# 구조화 문서에서 점진적 갱신을 위한 인덱싱 기법

김성완[1], 김선경°[2], 이재호[3], 임해철[2]
[1]삼육의명대학 컴퓨터정보과, [2]홍익대학교 컴퓨터공학과, [3]인천교육대학교 컴퓨터교육과
swkim@syu.ac.kr, {skkim°, lim}@cs.hongik.ac.kr, jhlee@mail.inue.ac.kr

# An Indexing Scheme for Incremental Updating in Structured Document

Sung Wan Kim[1], Sun Kyung Kim°[2], Jaeho Lee[3], Hae Chull Lim[2]
[1]Dept. of Computer Information, Sahm Yook College, [2]Dept. of Computer Eng., Hong Ik Univ.,
[3]Dept. of Computer Education, Inchon Nat'l Univ. of Education

## 요 약

구조화 문서(structured document)에 대한 효율적인 처리를 위해서는 문서의 임의 엘리먼트에 빠르고 직접적인 접근을 지원하는 인덱싱 기법이 필요하다. 이를 위한 기존의 연구들에서는 전통적인 정보 검색 분야에서 사용되는 역 리스트나 시그너처 파일을 응용한 기법들이 제안되었다. 그러나 기존의 연구들은 정적인 환경에 적합한 인덱스 구조로써, 문서에 대한 동적인 변경이 있을 경우 인덱스를 전체적으로 재구성해야 하는 부담이 있다. 본 논문에서는 역 리스트를 기반으로 문서에 대한 구조 변경과 내용 변경 등 동적인 변경에 대해 점진적 갱신을 지원하는 인덱스 구조를 설계하였다.

## 1. 서 론

최근 전자도서관 또는 출판 등의 응용에서 SGML이나 XML과 같은 마크업 언어를 사용하여 생성한 구조화된 문서를 관리하기 위한 연구들이 계속되어 왔다. 이러한 구조화 문서에 대한 처리는 내용 기반 질의뿐만 아니라 구조 기반 질의를 효율적으로 처리할 수 있어야 한다. 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 리스트나 시그너처 파일을 응용한 기법들이 제안되었다[1][2][3].

그러나 기존의 연구들은 정적인 환경에 적합한 인덱스 구조로써, 문서에 대한 동적인 변경이 있을 경우 인덱스를 재구성해야 하는 부담이 있다. 따라서 본 논문에서는 기존의 인덱스 구조를 기반으로 문서에 대한 구조 변경과 내용 변경 등 동적인 변경에 대해 점진적 갱신을 지원하는 인덱스 구조를 설계하였다.

2장에서는 기존 연구에서의 인덱스 기법과 그 문제점을 살펴보고 3장에서는 점진적 갱신을 지원하도록 제안한 인덱스 구조를 설명한다. 4장에서는 결론과 향후 연구를 설명한다.

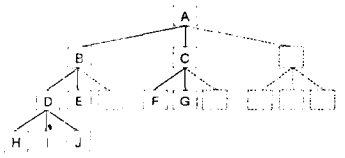
## 2. 관련 연구

[1]에서는 구조적 질의의 처리를 위한 역 인덱스 기법을 제안하였다. 여기서는 구조화 문서를 k-ary 완전 트리로 표현하였으며 이를 '문서 트리(document tree)'라 한다. k는 문서 구조에서 가장 많은 자식 엘리먼트를 가진 노드의 차수이다. 이 문서 트리를 레벨순 순회를 하며 각 엘리먼트에 고유한 식별자인 UID(Unique IDentifier)를 할당한다. 이렇게 UID를 할당할 경우 UID 값 i를 갖는 임의의 노드에 대한 부모 노드의 UID는 다음과 같은 간단한 수식에 의해 직접 계산할 수 있다.

$$부모(i) = \lfloor (i-2)/k + 1 \rfloor \quad \cdots\cdots\cdots (1)$$

본 연구는 정보통신연구진흥원 대학기초연구지원사업(과제번호 : 2001-122-3)의 지원을 받았음

예를 들어 <그림1>과 같은 문서 트리의 각 엘리먼트는 <표 1>과 같은 UID 값을 할당한다. <그림 1>에서 점선으로 된 엘리먼트 노드는 실제 존재하지 않는 가상 노드이다(단, k=3).




