이화여자대학교 과학기술대학원
2004학년도
석사학위 청구 논문

XML 문서의 내용 기반 검색을 위한 영역-경로 인덱싱 기법

컴 퓨 터 학 과

최 은 혜

2 0 0 4

XML 문서의 내용 기반 검색을 위한 영역-경로 인덱싱 기법

이 論文을 碩士學位 論文으로 提出 함

2004 年 6 月

梨花女子大學校 科學技術大學院

컴퓨터學科 崔 恩 慧

최 은 혜 의 碩 士 學 位 論 文 을 認准 함

指導教授 용 환 승 ________

審査委員 김 명 ________

반 효 경 ________

용 환 승 ________

梨花女子大學校 科學技術大學

목　　　차

그 림 목 차

표 목 차

I. 서 론

1.1 연구 배경

최근 XML[1]은 인터넷 상의 어떠한 애플리케이션들도 쉽게 통합할 수 있으므로 다양한 타입의 데이터를 쉽게, 일반 텍스트들처럼 자유롭게 표현할 수 있으면서 환경에 중립적인 특성을 가지고 있다. 또한 XML은 계층적으로 중첩되어 있는 엘리먼트들의 집합으로 구성되고 기존의 문서와는 달리 하나의 문서에 내용 정보와 구조 정보를 모두 포함하고 있다. 따라서 이러한 XML 문서를 검색하기 위하여 순수 XML 데이터 순회를 위한 대표적 표준인 XPath[2], XQuery[3] 와 같은 질의 언어들은 구조와 내용으로 이루어지는 경로 기반 질의 표현을 제공하는데, 이러한 경로 표현식을 처리하기 위해서는 구조와 내용 모두에 대해 조인 계산이 많이 요구된다.

XML 문서는 구조도 중요하지만 텍스트를 기반으로 하기 때문에 정보 검색 시스템에서와 같이 내용에 대한 검색을 많이 요구하므로 내용을 빨리 검색할 수 있는 인덱싱 기법이 필요하다. 이를 위해서는 XML 문서에 대한 구조만을 대상으로 하는 빠른 검색 지원 뿐 아니라 내용에 대한 빠른 검색을 지원할 수 있는 기법이 요구된다.

그러나 현재까지 빠른 검색을 위해 개발된 인덱싱 기법들은 XML 문서로부터 추출된 스키마를 저장 시에 텍스트 값에 대한 내용 정보를 손실하거나 표현이 가능하더라도 문서의 상당 부분을 검색하여야 하므로 실행 비용이 매우 높아지게 된다[9,15]. 그러므로 주로 XML 문서의 구조를 기반으로 하는 연구에 집중되어 있다[11,12,13,14]. 이러한 구조를 기반으로 하는 연구는 주로 XML 문서에서 구조를 이루는 엘리먼트들을 조인하는데 걸리는 시간을 줄일 수 있는 방법을 제시하고 있는데, 특히 영역 인덱스 기법[5,6,7]을 사용한 연구는 XML 문서 전체를 파싱하지 않고도 조상 후손간의 관계를 빨리 파악함으로써 효율적인 검색을 지원하고 있다.

그러나 이러한 모든 현재까지의 방법들은 조상/후손/형제 관계를 요구하는 형태의 질의를 지원할 수 있는 융통적이고 효율적인 인덱스 구조를 갖지 못하여 내용을 검색하고자 할 때 효율적인 기법을 제공하지 못하거나, 이러한 관계를 검색하기 위해 복잡한 연산을 수행하여 구조 깊이가 깊어질수록 질의에 관련되지 않은 부분도 같이 검색하는 등 성능 저하를 유발하고 있다[4,5,9]. 또한 XML 문서에 대한 내용 삽입 및 삭제와 같은 구조적인 변경이 자주 발생하는 환경에서는 위의 방법들은 구조적 정보를 포함하는 엘리먼트 식별자에 대한 많은 재할당이 요구된다[16]

XML 문서의 특성을 볼 때 XML 문서에 대한 내용 질의, XML 문서가 갖는 논리적인 계층 구조를 이용한 질의, 엘리먼트가 갖는 속성에 대한 질의 등을 모두 수용할 수 있어야 한다. 그러므로 기존 연구들의 미흡한 부분들을 해결하여 다양한 XML 문서의 질의를 효과적으로 지원하는 새로운 인덱싱 기법에 대한 연구가 필요하다. 그리고 동일한 기준을 사용한 성능 평가를 통해 제안한 인덱싱 기법의 효율성을 검증하는 과정도 필요하다.

1.2 연구 목적 및 내용

본 연구의 목적은 기존의 인덱싱 기법과의 차이를 비교·분석하고 성능을 평가함으로써 미흡한 부분들을 해결하여 XML 데이터를 빠르고 효율적으로 질의할 수 있는 새로운 인덱싱 기법을 제안하는 것이다.

기존의 연구들은 다양한 유형의 질의를 지원할 수 있는 융통적이고 효율적인 인덱스 구조를 갖지 못하여 내용을 검색하고자 할 때 효율적인 기법을 제공하지 못하거나 복잡한 구조적 조인 및 연산을 많이 요구하여 성능의 비효율성을 초래한다.

따라서 본 연구에서는 XML 문서의 내용 정보와 구조 정보를 손실하지 않고 인덱싱하기 위해 깊이 우선 탐색(Depth First Search) 방식으로 XML 문서를 탐색하면서 엘리먼트와 애트리뷰트에 대하여 각각의 영역을 할당한다. 이때 각 노드마다 조상의

시작 영역(start region)을 접두어로하여 이들을 연결한 구조 정보를 부여함으로써 XML 문서의 각 구성 요소간의 부모 자식 관계 뿐 아니라 해당 구성 요소에 대한 형제 관계 및 조상 후손에 관한 계층 정보를 쉽게 파악할 수 있도록 하였다. 또한 XML 문서 구조를 이루는 각각의 구성요소에 빠르게 접근하기 위해 인덱스를 조작하는 방법으로 가장 많이 사용되는 트리인 B+ tree를 이용하였다. 이 B+ tree의 단말 노드에는 같은 노드 이름을 가진 구성 요소들에 대해 이들의 경로 정보를 담고 있는 집합을 가리킬 수 있는 포인터를 추가하였다.

또한, 기존의 연구에서는 내용에 대한 인덱싱 기법을 제공한다할지라도 하나의 프리디킷을 처리하는 방법[6,10]만을 보여주고 있는 것에 반해 본 연구는 임의의 개수의 프리디킷을 효과적으로 처리할 수 있도록 하였다. 그리고 내용과 구조에 대한 인덱스를 따로 관리하지 않고 혼합한 인덱스 구조를 가지므로 기존 연구에 비하여 다양한 질의에 빠르게 응답하고 데이터 저장 공간의 효율성을 제공할 수 있다. 이렇게 XML 문서의 구조 뿐 아니라 내용에 대하여 인덱싱을 하는 우리의 기법은 기존의 검색 방식에 비해 여러 가지 유형의 질의를 지원할 수 있도록 한다. 또한 사용자가 XML 문서에 대한 구조적 정보를 정확히 표현하지 않아도 제시한 조건 정보를 기반으로 공통 조상을 파악하여 질의에 응답하기 때문에 문서 구조를 정확히 모르는 사용자에게 편의를 제공한다.

우리의 인덱싱 기법은 구조화된 데이터 및 반구조화된 데이터를 보유한 기업 및 시스템 개발자들에게 데이터 관리의 처리 부문에 있어 용이하게 해줄것으로 전망한다.

마지막으로, 본 논문에서 제안한 인덱싱 기법의 효율성을 검증하기 위해 성능 평가를 수행한다. 인덱싱 기법의 효율성은 저장된 XML 데이터에 대해 내용 검색과 구조 검색을 혼합한 다양한 형태의 질의문에 대하여 얼마나 빠르게 질의에 응답하는가에 따라 평가된다. 이때의 XML 질의는 정규 경로식(regular path expression)을 사용하여 절대 경로 및 상대 경로를 표현하는 XPath 구문을 기반으로 하되 값에 대하여 하나 이상의 프리디킷을 가지는 혼합 유형의 질의들을 선정하였다. 실험 결과 본 논문의 인덱싱 기법을 이용한 결과가 XML 전용 데이터베이스로 유명한 X-Hive사의 X-Hive/DB보다

평균적으로 1.5배 이상의 비율로 좋은 성능을 나타냈음을 알 수 있다. X-Hive/DB는 질의의 경로가 긴 경우에도 본 인덱싱 모델에 비해 수행 시간이 상대적으로 오래걸렸고, 질의의 경로가 짧아질수록 경로 표현식이 많아질수록 질의 수행 시간이 크게 증가하였다. 따라서 제안한 인덱싱 기법의 효율성을 증명할 수 있다.

본 논문의 전체적 구성은 다음과 같다. 2장에서는 XML 문서에 대한 기존 연구들의 인덱스 기법의 특징 및 문제점에 대하여 언급하고, 3장에서는 내용 기반 검색을 지원하는 새로운 인덱싱 기법을 소개한다. 4장에서는 3장의 설계를 바탕으로 실제 구현된 인덱싱 기법을 이용한 질의 처리 방법을 보여주고, 5장에서는 본 논문의 성능 평가 작업을 위해 XML 전용 서버인 X-Hive/DB 와의 비교 실험결과를 제시하고, 6장에서는 결론 및 향후 연구에 대하여 기술한다.

Ⅱ. 관련 기술 및 연구동향

본 장에서는, XML 문서를 위해 지금까지 제안된 인덱싱 기법을 유형별로 분류하고 이들에 대해 그 특징 및 한계점에 대해 살펴보기로 한다.

2.1 인덱싱 유형

XML 문서는 트리형태로 모델링하여 트리 내의 각 구성 요소인 엘리먼트/애트리뷰트와 이들의 값에 대한 구조 파악이 가능하다. 이러한 특성을 가지는 XML 문서에 대하여 효율적인 질의 처리를 위한 인덱싱 기법들이 연구되고 있다. 인덱싱 스킴은 경로 정보를 요약하면서 질의 처리를 최적화하는 것으로써, 질의 처리 동안 스캔될 데이터 베이스 부분을 줄이는데 그 목적을 둔다.

인덱싱 방식을 살펴보면 트리의 루트부터 검색하는 절대 경로 처리 인덱싱 방식과 임의의 위치에서 접근하여 질의 처리를 지원하는 상대 경로 인덱싱 방식으로 크게 두 가지로 분류할 수 있다. 또한 상대 경로 인덱싱 기법에서는 B+ tree 의 이용 여부에 따라 세부 분류를 달리한다.

