Journal of Korea Multimedia Society Vol. 10, No. 7, July 2007(pp. 807-816)

RDBMS를 이용하여 XML 문서 관리를 위한 경로 저장과 숫자 매칭 기법

봉하익[†], 황병연[††]

요 약

1996년 W3C에서 XML을 제안한 이래, 다량의 XML(eXtensible Markup Language) 문서들이 인터넷에 확산되고 있다. 이런 이유로, XML과 관련된 연구의 필요성이 증가하고 있는 실정이다. 특히, XML 문서들을 저장, 검색, 그리고 관리하기 위한 XML 관리 시스템에 대한 연구가 활발히 진행되고 있다. 이런 연구들 중에서 XRel은 XML 문서 관리를 위한 대표적인 연구로써 인정되고 있으며, 비교 대상의 연구로서 사용되고 있다. 본 논문에서는 관계형 데이터베이스 시스템을 기반으로 한 XML문서에 대한 관리 기법을 제시한다. 이는 XRel처럼 모든 가능 경로를 저장하는 것이 아니라, 노드의 텍스트 값이나 속성 값이 존재하는 경로만을 저장하는 방식이다. 또, 노드 표현에 따라 고유 노드명 식별자(Node Expression Identifier)를 부여함으로써 부여된 노드 식별자를 매칭하는 숫자 매칭(Number Matching)기법을 제안한다. 마지막으로 제안 방식의 효율성을 입증하기 위해, 기존 방법과 XPath 질의에 대한 처리 성능을 비교함으로써 제안한 방법의 효율성을 제시한다.

A Path Storing and Number Matching Method for Management of XML Documents using RDBMS

Ha-Ik Vong[†], Byung-Yeon Hwang[††]

ABSTRACT

Since W3C proposed XML in 1996, XML documents have been widely spreaded in many internet documents. Because of this, needs for research related with XML is increasing. Especially, it is being well performed to study XML management system for storage, retrieval, and management with XML Documents. Among these studies, XRel is a representative study for XML management and has been become a comparative study. In this study, we suggest XML documents management system based on Relational DataBase Management System. This system is stored not all possible path expressions such as XRel, but filtered path expression which has text value or attribute value. And by giving each node Node Expression Identifier, we try to match given Node Expression Identifier. Finally, to prove efficiency of the suggested technique, this paper shows the result of experiment that compares XPath query processing performance between suggested study and existing technique, XRel.

Key words: XML Document(XML 문서), Path Storing(경로 저장), Number Matching(숫자 매칭), RDBMS(관계형 데이터베이스)

※ 교신저자(Corresponding Author) : 황병연, 주소 : 경기도 부천시 원미구 역곡2동 산 43-1(420-743), 전화 : 02)2164-4363, FAX : 02)2164-4777,
E-mail : byhwang@catholic.ac.kr
접수일 : 2007년 1월 29일, 완료일 : 2007년 5월 8일

[†] 가톨릭대학교 컴퓨터공학과 석사과정 재학
(E-mail : drvong@catholic.ac.kr)
[††] 종신회원, 가톨릭대학교 컴퓨터정보공학부 교수
※ 본 연구는 2007년도 가톨릭대학교 전공특성화 사업비 지원으로 이루어졌음.

1. 서 론

1996년 W3C(World Wide Web Consortium)에서 XML(eXtensible Markup Language)[1]이 제안된 이후 그 사용이 증가하였으며, 이로 인해 XML 데이터의 양도 빠른 속도로 늘어나고 있다. 이를 뒷받침해주기 위한 효율적인 접근 방법의 필요성이 커지면서 여러 연구가 발표되고 진행되어 왔다. 특히, 문서상의 트리 구조가 복잡해지면서도 깊이(depth)가 깊은 대용량의 XML 문서에 대한 효율적인 저장, 검색, 관리를 위한 XML 문서 관리 시스템의 연구가 활발히 진행되고 있다. 예를 들어, 최근 많이 사용되고 있는 블로그는 인터넷 상에서 문서를 교류할 때, XML 형태로 정의되어 있는 RSS(RDF Site Summary 또는 Rich Site Summary) 방식을 통하여 처리한다. 또한, 마이크로소프트사의 오피스 프로그램들(엑셀, 파워포인트, 워드 등)은 '오피스 2003'부터 '오픈 XML'을 사용한다. 이는 사용자가 만든 워드파일 및 엑셀 파일등을 XML을 이용하는 개발 프로그램 및 웹 프로그램에서의 분석 및 빠른 활용이 가능하기 때문이다. 앞의 사례뿐만 아니라, 비디오 정보를 XML 문서로 표준화하여 관리하기 위한 연구 등 많은 분야에서 그 활용이 높아지고 있다.

본 논문에서는 관계형 데이터베이스 시스템을 기반으로 한 대용량 XML 문서에 대한 효율적인 저장 및 질의 처리 기법으로, 노드의 텍스트 값이나 속성 값이 없는 경로를 저장하지 않음으로써 사용자가 요구하는 경로만 저장하는 방식을 제안한다. 또한 기존의 데이터베이스 저장 또는 검색 스키마에서 사용하던 경로에 대한 문자열 매칭(String Matching)기법[2,3]을 숫자 매칭(Number Matching)기법으로 변환한다. 이 숫자 매칭 기법은 노드의 지리적 위치 또는 상하 및 위치에 의존하여 고유 번호를 붙이는 Ordered Encoding 기법[4]이 아닌, 노드명칭에 따라 고유 번호를 부여하는 노드 식별자 개념을 적용하였다.

