효율적인 XML 질의 처리를 위한 적응형 경로 인덱스
(Adaptive Path Index for Efficient XML Query Processing)

민 준 기 † 심 규 석 †† 정 진 완 †††
(Jun-Ki Min) (Kyuseok Shim) (Chin-Wan Chung)

요 약 XML은 정형적(regular) 구조로부터 비정형적(irregular) 구조, 평탄한(flat) 구조로부터 깊게 내포된(deeply nested) 구조에 이르기까지 다양한 형태의 자료를 표현할 수 있다. 따라서, 현재 XML은 효율적인 데이타 교환 및 통합 기능을 제공하면서 웹 문서 표준으로 부각되고 있다. 또한, XML로 표현되어 있는 정보를 검색하기 위한 다양한 질의 언어들이 제안되었다. XPath와 XQuery같은 XML 질의 언어들은 XML 엘리먼트(element)로 이루어진 비정형적 구조를 탐색하기 위하여 경로 표현식(path expression)을 기반으로 한다. 이러한 경로 표현식을 효율적으로 처리하기 위하여 다양한 경로 인덱스들이 제안되었으나, 기존의 경로 인덱스들은 XML 데이타의 구조 정보만을 이용하여 생성된다. 따라서, 본 논문에서는 XML 데이타의 구조 정보와 질의 부하(workload)를 이용한 적응형 경로 인덱스를 제안한다. 본 논문에서 제안하는 적응형 경로 인덱스는 질의 성능을 향상하기 위하여 해쉬 트리와 그래프 구조를 이용하여 자주 사용되는 경로 정보와 XML 데이타의 구조 요약을 관리한다. 실험 결과로서, 적응형 경로 인덱스가 기존의 경로 인덱스들에 비하야 평균적으로 2배에서 69배 좋은 성능을 보였다.

키워드 : XML, 질의 처리, 경로 인덱스

Abstract XML can describe a wide range of data, from regular to irregular and from flat to deeply nested. Thus, XML is rapidly emerging as the de facto standard for the Web document format since XML supports an efficient data exchange and integration. Also, to retrieve the data represented by XML, several XML query languages are proposed. XML query languages such as XPath and XQuery use path expressions to traverse irregularly structured data which comprise XML elements. To evaluate path expressions, various path indexes are proposed. However, traditional path indexes are constructed by utilizing only the XML data structure. Therefore, in this paper, we propose an adaptive path index which utilizes the XML data structure as well as query workloads. To improve the query performance, the adaptive path index proposed by this paper manages the frequently used paths and the structural summary of the XML data using a hash tree and a graph structure. Experimental results show that the adaptive path index improves the query performance typically 2 to 69 times compared with the existing indexes.

Key words : XML, query processing, path index

1. 서 론

인터넷의 등장은 전자 형태의 다양한 정보의 극적인 증가를 유발하였다. XML[1]은 인터넷에서 기존에 사용하던 HTML의 한계를 극복하고 SGML의 복잡함을 해결하는 방안으로 1998년 W3C에서 발표한 것이다. XML은 정형적(regular) 구조로부터 비정형적(irregular) 구조, 평탄한(flat) 구조로부터 깊게 내포된(deeply nested) 구조에 이르기까지 다양한 형태의 자료를 표현할 수 있는 준구조적 데이타[2]의 일종이다. 이러한 XML의 유연성 때문에 XML은 차세대 웹의 데이타 표현 및 교환의 표준 언어로서 각광 받고 있다.

XML로 표현되어 있는 정보를 검색하기 위한 다양한 질의 언어들이 제안된 바 있다. XPath[3]와 XQuery[4] 같은 XML 질의 언어들은 XML 엘리먼트(element)로 이루어진 비정형적 구조를 탐색하기 위하여 경로 표현식(path expression)을 사용한다. 따라서, 주어진 경로

· 이 연구는 교육인적자원부의 BK21 사업과 정보통신부의 대학 IT연구센터(ITRC)의 지원을 받아 수행되었습니다.
† 비 회 원 : 한국과학기술원 전자전산학과
jkmin@islab.kaist.ac.kr
†† 정 회 원 : 서울대학교 전기컴퓨터공학부 교수
shim@ee.sun.ac.kr
††† 종신회원 : 한국과학기술원 전자전산학과 교수
chungcw@islab.kaist.ac.kr
논문접수 : 2003년 3월 21일
심사완료 : 2003년 10월 15일

표현식에 따라서 비정형적 구조의 XML 데이타를 탐색해 나가는 것이 XML 질의의 처리를 위한 중요한 요소이다. 그러나, XML 데이타를 구성하는 엘리먼트들은 디스크의 다른 위치에 분산되어 있을 수 있으므로, XML 질의 처리의 성능은 매우 저하된다. 더욱이, 부분 매칭(partial matching) 경로 표현식으로 이루어진 XML 질의를 처리하기 위하여서는 XML 데이타에 속한 모든 엘리먼트들을 검색하여야 하므로 매우 비효율적이다.

한편, 구조 요약(structural summary)[5]나 경로 인덱스(path index)[6,7,8]는 주어진 경로 표현식에 대하여 XML 데이타의 관련 있는 부분만을 검색할 수 있도록 하여 XML 데이타의 검색 속도를 향상시킨다. 따라서, XML 데이타의 검색 속도 향상을 위한 구조 요약 추출 및 경로 인덱스 생성 방안이 최근 많은 관심을 받고 있으며, 비정형 특성을 지니는 데이타의 검색 성능 향상을 위한 여러 가지 경로 인덱스가 제안되었다. 이러한 인덱스들의 예로는 Data Guide[7], 1-Index[8], Index Fabric[6] 등이 있다.

이러한 경로 인덱스들은 XML 데이타의 루트(root)부터 시작되는 모든 단순 경로들(simple paths)을 색인화 함으로서 부분 매칭 경로 표현식을 처리함에 있어서 인덱스를 소모적으로 탐색하여야 하는 부담이 존재한다. 따라서, 본 연구에서는 XML 데이타의 구조 정보와 질의 부하를 이용한 적응형 경로 인덱스인 APEX를 제안한다.

본 연구의 구성은 다음과 같다. 2장 관련연구에서는 기존의 다양한 경로 인덱스들의 특징을 설명하고 3장에서는 본 연구에서 제안하고자 하는 적응형 경로 인덱스인 APEX의 구조에 대하여 설명하고 4장에서는 질의 부하를 이용한 APEX 관리 기법 및 질의 처리에 대하여 설명한다. 5장에서는 다양한 XML 데이타를 이용하여 본 연구에서 제안하는 APEX와 기존의 경로 인덱스와의 성능 평가 결과를 제시한다. 마지막으로 6장에서는 결론에 대하여 논의한다.

2. 관련연구

지금까지 XML과 같은 비정형적 데이타에 대한 경로 표현식을 효율적으로 지원하기 위하여 다양한 경로 인덱스들이 제안되었다.

Goldman과 Widom은 strong DataGuide [7]라고 하는 경로 인덱스를 개발하였다. 이 인덱스들은 XML 과 같은 준구조적 데이타의 구조 정보를 추출하기 위한 방안으로 제안되었으며, XML 데이타의 루트 엘리먼트(root element)로부터 시작되는 단순 경로(simple path)들이 중복되어 나타나지 않도록 표현한다. DataGuide의 생성 방식은 비결정적 유한 오토마타(non-deterministic finite automata)를 결정적 유한 오토마타(deterministic finite automata)로 변환하는 알고리즘[9]과 비슷하게 동작한다. 따라서, 주어진 XML 데이타가 매우 복잡한 형태의 그래프 구조일 경우에 strong DataGuide의 크기가 원본 XML 데이타보다 더 커질 수 있다는 문제가 있다.