<그림 1> 문서 트리의 예

<표 1> UID할당

| Ele | UID | Ele. | UID |
|---|---|---|---|
| A | 1 | F | 8 |
| B | 2 | G | 9 |
| C | 3 | H | 14 |
| D | 5 | I | 15 |
| E | 6 | J | 16 |

한편, 임의 레벨의 엘리먼트에 대한 직접적 접근을 위해서는 문서 트리의 중간 노드는 실제 키워드(keyword 또는 term)을 포함하지 않지만, 리프 노드에 위치한 텍스트 엘리먼트의 데이터는 반드시 중간 노드들의 데이터로 간주되어야 한다. 특히, 이 기법에서는 임의 부모 엘리먼트의 모든 자식이 동일한 키워드를 포함하는 경우, 이 키워드를 모든 자식이 아니라 부모 레벨에 포함시켜 인덱싱하여 인덱싱에 대한 저장 오버헤드를 줄이는데 중점을 두고 있다. 그러나, 이 기법은 값 일치(exact match) 질의에는 유용하나, 키워드에 대한 가중치를 기반으로 하는 랭킹 질의를 고려하지 않고 있다.

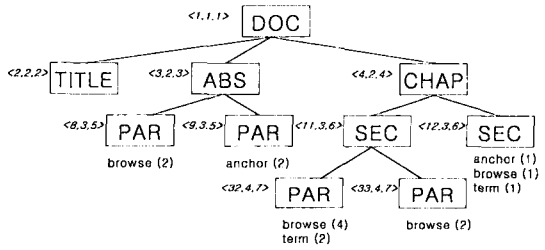
### 2.1. BUS(Bottom Up Scheme) 인덱싱 기법

BUS기법[2][3]에서는 [1]에서 사용된 UID를 확장하여 랭킹 질의가 가능하다. 이 기법은 문서 트리 구조의 리프 레벨만 인덱스를 구축하고, 임의 레벨 접근을 위한 질의는 리프 레벨에서부터 상향식으로 키워드의 빈도(frequency)를 누적하면서 처리하는 것이다. 이 기법에서는 UID를 확장한 GID(General element IDentifier)를 문서 트리의 각 엘리먼트 노드에 할당한다. GID는 UID외에 문서 번호(DID), 엘리먼트의 레벨(LEV), 엘리먼트의 타입(EID) 등 3가지 사항을 포함한다. DID는 엘리먼트가 속한 문서를 구별하기 위한 것이고, LEV과 EID는 사용자가 원하는 임의 레벨에서의 키워드 빈도를 재계산하는데 사용된다. LEV는 질의에 포함된 사용자 레벨과 실제 인덱싱 대상인 리프 레벨간의 레벨 차이(difference)를 계산하여 질의처리

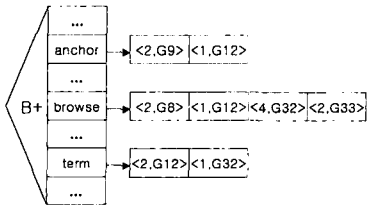
시 상향 진행 횟수를 결정한다. EID는 질의에 포함된 사용자의 원하는 타입과 다른 타입의 엘리먼트를 필터링하는데 사용한다. 여기서 엘리먼트 타입은 문서 트리의 실제 각 엘리먼트가 아니라, XML로 표현된 문서의 경우 DTD와 같은 구조에 포함된 엘리먼트 이름을 참조하여 부여한다.

