XML 데이타 색인을 위한 경로 분할 기법
(A Path Partitioning Technique for Indexing XML Data)

김 종 익 † 김 형 주 ††
(Jongik Kim) (Hyoung-Joo Kim)

요 약 XML에 대한 질의 언어는 데이타 그래프 내의 경로를 이용하여 질의를 표현한다. 특히, 경로에 패턴 (예를 들어, 정규식)을 사용함으로써, 데이타의 구조를 정확히 알지 못하더라도 질의가 가능하도록 한다. 이때, 패턴을 이용하는 질의는 데이타 그래프의 탐색범위를 크게 넓히게 된다. 기존의 XML색인 기법은 질의의 탐색범위를 줄이기 위해 데이타 그래프 내의 서로 동일한 경로들을 하나로 묶어 작은 크기의 색인 그래프를 생성하는 방법을 이용한다.. 하지만 이러한 색인들은 많은 경우 색인의 크기가 데이타 그래프의 크기만큼 증가하게 되어 질의의 탐색범위를 줄이지 못하고, 따라서 효율적인 질의 처리를 보장하지 못한다.

본 논문에서는 데이타 내에 존재하는 모든 경로를 분할(partitioning)하고 질의 처리시 질의에 맞는 분할 영역을 빠르게 찾아낼 수 있는 색인 그래프를 제안한다. 본 논문에서 제안하는 색인 그래프는 데이타 그래프의 크기와 상관없이 색인 그래프의 크기를 조절할 수 있다. 따라서 색인 그래프의 크기를 작게 구성함으로써 색인 그래프 탐색 비용을 크게 줄일 수 있다. 본 논문에서는, 실험을 통해 기존의 그래프 기반 색인 기법들보다 본 논문의 색인 기법이 보다 효율적임을 보이고 색인의 크기 변화에 따른 성능 변화에 대해 알아본다.

키워드 : XML, 비정형 데이타, 경로 색인, 경로 분할

Abstract Query languages for XML use paths in a data graph to represent queries. Actually, paths in a data graph are used as a basic constructor of an XML query. User can write more expressive queries by using patterns (e.g. regular expressions) for paths. There are many identical paths in a data graph because of the feature of semi-structured data. Current researches for indexing XML utilize identical paths in a data graph, but such an index can grow larger than source data graph and cannot guarantee efficient access path.

In this paper we propose a partitioning technique that can partition all the paths in a data graph. We develop an index graph that can find appropriate partitions for a path query efficiently. The size of our index graph can be adjusted regardless of the source data. So, we can significantly improve the cost for index graph traversals. In the performance study, we show our index much faster than other graph based indexes.

Key words : XML, semi-structured data, path index, path partition

1. 서 론

XML[1]이 인터넷 문서 교환의 표준으로 채택 되면서 XML을 효율적으로 관리하기 위한 연구가 계속되고 있다. XML은 정형화 된 구조를 가지고 있지 않다는 것이 특징이다. 이러한 특징으로 인해 XML을 비정형 데이타(semi-structured data)의 일종으로 볼 수 있다. XML과 비정형 데이타는 유연한(flexible) 구조를 손쉽게 표현할 수 있도록 그래프 형태의 데이타 모델[2,3]로 표현된다. 본 논문에서는 그래프 형태로 표현된 XML데이타를 데이타 그래프라고 부른다. 데이타 그래프는 에지(edge)또는 노드(node)에 레이블(label)이 달려 있는 형태로 표현된다. XML의 각 요소(element)와 속성(attribute)은 그래프 내의 하나의 노드에 대응될 수 있다. 또한, 요소의 태그(tag) 또는 속성의 이름은 해당 노드 또는 해당 노드로 들어오는 에지에 레이블로 표시될 수 있다.

· 본 연구는 두뇌한국21 사업과정보통신부의 대학 IT연구센터(ITRC) 지원을 받아 수행되었음
† 학생회원 : 서울대학교 컴퓨터공학부
jikim@oopsla.snu.ac.kr
†† 종신회원 : 서울대학교 컴퓨터공학부 교수
hjk@oopsla.snu.ac.kr
논문접수 : 2003년 7월 21일
심사완료 : 2004년 2월 11일

XML 데이타에 대한 질의는 데이타 그래프 내의 경로(일련의 레이블들)를 이용해 작성되며 질의 처리의 성능은 그래프 탐색에 따른 부담(overhead)에 좌우된다. XML데이타는 데이타의 불규칙성으로 인해 데이타 전체를 살펴보지 않고는 정확한 구조를 알아낼 수가 없으며 그래프 내의 경로의 길이가 매우(혹은 무한히) 길어질 수 있다. 따라서, 사용자가 질의를 작성하기 힘들다. 질의 작성의 편의와 데이타 내의 임의의 경로를 표현하기 위해 XML에 대한 질의 언어는 패턴을 이용해 경로를 표현한다. 경로를 표현하기 위한 패턴으로 주로 사용되는 것은 정규식(regular expression)[4-6]과 XPath식[7,8]이다. 비정형 데이타나 XML과 같이 구조가 불규칙한 데이타에 대한 질의를 표현하기 위해서 패턴을 이용한 경로의 표현은 필수적이다. 하지만 패턴을 포함하는 경로 질의는 데이타 그래프의 탐색범위를 매우 넓히게 되므로 질의처리 성능은 저하될 수 밖에 없다. 이러한 성능 저하를 막기 위해서는 효율적인 색인이 필수적이며 현재 여러 가지 색인 기법에 관한 연구가 진행 중이다.

XML 데이타는 그 구조가 고정되어 있지는 않지만, 유사한 구조가 반복되는 형태의 그래프로 만들어지며, 따라서 하나의 데이타 그래프 내에 동일한 경로가 많이 존재하게 된다. 이러한 특성을 이용해 기존의 연구는 데이타 그래프 내의 동일한 경로들을 하나로 합쳐서 데이타 그래프에 대한 구조요약[9-11]을 만들고, 구조요약을 색인으로 이용하는 몇 가지 기법들[9,12]을 제안하였다. 이렇게 만들어진 색인 역시 비정형 데이타와 같은 그래프 구조를 가지게 되므로 색인 그래프의 크기가 색인의 성능에 중요한 영향을 미친다. 이러한 그래프 기반의 색인 기법들은 데이타 그래프 내의 동일한 경로를 하나로 합치므로 일반적으로 데이타 그래프보다 색인 그래프의 크기가 작아질 것이라고 가정한다. 하지만 참조(reference) 에지를 가지고 있는 데이타 그래프에서 많은 경우 색인 그래프의 크기가 데이타 그래프의 크기와 유사하게 커져서 이러한 방법들은 실용적이지 못하다는 것을 알 수 있다(3.2에서 자세히 설명).

색인 그래프가 실용적으로 사용되기 위해서는 데이타 그래프의 구조와 상관없이 색인 그래프가 일정한 크기를 유지할 수 있어야 한다. 본 논문에서는 데이타 그래프 내의 가능한 모든 경로를 분할(partitioning)하고 질의 처리 시 분할영역(partition)을 빠르게 찾아낼 수 있는 색인 그래프를 제안한다. 본 논문에서 제안하는 색인 그래프는 데이타 그래프의 구조와 상관없이 원하는 크기로 생성할 수 있으며 따라서 메모리에 올라올 수 있는 크기로 구성하는 경우 색인 그래프의 탐색시간은 질의 처리의 부담(overhead)에서 제외할 수 있다.

본 논문은 다음과 같은 기여도(contribution)를 가진다.