Milo와 Suciu는 DataGuide와는 다른 1-index[8]를 제안하였다. 1-Index도 루트 엘리먼트로부터 시작되는 모든 경로들의 정보를 유지한다는 점에서는 strong DataGuide와 동일하다. 1-Index는 그래프 이론(graph theory)에 기반한 역방향 시뮬레이션(backward simulation)과 역방향 바이 시뮬레이션(backward bisimulation)을 이용하여 인덱스를 생성한다. 이 방식은 그래프 상의 두 노드가 존재할 때 각 노드로부터 시작되는 경로들 집합이 서로 같은 경우 이 두 노드를 하나의 노드로 합병 시키는 방식이다. 이와 같은 방식을 이용하면 strong DataGuide와는 달리 비결정적 유한 오토마타와 같은 그래프를 얻을 수 있다. 그러나 주어진 입력 그래프가 트리 형태일 경우에는 1-Index와 strong DataGuide는 서로 동일하다. 따라서, 1-Index는 strong DataGuide의 비결정적 변형(non deterministic version)이라고 볼 수 있다.

객체 지향 데이타베이스 분야에서는 임의의 두 객체 간의 자주 사용되는 참조(reference)들을 지원하기 위하여 접근 지원 관계(ASR : access support relation) [10]가 사용되어 왔다. 그러나 이 접근 지원 관계는 임의의 질의의 참조 고리(reference chain) 들을 구체화(materialize)한 것으로 미리 정의해 놓은 부분 경로들에 대하여서만 지원이 가능하다는 문제가 있다.

Cooper 등은 루트 엘리먼트로 시작되는 모든 경로를 유지한다는 점에서 strong DataGuide와 개념적으로 유사한 Index Fabric[6]을 제안하였다. Index Fabric은 데이타 값을 가지는 엘리먼트들의 경로들을 데이타 값과 같이 암호화(encoding)하여 문자열(string)로 변환하고 이를 Patricia Trie와 같은 문자열 인덱스를 이용하여 유지한다. 그러나, Index Fabric은 XML 엘리먼트들 간의 부모-자식 관계(parent-child relationship)를 유지할 수 없다는 단점이 존재한다. 따라서, Index Fabric은 부모-자식 관계를 이용하여야 하는 부분 매칭 경로 표현식에는 효과적이지 못하다.

XML 데이타의 사용자는 데이타의 구조를 고려하지 않으며 원하는 결과를 얻기 위하여 의도적으로 부분 매칭 경로 표현식을 이용한다. 그러나, strong DataGuide, 1-Index, Index Fabric 등 위에서 언급한 XML 경로 인덱스들은 루트 엘리먼트로부터 시작되는 모든 경로들

을 표현함으로서 부분 매칭 경로 표현식을 처리하기 위하여서는 인덱스를 소모적으로 탐색함으로써 성능 저하를 유발한다. 더욱이, 이러한 경로 인덱스들은 데이타만을 이용하여 생성되므로, 자주 사용되는 경로 표현식을 효과적으로 처리하기 위한 과거의 질의 정보들의 집합인 질의 부하(query workload)를 활용하지 못하고 있다. 따라서 본 연구에서는 자체 개선(self-tuning) 기법[11]을 이용한 적응형 경로 인덱스를 개발하였다.

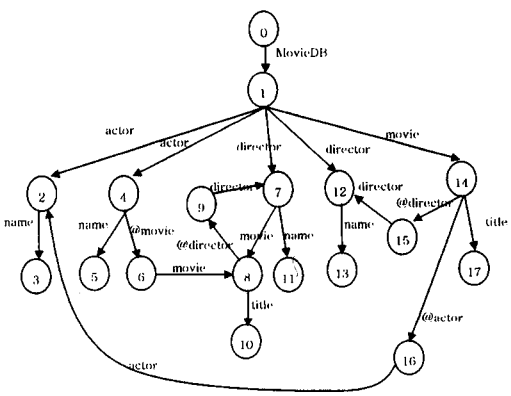
3. APEX의 구조

우선 본 연구에서 사용되는 XML 데이타의 구조에 대하여 간략히 기술하고, 이를 기반으로 본 연구에서 제안하는 적응형 경로 인덱스의 구조에 대하여 설명한다.

그림 1에서 보는 바와 같이, 우리는 XML 데이타 구조를 라벨-간선(Label-Edge) 그래프로 표현한다. 이를 XML 그래프라고 한다. 이때, 그림 1에 나타나는 ID-IDREF 관계는 IDREF 타입의 노드로부터 대응되는 ID 타입의 애트리뷰트(attribute)를 가지는 엘리먼트를 표현하는 노드까지의 간선(edge)으로 표현된다. 이 간선의 라벨은 해당 엘리먼트의 라벨을 가진다. 또한 IDREF 타입의 애트리뷰트를 위한 노드를 가리키는 간선의 라벨은 "@"으로 시작된다.

```
<MovieDB>
    <actor id = "a1">
        <name> actor1 </name></actor>
    <actor id = "a2"  movie = "m1">
        <name> actor2 </name></actor>
    <director id = "d1">
        <name> director1</name>
        <movie id = "m1" director = "d1">
            <title> movie2 </title></movie></director>
    <director id = "d2">
        <name> director2</name></director>
    <movie id = "m2" actor = "a1" , director = "d2">
        <title> movie1</title></movie>
</MovieDB>
```

(a) XML Data



(b) XML Graph

그림 1 XML Data 예제

XML 그래프에는 라벨의 연속으로 이루어지는 경로들이 존재하며 이는 XML 질의의 경로 표현식으로 사용된다. 각 경로들 간에는 다음과 같은 관계가 존재할 수 있다. 우선, 임의의 두 경로 A와 B에 대하여 다음 정의 1과 같은 관계가 있다면 "경로 B가 경로 A를 포함한다"라고 한다.

정의 1.

임의의 두 경로 $A=a_1.a_2.\cdots.a_n$과 $B=b_1.b_2.\cdots.b_m$에 대하여, 만약 $a_1=b_i$, $a_2=b_{i+1}$, $\cdots$, $a_n=b_{i+n-1}$이고 $1\le i$, $i+n-1\le m$이라고 할 때 **"경로 A가 경로 B에 포함된다"**라고 하거나 A가 B의 **하위 경로**(subpath)라고 한다. 더욱이, A가 B의 하위 경로이고 $m=i+n-1$이면, A를 B의 **서픽스**(suffix)라고 한다.

본 연구에서 제안하는 적응형 경로 인덱스인 APEX는 XML 데이타에 존재하는 경로들 중에서 자주 사용되는 경로 정보를 이용한다. 자주 사용되는 경로 인덱스를 추출하기 위하여, 우리는 데이타베이스가 경로 표현식으로 이루어진 질의 부하를 유지하고 있다고 가정하고 자주 사용되는 경로들을 식별하기 위한 순차적 패턴 마이닝(sequential pattern mining)[12,13]에서 사용되는 지지도(support) 개념을 이용한다. 임의 경로 p에 대한 지지도는 XML 그래프 상의 임의의 경로 p가 주어진 XML 경로 표현식들 중에서 얼마나 자주 사용되었는지를 나타내며 다음과 같이 정의한다.

