XML 질의 캐쉬의 저장 기법
(Storage Schemes for XML Query Cache)

김 영 현 † 강 현 철 ††
(Younghyun Kim) (Hyunchul Kang)

요 약 최근 XML 데이타베이스 기반 웹 응용을 위한 XML 질의 캐쉬 기법이 활발히 연구되고 있다. 이와 같은 XML 질의 캐쉬의 실용적인 중요성에도 불구하고, 캐쉬된 질의 결과를 어떻게 저장하는 것이 효율적인지에 대해서는 아직 아무런 연구가 없는 실정이다. 본 논문에서는 XML 질의 캐쉬의 저장 기법을 다룬다. XML 질의 캐쉬의 효율적인 저장 구조 설계에 있어 근본적으로 고려해야 하는 점은 캐쉬된 질의 결과에 대한 대표적인 두 종류 연산 간에 성능 트레이드오프가 존재한다는 것이다. 이 두 종류의 연산은 (1) 캐쉬된 질의 결과를 반환하기 위하여 캐쉬 **전체**를 **검색**하는 것과 (2) 소스 데이타의 변경에 대하여 캐쉬를 점진적으로 갱신하기 위하여 캐쉬의 **일부분**을 **변경**하는 것이다. 본 논문에서는 모두 여덟 개의 XML 질의 캐쉬 저장 기법을 제시한다. 이들은 크게 세 개의 그룹으로 나누어지는데, (1) 일반적인 **텍스트 화일**을 기반으로 한 기법 (2) **영속성 있는 DOM(PDOM) 화일**에 기반을 둔 기법 (3) **RDBMS**를 사용하는 기법이다. 이들 모두를 구현하여 성능을 비교하였고, 기존의 XML 저장 기술에 기반을 둔 질의 캐쉬 저장 기법과도 비교 평가하였다.

키워드 : XML 질의 캐쉬, 캐쉬의 점진적 갱신, XML 변경, DOM, PDOM

Abstract XML query caching for XML database-backed Web applications began to be investigated recently. Despite its practical significance, efficiency of the storage schemes for cached query results has not been addressed. In this paper, we deal with the storage schemes for XML query cache. A fundamental problem that needs to be considered in designing an efficient storage structure for XML query cache is that there exist performance tradeoffs between the two major types of operations on a cached query result. The two are (1) retrieving the **whole** of it to return the query result and (2) updating just a **small portion** of it for its incremental refresh against the updates done to its source. We propose eight different storage schemes for XML query cache, which are categorized into three groups: (1) the schemes based on the **plain text file**, (2) the schemes based on the **persistent DOM (PDOM) file**, and (3) a scheme employing an **RDBMS**. We implemented all of them, and compared their performance with each other. We also compared our proposal with a storage scheme based on a current state-of-the-art XML storage scheme, showing that ours is more efficient.

Key words : XML Query Cache, Incremental Refresh of Cache, XML Update, DOM, PDOM

1. 서 론

XML이 웹 상에서 데이타 교환의 표준으로 부각된 이래 XML 데이타의 효율적인 관리에 대한 연구가 활발하게 수행되고 있다. 웹 상에서 XML 소스에 대해 자주 제기되는 질의의 결과를 캐쉬하는 것은 XML 데이타베이스 기반 웹 응용(XML database-backed Web application)의 효율적인 지원에 중요하다. 그림 1은 응용 서버에서 중간 계층 캐싱(middle-tier caching)을 제공하는 XML 데이타베이스 기반 웹 응용의 다층(multi-tier) 구조를 나타낸 것이다. 웹 페이지에 내장되어 있는 XML 질의들은 웹 서버로 전달되고 웹 서버는 응용 서버인 미들웨어 층을 경유하여 적합한 데이타 서버로 이 질의를 전달한다. 처리된 XML 질의 결과는 응용 서버에 캐쉬되고 이후 XML 소스의 변경에 대해 점진적으로 갱신된다.