1. 데이타 그래프 내의 경로에 대한 분할 기법을 제시한다. 경로를 분할하고 질의를 만족하는 분할영역만을 찾아내어 처리함으로써 질의처리 시 탐색범위를 크게 줄일 수 있다.
2. 기존의 그래프 기반 색인의 문제점을 제시하고 고정된 크기의 색인 기법을 제공한다. 색인의 크기를 메모리에 올라올 수 있는 크기로 구성함으로써 색인 그래프 탐색의 부담을 없앨 수 있다.
3. 기존의 연구를 확장하여 각 분할영역에 대한 효율적인 후처리 방법을 제시한다.
4. 실험을 통해 기존의 색인 기법과 본 논문의 색인 기법을 비교하고 제안하는 색인 기법의 우수성을 입증한다.

본 논문의 구성은 다음과 같다.

2장에서는 관련연구에 대해 살펴보고, 3장에서 본 논문에서 사용되는 데이타 모델과 경로식에 대해 설명하고 기존의 그래프 색인 기법에 대해 자세히 설명한다. 4장에서는 제안하는 색인 기법의 경로 분할 기법과 분할영역 처리방법에 대해 알아본다. 5장에서는 실험결과를 설명하고, 6장에서는 결론 및 향후 연구를 기술한다.

2. 관련연구

비정형 데이타와 XML은 약간의 차이가 있긴 하지만 일반적으로 XML은 비정형 데이타의 일종으로 취급한다. [13]에서는 비정형 데이타와 XML의 유사성과 차이점에 대해 설명하고 있다.

XML은 흔히 레이블이 있는 그래프 모델로 표현되며, XML에 대한 질의 언어는 일련의 레이블을 조합해 만든 경로를 이용해 질의를 표현한다. 특히 정규식을 사용해 경로를 표현함으로써 질의의 표현력을 증대 시키며, 데이타 그래프 내의 임의의 경로를 표시할 수 있도록 도와준다. XML 및 비정형 데이타를 위한 질의 언어는 [4-8]에서 찾아볼 수 있다.

비정형 데이타에 대한 구조 요약(data guide)은 비정형 데이타에 대한 스키마 역할을 수행한다. [9-11]은 비정형 데이타에 대한 스키마 추출 기법에 대해 설명하고 있다. [14]에서는 [10]을 색인 그래프로 사용할 수 있다는 것을 보여주며 [9]에서는 경로의 색인 역할을 수행하는 스키마 그래프에 대해 설명하고 있다. [9]는 비 결정적 오토마타를 결정적 오토마타로 변환하는 방식을 데이타 그래프에 적용해 색인 그래프를 생성한다.

[12]에서는 1/2/T-index의 색인들을 제안하고 있다. 1-index는 루트로부터 시작되는 경로의 집합이 동일한 노드들을 모아 인덱스를 구축하는 방법에 대해 설명한

다. 2-index는 데이타 그래프 내의 임의의 노드로부터 시작하는 경로를 처리하기 위한 색인으로, 하나의 질의 내에 여러 개의 경로식을 포함하는 경우에 사용할 수 있으며, T-index는 자주 쓰일 수 있는 정규 경로식에 대해 색인을 구축함으로써 질의 처리 성능을 향상시킬 수 있는 방법에 대해 설명한다.

[15]에서는 경로를 데이타 그래프 내에서 나타나는 레이블들을 알파벳으로 하는 문자열로 취급하여 문자열 탐색에 유용한 구조인 Patricia trie에 경로를 저장하는 방법에 대해 설명한다. 이때, 색인 탐색 시의 입출력 횟수를 줄이기 위해 Patricia trie에 대한 다단계 색인을 구축한다.

[16,17]에서는 트리 내의 임의의 두 노드에 대해서 조상 후손의 관계를 알아낼 수 있는 방법을 제시하고 이를 이용해 A//B형태의 질의를 최적화 하는 방법을 제안하고 있다. 하지만 이러한 방법들은 데이타가 반드시 트리형태여야 한다는 제약을 가지며, A/B와 같은 단순 경로를 따라갈 때에 비효율적이라는 단점을 가진다. [18]에서는 그래프 내의 임의의 두 노드에 대해 조상 후손 관계를 알아낼 수 있는 방법을 제시하고, 이를 이용해 하나의 질의 내에 여러 개의 경로식이 포함된 질의를 최적화 하는 방법을 제안하고 있다.

APEX[19]에서는 데이타 마이닝 기법을 이용해 자주 사용되는 경로를 빠르게 찾을 수 있도록 구조를 변경하는 색인 그래프에 대해 설명하고 있다. APEX은 기본적으로 부정확한 색인 그래프를 유지하고 있으며 자주 사용되지 않는 질의를 처리하기 위해서는 질의 수행 시 색인 그래프를 갱신해야 하는 문제점을 가지고 있다.

[20]은 부정확한 색인 그래프를 생성한 후 질의 처리 시 색인 그래프를 통해 결과를 포함하는 객체들의 집합을 얻고, 각각의 객체들을 데이타 그래프를 통해 유효화(validation)함으로써 최종적으로 결과를 얻어내는 색인 기법에 대해 설명하고 있다.

3. 배경 지식

3.1에서는 본 논문에서 사용되는 데이타 모델과 경로식에 대해 설명하고, 3.2에서는 기존의 대표적인 그래프 색인 기법인 1-index와 Dataguide에 대해 자세히 소개하며 이 기법들의 문제점을 살펴본다.

3.1 데이타 모델 및 경로식

본 논문에서는 XML과 비정형 데이타를 레이블이 있는 유향 그래프 G = (V, E, root, S_{OID}, C)로 표현한다. 여기서, V는 그래프의 각 노드를 나타내며, E = V X C X V를 나타낸다. 이때, C는 무한한 상수의 집합을 말한다. 데이타 그래프 내에는 다른 노드와 구별되는 root노드가 존재한다. 그래프 내의 각 노드 V는 스콜렘 함수 SOID로부터 유일한 객체 식별자를 할당 받는다.

XML의 경우 ID/IDREF를 통해 노드들 사이의 참조 관계를 표시하며, 이러한 참조 관계를 포함하는 XML데이타는 그래프 형태로 표현된다. 본 논문에서는 XML데이타의 노드의 종류(e.g. 엘리먼트, 에트리뷰트,…)를 구별하지 않으며, 노드들 사이의 순서정보를 사용하지 않는다.

데이타 그래프 내의 일련의 레이블을 경로라 부르며, 별다른 언급이 없는 이상 본 논문에서 사용되는 경로는 루트 노드로부터 시작되는 경로를 말한다. 본 논문에서는 패턴을 포함하는 경로를 정규식을 이용하여 표현하며 이러한 경로식을 정규 경로식이라 부른다. 정규 경로식 R은 다음과 같이 정의된다.

$$R ::= C \mid \_ \mid R.R \mid (R|R) \mid R^* \mid R^+$$

이때, 기호 _은 임의의 레이블과 매치될 수 있으며, 본 논문에서는 마지막 레이블이 _로 끝나는 정규 경로식을 인정하지 않는다.

3.2 기존의 그래프 기반 색인 기법 : 1-index, DataGuide

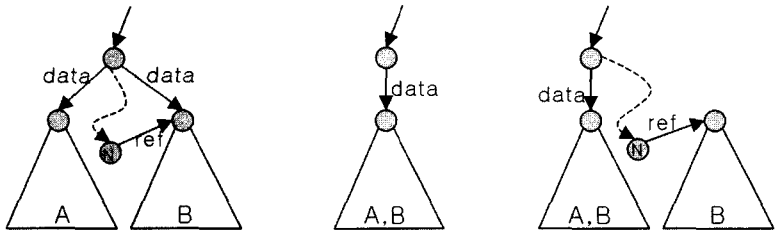
이 절에서는 두 가지 대표적인(major) 그래프 기반 색인인 1-index와 DataGuide에 대해 설명하고 그래프 기반 색인 기법의 문제점에 대해 설명한다.