제안하는 두 가지 방법은 문자열 경로가 저장되는 테이블 저장 공간의 축소와 매칭 시간의 절약을 통해 질의 처리 성능을 향상시키는 효과를 보인다.

본 논문의 구성은 다음과 같다. 2장에서는 관련 연구에 대해 소개하고, 3장에서는 본 논문에서 제안하는 XML문서의 데이터 모델 및 저장 스키마에 대해서 설명한다. 4장에서는 성능 분석에 대해 서술하며, 5장에서는 결론 및 향후 연구과제에 대해 기술한다.

2. 관련 연구

XML 문서의 중요한 특징 중의 하나는 논리적인 구조에 기반한 연산을 수행할 수 있다는 점이다. 그러므로 XML 문서를 관리하는 데이터베이스는 논리적인 구조와 내용들의 질의에 대한 정확한 결과를 지원해야 한다. 그런 데이터베이스로부터 XML 문서들을 검색하기 위해서 XML 질의는 데이터베이스 질의로 (일반적으로 SQL-Structured Query Language) 변환되어져야 한다.

XML문서를 위해 데이터베이스 스키마를 디자인하기 위한 접근법으로는 구조 매핑 방식(Structure-mapping approach)과 모델 매핑 방식(Model-mapping approach) 2가지가 있다[2]. 구조 매핑 방식의 기본적인 디자인 방법은 하나의 릴레이션이나 클래스가 XML문서상에서 각각의 요소 유형으로 만들어지며, 각각의 XML 문서 구조나 DTD의 논리적인 구조를 통해 데이터베이스 스키마가 정의되어진다.

모델 매핑 방식은 고정된 데이터베이스 스키마가 DTD에 관한 정보 없이 XML 문서 모델의 구조를 나타낸다. 이 접근법으로 고정된 데이터베이스 스키마는 모든 XML 문서들의 구조를 저장하기 위하여 사용되어진다. 그 예로는 XML 문서 트리 내 에지(edge)가 관계형 튜플로서 저장되어지는 에지 접근법(edge approach)[5]과 경로와 영역(region)의 결합이라는 방식으로 XML 트리 구조를 표현한 XRel[2], 경로와 노드식별자로 표현하는 XParent[3], 관계형 테이블을 이용한 스키마 레벨 방법에 기반하면서 역인덱스 기술을 이용하는 방법[6-8]등이 있다. 또한 노드에 순서화된 숫자를 부여하는 방식을 사용하여 노드간의 관계를 파악하여 정의하는 Ordered Encoding 기법[4]이 있다.

2.1 XRel

XRel은 XML 문서의 트리 구조 내에서 인스턴스 내의 루트노드를 제외한 루트로부터 각 노드까지의 모든 경로들을 열거했으며, 관계형 속성들내의 경로 표현들 그 자체를 저장시켰다. 또한 모든 가능한 경로 표현들이 하나의 문자열로써 데이터베이스에 저

장되기 때문에 문자열 매칭(String Matching)이라는 방식으로 처리할 수 있다. XRel에서 XML 데이터 모델을 저장하기 위한 데이터베이스 스키마는 다음과 같다.

Element table(docID, pathID, start, end, index, reindex)
Attribute table(docID, pathID, start, end, value)
Text table(docID, pathID, start, end, value)
Path table(pathID, pathExp)
Document table(docID, docExp)

각각은 엘리먼트 정보, 속성 정보, 자료 값 정보, 경로 표현식, 그리고 문서 정보를 저장하고 있다. 여기서 ID와 Exp는 각각의 고유 식별자와 표현명을 나타내며, value는 해당 자료 값을 나타낸다. start와 end는 포함관계를 통하여 XML 문서 내 해당 영역을 나타낸다. index와 reindex는 같은 경로를 갖는 노드들 사이의 순서 관계를 preorder 방식으로 나타내기 위하여 사용된다[2]. 그러나 XRel은 문자열로 된 경로 표현식을 데이터베이스에 저장시키면서 자료 값을 갖지 않는 경로까지 저장시킴으로써 실제 사용자가 검색 시에 사용하지 않는 경로까지 저장시킨다. 이에 제안 방식은 자료 값을 갖는 경로만 저장시키며, 문자열의 경로를 숫자 경로로 저장시키는 방식을 제안한다.

2.2 XParent

XParent는 XRel의 문자열 매칭을 기본 골격으로 하되 부모-자식 관계 및 조상-후손 관계 개념을 도입하여 질의 처리 기능을 개선시킨 질의 처리 방식이다. (부모-자식 관계란 'A-B-C-D-E'의 레벨 순서를 갖는 경로를 가정했을 때, 레벨 3인 'C'노드보다 한 단계 앞 레벨인 'B'노드(레벨 2)가 'C'노드의 부모 노드가 되며, 한 단계 다음 레벨인 'D'노드(레벨 4)가 'C'노드의 자식노드가 되는 관계를 말하는 것이다. 이어서 나오게 될 조상-후손 관계에서 'C'노드의 조상은 'C'노드보다 전 레벨(레벨 1,2)의 모든 노드(노드 A,B)를 나타내며, 'C'노드의 후손은 'C'노드의 다음에 나오는 레벨(레벨 4,5)의 모든 노드(노드 D,E)를 나타낸다.) XParent에서 XML 데이터 모델을 저장하기 위한 데이터베이스 스키마는 다음과 같다.