2.2 절대 경로 인덱싱

절대 경로 검색은 트리로 모델링된 문서에서 루트부터 검색하는 처리 기법을 이용한다. 이에 대한 초창기 연구들은 경로 표현식(path expression)을 통해 원하는 값을 찾는데 걸리는 시간을 줄이기 위해 XML 문서 구조에 대한 요약(Structural summary)[17,18,19]을 만들고, 이것을 이용하여 인덱싱을 하였다[14,17]. 이때의 구조적

요약은 반구조적 데이터나 XML 문서와 같은 구조화된 그래프 데이터에 대하여 경로 기반 질의를 쉽게 지원하기 위한 인덱스로 생각하면 된다. 이것은 데이터 그래프의 경로는 그대로 유지하되 중복되는 경로를 제거하고 문서에 대한 최소화된 스키마를 유지하는 방식으로써, 이를 이용한 대표적인 기법으로는 DataGuides 와 1-index 가 있는데, 후에 이러한 구조 검색 뿐 아니라 내용 검색도 같이 지원하기 위해 Index Fabric 방식이 제안 되었다.

2.2.1 DataGuides

DataGuides[13]는 Object Exchange Model(OEM) 데이터베이스에 저장하고 있는 반구조적 데이터(semistructured data)의 구조에 대한 요약 정보를 데이터 그래프로 제공한다. DataGuides 자신도 반구조적 데이터로 표현되므로 OEM 데이터베이스에 저장하고 있는 일반 데이터와 동일한 형식으로 저장할 수 있다. DataGuides는 반구조적 데이터의 요약 정보를 구성하며 이를 인덱스로 사용하는 것이 가능하므로 임의의 정규 경로 표현(regular path expressions)을 지원해주면서 반구조적 데이터에 대한 인덱싱 기법의 근간이 된다.

하지만, 기본적으로 논리적인 구조만을 제시하고 있으며, OEM 데이터베이스가 아닌 일반적인 관계형 데이터베이스나 OS 파일 시스템을 기반으로 시스템을 구성할 경우의 효율적인 처리 방안은 제시하고 있지 않고, 많은 사이클을 가진 불규칙한 데이터에 대해서는 노드 중복 문제가 발생하므로 인덱스로서 부적절하다. 또한 반구조적 데이터를 트리 형태로 표현한 데이터 그래프에서의 모든 경로를 정확히 인코딩하는 전체 경로 인덱싱(full path indexing) 기법으로서 임의의 지점에서 시작하는 질의 처리가 불가능하다는 제한점이 있다.

그러나 이 연구에서 제안한 인덱스의 논리적인 구조는 XML 문서의 구조를 이루는 모든 경로에 대한 최소화된 트리로서, 이후 이에 대한 많은 연구가 진행되어 왔다.

2.2.2 1-index

1-index[14]는 루트로부터 시작되는 경로의 집합이 동일한 노드들을 모아 인덱스를 구축한 것으로 절대 경로 인덱싱 기법이다. 즉, 데이터 그래프 내에 동일한 경로가 매우 많이 존재한다는 점을 이용하는 인덱스 기법이다.

그러나 참조(reference) 간선을 가지고 있거나 혹은 동일한 경로가 많지 않을 경우, 데이터 그래프에서 많은 경우 인덱스 그래프의 크기가 데이터 그래프의 크기와 유사하게 커져서 이러한 방법은 실용적이지 못하다. 또한 정규 경로식 등을 포함하거나 루트가 아닌 엘리먼트/애트리뷰트로 시작되는 질의 처리에 대해서는 언급하지 않고 있다.

2.2.3 Index Fabric

Index Fabric[8]은 문서 내의 여러 경로들을 각각의 알파벳으로 하는 문자열로 인코딩하여, 문자열 탐색에 유용한 구조인 Patricia trie에 최적화된 인덱스를 삽입시키는 효율적이고 융통적인 인덱스 구조이다.

Index Fabric은 길이가 긴 여러 문자열들을 인덱싱 하기위해 우선 데이터 값을 가진 각 태그(tag)마다 압축된 지정자(designator)를 부여한 후 이를 이용하여 루트부터 각각의 데이터 값까지의 경로를 모두 유지한다. 그래서 같은 뿌리의 경로 상에 있는 모든 지정자 정보와 데이터 값을 엮어서 하나의 문자열 키로 압축하여 Patricia trie 에 경로를 저장한다. 이때, 색인 탐색 시의 입출력 횟수를 줄이기 위해 Patricia trie에 대한 다단계 색인을 구축한다. 따라서 질의 평가시 Patricia trie 에 있는 인덱스 키를 활용하므로 트라이를 따라 네비게이션을 할 때 인덱싱된 엘리먼트가 수백만 개가 되더라도 질의에 응답하는데 기껏해야 몇 개의 인덱스 I/O만 필요하다.

또한 인코딩한 경로가 데이터 그 자체로부터 추출되기 때문에 XML 문서에 대한 사전 스키마 지식이 필요없다. 그리고 새로운 엘리먼트 및 애트리뷰트가 삽입될 때에는 구

조가 천천히 성장하고 삭제시에는 그 노드가 속한 하나의 블록에만 영향을 미치므로 데이터 구조가 변경되어도 높은 성능을 유지한다.

그러나 데이터 값을 갖지 않는 XML 엘리먼트들에 대한 정보는 유지하지 않기 때문에 엘리먼트들 사이의 부모-자식 관계는 알 수 없다. 그러므로, Index Fabric은 문자열의 완전한 매칭만을 지원하기 때문에 부분적인 매치를 요구하는 질의를 처리하는데에는 효율적이지 않고, 임의의 지점부터 검색하는 상대경로 검색('//')도 지원하지 않으며, 찾고자하는 엘리먼트까지의 완전한 경로가 질의에 명시되어야 한다. 그리고 문자열의 비교만을 고려하므로 다양한 문서 구조의 탐색을 지원할 수 없다.

2.2 상대 경로 인덱싱

루트부터 시작하는 모든 경로에 대한 정보를 저장하는 1-index 와 DataGuides와 같이 단순 트리 운행 방식에 의해서는 XML 문서에 대한 질의처리에 있어 정확한 경로식으로 기술해야하므로 전체 문서를 검색하는데 있어 실행 비용이 매우 높아지게 된다. 따라서 정규 경로식(regular path expression)을 사용하는 상대 경로 질의를 위해 여러 인덱싱 기법들이 제안되었고 대표적인 기법으로 A(k)-index와 D(k)-index가 있다.

또한 상대 경로 인덱싱 기법의 세부 유형으로 B+ tree 트리 기반의 인덱싱 기법들이 있다. 이러한 인덱싱 기법은 트리 형태의 XML 문서의 각 구성 요소간 구조 파악을 위한 '영역 인코딩 방식' 을 적용하는데, 이러한 인코딩 방식은 트리의 루트노드부터 순차적으로 탐색하는 깊이우선탐색 (depth-first traversal)을 통해 각 노드마다 고유의 영역(시작영역,끝영역)을 나타내는 숫자를 할당함으로써 노드간의 계층적인 정보 뿐 아니라 순서 정보까지 파악할 수 있다.

2.2.1 A(k)-index

DataGuides와 1-index의 경우, 루트로부터 시작하는 절대 경로에 대해서만 질의 처리가 가능하고, 정확한 구조 요약을 하기때문에 길고 복잡한 경로를 가지는 XML 문서의 경우, 인덱스 그래프의 사이즈가 매우 커진다. 또한 실제로 길이가 긴 질의문의 희귀성 때문에 이런 문제점들을 줄이기 위하여 임의의 K 길의의 유사성을 가진 경로에 대해서만 인덱싱을 하는 A(k)-index[19]가 제안되었다.

그러나 인덱싱을 원하는 경로의 개수 K를 미리 지정해야 질의 처리가 가능한 제한점이 있다. A(k)-index 성능은 K 파리마터의 선택에 달려있다. 따라서 질의 유형에 맞게 K 개수를 적절히 잘 선택해야하므로 다양한 경로의 질의문에 대해서는 인덱스를 각각 구성해야하므로 불필요한 저장 공간을 초래한다.

2.2.2D(k)-index

D(k)-index[11]는 1-index 와 A(k)-index 구조의 일반화로서 여기서는 두 종류의 업데이트 알고리즘에 초점을 두고 연구했다.

하나는 서브 그래프 추가로써, 1-index 에서의 업데이트 알고리즘의 변형으로 볼 수 있는데, 즉 새로운 서브 그래프를 기존의 데이터 그래프의 루트 아래에 삽입시키는 것이다.

두번째는 데이터 그래프에 대한 간선(edge)을 삽입으로써, 이러한 간선 추가는 1-index, A(k)-index 에서의 모든 자손들에게 영향을 미칠 수가 있다.

D(k)-index 는 인덱스 그래프에서 같은 이름을 가지는 구성 요소들의 클래스에 대해, 똑같은 로컬 유사도(local similarity)를 주지않고, 질의 로드에 따라 다이나믹하게 서로 다른 약 유사도(bisimilarity requirement)를 할당해줌으로써 그 구조를

융통성있게 조절하는 인덱스이다. 따라서 간선 추가에 있어 D(k)-index 의 업데이트 알고리즘은 효율적이라 할 수 있다.

그러나 질의 패턴이 변화하면서 질의에서 자주 사용되지 않는 경로에 대하여는 인덱스를 유지할 필요가 없다는 가정하에 같은 이름을 가진 엘리먼트들을 통합하여 인덱싱을 하기 때문에 이러한 구조적 요약은 그 정확성이 떨어진다는 단점을 가지고 있다. 그리고 질의에 따라 인덱스 구조가 실시간으로 업데이트 되지않고 독립적으로 수행되므로 즉각적인 질의 처리 속도는 빠르지 않다. 또한 D(k)-index 는 XML 문서에 대한 구조 검색에만 효율적이고 내용 기반 검색은 지원하지 않는다는 제한점이 있다.

2.2.3B+ tree 기반 인덱싱

[5,9] 기법은 XML 트리 내의 임의의 두 노드에 대해서 조상 후손의 관계를 알아낼 수 있는 방법을 제시하고 이를 이용해 A//B 형태의 질의를 최적화 하는 방법을 제안하고 있다. 이들 기법은 A/B 유형과 같은 단순 경로를 따라갈 때에 비효율적이라는 단점을 가지고 하나의 질의 내에 여러 개의 경로식이 포함된 질의를 최적화 하는 방법은 지원하지 않는 제한점을 가진다. 이러한 부분들을 고려하여 여러 기법들이 제안되고 있는데 그중에서도 [6.7]의 두가지 대표적 기법의 특징에 대해 설명하고 한계점을 언급한다.

가) [6]은 엘리먼트 집합에 이 엘리먼트의 영역을 나타내는 숫자(시작영역,끝영역)를 부여하고 이 시작 영역을 인덱싱을 하는 B+ tree 기반의 인덱스 구조를 제안하였다. 특히 조상-후손간의 구조 기반 검색의 효율을 높이기 위하여 기본 B+ tree에 포함(containment) 개념 기반의 형제 포인터(sibling pointers)를 추가하였다. 따라서 예를 들어, “Movies//Actor” 와 같은 질의문에 대하여 Movies 와 Actor 이외에 조인에 참여하지 않는 엘리먼트를 건너뛰는 탐색 전략을 가졌다.

