효율적 XML 키워드 검색을 인덱스 분할 및 합병
(Partitioning and Merging an Index for Efficient XML Keyword Search)

김 성 진 † 이 형 동 † 김 형 주 ††
(Sung Jin Kim) (Hyung Dong Lee) (Hyoung Joo Kim)

요 약 일반적으로 XML 키워드 검색에서 검색 결과는 질의 키워드들을 모두 포함하는 가장 작은 원소(최소 공통 선조)로 정의되며 색인의 기본 단위는 XML 원소가 된다. 기존의 인덱스 구조 하에서는 질의 키워드를 포함한 각 원소의 조합으로 생성된 모든 최소 공통 선조가 검색 결과로 고려된다. 본 논문에서는 - 불필요한 최소 공통 선조 산출 연산을 피하고 검색 시간을 단축시키기 위한 목적으로 - 인덱스를 파티션이라고 불리는 물리적 단위로 분할하고 질의 처리 시 필요에 따라 파티션을 동적으로 합병하여 검색 결과를 산출하는 기법을 기술한다. 주어진 깊이 이상의 최소 공통 선조가 검색 결과로 반환되어야할 경우, 검색 시스템은 제안된 인덱스 구조 하에서 동일 파티션에 속한 원소들 간의 조합만으로 검색 결과를 반환함으로써 검색 시간을 단축시킬 수 있다. 검색 결과에 대한 깊이 제한이 주어지지 않을 경우에도 검색 시스템은 분할된 인덱스를 사용하여 검색 결과를 얻을 수 있으며, 이때 분할되지 않은 기존의 인덱스를 사용하는 검색과 동일한 시간이 소요된다. 실험은 DBLP 사이트와 INEX2003에서 제공되는 XML 문서들로 진행되었으며, 제안된 인덱스는 검색 결과의 최소 깊이가 주어질 경우 질의 처리 시간을 상당히 감소시켰다.

키워드 : XML, XML 키워드 검색, 분할 인덱스

Abstract In XML keyword search, a search result is defined as a set of the smallest elements (i.e., least common ancestors) containing all query keywords and a granularity of indexing is an XML element instead of a document. Under the conventional index structure, all least common ancestors produced by the combination of the elements, each of which contains a query keyword, are considered as a search result. In this paper, to avoid unnecessary operations of producing the least common ancestors and reduce query process time, we describe a way to construct a partitioned index composed of several partitions and produce a search result by merging those partitions if necessary. When a search result is restricted to be composed of the least common ancestors whose depths are higher than a given minimum depth, under the proposed partitioned index structure, search systems can reduce the query process time by considering only combinations of the elements belonging to the same partition. Even though the minimum depth is not given or unknown, search systems can obtain a search result with the partitioned index, which requires the same query process time to obtain the search result with non-partitioned index. Our experiment was conducted with the XML documents provided by the DBLP site and INEX2003, and the partitioned index could reduce a substantial amount of query processing time when the minimum depth is given.

Key words : XML(eXtensible Markup Language), XML Keyword Search, Partitioned Index

· 본 연구는 BK-21 정보기술사업단과 정보통신부 및 정보통신연구진흥원의 대학 IT연구센터 육성·지원 사업(IITA-2006-C1090-0603-0031)의 연구결과로 수행되었음
† 학생회원 : 서울대학교 전기컴퓨터공학부
sjkim@oopsla.snu.ac.kr
hdlee@oopsla.snu.ac.kr
†† 종신회원 : 서울대학교 전기컴퓨터공학부 교수
hjk@oopsla.snu.ac.kr
논문접수 : 2005년 9월 9일
심사완료 : 2006년 10월 10일

1. 서 론

W3C[1]에서 XML(Extensible Markup Language)이 문서 형식의 표준으로 발표된 이후, 많은 문서들이 XML로 표현되고 있으며 더불어 XML 문서 검색의 중요성이 커지고 있다. 키워드 검색[2]은 현재 대부분의 웹 검색 시스템에 도입되어 사용 중이다. XML 문서에 대한 키워드 검색[3-8]은 사용자가 문서의 구조적 정보

나 복잡한 질의 문법을 알지 못하더라도 원하는 정보를 쉽게 검색할 수 있도록 한다. 일반적으로 XML 키워드 검색에서 검색 결과는 질의 키워드를 포함한 XML 원소가 되며[5,8], 사용자는 문서 전체를 검색 결과로 얻는 것에 비해 검색 결과를 효과적으로 이해할 수 있다.

키워드 검색에서는 의미 있는 모든 키워드들에 대한 인덱스가 구축된다. 인덱스는 다양한 구조로 구성될 수 있으나 기본적으로 키워드를 포함한 문서 식별자와 문서 내에서 단어가 나타난 위치 정보들로 구성된다. XML 키워드 검색에서 인덱스는 문서 대신 키워드를 포함한 원소들을 색인한다. XML 문서들은 종종 수천 또는 수백만 개의 원소들을 포함하고 있고[9], 원소 기반의 인덱스는 문서 기반의 인덱스에 비해 색인할 양이 현저히 많아짐에 따라 효율적 인덱스 처리에 대한 연구가 중요하다[10-13].

두개 이상의 키워드가 질의로 입력되었을 때 검색 시스템은 각 키워드의 인덱스에 있는 원소들의 모든 조합을 고려하여 검색 결과를 반환한다. 최악의 경우에 한 인덱스는 나머지 인덱스에 있는 원소 개수만큼 반복적으로 읽혀야 할 수도 있다. [5]는 DIL이라는 인덱스 구조를 제안하고 제안된 인덱스 구조 하에서의 검색 방법을 제안하였다. [5]는 원소 식별자에 따라 원소들을 정렬하고 각 인덱스를 순차적으로 비교하여 검색 결과를 산출하였다. 따라서 하나의 인덱스는 두 번 이상 읽히지 않고 검색 결과를 반환할 수 있도록 하였다. [8]은 XML 원소들을 B+트리 구조로 색인하여 내부 노드들을 메모리에 캐시하고 디스크 접근 횟수를 줄였다.

본 논문에서는 검색 결과가 되는 원소의 최소 깊이를 예상할 수 있을 때(또는 최소 깊이를 일정 수준으로 제한하는 경우에), 검색 시스템이 인덱스를 다수의 파티션들로 분할하고 동일 파티션에 속한 원소들만을 고려하여 최소 깊이 이상의 결과 원소를 검색하는 방법을 기술한다. 예를 들어, DBLP 사이트[14]는 수십만 개의 논문 목록을 하나의 XML 문서에 저장하고, 논문 목록이나 논문 저자를 검색 하는 서비스를 제공한다. 문서 전체를 표현하는 루트 원소(깊이 0)는 <dblp>이고, 그 하위의 원소(깊이 1)로서 하나의 논문을 의미하는 <article>, <phdthesis>, <journal>, <proceedings> 등이 있다. 저자 정보를 포함한 <author> 태그는 깊이 2에 존재한다. 이때 검색 결과 원소의 깊이는 논문일 경우에 1이 되고 저자일 경우에 2가 된다. 따라서 검색 결과의 최소 깊이가 1임을 알 수 있고, 동일 논문에 나타나는 키워드는(또는 동일 저자명에 나타나는 키워드일 경우) 동일 파티션에 할당하여 검색 속도를 향상시킬 수 있다.

질의 처리 시에 분할 인덱스를 사용하면 다음과 같은 이점을 얻을 수 있다. 첫째, 검색 결과 여부를 판단하기 위해 비교되는 원소들 간의 조합 연산을 줄일 수 있다. 검색 시스템은 질의 키워드를 포함한 원소와 또 다른 질의 키워드를 포함하고 있는 원소들 사이의 모든 조합을 고려하지 않고 동일 파티션에 속한 원소들 간의 조합만 고려하여 검색 결과를 산출할 수 있다. 둘째, 인덱스를 읽는 데에 필요한 입/출력의 양을 줄일 수 있다. 한 키워드의 인덱스에서 임의의 파티션에 속한 원소가 하나도 없을 경우 다른 키워드의 동일 파티션에 속한 원소들은 읽히지 않는다. 셋째, 분할 인덱스는 다양한 인덱스 구조 및 질의 처리 알고리즘에 적용되어 사용될 수 있으며, 최악의 경우에도 분할 인덱스 적용으로 인해 검색 결과 산출을 위한 조합 연산이 증가하지 않는다. 예를 들어, 검색 시스템은 인덱스를 DIL 구조의 파티션들로 나누고 동일 파티션에 속한 원소들로부터 검색 결과를 [4]의 방법으로 구할 수 있다. 또한 최악의 경우에도 각각의 동일 파티션에서 발생하는 조합 연산 횟수의 합이 분할되지 않은 DIL 구조의 인덱스로부터 발생하는 조합 연산 횟수보다 많지 않다.