인덱싱 구축은 첫째, 각 노드에 대해 GID를 할당하고, 텍스트(리프) 노드를 대상으로 키워드와 빈도를 추출한다. <그림 2>은 한 문서 트리에 대해 각 엘리먼트 노드에 GID와 리프 레벨의 각 텍스트 노드에 대해 키워드를 추출하여 그 빈도와 함께 표현한 간단한 예이다. 예제를 단순화 하기 위해 하나의 문서만을 가정하면 GID는 <UID, LEV, EID>로 구성된다. 예를 들어 'ABS' 엘리먼트의 가장 왼쪽 자식인 'PAR' 엘리먼트의 GID는 <8,3,5> 값을 가지며(이를 G8이라 하자), 추출된 키워드는 'browse'이며 빈도는 2이다. 여기서, 같은 엘리먼트 이름도 위치한 경로에 따라 구별되어야 하므로 [2][3]의 논문에는 명시되어 있지 않으나 <표 2>와 같은 정보를 유지하여 문서 구조상의 각 경로에 대해 엘리먼트 타입을 부여해야 한다.

<그림 2> GID, 키워드, 빈도가 포함된 문서 트리의 예




둘째, GID, 추출된 키워드, 빈도를 사용하여 포스팅을 생성한다. 예를 들어 UID가 32인 'PAR' 엘리먼트의 경우 <browse, 2, G32>, <term, 2, G32>가 생성된다. 여기서 첫 번째 항목은 추출된 키워드, 두 번째는 빈도수, 세 번째는 해당 엘리먼트의 GID 이다. 이렇게 각 텍스트 노드에 대해 생성된 포스팅들을 가지고 각 키워드에 기반한 B+트리와 포스팅 리스트로 구성된 역 리스트 구조의 인덱스를 구축한다. <그림 3>은 <그림 2>에 대한 역 리스트 구조이다.




<그림 3> B+트리 기반의 포스팅 구조

"browse를 포함하는 섹션(SEC)을 검색하라"와 같은 질의 처리는 먼저 <표 2>와 같은 정보 테이블을 참조하여 사용자 질의를 분석 후 질의에 포함된 목표 레벨(이 예에서는 3)과 질의 처리에 포함할 EID(이 예에서는 6, 7)를 추출한다. 그리고 역 리스트 인덱스를 통해 'browse'를 포함하는 포스팅들만 추출한다. 질의 처리에 포함되지 않는 EID를 갖는 포스팅은 필터링하여 제거하고(이 예에서는 <2, G8>이 제거됨), 각 포스팅의 GID에 포함된 레벨과 목표 레벨의 차이를 구한다. 레벨 차이가 0이면 해당 UID를 포함하는 포스팅은 결과에 바로 포함되며, 레벨 차이가 1이상 발생하면 부모 UID를 구하는 계산 공식에 의해 포스팅의 UID로부터 부모 노드의 UID를 구한다. 이때, 부모 UID별로 같은 부모를 갖는 포스팅의 빈도값은 합산하여 저장한다. 이 예에서는 UID 값 11에 대한 누적된 빈도는 6이며, UID 값 12에 대한 빈도는 1이 된다. 이와 같은 상향식 방식으로, 질의 처리기는 목표 레벨 엘리먼트의 대한 빈도를 리프 레벨부터 누적하여 구해내며, 이러한 빈도값은 결과의 랭킹을 위한 가중치로 사용된다.

## 3. 점진적 변경을 지원하기 위해 확장된 인덱싱 기법

BUS 구조는 문서 트리를 완전 트리로 가정하고 UID를 부여하므로 엘리먼트 삽입 또는 k값의 변경과 같은 문서에 대한 갱신이 있을 경우 UID를 재조정 하고 역 리스트도 재구성해야하는 문제점이 있다. 따라서, 본 절에서는 BUS기법을 기반으로 문서에 대한 점진적 갱신을 지원하도록 변경 및 확장된 인덱싱 기법을 제안한다. 먼저, 문서에 대한 갱신은 구조 갱신과 내용 갱신으로 분류할 수 있다. 구조 갱신은 노드의 삽입과 삭제를 의미하며, 내용 갱신은 텍스트 내용에 대한 갱신을 의미한다. 사실 삭제는 내용 갱신의 범주에 포함된다.