LabelPath table(ID, Len, Path)
DataPath table(Pid, Cid)
Element table(PathID, Ordinal, Did)
Data table(pathID, Did, Ordinal, Value)
Ancestor table(Did, Ancestor, Level)

LabelPath 테이블은 XRel에서 *Path* 테이블과 같이 경로 표현식을 저장하며, Len은 해당 경로의 길이를 나타낸 것이다. *DataPath* 테이블에는 한 쌍의 노드 식별자를 저장하는데 Pid는 부모 노드 ID이며, Cid는 자식 노드 ID이다. 즉, 모든 노드들의 부모-자식 관계를 표현하여 하나의 테이블 내에 저장하는 것이다. *Element* 테이블과 *Data* 테이블은 엘리먼트 정보와 자료 값에 대한 정보를 나타낸 것이다. 여기서 PathID는 *Path* 테이블의 참조키이며, Ordinal은 형제 노드들 간의 순서를 나타낸 것이다. Did는 노드 식별자이며, Value는 자료 값을 나타내는 텍스트 정보를 나타낸다. *Ancestor* 테이블은 조상-후손 관계를 나타낸 것이다. 즉 Did는 노드 식별자이며, Ancestor는 해당 노드의 모든 조상 노드를 나타낸 것이다. 여기서 Level에는 트리 구조 내 깊이를 저장함으로써 조상 노드들 간의 깊이를 구분하였다. 이는 *Ancestor* 테이블을 통하여 엘리먼트 간의 조상-후손 관계를 빨리 결정함으로써 질의 처리 성능을 개선시켰지만, 너무 많은 저장 공간을 필요로 하며, 갱신 비용이 너무 많이 드는 단점이 있다[3].

3. 제안하는 데이터 모델 및 저장 스키마

XML 문서는 트리로 볼 수 있으며, 단말 노드는 자료 값 즉, 텍스트에 해당되고 내부노드는 XML 엘리먼트에 해당된다. 그러나 일반적인 XML 문서에서는 단말 노드만이 자료 값을 갖는 형태를 벗어나 내부 노드도 자료 값을 갖는 문서가 존재하므로 내부 노드 역시 자료 값을 가진 문서를 바탕으로 한다. XML 질의어로는 Lorel[9], XML-QL[10], XQuery [11], XPath[12] 등의 다양한 방식이 제안되었으며, 제안하는 방식에서는 XPath(XML Path Language)를 질의 모델로 사용한다. 또한 XML 트리에 대한 분해 및 저장 방법은 경로를 저장하는 방식을 사용하며, 그림 1은 RDBMS에 기반하여 본 연구에서 제안한 관리 시스템 구조를 나타낸다.

RDBMS와 더불어 사용되는 주요 구조는 XML 데

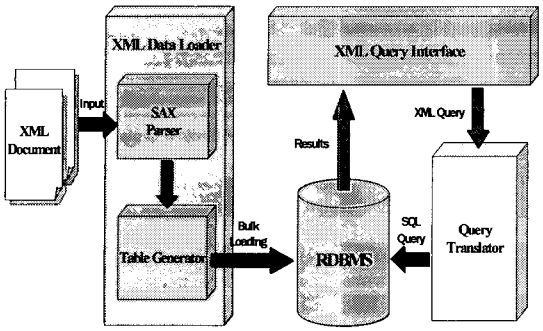

그림 1. XML 관리 시스템의 기본 구조

이터 로더, XML 쿼리 인터페이스, XML 쿼리 변환기이다. XML 데이터 로더로 문서가 입력되면, 해당 문서를 SAX Parser를 통해 파싱(parsing)하고 데이터 테이블을 생성시킨다. 문서가 관계형 데이터베이스 시스템에 입력되면 사용자는 XML 쿼리 인터페이스를 통해 XML 문서 구조를 보거나, 질의를 하게 된다. 사용자가 내린 질의 명령은 쿼리 변환기가 사용자의 XML 쿼리를 SQL 쿼리로 변환시켜주면서 관계형 데이터베이스 시스템에서 처리하게 된다. 이는 다시 XML 쿼리 인터페이스를 통해 확인할 수 있다.

3.1 XML 문서의 데이터 모델

본 논문에서는 XML 문서를 표현하기 위하여, 각 노드의 순서 및 위치가 아닌 명칭에 따라 번호를 부여하는 데이터 모델을 제시한다. 그림 2는 제안 방식을 바탕으로 한 XML 문서의 한 예이며, 이 방식은 노드명이 반복 사용됨에 따라 번호 역시 중복 사용됨을 알 수 있다. 이 때 같은 노드명(번호)을 가짐과 동시에 노드간의 순서 역시 같을 경우, 중복되는 동일한 경로 표현마다 각각의 고유 인덱스 값을 부여하여 동일 경로를 구분한다.