정의 2.

sup(p) = |Q|/|W|

여기서, |W| = 주어진 XML 경로 표현식들의 수, 즉 질의 부하로 유지하고 있는 경로들의수, |Q| = p를 포함하는 XML 경로 표현식들의 수이다.

또한, 적응형 경로 인덱스 관리 방법에서는 다음 정의 3에 정의된 필요한 경로(required path)들을 파악하고 이를 활용하여 경로 인덱스를 관리한다.

정의 3.

XML 그래프에 존재하는 경로 p의 지지도 sup(p)가 주어진 사용자 정의 최소 지지도(user specified minimum support), minSup 보다 크거나 p의 길이가 1일 때, p를 **필요한 경로**(required path)라 한다. 특별히, 사용자 정의 최소 지지도 minSup보다 큰 지지도를 가지는 경로를 **자주 사용되는 경로**(frequently used path)라 한다.

일반적으로 경로 인덱스는 XML 데이타 상에 존재하는 경로들과 각 경로를 이용하여 도달할 수 있는 XML 데이타 상의 노드 또는 간선들의 집합을 색인화하여 관

리하게 된다. APEX는 필요한 경로들과 각 필요한 경로들을 이용하여 도달할 수 있는 간선들의 집합을 색인화한다. 임의 경로 p에 대한 간선 집합 T(p)는 다음과 같이 정의된다.

정의 4.

$p=l_i.\cdots.l_{j-1}.l_j$일 때, $T(p)=\{\langle o_{i-1},o_i\rangle | o_{i-1}.l_i.o_i.\cdots.l_{j-1}.o_{j-1}.l_j.o_j\}$

여기서, p에 나타나는 라벨 l_k에 대하여 o_{k-1},o_k는 XML 그래프에 존재하는 노드들이고 o_{k-1}에서 o_k로 라벨이 l_k인 간선이 XML 그래프에 존재한다.

정의 4로부터 임의 경로 p와 q에 대하여 p가 q의 서픽스라면 $T(p) \supseteq T(q)$임을 알 수 있다. 예를 들어 그림 1에서 경로 name은 actor.name의 서픽스이다. 이때 T(name) = {<2,3>, <4,5>, <7,11>, <12,13>}이며, T(actor.name) ={<2,3>, <4,5>}임을 알 수 있다. 따라서, T(name) ⊇ T(actor.name)을 보인다. 필요한 경로들의 집합에 속한 모든 경로들에 대하여 간선 집합 T(p)들을 관리하는 것은 많은 저장 부담이 요구된다. 따라서, APEX에서는 다음과 같이 경로 p의 익스텐트(extent), E(p)를 정의하고 이를 관리한다.

정의 5.

XML 그래프의 경로들 중 루트로부터 시작되는 경로들의 집합을 Q_{XML}이라고 하고 필요한 경로들의 집합을 R이라 하자. 집합 R에 속한 경로 p에 대하여, $Q_G(p)$ = {l|l ∈ Q_{XML} (l은 p를 서픽스로 가진다.}, $Q_A(p)$ = {l|l ∈ Q_{XML} (l은 p를 서픽스로 가지는 경로 q (≠ p) ∈ R 를 서픽스로 가진다)이라고 하자. $Q(p) = Q_G(p)-Q_A(p)$라면 **익스텐트 E(p)**= $\cup_{r\in Q(p)}T(r)$로 정의된다.

위의 정의 5에서 임의의 경로 p에 대한 q는 R에 속한 경로들 중에서 p를 서픽스로 가지는 경로라고 할 때, $T(p) = \cup_{\forall q} E(q)$임을 알 수 있다. 예를 들어, 그림 1의 XML 그래프에 대하여, actor.name 만이 자주 사용되는 경로라 할 때, QG(name) = {MovieDB.actor.name, MovieDB.director.name, ···, MovieDB.director.movie.@director.director, MovieDB.movie.@actor.actor.name} 이고 QA(name) = {MovieDB.actor.name, MovieDB.movie.@actor.actor.name}이다. 따라서, Q(name) = {MovieDB.director.name, ···, MovieDB.director.movie.@director.director}이다. 그러므로, E(name) = {<7,11>, <12,13>}, E(actor.name) = {<2,3>,<4,5>}이고 T(name) = E(name) ∪ E(actor.name) 임을 알 수 있다.

그림 2는 그림 1에 대하여 필요한 경로가 길이가 1인 모든 경로와 direct.movie, @movie.movie, actor.name 일 때의 APEX를 도시한 것이다. 그림 2에서 보는 바와 같이 APEX는 해쉬 트리(hash tree)와 그래프 구조(graph structure)로 구성되어 있다.

해쉬 트리는 필요한 경로들에 대한 정보를 유지하며, 그래프 구조의 각 노드와 필요되는 경로를 대응시킨다. 그래프 구조는 주어진 XML 그래프의 구조 요약 정보를 제공하며, 그래프 구조의 각 노드는 해쉬 트리의 엔트리와 1 대 1 대응하며, 그래프 구조의 노드들은 해쉬 트리 엔트리가 표현하는 필요한 경로의 익스텐트를 가지고 있다. 이 경우 필요되어지는 메모리의 양은 기본적으로 주어진 XML 그래프의 구조와 자주 사용되어지는 경로들을 포함한 필요한 경로들의 수에 영향을 받게 된다. 기본적으로 해쉬 트리는 필요한 경로들에 대한 정보를 유지하고 있음으로 해쉬 트리의 크기는 필요한 경로들의 수에 비례한다. 그래프 구조의 각 노드는 필요한 경로와

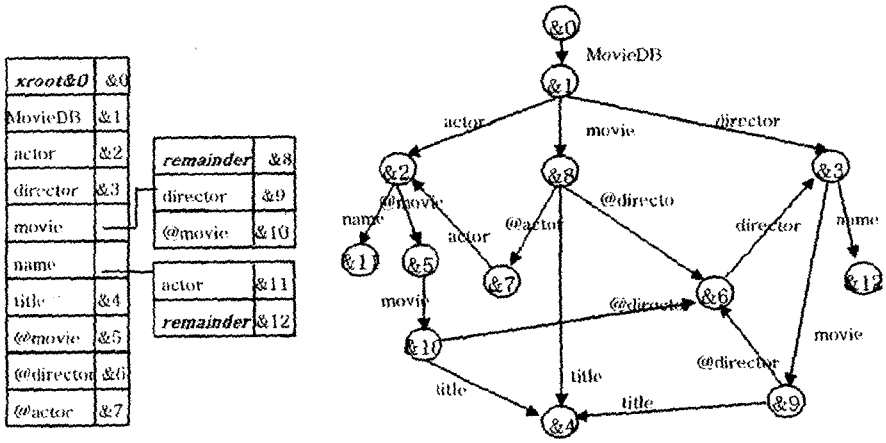


그림 2 APEX의 예

대응되므로 그 수가 필요한 경로의 수와 같고 그래프 구조의 각 노드의 출발 간선은 최대 주어진 XML 그래프 상의 라벨 수와 같게 된다. 따라서 메모리 요구 양은 O(R*L), R은 필요한 경로들의 수, L은 XML 그래프 상의 라벨 수, 임을 알 수 있다. 또한, 그래프 구조의 각 노드에서 관리하는 익스텐트의 수는 최대 O(E), E는 XML 그래프의 간선 수, 임으로 APEX를 위한 추가 메모리 양은 O(R*(L+E))임을 알 수 있다.