검색 결과의 깊이가 명확하지 않을 경우에, 검색 시스템은 문서 관리자가 주관적으로 예측한 결과 깊이에 기반을 두어 분할 인덱스를 구축하고 사용자에게 결과 깊이에 대한 선택권을 줄 수 있어야 한다. 예를 들어 INEX2003의 XML 문서는 컴퓨터 분야의 논문 정보를 저장하고 있다. 문서 구조를 살펴보면 논문지 정보와 논문지에 게재된 논문들을 저장하는 <journal> 원소가 있고 그 하위 원소로 논문 한편의 정보를 저장한 <article> 원소가 있다. <article> 원소에는 초록을 저장한 <fm> 원소와 논문의 본문을 저장한 <bdy> 원소가 있다. 또한 <bdy> 원소는 섹션(section)을 저장하는 <sec> 원소로 구성되며, 각 섹션은 문단을 저장하는 <p> 원소를 포함한다. 이때 관리자가 사용자의 검색 단위를 주로 논문 내용(즉 <bdy> 원소) 이하로 예측한다면, 검색 시스템은 논문 내용 수준에서 분할 인덱스를 구축한 뒤 보다 구체적 단위의 검색(문단 또는 섹션)이나 포괄적 단위의 검색(논문 또는 논문지)을 허용하는 옵션(option)을 사용자에게 제공할 수 있어야 한다.

본 논문에서는 관리자가 정한 결과 깊이보다 낮은 깊이의 검색 결과가 사용자에 의해 요구될 경우 이미 구축된 분할 인덱스를 이용하여 사용자 요구에 맞는 가상의 분할 인덱스를 동적으로 재구성하는 방법을 기술한다. 즉 사용자가 원하는 결과 깊이 수준의 분할 인덱스를 물리적으로 재구성하지 않으면서 검색 결과를 반환할 수 있다. 반대로 사용자가 요구하는 검색 결과의 깊이가 관리자가 정한 깊이보다 깊을 경우, 검색 시스템은 이미 생성된 분할 인덱스를 이용하여 검색 결과를 반환할 수 있다.

본 논문의 구성은 다음과 같다. 2장에서는 분할 인덱스 설계에 필요한 설계 전략들을 기술한다. 3장에서는 분할 함수를 소개하고 분할 함수를 이용하여 인덱스를 분할하고 질의를 처리하는 방법을 기술한다. 4장에서는 실험을 통하여 분할 인덱스의 사용에 대한 효율성을 검증한다. 5장에서는 결론을 맺고 향후 계획을 기술한다.

2. 분할 인덱스의 개념 및 요구 사항

본 장에서는 분할 인덱스를 위한 용어들을 정의하고 설계 전략을 기술한다. 그림 1은 논문 목록 정보를 포함하고 있는 예제 XML 문서이다. XML 문서는 XML 원소(element)들로 구성되며, 각 원소는 속성(attribute), 속성 값(attribute value), 데이타 값을 포함한다. 예제 XML 문서에서 <data> 원소는 세 개의 <collection> 원소들을 포함하고 있다. 첫 번째 <collection> 원소는 네 개의 <paper> 원소를 포함하고, 두 번째 <collection> 원소는 한 개의 <paper> 원소를 포함하며, 마지막 <collection> 원소는 어떠한 원소도 포함하지 않는다. 하나의 원소가 다른 원소를 포함할 때 전자를 부모 원소라고 하고 후자를 자식 원소라고 한다.

XML 문서는 트리 구조로 모델링되며, XML의 원소들은 트리의 노드가 된다. 속성은 속성을 포함한 원소의 자식 원소로 간주될 수 있으나, 본 논문에서는 설명의 간편함을 위하여 속성이 고려되지 않는다. 그림 2는 그림 1의 XML 문서를 트리로 표현한 예이다. 그림 2에서 자식 노드와 부모 노드는 반직선으로 표현되고, 데이타 값은 데이타를 직접 포함하고 있는 원소와 선분으로 연결되었다. 괄호 안에는 노드의 식별자가 표기되어 있다.

```
<data>
    <collection no="1">
        <paper no="1">    <author> Y. Wu </author>
                          <title> Using Histograms to Estimate Answer Sizes For XML Queries
                          </title>
        </paper>
        <paper no="2">    <author> A. Schmidt </author>
                          <title> Priority in DBMS resource scheduling </title>
        </paper>
        <paper no="3">    <author> L. Mignet </author>
                          <title> The XML Web: a first study </title>
        </paper>
        <paper no="4">    <author> S. Cohen </author>
                          <title> A Semantic Search Engine for XML </title>
        </paper>
    </collection>
    <collection no="2">
        <paper no="1">    <author> A. Schmidt </author>
                          <title> Why and How to Benchmark XML Databases </title>
        </paper>
    </collection>
    <collection no="3">
    </collection>
</data>
```

그림 1 예제 XML 문서

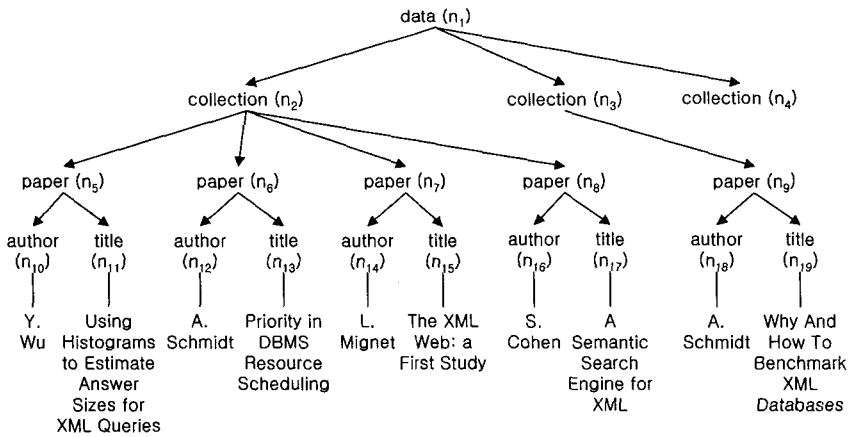


그림 2 XML 트리의 예

루트 노드(n_1)의 깊이는 0이며, 부모 노드의 깊이는 자식 노드의 깊이보다 하나가 더 크다.

다수의 키워드들이 하나의 질의로 입력될 경우 접속적(conjunctive) 의미와 분리적(disjunctive) 의미의 해석이 가능하다. 접속적 의미의 해석에서는 질의 키워드를 모두 포함한 원소가 검색 결과로 반환되고, 분리적 의미의 해석에서는 질의 키워드들 중 하나 이상의 키워드를 포함한 원소가 검색 결과로 반환된다. 본 논문에서는 접속적 의미로 질의를 해석한다.

정의 1. XML 문서의 "유효 결과 깊이"는 검색 결과로 반환될 수 있는 원소들의 최소 깊이이다.

그림 2에서 유효 결과 깊이가 0이라면 모든 원소들이 검색 결과로 반환될 수 있음을 의미한다. 유효 결과 깊이가 2라면 <paper>, <author>, <title>과 같이 깊이 2 이상의 원소들이 검색 결과로 반환될 수 있다.