제안하는 데이터 모델은 Tatarinov[4]가 제안한 모든 노드마다 고유한 순서를 부여하는 순서화된(ordered) XML 데이터 모델과 비교할 수 있다. 이 Ordered Encoding 방식은 부모-자식 관계 및 조상-후손 관계 그리고 형제관계를 고려하여 순서를 부여한다. 그러나 제안 방식은 노드간의 순서를 고려하지 않았기 때문에 갱신(입력) 작업이 발생했을 때 노드명에 새로운 번호를 부여하게 된다. 만약 노드명이 기존의 것과 동일하다면 기존 번호를 부여함으로써, 새로 입력된 노드에 다른 번호를 다시 부여하지는

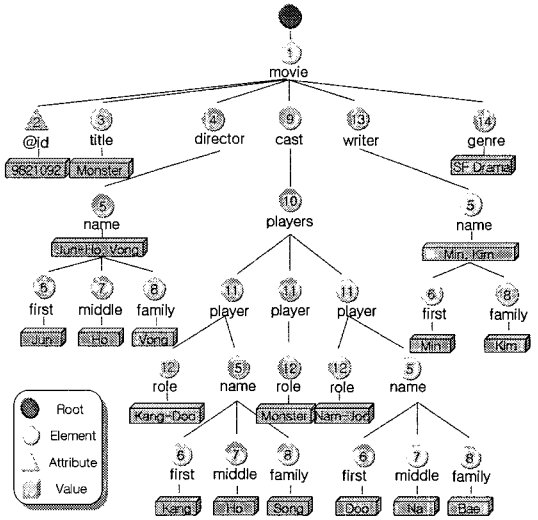

그림 2. XML 트리 모형의 예

않는다. 이는 데이터 변경 시 노드 간의 순서를 재정리해야하는 문제는 발생하지만, 노드에 해당하는 식별자에 관한 수정은 필요하지 않는 것을 의미한다. 즉, 데이터 변경 시 노드 간의 순서 재정립에 드는 비용이 한계점으로 작용한다. 데이터가 갱신되면서 발행하는 노드의 순서 재정리에 관한 문제를 해결하기 위해서 많은 연구들이 제시되고 있다. 본 논문에서는 Ordered Encoding 방식을 노드 순서를 규명하는 기본 모델로 채택한다. 제안방식이 노드간의 순서 규명에 초점을 둔 것은 아니므로 Ordered Encoding 방식과는 다른 의도라 할 수 있다. 그러나 순서 관계 질의에 대한 보완적인 방법으로 Ordered Encoding 방식을 적용하여 처리한다.

3.2 데이터베이스 저장 스키마

본 논문에서는 경로를 저장함에 있어서 모든 가능 경로를 저장하는 것이 아니라, 자료 값을 갖는 경로만 저장하는 방식을 사용한다. 이는 XML 문서를 검색하는 사용자가 실제로 텍스트 값이나 속성 값이 없는 사항에 대해서는 검색을 하지 않거나, 검색을 하더라도 null 값을 결과로 보이기 때문에 불필요한 저장 공간의 낭비를 막기 위함이 그 목적이다.

또한 데이터 타입의 크기에 따라서 메모리의 크기와 형태가 결정되는데, 기존 방식이 다수의 문자열을 경로 테이블에 저장시켰다면, 제안 방법은 숫자를 경로 테이블에 저장시킨다. 이는 경로 테이블에 할당되

는 메모리의 양을 줄이며 검색 시간을 절약하는 효과를 나타낸다. 이를 바탕으로 한 노드 타입별 관계형 저장 스키마는 다음과 같다.

Document table(DocID, DocExp)
Node table(NodeID, NodeExp)
Path table(PathID, NodeOrd)
Value table(ValueID, DocID, PathID, PathIndex, V_Text)
Attribute table(AttrID, DocID, PathID, PathIndex, A_Text)

여기서 *Document* 테이블은 문서 정보, *Node* 테이블은 노드 정보, *Path* 테이블은 노드 식별자로 표현된 경로 정보, *Value* 와 *Attribute* 테이블은 자료 값과 속성 값을 나타내는 테이블이다. DocID, NodeID, PathID, ValueID, AttrID는 문서, 노드, 경로, 자료 값, 그리고 속성 값의 고유 식별자이며 DocExp와 NodeExp는 해당 문서명과 노드명의 표현형이다. NodeOrd는 숫자로 표현된 경로 표현식, PathIndex는 동일 경로 구별자이며 V_Text와 A_Text는 자료 값과 속성 값이다. 이런 관계형 데이터베이스 저장 스키마를 바탕으로 한 그림 2의 데이터 모델에 해당하는 저장 테이블은 그림 3과 같다.

이때 그림 2의 데이터 모델이 갖고 있는 속성 값은 하나이기 때문에 *Value* 테이블과 *Attribute* 테이블을 구분하지 않고 *Value or Attribute* 테이블이라 임의로 명명하여 제시한다. 문서에 관한 정보 역시 문서가 하나라고 가정하고 예시로 보여준 것이기 때문에 *Document* 테이블을 따로 제시하지는 않았다.