그래프 구조의 임의의 두 노드 x, y와 x의 익스텐트에 속한 간선<$start_x$, end_x>와 y의 익스텐트에 속한 간선<$start_y$, end_y>에 대하여, 만약 end_x = $start_y$라면 그래프 구조의 노드 x에서 y로 가는 간선이 존재하도록 한다. 또한 그 간선의 라벨은 y에 속한 간선들을 분류한 라벨을 사용한다. 또한, 특별히 XML 그래프의 루트 노드를 위하여, 초기 적응형 경로 인덱스에 하나의 노드를 만들고 <null, 루트 노드>로 이루어진 가상의 간선을 해당 노드의 익스텐트로서 저장한다. 그리고 그 노드를 편리하게 xroot로 표현한다. 또한, 임의의 경로 p에 대하여 p를 서픽스로 하는 경로 q가 자주 사용되는 경로이고 E(p)가 공집합이 아닐 때에는, 이를 위하여 *remainder* 엔트리를 생성하고 이에 대응되는 노드를 만들어 E(p)를 관리한다. 예를 들어, 그림 2에서 E(name)은 &12 노드에 저장되며 이는 해쉬 트리의 루트 노드의 name 엔트리의 하위 노드 내에 있는 remainder 엔트리에 등록되어 있다.

그림 2에서 필요한 경로 direct.movie에 대한 정보는 해쉬 트리에서 역방향으로 검색(lookup)함으로서 파악할 수 있다. 예를 들어, 해쉬 트리의 루트 노드에서 movie 엔트리가 가리키는 하위 노드에 대하여 direct를 검색하여 direct.movie에 대한 정보를 검색할 수 있다.

4. APEX의 생성 및 관리

본 연구에서 제안하는 APEX의 관리 구조도는 그림 3에 나타나 있다.

그림 3에서 보듯이 적응형 경로 인덱스 관리 모듈은 크게 초기화 모듈(initialization module), 자주 사용되는 경로 추출 모듈(frequently used path extraction module), 갱신 모듈(update module) 이렇게 3개의 주요 모듈들로 구성되어 있다.

초기화 모듈은 과거에 사용된 질의 정보 없이 처음으로 적응형 경로 인덱스를 생성할 때 사용되는 모듈이다. 이 모듈은 적응형 경로 인덱스의 초기 인덱스인 초기 적응형 경로 인덱스, $APEX^0$를 생성한다. $APEX^0$는 적응형 경로 인덱스의 가장 단순화된 형태로 복잡한 적응형 인덱스인 APEX 생성의 씨앗 역할을 한다. 경로 추출 모듈과 갱신 모듈은 현 시점의 경로 인덱스를 이용

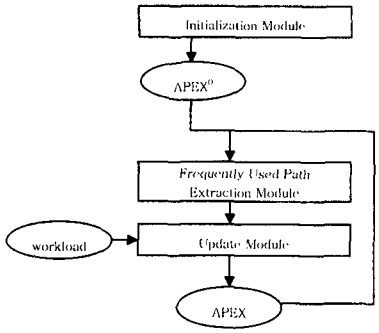


그림 3 APEX 관리 구조도

한 질의들의 집합인 질의 부하(workload)을 이용하는 모듈로서 자주 사용되는 경로들을 활용하여 더욱 효율적인 경로 인덱스를 생성하는 데 사용된다. 이때 질의 부하는 $l_i.l_{i+1}.\cdots.l_{j-1}.l_j$ 형태의 단일 경로 표현식(single path expression)의 집합으로 되어 있다.

4.1 $APEX^0$: 초기 인덱스 구조

$APEX^0$는 APEX를 생성하기 위한 초기 구조로서, $APEX^0$를 생성하기 위한 초기화 모듈을 수행하고자 할 때는 질의 부하가 없으므로 필요한 경로 집합은 정의 3에 의하여 XML데이타에 존재하는 라벨들의 집합과 같다.

그림 4는 그림 1에 대한 $APEX^0$를 나타낸다. 위에서 언급한 바와 같이, APEX는 해쉬 트리와 그래프 구조로 이루어져 있다. 따라서 $APEX^0$이 이러한 구조로 구성되어 있는 것이다.

그림 5의 $APEX^0$ 생성 알고리즘은 기본적으로 XML 그래프의 노드들을 깊이 우선 탐색(depth first traversal) 방식으로 탐색하면서 각 라벨에 따라서 간선(edges)을 분류(grouping)하여 익스텐트를 생성하고 이를 위한 노드를 그래프 구조에 생성하고 이를 해쉬 트리에 등록하는 기능을 수행한다. 따라서, $APEX^0$를 생성하는 비용은 O(E), E는 주어진 XML 그래프의 간선

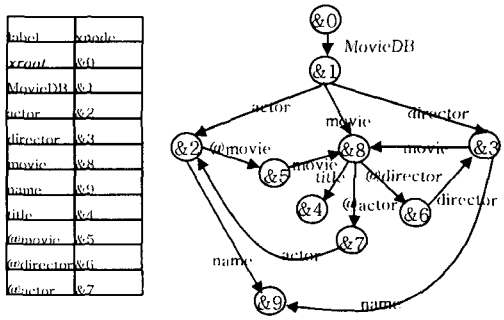


그림 4 $APEX^0$의 예

```
Procedure buildAPEX0(root)
begin
1. xnode := hash('xroot')
2. xnode.extent := {< NULL, root >}
3. exploreAPEX0(xnode, xnode.extent)
end
Procedure exploreAPEX0(x, ΔESet)
begin
1. ESet := ∅
2. for each < u, v >∈ ΔESet do
3.    ESet := ESet ∪ {o | o is an outgoing edge from v}
4. for each unique label l in ESet do {
5.    y := hash(l)
6.    if (y = NULL) {
7.       y := newXNode()
8.       insert y into hash table
9.    }
10.   make_edge(x, y, l)
11.   ΔnewESet := a set of edges having l in ESet - y.extent
12.   y.extent := y.extent ∪ ΔnewESet
13.   exploreAPEX0(y, ΔnewESet)
14. }
end
```

그림 5 APEX^0 생성 알고리즘

수, 임을 쉽게 알 수 있다. APEX^{0}를 생성한 후에는 질의 부하로부터 자주 사용되는 경로 정보를 추출하여 적응형 경로 인덱스를 증가적으로 갱신(update)한다. 이렇게 함으로서 현재 사용되고 있는 XML 질의를 보다 효율적으로 지원할 수 있도록 한다.

4.2 자주 사용되는 경로의 추출

APEX^{0}를 포함한 주어진 APEX를 현재 주로 사용되고 있는 질의 패턴에 맞도록 갱신하기 위하여서는 주어진 질의 부하로부터 자주 사용되는 경로를 추출하여야만 한다. 이러한 기능을 자주 사용되는 경로 추출 모듈에서 수행하게 된다. 이때 자주 사용되는 경로 추출 모듈은 APEX의 해쉬 트리를 변경하게 된다. 해쉬 트리의 각 노드는 해쉬 테이블을 지니고 있으며, 해쉬 테이블의 각 엔트리(entry)의 구조 다음의 5개의 필드로 구성되어 있다 : *label, count, new, xnode, next.* label은 각 엔트리의 키 값이 된다. count는 해당 엔트리가 표현하는 경로의 사용 빈도 즉 지지도를 표현한다. new 필드는 해당 엔트리가 새로 생성된 것인지 아닌지를 표현한다. xnode는 적응형 경로 인덱스의 그래프 구조에 있는 하나의 노드를 가리킨다. 마지막으로 next 필드는 해쉬 트리의 다음 노드를 가리킨다.