DataGuide는 비결정적(non-deterministic) 오토마타를 결정적(deterministic) 오토마타로 변환하는 과정과 동일한 과정으로 데이타 그래프를 색인하는 기법이다. 일반적으로 비결정적 오토마타를 결정적 오토마타로 바꾸면 그 크기가 커진다. 하지만, XML과 같은 비정형 데이타는 데이타 그래프 내에 겹치는(동일한) 경로들이 많이 존재하여 비결정적 오토마타를 결정적 오토마타로 바꾸는 기법을 사용하여 색인을 만들면 색인의 크기가 줄어들 것이라고 기대할 수 있다.

1-index는 데이타 그래프 내의 각 노드들을 언어 동등 클래스(language equivalent class)로 나누어 만든 색인 그래프이다(여기서 언어란 경로 문자열을 말한다). 이 기법 역시 데이타 그래프 내에 동일한 경로가 매우 많이 존재한다는 점을 이용하는 색인 기법이다.

그래프 기반의 색인은 색인 역시 비정형 데이타가 되므로 색인을 탐색하는 부담이 질의 성능을 좌우한다. 따라서 색인 그래프의 크기는 데이타 그래프의 크기에 비해 매우 작은 크기로 만들어져야 실용적으로 사용할 수 있다. DataGuide나 1-index는 동일한 경로가 많이 존재하는 비정형 데이타의 특성 때문에 많은 경우에 색인의 크기가 작게 생성될 것이라고 가정한다. 하지만 이러한 색인 기법은 작은 크기의 색인을 만들지 못한다.

예를 들어, 그림 1의 (a)와 같이 구조가 동일한 두 개의 부 그래프 A와 B로 이루어진 데이타 그래프를 생각



(a) data graph, 1-index (b) 1-Index, Dataguide (c) Dataguide

그림 1 참조 에지와 색인 그래프

해 보자. 그림 1의 (a)에서 노드 N이 없다고 하면, DataGuide나 1-index를 이용해 이 데이타를 색인할 경우, 색인의 크기는 그림 1의 (b)와 같이 데이타 크기의 절반 이하로 줄어들게 된다. 왜냐하면 구조가 동일한 두 개의 부 그래프는 하나로 합쳐지며, 부 그래프 내에서도 크기의 감소가 생길 수 있기 때문이다. 하지만 그림 1의 노드 N처럼 데이타 그래프 내의 임의의 노드가 부 그래프 A의 루트 노드나 부 그래프 B의 루트 노드 중의 하나를 참조하고 있는 경우, 1-index나 Dataguide는 비효율적인 색인 그래프를 구성하게 된다. 1-index의 경우 부 그래프 B의 각 노드는 부 그래프 A의 해당 노드보다 더 많은 경로(노드 N을 포함하는 경로)를 가지므로 두 개의 부 그래프는 더 이상 언어 동등 노드를 가지지 못하게 된다. 따라서 두 부 그래프는 하나로 합쳐질 수 없다. DataGuide의 경우 부 그래프 A와 부 그래프 B가 하나로 합쳐진 후, 참조 노드에 부 그래프 B에 해당하는 그래프가 참조 에지로 연결되어야 한다[9]. 그림 1과 같은 경우에 만약 부 그래프 내에서 크기 감소가 일어나지 않는다고 하면, 1-index는 그림 1의 (a)와 같이 데이타 그래프와 완전히 동일해 지며, DataGuide는 그림 1의 (c)와 같이 데이타 그래프와 같은 크기의 색인 그래프가 된다.

위의 설명과 같이 데이타 내의 몇 개의 참조 에지로 인해 색인의 크기가 데이타 그래프의 크기와 비슷한 크기로 생성될 수 있으며 따라서 이러한 색인 기법들은 매우 제약적인 상황(예를 들어 데이타 그래프가 트리인 경우)에서만 사용될 수 있으며 실용적이지 못하다는 것을 알 수 있다.

4. 분할 색인

4.1에서는 본 논문에서 제안하는 색인 그래프와 경로 분할방법에 대하여 설명하고 4.2에서는 고정된 크기의 색인을 생성하는 방법에 대해 설명한다. 4.3에서는 분할 색인을 탐색해 찾아낸 분할영역들을 통해 질의의 결과를 얻어내는 방법에 대해서 기술한다.

4.1 분할 색인

데이타 탐색영역을 줄이기 위해 데이타를 분할(partitioning)하는 기법은 기존의 데이타베이스 관리 시스템에서 많이 사용되었다. 기존의 분할방식은 값들의 집합을 분할을 위한 함수(예를 들어 해시함수)를 이용해 서로 교집합을 가지지 않는 여러 개의 부분집합으로 분할하는 방식이다. 이러한 기존의 분할 방식을 편의상 값 기반 분할방식이라 부르기로 한다. XML과 같은 비정형 데이타는 질의의 대상이 데이타 내의 경로이므로 이러한 데이타를 분할하는 경우 분할의 대상은 단순히 단말 노드의 값이 아니라 데이타 내의 경로가 되어야 한다. 경로를 분할하는 쉬운 방법은 문자열로 표현되는 경로의 집합을 값 기반 분할방식을 사용해 분할하는 것이다. 하지만 값 기반 분할방식으로 경로가 분할되어 있는 경우, 정규 경로식을 포함하는 질의에 대해 질의를 만족시키는 분할영역을 찾아내는 것이 불가능하다.

본 논문에서는 데이타 그래프 내의 가능한 모든 경로를 분할하는 분할방식과, 정규 경로식을 포함하는 질의에 대해서 질의의 결과를 포함하는 분할영역을 쉽게 찾을 수 있는 색인 그래프를 제안한다. 기존의 그래프 기반의 색인들이 색인 그래프의 각 노드에 객체들의 집합인 extent를 저장하는 것과는 달리 본 논문의 색인 그래프는 그래프의 각 노드에 경로들의 집합인 경로 분할영역(path partition)을 저장한다.

다음의 정의 1과 정의 2는 분할 색인을 설명하기 위해 필요한 정의이다.

정의 1. 데이타 그래프 내에서 들어오는(incoming) 에지의 레이블이 서로 동일한 노드들을 레이블 동등 집합이라 부른다. 레이블 L에 대한 레이블 동등 집합을 L_{eq}로 표시한다.

정의 2. 레이블 동등 집합 내의 각 노드의 나가는(outgoing) 에지의 레이블들을 모두 합쳐 놓은 집합을 외부집합이라고 부른다(외부집합은 집합이므로 중복된 레이블은 하나만 포함한다).

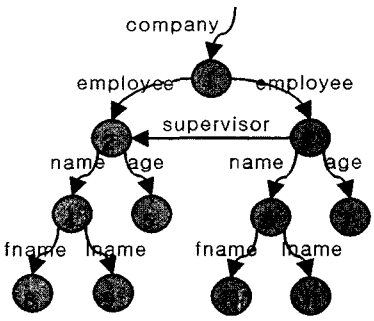
분할 색인은 데이타 그래프로부터 만들어지는 색인

그래프로 데이타 그래프 내의 서로 다른 레이블마다 하나씩 색인 노드를 생성하며, 색인의 각 노드는 데이타 그래프 내의 경로에 대한 분할영역을 저장한다. 분할 색인의 노드와 에지는 정의 3과 정의 4와 같이 구성된다.

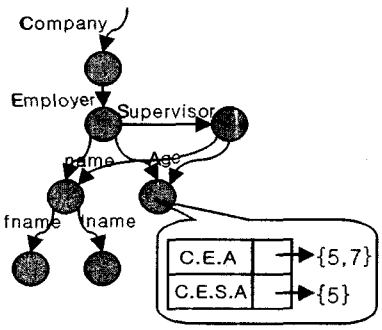
정의 3. 분할 색인 노드: 데이타 그래프 내의 어떤 레이블 L에 대해, 분할 색인의 노드 V는 (L, P, O)로 구성된다. 여기서 O는 L_{eq}에 대한 외부집합이며, P는 L_{eq}내의 모든 노드에 대해, 마지막 레이블이 L로 끝나는 경로를 모아놓은 집합(또는 집합에 대한 포인터)이다. 즉, P는 데이타 그래프 내의 경로들에 대한 하나의 분할영역(Partition)이 된다. 이때, L을 노드 V의 식별 레이블이라 부른다.