정의 2. 두 개의 질의 키워드 x와 y를 고려하자. XML 트리에서 노드 n_x는 질의 키워드 x를 포함한 노드이고 n_y는 질의 키워드 y를 포함한 한 노드라고 하자. n_x와 n_y의 최소 공통 선조(두 노드의 공통 선조 중 가장 깊이가 큰 노드)의 깊이가 유효 결과 깊이보다 크거나 같으면, 이때의 최소 공통 선조를 n_x와 n_y의 "유효 최소 공통 선조"라고 한다. n_x와 n_y는 유효 최소 공통 선조를 가진다고 한다.

그림 2에서 유효 결과 깊이가 2이고, 질의 키워드가 "XML"과 "Schmidt"라고 하자. 노드 n_{18}과 n_{19}의 최소 공통 선조는 n_9이다. n_9는 깊이가 2이므로 유효 최소 공통 선조가 된다. n_{11}과 n_{12}의 최소 공통 선조는 n_2이다. n_2의 깊이는 유효 결과 깊이보다 작으므로 유효 최소 공통 선조가 아니다.

분할 함수는 $PF_d(n)$으로 표기된다. 분할 함수의 입력은 문서의 유효 결과 깊이 d와 노드 n이고, 입력된 노드가 할당될 파티션 번호를 반환한다. 키워드 x와 y의 인덱스가 각각 M개의 파티션 ${}_xP_0$, ${}_xP_1$, ${}_xP_2$, ..., ${}_xP_{M-1}$과 파티션 ${}_yP_0$, ${}_yP_1$, ${}_yP_2$, ..., ${}_yP_{M-1}$로 구성된다면, $PF_d(n)$은 다음의 조건을 만족하도록 인덱스를 분할하여야 한다.

분할 요구 사항 :

(1) $0 \le i < M$, $0 \le j < M$, $i \neq j$ 일 때, ${}_xP_i \cap {}_xP_j = \{\}$.

(2) $n_x \in {}_xP_i$ $(0 \le i < M)$ 와 $n_y \in {}_yP_j$ $(0 \le j < M)$일 때, n_x와 n_y가 유효 최소 공통 선조를 가지면 $i = j$ 이다.

분할 요구 사항은 "유효 최소 공통 선조"를 얻기 위해 동일 파티션에 속한 원소들만을 비교하여 검색 결과를 산출할 수 있도록 한다. 노드 n_x는 키워드 y의 인덱스에 포함된 모든 노드와 비교될 필요 없이 n_x와 동일 파티션에 속한 노드들만을 고려하여 검색 결과를 얻을 수 있다. 예를 들어, 키워드 "XML"을 포함한 n_{19}와 키워드 "Schmidt"을 포함한 n_{18}은 "유효 최소 공통 선조"를 가지므로 동일 파티션에 할당된다.

임의의 XML 문서에서 기존의 유효 결과 깊이 보다 작은 깊이의 최소 공통 선조들을 결과원소로 반환하기 위하여, 유효 결과 깊이를 d에서 $d'(d > d')$로 바꾸었다고 하자. 두 노드의 최소 공통 선조의 깊이가 d보다 작고 d'보다 클 때, 두 노드는 유효 최소 공통 선조를 가지나 d에 대하여 물리적으로 구축된 분할 인덱스에서 동일 파티션에 존재하는 것이 보장되지 않는다. 키워드 x의 인덱스가 d'에서 분할 요구 사항을 만족하는 VM개의 파티션 ${}_xVP_0$, ${}_xVP_1$, ${}_xVP_2$, ..., ${}_xVP_{VM-1}$들로 구성된다고 하자. d'에 대한 인덱스를 물리적으로 생성하지 않고 d에 대하여 구축된 분할 인덱스로부터 유효 최소 공통 선조를 반환하고자할 때, 분할 함수가 다음의 합병 요구 사항을 만족시키도록 한다.

합병 요구 사항 :

(1) ${}_xP_0 \cup {}_xP_1 \cup \ldots \cup {}_xP_{M-1} = {}_xVP_0 \cup {}_xVP_1 \cup \ldots \cup {}_xVP_{VM-1}$.

(2) $\forall n \in {}_xP_i$ $(0 \le i < M)$, $n \in {}_xVP_j$ $(0 \le j < VM)$.

합병 요구 사항 (1)은 유효 결과 깊이 d에 대해 생성된 인덱스와 d'에 대해 생성된 인덱스에 포함되어있는 노드의 개수가 같도록 요구한다. 즉 유효 결과 깊이 d'에 대한 검색 결과가 되는 노드들이 유효 결과 깊이 d에 대한 인덱스에 포함되도록 한다. 따라서 d에 대한 인덱스만을 참조하여 d'에 대한 검색 결과를 반환할 수 있도록 한다. d'의 최소값은 0이므로, d에 대한 키워드 x의 인덱스는 x를 포함한 모든 노드들을(유효 결과 깊이에 관계없이) 색인하고 있어야 한다. 합병 요구 사항 (2)는 하나의 물리적 파티션에 속한 모든 노드들이 d'에 대한 인덱스에서 하나의 가상 파티션에 속하도록 요구한다. 물리적으로 동일한 파티션에 속한 두 노드가 서로 다른 가상의 파티션에 속하는 것은 두 개의 가상 파티션을 읽을 때 물리적 파티션 하나가 두 번 읽힐 수 있음을 의미한다. 합병 요구 사항 (2)는 전체의 가상 파티션들을 읽기 위하여 하나의 물리적 파티션이 한번만 읽히도록 한다.

3. 분할 인덱스의 생성 및 활용

본 장에서는 설계 전략에 나타난 분할 요구 사항과 합병 요구 사항을 만족하는 분할 함수를 제안하고 분할 인덱스의 생성 과정과 질의 처리 과정을 기술한다.

3.1 분할 함수

분할 요구 사항을 만족시키는 함수 작성의 기본 아이디어는 다음과 같다. 유효 결과 깊이 d와 동일한 깊이

를 가지는 서로 다른 두 노드 n_α와 n_β가 있다고 하자. n_α와 n_β의 모든 공통 선조들의 깊이는 유효 결과 깊이보다 작으므로(즉 유효 최소 공통 선조를 가지지 않으므로), n_α의 후손 노드들과 n_β의 후손 노드들을 동일한 파티션에 할당할 필요가 없다. 한편 n_α의 후손 노드들 간에는 서로 유효 최소 공통 선조를 가질 수 있으므로 동일한 파티션에 할당한다. 마찬가지로 n_β의 후손 노드들도 서로 동일한 파티션에 할당한다. 이와 같은 방법으로 검색 결과가 될 가능성이 있는 조합의 노드들을 동일 파티션에 할당하고 가능성이 없는 조합의 노드들을 서로 다른 파티션에 할당하여 분할 요구 사항을 만족하는 분할 인덱스를 생성할 수가 있다.

가장 기본적인 인덱스의 분할 방법은 유효 결과 깊이에 있는 노드 개수만큼 파티션을 만들고 각 노드와 그 후손 노드들을 동일 파티션에 할당할 수 있다. 이러한 분할 방법은 노드의 팬-아웃(fan-out)이 커짐에 따라 시스템이 다루어야할 파티션 개수가 많아지고, 유효 결과 깊이의 변화에 따른 가상 파티션의 구성이 용이하지 않은 단점이 있다.

분할 요구 사항은 유효 최소 공통 선조를 가지지 않는 두 노드가 반드시 서로 다른 파티션에 할당 되도록 강요하지 않는다. 따라서 일정 수준의 파티션 개수를 유지하기 위해 n_α와 n_β의 후손 노드가 동일 파티션에 할당되는 것을 부분적으로 허용한다. 파티션 요소 δ는 유효 결과 깊이에 있는 형제 노드들에 허용되는 최대 파티션 개수이다. δ가 작아지면 전체 파티션의 개수는 줄일 수 있으나 n_α와 n_β가(또한 n_α와 n_β의 후손 노드들이) 동일 파티션에 속할 확률이 높아진다. δ가 커지면 전체 파티션의 개수는 늘어나지만 n_α와 n_β가 동일 파티션에 속할 확률이 낮아진다.