기존의 B+ tree 의 경우에는, 질의에 이용되는 모든 관련된 페이지들이 메

모리 내에 있다 하더라도, 이들 페이지 내에서 새로운 조상 엘리먼트에 대해 이진탐색과 같은 검색방법을 필요로 하였다. 그러나 여기서의 형제 포인터를 이용하면, 그러한 수고가 필요없이 조인에 참여하지 않는 엘리먼트들은 건너뛸 수가 있으므로, 임의의 조상노드 아래에 있는 자손 노드까지로의 검색이 가속도를 낼 수 있다.

그러나 이 기법은 검색에 사용되지 않는 Movies와 Actor 사이의 스킵은 가능하지만 Movies의 상위 엘리먼트 스킵에 있어서는 효율성이 떨어지는 단점을 가진다.

나) 따라서 이러한 문제를 해결하기 위해 XR-Tree [7]가 제안되었다. 깊게 내포된 XML 데이터를 위해 특별히 디자인된 인덱스 트리로서 이 모델도 역시 B+ tree를 이용하되, 태그와 시작 영역에만 인덱싱을 하는 [6]과는 달리, 동일한 영역을 포함하고 있는 기본 엘리먼트 영역을 유일한 키(key)들로 묶은 스탭 리스트(stab list)와 포인터를 이용하여 검색을 수행하므로 기존 알고리즘에 비해 조인에 참여하는 엘리먼트 집합에 대한 선별력이 높다. 따라서 2개의 엘리먼트 집합들에 대한 구조적 조인시 이 XR-tree를 이용하여 필요없는 조상과 후손 엘리먼트 모두를 효율적으로 스킵할 수 있다

그러나 내용 검색을 지원하지 않고, 각각의 엘리먼트마다 B+ tree를 따로 구축해야하므로, 자주 나타나지 않는 엘리먼트 검색에 있어서는 효율이 떨어진다.

Ⅲ. 인덱싱 기법 설계

XML 문서의 추상화된 논리적 정보 표현의 기본 단위는 엘리먼트이다. 따라서 XML 문서는 특정 엘리먼트의 부모/자식, 조상/후손 관계에 대한 구조 정보 뿐 아니라 엘리먼트 및 애트리뷰트의 값에 대한 정보를 유지하고 있다. 이러한 XML 문서에 대하여 효율적인 검색을 위하여 그동안 많은 연구들이 진행되어 왔다. 그러나 기존의 연구들에서는 주로 순수하게 XML 문서의 구조만을 대상으로 하는 인덱싱 및 질의 처리 방법만을 고려하거나, 또는 엘리먼트/애트리뷰트의 값을 기반으로하는 내용 검색의 정확도를 높이기 위해 구조적 정보를 이용하는 방법을 제시하고 있으나 이러한 기법들은 일부 질의 유형에만 효율적이고 다양한 유형이 혼합된 질의문이나, 또는 사용자가 XML문서에 대한 구조 정보를 정확히 표현하지 않았을 경우에는 지원이 불가능하거나 질의 처리를 위하여 추가적인 연산이 많이 요구된다.

따라서 본 논문에서는 사용자 입장을 고려하여 XML 문서에 대한 내용 기반 검색을 효과적으로 지원하기 위해 엘리먼트/애트리뷰트의 이름과 이들의 값이 문서에서 위치하는 영역과 이들의 구조를 파악할 수 있는 ‘영역-경로 인덱스’ 라는 새로운 기법을 개발하였다. 영역-경로 인덱스는 XML 문서의 구성 요소인 엘리먼트와 애트리뷰트에 대하여 각각의 영역을 할당한 후, 이들 구성 요소에 대하여 그것의 경로에 대한 구조적 정보를 저장한 것이다. 이제부터 이러한 영역-경로 인덱스 생성 방법 및 이 영역-경로 인덱스를 이용하여 제안한 인덱싱 기법에 대하여 설명한 후 인덱스 구조에 대한 세부적인 모듈 및 알고리즘에 관하여 살펴본다.

3.1 영역-경로 인덱스

영역-경로 인덱스는 XML 문서를 트리로 모델링하고 이 트리에 나타나는 각 노드에

영역 정보를 할당하여 생성한다.

3.1.1 XML 문서 트리

XML 문서의 내용 정보 및 구조 정보를 나타내기 위해서는 엘리먼트 및 애트리뷰트 간의 부모/자식과 조상/후손과 같은 계층적 포함관계와 형제와 같은 수평적 관계를 모두 표현할 수 있는 모델링 과정이 필요하다. XML 문서는 엘리먼트를 노드로 표현하는 트리 형태로 나타낼 수 있는데 이를 XML 문서 트리로 정의한다.

그림 3-1은 본 논문에서 예제로 사용될 영화에 대한 정보를 담고있는 하나의 XML 문서의 DTD 및 인스턴스이다. DTD에는 문서의 논리적 구조와 규칙, 엘리먼트의 타입 등이 나타나 있다. 인스턴스의 각 구성요소는 선택적 혹은 반복적 요소로 정의되며, 컨텐츠는 #PCDATA 타입의 값을 가진다. 이러한 XML 문서를 파싱을 하면서 그림 3-2와 같이 트리로서 모델링한다. 이때 애트리뷰트와 엘리먼트에 대한 구별을 두지않고 애튜리뷰트도 상위 엘리먼트의 하위 엘리먼트로 취급하고, 실제 텍스트 데이터는 단말 노드로 표현하였다.

```
<?xml version ="1.0"?>
<!DOCTYPE Movies [
<!ELEMENT Movies (Movie*)>
<!ELEMENT Movie ( Title,Country,Genres,People)>
<!ATTLIST Movie Year  CDATA #REQUIRED>
<!ELEMENT Title (#PCDATA)>
<!ELEMENT Country (#PCDATA)>
<!ELEMENT Genres (Genre*)>
<!ELEMENT Genre (#PCDATA)>
<!ELEMENT People (Director,Actor)>
<!ELEMENT Director (#PCDATA)>
<!ELEMENT Actor (Name*)>
<!ELEMENT Name (#PCDATA)>
]>


<Movies>
  <Movie Year="1998">
     <Title>Shakespeare in Love</Title>
     <Country>America</Country>
     <Genres>
       <Genre>ScreenPlay</Genre>
     </Genres>
     <People>
       <Director>Mark Norman</Director>
       <Actor>
         <Name>Gyeenes Peltrow</Name>
       </Actor>
     </People>
  </Movie>
  <Movie Year="2000" >
     <Title>Erin Brockovich</Title>
     <Country>America</Country>
     <Genres>
       <Genre>Drama</Genre>
     </Genres>
     <People>
       <Director>Steaven Soderberghs</Director>
       <Actor>
         <Name>Julia Roberts</Name>
         <Name>Aarron Eckhart</Name>
       </Actor>
     </People>
  </Movie>
  <Movie Year="2000" >
     <Title>Erin Brockovich</Title>
     <Country>America</Country>
     <Genres>
       <Genre>Drama</Genre>
     </Genres>
     <People>
       <Director>Steaven Soderberghs</Director>
       <Actor>
         <Name>Julia Roberts</Name>
         <Name>Aarron Eckhart</Name>
       </Actor>
     </People>
  </Movie>
</Movies>
```

그림 3-1 예제 XML 문서

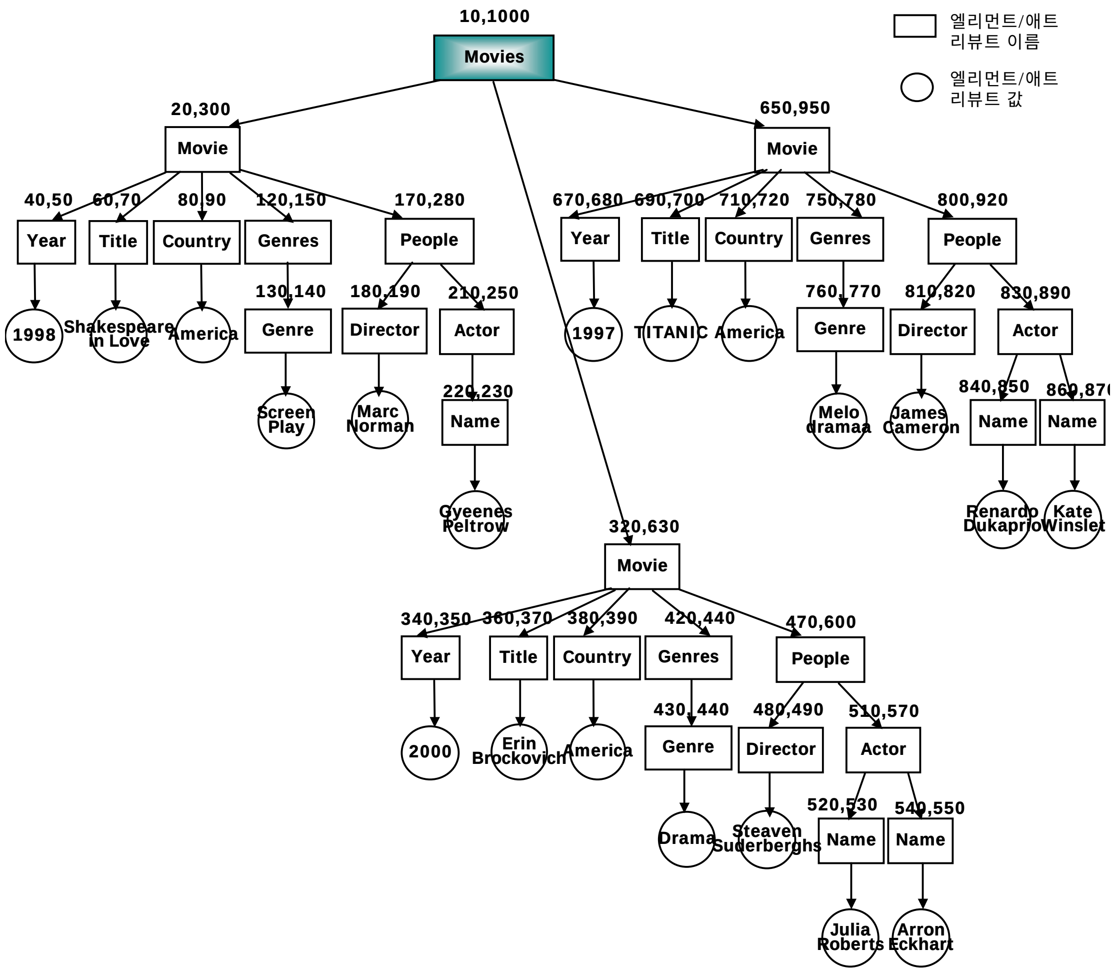

그림 3-2 XML 문서 트리

3.1.2 영역 정보 할당

트리로 모델링된 각 노드마다 시작 영역과 끝 영역에 해당하는 숫자를 부여한다. 이때 만일 구성 요소의 부분 삽입과 같은 업데이트가 발생할 경우 기존의 트리 구조에 커다란 영향을 주지 않기 위해 적당히 느슨한 간격으로 영역을 배당한다. 그림 3-2에서 보면, Movies 라는 루트 엘리먼트는 10에서 1000까지의 영역을 유지하고, 그것의 첫번째

자식 엘리먼트인 Movie는 10보다 훨씬 큰 20을 시작 영역으로 가지고 300이 끝 영역이다. 이때의 Movie 는 21부터 299까지의 자손 영역을 할당받았으므로 새로운 엘리먼트 삽입시 여유있게 대처할 수 있다. 예를 들어 현재 Movie의 첫번째 자식은 (40,50)의 영역을 가지는 Year 엘리먼트이지만 Time 이라는 엘리먼트를 추가하고자 할 경우 (30,35)의 영역을 할당하여 첫번째 자식 위치로 삽입이 가능하다.