정의 4. 분할 색인 에지: 분할 색인 내의 각 노드는 다음과 같은 방법으로 연결되어 있다. 분할 색인 내의 한 노드 N에 대해 N의 식별 레이블을 L이라고 하면, 색인 그래프 내에서 외부집합에 L을 포함하고 있는 모든 노드는 N으로 향하는 에지를 가지게 되며 이 에지의 레이블은 L이 된다.

그림 2는 예제 데이타와 해당 분할색인을 나타낸다. 분할색인 내의 각 노드는 해당 경로 집합(partition)을 가지며 각 경로는 그 경로에 해당하는 객체의 집합(extent)에 대한 포인터를 가지게 된다.



(a) 예제 데이타



(b) 분할 색인 및 분할 영역

그림 2 분할 색인

알고리즘 1은 분할 색인 생성 알고리즘이다. 알고리즘 1에서 makePartitionIndex 함수는 데이타 그래프의 루트 노드를 인자로 받아들여 색인 그래프의 루트 노드를 생성한 후 recursiveMake함수를 호출한다. 이때, 알고리즘 1의 5번째 줄에서와 같이 색인 그래프의 노드는 노드 내에 분할영역(partition)을 포함하며, 분할 영역은 <경로, extent>쌍의 집합으로 표현된다. 알고리즘 1에서 경로는 stack과 같은 자료 구조로 표현되며, push-Label(*l*)은 현재 경로에 *l*을 마지막 레이블로 추가하는 역할을 수행하고, popLabel()은 경로의 마지막 레이블을 제거하는 역할을 수행한다. recursiveMake함수는 새로운 색인 노드를 생성하거나 기존에 존재하는 색인 노드들 사이에 에지를 연결하는 역할을 한다. recursive-Make함수의 두 번째 인자인 extent는 현재 탐색되고 있는 데이타 그래프 내의 노드들의 집합을 나타낸다. recursiveMake 함수는 extent내의 각 노드들로부터 도달될 수 있는 노드들을 같은 레이블을 가지는 노드들끼리 묶어, 새로운 색인 노드를 생성하거나 기존의 색인 노드에 추가한다. recursiveMake의 세 번째 인자인 path는 데이타 그래프 내에서 현재 탐색하고 있는 노드에 대한 경로를 포함하고 있다. 알고리즘 1의 27행에서 appendPath(path, $eset_1$)함수는 현재 탐색하고 있는 경로(path)와 해당 데이타 노드들($eset_1$)를 색인 노드의 분할영역에 추가하는 역할을 수행한다. 이때, 추가하려는 경로가 이미 색인 노드의 분할영역에 존재하는 경우 그 경로에 해당하는 extent에appendPath의 두 번째 인자인 eset1의 각 원소를 추가한다.

알고리즘 1. 분할색인 생성 알고리즘

```
01: makePartitionIndex(DataNode root)
02: {
03:     index_root := new IndexNode();
04:     index_root.label := root.label; // 식별 레이블
05:     path = new Path();
06:     path.pushLabel(root.label);
07:     index_root.partition := { <path, {root}> }   // path is copied
08:     recursiveMake(index_root, {root}, path)
09: }
10:
11: recursiveMake(IndexNode node, Extent extent, Path path)
12: {
13:     label_set := ∅;
14:     for each object o in extent do
15:         label_set := label_set ∪ {l | l is an outgoing label(edge) from o};
16:     for each unique label l in label_set do{
17:         nxt_node := findInHash(l);
18:         if(nxt_node = NULL){
19:             nxt_node = new IndexNode();
20:             nxt_node.label = l; // 식별 레이블
21:             addToHash(nxt_node);
22:         }
23:         node.appendChild(nxt_node);
24:         eset1 := a set of objects reached from the objects in extent by l;
25:         eset2 := all object in the partition of nxt_node;
26:         path.pushLabel(l);
27:         nxt_node.appendPath(path, eset1);
28:         recursiveMake( node, eset1-eset2, path);
29:         path.popLabel();
30:     }
31: }
```

아래의 Lemma와 Theorem은 분할 색인의 특성을 설명하며, 분할 색인이 데이타 그래프 내의 경로를 올바르게 분할하며 데이타 그래프를 통해 얻을 수 있는 모든 결과는 색인을 통해서도 얻을 수 있다는 것을 보여준다.

Lemma 1. 분할 색인의 각 노드는 서로 다른 식별 레이블을 가진다.

증명 : 만약 분할 색인 내의 두 노드 (L, P1, O1)과 (L, P2, O2)가 같은 식별 레이블 L을 가진다고 하면, 정의 1과 정의 3에 의해 P1과 P2는 동일한 집합이 되고 정의 2에 의해 O1과 O2도 같은 집합이 된다. 따라서 두 노드는 동일한 노드이다.

Lemma 1.에 의해 분할 색인의 크기는 데이타 그래프 내의 서로 다른 레이블의 수와 동일하다는 것을 알 수 있다.

Theorem 1. 분할 색인 내의 각 분할영역 사이에는 교집합이 존재하지 않는다.

증명 : 분할 색인의 두 노드 N1과 N2의 분할영역을 P1과 P2라 하자. Lemma 1.에 의해 N1과 N2는 서로 다른 식별 레이블을 가진다. 이때, 정의 3에 의해 분할영역 P1의 모든 경로는 N1의 식별 레이블로 끝나고, 분할영역 P2내의 모든 경로는 N2의 식별 레이블로 끝나는 경로 이므로 P1과 P2는 교집합을 가질 수 없다.

Theorem 2. 분할 색인은 데이타 그래프 내의 가능한 모든 경로를 분할한다.

증명 : 데이타 그래프 내의 임의의 경로 P에 대해, 경로 P의 마지막 레이블을 L이라 하고 데이타 그래프 내에 경로 P가 가리키는 노드를 N이라 하자. 정의 3에 의해 색인 내에는 L을 식별 레이블로 가지는 노드(분할영역)이 존재하며, 노드 N은 L_{eq}에 포함되므로 따라서 경로 P는 L을 식별 레이블로 가지는 노드의 분할영역에 포함된다.

Theorem 3. 데이타 그래프 내에 존재하는 경로는 색인 그래프에도 존재한다.

증명 : 정의 3에 의해 데이타 그래프 내의 경로의 길이가 1인 경로들은 색인 그래프 내에도 존재한다. 데이타 그래프 경로의 길이가 k인 경로들이 색인 그래프 내에 존재한다고 가정하자. 데이타 그래프 내의 길이가 k인 어떤 경로 P를 생각해보자. P의 마지막 레이블을 L이라고 할 때, 정의 3에 의해 색인 그래프 내에는 L을 식별 레이블로 가지는 노드 N이 존재하며, 위의 가정에 의해 루트부터 노드 N까지 P인 경로가 존재한다. 이때, 정의 3에 의해 노드 N은 데이타 그래프 내에서 경로 P가 가리키는 모든 노드를 포함하고 있으며, 정의 4에 의해 데이타 그래프 내에 P를 prefix로 하는 길이 k+1인 경로가 색인 그래프에도 만들어지게 된다.

위와 같은 방법으로 만들어지는 분할 색인은 1-RO[23]와 동일한 형태의 그래프로 만들어지지만, 분할 색인의 각 노드에 경로에 대한 분할영역을 저장한다는 것이 다른 점이다. 사실상 이러한 분할 방식은 경로의 마지막 레이블이 동등한 집합으로 데이타 그래프 내의 모든 경로들을 분할한 것과 동일하다. 따라서 질의 처리시 색인 그래프를 탐색하지 않고 질의 내의 경로식의 마지막 레이블을 보고 해당 분할영역을 직접 찾아낼 수 있다. 만약 이렇게 분할된 각 분할영역의 크기의 편차가 크지 않은 경우에는 이러한 분할방식이 유용하게 사용될 수 있으며, 따라서 색인 그래프가 필요 없다. 하지만 분할영역에 치우침이 생기는 경우(즉, 분할영역 사이의 크기의 편차가 커지는 경우) 효율적인 색인을 보장하지 못한다. 분할영역 사이의 크기의 편차를 줄이기 위해 크기가 특정 값(threshold)보다 큰 분할영역과 그 분할영역을 포함하는 노드를 2개로 분할(split)한다. 이러한 과정을 노드분할이라고 부른다. 다음은 노드분할을 설명하기 위한 정의이다.

