실체뷰 캐쉬 기법을 이용한 XML 질의 처리 시스템의 구현

문 찬 호† · 박 정 기† · 강 현 철††

요 약

데이터베이스 기반의 웹 응용을 위한 캐싱 기법이 최근 많이 연구되고 있다. 자주 제기되는 질의의 결과를 캐쉬해두면 반복 질의를 위한 재사용은 물론 관련 질의의 처리에 이용될 수 있다. 웹 상에서 데이터 교환의 표준으로 XML이 등장한 이래 현재 웹 응용들은 네트워크 상의 원격 XML 소스로부터 데이터 검색을 수행하는 경우가 많아졌는데 이의 효율적인 지원을 위해 검색 결과를 캐쉬하는 것은 유용하다. 본 논문은 XML 질의를 관련 XML 캐쉬를 이용하여 처리하는 시스템의 구현 및 성능 평가에 관한 것이다. XML 질의로 XQuery, XPath, XQL 등과 같은 모든 XML 질의어의 핵심 요소인 경로 표현식을 대상으로 하였고, XML 캐쉬는 XML 실체뷰를 고려하였고, 캐쉬를 이용한 XML 질의 변환 알고리즘은 [13]에 제시된 것을 대상으로 하였다. [13]의 질의 변환 알고리즘을 지원하는 프로토타입 XML 저장 시스템이 관계 DBMS를 이용하여 구현되어 다양한 성능 실험에 이용되었다. 구현의 주요 이슈에 대하여 자세히 기술한다. 성능 실험 결과를 통해 캐쉬를 이용한 XML 질의 처리의 효율성을 확인하였고, 기존 연구와의 성능 비교도 기술하였다.

Implementation of XML Query Processing System Using the Materialized View Cache-Answerability

ChanHo Moon† · Jung Kee Park† · Hyunchul Kang††

ABSTRACT

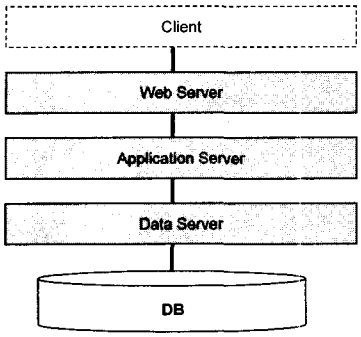
Recently, caching for the database-backed web applications has received much attention. The results of frequent queries could be cached for repeated reuse or for efficient processing of the relevant queries. Since the emergence of XML as a standard for data exchange on the web, today's web applications are to retrieve information from the remote XML sources across the network, and thus it is desirable to maintain the XML query results in the cache for the web applications. In this paper, we describe implementation of an XML query processing system that supports cache-answerability of XML queries, and evaluate its performance. XML path expression, which is one of the core features of XML query languages including XQuery, XPath, and XQL was considered as the XML query. Their result is maintained as an XML materialized view in the XML cache. The algorithms to rewrite the given XML path expression using its relevant materialized view proposed in [13] were implemented with RDBMS as XML store. The major issues of implementation are described in detail. The results of performance experiments conducted with the implemented system showed effectiveness of cache-answerability of XML queries. Comparison with previous research in terms of performance is also provided.

키워드 : XML, 경로 표현식(Path Expression), 질의 변환(Query Rewriting), 실체뷰(Materialized View), 데이터베이스 기반 웹 응용(Database-Backed Web Application)

1. 서 론

자주 제기되는 질의의 결과를 캐쉬해 두었다가 후속 질의에 사용하는 것은 중요한 질의 최적화 기법 중의 하나이다. 소스 데이터에 대한 변경을 반영하여 캐쉬를 갱신(refresh)하는 것이나 캐쉬 교체(replacement)와 같은 캐쉬 관리가 적절히 이루어진다고 할 때, 결과를 캐쉬해 둔 질의와 동일한 질의가 반복 제기되면 그 결과는 즉각 주어질 수 있고, 완전히 동일하지는 않지만 관련된 질의가 제기되면 해당 캐쉬에 대한 질의로 변환(rewrite)되어 캐쉬를 이용하여 처리될 수 있다. 이와 같은 캐쉬를 이용한 질의 변환 및 처리 기법은 관계 데이터베이스 시스템에서 많이 연구되었지만[1], 1990년대 후반부터 웹 환경에서 데이터베이스 기반의 웹 응용을 위한 질의 처리 기법을 위해 반구조적 데이터나 XML 데이터에 대해 많이 연구되고 있다 [2-7].

※ 이 논문은 2003년도 중앙대학교 학술연구비 지원에 의한 것임.
† 준 회 원 : 중앙대학교 대학원 컴퓨터공학과
†† 종신회원 : 중앙대학교 컴퓨터공학과 교수
논문접수 : 2003년 5월 30일, 심사완료 : 2003년 12월 11일

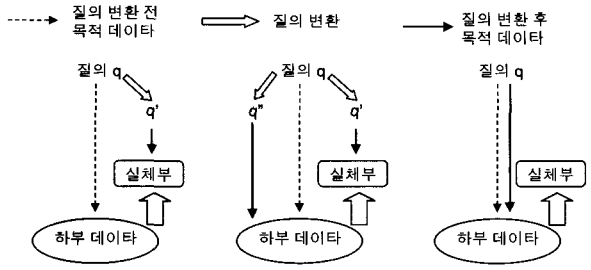


(그림 1) 데이터베이스 기반 웹 응용을 위한 다계층 구조

(그림 1)은 널리 사용되고 있는 데이터베이스 기반의 웹 응용(database-backed web application)을 지원하는 웹 서버, 응용 서버, 데이터 서버의 다계층(multi-tier) 구조를 나타낸 것이다. 이 구조에서 웹 서버는 웹 페이지에 포함된 질의를 미들웨어 층인 응용 서버를 통해 데이터 서버로 보내게 된다. 질의 결과는 데이터 서버에 캐쉬되어 그 소스와 함께 존재할 수도 있고 응용 서버에 캐쉬되어 있을 수도 있다.

XML이 웹에서 데이터 교환의 표준으로 사용되고 있으므로 XML 데이터 기반의 웹 응용을 위한 XML 질의 처리에서 캐쉬를 이용하는 기법의 연구는 아주 중요하다. 이 문제는 최근에 이르러 활발히 연구되기 시작했다[5-7]. 본 논문은 XML 캐쉬를 이용하여 주어진 XML 질의를 변환하여 처리하는 기법의 구현 및 성능 평가에 관한 것이다. 캐쉬를 이용하여 XML 질의를 처리하는 기능을 지원하는 프로토타입 XML 저장 시스템이 관계 DBMS를 이용하여 구현되어 다양한 성능 실험에 이용되었다. 관계 DBMS가 널리 사용되고 있는 점 때문에 XML 데이터를 관계 DBMS에 저장하고 검색하는 기법은 널리 연구되고 있는데[8-12], XML 데이터와 캐쉬가 모두 관계 DBMS에 저장되어 있을 때 캐쉬를 이용한 XML 질의 처리 기법 또한 중요하다 하겠다. 본 논문에서는 XML 질의로 XQuery, XPath, XQL 등과 같은 모든 XML 질의어의 핵심 요소인 경로 표현식(path expression)을 대상으로 하였고, XML 캐쉬는 XML 실체뷰를 고려하였고, 캐쉬를 이용한 XML 질의 변환 알고리즘은 [13]에 제시된 것을 대상으로 하였다.

(그림 2)는 XML 질의 처리에 실체뷰를 이용하는 두가지 경우와 그렇지 못한 경우를 나타낸 것이다. (그림 2)(a)는 주어진 질의의 결과를 모두 관련 실체뷰에서 얻을 수 있는 경우를 나타낸 것으로 질의 q가 실체뷰에 대한 질의 q′으로 변환되어 처리된다. (그림 2)(b)는 질의 q의 결과 일부는 실체뷰로부터, 나머지는 하부 XML 소스로부터 얻는 경우를 나타낸 것으로, q는 실체뷰에 대한 질의 q′과 하부 데이터에 대한 질의 q″로 변환되어 처리되고 q′과 q″의 결과는 통합되어 최종 결과를 얻는다. (그림 2)(c)는 질의 처리에 관련 실체뷰를 이용할 수 없거나 이용하지 않는 경우를 나타낸 것으로 질의 변환은 일어나지 않는다. 본 논문에서는 이들 질의 처리 유형을 각각 MV(Materialized View Only), BD+MV(Both Base Data and Materialized View), 그리고 BD(Base Data Only)라 부른다.



(a) MV 유형 (b) BD+MV 유형 (c) BD 유형

(그림 2) 실체뷰를 이용한 질의 처리의 유형

본 논문의 구성은 다음과 같다. 2장에서는 관련연구를 살펴본다. 3장에서는 관계 DBMS를 기반으로 수행된 실체뷰 캐쉬 기법을 이용한 XML 질의 처리 시스템의 구현에 대해 기술하고, 4장에서는 구현된 시스템을 이용한 다양한 실험 결과 및 기존 연구와의 성능 비교를 기술한다. 5장에서는 결론을 맺고 향후 연구 과제를 제시한다.