다음은 자주 사용되는 경로 추출 모듈에 대해 설명한다. 이때, 비단조(anti-monotonicity) 특성을 전지(pruning) 단계에서 사용하는 순차적 패턴 마이닝 기법들[12,13]을 자주 사용되는 경로 인덱스 추출에 활용할 수 있을 것이다. 그러나, 기존의 순차적 패턴 마이닝 기법들의 대상인 시퀀스(sequence)의 특성이 경로와는 다르기 때문에 수정이 필요하다. 예를 들어, 시퀀스(A, B, C)가 자주 나타난다고 할 경우, (A, B, C)의 하위 시퀀스(A, B), (B, C), (A, C)도 자주 나타난다고 할 수 있다. 그러나, 경로 A.B.C가 자주 사용된다고 해서 A.C가 자주 사용된다고 할 수는 없다. 왜냐하면 A.C는 A.B.C의 하위 경로가 아니기 때문이다.

```
Procedure frequentlyUsedPathExtraction()
begin
1. reset all count fields to 0 and new fields to FALSE
2. frequencyCount()
3. pruning H_APEX(HashHead)
end
Function pruning H_APEX(hnode)
begin
1. is_empty := FALSE
2. if hnode = NULL return is_empty
3. for each entry t ∈ hnode do {
4.    if (t.count < minsup) {
5.       t.next := NULL
6.       if (t ∉ HashHead) {
7.          is_empty := hnode.delete(t)
8.       }
9.    } else {
10.      if (pruning H_APEX(t.next) = TRUE)
11.         t.next := NULL
12.      if (t.next ≠ NULL) and (t.xnode ≠ NULL)
13.         t.xnode := NULL
14.      if (t.new = TRUE) and (hnode.remainder.xnode ≠ NULL)
15.         hnode.remainder = NULL
16.   }
17. }
18. return is_empty
end
```

그림 6 자주 사용되는 경로 추출 알고리즘

따라서, 본 연구에서는 그림 6과 같이 자주 사용되는 경로 추출 알고리즘을 개발하였다. 그림 6에서 보인 바와 같이 자주 사용되는 경로 추출 알고리즘은 지지도를 산출하는 부분과 전지 부분으로 이루어져 있다. 해쉬 트리는 XML 질의들의 사용 패턴의 변화를 유지하는데 사용되며, 자주 사용되는 경로의 추출 단계가 종료된 후에 해쉬 트리는 주어진 XML 경로 표현식들 중에서 필요한 경로들에 대한 정보만을 유지하게 된다.

우선, 사용빈도(지지도)를 산출하는 부분에서는 경로 표현식의 집합에서 나타난 모든 경로의 사용빈도를, 즉 자주 사용되는 경로를 추출하기 위해서 적응형 경로 인덱스를 구성하는 중요한 자료 구조인 해쉬 트리를 이용한다. 이때, 현재 사용되어진 XML 경로 표현식들의 사용 빈도를 계산하기 위하여 해쉬 트리 내에 존재한 모든 엔트리의 count와 new 필드를 0과 false로 설정한다. 그리고 주어진 XML 경로 표현식 집합인 Qworkload에 나타나는 모든 경로들의 사용빈도를 계산한다. 이 단계에서, 새로운 엔트리가 해쉬 테이블에 생성되면 해당 엔트리의 new 필드를 true로 설정한다.

전지 단계에서는 주어진 사용자 정의 최소 지지도, minSup 보다 작은 지지도를 지닌 엔트리들을 모두 제거한다. 이때 순차적 패턴 마이닝과 같이, 임의 경로 p의 sup(p)가 minSup 보다 작다면 p를 포함하는 모든

경로들의 sup(p)는 minSup보다 작다는 비단조 특성을 활용하여, 보다 효율적으로 자주 사용되지 않는 경로 정보를 해쉬 트리에서 제거한다.

4.3 APEX의 갱신

주어진 경로 표현식으로부터 자주 사용되는 경로들에 대한 정보를 추출하였으면 이를 이용하여 현재의 적응형 경로 인덱스를 갱신하여야 한다. 이를 위하여 XML 그래프를 처음부터 다시 소모적으로 검색하여 새로운 적응형 경로 인덱스를 만들 수 있을 것이다. 그러나, 이러한 방법은 많은 시간과 비용을 요하는 작업이므로 본 연구에서는 그림 7의 갱신 알고리즘을 이용하여 인덱스 갱신 비용을 감소시켰다.

이 갱신 알고리즘의 기본 방식은 적응형 경로 인덱스의 그래프 구조를 탐색하면서 그래프 구조의 각 노드들이 해쉬 트리와 관련되어 유효한가를 검사하고, 유효하지 않은 노드들을 제거하고 새로 생긴 엔트리에 대하여 노드들을 생성한다. 그리고 각 새로 생긴 노드들의 익스텐트를 생성한다.

이해를 돕기 위하여, APEX 갱신의 기본 동작 원리를 설명하도록 하겠다. 우선 APEX 갱신 모듈이 APEX의 그래프 구조에 있는 임의 노드 x를 경로 $path_l$와 간선 집합 △ESet을 가지고 방문하였다고 하자. 여기서

```
Procedure updateAPEX(xnode, ΔESet, path)
begin
1.  if (xnode.visited = TRUE) and (ΔESet = ∅) return
2.  xnode.visited := TRUE
3.  EdgeSet := ∅
4.  if ΔESet = ∅ {
5.    for each e that is an outgoing edge of xnode do {
6.      newpath := concatenate(path, e.label)
7.      xchild := lookup(newpath)
8.      if (xchild = NULL) xchild := newXNode()
9.      if xchild != e.end {
10.       if EdgeSet = ∅ {
11.         for each < u, v >  ∈ xnode.extent do
12.           EdgeSet := EdgeSet ∪ {o | o is an outgoing edge from v}
13.       }
14.       subEdgeSet := a set of edges with the label e.label in EdgeSet
15.       ΔEdgeSet := subEdgeset - xchild.extent
16.       xchild.extent := xchild.extent ∪ ΔEdgeSet
17.       make_edge(xnode, xchild, e.label);
18.       hash.append(newpath, xchild);
19.     }
20.     else ΔEdgeSet := ∅
21.     updateAPEX(xchild, ΔEdgeSet, newpath);
22.   }
23. } else {
24.   for each < u, v >  ∈ ΔESet do
25.     EdgeSet := EdgeSet ∪ {o | o is an outgoing edge from v}
26.   for each unique label l in EdgeSet do {
27.     newpath := concatenate(path, e.label)
28.     xchild := lookup(newpath)
29.     if (xchild = NULL) xchild := newXNode()
30.     subEdgeSet := set of edges labeled l in EdgeSet
31.     ΔEdgeSet := subEdgeset - xchild.extent
32.     xchild.extent := xchild.extent ∪ ΔEdgeSet
33.     make_edge(xnode, xchild, l)
34.     hash.append(newpath, xchild)
35.     updateAPEX(xchild, ΔEdgeSet, newpath);
36.   }
37. }
end
```

그림 7 APEX 갱신 알고리즘

△ESet은 노드 x의 방문하기 전에 APEX 갱신 모듈에 의하여 x의 익스텐트에 새롭게 추가된 간선들의 집합으로 x의 하위 노드들을 갱신하는데 필요한 정보로 활용되어진다. 만약 x가 이전에 방문되어진 노드이고 △ESet이 공집합이면 APEX 갱신 모듈은 어떠한 작업도 수행하지 않는다.