파티션 요소 δ에 따른 전체 파티션의 개수는 δ^d가 이다. 유효 결과 깊이 0은 루트 노드와 루트 노드의 모든 후손들을 동일한 파티션에 할당하는 것을 의미하므로, 전체 파티션의 개수는 1이 된다. 깊이 1의 형제 노드들은 루트 노드를 부모로 하는 형제 노드들만이 존재하므로, 유효 결과 깊이가 1일 때 전체 파티션의 개수는 δ가 된다. 깊이 1의 형제 노드들의 개수가 δ보다 작을 때 노드를 포함하지 않는 빈 파티션이 생성될 수 있다. 유효 결과 깊이가 2일 때, 깊이 1의 각 노드들은 자신의 자식 노드들을 δ개의 파티션으로 나누어 할당한다. 깊이 1의 노드들의 개수가 δ 보다 많아서 깊이 1의 두 노드가 동일 파티션에 속하였을 경우는 두 노드가 속한 파티션을 δ개로 나누어 노드들을 할당하는 의미가 된다. 따라서 유효 결과 깊이가 2일 때, 전체 파티션의 개수는 δ^2가 된다.

유효 결과 깊이가 d에서 d'으로 작아졌을 때, 깊이가 d인 형제 노드 n_α와 n_β가 있고, n_α와 n_β를 후손 노드로 가지는 깊이 d'의 n_ψ가 있다고 하자. 또한, 깊이 d에 대해 물리적으로 구축된 인덱스가 n_α와 그 후손 노드들을 하나의 파티션으로 구성하고 n_β와 그 후손 노드들을 또 다른 파티션으로 구성하고 있다고 하자. 그러면 d에 대하여 물리적으로 구축된 인덱스는 분할 요구 사항을 만족한다. d'에 대한 분할 인덱스에서 n_α, n_β, n_ψ을 모두 동일 파티션에 속하도록 하면(즉, n_ψ와 그 후손 노드들을 하나의 파티션에 포함되도록 하면), d'에 대한 인덱스도 분할 요구 사항을 만족시킨다. n_ψ는 유효 결과 깊이 d에 대한 검색 결과가 되지 않지만 d에 대해 물리적으로 구축된 인덱스에서 n_α나 n_β가 속한 파티션 중의 하나에 포함되도록 하면, 합병 요구 사항 (1)을 만족시킨다. 즉, d에 대한 인덱스에서 하나의 물리적 파티션은 n_α(또는 n_β), n_α(또는 n_β)의 후손 노드, 경우에 따라 n_ψ를 포함 할 수 있다. d'에 대한 인덱스에서 n_ψ와 그의 모든 후손 노드들이 동일 파티션에 있고, d에 대한 인덱스에서 n_α(또는 n_β)와 그의 후손 노드들이 모두 동일 파티션에 있다는 것은, d에 대한 인덱스에서 하나의 파티션에 있는 모든 노드들이 d'에 대한 인덱스의 한 파티션에 속하는 것을 의미하므로 합병 요구 사항 (2)를 만족시킨다. 즉, d'에 대한 가상 인덱스에서 n_ψ가 속한 가상의 인덱스를 읽는 것은 자신의 후손 노드들이 포함된 물리적 파티션만 읽으면 되고, 모든 가상 파티션을 읽는 동안 하나의 물리적 파티션은 두 번 이상 읽히지 않는다.

OR(n)은 노드 n의 형제 노드들 사이에서 n의 순서를 반환한다. 순서는 0부터 시작한다. 예를 들어, 그림 2에서 노드 n_8은 n_2를 부모 노드로 하는 형제 노드들(n5, n6, n7, n_8) 사이에서 네 번째에 위치하므로, OR(n_8)은 3이다.

함수 $OR_d(n)$은 수식 (1)과 같이 정의된다. $A_d(n)$은 n의 깊이가 d보다 큰 노드에 대해 정의되며 깊이가 d인 n의 선조 노드를 반환한다. 예를 들어, $A_2(n_{14})$은 n_7이다. 따라서 $OR_2(n_{14})$의 값은 2가 된다. $D_d(n)$은 n의 깊이가 d보다 작은 노드에 대해 정의되며 깊이 d인 n의 후손 노드들 중 첫 번째 후손 노드를 반환한다. 예를 들어, $D_2(n_2)$는 n_5이다. 즉 $OR_2(n_2)$은 0이다. 깊이 d의 후손 노드가 존재하지 않을 경우에는 깊이가 d인 첫 번째 후손 노드가 존재한다고 가정하여 값을 반환한다. 예를 들어, $OR_2(n_4)$는 0이다. $OR_d(n)$은 n의 깊이가 d보다 작을 경우 깊이가 d인 후손 노드의 존재 여부에 상관없이 항상 0을 반환한다.

$$
\begin{aligned}
OR_d(n) &= OR(A_d(n)) && (n\text{의 깊이} > d)\\
&= OR(n) && (n\text{의 깊이} = d)\\
&= OR(D_d(n))=0 && (n\text{의 깊이} < d)
\end{aligned}
\qquad (1)
$$

분할 함수 $PF_d(n)$은 유효 결과 깊이 d에 대하여 노드 n이 할당될 파티션 번호를 반환하며, 수식 (2)와 같이 정의된다. 노드 n의 깊이가 d보다 크면, $OR_d(n)$은 깊이 d인 n의 선조 노드와 동일한 함수 값을 가지도록 하여 분할 요구 사항을 만족시킨다. 즉, 깊이가 d인 한 노드와 그 후손 노드들은 모두 동일한 분할 함수 값을 할당받는다. $(OR_d(n) \bmod \delta)$은 동일한 부모를 가지는 깊이 d의 형제 노드들이 δ개의 파티션에 할당되도록 한다.

$$PF_d(n) = (OR_d(n) \bmod \delta) + (PF_{d-1}(n) * \delta) \quad (d > 0 \text{ 일 경우}) \qquad (2)$$
$$= 0 \quad (d = 0 \text{ 일 경우})$$

분할 함수 $PF_d(n)$은 깊이가 d보다 작은 모든 노드들을 깊이 d에 있는 첫 번째 후손 노드와 동일한 파티션에 속하게 하므로 합병 요구 사항 (1)을 만족시킨다. 즉, $OR_d(n)$에 의해서 깊이가 d인 n의 후손 노드 중 첫 번째 후손 노드가 포함될 파티션에 n이 할당된다. 분할 함수 $PF_d(n)$은 유효 결과 깊이가 하나 줄어들 때마다 유효 결과 깊이의 형제 노드들이 포함된 δ개의 파티션들을 하나의 파티션으로 합병하고, 유효 결과 깊이가 하나 증가할 때마다 깊이가 d인 각 형제 노드들에 할당된 파티션이 δ개의 파티션으로 분할하므로 합병 요구 사항 (2)를 만족한다. 깊이 d의 노드 n_α와 n_β가 파티션 m과 파티션 n에 속하고, $OR(n_\alpha) + 1 = OR(n_\beta)$라고 할 때, 유효 결과 깊이가 하나 증가되었다고 하자. 파티션 번호가 m보다 작은 모든 파티션들이 각각 δ개의 파티션으로 나누어진다. 파티션 m은 파티션 번호가 $((m-1)*\delta)$, $((m-1)*\delta + 1)$, ..., $((m-1)*\delta + (\delta-1))$인 δ개의 파티션들로 나누어진다. 파티션 n은 파티션 번호가 $(m*\delta)$, $(m*\delta + 1)$, ..., $(m*\delta + (\delta-1))$인 δ개의 파티션으로 나누어진다.

유효 결과 깊이 d가 1이상일 때 분할 함수는 수식 (3)과 같이 표현이 가능하다.

$$PF_1(n) = (OR_1(n) \bmod \delta)$$
$$PF_2(n) = (OR_2(n) \bmod \delta) + (OR_1(n) \bmod \delta) * \delta$$
$$PF_3(n) = (OR_3(n) \bmod \delta) + (OR_2(n) \bmod \delta) * \delta + (OR_1(n) \bmod \delta) * \delta^2$$
$$\ldots = \ldots$$
$$PF_d(n) = \sum_{i=1}^{d} (OR_i(n) \bmod \delta) * \delta^{d-i} \qquad (3)$$

그림 2의 XML 트리를 고려하자. 유효 결과 깊이 d와 모듈러 δ에 따른 각 노드들의 분할 함수 결과는 그림 3에 나타나 있다. $\delta = 1$ 이거나 $d = 0$일 때는 한 문서에 대해 하나의 파티션(첫 번째 파티션)만을 가지므로, 모든 원소들은 분할 함수 값으로 0을 가진다.