먼저 구조 갱신 지원을 위해, 본 제안 방법에서는 문서 트리 표현시 완전 트리를 가정하지 않는다. 즉, 특별한 순서 규칙없이 임의의 UID값을 각 노드에 할당한다. 그러므로, 질의 처리시 부모 노드에 대한 접근을 위해 (1)식과 같은 부모 계산식을 이용할 수 없으므로, 부모 노드에 대한 정보를 유지하기 위해 <표 3>과 같이 각 엘리먼트에 대해 부모 UID를 포함하는 테이블을 유지한다. 또한, 구조 갱신과 내용 갱신을 위해 기존에 사용된 포스팅 구조를 변경하였으며 각 절에서 설명한다.

<표 2> EID 정보 테이블

| 경로 | EID | LEV |
|---|---|---|
| /DOC | 1 | 1 |
| /DOC/TITLE | 2 | 2 |
| /DOC/ABS | 3 | 2 |
| /DOC/CHAP | 4 | 2 |
| /DOC/ABS/PAR | 5 | 3 |
| /DOC/CHAP/SEC | 6 | 3 |
| /DOC/CHAP/SEC/PAR | 7 | 4 |

<표 3> 정보 테이블

| UID | 부모 UID | LEV | ... |
|---|---|---|---|
| 1 | Null | 1 | |
| 2 | 1 | 2 | |
| 3 | 1 | 2 | |
| 4 | 1 | 2 | |
| 8 | 3 | 3 | ... |
| 9 | 3 | 3 | |
| 11 | 4 | 3 | |
| 12 | 4 | 3 | |
| 32 | 11 | 4 | |
| 33 | 11 | 4 | |

### 3.1 구조 갱신의 경우

<그림 2>와 같이 초기에 문서 트리를 구축하고 <표 3>와 같은 정보를 유지하며, <그림 3>과 같은 역 리스트 구조의 인덱스를 생성했다고 하자. 첫째, UID 값이 4인 'CHAP' 엘리먼트의 제일 왼쪽 자식으로 새로운 'SEC' 엘리먼트가 리프 노드로 삽입될 경우, 기존의 BUS 방법에서는 삽입된 노드의 오른쪽 형제들를 루트로하는 서브트리들의 UID값 변경이 요구되며, 이는 역 리스트의 재구성이 요구된다. 특히, k의 값이 변경될 경우는 전체적인 재구성이 요구된다. 그러나 본 논문에서 제안한 기법에서는 새로 삽입된 노드에 대해 임의의 UID, 예를 들어 다음 번 UID값 34를 할당하고 부모 값 4와 함께 <표 3>의 마지막 엔트리로 삽입만 하면 된다. 그리고 삽입된 노드에 대해 포스팅을 생성하여 해당 역 리스트에 삽입하면 된다. 즉, 다른 노드들의 GID에는 아무런 변경이 없으므로 기존 인덱스 구조에 대한 재구성 없이 새로 추가된 노드에 대한 포스팅만 추가하면 된다. 여기서 부모 정보를 각 포스팅에 유지할 수도 있으나 일반적으로 포스팅의 개수가 엘리먼트의 개수보다 크므로 각 엘리먼트에 유지하는 것이 바람직하다.

둘째, 문서 트리구조의 중간에 새로운 노드의 삽입 즉, XML로 문서를 표현할 경우 먼저 DTD에 대한 변경이 있는 경우이다. 이 경우 삽입되는 노드를 새로운 루트로 하는 서브트리의 모든 노드의 레벨 및 UID가 변경되며, 이는 해당 노드에서