기존 연구에서 사용한 문자열 매칭(String Matching)방식(예: movie/cast/ players/player/role)은 경로에 해당하는 노드명을 모두 나열하여 저장하고 매칭하는 방식을 사용했다. 그러나 제안 방식은 경로에 해당하는 노드 표현 식별자를 저장하여 매칭하는 숫자 매칭(Number Matching)방식(예: 1/9/10/11/12)을 채택하여 Path 테이블의 NodeOrd에 저장한다.

예로 들은 경로 표현이 문자열 매칭 방식으로 저장된다면 30Byte를 차지하게 되는 반면 숫자 매칭은 12Byte만 차지하게 된다. 그리고 이 경로에 해당하는 자료 값은 'Kang-Doo', 'Monster'와 'Nam-Joo'로써 서로 다른 세 개의 경로가 같은 표현형을 갖고 있는데, 이는 *Value* 테이블에 동일한 경로마다 인덱스를 부여한 PathIndex라고 하는 동일 경로 구별자를 통해 구별할 수 있다. 세부 질의 처리 과정은 다음의 질의 처리 예제를 통하여 알아본다.

3.3 질의 처리 예제와 알고리즘

그림 4는 앞에서 예로 들은 "movie/cast/players /play/role"에서 2번째 순서의 자료 값을 찾는 결과로 'Monster'를 추출하는 과정을 나타낸 것이다.

"movie/cast/players/play/role[2]"라는 질의가 들어오면, *Node* 테이블에서 각 노드에 해당하는 노드 식별자를 확인한다. 이 때, movie에 해당하는 노드 식별자인 '1', cast에 해당하는 노드 식별자인 '9'를 확인하는 방법을 계속하여 '1','9','10','11','12'의 노드를 추출한다. 이 노드들을 연결하여 "1/9/10/11/12"라는 질의 경로를 확인하고, 이 질의 경로를 *Path* 테이블에 미리 저장한 경로 표현식과 매칭시킨다. 이렇

Node

NodeID	NodeExp
1	movie
2	@id
3	title
4	director
5	name
6	first
7	middle
8	family
9	cast
10	players
11	player
12	role
13	writer
14	genre

Path

PathID	NodeOrd
1	1/2
2	1/3
3	1/4/5
4	1/4/5/6
5	1/4/5/7
6	1/4/5/8
7	1/9/10/11/12
8	1/9/10/11/5/6
9	1/9/10/11/5/7
10	1/9/10/11/5/8
11	1/13/5
12	1/13/5/6
13	1/13/5/8
14	1/14

Value or Attribute

ValueID	DocID	PathID	PathIndex	Value
1	1	1	1	9821092
2	1	2	1	Monster
3	1	3	1	Jun-Ho, Vong
4	1	4	1	Jun
5	1	5	1	Ho
6	1	6	1	Vong
7	1	7	1	Kang-Doo
8	1	8	1	Kang
9	1	9	1	Ho
10	1	10	1	Gong
11	1	7	2	Monster
12	1	7	3	Nam-Joo
13	1	8	2	Doo
14	1	9	2	Na
15	1	10	2	Bae
16	1	11	1	Min, Kim
17	1	12	1	Min
18	1	13	1	Kim
19	1	14	1	SF Drama

그림 3. 데이터베이스 저장 테이블

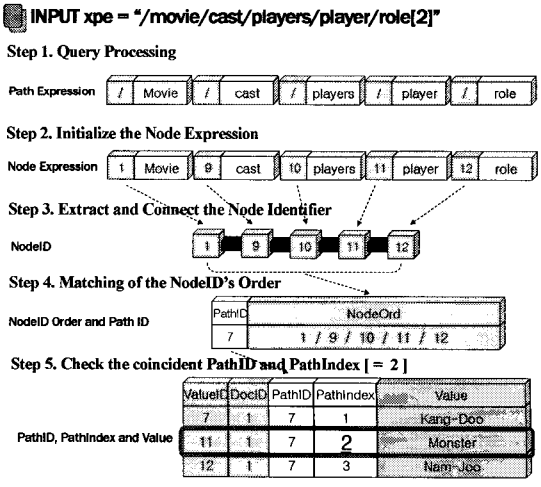

그림 4. 경로 질의 검색 과정

게 매칭된 PathID '7'을 참조키로 하는 *Value* 테이블의 PathID를 확인하며, 이에 매칭되는 *Value* 테이블의 행들은 그림 4의 step 5와 같다. 마지막 단계로서 요구하는 동일 경로간의 pre-order 순서를 나타낸 PathIndex를 바탕으로 동일 경로 중에서 2번째 순서인 'Monster'를 결과로 도출할 수 있다. 대용량의 XML 문서일수록 노드 테이블에 저장되는 노드의 양은 많아질 것이다. 이것은 사용자의 문자열 질의를 숫자형 질의로 바꿀 때 노드 테이블에서 비교해야할 데이터의 양이 많아지는 것을 의미한다. 즉, 그림 4의 1번째 단계에서 2번째 단계로 넘어가는 과정의 시간이 늘어나는 것을 말한다. 그러나 XRel도 XML 문서의 깊이나 경로가 복잡해질수록, 저장되는 문자열 경로 역시 늘어나게 된다. 관리되는 문서가 커질수록 제안 방법보다 XRel이 저장해야할 경로의 길이가 길어지게 되며, 중복되는 노드가 많을수록 제안 방법이 XRel에 비하여 효율적임을 알 수 있다.

