최종결과집합 R <- 엔트리 E1 의 UID 추가
    Else // Skip
    엔트리 E1 <- E1의 PUID를 UID로 갖는 새로운 엔트리 검색
}}
Else If (ORD > 0 AND FLG == 1) {     // 특정 레벨만 검색
  CNT <- 1    // ORD가 1, FLG가 1일 경우 부모 검색 포함
  엔트리 E1 <- E의 PUID를 UID로 가지는 새로운 엔트리 검색
  While (엔트리 E1 != NULL) {
    If (CNT > ORD ) break
    If (TN == Null AND CNT == ORD) // 특정 레벨 조상 검색
      최종결과집합 R <- E1 의 UID 추가
    Else If (TN == E1의 Name 필드 AND CNT == ORD)
      최종결과집합 R <- E1 의 UID 추가  // 특정레벨 중
    Else // Skip                          특정이름 검색
    CNT++
    엔트리 E1 <- E1의 PUID를 UID로 갖는 새로운 엔트리 검색
}}                    // 특정 레벨까지 모두 포함하여 검색
Else If (ORD > 0 AND FLG == 2) {
  // 위의 Else If절과 동일
  // 단, 내부의 If문 조건에서 CNT == ORD를 CNT <= ORD로 수정
}
```

<그림 3> 조상/부모 검색 처리 루틴

4.2 질의 처리 루틴

질의 처리 루틴은 위의 분류에 따라 4가지로 설계하였다. 후손 검색은 재귀적 방법으로 처리 가능하며, 나머지 3개의 검색 유형에 대한 처리는 반복적 방법으로 처리 가능하다. 특히, 첫째 조상(부모), 첫째 왼쪽 또는 오른쪽 형제, 첫째 자식 검색의 경우는 single step으로 빠르게 처리가 가능하다.

<그림 3>은 4가지 순수 질의 유형 중 조상 및 부모(첫째 조상)검색에 대한 루틴이다. 처리 방식은 상향식으로 처리하며, 6가지 질의 조합에 대해 목표 레벨이 없는 전체 조상에 대한 검색의 경우, 목표 레벨을 한정하는 검색의 경우, 목표 레벨까지 포함하는 검색의 경우 등 3가지로 구분하여 루틴을 작성하였다. 입력 매개 변수의 의미는 다음과 같다. SN은 사용자로부터 주어지는 시작 엘리먼트의 이름이며, TN은 목표 엘리먼트의 이름, ORD는 레벨, FLG는 주어진 레벨만을 한정하는 경우 또는 포함하는 경우를 구별하기 위한 매개 변수이다. 이외에 자식 검색, 형제 검색도 이와 유사한 방식으로 처리할 수 있다. <그림 4>는 후손 검색중 특정 레벨이 주어지지 않은 경우 즉, 모든 후손 엘리먼트 검색에 대한 처리 루틴이며, 재귀적으로 작성하였다. 특정 레벨에 대한 질의도 이를 응용하여 통합 처리 루틴을 작성할 수 있다.

```
All_Descendant (str SN, str TN=Null )
S1. Name 필드가 SN 인 엔트리들 추출하여 후보 집합 A 구성
S2. For Each 엔트리 E of 후보 집합 A
      call Func(엔트리 E->FirstCHD, TN)

Func(엔트리 A, str TN)
If (엔트리 A != NULL) {
  If (TN == NULL)                  // 모든 후손 엘리먼트 검색
    최종결과집합 R <- 엔트리 A 추가 (엔트리 A의 UID 추가)
  Else If (TN != Null And 엔트리 A의 Name 필드 == TN)
                  // 모든 후손 중 특정 이름 가진 것만 검색
    최종결과집합 R <- 엔트리 A 추가 (엔트리 A의 UID 추가)
  Else // Skip
  call Func (엔트리 A의 FirstCHD가 가르키는 새로운 엔트리 A1)
  call Func (엔트리 A의 RightSIB가 가르키는 새로운 엔트리 A1)
}
```

<그림 4> 후손 검색 처리 루틴 (일부)

5. 결론 및 향후 연구

본 논문에서는 XML 문서에서 동적인 변경을 고려한 순수 구조 질의 처리를 위한 인덱스 구조를 설계하고, 질의 유형을 분류하고 이에 대한 처리 루틴을 설계하였다. 현재, 본 제안 기법에 대한 실제적 구현을 하고 있으며 향후, 성능평가와 처리 기법에 대한 추가적인 연구를 수행할 예정이다.

<참고문헌>

1. Y.K. Lee et al., "Index structures for structured documents", Proc.of the 1st ACM Int'l Conf.on Digital Libraries, 1996
2. D. Shin et al., "BUS:An Effective Indexing and Retrieval Scheme in Structured Documents", ACM Digital Libraries, 1998
3. D. Shin, "XML Indexing and Retrieval with a Hybrid Storage Model", Knowledge and Information Systems, 2001
4. Q. Li, B. Moon, Indexing and Querying XML Data for Regular Path Expressions, VLDB 2001.
5. Evangelos Kotsakis, Structured Information Retrieval in XML documents, ACM SAC 2002
6. 연제원 외 3인, "XML 문서 구조검색을 위한 저장 시스템 설계", 한국정보과학회 봄 학술발표 논문집, Vol. 26, 1999
7. E. Horowitz, et al., Fundamentals of Data Structures in C, Computer Science Press, 1993