질의 결과를 캐쉬하여 그 후 동일하거나 관련된 질의 처리에 재사용 하는 것은 관계 데이타 베이스 시스템에

· 본 논문은 한국과학재단 특정기초연구사업(R01-2006-000-10609-0) 지원으로 수행되었음
† 정 회 원 : 삼성전자 DM총괄
yhyun.kim@gmail.com
†† 종신회원 : 중앙대학교 컴퓨터공학부 교수
hckang@cau.ac.kr
논문접수 : 2005년 8월 10일
심사완료 : 2006년 7월 31일

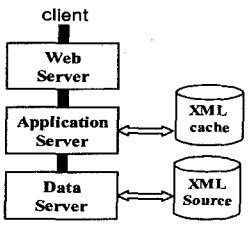

그림 1 XML 데이타베이스 기반 웹 응용의 다층구조

서 많은 관심을 받았던 중요한 질의 최적화 기술이다[1,2]. 최근 이 문제는 XML 데이타에 대해서도 연구되고 있다[3-13]. 이러한 실용적 중요성에도 불구하고 캐쉬된 질의 결과를 어떻게 저장하는 것이 효율적인지에 대한 연구는 없었다. 캐쉬된 질의 결과에 대한 두 종류의 대표적 연산은 (1) 질의 결과를 반환하기 위하여 캐쉬된 내용 **전체**를 **검색**하는 것과 (2) 캐쉬의 하부 소스에 대한 변경을 반영하여 캐쉬를 점진적으로 갱신하기 위하여 캐쉬의 **일부분**을 **변경**하는 것이다. 이들 두 종류 연산의 성능은 모두 캐쉬된 질의 결과의 저장 구조에 밀접하게 영향을 받는다. 따라서 XML 질의 캐쉬를 위한 효율적인 저장 구조를 설계하는 데 있어 이들 두 연산 간의 성능 트레이드오프를 고려하여야 한다.

웹 환경의 XML 질의 처리에 대하여 다음 두가지를 가정하자.

- W3C의 XML 질의어 XQuery 또는 XPath의 명세에는 질의 결과를 어떤 형태로 반환할 것인지에 대한 규정은 없지만, 보통 XML 질의 결과는 웹 응용을 위하여 반환될 것이기 때문에 본 논문에서는 XML 질의의 결과는 XML 문서라고 가정한다.
- 질의 결과는 텍스트 형식으로 반환된다고 가정한다. 왜냐하면 텍스트 형식이 웹 상에서 이질적이고 자율성을 갖는 응용과 시스템들 간에 데이타를 교환하는 데 있어 가장 보편적이고 단순하며 범용성을 갖기 때문이다.

상기 가정에 따라 만약 캐쉬된 XML 질의의 결과가 일반 텍스트 화일(plain text file)에 저장된다면, 문서 전체를 반환하는 것은 최적이다. 그러나 그것의 적은 일부분을 변경해야 하는 점진적 갱신에는 효율적이지 못하다. 왜냐하면 텍스트 화일은 컴팩트한 저장 구조를 갖기 때문이다. 이에 반해 캐쉬된 XML 질의 결과가 RDBMS에 저장된다면 즉, 캐쉬된 XML 질의 결과를 나타내는 XML 문서의 엘리먼트와 서브 엘리먼트들이 관계 튜플로 사상(map)되어 저장된다면, 점진적 갱신에는 효율적일 것이다. 왜냐하면 캐쉬된 XML 질의 결과인 XML 문서는 이미 파싱되어 분할되어 있기 때문이다. 반면 문서 전체를 반환하는 것은 튜플들을 검색하여 XML 태깅 과정을 거쳐야 하므로 텍스트로 저장한 경우에 비해 비효율적일 것이다. 이들 외에 XML 질의 캐쉬를 저장하기 위한 다른 대안은 어떠할까? 예를 들어, 이진(binary) 포맷으로 되어있는 영속적인 DOM(persistent DOM, 이하 PDOM)[14]을 고려해 볼 수 있다. 캐쉬된 XML 질의 결과인 XML 문서를 한번 파싱하여 PDOM 화일에 저장해 놓으면 향후 추가적인 파싱 부담 없이 DOM API를 통하여 접근 및 처리가 가능하다. 캐쉬된 XML 질의 결과를 PDOM 화일로 저장하였을 경우 그에 대한 상기 두 종류 연산의 성능이 어떠할지는 텍스트 화일이나 RDBMS에 저장한 경우에 비하여 직관적으로 이해하기는 어렵다.

본 논문에서는 XML 질의 캐쉬를 저장하는 기법을 다룬다. XML 질의 캐쉬를 저장하는 기법에 대한 요건은 다음 두가지로 요약할 수 있다:

- 캐쉬로부터 웹 응용이 원하는 형식으로 XML 질의 결과를 반환하는 과정이 효율적으로 수행될 수 있어야 한다.
- 보통 캐쉬의 아주 작은 일부분의 변경을 요하는 캐쉬의 점진적 갱신 과정도 역시 효율적으로 수행될 수 있어야 한다.