2. 관련 연구

뷰 또는 실체뷰를 이용하여 관련 질의를 변환하는 문제는 관계 데이터베이스에서 많이 연구되었고 [1], 90년대 후반부터는 반구조적 데이터에 대해서도 연구되기 시작했다[2-4]. 최근에 이르러서는 XML 질의어를 대상으로 연구되기 시작했다[5-7].

[5]에서는 XQuery의 일부 구문에 의해 정의되는 XML 실체뷰를 이용한 질의 처리를 지원하는 XCacher라는 시스템을 제시하였다. XCacher가 지원하는 XQuery 구문은 FOR-WHERE 절로 구성된 extract 부분과 RETURN 절로 구성된 construct 부분으로 구성되며, XCacher는 extract 부분을 캐쉬한다. 캐쉬는 그 자료구조로 [14]에서 제시된 MIT(modified incomplete tree)를 사용한다. XCacher 모듈은 응용 서버에 존재하며, 웹 서버가 데이터 서버로 전송하는 XML 질의를 중간에서 받아 캐쉬에 있는 내용을 활용할 수 있을 경우 캐쉬에 대한 질의와 데이터 서버의 XML 소스에 대한 질의(전자를 refinement 질의, 후자를 remainder 질의라 부른다)로 변환하고 각 질의로부터 계산된 부분 결과들을 통합하여 최종 질의 결과를 구한다.

[6]에서는 XQuery의 일부 구문에 대한 질의 포함 사상(containment mapping)과 변환을 수행하는 알고리즘을 제시하였다. 이 알고리즘은 질의 및 실체뷰를 정의하는 두개의 XQuery 구문이 주어지면 패턴 변수(pattern variable)를 이용하여 질의 포함 사상과 질의 변환을 수행한다. 제시된 알고리즘에 의해 실체뷰를 이용한 XML 질의 처리를 지원하는 ACE-XQ라는 프로토타입 시스템을 개발하고 성능

평가를 수행하였다.

[7]에서는 XPath로 정의된 실체뷰를 이용하여 관련된 XPath 질의를 처리하는 작업에 대한 형식 모델(formal model)을 제시하였다. 이 모델 하에 동일(equivalent), 약한 동일(weak equivalent), 그리고 부분 동일(partial equivalent)이라고 부르는 세 가지 형태의 질의 변환 유형을 정의하였다. 제안된 모델은 형식 모델이므로 이를 실제로 구현하기 위해서는 XML 데이터의 저장 및 검색 기능, XPath 부분 질의(subquery)의 순차적인 처리 기능, 그리고 캐쉬의 저장 구조 등이 제공되어야 한다.

3. 캐쉬를 이용한 XML 질의 처리 시스템의 구현

본 장에서는 캐쉬를 이용한 XML 질의 변환 및 처리를 지원하는 XML 질의 처리 시스템 프로토타입의 구현에 대하여 기술한다. 본 시스템은 XML 저장소로 이용되는 관계 DBMS로 Oracle 8i를 사용하였고, Java로 구현하였으며 Windows 2000 Server에서 구동한다. XML 파서로는 SAX 2.0을 지원하는 Xerces 2 Java Parser 2.4.0 Release를 사용하였다. 구현을 위해 고려되어야 할 주요 항목으로는 (1) 뷰의 하부 XML 소스 및 실체뷰 등을 저장하는 테이블 스키마, (2) 경로 표현식으로 표현된 XML 질의의 SQL로의 변환, 그리고 (3) SQL 결과 셋의 XML 태깅을 들 수 있다. 이들을 각각 설명하면 다음과 같다.

3.1 테이블 스키마

XML 저장소는 크게 하부 데이타 영역과 XML 실체뷰 영역으로 나누어진다. 하부 데이타 영역은 뷰의 소스 XML 문서들을 엘리먼트 단위로 분할하여 XML 문서의 구조 정보와 내용 정보를 각각 따로 저장한 Element_Info 테이블과 Element_Content 테이블, XML 문서의 기타 정보를 저장한 Doc_Info 테이블, 그리고 DTD 정보를 저장한 DTD 테이블로 구성된다. 하부 XML 문서를 내용 정보(Content 테이블)와 구조 정보(Info 테이블)로 나누어 따로 저장한 것은 [10]에서 제시된 Edge/Separate Value Table 기법에 바탕을 둔 것인데 경로 정보 등을 추가하여 확장한 것이다. XML 실체뷰는 XML 질의 결과인 XML 문서가 된다. 따라서 XML 실체뷰 영역 또한 XML 실체뷰들을 엘리먼트 단위로 분할하여 뷰의 구조 정보와 내용 정보를 각각 따로 저장한 View_Info 테이블과 View_Content 테이블, 뷰 정의를 저장하는 View_Definition 테이블, 그리고 DTD 테이블로 구성된다.

3.1.1 하부 데이타 영역의 테이블 스키마

(그림 3)은 서점(bookstore)에 대한 DTD와 해당 DTD 테이블의 구조 및 예를 나타낸 것이다. DTD 테이블은 DTDID, Doc_Type, Content 속성으로 구성된다. DTDID에는 DTD의 ID가, Doc_Type에는 DTD를 저장한 문서의 형식이, Content에는 DTD 문서의 내용이 저장된다. (그림 4)는 Doc_Info 테이블의 구조 및 예를 나타낸 것이다. DID에는 XML 문서의 ID, DTDID에는 해당 XML 문서가 참조하는 DTD의 ID, 그리고 DocName에는 XML 문서의 이름이 저장된다.

```
<!ELEMENT bookstore     (book)>
<!ELEMENT book          (title, author+, content, price)>
<!ELEMENT title         (#PCDATA)>
<!ELEMENT author        (first-name, last-name)>
<!ELEMENT first-name    (#PCDATA)>
<!ELEMENT last-name     (#PCDATA)>
<!ELEMENT content       (title, chapter)+>
<!ELEMENT chapter       (title, section)+>
<!ELEMENT section       (sentence+)>
<!ELEMENT sentence      (#PCDATA)*>
<!ELEMENT price         (#PCDATA)*>
```

(a) DTD

DTDID	Doc_Type	Conten
1	bookstore	<!ELEMENT bookstore (book)> <!ELEMENT book (title, ···>

(b) DTD 테이블

(그림 3) Bookstore DTD와 DTD 테이블의 구조 및 예

DID	DTDID	DocName
1	1	bookstore1.xml
2	1	bookstore2.xml

(그림 4) Doc_Info 테이블의 구조 및 예

본 논문에서의 하부 XML 문서들은 해당 DTD를 준수하는 유효(valid) XML 문서들이며, XML 질의는 단일 DTD를 준수하는 문서들 전체를 대상으로 제기된다. 각 문서는 엘리먼트 단위로 분할되어 문서의 구조 정보는 DTDID, DID, EID, EName, EPath, ParentID 속성으로 구성된 Element_Info 테이블에 저장되고, 문서의 내용 정보는 DTDID, DID, EID, EPath, Content 속성으로 구성된 Element_Content 테이블에 저장된다. DTDID에는 DTD의 ID가, DID에는 XML 문서의 ID가, EID에는 엘리먼트의 ID(깊이 우선 탐색(DFS) 순으로 ID 부여)가, EName에는 엘리먼트의 이름이, EPath에는 문서내 루트 엘리먼트로부터 해당 엘리먼트까지의 경로가, ParentID에는 부모 엘리먼트의 ID가 저장된다. 경로 정보에 대해서는, 저장 공간의 절약을 위하여 [15]에서와 같이 각 경로와 그 식별자만 따로 저장하는 별도 테이블을 두고 EPath 속성에는 그 식별자 값만 저장할 수도 있다. (그림 5)는 (그림 3)(a)의 bookstore DTD를 준수하는 XML 문서들을 Element_Info 테이블과 Element_Content 테이블에 저장한 예이다.