또한 XML 문서 트리에서 텍스트 데이터에 해당하는 단말 노드는 자신의 직속 상위 노드의 영역 정보를 그대로 유지한다. 즉, 그림 3-2의 가장 왼쪽에 위치한 1998 노드는 상위 노드인 Year와 마찬가지로 (40,50)의 영역 정보를 가진다.

3.1.3 영역-경로 정보

이렇게 XML 문서의 각 노드마다 부여한 이 영역 정보를 이용하여 영역-경로 정보를 생성한다. 이 영역-경로 정보는 기존 연구들처럼[6,7] 노드의 시작 영역, 끝 영역 정보를 그대로 이용하지 않고, 문서의 구조 정보를 빠르게 검색하기 위하여 그림 3-3처럼 XML 문서 트리의 루트부터 자신 노드까지의 시작 영역 숫자를 접두어처럼 연결한다. 그리하여 문서 내에서 각 노드에 대하여 고유의 절대적인 구조 정보가 되도록 한다. 그림 3-4의 제일 왼쪽 노드인 Year의 경우 영역-경로 정보가 10.20.40이 되는데, 여기서 접두어 10에 해당하는 Movies가 Year의 최고 조상이고 10.20의 Movie가 그것의 부모 노드이다. 또한 형제 관계를 파악하기 위해서는 자신의 시작 영역인 제일 마지막 숫자를 제외한 접두어로 시작하는 노드를 검색하면 된다. 즉, 여기서는 10.20으로 시작하는 Title, Country, Genres, People 이 Year의 형제 노드임을 알 수 있다.

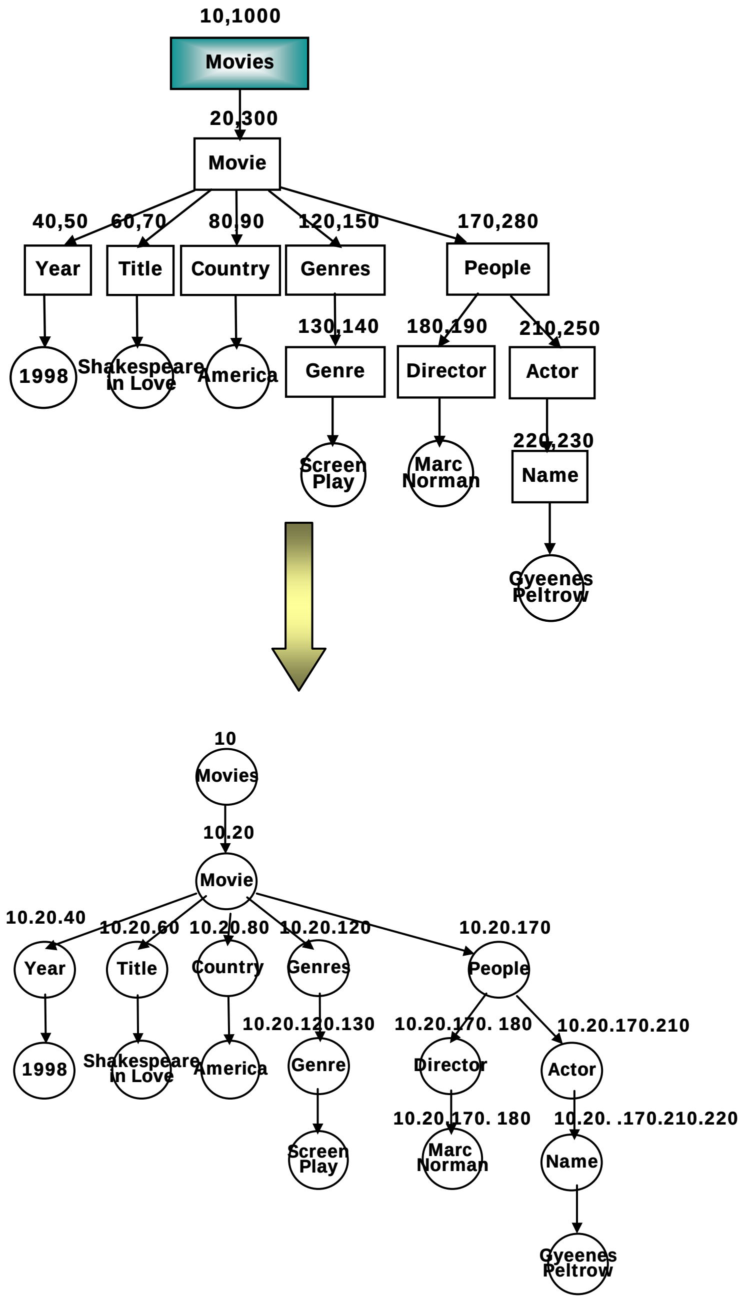

그림 3-3 영역-경로 정보

이렇게 XML 문서의 각 노드마다 부여한 이 영역 정보를 포함하는 그림 3-3의 XML 문서를 관계형 데이터베이스에 저장하면 실제 저장된 모습은 그림 3-4, 3-5와 같다. 필요한 테이블은 이름 테이블과 값 테이블로써 두가지인데, 엘리먼트 이름과 애트리

뷰트의 이름은 이름 테이블에 저장되고, 각각의 텍스트 내용은 값 테이블에 저장된다.

이들 스키마는 각각 다음과 같이 문서의 번호, 시작영역, 노드 내용, 영역-경로 정보로 이루어진다.

Doc_ID	Start_Region	Node_Name	Path_Info
1	10	MovieList	10
1	20	Movie	10.20
1	30	ID	10.20.30
1	40	Year	10.20.40
1	50	Title	10.20.50
1	60	Country	10.20.60
1	70	Staff	10.20.70
1	80	Scenarist	10.20.70.80
1	90	Directed_By	10.20.70.90
1	100	Director	10.20.70.90.100
1	110	Link	10.20.70.90.110
1	120	CreateYear	10.20.70.90.110.120
…	…	…	…

그림 3-4 이름 테이블

Doc_ID	Start_Region	Text_Value	Path_Info
1	30	Sw	10.20.30
1	40	1977	10.20.40
1	50	Star War	10.20.50
1	60	America	10.20.60
1	80	Jim Andrew	10.20.70.80
1	100	George Lucas	10.20.70.90.100
1	120	1980	10.20.70.90.110.120
1	140	PeterMorison	10.20.70.140
1	170	Adventure	10.20.160.170
1	180	Action	10.20.160.180
1	190	Sci-Fi	10.20.160.190
…	…	…	…

그림 3-5 값 테이블

3.1.4 영역-경로 인덱스 생성

본 논문에서는 명시한 영역-경로 정보는 인덱스 값을 빨리 찾기 위해 키 값의 중복을 허용하지 않는다. 따라서 같은 엘리먼트/애트리뷰트 이름이나 같은 값에 대해 하나의 집합을 이루도록 하여 인덱싱한다. 예를 들어, 그림 3-2의 트리에서는 Movie라는 노드가 세번 나타나는데, 이것의 영역-경로 정보는 (10.20), (10.320), (10.650)으로 세 개가 존재한다. 이렇게 동일한 이름의 노드가 여러 개 존재하는 경우, 트리에 대한 검색 비용을 줄이기 위하여 같은 이름의 노드들에 대하여 그들의 영역과 경로에 대한 정보들을 하나의 그룹으로 묶어서 처리한다. 이것을 본 논문에서는 영역-경로 인덱스로 정의한다. 이것을 형식화한다면 그림 3-6와 같이 XML 트리에서 유일한 내용의 엘리먼트 및 애트리뷰트 노드의 개수가 5개인 경우, 영역-경로 정보를 담고있는 집합 k는 총 5개가 되고

이들 각 집합은 동일한 엘리먼트/애트리뷰트 이름이나 또는 같은 값을 가지는 노드에 대한 영역-경로 정보를 원소로 갖는다. 예를 들어, k_0라는 노드는 XML 문서에서 세 번 발생하고 이들에 대한 영역-경로 정보는 a_0 ,a_1 ,a_2 이다. 이러한 영역-경로 인덱스는 XML 문서에 나타나는 엘리먼트/애트리뷰트 이름뿐만 아니라 이들의 값에 대해 모두 적용함으로써 구조 기반 검색 뿐 아니라 내용 기반 검색을 빠르게 지원할 수 있도록 한다.

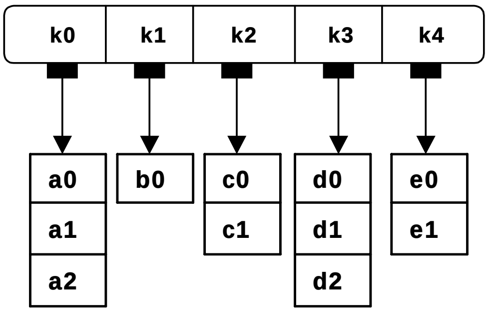

그림 3-6 영역-경로 인덱스

3.2 인덱싱

본 논문에서는 B+ tree를 이용하여 인덱싱 처리를 한다. B+ tree는 문서의 부분 삽입 및 삭제가 발생하게되면 기존의 인덱스 정보에 대한 변경없이 효율적으로 인덱스 구조에 반영할 수 있고 또한 균형트리로서 노드로의 이동 횟수가 적으므로 XML문서의 구성요소인 엘리먼트와 애트리뷰트에 대한 빠른 탐색을 유도하므로 기존의 인덱싱 연구에서도 B+ tree를 이용하는 추세이다. 그러나 우리의 인덱싱 기법은 일반적인 B+ tree를 변형한 형태로써 XML 문서에 있는 모든 엘리먼트/애트리뷰트 이름과 이들의 값 자체를 B+ tree 형태로 인덱스하는 것과, 이들에 대한 영역-경로 정보 집합들을 해시 형태로 인덱스하는 구조로 이루어진다. 이때 엘리먼트/애트리뷰트 값에 대한 인덱스와 이들의 영역-경로 정보에 대한 인덱스를 Value B+ tree라 하고, 엘리먼트/애트리뷰트 이름에 대한 인덱스와 이들의 영역-경로 정보에 대한 인덱스를 Name B+ tree라 한다.