만약 x가 새롭게 방문된 노드이고 △ESet이 공집합이면 x노드의 모든 하위 노드를 해쉬 트리를 이용하여 검증하기 위하여 x로부터의 모든 출발 간선(outgoing edge)들을 탐색한다.

또한, x가 이전에 방문되었건 새롭게 방문되었건 간에 △ESet이 공집합이면 아니면, APEX 갱신 모듈은 x의 익스텐트의 변화(즉, △ESet)에 따라서 x를 루트로 하는 모든 서브 노드들을 갱신한다.

APEX의 갱신 부담은 질의 부하로부터 추출되어진 자주 사용되는 경로들에 주로 영향을 받는다. 페이지 15, 16페이지에서 언급하였듯이, 주어진 질의 부하가 거의 변화되지 않아서 자주 사용되는 경로들의 변화가 없을 경우에는 자주 사용되는 경로들의 변화가 없다는 것을 파악하기 위하여 $O(E_{APEX})$, E_{APEX}는 APEX 그래프 구조의 간선 수,의 비용이 필요함을 알 수 있다. 그러나, 이 경우, 4.2절의 자주 사용되는 경로 추출 기법에서 새로운 자주 사용되는 경로가 추출되었는지 아닌지를 파악할 수 있기 때문에, 새로운 자주 사용되는 경로들이 추출되었을 때만 APEX의 갱신을 수행하도록 하여 위에서 언급한 비용을 제거할 수 있다. 임의의 경로 $P = p_1.p_2.\cdots.p_i.p_{i+1}$이 새로운 자주 사용되는 경로라고 하자. 이 경우 $P' = p_1.p_2.\cdots.p_i$을 위한 노드(xnode) 및 익스텐트 변화량(△Eset)은 그림 7에서 예시한 바와 같이 이미 계산되어져 있다. 이때, 그림 7의 라인 14 및 30에서 보듯이 라벨 p_{i+1}을 가지는 간선 집합 중에서 임의의 간선들을 추출하게 된다. 라인 7의 경우에는 P' 위한 익스텐트에 속한 간선들의 끝 노드를 시작 노드로 하면서 라벨이 p_{i+1}인 간선들을 계산하고, 라인 30의 경우 ΔEset에 속한 간선들의 끝 노드를 시작 노드로 하면서 라벨이 p_{i+1}인 간선들을 추출하게 된다. 여기서, 각 간선들이 라벨이 따라서 클러스터링 되어 있다고 가정한다면, 이를 위한 비용이 sort merge 조인을 이용할 경우 $O((Ep_i+Ep_{i+1})\log(Ep_i+Ep_{i+1}))$, Ep_i는 p_i를 라벨로 가지는 간선들의 수, 임을 알 수 있다. 이후에 P를 위한 익스텐드의 변화량을 계산하기 위하여 $O(Ep_{i+1})$의 부담이 따르게 된다. 따라서, 하나의 새로운 경로 P를 위한 APEX 갱신 부담은 $O((Ep_i+Ep_{i+1})\log(Ep_i+Ep_{i+1}))$임을 알 수 있다. 따라서, 새로운 자주 사용되는 경로가 k개 나타났다면 전체 APEX 갱신 부담은 $O(k*(Ep_i+Ep_{i+1})\log(Ep_i+Ep_{i+1}))$임을 알 수 있다. 또한, 위에서 각 간선은 라벨별

로 클러스터링 되어 있다고 가정하였다. 이것은 $APEX^0$의 기본 성질로서 APEX의 갱신을 위한 $APEX^0$를 생성하여 효율적으로 지원할 수 있다.

4.4 APEX를 이용한 질의 처리

본 연구에서 제안한 적응형 경로 인덱스인 APEX를 이용한 XML 질의 처리 방식은 다음과 같다. 임의의 경로 $p_{in} = l_i.\cdots.l_{n-1}.l_n$가 질의로 주어졌을 때, p_{in}를 이용하여 해쉬 트리를 검색한다. 만약 $p_{kn} = l_k\cdots l_n$ $(i < k \le n)$인 경로까지만 해쉬 트리에 존재한다면 이를 이용하여 p_{kn}의 간선 집합 $T(p_{kn})$을 구하고, 이와 같은 단계를 p 하위 경로 $p_{ij}=l_i.\cdots.l_j$ $(i \le j < n)$의 정보가 해쉬 트리 상에 존재할 때, 즉 $T(p_{ij})$를 구할 때까지 반복한다. 그리고 얻어진 간선 집합 $T(p_{ij}),\cdots,T(p_{kn})$에 대하여 조인 연산을 하여 $T(p_{in})$을 구할 수 있다.

또한, 복합 경로 질의 $l_i.*.l_j$와 같은 것은 질의 전지 및 변환 기법 [14,15]을 적용하여 $l_i.l_{i+1}.\cdots .l_j$ 형태의 단일 경로 집합으로 변환 한 다음 질의 결과를 구할 수 있다. "임의의 경로 표현식을 만족하는 노드들 중 임의 조건을 만족하는 노드들을 추출하라"와 같은 값-경로 혼합 질의에 대하여서는, 임의 경로를 만족하는 노드들을 APEX를 이용하여 추출하고 그 다음 임의 조건을 만족하는지는 검사하는 것과 같은 방식, 또는 XML 데이타에 존재하는 노드들 중 임의 조건을 만족하는 노드들과 임의 경로 표현식을 만족하는 노드들을 구한 후 두 노드 집합의 교집합을 구하는 방식 등 다양한 방법을 이용하여 구할 수 있다.

만약 p_{in}가 필요한 경로라면 해쉬 트리에 p_{in}을 서픽스로 하는 경로 q가 해쉬 트리에 존재하며, 각 q에 대응되는 엔트리의 노드 포인터가 가리키는 그래프 구조의 노드들의 익스텐트 $E(q)$들에 대하여 $T(p_{in}) = \cup_{\forall q}E(q)$ 결과를 얻을 수 있다.

5. 실험 및 분석

5.1 실험 환경

본 실험은 MS-Windows 2000 환경의 512 MByte 메인 메모리를 가진 Pentium III-866MHz 에서 수행되었다. XML 질의에 대한 APEX의 효율성을 측정하기 위하여 여러 가지 형태의 XML 질의와 여러 가지 XML 데이타에 대하여 실험하였다. 실험 데이타로는 실제 XML 데이타 및 XML 데이타의 구조를 정의하는 실제 DTD 를 이용하여 생성한 3가지 XML 데이타를 이용하였으며, 이는 다음과 같다.

(1) Play : 셰익스피어 희곡들을 XML 형태로 변환하여 만든 XML 문서*[출처: http://www.oasis-open.org/cover/xml.html]*.

(2) FlixML : B급 영화 안내를 위한 해설용 문서에 사용되는 XML 문서로서 약간 복잡한 구조를 지님*[출처: http://www.xml.com]*.

(3) GedML : 계보학에서 사용되는 데이타를 표현하기 XML 문서로서 매우 복잡한 구조를 지님*[출처: http://www.oasis-open.org/cover/xml.html, 2001]*.

또한 다양한 크기의 XML 데이타에 대한 질의 성능을 측정하기 위하여 표 1과 같이 다양한 엘리먼트의 개수를 지니는 데이타를 생성하였다.