VID	ViewDef	DTDID
1	bookstore/book/author/[first-name= "Michael"]	1

(그림 6) View_Definition 테이블의 구조 및 예

VID	VEID	DTDID	DID	EID	EName	VEPath	VParentID
1	1	1	-	-	mv	mv	NULL
1	2	1	1	5	author	mv/author	1
1	3	1	1	6	first-name	mv/author/first-name	2
1	4	1	1	7	#PCDATA	mv/author/first-name/#PCDATA	3
1	5	1	1	8	last-name	mv/author/last-name	2
1	6	1	1	9	#PCDATA	mv/author/last-name/#PCDATA	5
1	7	1	1	10	author	mv/author	1
1	8	1	1	12	first-name	mv/author/first-name	7
1	9	1	1	13	#PCDATA	mv/author/first-name/#PCDATA	8
1	10	1	1	14	last-name	mv/author/last-name	7
1	11	1	1	15	#PCDATA	mv/author/last-name/#PCDATA	10
1	12	1	2	5	author	mv/author	1
1	13	1	2	6	first-name	mv/author/first-name	12
1	14	1	2	7	#PCDATA	mv/author/first-name/#PCDATA	13
1	15	1	2	8	last-name	mv/author/last-name	12
1	16	1	2	9	#PCDATA	mv/author/last-name/#PCDATA	15

(a) View_Info 테이블

VID	VEID	DTDID	DID	EID	VEPath	Content
1	5	1	1	7	mv/author/first-name/#PCDATA	Michael
1	7	1	1	9	mv/author/last-name/#PCDATA	Johnson
1	10	1	1	12	mv/author/first-name/#PCDATA	Michael
1	12	1	1	14	mv/author/last-name/#PCDATA	Kay
1	16	1	2	7	mv/author/first-name/#PCDATA	Michael
1	18	1	2	9	mv/author/last-name/#PCDATA	Johnson

(b) View_Content 테이블

(그림 7) View_Info 테이블과 View_Content 테이블의 구조 및 예

3.1.2 XML 실체뷰 영역의 테이블 스키마

View_Definition 테이블은 VID, ViewDef, 그리고 DTDID 속성으로 구성된다. VID에는 뷰의 ID가, ViewDef에는 뷰의 정의(즉, 경로 표현식으로 나타낸 XML 질의 구문)가, DTDID에는 뷰가 참조하는 DTD의 ID가 저장된다. 뷰 정의에 의해 생성된 XML 실체뷰는 엘리먼트 mv를 루트로 하여 검색된 엘리먼트들을 그 자식 엘리먼트로 하는 XML 문서이다(이는 eXcelon에서 복수개의 문서에 대한 질의 결과 형식과 같다.). 따라서 하부 XML 소스 문서처럼 엘리먼트 단위로 분할되어 View_Info 테이블과 View_Content 테이블에 저장된다. View_Info 테이블은 VID, VEID, DTDID, DID, EID, EName, VEPath, VParentID 속성으로 구성된다. VID에는 뷰의 ID가, VEID에는 뷰 엘리먼트의 ID가, DID와 EID에는 뷰 엘리먼트와 사상(mapping)되는 하부 XML 문서 엘리먼트의 DID와 EID가 저장된다. EName에는 뷰 엘리먼트의 이름이, VEPath에는 뷰 엘리먼트의 경로 정보가, VParentID에는 뷰 내 부모 엘리먼트의 ID가 저장된다. View_Content 테이블은 VID, VEID, DTDID, DID, EID, VEPath, Content 속성으로 구성된다. (그림 6)과 (그림 7)은 이들 테이블의 구조와 예를 나타낸 것으로 (그림 5)의 Element_Info 테이블과 Element_Content 테이블에 저장된, XML 문서를 대상으로 XML 질의 "/bookstore/book/author[first-name="Michael"]"를 수행한 결과가 실체뷰로 저장된 예를 나타내고 있다.

3.2 XML 경로 표현식의 SQL로의 변환

XML 소스 및 XML 실체뷰에 대한 XML 경로 표현식은 이들이 관계 DBMS의 테이블에 저장되어 있기 때문에 SQL로 변환되어 처리된다. MV 유형의 경우, 변환된 XML 경로 표현식은 View_Info 테이블과 View_Content 테이블에 대한 SQL 구문으로 변환되고, BD+MV 유형의 경우, XML 소스 및 실체뷰에 대한 XML 경로 표현식은 각각 Element_Info 테이블/Element_Content 테이블에 대한 SQL 구문과 View_Info 테이블/View_Content 테이블에 대한 SQL 구문으로 변환된다.

다음과 같은 DTD를 준수하는 XML 문서들이 저장되어 있다고 가정하자. 즉, 이들 문서는 a를 루트 엘리먼트로 하고, b와 c는 각각 a의 자식 엘리먼트와 손자 엘리먼트가 되며, d와 e는 c의 자식 엘리먼트로서 단말(leaf) 엘리먼트인 구조를 갖는다.

```
<!ELEMENT a (b)>
<!ELEMENT b (c)>
<!ELEMENT c (d | e)+>
<!ELEMENT d (#PCDATA)>
<!ELEMENT e (#PCDATA)>
```

이 DTD의 ID = n이라고 할 때, 이들 문서들에 대하여 경로 표현식 a/b/c가 주어지면, 이는 View_Info 테이블과 View_Content 테이블에 대해 각각 다음과 같은 SQL 문으로 변환된다. 여기서 ORDER BY 절은 질의 결과인 튜플 스트림에 대해 XML 태깅을 효율적으로 수행하기 위해 사용된 것이다(3.3절 참조). 변환된 SQL 문의 효율적인 실행을 위해서는 Element_Info 테이블과 Element_Content 테이블의 DTDID, EName, 그리고 EPath 속성에 대한 인덱스가 필요하다. 마찬가지로 View_Info 테이블과 View_Content 테이블의 VID, EName, VEPath 속성에 대해서도 인덱스가 필요하다.

```
SELECT *
FROM   Element_Info
WHERE  DTDID=n AND
       ((EName = 'c' AND EPath = 'a/b/c') OR
       (EName = 'd' AND EPath = 'a/b/c/d') OR
       (EName = '#PCDATA' AND EPath = 'a/b/c/d/#
            PCDATA') OR
       (EName = 'e' AND EPath = 'a/b/c/e') OR
       (EName = '#PCDATA' AND EPath = 'a/b/c/e/#
            PCDATA'))
ORDER BY DID, EID
```

DTDID	DID	EID	Ename	EPath	ParentID
1	1	1	bookstore	bookstore	NULL
1	1	2	book	bookstore/book	1
1	1	3	title	bookstore/book/title	2
1	1	4	#PCDATA	bookstore/book/title/#PCDATA	3
1	1	5	author	bookstore/book/author	2
1	1	6	first-name	bookstore/book/author/first-name	5
1	1	7	#PCDATA	bookstore/book/author/first-name/#PCDATA	6
1	1	8	last-name	bookstore/book/author/last-name	5
1	1	9	#PCDATA	bookstore/book/author/last-name/#PCDATA	8
1	1	10	author	bookstore/book/author	2
1	1	11	first-name	bookstore/book/author/first-name	10
1	1	12	#PCDATA	bookstore/book/author/first-name/#PCDATA	11
1	1	13	last-name	bookstore/book/author/last-name	10
1	1	14	#PCDATA	bookstore/book/author/last-name/#PCDATA	13
1	1	15	author	bookstore/book/author	2
1	1	16	first-name	bookstore/book/author/first-name	15
1	1	17	#PCDATA	bookstore/book/author/first-name/#PCDATA	16
1	1	18	last-name	bookstore/book/author/last-name	15
1	1	19	#PCDATA	bookstore/book/author/last-name/#PCDATA	18
1	1	20	content	bookstore/book/content	2
1	1	21	title	bookstore/book/content/title	20
1	1	22	#PCDATA	bookstore/book/content/title/#PCDATA	21
1	1	23	chapter	bookstore/book/content/title/chapter	20
1	1	24	title	bookstore/book/content/title/chapter/title	23
1	1	25	#PCDATA	bookstore/book/content/title/chapter/title/#PCDATA	24
1	1	26	section	bookstore/book/content/title/chapter/section	23
1	1	27	sentence	bookstore/book/content/title/chapter/section/sentence	26
1	1	28	#PCDATA	bookstore/book/content/title/chapter/section/sentence/#PCDATA	27
1	1	29	sentence	bookstore/book/content/title/chapter/section/sentence	26
1	1	30	#PCDATA	bookstore/book/content/title/chapter/section/sentence/#PCDATA	29
1	1	31	price	bookstore/book/price	2
1	1	32	#PCDATA	bookstore/book/price/#PCDATA	31
1	2	1	bookstore	bookstore	NULL
1	2	2	book	bookstore/book	1
1	2	3	title	bookstore/book/title	2
1	2	4	#PCDATA	bookstore/book/title/#PCDATA	3
				…	

(a) Element_Info 테이블

DTDID	DID	EID	EPath	Content
1	1	4	bookstore/book/title/#PCDATA	XML Database
1	1	7	bookstore/book/author/first-name/#PCDATA	Michael
1	1	9	bookstore/book/author/last-name/#PCDATA	Johnson
1	1	12	bookstore/book/author/first-name/#PCDATA	Michael
1	1	14	bookstore/book/author/last-name/#PCDATA	Kay
1	1	17	bookstore/book/author/first-name/#PCDATA	Andy
1	1	19	bookstore/book/author/last-name/#PCDATA	Kay
1	1	22	bookstore/book/content/title/#PCDATA	1. Introduction
1	1	25	bookstore/book/content/title/chapter/title/#PCDATA	1.1 Why XML and Databases
1	1	28	bookstore/book/content/title/chapter/section/sentence/#PCDATA	There are may reasons why we might wish to…
1	1	30	bookstore/book/content/title/chapter/section/sentence/#PCDATA	In this book, we'll see how XML may…
1	1	31	bookstore/book/price/#PCDATA	$22
1	2	4	bookstore/book/title/#PCDATA	XML Beginning
			…	