위에서 일반 텍스트 화일 또는 RDBMS에 질의 캐쉬를 저장하는 경우에 대해 언급한 것처럼 이들 두 요건은 서로 상충 관계에 있다. 따라서 이들 두 요건 간에 적절한 절충을 이루는 저장 기법을 고안하는 것이 필요하며 이를 위해서는 해당 두 연산 간의 성능 트레이드오프를 면밀히 검토하는 것이 필요하다.

본 논문에서는 모두 여덟 개의 XML 질의 캐쉬 저장 기법을 제시한다. 이들은 크게 세개의 그룹으로 나누어지는데, (1) 일반적인 텍스트 화일을 기반으로 한 기법 (2) PDOM 화일에 기반을 둔 기법 (3) RDBMS를 사용하는 기법이 그들이다. 이들 모두를 구현하여 기법 간에 비교 평가하였다. 또한 기존의 XML 저장 기술에 기반을 둔 질의 캐쉬 저장 기법과도 비교 평가하였다.

본 논문의 구성은 다음과 같다. 2절에서는 관련 연구를 기술하고 본 논문과의 차이점을 언급한다. 3절에서는 본 논문의 XML 질의 캐쉬 모델과 그것의 점진적 갱신 모델을 기술한다. 4절에서는 상기 여덟 개의 저장 기법을 설명하고, 각각에 대해 상기 두 가지 연산이 어떻게 수행되는지 기술한다. 5절에서는 이들의 구현과 성능 평가 결과를 기술한다. 마지막으로 6절에서는 결론을 맺는다.

2. 관련 연구

XML 질의 결과를 캐쉬해 두었다가 동일한 질의 또는 관련된 질의의 처리에 사용함으로써 성능 향상을 꾀하기 위한 기법들이 최근 들어 주목을 받고 있다[3-13].

이들 연구는 관계 데이타베이스에서의 질의 캐쉬 또는 실체뷰(materialized view) 연구에서 그랬던 것처럼 두 계열로 대별할 수 있다. 하나는 소스의 변경 중 캐쉬와 관련된 것을 찾아 캐쉬에 점진적으로 반영하는 기법[3-6]이고 다른 하나는 캐쉬와 관련된 질의를 캐쉬를 이용한 질의로 재작성하는 기법[7-13]이다. 그러나 이들 연구들에서는 캐쉬된 질의 결과를 어떻게 저장하는 것이 효율적인가 하는 문제는 다루지 않았다. [3,4]에서는 캐쉬 일관성 문제를 다루면서 캐쉬의 저장 구조에 대해서는 언급하지 않았다. [5,8]에서는 캐쉬를 메모리에 상주하는 트리로 표현하는 것을 고려하였다. 웹에서 XML 질의 캐쉬를 메모리에 표현하는 것은 확장성(scalability)을 보장할 수 없는 방법이다. 뿐만 아니라 상기 두 계열의 연구 중 후자인 [7-13]의 연구에서는 캐쉬의 갱신 기능은 고려하지 않았다. 즉, XML 소스는 정적이라고 가정하였다. 따라서 캐쉬된 질의 결과의 전체를 반환하거나 아주 작은 일부를 변경하는 서로 상충된 두 종류의 연산을 적절히 절충할 수 있는 저장 구조에 대한 고려는 필요 없다. 이들 연구 중 [9]에서는 XPath 질의를 관련 캐쉬를 사용하여 재작성하여 처리하기 위한 형식 모델(formal model)을 제안하였는데, 그것이 구현되기 위해서는 캐쉬의 저장 구조가 제공되어야 한다는 점을 언급하고 있다. [6]에서는 XPath로 표현된 XML 질의의 결과를 소스 XML 트리의 노드 식별자 집합으로 캐쉬한다고 가정하였다. XML 소스에 변경이 발생했을 때 캐쉬된 노드 식별자 집합을 점진적으로 갱신하는 기법을 제안하였다. 캐쉬된 질의가 다시 제기되면 노드 식별자로 해당 노드 즉, XML 엘리먼트를 소스로부터 검색하여 질의 결과를 구성한다. 이 기법은 질의 캐쉬로서 식별자 집합을 유지하므로 질의 결과를 실체뷰로 캐쉬하는 것은 아니다. 따라서 본 논문에서 다루는 캐쉬 저장 구조의 효율성이 이슈가 되지는 않으나 질의 결과를 반환할 때는 소스의 검색을 요하므로 그 부담이 크다. 본 논문에서는 이상 언급된 기존 관련 연구와는 달리 디스크 기반의 다양한 캐쉬 저장 구조를 제시하고 장단점을 비교, 평가하였다.