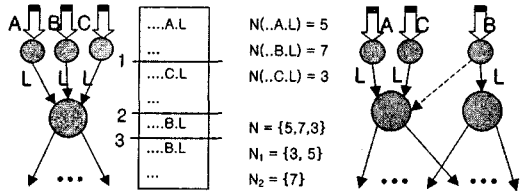
정의 5. 데이타 그래프 내의 임의의 레이블 L에 대해 N(L)은 L로 끝나는 경로의 수를 나타낸다. 어떤 경로 집합 (또는 분할영역) P에 대하여, N(P)는 P내의 경로의 수를 나타낸다.

정의 6. 길이가 k인 경로 $l_1.l_2.\cdots.l_k$에 대해 l_m을 이 경로의 *anchor*(k-m) 레이블이라고 부른다. 특히, *anchor* (1) 레이블(l_{k-1})을 경로의 anchor또는 *anchor 레이블*이라고 부른다.

데이타 그래프 내에 n개의 서로 다른 레이블을 L_1, ..., L_n이라고 하자. 이때 분할영역의 개수는 n이 되며 각 분할영역의 크기는 $N(L_1)$, ..., $N(L_n)$로 나타낼 수 있다. $N(L_1)$, ..., $N(L_n)$의 평균을 E라고 하고, $E - N(L_m) > \delta$인 분할영역(partition)들에 해당하는 색인 그래프 내의 노드들의 집합을 S라고 할 때, S내의 각 노드를 분할(split)함으로써 분할영역의 크기를 줄일 수 있다. 분할 영역의 크기의 치우침을 최소화하기 위해 이러한 과정을 반복하여 수행할 수도 있다.

노드 분할 시, 분할된 두 노드에 분할되기 전 노드의 들어오는 에지를 분배하게 된다. 이때, 색인 그래프의 비결정적 속성(nondeterministic feature)[1]이 증가하지 않게 하기 위해 각 에지는 분할된 두 노드 중 하나의 노드에 분배되어야 하며 두 노드에 모두 분배되지 않도록 해야 한다. 그림 3은 노드 분할 방법을 설명하고 있다. 그림 3의 (a)에서 분할영역(partition)을 1또는 2번 라인으로 분할(split)하는 경우 분할된 두 노드에 중복되지 않게 에지가 분배되지만, 3번 라인으로 분할하는 경

1) 색인 그래프의 비결정적 속성이 증가할수록 질의 처리시 그래프의 탐색 범위가 커지며 따라서 색인 그래프의 효율성이 감소한다.



(a) 분할(split) 전 (b) anchor별 경로 수 (c) 분할(split) 후

그림 3 노드 분할

우 그림 3의 (c)의 점선과 같이 분할된 두 노드에 중복되어 분배되는 에지가 생기게 된다. 그림 3에서 3번 라인은 anchor레이블이 B인 두 경로를 서로 다른 분할영역에 속하도록 분할하였기 때문에, 에지 분배의 중복이 생기게 되었다. 따라서 anchor가 같은 두 경로는 서로 동일한 분할영역에 속하도록 노드를 분할하여야 한다. 이러한 방법으로 노드를 분할할 때, 분할된 두 노드의 분할영역(partition)의 크기의 편차가 가장 작아지는 방법으로 분할하여야 한다. 예를 들어 그림 3에서 1번 라인과 2번 라인으로 분할하는 것이 모두 올바른 분할 방법이지만, 2번 라인으로 분할 하는 것이 두 분할 영역의 크기의 편차가 가장 작아지므로 2번 라인으로 분 할 하였다. 일반적으로 anchor별 경로의 수를 나타내는 집합 $N = \{m_1, \cdots, m_k\}$(그림 3의 (b) 참고)을 $N_1 \cap N_2 = \emptyset$ 이고 $N_1 \cup N_2 = N$이며 두 집합의 원소의 합의 차가 최소가 되는 두 개의 부분집합 N_1과 N_2로 나누는 것은 NP문제에 속한다. 본 논문에서는 집합 N의 원소를 내림차순으로 정렬한 후, 정렬된 원소를 앞에서부터 더해 그 값이 전체 합의 절반보다 커 지는 지점에서 두 개의 집합으로 나누는 근사 알고리즘을 사용하였다.

알고리즘 2는 노드 분할 알고리즘이다. 알고리즘 2에서는 동일한 anchor레이블을 가지는 경로의 수를 표현하기 위해 Anchor라는 자료 구조를 사용하였다. 7행의 makeAnchorList함수는 인자로 넘겨 받은 분할영역 내에서 각 경로의 anchor를 조사해 <anchor label, 경로수>쌍의 리스트(집합)을 만드는 함수이다. 9행에서 16행까지는 분할 영역을 두 부분으로 자르되, 두 분할 영역의 크기의 편차가 최소가 되게 하는 근사 알고리즘이다. 18행부터 26행까지는 새로운 색인 노드 2개를 생성한 후 분할영역내의 각 경로를 각 노드로 분배하는 일을 수행한다. 28행부터 33행까지는 분할된 노드와 부모 노드들 사이의 에지를 연결하는 과정이며, 35행부터 46행까지는 분할된 노드와 자식 노드들 사이의 에지를 연결하는 과정을 보여준다.

노드 분할 시 색인 그래프 내의 경로가 추가되거나 감소되지 않으며 분할되는 분할 영역은 서로 겹치지 않게 분할되므로, 노드 분할 후의 분할색인 역시 Theorem

알고리즘 2. 분할색인의 노드분할 알고리즘

```
01: struct Anchor {
02:     string label; // anchor label
03:     integer count;    // number of paths whose anchor is label
04: };
05: splitNode(IndexNode node){
06:     partition = node.partition;
07:     vector<Anchor> a_list := makeAnchorList(partition);
08:
09:     sort a_list by the count field in descending order;
10:     idx := size := 0;
11:     for(idx := 0; idx < a_list.size(); idx++){
12:         size := size + a_list[idx].count;
13:         if(size >= partition.size/2) break;
14:     }
15:     if(size - partition.size/2 > partition.size/2 - size)
16:         size := size - a_list[idx--].count;
17:
18:     s_node1 := new IndexNode();
19:     s_node2 := new IndexNode();
20:     s_node1.label := s_node2.label := node.label;
21:     for each entry e in partition do{ // e consists of (path, extent)
22:         anchor := the anchor of e.path;
23:         position := a_list.find(anchor);
24:         if(position <= idx) insert e into the partition of s_node1;
25:         else insert e into the partition of s_node2;
26:     }
27:
28:     for each parent p of node do{
29:         p.removeChild(node);
30:         position := a_list.find(p.label);
31:         if(position <= idx) p.appendChild(s_node1);
32:         else p.appendChild(s_node2);
33:     }
34:
35:     for each child c of node do {
36:         for each path p in c.partition do{
37:             child_of_s_node1 := child_of_s_node2 := false;
38:             if(anchor of p = node.label){
39:                 position := a_list.find(anchor(2) of p);
40:                 if(position <= idx) child_of_s_node1 := true;
41:                 else child_of_s_node2:= true;
42:             }
43:         }
44:         if(child_of_s_node1) s_node1.appendChild(c);
45:         if(child_of_s_node2) s_node2.appendChild(c);
46:     }
47:     delete node from the index;
48: }
```

1, 2, 3를 여전히 만족시킨다는 것을 쉽게 알 수 있다.