생성된 포스팅에 포함된 LEV과 UID값의 갱신이 요구된다. 또한, <표 2>와 같은 엘리먼트 타입 분류 테이블도 새롭게 갱신되어야 한다. 따라서, 본 논문에서는 포스팅 구조에서 LEV 필드를 제거하고, 대신 각 포스팅이 생성된 엘리먼트를 식별하기 위해 <표 3>과 같이 각 엘리먼트마다 LEV 값을 유지하도록 한다. 또한, 엘리먼트 타입 정보에 대한 갱신을 최소화 하기 위해 <표 2>와 같은 절대 경로를 이용한 테이블의 유지의 오버헤드가 클 경우, DataGuide[5]와 유사한 구조적 정보에 대한 요약을 통해 관리할 수도 있다. 예를들어 XML로 문서를 표현한 경우, 이 구조 요약 정보도 <그림 5>와 같이 XML 표현 형태를 사용하여 유지한다. EID와 LEV 값 정보를 유지하기 위해 엘리먼트의 애트리뷰트를 사용할 수 있다. 예를 들어 'CHAP' 엘리먼트와 'SEC' 엘리먼트 사이에 'PAR' 엘리먼트가 삽입될 경우, <그림 5>는 <그림 6>과 같이 변경할 수 있다. 여기서 새롭게 삽입된 엘리먼트는 자신의 레벨에 추가되고 EID값은 이미 사용된 값의 다음 순서의 값을 사용하며 LEV 값은 3이 된다. 또한, 기존의 'SEC'과 'PAR' 엘리먼트의 lev 애트리뷰트의 값은 1씩 증가시키면 되며, <표 3>에서 부모 UID만 변경하면 된다. 이러한 갱신은 기존에 구축된 포스팅 구조에는 영향이 없으며, 일부분에서만 수행되므로 포스팅에 대한 갱신보다 처리 오버헤드를 줄일 수 있다.

```
<DOC eid=1 lev=1>
   <TITLE eid=2 lev=2/>
   <ABS eid=3 lev=2>
      <PAR eid=4 lev=3/>
   </ABS>
   <CHAP eid=5 lev=2>
      <SEC eid=6 lev=3>
         <PAR eid=7 lev=4/>
   ....
</DOC>
```

<그림 5> 구조 요약

```
<DOC eid=1 lev=1>
   <TITLE eid=2 lev=2/>
   <ABS eid=3 lev=2>
      <PAR eid=4 lev=3/>
   </ABS>
   <CHAP eid=5 lev=2>
      <PAR eid=8 lev=3>
         <SEC eid=6 lev=4>
            <PAR eid=7 lev=5/>
   ....
</DOC>
```

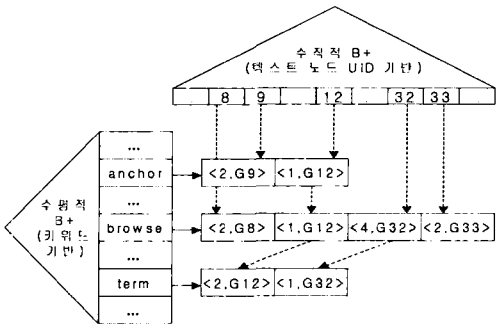
<그림 6> 구조 요약의 변경

질의 처리시, 사용자 질의에 대한 분석은 <그림 5>와 같은 구조 정보를 통해 목표 엘리먼트의 레벨과 질의 처리에 포함될 엘리먼트 타입들을 식별해낼 수 있다. 이 후, <표 3>에서 포스팅 UID와 동일한 값을 갖는 엔트리를 찾아 그 LEV과 부모 UID 값을 찾아 상향 계산 횟수의 결정 및 부모 엘리먼트로 접근한다. 필터링과정은 BUS기법과 동일하다.

### 3.2 내용 갱신의 경우

텍스트 노드에 대한 내용 갱신 또는 노드 삭제의 경우 기존 노드에 대한 포스팅 정보를 역 리스트에서 제거하고 새로 갱신된 내용을 반영해야 한다. 그러나 필요없는 포스팅을 삭제하는 작업은 <그림 3>의 역 리스트상에서 수행할 경우 각 포스팅을 일일이 검색해야하는 큰 부담이 발생한다. 따라서, 이러한 내용 갱신을 지원하기 위해 포스팅 구조를 다음과 같이 변경한다.포스팅 구조에 같은 UID를 갖는 텍스트 엘리먼트에서 생성된 포스팅들을 그룹핑하도록 하기 위해 다음 포스팅을 가리키는 필드를 추가한다. 또한, 같은 텍스트 엘리먼트에서 생성된 포스팅들을 구별하기 위해 기존의 역 리스트 구조에 다른 차원의 B+트리를 유지하여 총 2개의 B+트리로 구성된 역 리스트 구조의 인덱스를 구축한다.