XML 저장 기법이 여럿 제안됨에 따라 그들을 비교 평가하기 위하여 XMark[15], XBench[16] 등의 XML 벤치마크가 개발되었다. 이들은 실세계의 혹은 인위적으로 생성한(synthetic) XML 데이타에 대한 다양한 질의 처리의 성능 평가를 통해 여러 XML 저장 체계들을 비교, 평가하기 위한 것이다. 본 논문에서도 XML 질의 결과인 XML 데이타의 저장 기법들을 비교하고 있으나, XML 질의 캐쉬의 점진적 갱신을 위해 아주 작은 일부분을 변경하거나 캐쉬된 질의 결과를 반환하기 위하여 캐쉬 전체를 검색하는 성능 면에서 상충되는 두 연산에 초점을 두고 두 연산 간에 적절한 절충을 보이는 XML 질의 캐쉬 저장 기법을 찾고자 하였다.

3. XML 질의 캐쉬 및 점진적 갱신 모델

그림 1의 다층 구조에서 데이타 서버에 저장된 XML 문서 집합을 대상으로 XML 질의가 제기되면, 그것의 중요도와 빈도에 따라 그 결과를 응용 서버에 캐쉬할 수 있다. 데이타 서버로부터 응용 서버로 캐쉬된 질의 결과 XML 문서의 형식은 그림 2와 같다. 루트 엘리먼트는 'cache'이고, 그 아래 n개의 qdocument 서브 엘리먼트로 구성된다. 여기서 n 이란 질의 조건을 만족하는 XML 소스 문서의 수이다. 즉, 하나의 qdocument 서브 엘리먼트는 질의 조건을 만족하는 하나의 XML 소스 문서로부터 생성된다. qdocument 서브 엘리먼트의 형식은 XML 질의에 명시되어 있다. (예를 들면, XQuery의 RETURN 절에 명시된 형식을 따를 수 있다.) qdocument의 DID 속성은 해당 XML 소스 문서의 식별자를 저장한다. 이 DID 값을 캐쉬된 질의 결과와 함께 유지해야 하는 이유는 데이타 서버에 있는 XML 문서가 변경되어 이를 캐쉬에 반영하고자 할 때 그에 상응되는 대상 qdocument 서브 엘리먼트를 식별하기 위해서이다. qdocument 서브 엘리먼트는 그것의 DID 속성과 더불어 XML 소스와 XML 질의 캐쉬 간에 사상(mapping) 정보를 제공하는 역할 외에 서로 다른 XML 소스 문서로부터 검색된 부분 질의 결과들 간의 구분자 역할도 한다.

```
<cache>
  <qdocument DID="1">
   ...
  </qdocument>
  <qdocument DID="5">
   ...
  </qdocument>
....
</cache>
```

그림 2 캐쉬 XML 문서의 양식

만약 부분 질의 결과들 간에 그 소스 문서에 대한 구분이 필요 없다면 qdocument의 start/ends 태그와 DID 속성은 캐쉬된 질의 결과를 질의를 제기한 클라이언트(예를 들어, 웹 응용)에게 반환될 때 제거하면 된다.[1] 따라서, XML 질의에 대해 데이타 서버에서 응용 서버

1) 상용 XML 데이타베이스 시스템인 eXcelon에서는 XML 문서 집합에 대해 제기된 XML 질의 결과의 디폴트 형식으로 각 소스 문서로부터 검색된 부분 질의 결과들을 구분하지 않는 것을 사용한다

로 반환되는 XML 문서는 일반적으로 응용 서버에서 웹 서버를 통하여 클라이언트에게 최종 반환되는 XML 문서와 완전히 동일하지 않을 수 있다. 이 둘간의 구분을 위하여, 전자를 **캐쉬(XML) 문서**, 후자를 **결과(XML) 문서**라고 하자.