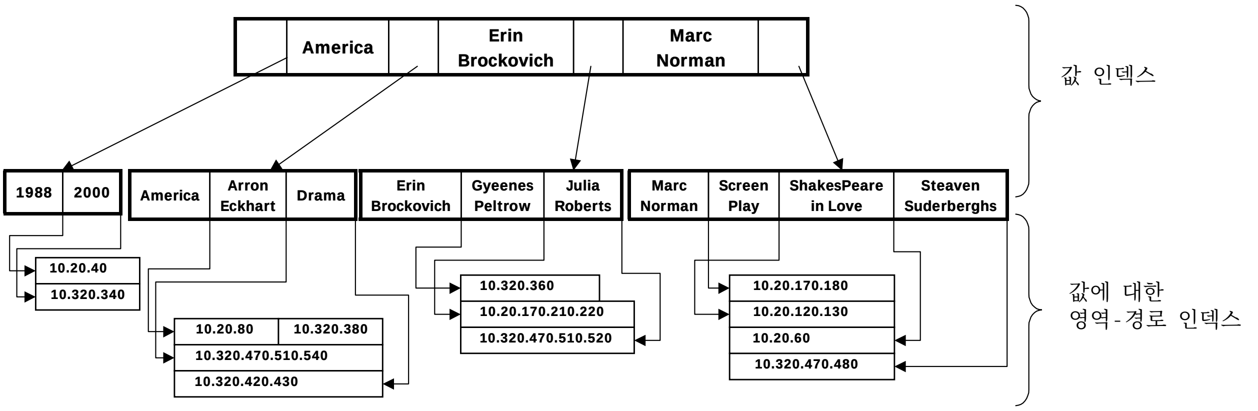

그림 3-7 Value B+ tree

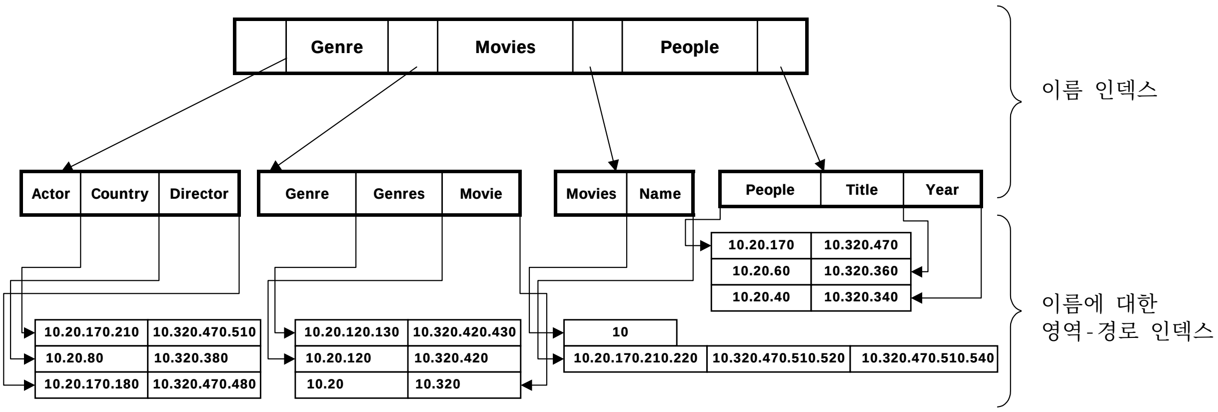

그림 3-8 Name B+ tree

3.2.1 Value B+ tree

그림 3-7의 Value B+ tree에서 각 노드의 키 엔트리(key entry)는 엘리먼트와 애트리뷰트의 값에 해당하고, 각 단말 노드마다 그것의 영역-경로 인덱스인 그림-6의 영역-경로 집합을 연결하는 포인터를 추가하였다. 즉, Value B+ tree는 엘리먼트와 애트리뷰트에 대한 값 인덱스와 각각에 대한 영역-경로 인덱스 정보를 함께 담고있으므로 사용자가 내용 기반 질의를 할 때 처리 결과를 빠르게 리턴한다. 예를 들어 XML 문서

에 대하여 [Year=”1998”]인 프리디킷을 가지는 XPath 기반의 질의문이 주어질 경우, 그림 3-7에서 만족하는 1998 값을 찾기위해 Value B+ tree의 노드를 탐색한 후 해당 단말 노드에 링킹된 영역-경로 인덱스에서 1998 값에 대한 위치 정보(10.20.40)를 쉽게 얻을 수 있다. 이런식으로 [Year=”1998” AND Country=”America”]와 같이 여러 개의 프리디킷을 가지는 질의문에 대해서도 빠른 검색이 가능하므로 다양한 내용 기반 질의 처리에 효율적이다.

3.2.2 Name B+ tree

그림 3-8의 Name B+ tree에서 각 노드의 키 엔트리는 엘리먼트와 애트리뷰트의 이름이 되고, XML 트리의 각 노드마다 부여한 영역과 경로에 대한 정보인 영역-경로 정보가 해당 단말노드에 링킹된 구조를 이룬다. 즉, Name B+ tree는 엘리먼트 및 애트리뷰트에 대한 이름 정보를 갖는 이름 인덱스와 각 이름마다 그것의 영역-경로 인덱스 정보를 함께 담고있으므로 경로 기반의 질의 처리를 빠르게 지원한다. 한 예로 Movies//Movie[Title=”Erin Brockovich”] 라는 XPath 기반의 질의문이 주어질 경우 Movies의 후손인 Movie 아래에 있는 Title을 검색하기 위해 우선, Name B+ tree에서 질의문에 주어진 각각의 노드를 탐색한 다음, 해당 단말 노드의 영역-경로 인덱스 정보를 통해 그들간의 계층 관계를 빠르게 확인할 수 있다. 즉, Movies의 영역-경로 정보는 (10) 이고, Movie는 (10.20),(10.320),(10.650) 이므로 Movies의 자식이 된다. 따라서 Title의 영역-경로가 (10.20).(10.320)과 (10.650)으로 시작되는 것만 찾으면 Movie 아래에 존재하는 만족하는 경로를 가지는 노드임을 바로 알 수 있다 (10.20.60),(10.320.360),(10.650,690). 그러므로 Name B+ tree를 이용하면 XML 문서의 엘리먼트 및 애트리뷰트간의 부모 자식 관계 뿐 아니라 서로간의 형제 관계 및 조상 후손에 관한 계층 정보를 빠르게 파악할 수 있으므로 경로 기반의 질의 처리시 효율적이다.

IV. 질의 처리

이 절에서는 본 논문에서 제안한 영역-경로 기반 인덱스 트리인 Value B+ tree와 Name B+ tree를 이용하여 질의 결과를 반환하는 과정을 예제 질의를 통해 설명한다. 특히 제안한 인덱싱 기법을 이용하면 사용자가 질의할 때 XML 문서에 대한 구조 정보를 정확히 표현하지 않아도 프리디킷 정보만으로 공통 조상을 파악하여 내용 기반 질의에 응답이 가능하고 각 구성 요소들 간의 조상-후손 관계를 빠르게 파악할 수 있으므로, 넓고 깊이 내포된 구조를 가지는 XML 문서의 내용 및 구조 혼합 검색에 유리하다. 여기서는 이러한 혼합 유형의 예제 질의문에 대하여 질의 처리시 이용되는 함수인 내용 기반 검색 알고리즘과 구조 기반 검색 알고리즘을 같이 설명하면서 이를 이용하여 질의문을 검색하는 방법을 보일 것이다.

4.1 내용 기반 검색 알고리즘

본 논문은 하나 이상의 프리디킷을 가지는 내용 기반의 질의문을 빠르게 처리하는 데에 초점을 두고있다. 따라서 주어진 조건문에서 각각의 프리디킷 값을 만족하는 프리디킷 노드를 우선적으로 찾고 이러한 조건들을 모두 만족하는 공통의 구성 요소를 찾는 식으로 상향(bottom-up) 탐색을 제안하였다. 이를 내용 기반 검색으로 정의하고 그에 대한 과정은 그림 4-1의 알고리즘을 통해 이루어진다.

첫째, 질의문을 입력으로 받아 구문 분석을 시작한다. 우선 사각괄호([ ]) 로 둘러 쌓인 조건문에서 AND로 구분되는 각각의 프리디킷을 확인한다. 프리디킷은 노드와 값을 한쌍으로 가지는데, 이때의 노드는 엘리먼트/애트리뷰트의 이름이고 값은 이들의 내용에 해당되는데 XML 문서 트리에서처럼 서로간에 조상-후손 관계가 성립한다.

둘째, 인덱스 트리인 Value B+ tree에서 1부터 j개까지의 프리디킷 값을 하나씩 대

입하여 해당 단말 노드를 통해 각각에 대한 영역-경로 정보를 출력한다. 프리디킷 노드에 대해서도 같은 방식으로 Name B+ tree를 검색하여 영역-경로 정보를 얻는데 이때 각 쌍의 프리디킷에 대하여 그것의 노드와 값이 서로 조상-후손 관계인 것만을 선택해야하므로, 앞에서 구한 프리디킷 노드의 영역-경로 정보가 그것과 대응하는 프리디킷 값의 영역-경로 정보에 포함되거나 일치하는 것만을 선택한다.

셋째, 추려진 각 프리디킷 노드의 영역-경로 정보들을 제일 처음의 접두어부터 서로 비교하여 가장 많이 일치하는 영역-경로 정보를 구하게 되고 이것이 바로 조건을 만족하는 공통의 조상 경로가 된다. 이러한 공통의 조상 경로를 가지는 엘리먼트 및 애트리뷰트가 주어진 질의문에서 정규 표현 (*)에 해당하는 임의의 조상 노드로써 내용 기반 질의를 완성한다.

알고리즘 1

Definition:
-$pn_1, pn_2, \dots pn_j$: 프리디킷 노드
-$pv_1, pv_2, \dots pv_j$: 프리디킷 값
-$qn_1, qn_2, \dots qn_x$: 이름에 대한 영역-경로 정보
-$qv_1, qv_2, \dots qv_y$: 값에 대한 영역-경로 정보
-$conA_1, conA_2, \dots conA_n$: 조건값에 만족하는 노드

Description:
하나 이상의 프리디킷을 가지는 질의문 처리를 위해 조건을 만족하는 공통의 조상을 검색하는 고리즘

Input: Q = //*[pn_1 =" pv_1 " AND pn_2 =" pv_2 " … AND pn_j =" pv_j "]
Value B+-tree,
Name B+-tree
Output: 조건에 만족하는 공통 조상

Function ***SearchCommonAncestor()*** {
while(1부터 j까지) **do**
$qv_y \leftarrow$ Value B+-tree에서 pv_j 의 영역-경로 정보 검색.;
$qn_x \leftarrow$ Name B+-tree에서 pn_j 의 영역-경로 정보.검색;
end

/* (qn_x, qv_y) 쌍을 비교하여 순수한 조상-후손 관계를 가지는 것을 추려냄 */
while (1부터 x까지) **do**
for (1부터 y까지) **do**
$comA_n \leftarrow$ qn_x 이 qv_y 와 일치하거나 그것에 포함되는 qn_x 를 찾음.
만일, 그러한 포함관계를 만족하지 않는 qn_x 는 제거.;
end
end

while(1부터 n까지) **do** /*$conA_n$ 에 대한 *self join* 수행*/
for (1부터 n까지) **do**
comPrefix $\leftarrow$ 공통된 영역-경로의 시작경로를 구함
end
end

return *comPrefix* ; // 공통 조상반환
}

그림 4-1 내용 기반 검색 알고리즘

4.2 구조 기반 검색 알고리즘

내용 기반 검색 과정이 끝나고 조건을 만족하는 공통의 조상 노드가 추출되면 입력된 질의문 처리를 완수하기위해 그것과 조상-후손 관계에 있는 최종 질의 결과 노드에 대한 검색을 수행해야한다. 특정 조상 노드의 후손 노드를 검색하므로 이러한 과정을 구조 기반 검색이라 하고 이에 대한 알고리즘은 그림 4-2와 같다.