그러나 XML 문서 내에서 노드명이 반복 사용되는 것이 적다면, 노드 테이블 내에 저장되는 노드명의 수가 많아질 것이다. 이로 인해, 노드 테이블이 참조되는 회수가 많아질 것이고 이는 성능 저하를 야기할 것이다. 그러나 대부분의 XML 문서들은 노드명을 반복 사용함으로써, 문서를 일목요연하게 확인할 수 있도록 관리하고 있다. 이는 [8]의 실험에서 알 수 있다. 이 실험은 웹 상에서 임의로 1000개의 XML 문서를 수집하여, 약 70%의 XML 문서가 동일한 노드명을 여러 레벨에서 반복 사용함을 확인하고 있다. 이에 따라, 본 논문의 실험에서도 노드명이 반복 사용되는 문서를 대상으로 성능 평가를 실시하였다.

제안 방법이 수행하는 질의 처리 알고리즘은 그림 5와 같다.

질의어에서 문자열을 노드명 식별자로 전환하고 *Path* 테이블에 저장된 노드명 식별자 표현과 일치하는 PathID를 찾는다[그림 6]. 이어서 *Value* 또는 *Attribute* 테이블에 저장된 해당 노드를 찾는다[그림 7].

```
Procedure SearchNodes(PEQ : Path Expression of Query)
returns matched : set of nodes
begin
   //replace PEQ by NodeIDs' Expression
   begin
        replace PEQ by NodeID that is matched in Node Table
        if(PEQ has "//" at the beginning) then
            replace "//" by "%/";
        elesif(PEQ has "//" in the middle or its end) then
            replace "//" by "/%/";
        else store replced ones in replacedNodeOrdExp;
   end
   // find PathIDs that are matched with NodeOrd stored in Path Table
   initialize searchedPathIDset;
   begin
        if( replaced ) then searchPathID;
        store PathIDs in searchedPathIDset;
   end
  // find nodes with the equivalent PathID, Index and Value in VALUE Table
  initialize nodeSet;
  for(i=0; searchedPathIDset[i] != NULL; i++) do
  begin
       if (PEQ has Index) then
          nodeSet += findMatchedValue_IX(matchedPathIDset[i], Index);
       elseif (PEQ has Text value) then
          nodeSet += findMatchedValue_TEXT(matchedPathIDset[i], txt_value);
       elseif (PEQ has Attribute value) then
          nodeSet += findMatchedValue_Attribute(matchedPathIDset[i], attr_value);
       else nodeSet += findMatchedValue_EQ(matchedPathIDset[i]);
  end
  return nodeSet;
end
```

그림 5. 질의 처리 알고리즘

```
Procedure searchPathID(replacedNodeOrdExp)
returns matched(PathID) in Path Table
begin
        SELECT PathID
        FROM Path p1
        WHERE p1.NodeOrd LIKE ' "+ replacedNodeOrdExp +"'
end
```

그림 6. *Path* 테이블에서 해당 ID를 찾는 알고리즘

```
Procedure findMatchedValue_EQ(pathID)
returns matched(DocID, ValueID)s in Value Table
begin
      SELECT DocID, ValueID
      FROM Value v1
      WHERE v1.PathID = pathID
end
Procedure findMatchedValue_IX(pathID, Index)
returns matched(DocID, ValueID)s in Value Table
begin
      SELECT DocID, ValueID
      FROM Value v1
      WHERE v1.PathID = pathID AND v1.PathIndex = Index
end
Procedure findMatchedValue_TEXT(pathID, text_Value)
returns matched(DocID, ValueID)s in Value Table
begin
      SELECT DocID, ValueID
      FROM Value v1
      WHERE v1.PathID = pathID AND v1.value LIKE '%"text_Value"%';
end

Procedure findMatchedValue_Attribute(pathID, attr_Value)
returns matched(DocID, ValueID)s in Value Table
begin
      SELECT DocID, ValueID
      FROM Value v1
      WHERE v1.PathID = pathID AND v1.value LIKE '%"attr_Value"%';
end
```

그림 7. *Value* 테이블에서 해당 노드를 찾는 알고리즘

4. 실험 및 성능 평가

본 실험에서 사용한 운영체제는 Window XP, 하드웨어는 Pentium 4-1.72GHz, 1GB RAM이며, 데이터베이스 시스템으로 MS SQL-Server 2000을 사용하였다. 비교 평가를 위한 검색 기법으로는 XRel[2]을 사용하였으며, 실험용 데이터로는 셰익스피어 전집을 XML 문서로 변환하여 저장하고 있는 'Bosak Shakespeare collection'[13]을 사용하였다. 이 XML 문서의 DTD 구조는 그림 8과 같다.

그림 9는 실험 시 사용한 질의 처리기이다. 질의를 입력하고 원하는 방식의 실험을 선택하여 실행한 결과의 평균을 구한 것이다. 질의 처리기를 통하여 그림 10의 질의를 처리하였으며 그림 9는 6번 질의를 처리한 결과를 실제 예로 보여주고 있다.