(b) Element_Content 테이블

(그림 5) Element_Info 테이블과 Element_Content 테이블의 구조 및 예

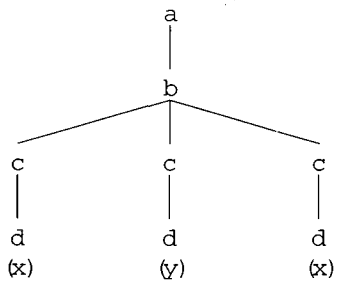
```
SELECT *
FROM   Element_Content
WHERE  DTDID=n AND
       (EPath = 'a/b/c/d/#PCDATA' OR EPath =
       'a/b/c/e/#PCDATA')
ORDER BY DID, EID
```

한편, 만약 경로 표현식 내에 필터 연산이 포함되어 있으면 이는 SQL 구문의 WHERE 절에 반영되어야 한다. 예를 들어, ID = n인 DTD를 준수하는 하부 XML 문서들에 대하여 경로 표현식 a/b/c/[d='x']가 주어지면, 이는 Element_Info 테이블과 Element_Content 테이블에 대해 각각 다음과 같은 SQL 구문으로 변환된다.

```
SELECT *
FROM   Element_Info
WHERE  DTDID=n AND
         ((EName = 'c' AND EPath = 'a/b/c') OR
         (EName = 'd' AND EPath = 'a/b/c/d') OR
         (EName = '#PCDATA' AND EPath =
                'a/b/c/d/#PCDATA') OR
         (EName = 'e' AND EPath = 'a/b/c/e') OR
         (EName = '#PCDATA' AND EPath =
                'a/b/c/e/#PCDATA'))
         AND DID IN(SELECT DID
                    FROM Element_Content
                    WHERE DTDID = n
                          AND EPath = 'a/b/c/d/
                              #PCDATA'
                          AND Content = 'x'
ORDER BY DID, EID
```

```
SELECT *
FROM Element_Content
WHERE DTDID=n
      AND EPath = 'a/b/c/e/#PCDATA'
    AND Content = 'x'
ORDER BY DID, EID
```

그런데 상기 SQL 구문은 해당 필터 연산([d='x'])을 항상 정확하게 처리하지 못하는 문제가 있다. 예를 들어, 다음 그림과 같은 트리 구조를 갖는 XML 문서가 있다고 가정하자 (노드는 엘리먼트를, 괄호 안의 내용은 값을 나타낸다).



경로 표현식 a/b/c/[d='x']을 변환시킨 상기 SQL 문을 이 문서를 대상으로 수행하면 c를 루트로 하는 세개의 서브 트리를 결과로 반환하게 된다. 즉, 엘리먼트 d의 값이 x인 서브 트리 두 개와 y인 서브 트리 한 개가 이들이다. 즉, 상기 SQL 문은 엘리먼트 d의 값이 x가 아니고 y인 서브 트리를 그 결과에서 제외시키지 못한다. 따라서 경로 표현식에서 변환된 SQL 문의 결과로 반환되는 튜플 스트림은 추가 필터링 작업을 요한다. 이 작업은 SQL 문의 결과 셋을 구성하는 튜플 스트림이 ORDER BY DID, EID 절에 의해 문서 별로 클러스터링되어 반환되므로 문서 단위로 해당 튜플들을 버퍼링한 후 수행할 수 있다.

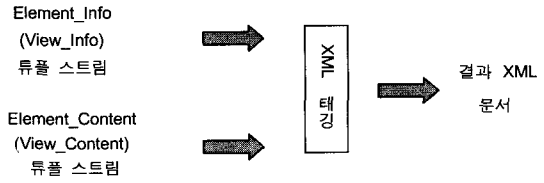
경로 표현식 내에 제외 연산자[13]가 포함된 경우의 SQL 변환은, 제외 연산자가 없는 경우(즉, a/b/c)의 SQL 문의 WHERE 절에서 제외시키고자 하는 엘리먼트에 대한 조건을 삭제하면 된다. 예를 들어, ID = n인 DTD를 준수하는 하부 XML 문서들에 대하여 경로 표현식 a/b/c\d가 주어지면, 이는 Element_Info 테이블과 Element_Content 테이블에 대해 각각 다음과 같은 SQL 구문으로 변환된다.

```
SELECT *
FROM   Element_Info
WHERE  DTDID = n AND
         ((EName = 'c' AND EPath = 'a/b/c') OR
         (EName = 'e' AND EPath = 'a/b/c/e') OR
         (EName = '#PCDATA' AND EPath =
         'a/b/c/e/#PCDATA'))
ORDER BY DID, EID
```

```
SELECT *
FROM Element_Content
WHERE DTDID = n
      AND EPath = 'a/b/c/e/#PCDATA'
ORDER BY DID, EID
```

3.3 XML 태깅

XML 경로 표현식으로부터 변환된 SQL 문의 수행 결과는 튜플 집합이다. 이 결과를 가지고 질의의 최종 결과를 XML로 구성하기 위해서는 적절한 태깅이 이루어져야 한다. 본 구현에서는 SQL 문의 결과를 구성하는 튜플이 반환될 때마다 즉각 그것을 태깅 프로세스로 넘겨 질의 결과를 XML로 생성하도록 하였다. 즉, SQL 문의 수행 결과인 튜플 집합을 모두 한 곳에 버퍼링한 후에 비로소 태깅 작업에 들어가는 것이 아니라, SQL 문의 결과 튜플이 하나씩 발생할 때마다 태깅 프로세스에 바로 입력되어 최종 결과 XML 문서를 생성하는 파이프라인(pipeline) 방식의 태깅이 수행된다. 단, 앞 절에서 기술한 것과 같이 필터 연산자가 있는 경우에는 반환되는 결과 튜플 스트림에서 문서 단위로(즉, 동일한 DID 값을 갖는 튜플들에 대해) 일시적인 버퍼링이 요구된다. 구현된 XML 태깅의 유형은 크게 경로 표현식에 조건(필터 연산)이 명시된 경우와 명시되지 않은 경우로 구분되고, 이들을 다시 BD 또는 MV 유형의 경우와 BD+MV 유형의 경우로 구분된다.



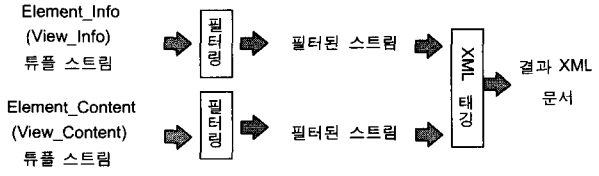
(그림 8) BD 또는 MV 유형의 태깅 : 조건이 명시되지 않는 경우

3.3.1 BD 또는 MV 유형의 태깅 : 조건이 명시되지 않은 경우

(그림 8)은 경로 표현식 내에 조건이 명시되지 않은 BD 또는 MV 유형의 질의 결과 튜플 스트림에 대한 XML 태깅 과정을 나타낸 것이다. BD 유형의 경우 Element_Info 테이블과 Element_Content 테이블로부터 튜플 스트림이 추출되고, MV 유형의 경우 View_Info 테이블과 View_Content 테이블로부터 튜플 스트림이 추출된다. 이들 튜플 스트림은 해당 SQL 문의 ORDER BY 절에 의해 DID, EID 값에 의해 정렬된 상태로 생성된다. 3.1절에서 기술한 바와 같이 XML 문서의 엘리먼트는 EID 값에 의해 DFS 순으로 저장되기 때문에 SQL 문의 결과 튜플 스트림이 EID 값에 의해 정렬되어 생성된다면 이들을 임시 공간에 버퍼링할 필요없이 튜플이 반환될 때마다 하나씩 순차적으로 태깅 프로세스에 입력하여 최종 결과 XML 문서를 생성할 수 있다. 즉, 테이블 검색으로부터 태깅 과정으로 이어지는 파이프라인 방식의 태깅이 가능한 것이다.

3.3.2 BD 또는 MV 유형의 태깅 : 조건이 명시된 경우

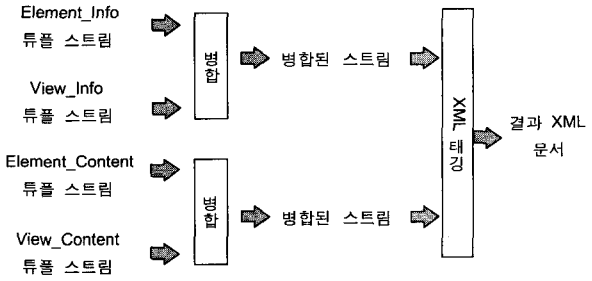
(그림 9)는 질의의 경로 표현식 내에 조건이 명시된 BD 또는 MV 유형의 질의 결과 튜플 스트림에 대한 XML 태깅 과정을 나타낸 것이다. 조건이 명시되지 않은 경우와는 달리 각 테이블에서 추출된 튜플들을 태깅 과정으로 넘기기에 앞서 문서별로 추가 필터링을 수행한다. 이는 3.2절에서 기술한 것과 같이 경로 표현식에 조건이 명시된 경우에 생성된 SQL 문은 해당 필터 연산을 항상 완전히 처리하지 못하기 때문이다. 이 추가 필터링을 거친 튜플 스트림에 대해서는 조건이 명시되지 않은 경우와 동일하게 파이프라인 방식의 태깅을 수행한다.