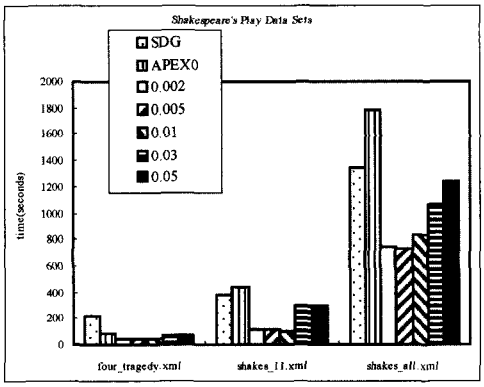
질의로는 $l_i.l_{i+1}.\cdots.l_n$과 같은 라벨의 연속인 5000개의 단일 경로 표현식(single label path expression)과 500개의 $l_i.*.l_j$ 형태의 복합 경로 표현식, 그리고 1000 개의 경로 표현식과 값의 검색이 같이 표현되는 값-경로 표현식을 사용하였다. 단일 경로 표현식의 XML 질의 표준인 XPath의 $//l_i/l_{i+1}/\cdots/l_n$으로 표현되며, 복합 경로 표현식은 $//l_i//l_j$로 표현되며, 값-경로 표현식은 $//l_i/l_{i+1}/\cdots/l_j$[text()=value]형태로 표현된다. 또한 APEX의 효율성을 기존의 경로 인덱스들 중에서 DataGuide(SDG)와 IndexFabic 그리고 초기 적응형 경로 인덱스($APEX^0$)와 비교하였다. 또한, 사용자 정의 지지도 minSup에 의하여 적응형 경로 인덱스의 형태가 달라질 수 있으므로, 실험에서도 minSup의 값을 0.002부터 0.05까지 변화시켜가면서 실험을 수행하였으며, 자주 사용되는 경로를 추출하기 위한 XML 데이타 질의 집합으로는 단일 경로 표현식 중 20%를 사용하였다.

5.2 실험 결과

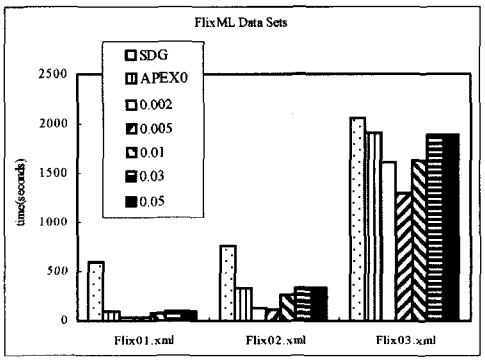
그림 8은 5000개로 이루어진 단일 경로 표현식을 각 경로 인덱스를 이용하여 처리하는데 걸리는 시간을 보여준다. 그림 8에서 보듯이, 약간 복잡하거나 매우 복잡한 구조의 XML 데이타에 대하여서는 strong DataGuide가 $APEX^0$보다도 비효율적임을 알 수 있다. strong DataGuide는 XML 데이타 상에 존재하는 루트 엘리먼트로부터 시작되는 단순 경로들만을 모두 유지하고 있기 때문에, 임의 엘리먼트로부터 시작되는 단일 경로 표현식을 처리하기 위하여서 strong DataGuide상에 존재하는 모든 노드들을 소모적으로 탐색하여야 한다.

표 1 XML 데이타 세트

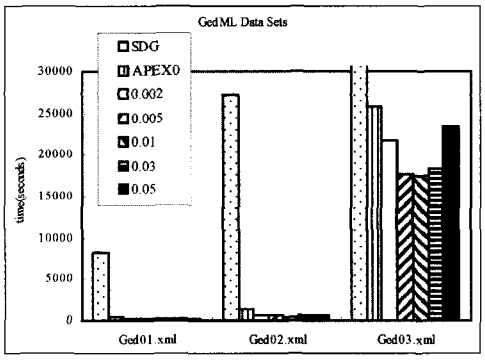
데이타 종류	Play			FlixML			GedML		
이름	four_tragedy.xml	shake_11.xml	shake_all.xml	Flix01.xml	Flix02.xml	Flix03.xml	Ged01.xml	Gedo2.xml	Ged03.xml
엘리먼트 수	22791	48818	179691	14734	41691	335401	8259	30875	381046



(a) PLAY



(b) FlixML



(c) GedML

그림 8 총 단일 경로 표현식 처리 시간

따라서, 주어진 XML 데이타의 구조가 복잡할수록 질의 처리 시간이 더욱 증가하게 된다.

APEX의 성능은 사용자 정의 최소 지지도 *minSup*에 영향을 받는다. minSup이 작아질수록 자주 사용되는 경로들의 개수가 증가하게 된다. 따라서, 많은 단일 경로 표현식들이 APEX의 그래프 구조의 탐색이 없이 해쉬 트리를 이용하여 바로 처리되어질 수 있게 된다.

그러나, 자주 사용되는 경로들의 개수가 증가한다고

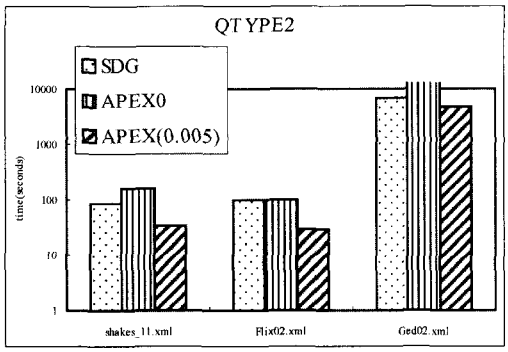


그림 9 총 복합 경로 표현식 처리 시간 [log scale]

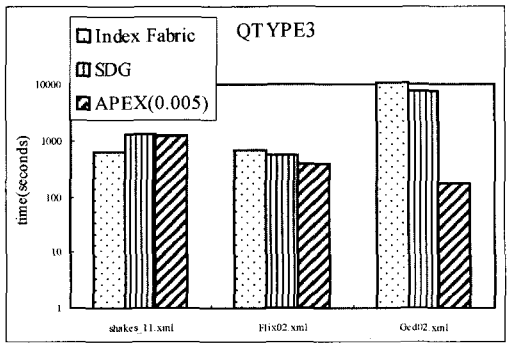


그림 10 값-경로 표현식 처리 시간

해서 항상 질의 처리 성능이 향상되는 것은 아니다. 이는 위에서 언급한 바와 같이 경로 인덱스의 구조가 복잡해짐에 따라서 검색하여야 할 영역이 증가하여 성능이 저하될 수 있기 때문이다. 본 실험에서는 평균적으로 minSup이 0.005일 때의 적응형 경로 인덱스가 가장 좋은 성능을 나타냈다.

그림 9는 500개의 복합 경로 표현식을 처리하는 걸리는 시간을 보여준다. 위에 언급한 바와 같이, 복합 경로 표현식을 처리하기 위하여서는 질의 전지 및 변환 기법을 수행하여야 한다. 이때, APEX는 strong DataGuide에 비하여 간단한 구조를 지니고 있으므로 보다 빨리 질의 전지 및 변환 기법을 수행하게 된다. $APEX^0$의 경우에는 질의 전지 및 변환 기법을 통하여 생성된 단일 경로식 집합을 수행하기 위하여 많은 조인 연산을 수행하게 된다. 따라서, 질의 수행 시간이 매우 커진다. Ged02.xml의 $APEX^0$의 복합 경로 표현식 처리 시간은 21000초에 달한다.