한편, XML 소스 문서가 데이타 서버에서 변경되었을 때 응용 서버에 있는 관련된 캐쉬 XML 문서는 일관성 유지를 위해 점진적으로 갱신된다. 이를 위한 핵심 알고리즘은 소스에 대한 XML 변경과 캐쉬된 질의 결과 간의 연관성을 검사하는 것과, 캐쉬에 대한 일련의 XML 변경 연산 및 캐쉬 갱신에 필요한 데이타를 검색하기 위한 소스에 대한 질의로 구성된 캐쉬 갱신 정보를 생성하는 것이다. 이들 알고리즘은 본 논문의 범위에서 벗어난다([3-6] 등의 연구를 참조). 본 논문의 초점은 캐쉬 문서의 저장 기법으로서 상기 알고리즘들에 의해 생성된 캐쉬에 대한 일련의 XML 변경 연산이 캐쉬 문서에 대해 효율적으로 수행되고 또한 그러한 변경의 누적에도 불구하고 결과 문서를 제공하는 과정의 효율성도 담보되는 저장 구조에 있다.

XML 질의 캐쉬의 점진적 갱신은 일련의 XML 변경 연산에 의해 이루어진다. 본 논문에서는 엘리먼트 삽입과 엘리먼트 삭제의 두가지를 고려한다. 이들 두 종류의 기본 XML 변경 연산으로는 임의의 복잡한 XML 변경 연산을 구성할 수 있다. XML 변경 연산을 명시하는 정보로는 다음을 들 수 있다: (1) 변경된 XML 소스 문서의 식별자(즉, DID 속성의 값), (2) 해당 qdocument 엘리먼트 내에서 변경 연산을 적용할 목적 엘리먼트 및 그것이 만족해야 하는 조건을 포현하는 경로 표현식, (3) 변경 연산이 필요로 하는 데이타(예를 들면, 엘리먼트 삽입의 경우 삽입될 엘리먼트).

4. XML 캐쉬 문서의 저장 기법

본 절에서는 여덟 개의 XML 질의 캐쉬 저장 기법을 제시한다. 이들은 크게 세개의 그룹으로 나누어지는데, (1) 일반적인 텍스트 화일을 기반으로 하는 기법 (2) PDOM 화일에 기반으로 하는 기법 (3) RDBMS를 사용하는 기법이다. 표 1은 이들 기법들을 요약한 것이다.

4.1 일반 텍스트 화일에 기반을 둔 기법

본 절에서는 일반 텍스트 화일에 XML 캐쉬 문서를 저장하는 기법들을 기술한다.

4.1.1 기법 TD(Text/DOM)

가장 단순하고 기본적인(baseline) 기법으로 그림 2의 형식으로 된 캐쉬 XML 문서를 일반 텍스트 화일에 저장하고, 질의 결과의 반환 및 캐쉬 갱신은 DOM 파서를 사용하여 메모리에 적재된 DOM 트리를 DOM API를 통하여 처리한다. 점진적 갱신을 위한 변경 연산은 아래

표 1 본 논문이 제시하는 XML 질의 캐쉬 저장 기법

기법	화일 혹은 데이타베이스	저장 구조
TD	text file	text file
IT	text file	text file + index
ILT	text file	log-structured text file+ index
PD	PDOM file	PDOM file
IPD	PDOM file	PDOM file + index
PDF	PDOM file	a set of XML fragment PDOM files
PDTF	PDOM file	a set of XML fragment PDOM files a set of XML fragment text files
RDB	RDBMS	XRel [17] w/ update robust XML numbering scheme

와 같이 이루어진다. (1) DOM 트리가 생성되고 메모리에 적재된다. (2) 적절한 DOM API 함수가 호출되어 목적 엘리먼트를 찾아 요구된 변경 연산을 실행한다. (3) 마지막으로, 변경된 DOM 트리는 일반 텍스트 화일로 다시 저장된다. 질의 결과를 반환할 때 결과 XML 문서는 아래와 같이 생성된다. (1) DOM 트리가 생성되고 메모리에 적재된다. (2) 적절한 DOM API 함수가 호출되어 qdocument 엘리먼트의 start/end 태그와 DID 속성을 모두 삭제한다. (3) 이와 같이 변경된 DOM 트리를 텍스트로 변환한다.