본 논문에서의 구조 기반 검색은 질의문에서 절대 경로(‘/’) 혹은 상대 경로(‘//’) 관계에 놓인 2개의 노드간의 구조 정보를 따져서 후손 노드를 검색한다.

조상 노드를 Anc라 하고 후손 노드를 Des라 할 때, 우선 Name B+ tree에서 Anc와 Des 노드를 키 값으로 주고 각각이 가리키는 영역-경로 인덱스를 호출하여 그것의 영역-경로 정보들을 출력한다. 이때 주어진 이름의 Anc를 정확히 조상으로 가지는 Des만을 추려내기 위해, 조건절의 위치에 따라 검색을 달리한다.

만일 Anc에 조건절이 달려있을 경우 Des의 영역-경로 정보와 일치하거나 그것에 포함되는 Anc의 영역-경로 정보를 찾는다. 그런 다음, 4.1절에서 구한 공통의 조상 경로와 일치하는 Anc의 영역-경로 정보를 구하고 그러한 경로로 시작되는 Des 를 찾는다.

조건절이 Des에 있을 경우, Anc의 영역-경로 정보와 일치하거나 그것을 포함하는 Dec의 영역-경로 정보를 검색한 후, 4.1절에서의 결과물인 공통의 조상 경로와 일치하는 Dec의 영역-경로 정보를 추출한다.

이러한 과정의 결과로 주어진 조건을 만족하는 최종 Dec의 영역-경로 정보를 얻게되고 구조 기반 검색을 마무리짓는다.

알고리즘 2

Definition:
-Anc : 조상노드
-Des : 후손노드
-$ap_1, ap_2, \ldots ap_j$: 조상노드의 영역-경로 정보
-$dp_1, dp_2, \ldots dp_j$: 후손노드의 영역-경로 정보
-$cap_1, cap_2, \ldots cap_n$: 최종 후손노드의 시작경로

Description:
하나 이상의 프리디킷을 가지는 질의문에서 조건을 만족하는 후손을 검색하는 알고리즘

Input: Q = //Anc[pn_1 =" pv_1 " AND pn_2 =" pv_2 " … AND pn_j =" pv_j "]//des
Name B+-tree,
Value B+-tree
Output: 조건에 만족하는 후손

```
Function SearchDescendant() {

   /* 조상 노드와 후손노드 영역-경로 정보를 알아냄*/
    ap_x ←  Name B+-tree에서 anc 의 영역-경로 정보.검색;
    dp_y ←  Value B+-tree에서 des 의 영역-경로 정보 검색.;

   /* 프리디킷을 만족하는 공통 조상 경로 검색 */
    cp =  SearchCommonAncestor();

  while (1부터x까지) do
    cap_n ← ap_x 이 cp 와 일치하거나 그것에 포함되는 ap_x 를 찾음.
  end

 /* 최종 후손노드 정보 출력 */
  return  cap_n 의 경로로 시작하는 모든 dp_j ;
}
```

그림 4-2 구조 기반 검색 알고리즘

4.3 영역-경로 인덱스를 이용한 질의 처리

본 논문에서 사용하는 XML 질의문은 XPath 구문을 이용한 것으로써 사각괄호([])로 묶인 하나의 섹션 내에 정규 경로 표현식을 이용하여 값에 대하여 하나 이상의 프리디킷을 지닌 질의문으로 한정하였다. XML 문서의 구성 요소간의 계층 관계는 ‘ / ’와 ‘ // ’

로 표현하였는데, 전자는 두 구성 요소간의 부모-자식 관계를 가리키고, 후자는 각각 임의의 깊이에 위치한 조상-후손 관계를 지칭한다. 또한 임의의 엘리먼트에 대해서는 와일드 카드 ‘*’를 사용하였다.

본 절에서는 앞에서 설명한 내용 기반 검색 알고리즘과 구조 기반 검색 알고리즘을 적용하여 그림 3-1의 예제 XML 문서에 대하여 XPath를 기반으로 표현한 질의문에 대한 처리 과정을 보여준다.

4.3.1 예제 질의 1

예를 들어 Country가 ‘America’인 조건을 가지는 임의의 노드에 대한 Title 정보를 찾고자 한다면 이를 XPath 구문을 이용하여 표현한 질의문은 다음과 같다.

Query = //*[Country=”America”]//Title

위 질의문에서 하나의 조건절 내에 있는 [Country=”America”] 는 엘리먼트 및 애트리뷰트의 이름과 값 정보를 이용하는 내용 기반 검색을 의미하고, 더블 슬래시(상대경로)를 사용한 */Title 은 조상-후손 관계를 위해 구조 정보를 필요로 한다.

질의 처리는 내용 검색을 먼저 한 후 구조 검색을 하는 과정을 밟는다

가. 질의문 수행을 위해 우선 조건절에 있는 프리디킷 값인 ‘America를’ 검색한다. 즉, Value B+ tree에서 ‘America’를 탐색하여 해당 단말 노드에서 그것의 영역-경로 정보를 검색한다.

- America : (10.20.80),(10.320.380)

그런 다음 ‘America’에 대응하는 ‘Country’노드 이름을 검색한다. 즉, Name B+ tree에서 ‘Country’의 영역-경로 정보를 구한다.

- Country : (10.20.80),(10.320.380),(10.690.730)

이때, Country와 America가 조상-후손 관계인 것만을 추려내기 위해 America의

영역-경로 정보와 일치하거나 그것에 포함되는 Country의 영역-경로 정보를 찾는다.

- Country : (10.20.80),(10.320.380)

나. 임의 노드에 대한 후손 노드인 ‘Title’ 정보를 알아내기 위해 Name B+ tree에서 ‘Title’을 탐색하여 영역 경로 정보를 알아낸다.

- Title : (10.20.60), (10.320.360)

이때, 가)의 결과물인 Country의 영역-경로 정보와 Title의 영역-경로 정보를 서로 비교하여 같은 조상의 경로로 시작되는지를 확인한다.

- Country : (10.20.80),(10.320.380)

- Title : (10.20.60), (10.320.360)

밑줄친 10.20 과 10.320 이라는 공통의 조상 경로를 가지므로 Country와 Title이 공통 조상을 가지는 후손 노드임을 알 수 있다.

다. 따라서 (10.20.60) 과 (10.320.360) 의 경로를 키로 가지는 데이터 값인 ‘Shakespeare in Love와 ‘Erin Brockovich’가 질의를 만족하는 Title이 된다.

4.3.2 예제 질의 2

아래 질의문은 예제 1)의 질의와 거의 비슷하나 조건문을 만족하는 조상 노드가 임의 노드가 아닌 Movie로 한정되었다.

Query = //Movie[Country=”America”]/Title

가. 수행 과정은 예제 1)의 가) 단계까지 동일하다. 따라서 ‘America’ 값을 가지는 Country의 영역-경로 정보는 다음과 같다.

- Country : (10.20.80),(10.320.380)

나. Movie 노드에 대한 정보를 알기 위해 Name B+ tree에서 ‘Movie’를 탐색하여

해당 단말 노드에서 그것의 영역-경로 정보를 검색한다

- Movie : (10.20),(10.320)

이때, 가)의 Country의 영역-경로 정보와 Movie의 영역-경로 정보를 비교하여 두 노드가 조상-후손 관계에 있는지 살핀다. 이때 모든 Country의 영역-경로 정보가 Movie의 영역-경로 정보로 시작되므로 이 둘은 조상-후손 관계를 만족함을 알 수 있다.

다. Name B+ tree에서 ‘Title’을 탐색하여 영역 경로 정보를 알아낸다.

- Title : (10.20.60), (10.320.360)

이때, 조건을 만족하는 Movie 아래에 있는 Title 을 찾기 위해 나)에서 구한 Movie의 영역-경로 정보로 시작하는 Title 값을 검색한다.

- Movie : (10.20),(10.320)
- Title : (10.20.60), (10.320.360)

라. 결국 (10.20.60) 과 (10.320.360) 을 탐색하여 ‘Shakespeare in Love와 ‘Erin Brockovich’가 질의를 만족하는 Title이 된다.

4.3.3 예제 질의 3

한 개의 프리디킷을 갖는 예제 1,2와 달리 다음은 2개의 프리디킷을 가지는 질의문이다. 또한 본 논문에서는 사용자의 편의를 돕기 위해 질의문에서 문서의 계층 구조를 정확히 표현하지 않아도 질의 응답이 가능하다. 따라서 기존 XPath에서와 같이 프리디킷의 노드와 값의 부모-자식 관계에 있지않고 조상 후손 관계에 있는 것도 지원해준다. 즉, 그림 1-2 XML 문서 트리에서 보면, Genres 아래 Genre 엘리먼트가 있고 그 아래에 데이터 값이 존재한다. 그렇지만 Genres와 데이터 값 정보만 입력해도 질의 수행이 가능하다.

예를 들어 Country가 ‘America’이고 Genres가 ‘Drama’인 영화의 Title을 찾고자 한다면 XPath를 기반으로 표현한 질의문은 다음과 같다.

Query = //Movie[Country=”America” AND Genres=”Drama”]//Title

가. 2개의 프리디킷을 가지는 질의문 수행을 위해 우선 조건절에 있는 프리디킷의 값인 America와 Drama를 검색한다. 즉, Value B+ tree 에서 America와 Drama를 탐색하여 해당 단말 노드에서 각각에 대한 영역-경로 정보를 검색한다.

- America : (10.20.80),(10.320.380)
- Drama : (10.320.420.430)

나. 그런 다음, 이들 값에 대응하는 Country와 Genres 노드 이름을 검색한다. 즉, Name B+ tree에서 Country와 Genres의 영역-경로 정보를 구한다.

- Country : (10.20.80),(10.320.380)
- Genres : (10.320.420)

따라서 조건절의 구성 요소인 Country와 Genres의 영역-경로 정보들을 서로 비교하여 둘 간에 가장 많이 일치하는 영역-경로 정보(10.320)를 얻게되고 이것이 바로 공통의 조상 경로가 된다

다. 공통의 조상 경로를 알아낸 후 최종 질의 결과를 얻으려면 조상인 Movie의 후손인 Title을 찾아야 하는데 질의문에서 Movie//Title 에 대한 처리를 위해 Name B+ tree에서 Movie와 Title의 영역-경로 정보를 구한다.

-Movie : (10.20),(10.320)

-Title : (10.20.60),(10.320.360)

그런 다음 미리 구한 공통의 조상 경로(10.320)과 일치하는 Movie(10.320)를 찾고, 이러한 경로로 시작되는 최종 Title 을 얻는다. (10.320.43)

라. (10.20.60) 과 (10.320.360) 을 탐색하여 ‘Shakespeare in Love와 ‘Erin Brockovich’을 리턴한다.