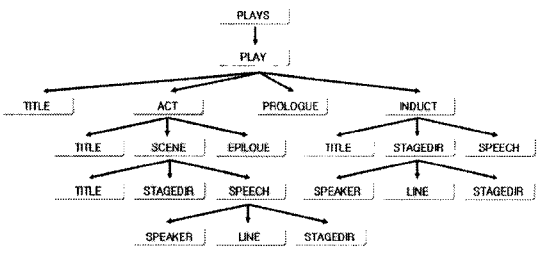

그림 8. XML 데이터의 DTD 그래프

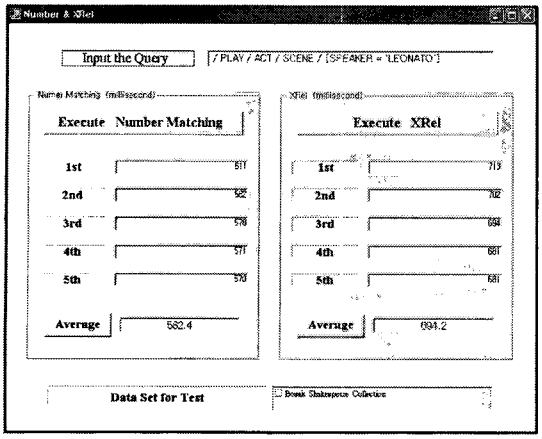

그림 9. 질의 처리기

QR	Query Expression
QR1	/PLAY/FM/P
QR2	/PLAY/PERSONAE/PGROUP/GRPDESCR
QR3	/PLAY/PROLOGUE/SPEECH/LINE
QR4	/PLAY/INDUCT/SCENE/SPEECH/LINE
QR5	/PLAY/PROLOGUE/TITLE[4]
QR6	/PLAY/ACT/SCENE/[SPEAKER='LEONATO']
QR7	/PLAY//SUBHEAD
QR8	//INDUCT/SCENE//LINE
QR9	//EPILOGUE//LINE
QR10	//PLAYSUBT

그림 10. 경로 질의

비교 분석을 위한 실험을 위하여 다음 그림 10의 10가지 질의를 선택하였다.

다양한 경우의 질의 처리 분석을 위하여 여러 가능성의 질의를 해 보았다. 그림 10에 나타난 질의문은 QR1-4번이 부모-자식 관계에 대한 질의, QR5-6번이 특정 인덱스 및 키워드를 갖는 질의 그리고 QR7-10번이 조상-후손 관계에 관한 질의이다. 여기서 부모-자식 관계를 나타내는 구분자인 "/"가 조상-후손 관계를 나타내는 구분자인 "//"보다 많은 이유는 "/" 구분자의 사용이 경로 질의 패턴의 70%이상을 차지한다는 사실을 밝힌 N. Zhang[14]의 연구 결과에 의거해서이다. 그림 10에서 제시하고 있는 질의에 대해, 질의 처리기를 통하여 5회 반복 시행한 후 얻은 결과는 그림 11과 같다.

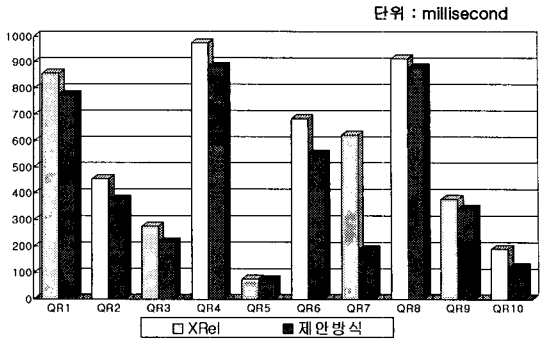

그림 11. 경로 질의 처리 시간

실험 결과는 그림 11에서 보이는 것처럼 제안한 방식이 기존 방식인 XRel보다 모든 측면에서 우수하게 나타난 것을 알 수 있다. 이는 그림 12에서 알 수 있듯이 제안 방식이 기존 방식보다 저장되는 경로 수가 적고, 문자열로 저장되는 것보다 고유 번호로 저장되는 것이 경로 테이블의 크기를 줄이는 효과를 나타내기 때문이다. 물론 기존 연구는 검색 시 노드 정보를 불러올 노드 테이블을 사용할 필요가 없이 경로 테이블만 검색하면 되지만, 제안 방식은 노드 테이블도 사용해야 한다. 그러나 앞에서 밝힌 제안 방식의 장점들이 이 점을 보완해주고 있어 여러 질의에서 성능 개선을 나타냈다.

	저장되는 경로 수	Path Table의 문자열 크기	Node Table의 문자열 크기
XRel	57개	1308Byte	없음
제안 방식	40개	439Byte	127Byte

그림 12. 경로 검색시 사용되는 테이블 정보

5. 결론 및 향후 연구

본 논문에서는 XML 문서의 효율적인 질의처리를 위한 경로 기반 저장 기법의 보완점을 제시하였다. 제안 방법은 i) 숫자 매칭을 통한 검색 시간의 단축, ii) 사용자가 실제로 필요로 하는 정보만 저장하는 방법을 통하여 검색 시간이 단축됨을 알 수 있다. 이는 기존 방식처럼 XML 문서가 갖는 모든 경로를 저장하는 것이 아니라 자료 값이나 속성 값을 갖는 경로만 저장시켜 저장 공간을 축소시키고, 경로 저장 및 검색 시 문자열이 아닌 숫자 매칭을 사용함으로써

검색해야할 저장 공간 역시 축소시켰기 때문이다.