δ가 3이고 d가 2인 경우를 보자. 깊이 2에서 n_2를 부모 노드로 하는 형제 노드 n_5, n_6, n_7, n_8와 n_3을 부모로 하는 노드 n_9가 있고, n_4를 부모 노드로 하는 노드는 없다. 각 형제 노드들에 세 개의 파티션이 허락되었으므로, n_5는 첫 번째 파티션(0번 파티션), n_6은 두 번째 파티션, n_7은 세 번째 파티션에 할당되고, n_8은 n_5와 동일한 첫 번째 파티션에 할당된다. 즉 $PF_2(n_5) = PF_2(n_8) = 0$, $PF_2(n_6) = 1$, $PF_2(n_7) = 2$이다. n_3을 부모로 하는 노드는 n_9 하나가 존재하며 네 번째 파티션에 할당된다. 다섯 번째 파티션과 여섯 번째 파티션은 할당된 노드가 없는 빈 파티션이 된다. 또한 n_4를 부모로 하는 파티션

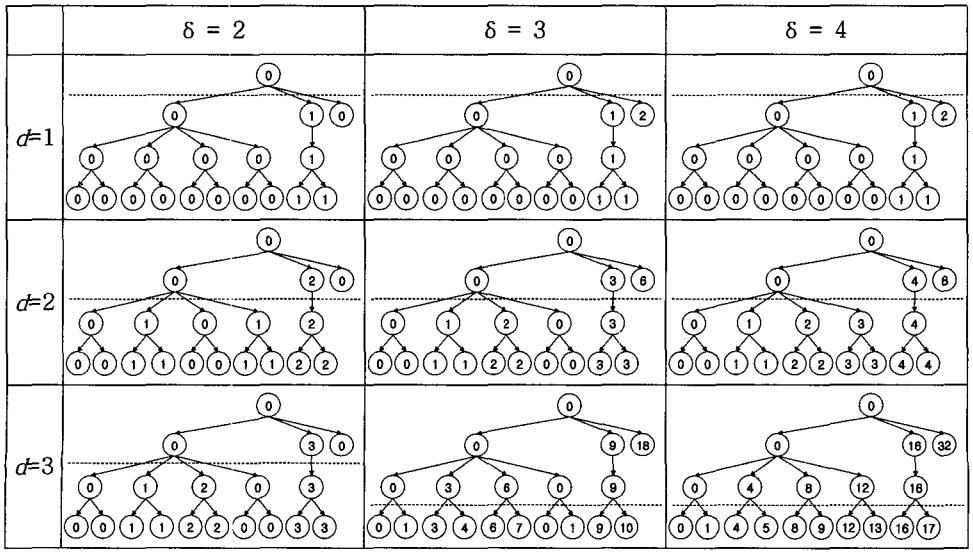


그림 3 분할 함수의 적용 예

이 세 개가 할당되었으나 n_4를 부모로 하는 형제 노드들이 없다. n_4는 깊이 2인 자신의 첫 번째 후손 노드가 속할 파티션 6에 할당되고, 파티션 7과 8은 빈 파티션이 된다.

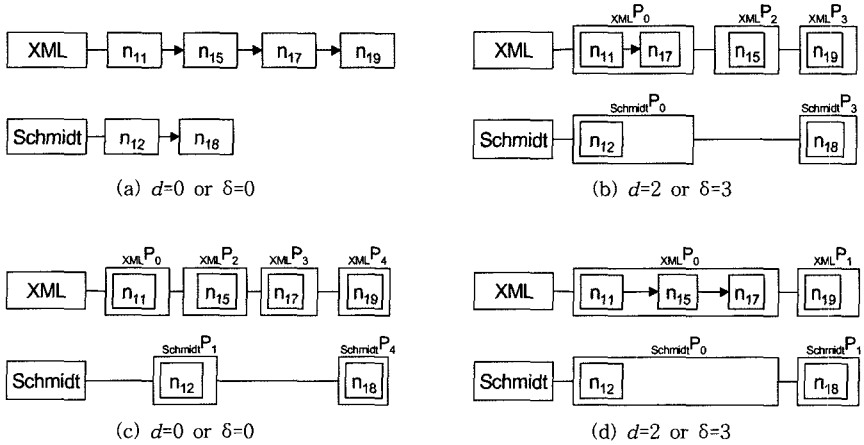
유효 결과 깊이 d = 2에 대하여 구축된 분할 인덱스로부터 1 이상의 깊이를 가지는 원소를 검색 결과로 반환하기 위하여 n_2가 n_5와 동일한 파티션에 할당되어 있다. d = 1에 대한 가상 인덱스를 구축할 때, d = 2에서의 인덱스에서 δ = 3개의 파티션들(파티션 0, 1, 2)이 하나의 가상 파티션으로 구성된다. $VP_0 = P_0 \cup P_1 \cup P_2$가 된다.

그림 4에는 키워드 "XML"과 "Schmidt"에 대한 분할 인덱스가 나타나 있다. 빈 파티션들은 그림에서 생략되었다. (a)는 하나의 파티션으로 구성된 인덱스로서 인덱스가 분할되지 않은 것과 같다. 그림 4(b)는 d가 2이고 δ가 3일 경우의 인덱스를 보인다. d가 2이므로 각 키워드에 대한 분할 인덱스는 9개의 파티션으로 구성된다. "XML" 키워드에 대한 인덱스는 첫 번째, 세 번째, 네 번째 파티션에 각각 2개, 1개, 1개의 노드를 포함하고 있으며, 나머지 파티션에는 할당된 노드가 없다. "Schmidt" 키워드에 대한 인덱스는 첫 번째와 네 번째 파티션에 각각 한 개의 노드를 포함하고 있다. 질의가 "Schmidt"와 "XML"이 주어졌을 때, 검색 시스템은 검색 결과를 얻기 위하여 8개의 조합을 고려하지 않아도 된다. 분할 인덱스를 사용하는 경우에는 세 개의 조합 - n_{11}과 n_{12}, n_{17}과 n_{12}, n_{19}와 n_{18} - 만이 유효 최소 공통 선조를 찾는 데에 고려된다. 키워드 "XML"의 인덱스에서 파티션 2에 해당하는 원소가 있더라도 파티션 2에 해당하는 원소는 "Schmidt"에 해당하는 원소와 유효 최소 공통 선조를 가지지 않음을 알 수 있으므로 "XML" 키워드의 파티션 2를 읽을 필요가 없다.

그림 4(c)는 유효 결과 깊이가 2일 때 파티션 요소 δ가 4인 경우의 분할 인덱스를 나타낸다. 분할 인덱스는 총 16개의 파티션으로 분할된다. 이중 파티션 4에서 두 키워드의 인덱스에 노드가 상호 존재하므로, 한 개의 노드 조합(n_{19}와 n_{18})에 대해 유효 최소 공통 선조의 유무가 검사된다. 그림 4(b)와 같이 구성된 분할 인덱스로부터 깊이가 1 이상인 최소 공통 선조를 얻고자 할 때(즉, 유효 결과 깊이를 1로 조정), 파티션 0, 1, 2가 그림 4(d)에서 0번 파티션을 구성한다고 하고, 파티션 3, 4, 5가 그림 4(d)의 1번 파티션을 구성한다고 가정하여, 물리적으로 그림 4(d)와 같은 인덱스를 구축한 것과 동일하게 검색 결과를 산출할 수 있다.