Ⅴ. 실험 및 분석

본 장에서는 3장에서 제안한 XML 문서에 대한 인덱싱 기법에 대한 성능 평가를 위해 실험 데이터를 변환하고, 수행한 결과를 제시한다. 본 논문에서 제안한 방식대로 구축한 인덱스를 이용해서 내용과 구조 정보를 모두 포함한 혼합 질의를 처리하는 방법이 XML 전용 데이터베이스로 유명한 X-Hive사의 X-Hive/DB 솔루션을 이용하는 방법에 비해서 성능이 월등하게 뛰어남을 보인다.

5.1 실험 환경

실험 환경을 설명하기 전에 우선 본 실험에서 성능을 비교한 X-Hive/DB를 잠시 소개하면, X-Hive/DB는 XML 전용 데이터베이스(Native XML Database)로서 XML 문서에 대한 (논리적)모델을 정의하고, 그 모델에 따라 XML 문서를 저장하고 질의에 대한 결과를 추출하는데 특히 DOM 트리처럼 XML 문서의 바이너리 모델을 저장한 모델 기반의 대표적 솔루션이다.

운영체제	Windows 2000 Advanced Server
CPU/RAM	Pentium IV processor 2.40MHz/512M RAM
구현 언어	JAVA(JDK 1.4.2)
DBMS	Microsoft SQL Server
개발도구	Borland JBuilder 9.0

표 5-1 시스템 모듈 구현 환경

본 논문에서 인덱스 시스템을 구현하고 테스트한 환경은 그림 5-1과 같다.

인덱스 시스템 처리 모듈은 Intel Pentium 4 CPU 2.40GHz 컴퓨터의 Windows 2000 Advanced Server 운영체제에서 수행되었으며 컴퓨터의 메인 메모리의 크기는

512MB 이다. RDBMS로 MS SQL Server를 사용하였고 MS SQL Server에 저장되어 있는 XML문서에 대한 테이블 데이터를 처리하고, JBuilder 9.0을 사용하여 JAVA로 구현하였다.

5.1.1 실험 데이터

표 5-2는 실험에서 사용된 XML 데이터의 부류와 데이터 사이즈 및 특징을 나타낸다. 실험 데이터는 두 가지 부류로 나뉜다. 각종 영화들에 대한 정보를 담고 있는 XML 문서들의 집합인 Movie 데이터와, 책에 대한 내용을 XML로 표현한 문서들의 집합인 Book 데이터들이다.

내용	Movie 데이터	Book 데이터
전체 문서 수	100	19
평균 문서의 크기	155KB	570KB
문서당 평균 엘리먼트 수	4776	5445
엘리먼트의 평균 자식 수	5	9
문서당 평균 텍스트값의 수	2908	3630
전체 엘리먼트 개수	477600	103455

표 5-2 실험 XML 데이터

5.1.2 실험 질의

		질의문
Movie	Q1	//*[Country=”America”]//Title
	Q2	//Movie[Country=”America”]//Title
	Q3	/MovieList/Movie[Country=”America”]//Title
	Q4	//*[*=”America”]//Title
	Q5	//*[Country=”America” AND Year=”1998”]//Title
	Q6	//Movie[Country=”America” AND Year=”1998”]//Title

Book	Q1	//*[title=”PURPOSES AND PRINCIPLES”]//para
	Q2	//chapter[title=”PURPOSES AND PRINCIPLES”]//para
	Q3	/chapters/chapter[title=”PURPOSES AND PRINCIPLES”]//para
	Q4	//*[*=”PURPOSES AND PRINCIPLES”]//para
	Q5	//*[title=”PURPOSES AND PRINCIPLES” AND number=”1”]//para
	Q6	//chapter[title=”PURPOSES AND PRINCIPLES” AND number=”1”]//para

표 5-3 실험에 사용된 질의 유형

2가지 부류로 나뉘어진 Movie와 Book에 대한 각각의 XML 데이터에 대하여 동일한 유형의 6개의 질의를 수행하였다.

실험에 사용된 질의는 표 5-3과 같다.

질의의 경로가 짧게는 1부터 길게는 4까지, 조건 정보가 하나인 것과 2개인 것을 조합하여 다양하게 선택되었고 XML 문서상에서 엘리먼트/애트리뷰트의 발생 빈도 및 텍스트 단어의 발생 빈도의 여러 조합을 고려하여 만든 것이어서 특정 질의 패턴으로 한정되는 경향에 대한 가능성을 배제하였다. 즉, 임의개의 프리디킷을 가지는 내용 및 구조의 혼합검색 형태의 질의에 대하여 영역-경로 인덱싱 기법을 적용한 검색 성능의 우수성을 보여주기 위해 다양한 형태의 질의문을 가지고 실험하였다. 프리디킷의 개수와 경로 구조에 따라 질의 유형을 구분하였고 XPath 구문을 이용하였다.

Q1에서 Q5까지는 하나의 프리디킷을 가지는 질의문인데, 이중 Q1은 엘리먼트 이름과 텍스트 값의 정보만이 주어진 임의의 조상 노드의 후손을 검색하는 것이고, Q2는 Q1과 비슷한 구조를 가졌지만 특정 조상 노드에 한정된 후손 값을 찾는 질의문이다. Q3은 Q1,Q2 와 동일한 내용의 프리디킷 정보를 가졌지만 절대경로로 주어진 질의문이고, 임의 노드에 대한 값 정보만을 가지고 후손 노드를 검색하기 위해 여러 개의 와일드카드를 부여한 Q4를 실험하였다.

Q5와 Q6은 Q1,Q2,Q3과 같은 내용을 포함하며 2개의 프리디킷을 가지는 질의문으로써 여러 개의 조건 정보와 조상 노드간의 조인 유무에 따라 질의 처리 시간에 대한 차이를

평가하기 위해 구분하여 주어졌다.

5.1.3 실험 평가

2개 부류의 XML 문서에 대한 위의 6개의 질의에 대하여 그림 5-1는 XML 문서 사이즈에 따라 표 5-3에서 제시된 질의의 첫번째 유형의 질의 수행 결과에 대한 시간을 각각 비교한 것이다. 또한 X-Hive와 본 인덱싱 모델에서 각각 실행한 결과를 Book 데이터와 Movie 데이터로 나누어 각각 6개 유형에 대하여 그림 5-2과 같이 그래프로 나타내었다. 참고적으로 수치의 단위는 msec이다. 전체적으로 그림 5-2에서와 같이 2개 부류의 XML 문서 결과 내용은 비슷한 성능을 가졌고, 특히 본 논문의 인덱싱 기법을 이용한 결과가 X-Hive보다 평균적으로 약 1.5배의 비율로 좋은 성능을 나타냈음을 알 수 있다. X-Hive는 질의의 경로가 긴 경우에도 본 인덱싱 모델에 비해 수행 시간이 상대적으로 오래걸렸고, 질의의 경로가 짧아지고 경로 표현식이 많아질수록 질의 수행 시간이 크게 증가하였다.

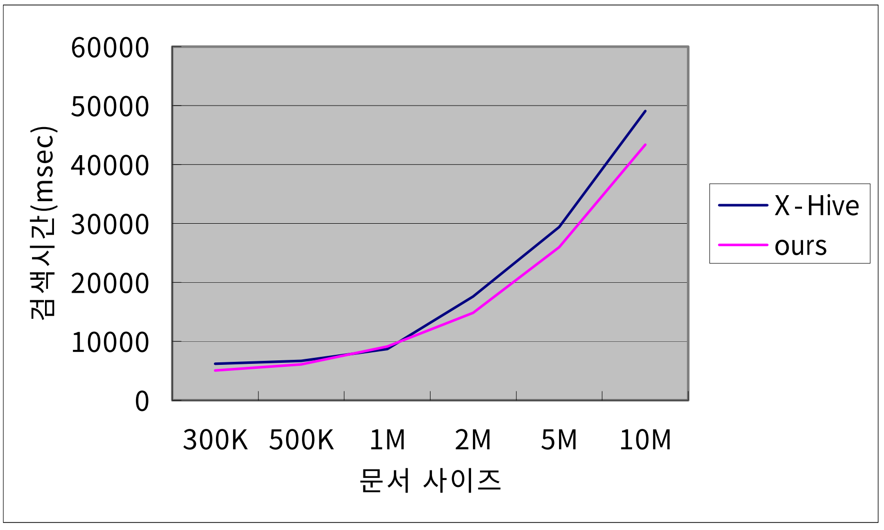

그림 5-1 문서 사이즈에 따른 질의 검색 시간

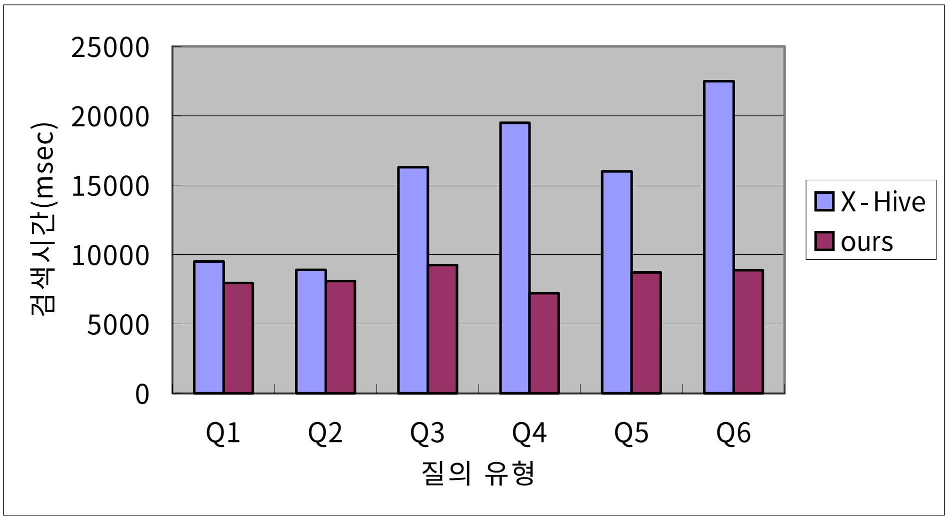

그림 5-2 (a) 질의 유형에 따른 질의 검색 시간 (Book 데이터)