또한 이에 대한 기존 연구와의 비교 실험을 통해 우수한 검색 성능을 보임을 입증하였다. 본 논문에서 실험을 위하여 사용한 문서보다 더 많은 양의 XML 문서가 입력되었을 경우에는 더 많은 검색 시간의 축소와 저장 공간의 절약을 예상할 수 있다. 대부분의 XML 문서가 그렇듯이 단말 노드에만 자료 값이나 속성 값을 갖고 있거나, 문서 내의 경로 깊이가 크다고 가정하면 제안하는 방식의 효과가 더 커질 것은 자명한 일이다. 이는 중간 노드에 자료 값이 없으면 루트로부터 각 중간 노드까지의 경로를 저장할 필요 없이, 루트 노드로부터 단말 노드까지의 경로 하나만 저장하면 되므로 경로 테이블 내 행의 개수가 줄어드는 것을 의미한다. 또한 경로 깊이가 크면 문자열이 숫자로 변환되는 노드가 많아지는 것을 의미하며, 이로 인해 경로 테이블의 행에 있는 경로 표현식 자체가 축소되며 검색해야 할 데이터의 양도 줄어들게 되는 것이다.

향후에 데이터베이스 저장 스키마나 데이터 모델에 의거하여 질의 처리 방식을 연구하는 것이 아니라, 실제 사용자의 질의 유형 패턴과 XML 문서의 실제 저장 유형에 의거하여 질의를 처리하는 방식을 연구할 것이다. 또한 조상-후손 및 형제 관계 규명을 효과적으로 처리할 수 있는 방법과 좀 더 효율적인 경로 축약 및 검색 기법에 대하여 연구할 계획이다.

참 고 문 헌

[1] "Extensible Markup Language(XML) 1.1," *W3C Candidate Recommendation*, 2002.

[2] M. Yoshikawa and T. Amagasa, "XRel: A Path-Based Approach to Storage and Retrieval of XML Documents using Relational Databases," *ACM Transactions on Internet Technology*, Vol. 1, No. 1, pp. 110-141, 2001.

[3] H. Jiang, H. Lu, W. Wang, and J. X. Yu, "Path Materialization Revisited: An Efficient Storage Model for XML Data," *In Proc. of the 13th Australasian Database Conference (ADC)*, Melbourne, Australia, pp. 85-94, 2002.

[4] I. Tatarinov, S. D. Viglas, K. Beyer, J. Shanmugasundaram, E. Shekita, and C. Zhang, "Storing and Querying Ordered XML Using a Relational Database System," *In Proc. of ACM SIGMOD Conf.*, 2002.

[5] D. Florescu and D. Kossmann, "Storing and Querying XML Data Using an RDBMS," *IEEE Data Engineering Bulletin*, Vol. 22, No. 3, pp. 27-34, 1999.

[6] 이혜자, 정병수, 이영구, "대용량 XML데이터베이스에서 경로정보의 중복을 제거한 효율적인 질의 처리," 정보과학회 데이터베이스연구, 제21권, 제3호, pp. 15-32, 2005.

[7] 김영자, 김현주, 배종민, "구조 기반 검색을 위한 색인 구조에 대한 분석," 멀티미디어학회논문지, 제7권, 제5호, pp. 601-616, 2004.

[8] 이윤호, 최일환, 김종익, 김형주, "색인된 XML 문서에서 레벨 정보를 이용한 효과적인 구조 조인 기법," 정보과학회논문지, 제32권, 제6호, pp. 641-649, 2005.

[9] S. Abiteboul, D. Quass, J. McHugh, J. Widom, and J. L. Wiener, "The Lorel Query Language for Semistructured Data," *International Journal on Digital Libraries*, Vol. 1, No. 1, pp. 66-88, 1997.

[10] A. Deutsch, M. F. Fernandez, D. Florescu, A. Y. Levy, and D. Suciu, *XML-QL*, QL, 1997.

[11] XQuery 1.0: An XML Query Language W3C Working Draft, http://www.w3.org/TR/2006/PR-xquery-20061121/, 2006.

[12] W3C, "XML Path Language (XPath) 2.0," http://www.w3c.org/TR/2003/WD-xpath20-20031112/, Nov. 2003.

[13] "Bosak Shakespeare Collection," http://www.oasis-open.org/cover/bosakShakespeare200.html.

[14] N. Zhang, V. Kacholia, and M. T. Özsu, "A Succinct Physical Storage Scheme for Efficient Evaluation of Path Queries in XML," *In Proc. of ICDE, Boston*, MA, March 2004.

봉 하 익

2006년 가톨릭대학교 수학과 (이학사), 컴퓨터공학과 (공학사)
2006년~현재 가톨릭대학교 컴퓨터공학과 석사과정 재학
관심분야 : XML, 정보검색, 데이터베이스

황 병 연

1986년 서울대학교 컴퓨터공학과(공학사)
1989년 한국과학기술원 전산학과(공학석사)
1994년 한국과학기술원 전산학과(공학박사)
1994년~현재 가톨릭대학교 컴퓨터정보공학부 교수
1999년~2000년 University of Minnesota Visiting Scholar
관심분야 : XML 데이터베이스, 데이터마이닝, 지리정보시스템, 정보검색