3.2 분할 인덱스를 이용한 검색

분할 인덱스를 이용한 검색 알고리즘은 그림 5에 나타나 있다. 두 번째 라인에서에서는 각 키워드의 분할 인덱스 헤더를 적재하기 위한 2차원 배열 pheaders을 구축한다. pheaders는 키워드 x의 i번째 파티션에 속한 노드들이 물리적으로 저장된 주소를 가리키는 포인터를 저장한다. i번째 파티션에 속한 노드들이 없을 경우에는 널(NULL)이 할당된다. 세 번째 라인에서 헤더의 내용을 메모리에 적재한다. 다섯 번째 라인에 나타난 "check_empty_partitions" 함수는 각 키워드의 i번째 파티션이 빈 파티션인가를 확인하고 존재하면 "참"을 반환한다. 어떤 인덱스의 i번째 파티션이 비어있으면, 다른 키워드의 i번째 파티션에 속한 노드가 있다고 하더라도 "유효 최소 공통 선조"를 이룰 수 없으므로 읽지 않는다. 반대로 모든 질의 키워드들의 i번째 파티션이 비어있지 않을 경우에 각 파티션들로부터 "유효 최소 공통 선조"를 찾는다. 여덟 번째 라인의 "find_results"는 각 질의 키워



(a) d=0 or δ=0 (b) d=2 or δ=3

(c) d=0 or δ=0 (d) d=2 or δ=3

그림 4 키워드 "XML"과 "Schmidt"의 분할 인덱스

```
1 : // M: Number of partitions, K: Number of query keywords
2 : Declare pheaders [K][M];
3 : pheaders = load_pheaders(keywords);
4 : for i = 0 to M
5 :     if check_empty_partitions (pheaders, i) then
6 :         continue;
7 :     else
8 :         find_results_in_partitions (pheaders, i, d);
9 : end loop
```

그림 5 분할 인덱스를 이용한 검색 I

드들의 i번째 파티션들을 입력으로 하여, 검색 결과를 산출하는 함수이다. 각 파티션은 하나의 인덱스로 볼 수 있기 때문에 검색 결과를 산출하는 방법은 "최소 공통 선조"를 찾는 기존의 알고리즘들[5,8]이 사용될 수 있다.

d에 대하여 분할된 인덱스로부터 d보다 낮은 깊이의 검색 결과를 얻는 방법은 그림 6에 기술되어 있다. N은 하나의 가상 파티션을 이루기 위해 합병되어야 하는 물리적 파티션의 개수이다. 유효 검색 결과가 d에서 d'로 작아질 때, N은 $\delta^{(d-d')}$가 된다. 네 번째 라인에서 vpheaders는 하나의 가상 파티션을 구성하기 위한 물리적 파티션들의 주소를 저장하기 위해 정의된다. 일곱 번째 라인에서 함수 "get_partitions_to_merge" 함수는 i번째 가상 파티션을 구성하기 위한 물리적 파티션들의 주소를 vpheaders에 반환한다. vpheaders는 "check_empty_virtual_partitions"는 각 키워드의 i번째 가상의 파티션에 하나라도 빈 가상 파티션이 존재하면 참을 반환한다. 하나의 가상 파티션이 비어있다는 것은 가상 파티션을 구성하는 모든 물리적 파티션이 비어있음을 의미한다. "find_results_in_virtual_partitions"은 동일한 가상 파티션들에 속한 원소들만을 고려하여 검색 결과를 산출하는 함수이다.

4. 실험 및 평가

본 실험은 제안된 분할 인덱스의 구현 가능성을 보이고, 문서의 최소 깊이를 알 때 분할 인덱스를 사용하여 나타나는 성능 향상을 보이는 것을 목적으로 한다. 본 실험을 위하여 DBLP[14]에서 사용되는 XML 문서와 INEX2003 테스팅 셋[15]의 XML 문서들이 사용되었다. DBLP는 수십만 개의 논문 목록을 하나의 XML 문서에 저장하고 논문 제목이나 논문 저자를 키워드로 검색하는 서비스를 제공하고 있다. INEX2003 테스팅 셋은 XML 검색의 성능 평가를 위한 XML 문서들과 질의들이 정의되어 있다.

우리는 C++로 XML 검색 시스템을 구현하였으며, 인덱스는 버클리(Berkely) 데이타베이스[16]를 사용하여 저장하였다. 분할 함수는 $PF_d(n)$가 사용되었다. 우리는 각각의 파티션을 직접 접근(access)하기 위하여 인덱스의 시작에 각 파티션들의 주소 정보를 저장하는 헤더를 두었다. 각 파티션들은 DIL 인덱스 구조[5]로 구성되었다. 인덱스를 다수의 파티션들로 나눈 후 동일 파티션에 속한 원소들만으로 DIL 인덱스를 구성하였다.

실험에 사용된 DBLP의 XML 문서는 약 50만개의 논문 제목의 목록을 포함하고 있다. XML 문서의 크기는 약 210 메가바이트(mega bytes)이다. XML문서에 대한 트리는 최대 깊이가 2이고 유효 결과 깊이는 1이다. 실험 질의는 4개의 키워드 "computer", "system", "architecture", "2002"로 하였다. 그림 7은 유효 결과 깊이 1에 대하여 분할되지 않은 인덱스와 파티션 요소 δ의 값이 1,000, 5,000, 10,000인 분할 인덱스를 이용하여 얻어진 실험 결과이다. 유효 결과 깊이 1에 대한 분할 인덱스의 파티션 수는 파티션 요소의 크기와 같다. 분할되지 않은 기존의 인덱스를 사용하는 것에 비해 파티션 요소가 5,000인 분할 인덱스를 사용하는 경우에는 25%, 파티션 요소 10,000의 분할 인덱스를 사용하는 경우에는 45%의 응답 시간 감소가 나타났다. 파티션 요소 1,000의 분할 인덱스를 사용하는 경우 응답 시간의 감소가 거의(2%) 나타나지 않았다.

```
1  : // M: number of partitions, K: number of query keywords
2  : // N: number of partitions to merge, VM: number of virtual partitions
3  : Declare pheaders[K][M];
4  : Declare vpheaders[K][N];
5  : pheaders = load_pheaders (keywords);
6  : for i = 0 to VM
7  :     vpheaders = get_partitions_to_merge(pheaders, i);
8  :     if check_empty_virtual_partitions (vpheaders) then
9  :         continue;
10 :     else
11 :         find_results_in_virtual_partitions (vpheaders, d);
12 : End loop;
```

그림 6 분할 인덱스를 이용한 검색 II

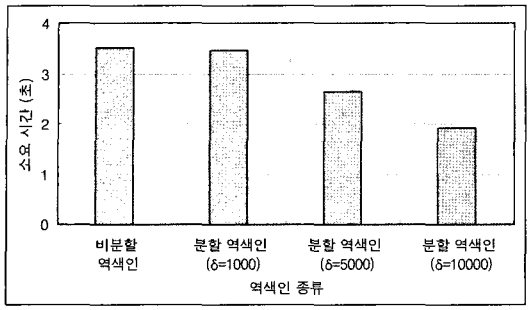


그림 7 인덱스의 종류에 따른 질의 응답 시간 I (DBLP)

INEX2003에는 125개의 XML 문서들이 있으며 각 XML 문서는 IEEE 저널들에 게재된 논문 제목과 내용을 포함하고 있다. XML 문서 하나의 크기는 평균 4.5 메가바이트(mega bytes)이다. 우리는 110개의 문서들의 최소 검색 깊이를 0, 2, 4, 6으로 각각 가정하여 실험하였다. 시험 질의는 크게 CO(Content Only)와 CAS (Content And Structure)로 구분된다. 본 논문에서는 XML 문서의 구조를 고려하지 않으므로 CO에 속하는 36개의 질의 중에서 6개의 질의를 선택하여 실험하였다. 6개 질의에 대한 키워드의 구성은 표 1에 나타나 있다. 6개의 선택된 질의 중 앞의 세 개(Q1~Q3)는 세 개의 질의 키워드로 구성된 비교적 간단한 질의들이고, 뒤의 질의 세 개(Q4~Q6)는 최소 다섯 개 이상의 키워드로 구성된 상대적으로 복잡한 질의들이다.