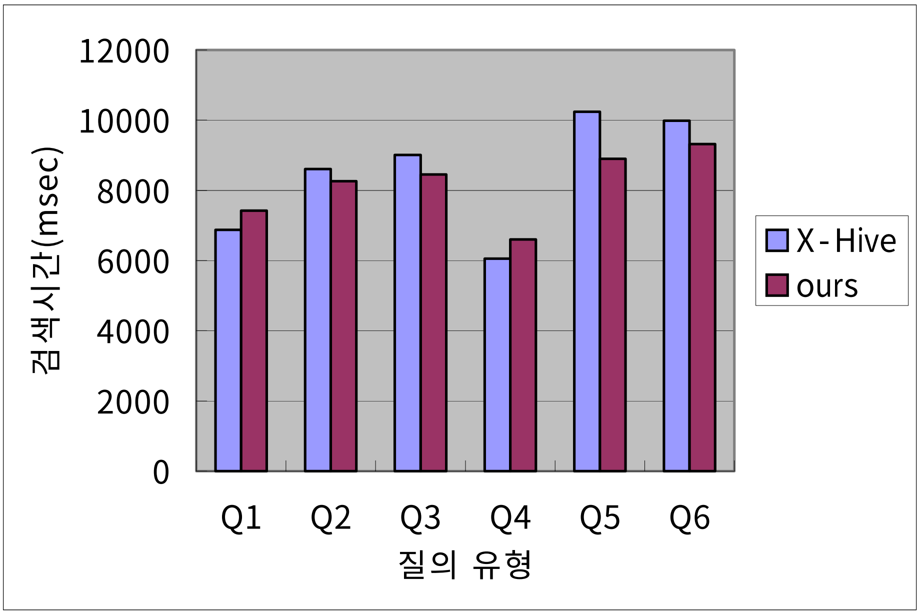

그림 5-2 (b) 질의 유형에 따른 질의 검색 시간 (Movie 데이터)

본 인덱싱 모델을 이용한 질의문 Q1,Q2,Q3의 실험 결과를 분석하면 이름과 값에 대한 조건 정보만 있는 Q1의 성능이 가장 좋고 그 다음 Q2,Q3 순서로 성능이 떨어졌다. 본 논문에서는 내용 기반 질의를 빠르게 지원을 위해 상향식으로 질의문 분석을 수행하는데 구체적인 조상 경로 구조가 더 주어질수록 조인 계산이 요구되므로 오히려 검색 시간이

오래걸린다. 따라서 절대경로 질의보다는 상대경로 질의 검색에 더 효율적이고 조건 정보가 적을수록 빠르게 결과 값을 반환한다. Q4는 프리디킷 내에 하나의 값만 주어진 질의문으로써, Q1과 비교하면 조건의 개수가 적으므로 더 빠르게 질의 결과를 얻을 수 있다. 2개의 프리디킷을 가진 Q5,Q6의 경우는 1개의 프리디킷을 제공하는 Q1,Q2와 비교시 검색시간의 차이가 크지않았다. 따라서 조건 정보의 개수가 비례적으로 증가해도 실제로 질의 수행 시간은 프리디킷 개수와 상관없이 거의 일정함을 알 수 있었다.

따라서 이 논문에서 제안한 인덱싱 기법은 프리디킷 개수와 상관없이 다양한 유형의 내용과 조건 혼합 질의문에 대하여 효율적인 검색을 지원함을 알 수 있다.

VI. 결론 및 향후과제

XML은 웹상에서 구조화된 문서를 전송 가능하도록 설계된 표준화된 텍스트 형식의 마크업 언어로써 인터넷에서 바로 사용 가능한 문서를 표현하는 표준이다. 오늘날 특히 인터넷을 이용한 전자 문서의 중요성이 부각되면서 내용 및 계층적 구조를 가진 대용량의 XML 문서에 대해 효율적으로 관리하고자 하는 연구들이 진행되고 있다. 그러나 기존 연구들은 질의시 사용자에게 정확한 경로 표현식을 요구하거나 또는 다양한 질의 형태를 고려하지 않은 비효율적인 인덱스 구조를 갖고 있기 때문에 이러한 기존 연구들의 미흡한 부분들을 해결해줄 수 있는 효율적인 인덱스 구조가 필요하다.

본 연구는 내용 검색, 구조 검색을 혼합한 다양한 유형의 질의문을 처리해주기 위하여 XML 문서의 구조 정보와 내용에 대한 정보까지 효율적으로 관리해주는 새로운 인덱싱 기법을 제안하였다

각 노드마다 조상의 시작 영역을 접두어로하여 연결한 '영역-경로 정보'를 부여함으로써 XML 문서의 각 구성 요소간의 부모 자식 관계 뿐 아니라 해당 구성요소에 대한 형제 관계 및 조상 후손에 관한 계층 정보를 쉽게 파악할 수 있다. 따라서 깊이 내포된 계층구조와 더불어 공통된 조상 아래 다양한 자손 노드를 가지는 수평적으로도 넓은 XML 문서에 대하여 효율적이고 빠른 검색을 지원할 수 있다.

또한 실험을 통해, 제안한 인덱싱 기법을 이용하여 내용과 구조정보를 혼합한 다양한 유형의 질의문 처리시 프리디킷 내의 조건 개수와 관계없이 거의 일정하게 좋은 성능을 보였다.

본 논문의 의의는 크게 다음과 같이 두 가지로 요약될 수 있다.

첫번째, 본 논문에서 제시한 영역-경로 인덱스 기법은 XML 문서를 질의하기 위한 검색 시스템은 물론, 인터넷의 XML 기반 지식 관리 시스템에서 이용될 수 있다. XML의 문서의 내용과 구조 정보를 담는 특징을 효율적으로 지원하여 사용자가 XML 문서의 구

조적 정보를 정확히 표현하지 않더라도 제시된 내용 정보를 기반으로 질의에 대한 응답하고 또한 찾고자 하는 내용 조건의 개수에 상관없이 빠르게 결과를 반환해줄 수 있는 제안된 인덱싱 기법을 이용한다면, 구조화된 데이터 및 반구조화된 데이터를 보유한 기업 및 시스템 개발자들에게 데이터 관리의 처리 부문에 있어 빠르게 지원해줄것으로 전망한다.

두번째, 본 논문에서 수행한 인덱싱 기법에 따른 질의 성능 평가는 여러 XML 관리 시스템들에 대한 성능 평가에 활용될 수 있다. 본 논문에서 적용한 XML 질의 유형의 분류는 XPath를 기반으로 하되 XML 문서에 대하여 특히 사용자들의 편의를 생각하여 실제로 많이 쓰이는 질의 유형들로 고려하여 분류해 놓은 것이다.

향후 본 연구의 성능을 더 높일 수 있는 개선점을 찾고, 더 많은 다른 기존 논문들과의 성능 평가를 실시하는 연구를 향후 연구 과제로 제시한다.

참 고 문 헌

[1] W3C Consortium, XML 1.0 (Second Edition), W3C Recommendation 06 Oct. 2000, available at http://www.w3.org/TR/REC-xml.

[2] W3C Consortium, XML Path Language (XPath), version 2.0, W3C Recommendation Nov 12, 2003. http://www.w3.org/TR/xpath20.html.

[3] S.Boag, D. Chamberlin, M. F. Fernanderz, D. Florescu, J. Robie, and J. Simeon, XQuery 1.0: An XML Query Language, W3C Working Draft Nov 12, 2003. http://www.w3.org/TR/xquery/.

[4] D. Srivastava, S. Al-Khalifa, H. V. Jagadish, N. Koudas, J. M. Patel, and Y. Wu. Structural joins: A primitive for efficient XML query pattern matching. Proc. Of Int. Conf. On Data Engineering, pages 141-151, 2002

[5] C. Zhang, J. F. Naughton, D. J. DeWitt, Q. Luo, and G. M. Lohman. On supporting containment queries in relational database management systems. Proc. of the ACM SIGMOD Int. Conf. on Management of Data, pages 425-436, 2001.

[6] S.-Y. Chien, Z. Vagena, D. Zhang, V. Tsotras, and C. Zaniolo. Efficient Structural Joins on Indexed XML documents. Proc. Of the VLDB, pages 263-274, 2002.

[7] H. Jiang, H. Lu, and W. Wang. XR-Tree: Indexing XML Data for Efficient

Structural Joins. Proc. Of International Conference On Data Engineering, pages 253-264, 2003.

[8] B. F. Cooper, N. Sample, M. J. Franklin, G. R. Hjaltason, and M. Shadmon. A fast index for semistructured data. Proc. Of the VLDB, pages 341-350, 2001

[9] Q. Li and B. Moon. Indexing and Querying XML Data for Regular Path Expressions, Proc. Of the VLDB, pages 361-370, 2001.

[10] A. M. Flavio Rizzolo. Indexing XML Data with Toxin. In WebDB, pages 49--54, 2001.

[11] Q. Chen, A. Lim, and K. W. Ong, D(k)-Index: An adaptive structural summary for graph-structured data, Proc. of the ACM SIGMOD Int. Conf. on Management of Data, 2003

[12] H. Jiang, H. Lu, and W. Wang. XR-Tree: Indexing XML Data for Efficient Structural Joins. Proc.Of International Conference On Data Engineering, pages 253-264, 2003.

[13] Roy Goldman , Jennifer Widom, DataGuides: Enabling Query Formulation and Optimization in Semistructured Databases, Proc. Of the VLDB, pages 436-445, August 25-29, 1997.

[14] T. Milo and D. Suciu, Index Structures for path expressions, Proc. of the IEEE Int.

Conf. On Data Theory, pages 277-295, 1999

[15] Albrecht Schmidt, Martin Kersten, Menzo Windhouwe. Querying XML Documents made Easy : Nearest Concept Queries, Proc.Of International Conference on Data Engineering, 2001

[16] S.W.Kim, et al. Indexing and Retrieval of XML-encoded Structured Documents in Dynamic Environment. Lecture Notes in Computer Science (LNCS) Vol.2480, 2002

[17] Peter Buneman, Susan Davidson, Gerd Hillebrand, and Dan Suciu. A query language and optimization techniques for unstructured data. Pro of the ACM SIGMOD International Conference on the Management of Data, pages505-516.

[18] Svetlozar Nestorov, Jeffrey Ullman, Janet Wiener, and Sudarshan Chawathe. Representative objects: concise representations of semistructured, hierarchical data. Proc. of the IEEE International Conference on Data Engineering, 1997:79-90.

[19] R. Kaushik et al., “Exploiting Local Similarity for Indexing Paths in Graph-Structured Data”, Proc.Of International Conference On Data Engineering, 2002.

ABSTRACT

A Region-Path Indexing Technique for Content Based Retrieval of XML documents

Department of Computer Science & Engineering
Ewha Institute of Science and Technology
Choi Eun Hye

XML is a markup language for documents containing contents and structured information. Especially data on the internet are usually represented and transferred as XML. The XML data is represented as a tree and therefore, indexing techniques are needed to efficiently support hierarchical structural properties. XML queries are represented as regular path expressions and evaluated by traversing each object of the tree. Several indexes are proposed to fast evaluate regular path expressions.

However, in some cases they may not cover all possible paths because they require a great amount of disk space. In order to efficiently evaluate the queries in such cases, we propose an optimized traversing which uses the region-path index. Minimizing traverse of not only data object but also index object, we can execute the XPath expressions fast. That is, we propose a region-path index scheme as a new index scheme to efficiently process queries with extended path expressions Our proposed index scheme allocates a unique path identifier for every possible single

path in as extended path expression and provides functionalities of both single path indexing and multiple path indexing through the composition of index key and path identifier while using only a index structure. The proposed index scheme provides better performance than single-path index schemes, and is practical since it can be implemented by little modification of leaf record of a B+ tree index.

We conducted diverse experiments to show that our indexing techniques achieves a better performance than the other approach.