4.2 고정 크기 분할 색인

분할 색인의 크기는 데이타 그래프 내의 서로 다른 레이블의 수와 분할되는 노드의 수에 의해 결정된다. 만약 데이타 그래프 내에 서로 다른 레이블의 수와 분할되는 노드의 수가 많은 경우 색인 그래프의 크기가 증가하게 된다. 색인 그래프의 크기(노드의 개수)가 증가한다는 것은 각 분할영역이 작은 크기로 생성된다는 것을 의미한다. 이와 같이 작은 크기의 많은 분할영역이 생성되는 것을 막기 위해 본 논문에서는 고정된 크기의 분할색인을 생성하는 방법을 제공한다.

정의 7. $m^k(l)$는 레이블 l을 1에서 k사이의 정수(새로운 레이블)로 대응시키는 함수이다.

고정 크기의 분할 색인은 다음과 같은 방법으로 만들어질 수 있다.

가. 알고리즘 1을 이용해 데이타 그래프를 색인한다.

나. 색인 그래프의 각 레이블을 $m^k(l)$함수를 이용해 변환한다.

다. 변환된 색인 그래프 에서 레이블이 서로 동일한 노드들을 하나로 합친다.

라. 알고리즘 2를 이용해 변환된 색인 그래프의 노드를 분할한다.

위와 같은 방법으로 색인 그래프를 만들 때, 노드 분할 후의 분할 색인의 크기변화에 대해 살펴보자. 노드를 분할하기 전(단계 3에서)의 색인의 크기는 k이하이다. 노드 분할 시 δ(threshold)값을 조절해 분할되는 노드의 개수를 결정할 수 있으므로 전체 노드 중 절반의 노드만이 분할되도록 값을 결정하는 경우 분할 색인의 크기는 $k+k/2$가 된다. 즉, 분할 색인의 크기는 $m^k()$에 의해 결정된다. 따라서 색인을 위해 사용 가능한 메모리의 크기가 M이라고 할 때, k값을 $2/3M$으로 정함으로써 항상 메모리에 올라올 수 있는 크기의 색인을 생성하는 것이 가능하다. 이러한 방법으로 분할 색인을 항상 메모리에 올라올 수 있는 크기로 생성할 수 있으며 따라서 질의 처리를 위한 색인 탐색 시 입출력 부담이 없으므로 효율적으로 분할영역을 찾아낼 수 있다.

이렇게 만들어진 색인 그래프를 이용해 질의를 처리하기 위해서는 질의 내의 경로식에 포함된 각 레이블을 $m^k()$함수를 이용해 변환한 후 색인 그래프를 탐색해야 한다.

$m^k()$함수는 데이타 그래프 내에 서로 다른 레이블의 수가 적은 경우 분할 영역의 크기의 편차가 가장 작아질 수 있도록 하는 변환 테이블로 만들 수 있다. 만약 데이타 그래프 내의 서로 다른 레이블의 수가 많아서 질의 처리시 매번 변환 테이블을 탐색(lookup)하는 것이 성능에 지장을 주는 경우, 일반 해시 함수를 $m^k()$함수로 사용할 수도 있다.

4.3 분할영역 처리

분할 색인의 각 분할 영역 내에 있는 경로들을 조사함으로써 질의를 처리할 수 있다. 만약 색인 그래프를 이용하지 않고 분할 영역들을 모두 조사하여 질의를 처리하는 경우 분할 색인 기법은 경로 테이블 기법과 동일해 질 수 있다. 하지만, 분할 색인 기법은 색인 그래프를 이용해 질의의 결과를 포함하지 않는 많은 수의 분할 영역들을 미리 제거하므로, 경로 테이블 기법보다 우수한 성능을 보인다. 분할 색인을 이용해 질의를 처리하는 방법은 (1) 색인 그래프를 탐색해 질의에 맞는 분할 영역을 찾아낸 후, (2) 찾아낸 분할 영역의 경로들을 조사해 결과를 얻어내는 것이다. 이때, (1)과정은 기존의 색인 그래프 탐색과 동일한 방법으로 진행된다. 이 절에서는 (2)과정의 처리의 최적화에 관해 설명한다.

색인 그래프를 통해 질의에 해당하는 분할영역을 찾은 후, 분할영역을 처리해 질의의 최종 결과를 얻어내야 한다. 질의 내에 포함된 정규 경로식을 오토마타로 만들 수 있으며, 본 논문에서는 이 오토마타를 질의 오토마타라고 부르기로 한다. 분할영역은 기본적으로 질의 오토마타를 만든 후, 이 오토마타에 분할영역 내의 각 경로를 통과시킴으로써 처리된다.

노드분할을 통해 분할영역의 편차를 줄였지만, 여전히 크기가 큰 분할영역이 존재할 가능성이 있으며, 이러한 분할영역들의 처리를 최적화하기 위해 본 논문에서는 분할영역의 각 경로를 질의 오토마타를 통과시키는 방법을 포함해 다음과 같이 세 가지의 분할영역 처리방법을 제시한다.

1. 질의 오토마타를 생성한 후 분할영역 내의 각 경로를 오토마타를 통과시켜 결과를 얻는다.
2. [21]의 방법을 이용해, 분할영역 내의 각 경로에 대한 시그니처(signature)를 생성하고, 이 시그니처를 통해 분할영역의 경로들을 걸러낸다. 걸러진 경로들을 1번의 방법을 사용하여 결과를 얻어낸다.
3. 데이타 그래프가 아닌 데이타 그래프 내의 경로에 대한 색인 기법인 Index Fabric[15]의 기법을 이용해 분할영역을 색인한 후 색인을 이용해 결과를 얻어낸다.

질의 오토마타를 이용하는 방법은 분할영역의 크기가 작아 메모리로 읽어 들이는 부담이 질의 처리의 성능에 별 영향을 미치지 않는 경우에 사용할 수 있는 방법이다. 시그니처를 이용하는 방법은 분할영역의 크기가 질의 처리 성능에 약간의 영향을 미칠 수 있을 만큼 큰 경우에 사용할 수 있는 방법이다. 이 방법을 사용하기 위해서는 색인 생성시 각 분할영역에 시그니처 정보를 저장해 두어야 한다. 이 방법은 만약 분할영역 별로 클러스터링이 잘 되어 있다고 하면, 시그니처를 이용해 분할영역의 경로를 걸러내는 것이 디스크 입출력의 횟수를 줄이지 못할 수도 있다는 단점을 가진다. 분할영역의 크기가 매우 커 질의처리 성능이 저하될 경우 Index Fabric의 기법을 이용해 분할영역 내의 경로들을 색인할 수 있다.

5. 실 험

이 절에서는 본 논문의 색인 기법인 분할색인의 성능에 대해 살펴본다. 우선, 본 논문에서 사용된 실험을 위한 플랫폼과 실험 데이타, 그리고 실험에 사용된 경로 질의에 대해 기술하고, 기존의 대표적인 색인 그래프인 Dataguide와 성능 비교를 한다. 이후에 노드분할에 따른 성능 변화를 살펴보며 마지막으로 본 논문의 실험결과를 요약한다.

5.1 실험 플랫폼과 데이타

본 논문에서는 실험을 위한 간단한 객체 저장 시스템을 구현하였다. 구현된 시스템은 단일 사용자 시스템으로 4K 크기의 페이지와 50개의 페이지 버퍼를 가지며, 본 논문의 성능 평가는 디스크와 페이지 버퍼 사이의 페이지 I/O의 횟수로 측정되었다. 각 디스크 페이지는 데이타/색인 그래프의 노드들과 분할영역 그리고 데이타 그래프 내의 노드들의 OID집합인 extent를 포함한다.