그림 10은 1000개의 값-경로 표현식을 처리하는 데 걸리는 시간을 보여준다. 이 실험에서는, 2절에서 언급한 바와 같이, 값-경로를 함께 암호화하여 색인화하는 Index Fabric과의 성능을 비교하였다.

그림 10에서 Index Fabric과 strong DataGuide는 루

트부터 시작되는 모든 단순 경로만을 기록하기 때문에, 값-경로 표현식을 처리하는데 있어서도 APEX가 더욱 효율적이다. 결론적으로 XML 데이타의 구조가 복잡할수록 더욱 APEX가 효율적임을 알 수 있으며, 다양한 minSup들 중에서 평균적으로 minSup이 0.005일 때 APEX가 가장 좋은 성능을 나타냈다.

6. 결 론

본 연구에서 우리는 XML 데이타를 위한 적응형 경로 인덱스인 APEX를 제안하였다. APEX는 기존의 경로 인덱스들과는 달리 질의 부하와 XML 데이타의 구조를 함께 활용하여 인덱스를 생성한다. 또한 기존의 경로 인덱스들은 루트부터 시작되는 모든 단순 경로들을 색인화 하는데 반하여 APEX는 질의 부하로부터 자주 사용되는 경로들을 추출하고 이를 이용하여 색인화한다.

또한, 효율적인 질의 처리를 위하여 APEX는 그래프 구조와 해쉬 트리로 구성되어 있다. 그래프 구조는 주어진 XML 데이타의 구조 정보를 표현하며, 해쉬 트리는 자주 사용되는 경로 정보를 관리한다. 이때, 해쉬 트리 상의 각 노드에 존재하는 엔트리들은 그래프 구조의 각 노드를 가리키고 있다.

APEX와 기존의 경로 인덱스들과의 성능 평가에서는 본 연구에서 제안한 APEX가 다양한 형태의 경로 표현식을 처리하는 데 있어서 월등히 뛰어난 성능을 보여주었으며 XML데이타의 구조가 복잡할수록 보다 좋은 성능을 나타냄을 알 수 있다.

위에서 설명한 바와 같이 본 연구에서 제안한 APEX를 이용한 XML 데이타의 검색 시에 자주 사용되는 경로를 활용하여 보다 효율적으로 XML 질의 결과를 찾아낼 수 있다. 또한 APEX는 XML 질의 패턴이 바뀌는 경우에 이를 반영하여 인덱스를 갱신하는 방안을 제시하고 있다. 따라서, 본 연구의 결과는 전자 상거래, 인터넷 문서 검색 등 XML 데이타 응용 분야에 크게 기여할 것이다.

본 연구의 추후 연구 방향은 제안되어진 적응형 경로 인덱스, APEX를 이용한 질의 처리시의 비용 모델(cost model)을 고안하고 이를 이용하여 질의 처리 시 보다 효율적인 질의 계획을 수립하는 질의 최적화 방안에 대한 연구이다.

참 고 문 헌

[1] B. Bray, J. T. Bray, J. Paoli, C. M. Sperberg-McQueen and E. Maler, "Extensible Markup Language(XML) 1.0," W3C Recommendation, http://www.w3.org/TR/REC-xml, 1998.

[2] P. Buneman, S. Davidson, M. Fernandez, and D. Suciu, "Adding Structure to Unstructed Data," In Proceedings of ICDT, pp.336~350, 1997.

[3] J. Clark and S. DeRose, "XML Path Language (XPath) Version 1.0," W3C Recommendation, 1999. http://www.w3.org/TR/xpath

[4] D. Chamberlin, D. Florescu, J. Robie, J. Simeon, and M. Stefanescu, "XQuery : A Query Language for XML," W3C Working Draft, 2001. http://www.w3.org/TR/xquery

[5] S. Nestorov, J. Ullman, J.Wiener, and S. Chawathe, "Representative Objects: Concise Prepresentation of Semistructured, Hierarchical Data," In Proceedings of IEEE ICDE, pp.79~90, 1997.

[6] B. Cooper, N. Sample, M. Franklin, G. Hjaltason, and M. Shadmon, "A Fast Index for Semistructured Data," In Proceedings of VLDB, 2001.

[7] R. Goldman and J. Widom, "DataGuides: Enable Query Formulation and Optimization in Semistructured DataBases," In Proceedings of VLDB, pp.436~445, 1997.

[8] T. Milo and D. Suciu, "Index Structures for Path Expression," In Proceedings of ICDT, pp.277~295, 1999.

[9] J. Hopcraft and J. Ullman, Introduction to Automata Theory, Language, and Computation, Addison-Wesley, 1979.

[10] A. Kemper and G. Moerkotte, "Access Support Relations: An Indexing Method for Object Bases," Information Systems, 17(2), 1992.

[11] S. Agrawal, S. Chaudhuri, and V. Narasayya, "Automated Selection of Materialized Views and Indexes for SQL Databases," In Proceedings of VLDB, pp.496~505, 2000.

[12] R. Agrawal and R. Srikant, "Mining Sequential Patterns," In Proceeding of IEEE ICDE, pp.3~14, 1995.

[13] M. Garofalakis, R. Rastogi, and K. Shim, "SPIRIT: Sequential Pattern Mining with Regular Expression Constrains," In Proceedings of VLDB, pp.223~234, 1999.

[14] M.Fernandez and D. Suciu, "Optimizing regular path expression using graph schema," In Proceedings of IEEE ICDE, pp.14~23, 1998.

[15] J. McHugh and J. Widom, "Compile-Time Path Expansion in Lore," In Proceedings of the Workshop on Query Processing for Semistructured Data and Non-Standard Data Formats, 1999.

민 준 기

1995년 숭실대학교 전자계산학과(학사) 1997년 한국과학기술원 전산학과(석사) 2002년 8월 한국과학기술원 전산학전공(박사). 2003년 8월 한국과학기술원 박사 후과정. 현재 한국과학기술원 전산학 전공 초빙교수. 관심분야는 데이타베이스, XML, 시공간 DB, OLAP

심 규 석

1986년 서울대학교 전기공학과 수석 졸업(학사). 1998년 University of Maryland, College Park(석사). 1993년 University of Maryland, College Park(박사). 1994년~1994년 Federal Reserve Board, Research Staff. 1994년~1996년 IBM Almaden Research Center, Restearch Staff. 1996년~2000년 Bell Laboratories, Member of technical Staff. 1999년~2002년 KAIST 전산학과 조교수. 2001년~2002년 Microsoft Research, Visiting Scientist. 2002년~현재 서울대학교 전기컴퓨터 공학부 부교수. 현재 VLDB저널과 KAIS저널의 Editorial Board ACM SIGKDD Curriculum Committee Member ACM SIGMOD, VLDB, ICDE, ACM SIGKDD, PAKDD, ICDM, EDBT을 비롯 다수의 국제 학술대회의 Program Committee Member임. 관심분야는 데이타마이닝, 데이타베이스, XML, Stream Data

정 진 완

1973년 서울대학교 공과대학 전기공학과(학사). 1983년 University of Michigan 컴퓨터공학과(박사). 1983년~1993년 미국 GM 연구소 전산과학과 선임연구원 및 책임연구원. 1993년~현재 한국과학기술원 전산학과 부교수 및 교수. 관심분야는 XML, 멀티미디어 데이타베이스, GIS, 웹 정보검색, 객체지향 데이타베이스