(그림 9) BD 또는 MV 유형의 태깅 : 조건이 명시된 경우

3.3.3 BD+MV 유형의 태깅

(그림 10)은 BD+MV 유형의 질의 결과 튜플 스트림에 대한 XML 태깅 과정을 나타낸 것이다. 이 유형에서는 모두 네 개의 테이블 즉, Element_Info 테이블, Element_Content 테이블, View_Info 테이블, 그리고 View_Content 테이블로부터 튜플 스트림이 생성되는데, XML 태깅을 수행하기 전에 Element_Info 테이블의 튜플 스트림과 View_Info 테이블의 튜플 스트림이 서로 병합(merge)되고, Element_ Content 테이블의 튜플 스트림과 View_Content 테이블의 튜플 스트림 역시 단일 스트림으로 병합된다. 병합 대상 튜플 스트림은 각각 DID와 EID 값에 의해 정렬되어 있는데 병합 결과도 이 정렬 순서를 유지한다. 이 병합은 BD+MV 유형의 질의 처리에서 실체뷰에 대한 질의(q')의 결과와 하부 XML 소스에 대한 질의(q")의 결과를 통합하는 효과를 갖는다. 즉, 뷰의 하부 XML 소스로부터 검색된 내용과 실체뷰로부터 검색된 내용이 한 튜플 스트림으로 합쳐짐으로써, 그 내용이 하부 문서로부터 온 것인지 실체뷰로부터 온 것인지의 구분이 없어진다. 따라서 병합된 튜플 스트림에 대한 이후 태깅 프로세스는 MV 또는 BD 유형에서의 그것과 동일해진다.



(그림 10) BD+MV 유형의 태깅

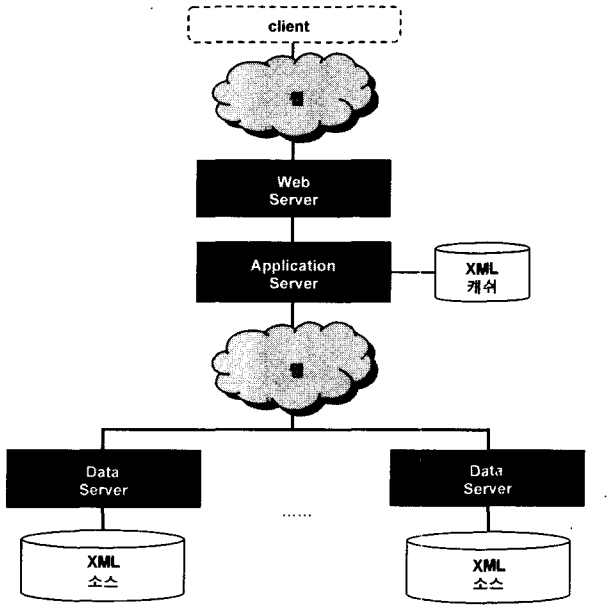
4. 성능 평가

본 장에서는 구현된 프로토타입 XML 저장 시스템을 대상으로 웹 상에서 실체뷰를 이용한 XML 질의 처리의 성능을 다양한 실험을 통해 평가한 결과를 기술한다. 아울러 기존 관련 연구에서 제시된 기법과의 성능 비교 검토 결과도 기술한다. 본 논문의 실험은, XML 데이터베이스 기반 웹 응용을 지원하는 웹 서버, 응용 서버, 데이타 서버의 다계층 구조를 대상으로, XML 소스는 웹 상의 원격 사이트에 존재하고 XML 실체뷰는 [5-7] 등의 연구에서와 같이 응용 서버에 캐쉬되는 환경을 대상으로 하였다((그림 11) 참조). 본 실험은 Pentium III 550MHz Dual CPU와 1024 MB RAM이 장착된 Windows 2000 Server상에서 수행되었으며 웹 환경은 네트워크 대역폭을 1Mbps로 가정하고 시뮬레이션되었다.

4.1 개 요

본 실험의 성능 척도는 웹 응용에서 XML 질의 처리를 수행할 때의 응답 시간이다. 질의 처리시 응용 서버에 캐쉬된 XML 실체뷰를 이용할 수도 있고 그렇지 않을 수도 있는데 질의가 응용 서버에 전달되기까지의 시간과 질의 결

과가 응용 서버로부터 사용자에게 전달되는 시간은 양자 모두 동일하므로 측정에서 제외하였다. 즉, 응용 서버에 XML 질의가 제기된 시점부터 질의 결과(관계 DBMS로부터 검색된 결과 셋을 태깅 처리한 XML 문서)가 응용 서버에 확보되기까지 걸린 시간으로, MV, BD+MV, 그리고 BD 유형의 XML 질의가 주어졌을 때, 이들 간에 처리 시간을 비교하였다.



(그림 11) XML 데이타베이스 기반 웹 응용을 지원하는 다계층 구조

실험에 사용된 XML 문서는 그 크기 측면에서 두 종류로 나눌 수 있는데, 첫째는 엘리먼트 개수의 평균이 25개 정도이고 그 평균 크기가 1.17 KB 정도인 작은 "Bookstore" XML 문서(이하, "Bookstore" 문서)이고, 둘째는 엘리먼트 개수의 평균이 8,500개 정도이고 그 평균 크기가 289KB 정도인 큰 "The Plays of Shakespeare" XML 문서[16](이하, "Play" 문서)이다. 여기서 문서 크기의 의미는 다음과 같다. 보통 웹 상의 XML 문서는 단순히 엘리먼트와 그 내용만으로 구성되어 있는 것이 아니라, 엘리먼트 간의 계층 구조를 이해하기 쉽도록 공백나 탭을 이용하여 들여 쓰기(indentation)가 되어 있다. 즉, 같은 내용의 XML 문서라도 공백이나 탭을 얼마나 사용했는가에 따라 XML 문서의 크기가 달라진다. 본 실험에서는 공백이나 탭을 제외한 XML 문서를 대상으로 하였다. 따라서 질의 결과도 공백이나 탭이 제외된 XML 문서다.

<표 1>은 본 실험에서 사용된 질의 q, 실체뷰 정의 v, 질의 변환에 의해 생성된 실체뷰에 대한 질의 q′, XML 소스에 대한 질의 q″을 정리한 것이다. 예를 들어 (그림 13)의 실험에서 사용된 q와 v의 경로 표현식은 PLAY/FM이고, 변환된 q′의 경로 표현식은 mv/FM임을 나타낸다[13]. (그림 12)의 실험에서는 두개의 서로 다른 뷰 정의로 구성된 실체뷰를 대상으로 MV 유형의 성능을 측정했기 때문에 MV-I과 MV-II로 나누어 표시하였는데 [ ] 안에 표시된 % 값은, MV-I의 실체뷰는 뷰 정의에 의해 XML 소스의 31%에 해당되는 뷰를 실체화한 것이고, MV-II의 실체뷰는 18%에 해당되는 뷰를 실체화한 것임을 나타낸다. (그림 14)와 (그림 15)의 실험에서는 BD+MV 유형의 질의 처리시 세 개의 서로 다른 뷰 정의로 구성된 실체뷰를 대상으로 실험하였는데 ( ) 안에 표시된 % 값은, 질의 q의 결과 엘리먼트들 중 실체뷰로부터 검색된 것의 비율 p(즉, 실체뷰에 대한 질의 q′의 결과 엘리먼트가 q의 최종 결과에서 차지하는 비율)를 나타낸다. 예를 들어, 질의 q의 결과 엘리먼트를 XML 소스와 실체뷰에서 각각 반씩 얻었다면, p = 50%, 결과 전체를 XML 소스로부터 얻었다면 p = 0%, 그리고 결과 전체를 실체뷰로부터 얻었다면 p = 100%가 된다.