예를 들어, 문서 트리를 순회하며 각 텍스트 노드별로 포스팅을 생성할 때, 같은 UID를 갖는 텍스트 노드에 대해 생성된 포스팅은 추가된 필드를 사용하여 순차적인 연결 리스트 구조로 그룹핑한다. 한편, 새롭게 추가된 또 다른 B+트리 기반의 인덱스(수직적 구조)는 텍스트 엘리먼트 노드의 UID를 기반으로 구축된 인덱스이며 각 리프는 특정 UID 값을 가지게 된다. 각 리프는 특정 UID를 갖는 텍스트 노드에 대한 포스팅 생성시 그룹핑된 포스팅 리스트의 첫 번째 포스팅을 가리키도록 연결한다. 이와 같은 방법으로 한 텍스트 노드에 대한 포스팅 리스트 생성과 수직적 B+트리 구조와의 연결이 되면, 기존과 같은 방법으로 키워드를 기반으로 구축된 B+-트리(수평적 구조) 역 리스트에 각 포스팅을 삽입한다. 이 때, 키워드 별로 분산되어 삽입되는 포스팅들의 연결 구조는 유지된다. <그림 3>에 대한 변형된 이중 B+트리 기반의 역 리스트 인덱스 구조는 <그림 7>과 같다.




<그림 7> 제안된 인덱스 구조의 예

예를 들어 UID 값 32를 갖는 'PAR' 엘리먼트의 내용 갱신의 경우 첫째, <그림 7>의 수직적 인덱스 구조로부터 키, 즉 UID 값이 32인 리프 노드를 검색하여 포스팅을 접근하면 UID값 32인 텍스트 엘리먼트에서 생성된 내용 갱신 이전의 포스팅들의 연결 구조로 접근할 수 있다. 따라서, 이 포스팅 연결 리스트를 순차적으로 접근하여 해당 포스팅들을 삭제하면 된다. 포스팅을 삭제할 경우 계속적으로 수평적 구조상에서 포스팅들의 순서를 유지해야 하도록 링크를 조정해야 한다. 이러한 오버헤드를 줄이기 위해서는 논리적 삭제를 이용할 수 있다. 둘째, 새로 갱신된 내용을 바탕으로 새로운 포스팅을 생성하고 연결하여 그룹핑한 후 수직적 역 인덱스 구조상의 같은 UID를 갖는 리프 노드와 연결하고, 이 포스팅들을 키워드를 기반으로 수평적 B+ 인덱스 구조에 새로 삽입하면 된다.

## 4. 결론 및 향후 연구

본 논문에서는 기존의 BUS 인덱싱 구조에 대해 점진적 갱신을 지원하도록 변경 및 확장하였다. 구조 갱신을 지원하기 위해 완전 트리를 가정하지 않고 임의의 UID를 할당하도록 하였다. 또한, 구조 및 내용 갱신을 위해 포스팅 구조를 변경하고 수직적 차원의 B+트리로 구성된 인덱스 구조를 추가하여 이중 B+-트리로 구성된 역 리스트 구조를 제안하였다.

제안 기법에 대한 효율적 구현 및 적용을 위해서는 유지해야 하는 정보들의 저장 오버헤드와 이중 B+트리 역 리스트 구현시 오버헤드의 축소 등의 실제적인 기술을 적용해야 한다.

### <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. Dongwook Shin et al., "BUS: An Effective Indexing and Retrieval Scheme in Structured Documents", Proc.of the 3rd ACM Int'l Conference on Digital Libraries, 1998
3. Dongwook Shin, "XML Indexing and Retrieval with a Hybrid Storage Model", Knowledge and Information Systems, 2001
4. J. McHugh, J. Widom, S. Abiteboul, Q. Luo, and A. Rajaraman. Indexing Semistructured Data. Technical Report, January 1998.
5. R. Goldman and J. Widom. DataGuides: Enabling Query Formulation and Optimization in Semistructured Databases. Proceedings of the 23rd Int.l Conerence on VLDB 1997.