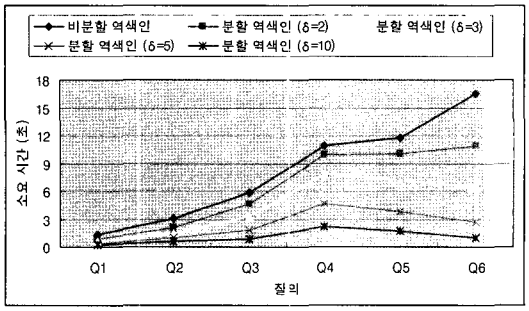
그림 8은 유효 결과 깊이가 2, 4, 6일 때에 분할되지 않은 일반 인덱스와 분할된 인덱스를 사용한 6개 실험 질의들의 응답 시간을 나타낸다. 분할되지 않은 기존의 인덱스를 사용할 경우, 각 질의 키워드들의 인덱스에 포함된 모든 원소들의 조합이 고려되어 유효 결과 깊이 이상의 최소 공통 선조가 검색 결과로 산출된다. 그림 8(a)는 문서들의 유효 결과 깊이가 2이고 유효 결과 깊이 2로 분할된 인덱스를 사용할 경우에 각 질의의 응답 시간을 보여준다. 분할된 인덱스를 사용하는 경우에는 동일 파티션에 속한 원소들 간의 조합만이 고려되어 검색 결과가 산출되고, 각 인덱스에 공통으로 원소를 가지지 않는 파티션들은 읽히지 않으므로 질의 처리 시간이 빨라지는 것을 볼 수 있다. 파티션 요소가 커지면, 하나의 키워드에 대한 인덱스가 가지는 파티션 개수가 많아지고, 유효 결과 깊이보다 작은 깊이의 최소 공통 선조를 가지는 노드들이 동일 파티션에 속할 확률이 적어지게 된다. 파티션 요소가 큰 인덱스를 사용할수록 질의 처리 시간이 빠르게 나타났다. 파티션 요소 10의 분할 인덱스를 사용할 경우 분할되지 않은 인덱스를 사용할 때 보다 최소 79%(Q4)에서 최대 94%(Q6)까지의 응답 시간 감소가 나타났으며, 평균적으로 84%에 달하는 응답 시간이 감소되었다. 파티션 요소가 5, 3, 2인 분할 인덱스를 사용할 경우에도 분할 인덱스를 사용하지 않은 것보다 평균적으로 70%, 46%, 24%에 달하는 응답 시간의 감소가 나타났다.

그림 8(b)는 문서들의 유효 결과 깊이가 4이고 유효 결과 깊이가 4로 분할된 인덱스를 사용하여 6개의 질의에 대한 응답 시간을 보여준다. 유효 결과 깊이가 4일 때도 마찬가지로 분할을 사용하는 것이 분할 인덱스를 사용하지 않은 것보다 더 빠른 응답 시간을 나타내었으며, 파티션 요소가 큰 분할 인덱스일수록 더 빠른 응답 시간을 나타내었다. 파티션 요소 10의 분할 인덱스를 사용하는 경우 분할되지 않은 인덱스를 사용하는 것보다 최소 83%(Q2)에서 최대 92%(Q6)까지 응답 시간의 감소를 보였으며, 평균적으로 88%의 응답 시간 감소가 나타났다. 유효 결과 깊이가 4일 때 분할 인덱스를 사용하는 것이 유효 결과 깊이가 2일 때 분할 인덱스를 사용하는 것에 비해 응답 시간에 보다 큰 영향을 미치는 것으로 나타났다. 파티션 요소가 5, 3, 2인 분할 인덱스를 사용할 경우 분할 인덱스를 사용하지 않은 것보다 평균적으로 87%, 71%, 45%에 달하는 응답 시간이 감소되었다.

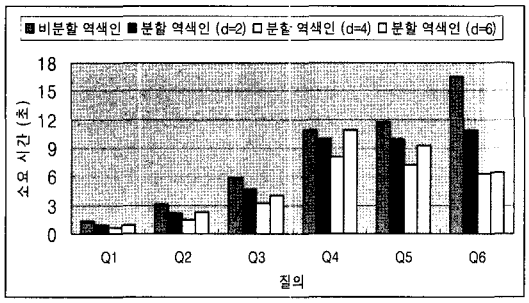
그림 8(c)는 문서들의 유효 결과 깊이가 6이고 유효 결과 깊이의 분할 인덱스를 사용한 질의 처리 시간을 보여준다. 이 경우에도 분할 인덱스를 사용하는 경우에 분할되지 않은 인덱스를 사용하는 것보다 파티션 요소에 따라 응답 시간이 평균적으로 77%(δ=10), 80%(δ=5), 83%(δ=3), 63%(δ=2)의 응답 시간 감소가 나타났다. 유효 결과 깊이가 6일 때는 파티션 요소 10의 인덱스를 사용하는 경우에 오히려 파티션 요소 5의 인덱스들을 사용하는 것 보다 질의 처리 시간이 많이 걸리는 것으로 나타났다. 또한 파티션 요소 5의 인덱스를 사용하는 것이 파티션 요소 3의 인덱스를 사용하는 것보다

표 1 실험 질의

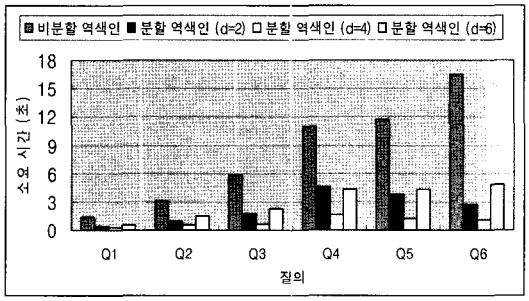
질의 번호	질의	질의 번호	질의
Q1	singular value decomposition	Q4	information data visualization technique hierarchy space
Q2	wireless security application	Q5	concurrency control semantic transaction management application performance benefit
Q3	recommender system agent	Q6	machine learning adaptive algorithm probabilistic model neural network support vector machine



(a) d = 2

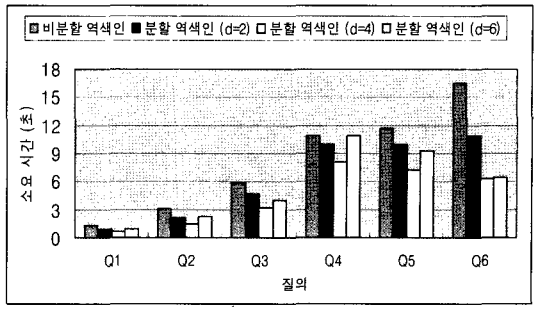


(b) d = 4

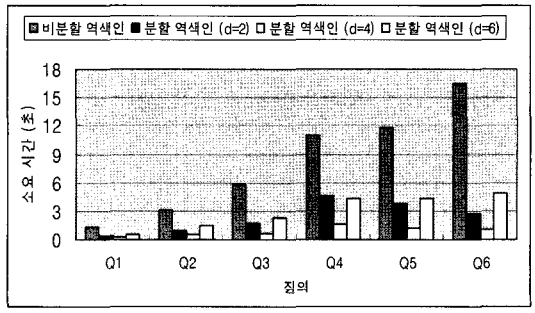


(c) d = 6

그림 8 인덱스의 종류에 따른 질의 응답 시간 II (INEX2003)



(a) δ = 2



(b) δ = 5

그림 9 인덱스의 종류에 따른 질의 응답 시간 III (INEX2003)

질의 처리 시간이 빠르게 나타났다. 이러한 현상이 나타나는 이유는 유효 결과 깊이가 6일 때 파티션 요소가 전체 파티션 개수에 미치는 영향이 크기 때문이다. 즉, 분할 인덱스는 파티션 요소가 2, 3, 5, 10일 때 각각 6^2=36, 6^3=216, 6^5=7,776, 6^{10}= 60,466,176 개의 파티션을 생성하게 되고, 질의 처리 시 파티션들의 헤더를 읽어서 공통으로 원소를 지니고 있는 파티션을 찾는 처리에 대한 오버헤드가 증가하였다.