〈표 1〉 실험에서 사용된 q, v, q′, q″의 경로 표현식

실험 결과	구 분		경로 표현식
(그림 12)	q		bookstore/book/author/first-name
	MV-I [31%]	v	bookstore/book/author
		q′	mv/author/first-name
	MV-II [18%]	v	bookstore/book/author/first-name
		q′	mv/first-name
(그림 13)	q		PLAY/FM
	MV [1.2%]	v	PLAY/FM
		q′	mv/FM
(그림 14)	q		bookstore/book/content
	MV [28%]	v	bookstore/book/content
		q′	mv/content
	BD+MV (96%)	v	bookstore/book/content/chapter
		q′	mv/chapter
		q″	bookstore/book/content\chapter
	BD+MV (55%)	v	bookstore/book/content/chapter/section
		q′	mv/section
		q″	bookstore/book/content\chapter/section
	BD+MV (30%)	v	bookstore/book/content/chapter/section/sentence
		q′	mv/sentence
		q′	bookstore/book/content\chapter/section/sentence
(그림 15)	q		PLAY/PERSONAE
	MV [2.3%]	v	PLAY/PERSONAE
		q′	mv/PERSONAE
	BD+MV (26%)	v	PLAY/PERSONAE/PGROUP
		q′	mv/PGROUP
		q″	PLAY/PERSONAE\PGROUP
	BD+MV (17%)	v	PLAY/PERSONAE/PGROUP/PERSONA
		q′	mv/PERSONA
		q″	PLAY/PERSONAE\PGROUP/PERSONA
	BD+MV (6%)	v	PLAY/PERSONAE/PGROUP/PERSONA/PERSON
		q′	mv/PERSON
		q′	PLAY/PERSONAE\PGROUP/PERSONA/PERSON

4.2 실험 결과

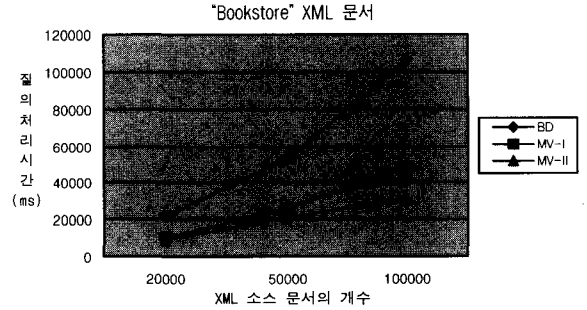
4.2.1 MV와 BD의 비교

BD 유형의 질의 처리 시간은 다음 식과 같다.

BD 유형 질의 처리 시간 =
질의 q의 전송 시간
+ 원격 XML 소스를 대상으로 튜플 스트림 검색 시간
+ XML 태깅 시간
+ 결과 XML 전송 시간

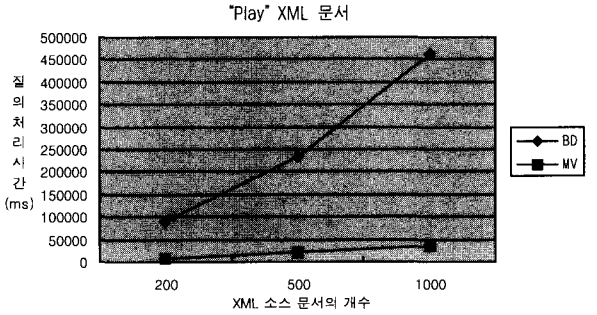
첫 번째 항 질의 q의 전송 시간은 응용 서버에서 원격 데이타 서버로 질의 q를 전송하는 데 소요되는 시간이다. 두번째 항은 원격 데이타 서버에서 XML 소스를 대상으로 튜플 스트림을 검색하는 데 소요되는 시간이다. 이 튜플 스트림에 대한 XML 태깅은 데이터 서버에서 수행되어 최종 결과 XML을 생성하게 된다(세 번째 항). 이후 결과 XML을 원격 데이타 서버에서 응용 서버로 전송하는 시간이 소요된다(네 번째 항). 본 실험에서는 질의 q의 전송 시간은 미미하므로 무시하였고 최종 결과 XML 전송 시간은 최종 결과 XML의 크기를 네트워크 대역폭으로 나누어 계산하였다.

MV 유형의 질의 처리 시간은 위 식에서 첫 번째와 네 번째 항을 제외하고 두 번째 항은 원격 XML 소스에 대한 것이 아니라 응용 서버의 XML 캐쉬에 대한 것으로 바뀐 것이다.



(그림 12) XML 소스 문서 개수의 변화에 따른 질의 처리 시간 비교("Bookstore" 문서)

(그림 12)는 XML 문서의 엘리먼트 수가 작은 "Bookstore" 문서를 대상으로 한 실험 결과를 나타낸 것으로 XML 소스의 18%에 해당되는 데이타를 검색하는 질의를 처리한 것이다. 곡선 MV-I는 XML 소스의 31%에 해당되는 실체뷰를 이용하여 MV 유형의 질의 처리를 수행했을 때의 성능을 나타낸 것이고, MV-II는 XML 소스의 18%에 해당되는 실체뷰(즉, 실험 질의의 답과 완전 일치하는 경우에 해당)를 이용한 MV 유형 질의 처리의 성능을 나타낸 것이다. (그림 12)에 나타난 바와 같이 BD보다 MV의 성능이 우수하고, XML 소스 문서의 개수가 증가할수록 BD와 비교하여 MV-I 및 MV-II의 성능 향상폭이 점점 커짐을 알 수 있다. XML 소스 문서의 개수가 20000개일 때 BD : MV-I : MV-II의 비율이 1.19 : 1 : 1이었는데, 50000개 일 때는 2.6 : 1.2 : 1, 그리고 100000개일 때는 3.5 : 1.58 : 1로 성능 향상폭이 증가하였다. 이는 XML 소스 문서의 개수가 커질수록 BD의 경우 검색 시간이 증가하며 또한 네트워크 상으로 전송되어야 하는 그 결과 크기도 커지기 때문이다.



(그림 13) XML 소스 문서 개수의 변화에 따른 질의 처리 시간 비교("Play" 문서)

(그림 13)은 XML 문서의 엘리먼트 수가 큰 "Play" 문서를 대상으로 한 실험 결과를 나타낸 것으로 XML 소스의 1.2%에 해당되는 데이타를 검색하는 질의를 처리한 것이다. MV는 XML 소스의 1.2%에 해당되는 실체뷰를 이용한 경우의 성능을 나타내고 있다. XML 소스 문서의 개수가 200개, 500개, 그리고 1000개일 때 BD : MV의 비율이 모두 약 13.9 : 1로 나타났다. 이와 같이 "Bookstore" 문서에 대한 실험에 비해 BD 대비 MV의 성능 향상폭이 훨씬 커진 것은 XML 소스 문서의 양이 훨씬 커져서 BD의 성능 저하를 초래했기 때문이다.

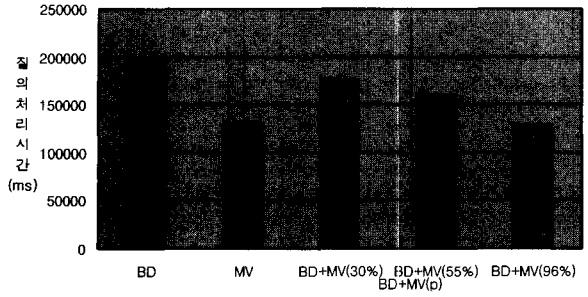
4.2.2 BD+MV와 BD의 비교

BD+MV 유형의 질의 처리 시간은 다음 식과 같다.

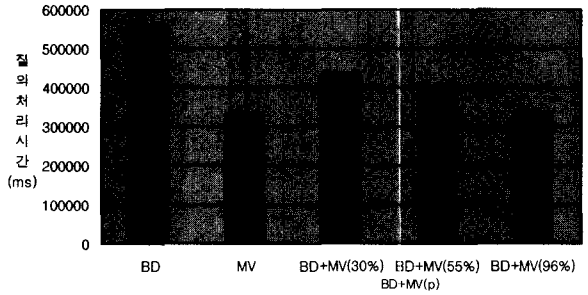
BD+MV 유형 질의 처리 시간 =
질의 q″의 전송 시간
+ max(XML 소스 대상 튜플 검색 시간 + 튜플 스트림 전송 시간, XML 실체뷰 대상 튜플 검색 시간)
+ 튜플 스트림 병합 및 XML 태깅 시간

첫 번째 항 질의 q″의 전송 시간은 응용 서버에서 원격 데이타 서버로 질의 q″전송하는 데 소요되는 시간이다. 두 번째 항은 응용 서버에서 실체뷰를 대상으로 q′을 수행하는 과정과 원격 데이타 서버에서 q″수행한 후 그 결과 튜플 스트림을 응용 서버로 전송하는 과정이 병렬로 처리되기 때문에 이들 양자 중 오래 걸리는 쪽의 시간이다. BD 유형에서는 XML 태깅 처리까지 완료된 최종 결과 XML이 데이터 서버에서 응용 서버로 전송되지만, BD + MV 유형에서는 원격 데이터 서버에서 검색된 XML 태깅 처리 전의 튜플 스트림이 응용 서버로 전송되어, 응용 서버에서 실체뷰로부터 검색된 튜플 스트림과 병합된 후 태깅 처리된다(3.3.3절 참조). 세 번째 항은 응용 서버에서 수행되는 튜플 스트림 병합 및 XML 태깅에 소요되는 시간이다. 본 실험에서는 질의 q″의 전송 시간은 미미하므로 무시하였다. 데이터 서버에서 응용 서버로의 튜플 스트림 전송 시간은 XML 소스를 대상으로 q″을 처리한 결과 셋의 크기를 네트워크

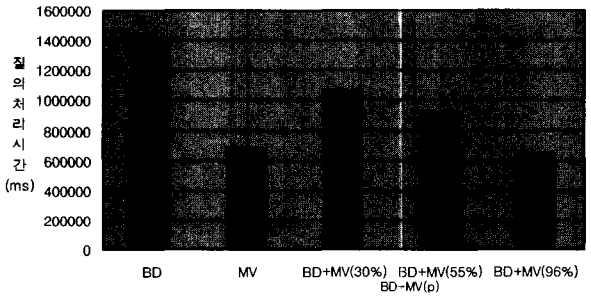
대역폭으로 나누어 계산하였다.