이 기본적인 기법에는 많은 문제가 있다. 보통 DOM 파서는 XML 문서에 대한 DOM 트리 전체를 동시에 메모리에 적재한다. 물론 일부 제품들에서는 전체 DOM 트리의 일부분만 적재시킬 수도 있지만, 전자 유형의 DOM 파서로는 사용 가능한 메모리의 용량에 따라 캐쉬 문서의 크기에 제약이 있을 수 있다. 또 다른 문제는 캐쉬 문서를 변경하는 작업과, 결과 문서를 생성하는 과정의 비효율성이다. 전자의 경우, 문서 전체를 파싱해야 하고, 변경되어야 할 목적 엘리먼트를 찾기 위하여 DOM 트리에서 평균적으로 절반 가량의 qdocument 엘리먼트 노드를 탐색해야 한다. 또한 변경된 DOM 트리가 다시 저장될 때 텍스트 화일 전체가 다시 쓰여져야 하는데 점진적 갱신을 위해서 보통 캐쉬 문서 중 단지 작은 일부분만 변경되면 된다는 점을 감안할 때 아주 비효율적이다. 후자의 경우, qdocument 엘리먼트의 start/end 태그와 DID 속성을 삭제한다는 것은 DOM 트리에서 대규모의 구조 변경을 초래한다. 이들 문제는 다음 절에서 설명할 향상된 기법에 의해 부분적으로 해결될 수 있다.

4.1.2 기법 IT(Indexed Text)

이 기법에서는 그림 2와 같은 형식의 캐쉬 문서가 파싱된 후 DID 속성 값에 대해 색인된 결과 문서를 만들어 텍스트 화일로 저장한다. 그 구조는 그림 3에 나타낸 것과 같이 두개의 화일 Xt와 i로 이루어져 있다. Xt 화

일은 결과 문서를 저장하고 있는 일반 텍스트 화일이다. i 화일은 Xt 화일의 DID에 대한 색인이다. Xt 화일에 있는 각각의 qdocument 엘리먼트마다, i 화일에 하나의 엔트리가 존재하는데, DID, Start_Offset, Element_Length의 세가지 값으로 구성된다. Start_Offset은 Xt 화일 시작부터 자신의 <qdocument> 태그까지의 바이트 오프셋이다. Element_Length는 해당 qdocument 엘리먼트의 바이트 길이이다. i 화일의 엔트리들은 DID 값으로 정렬되어 있다. 만약 i 화일의 크기가 디스크 블록의 크기보다 커짐에 따라 원하는 엔트리의 검색 부담이 생기면, DID 값에 대한 B+ 트리를 화일 i의 상단에 추가하여 해결할 수 있다.

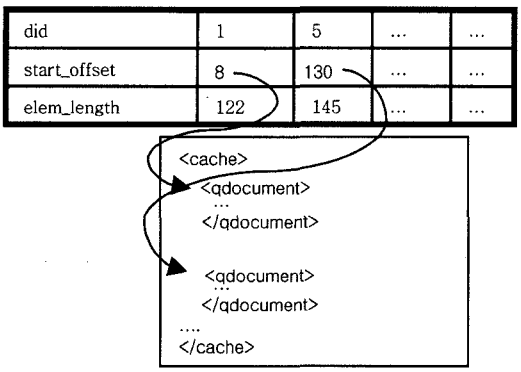

그림 3 기법 IT의 저장 구조

Xt 화일은 일반 텍스트 문서로 된 결과 문서 그 자체이기 때문에 이 기법에서 질의 결과를 반환하는 것은 원천적으로 최적이며, 기법 TD에서의 가용 메모리 용량에 따른 문제도 해결된다. 그러나 점진적 갱신의 효율성은 보증하지 못한다. 캐쉬의 점진적 갱신을 위한 Xt 화일의 변경은 아래와 같이 이루어 진다. (1) 주어진 DID 값과 동일한 DID 값의 인덱스 엔트리를 화일 i에서 검색한다. (2) Xt 화일의 변경될 대상 qdocument의 텍스트가 Start_Offset과 Element_Length 값에 의해 추출된다. (3) 그것의 DOM 트리가 만들어지고 메모리에 적재 된다. (4) 적절한 DOM API 함수가 호출되어 요청된 변경 연산을 수행한다. (5) 변경된 DOM 트리가 텍스트로 변환되고 변경 전 Xt 화일의 해당 qdocument 엘리먼트의 텍스트를 대체한다(즉, in-place update). 이때 만약 변경 전후의 텍스트 길이 간에 변화가 있으면 변경된 텍스트 이후의 qdocument 엘리먼트들의 텍스트를 모두 먼저 읽어 두었다가 변경된 qdocument 엘리먼트의 텍스트를 쓴 후 그 뒤에 모두 써준다. (6) Xt 화일의 변경에 영향을 받는 i 화일의 모든 엔트리를 변경해 준다.