그림 9는 임의의 유효 결과 깊이에 대해 분할된 인덱스를 사용하여 사용자 요구에 부합하는 유효 결과 깊이의 검색 결과를 산출한 실험 결과를 나타낸다. 본 실험을 위하여 분할되지 않은 인덱스와 유효 결과 깊이 2, 4, 6 각각에 대해 파티션 요소를 2와 5로 분할된 인덱스를 물리적으로 구축하였다. 이때 사용자가 원하는 유효 결과 깊이는 4로 가정하여 실험하였다. 그림 9(a)는 파티션 요소 2로 분할된 인덱스를 이용한 질의 처리 시간을 나타낸다. 유효 결과 깊이 2와 4에 대해 분할된 인덱스를 사용하여 검색 결과를 얻고자 할 경우, 동일 파티션에 속한 원소들 간의 조합만을 고려하여 깊이가 4보다 큰 최소 공통 선조를 검색 결과로 반환한다. 따라서 분할되지 않은 인덱스를 사용하여 검색 결과를 얻는 것 보다 빠른 응답 시간을 나타냈다. 또한 유효 결과 깊이 4에 대해 분할된 인덱스를 사용하는 것이 유효 결과 깊이 2에 대한 분할된 인덱스를 사용하는 것보다 더 빠른 응답 시간을 나타냈다.

유효 결과 깊이 6에 대한 인덱스를 이용할 경우 물리적인 파티션들을 가상의 인덱스로 합병하고 유효 결과 깊이 4에 대한 검색 결과를 검색한다. 유효 결과 깊이 4에 대한 인덱스를 사용하는 것과 비교하여 33%, 49%, 25%, 35%, 28%, 2%의 응답 시간의 증가가 나타났으나, 분할되지 않은 인덱스를 사용하는 것에 비해 34%, 27%, 31%, 0%, 21%, 61%의 응답 시간 감소를 나타냈다.

그림 9(b)는 파티션 요소 5로 분할된 인덱스를 이용한 각 질의의 질의 처리시간을 나타낸다. 유효 결과 깊이 4에 대한 인덱스를 사용하는 것과 비교하여 평균적

으로 3.2배나 더 많은 시간이 소요되었으나, 분할되지 않은 인덱스를 사용하는 것보다는 평균적으로 60%의 응답 시간 감소가 나타났다. 파티션 요소가 크면 하나의 가상 파티션을 구성하는 물리적 파티션들의 개수가 많아지고 파티션 요소가 2일 때에 비해서 파티션 합병을 통한 검색 결과 검색에 필요한 응답 시간이 더 크게 나타났다.

5. 결론 및 향후 계획

유효 결과 깊이는 XML 키워드 검색에서 매우 유용한 정보이다. 기존의 인덱스 구조 하에서는 검색 결과 원소들의 최소 깊이를 효율적으로 이용할 수 없었으나, 본 논문에서는 유효 결과 깊이를 이용하여 분할된 인덱스를 생성하여 효율적인 XML 키워드 검색을 수행하는 방법을 제안하였다. 이를 위하여, 분할 함수가 가져야 할 분할 요구 사항과 합병 요구 사항을 기술하고, 두 종류의 요구 사항을 만족하는 실제 분할 함수 $PF_d(n)$를 제안하였다. 또한 제안된 분할 함수로부터 분할된 인덱스를 구축하는 과정과 동일 파티션에 속한 원소들만을 고려하여 검색 결과를 산출하는 방법을 기술하였다.

XML 문서에 대한 유효 결과 깊이를 알 경우 분할 인덱스를 사용하는 것은 검색시간을 상당히 감소시키는 것으로 관찰되었다. 특히 유효 결과 깊이가 깊을수록 분할 인덱스를 통한 성능 향상이 크게 나타났다. 또한 파티션 요소를 크게 하여 전체 파티션 개수가 많아질수록 분할 인덱스들 간의 공통 파티션 개수가 줄어들고 질의 처리 시간이 빠르게 나타났으나, 너무 많은 파티션은 운용상의 오버헤드로 인하여 오히려 질의 처리 시간이 증가하였다. 기존의 인덱스에서는 질의를 구성하는 키워드가 많아질수록 고려해야 하는 원소들 간의 조합들이 급격히 많아졌으나, 분할 인덱스를 사용하는 경우에는 질의 키워드의 수가 많아질수록 질의 키워드들의 분할 인덱스들 간의 공통 파티션 개수가 줄어들어 고려해야 하는 원소들 간의 조합이 감소하였다.

사용자는 관리자가 정한 유효 결과 깊이보다 낮은 깊이의 원소를 검색 결과로 얻을 수 있다. 본 논문에서는 분할 인덱스 구조를 통하여 물리적으로 구성된 분할 인덱스로부터 사용자가 원하는 다양한 수준의 검색 결과를 얻을 수 있도록 하였다. 즉 물리적인 파티션들을 합병하여 논리적인 가상의 파티션을 구성하고, 동일 가상 파티션에 속한 원소들만을 고려하여 검색 결과를 얻을 수 있도록 하였다. 이러한 방법은 사용자가 원하는 수준에서의 분할 인덱스를 물리적으로 생성하는 것보다는 많은 시간이 걸리지만 분할되지 않은 인덱스를 사용하여 검색 결과를 반환하는 것보다는 빠른 응답 시간을 나타내었다.

참 고 문 헌

[1] WWW Consortium, http://www.w3.org/XML/

[2] Salton, G., and McGrill, M.J., "Introduction to Modern Information Retrieval," McGraw-Hill, New York, 1983.

[3] Carmel, D., Maarek, Y,S., Mandelbrod, M., Mass, Y., and Soffer, A., "Searching XML Documents via XML Fragments," In Proceedings of the 26th annual international ACM SIGIR conference on Research and development in informaion retrieval, pp. 151-158, 2003.

[4] Cohen, S., Mamou, J., Kanza, Y., and Sagiv, Y., "XSEarch: A Semantic Search Engine for XML," In Proceedings of 29th International Conference on Very Large Data Bases, pp.45-56, 2003.

[5] Guo, L., Shao, F., Botev, C., and Shanmugasundaram, J., "XRANK: Ranked Keyword Search over XML Documents," In Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data, pp.16-27, 2003.

[6] Hritidis, V., Papakonstantinou, P., and Balmin, A., "Keyword Proximity Search on XML Graph," In Proceedings of the 19th International Conference on Data Engineering, pp.367-378, 2003.

[7] Theobald, A., and Weikum, G., "Adding Relevance to XML," In Proceedings of the 3th International Workshop on the Web and Databases, pp.105-124, 2000.

[8] Xu, Y., and Papakonstantinou, Y., "Efficient Keyword Search for Smallest LCAs in XML Databases," In Proceedings of the 2005 ACM SIGMOD international conference on Management of data, pp.527-538, 2005.

[9] Mignet, L., Barbosa, D., and Veltri, P., "The XML Web: a First Study," In Proceedings of the 12th International World Wide Web Conference, pp.500-510, 2003.

[10] Anh, V., Krester, O., and Moffat, A., "Vector-Space Ranking with Effective Early Termination," In Proceedings of the 24th Annual International ACM SIGIR Confenfrence on Research and Development in Information Retrieval, pp.35-42, 2001.

[11] Florescu, D., Kossmann, D., and Manolescu, L., "Integrating Keyword Search into XML Query Processing," Computer Networks, Vol.33, No.1-6, pp.119-135, 2000.

[12] Moffat, A., and Zobel, J., "Self-Indexing Inverted Files for Fast Text Retrieval," ACM Transactions on Database Systems, Vol.14, No.4, pp.349-379, 1996.

[13] Putz, S., Using a Relational Database for an Inverted Text Index. XEROX Technical Report '91.

[14] DBLP, http://www.informatik.uni-trier.de/~ley/db/

index.html
[15] Initiative for the evaluation of XML retrieval, http://inex.is.informatik.uni-duisburg.de:2003/
[16] BerkeylyDB, http://www.sleepycat.com

김 성 진
정보과학회논문지 : 데이타베이스
제 33 권 제 5 호 참조

이 형 동
정보과학회논문지 : 데이타베이스
제 33 권 제 5 호 참조

김 형 주
정보과학회논문지 : 데이타베이스
제 33 권 제 1 호 참조