(a) XML 소스 문서의 개수가 20000개인 경우



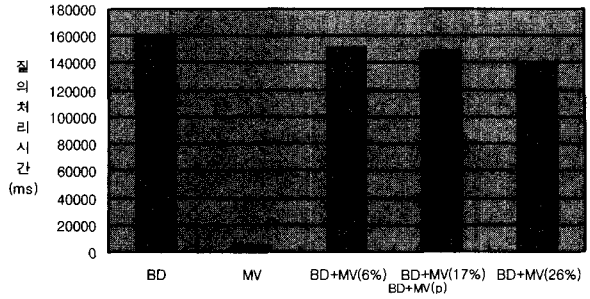
(b) XML 소스 문서의 개수가 50000개인 경우



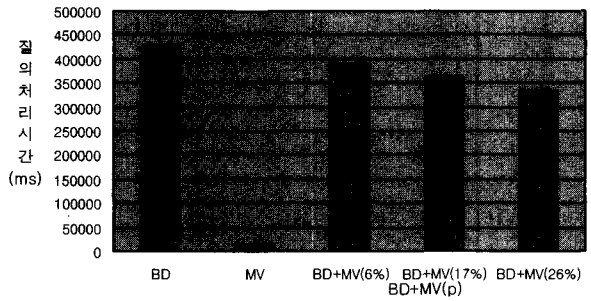
(c) XML 소스 문서의 개수가 100000개인 경우

(그림 14) 비율 p의 변화에 따른 BD+MV와 BD의 질의 처리 시간 비교("Bookstore" 문서)

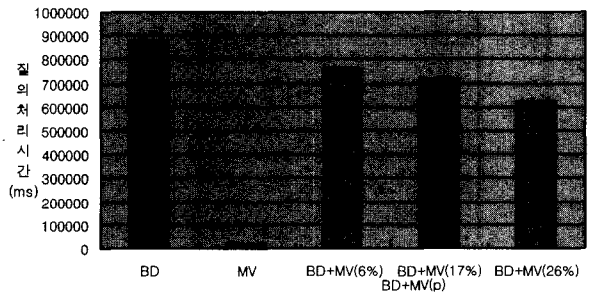
(그림 14)는 XML 문서의 엘리먼트 수가 작은 "Bookstore" 문서를 대상으로 한 실험 결과를 나타낸 것으로 XML 소스의 28%에 해당되는 데이타를 검색하는 질의를 처리한 것이다. XML 소스 문서의 개수가 20000개, 50000개, 100000개일 때 각각 BD+MV와 BD의 성능을 비교한 결과인데(질의 결과와 완전 일치하는 실체뷰를 이용한 MV의 성능도 참고로 함께 나타내었다.) BD+MV가 BD보다 우수하게 나타났으며 비율 p가 증가함에 따라 BD+MV의 성능이 계속 향상되는 것으로 나타났다. XML 소스 문서의 개수가 20000개일 때 BD : BD+MV(30%) : BD+MV(55%) : BD+MV(96%)의 비율이 1 : 0.88 : 0.80 : 0.64이었는데, 50000개일 때는 1 : 0.78 : 0.72 : 0.61로, 그리고 100000개일 때는 1 : 0.77 : 0.67 : 0.46으로 성능 향상 폭이 증가하였다. 이는 BD+MV의 경우 XML 소스에 대한 접근과 실체뷰에 대한 접근이 웹 상의 서로 다른 사이트 상에서 병렬로 처리되고, BD에 비해 통신 비용도 적기 때문이다.



(a) XML 소스 문서의 개수 200개인 경우



(b) XML 소스 문서의 개수 500개인 경우



(c) XML 소스 문서의 개수 1000개인 경우

(그림 15) 비율 p의 변화에 따른 BD+MV와 BD의 질의 처리 시간 비교("Play" 문서)

(그림 15)는 XML 문서의 엘리먼트 수가 큰 "Play" 문서를 대상으로 한 실험 결과를 나타낸 것으로 XML 소스의 2.3%에 해당되는 데이타를 검색하는 질의를 처리한 것이다. XML 소스 문서의 개수가 200개, 500개, 1000개일 때 각각 BD+MV와 BD의 성능을 비교한 결과인데(질의 결과와 완전 일치하는 실체뷰를 이용한 MV의 성능도 참고로 함께 나타내었다.) (그림 14)의 실험 결과와 동일한 패턴을 나타냈다.

4.3 기존 연구와의 성능 비교

4.3.1 XCacher와의 비교

[5]에서 제시된 XCacher와 본 논문의 기법 간의 성능 비교를 위해서 먼저 두 기법 간의 차이점을 정리하면 다음

과 같다. 첫째, XML 캐쉬의 내용에서 차이가 있다. [5]에서는 전위 선택(prefix-selection) 질의의 결과인 소스 XML 트리의 전위 트리를 캐쉬한다. 반면, 본 논문의 XML 캐쉬는 경로 표현식이 지정하는 XML 서브 트리를 캐쉬한다. 둘째, XML 캐쉬의 저장 구조가 다르다. [5]에서는 캐쉬를 트리로 표현하고 저장하는 반면 본 논문에서는 관계 데이타베이스에 엘리먼트 단위로 분할 저장한다. 셋째, 캐쉬의 저장소가 다르다. [5]에서는 캐쉬가 메모리에 상주한다. 반면 본 논문에서는 관계 DBMS의 디스크에 저장된다.

XML 질의가 주어졌을 때 그것에 대해 MV 유형의 질의 처리가 가능할 경우, XCacher가 메모리 상주 캐쉬를 이용하기 때문에 본 논문의 시스템보다 좋은 성능을 나타낼 것이다. 그러나 유의할 점은, XCacher는 메모리 상주 캐쉬만 지원하므로 확장성(scalability)에서 제약이 크다는 것이다. 즉, 캐쉬로 지원할 수 있는 XML 질의의 수가 가용 메모리 용량에 제약을 받게된다. 따라서 XML 질의가 주어졌을 때 그것이 자주 제기되는 것이라 할지라도 MV 유형의 질의 처리가 가능할 확률은 본 논문의 기법에 비해 현저히 낮다고 볼 수 있다.

BD+MV 유형의 질의 처리에서는 두 기법 모두 하부 XML 소스에 대한 접근 및 질의 처리가 필요하다. 하부 소스에 대한 질의 처리의 성능은 소스 XML 문서의 저장 기법에 직접적인 영향을 받는다. Native 시스템을 사용하는가 아니면 관계 데이타베이스를 이용한 분할 저장 기법을 사용하는가에 따라, 또한 후자의 경우 그 테이블 스키마 설계에 따라, XML 질의 처리의 성능이 결정된다. 이 점은 본 논문의 범위를 벗어나며 XML 벤치마크를 통해 검토되고 있는 문제다[17, 18]. 단, BD+MV 유형의 질의 처리에서 XCacher는 하부 소스에 대한 질의 처리의 결과로 XML 형식의 결과를 반환하는 것이 요구되므로 만약 본 논문에서와 같이 관계 데이터베이스에 XML 문서를 분할 저장할 경우, XML 태깅 과정을 필요로 한다. 반면, 본 논문의 기법에서는 결과 통합을 위하여 검색된 튜플 스트림만 제공하면 되므로 더 효율적이다.

전체적인 질의 처리 과정의 차이를 비교하면, XCacher에서는 XQuery 질의가 주어지면 이를 분해하여 추출 부분(FOR-WHERE 절)에 해당되는 서브 질의를 캐쉬 및 하부 소스에 대한 질의로 나누어 처리하고 그 결과를 통합한 후 이에 대해 반환 부분(RETURN 절)에 해당되는 서브 질의를 다시 수행하는 순차적인 방법을 사용한다. 이는 XCacher가 제공하는 XML 캐쉬가 소스 문서 트리의 전위 트리이므로 BD+MV 유형의 질의 처리가 가능한 경우가 캐쉬된 XML 트리의 리프 노드를 루트로 하는 서브 트리를 하부 소스 문서에서 검색할 때로 국한되기 때문이다. 반면 본 논문의 BD+MV 유형 질의 처리에서는 이러한 제약이 없고 XML 소스 및 캐쉬가 모두 엘리먼트 단위로 분할 저장되어 있으므로 반환 부분을 바로 추출하는 SQL문의 작성이 가능하여 보다 효율적이다.