디스크 페이지 내의 데이타/색인 그래프의 노드는 기본적으로 각 노드의 레이블과 자식 노드들의 OID집합을 가지고 있으며, Dataguide 노드의 경우 추가로 extent에 대한 OID를, PartitionIndex 노드의 경우 추가로 분할 영역에 대한 OID를 가지고 있다. 본 논문의 플랫폼은 일반적인 객체 저장 시스템이 동작하는 것과 정확히 일치하며, 본 논문의 저장 시스템이 사용하는 메모리는 앞서 언급한 50개(2M)의 페이지 버퍼가 전부이다.

본 논문의 실험은 인터넷 영화 데이타베이스(IMDB)[22]와 XML벤치마크인 Mark[23]의 데이타를 사용하였다. XMark데이타는 경매에 관련된 내용을 포함하는 데이타로, 데이타 내에 많은 수의 참조 에지를 포함하고 있으며, [23]에서 제공되는 프로그램을 사용해 원하는 크기의 데이타를 생성할 수 있다. 본 논문에서는 10M크기의 XMark데이타를 생성하여 사용하였다. IMDB데이타는 작은 크기를 가지는 많은 수의 배우 파일들과 영화 파일들로 구성되어 있다. 배우 파일은 배우에 관한 정보와 그 배우가 출연한 영화들로 구성되어 있으며, 영화 파일은 영화에 관한 정보와 영화에 출연한 배우들로 구성되어 있다. 본 논문에서는 배우 파일들과 영화 파일들을 하나로 묶은 후, 각 배우 엘리먼트 내의 영화 정보들을 해당 영화 엘리먼트로 참조 에지를 만들고, 영화 엘리먼트 내의 배우 정보를 해당 배우 엘리먼트로의 참조 에지를 만들었다. 이때, 영화 파일이나 배우 파일에서 누락되는 정보들은 모두 삭제하였다. 이렇게 만들어진 영화 데이타베이스는 직관적으로 알 수 있듯이 많은 참조에지를 가지고 있으며, 각 배우 엘리먼트들은 자신이 출연한 영화에 대한 포인터를 포함하고 있으므로, 데이타 내에 많은 수의 사이클이 생기게 된다. 하지만 데이타 자체가 매우 단순하고 규칙적이어서, 이 데이타에 대한 Dataguide는 매우 작은 크기로 생성된다.

5.2 경로 질의

본 논문에서는 경로에 대한 질의를 단순 경로 질의와 정규 경로 질의로 나누어 실험하였다. 이때, 단순 경로 질의는 데이타 그래프 내에서 무작위로 경로를 추출된 경로를 말하며 정규 경로 질의는 단순 경로 내의 몇 개의 레이블들을 '_*'로 치환한 경로를 말한다. 정규 경로 질의는 다시 '_*'의 위치에 따라 '_*'가 제일 앞에 위치하는 경우와 '_*'가 중간에 위치하는 경우로 나누어 실험하였다. '_*'를 포함하는 질의는 색인 그래프의 탐색 범위를 매우 넓혀 색인 그래프의 크기가 질의 처리 성능에 얼마나 큰 영향을 미치는지 알 수 있게 해준다. 특히, '_*'가 제일 앞에 위치하는 경우, 색인 그래프 전체를 탐색해야 하므로 색인 그래프의 노드의 수가 질의 처리의 성능을 크게 좌우하게 된다.

5.3 데이타 및 색인 그래프의 노드 수

그림 4는 본 논문에서 사용한 데이타의 크기와 각 데이타에 대한 Dataguide와 분할색인의 색인 그래프의 노드 수 를 보여주고 있다. 이때, 분할색인의 색인 그래프의 노드 수는 노드분할을 1회 실시한 경우의 수이다. XMark데이타에 대한 Dataguide의 경우 실험에서 사용되는 데이타의 참조 에지들로 인해 색인의 노드 수가 데이타의 노드 수와 거의 유사하게 증가하는 것을 알 수 있다.

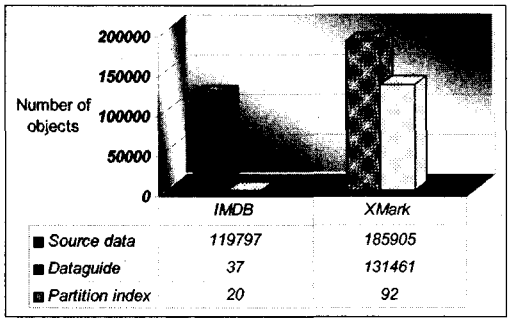


그림 4 데이타와 색인 그래프의 노드 수

그림 5는 노드분할에 따른 분할색인의 분할영역의 크기 변화를 보여준다. 분할영역의 크기는 분할색인의 성능에 영향을 미치게 되므로, 노드분할은 분할 색인의 성능향상을 위한 필수 조건이라고 할 수 있다. 그림 5에서 볼 수 있듯이 노드분할을 통해 분할 영역 사이의 편차를 크게 줄일 수 있다.

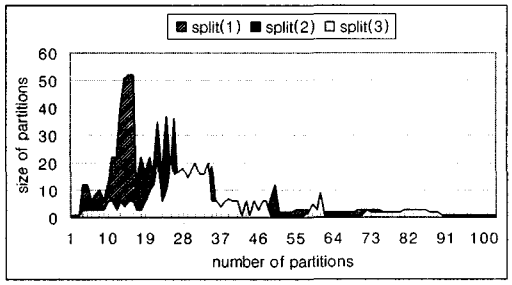


그림 5 XMark데이타에 대한 분할영역의 크기 변화

5.4 경로 질의 처리 결과

그림 6은 경로 질의의 처리 결과를 보여준다. 그림 6에서 실험 결과는 Dataguide의 결과에 따른 비율로 나타난다. 이때, 분할 색인의 색인 그래프는 노드분할을 1회 실시한 색인이다. 단순 경로 질의와 정규 경로 질의를 각각 50개씩 생성하여 실험한 후 평균값을 결과로 사용하였으며, 성능은 I/O횟수로 측정하였다.

그림 6에서 볼 수 있듯이, 단순 경로 질의의 경우,

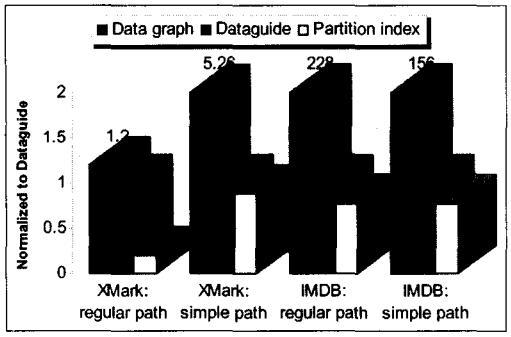


그림 6 질의 처리 결과

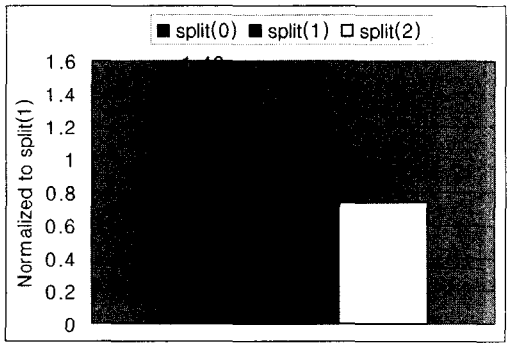


그림 7 분할 영역 처리 비용

Dataguide와 분할색인은 거의 유사한 성능을 보인다. 단순 경로 질의의 경우, Dataguide는 결정적 속성으로 인해 탐색해야 하는 노드의 수가 질의 내의 경로의 길이에 선형적이다. 하지만, Dataguide자체의 노드 수가 매우 많아 여러 페이지에 걸쳐 저장 되기 때문에, 최악의 경우 Dataguide 내의 각 노드를 탐색할 때마다 입출력이 발생할 수 있다. 반면에, 분할 색인의 경우 단순 경로 질의에 대해서도 색인 그래프의 많은 영역을 탐색해야 하지만, 색인 그래프의 크기가 메모리에 한번에 올라올 수 있을 만큼 작게 생성되므로 입출력의 부담이 Dataguide보다 훨씬 작다. 분할 색인의 경우 분할영역을 처리하는데 드는 비용을 포함해야 하므로 단순 경로 질의에 대해 Dataguide와 유사한 성능을 보이게 되는 것이다

그림 6에서 _*를 포함하는 정규 경로 질의의 경우 분할색인은 단순 경로의 처리시와 유사한 성능을 보이지만, Dataguide의 경우 데이타 그래프를 직접 탐색하는 것만큼 좋지 않은 성능을 보인다. 이것은 _*가 색인 그래프의 탐색범위를 매우 넓히기 때문이다. 특히, _*로 시작하는 질의의 경우 색인 그래프 전체를 탐색해야만 질의의 결과를 얻을 수 있으므로, Dataguide의 결정적 속성은 색인의 성능 향상에 도움을 줄 수 없다.