4.3.2 ACE-XQ와의 비교

[6]에서 제시된 ACE-XQ의 기법에서는 XQuery 질의 결과를 XML 문서로 캐쉬한다. 따라서 캐쉬를 이용한 관련 질의의 처리시 캐쉬된 질의 결과를 매번 파싱해야 하는 부담이 따른다. DOM 트리와 같은 파싱 결과에 대하여 ([6]에서는 [19]의 Quilt 파서를 사용) 필요한 연산을 수행한 후 그 결과를 XML 형식으로 반환하게 된다. 본 연구의 기법에서는 캐쉬가 관계 데이터베이스에 엘리먼트 단위로 분할 저장되어 있으므로 그러한 부담이 없어 MV 및 BD+MV 유형의 질의 처리에서 대부분의 경우 더 나은 성능을 얻을 수 있다(이러한 분할 저장을 위한 비용은 해당 하부 소스가 파싱될 때 한번만 지불하면 된다). 한편, BD+MV 유형의 경우, 캐쉬 및 소스에 대한 질의의 결과를 통합하는 과정에서 튜플 스트림을 정렬 순서에 의해 병합하는 것으로 처리되는 본 논문의 기법이 소스 및 캐쉬로부터 얻은 XML 트리 형태의 결과를 통합해야 하는 ACE-XQ보다 간단하고 효율적이므로 성능 차이의 또 다른 원인으로 작용할 것이다. ACE-XQ가 더 나은 성능을 나타낼 것으로 기대되는 경우는, MV 유형의 질의 처리에서 캐쉬된 질의와 동일한 질의가 제기되었을 경우이다. 이때는 파싱 없이 캐쉬된 XML 문서를 바로 반환만 하면 되므로 태깅 과정을 거쳐야 하는 본 연구의 기법보다 효율적이다.

5. 결 론

본 논문은 XML 질의를 관련 XML 캐쉬를 이용하여 처리하는 기법의 구현 및 성능 평가에 관한 것이다. 캐쉬를 이용한 XML 질의 처리 기능을 지원하는 XML 질의 처리 시스템 프로토타입이 관계 DBMS를 기반으로 구현되어 다양한 성능 실험에 이용되었다. XML 질의로는 모든 XML 질의어의 핵심 요소인 경로 표현식을 대상으로 하였고, XML 캐쉬는 XML 실체뷰를 고려하였다. 캐쉬를 이용한 질의 처리 유형 결정 및 변환 알고리즘은 [13]에 제시된 것을 이용하였다. 구현의 주요 이슈로는 XML 소스와 실체뷰를 저장하는 테이블 스키마, XML 질의의 SQL로의 변환, 그리고 SQL 결과 셋에 대한 XML 태깅을 들 수 있으며 본 구현에서 사용된 기법을 자세히 기술하였다. 구현된 시스템으로 데이터베이스 기반 웹 응용을 위한 웹 서버, 응용 서버, 데이터 서버의 다계층 구조 하에서 응용 서버에 캐쉬를 둘 때 원격 XML 소스에 대한 XML 질의 처리의 성능을 평가하였다. 실험 결과 실체뷰를 사용하지 않는 BD 유형의 처리에 비해, 질의의 답을 실체뷰로부터 모두 얻는 MV 유형이나 질의의 부분 답을 실체뷰와 하부 XML 소스로부터 각각 얻어 통합하는 BD+MV 유형의 성능이 모두 우수하게 나왔다.

향후 연구 과제로는, XML 소스 문서 및 실체뷰를 분할 저장하기 위한 테이블 스키마로 본 논문의 구현에서 사용된 것 이외의 것을 채택했을 때 캐쉬를 이용한 XML 질의

처리 성능에의 영향 파악, XML 경로 표현식에 정규 경로가 허용될 때 실체뷰를 이용한 질의 변환 알고리즘의 개발, 웹 환경에서 다수 사용자에 의한 질의 처리시 캐쉬 이용 여부에 따른 확장성(scalablilty) 비교 등이 있다.

참 고 문 헌

[1] A. Levy, A. Mendelzon, Y. Sagiv, D. Srivastava, "Answering Queries Using Views," Proc. of the ACM Int'l Symp. on PODS, pp.95-104, 1995.

[2] Y. Papakonstantinou and V. Vassalos, "Query Rewriting for Semistructured Data," Proc. of the ACM SIGMOD Int'l Conf. on Management of Data, pp.455-466, 1999.

[3] D. Calvanese, G. Giacomo, M. Lenzerini, and M. Vardi, "Answering Regular Path Queries Using Views," Proc. of the IEEE Int'l Conf. on Data Eng., pp.389-398, 2000.

[4] D. Florescu, A. Levy, and D. Suciu, "Query Containment for Conjunctive Queries with Regular Expressions," Proc. of the ACM Int'l Symp. on PODS, pp.139-148, 1998.

[5] V. Hristidis and M. Petropoulos, "Semantic Caching of XML Databases," Proc. of the Int'l Workshop on the Web and Databases, 2002.

[6] L. Chen and E. Rundensteiner, "ACE-XQ : A CachE-aware XQuery Answering System," Proc. of the Int'l Workshop on the Web and Databases, 2002.

[7] P. Marrón and G. Lausen, "Efficient Cache Answerability for XPath Queries," Proc. of the 2nd Int'l Workshop on Data Integration over the Web, pp.35-45, 2002.

[8] F. Tian, D. DeWitt, J. Chen, and C. Zhang, "The Design and Performance Evaluation of Alternative XML Storage Strategies," ACM SIGMOD Record, Vol.31, No.1, pp.5-10, Mar., 2002.

[9] A. Deutsch, M. Fernandez, D. Suciu, "Storing Semi-structured Data with STORED," Proc. of the ACM SIGMOD Int'l Conf. on Management of Data, pp.431-442, 1999.

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

[11] D. Florescu and D. Kossmann, "A Performance Evaluation of Alternative Mapping Schemes for Storing XML Data in a Relational Database," Tech. Rep., INRIA, France, 1999.

[12] J. Shanmugasundaram, K. Tufte, C. Zhang, G. He, D. DeWitt, and J. Naughton, "Relational Databases for Querying XML Documents : Limitations and Opportunities," Proc. of the Int'l Conf. on VLDB, pp.302-314, 1999.

[13] C. Moon, S. Kim, and H. Kang, "Processing XML Path Expressions Using XML Materialized Views," Proc. the 20-th British Nat'l Conf. on Databases, pp.19-37, Jul., 2003.

[14] S. Abiteboul, L. Segoufin, and V. Vianu, "Representing and Querying XML with Incomplete Information," Proc. of the ACM Int'l Symp. on PODS, 2001.

[15] T. Shimura, M. Yoshikawa, and S. Uemura, "Storage and Retrieval of XML Documents Using Object-Relational Databases," Proc. of the Int'l Conf. on Database and Expert Systems and Applications, 1999.

[16] J. Bosak, "The Plays of Shakespeare," http://www.ibiblio.org/bosak/, 1999.

[17] A. Schmidt, F. Wass, M. Kersten, M. Carey, I. Manolescu, and R. Busse, "XMark : A Benchmark for XML Data Management," Proc. of the Int'l Conf. on VLDB, 2002.

[18] B. Yao, M. Özsu, and J. Keenleyside, "XBench A Family of Benchmarks for XML DBMSs," Proc. of EEXTT 2002 and DiWeb, 2002.

[19] http://cheops.cis.upenn.edu/Kweelt, Oct., 2003.

문 찬 호

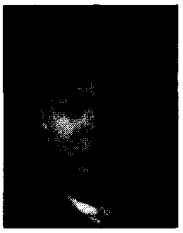
e-mail : moonch@dblab.cse.cau.ac.kr

1997년 중앙대학교 컴퓨터공학과 (공학사)

1999년 중앙대학교 컴퓨터공학과 대학원 (공학석사)

2003년 중앙대학교 컴퓨터공학과 대학원 (공학박사)

관심분야 : 웹 데이타베이스, XML, 질의 변환



박 정 기

e-mail : jkpark@dblab.cse.cau.ac.kr

2002년 중앙대학교 컴퓨터공학과(공학사)

2003년 중앙대학교 컴퓨터공학과 대학원 (공학박사)

관심분야 : XML Database-backed Web Applications, XML Query Rewriting, XML

강 현 철

e-mail : hckang@cau.ac.kr

1983년 서울대학교 컴퓨터공학과(공학사)

1985년 U. of Maryland at College Park, Computer Science(M.S.)

1987년 U. of Maryland at College Park, Computer Science(Ph.D.)

1988년~현재 중앙대학교 컴퓨터공학과 교수

관심분야 : XML 데이타베이스, 웹 데이타베이스, DBMS 저장 시스템