분할 영역의 크기는 색인 그래프의 탐색과 마찬가지로 분할 색인의 성능에 중요한 영향을 미친다. 4장에서 설명한 바와 같이 분할 영역을 나눔으로써 분할 영역의 처리 속도를 증가시킬 수 있다. 그림 7은 분할 색인의 노드 분할에 따른 분할 영역의 처리 속도의 변화를 보여준다. 노드 분할을 2회 실시한 경우 노드 분할을 하지 않았을 때보다 분할 영역의 처리 속도가 2배로 증가되는 것을 알 수 있다.

6. 결 론

본 논문에서는 XML과 같은 그래프 기반의 데이타에 대한 효과적인 색인 방법인 분할색인 기법을 제안하였다. 분할 색인은 데이타 그래프 내의 가능한 모든 경로를 분할(partitioning)하고, 질의 처리시 질의를 만족시키는 분할영역을 빠르게 찾을 수 있는 색인 그래프를 제공하는 색인 방식이다. 제안된 분할색인 방식은 데이타 그래프를 전혀 탐색할 필요가 없으며, 색인 그래프가 데이타 그래프의 구조에 비교적 영향을 받지 않으므로 다양한 데이타에 대해 안정적인 성능을 보장한다는 것이 특징이다.

분할색인은 색인 그래프의 크기를 고정시킴으로써 색인 그래프 탐색의 부담을 없앴으며, 기존의 연구를 이용해 각 분할영역의 처리를 최적화함으로써, 효율적인 질의처리를 지원한다.

본 논문에서는 제안된 분할색인을 구현하였고, 실제 사용되는 데이타와 임의로 가공된 데이타를 통해 색인의 성능을 보였다. 실험에서 분할색인은 기존의 색인기법보다 훨씬 더 작은 크기로 생성되며, 정규 경로 질의에 대해 10배 가량의 성능향상을 보임을 증명하였다.

색인 그래프의 갱신은 갱신되는 노드에 해당하는 분할영역을 수정함으로써 쉽게 이루어질 수 있다. 하지만, 데이타의 갱신으로 인해 특정 분할영역의 크기가 매우 커지거나 매우 작아지는 경우에 노드분할 또는 노드합병이 필요하며 이러한 경우에 대해 향후에 연구할 예정이다. 특히 보다 정교한 색인 그래프를 만들기 위해 *anchor*(1)대신 *anchor*(k)를 이용하는 노드분할 방법을 연구할 예정이다.

참 고 문 헌

[1] Tim Bray, Jean Paoli, and C. M. Sperberg-McQueen. Extensible markup language (XML) 1.0. W3C Recommendation, 1998.

[2] Arnaud Le Hors, Philippe Le Hegaret, Lauren Wood, Gavin Nicol, Jonathan Robie, Mike Champion, and Steve Byrne. Document Object Model Level2 Core. W3C Recommendation, 2000.

[3] Yannis Papakonstantinou, Serge Abiteboul, and Hector Garcia-Molina. Object exchange across heterogeneous information source. In *IEEE International Conference on Data Engineering*, 1995: 251-260.

[4] Alin Deutsch, Mary F. Fernandez, Daniela Florescu, Alon Y. Levy, and Dan Suciu. A Query language for XML. *Computer Networks* 31(11-16): 1155-1169, 1999.

[5] Peter Buneman, Mary F. Fernandez, and Dan Suciu. UnQL: a query language and algebra for semistructured data based on structural recursion. *VLDB Journal: Very Large Data Bases*, 9(1):76-110, May 2000.

[6] Serge Abiteboul, Dallan Quass, Jason McHugh, Jennifer Widom, and Janet Wiener. The lorel query language for semistructured data. *International Journal on Digital Libraries* 1(1):68-88, 1997.

[7] James Clark and Steve DeRose. XML Path Language (XPath) 1.0. W3C Recommendation, 1999.

[8] Don Chamberlin, Daniela Florescu, Jonathan Robie, Jerome Simeon, and Mugur Stefanescu. XQuery: A Query Language for XML. W3C Working Draft, February 2001.

[9] Roy Goldman and Jennifer Widom. DataGuides: enabling query formulation and optimization in semistructured databases. In *Proceedings of the Conference on Very Large Data Bases*, 1997: 436-445.

[10] Peter Buneman, Susan Davidson, Gerd Hillebrand, and Dan Suciu. A query language and optimization techniques for unstructured data. In *Proceedings of the ACM SIGMOD International Conference on the Management of Data*, 1996: 505-516.

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

[12] Tova Milo and Dan Suciu. Index structures for path expressions. In *Proceedings of the International Conference on Database Theory*, 1999: 277-295.

[13] Dan Suciu. Semistructured data and XML. In *Proceedings of International Conference on Foundations of Data Organization* 51(12):1050-1052, 1998.

[14] Mary F. Fernandez and Dan Suciu. Optimizing regular path expressions using graph schemas. In *IEEE International Conference on Data Engineering*, 1998:14-23.

[15] B. Cooper, N. Sample, M. J. Franlin, G. R. Hjaltason, and M. Shadmon. A fast index for semistructured data. In *Proceedings of the Conference on Very Large Data Bases*, 2001:341-350.

[16] Chun Zhang, Jeffrey Naughton, David DeWitt, Qiong Luo, and Guy Lohman. On Supporting Containment Queries in Relational Database Management Systems. In *Proceedings of the ACM SIGMOD International Conference on the Management of Data*, 2001:425-436.

[17] Quanzhong Li and Bongki Moon. Indexing and Querying XML Data for Regular Path Expressions. In *Proceedings of the Conference on Very Large Data Bases*, 2001:361-370.

[18] Jongik Kim and Hyoung-Joo Kim. Efficient Processing of Regular Path Joins using PID. *Information and Software Technology*, 45 (5):241--251, April 2003.

[19] Chin-Wan Chung, Jun-Ki Min, and Kyuseok Shim. APEX: An Adaptive Path Index for XML Data. In *Proceedings of the ACM SIGMOD International Conference on the Management of Data*, 2002:121-132.

[20] Raghav Kaushik, Pradeep Shenoy, Philip Bohannon, and Ehud Gudes. Exploiting Local Similarity for Indexing Paths in Graph-Structured Data. In *IEEE International Conference on Data Engineering*, 2002:129-140.

[21] Sangwon Park and Hyoung-Joo Kim. SigDAQ: an enhanced XML query optimization technique. *Journal of System Software* 61(2):91-103, 2002.

[22] The Internet Movie Database Ltd. Internet movie database. http://www.imdb.com.

[23] Xmark: The xml benchmark project. http://monetdb.cwi.nl/xml/index.html.

김 종 익

1998년 한국과학기술원 전산학과 학사
2000년 한국과학기술원 전산학과 석사
2000년~현재 서울대학교 컴퓨터공학부 박사과정. 관심분야는 XML, 데이타베이스

김 형 주

정보과학회논문지 : 데이타베이스
제 31 권 제 2 호 참조