이 기법의 문제는 단계 (5)와 (6)에 있다. 이들 단계에서는 Xt 화일의 일부분을 변경함에도 불구하고 평균적으로 Xt 화일과 i 화일의 약 절반 정도를 변경하게 된다. 다음 절의 기법이 이 문제를 해결한다.

4.1.3 기법 ILT(Indexed Log-Structured Text)

기법 IT 의 문제를 해결하기 위해서 Xt 화일을 로그 구조 화일[18]로 구현할 수 있다. 로구 구조 화일에서는 변경되는 데이타를 변경 전 데이타가 위치하던 곳에 덮어 쓰지(in-place update) 않고 화일의 맨 뒤에 첨부(append)함으로써 변경 전 내용도 그대로 남기게 된다. 기법 ILT에서는 이 방법을 사용한다. 즉, 변경된 qdocument의 텍스트를 Xt 화일로 다시 쓸 때 이전 버전의 텍스트를 대체하지 않고 기존 Xt 화일의 끝에 첨부하고 그에 따라 해당 i 화일 엔트리만 변경한다.

기법 ILT를 사용하면 요청된 XML 변경을 처리하는데 걸리는 시간은 캐쉬 문서의 크기에 독립적이게 된다. 그러나 질의 결과를 반환하기 위해서는 i 화일의 엔트리들을 통해 Xt 화일의 최신 텍스트 조각(fragment)을 모두 접근하여 결과 문서를 만들어야 한다. 이 과정은, i 화일 엔트리들의 전처리를 통하여 인접한 텍스트 조각들을 하나의 조각으로 병합 즉, 해당 엔트리들을 하나의 엔트리로 병합함으로써 화일 Xt에 대한 디스크 입출력 횟수를 줄이는 것을 통해 효율적으로 수행할 수 있다. 또 다른 문제는 변경 연산이 누적되면서 Xt 화일의 크기가 증가하는 것을 생각할 수 있는데, 이는 질의 결과의 반환을 위하여 결과 문서를 생성하였을 때 그것으로 기존 화일 Xt를 대체하면 해소된다. 질의 결과 반환의 요청은 없이 변경만 누적되어 Xt 화일의 크기가 너무 커지는 것을 방지하기 위해서는 결과 문서 생성을 통한 Xt 화일의 재구성을 수행하면 된다. 이러한 재구성은 주기적으로 실행되거나 별도의 요구에 의해 실행될 수 있다.

4.2 PDOM 화일에 기반을 둔 기법

본 절에서는 PDOM 화일로 XML 캐쉬 문서를 저장하는 기법들을 기술한다.

4.2.1 기법 PD(PDOM)

PDOM 화일에 기반을 둔 기본적인 기법은 그림 2의 형식으로 된 캐쉬 문서를 파싱하여 PDOM 화일로 저장하는 것이다. PDOM 화일은 DOM 트리를 이진 포맷으로 저장하기 때문에 캐쉬 문서의 변경이 필요할 때 파싱을 다시 하는 것이 아니라 메모리에 DOM 트리를 바로 적재하면 된다. PDOM의 API도 기본적으로 DOM API 이기 때문에 캐쉬 문서를 변경하는 것과 질의 결과를 반환하는 절차는 DOM 트리를 메모리에 적재하는 과정을 제외하면 기법 TD의 그것과 유사하다.

이 기법에서는 파싱 부담이 없고 PDOM의 구현에 따

라 가용 메모리 용량에 따른 TD 기법의 제약도 극복할 수 있다. 왜냐하면 DOM 트리 전체가 메모리에 적재되는 대신 DOM 트리의 일부분만 메모리에 적재되고 처리될 수 있기 때문이다[19]. PDOM 화일은 이진 포맷으로 되어 있기 때문에 DOM 트리의 일부분이 변경되었을 때 해당 비트 세그먼트의 길이가 바뀌게 되는 것이 보통이므로 원래의 비트 세그먼트 자리에 새 내용을 덮어쓰는 변경(in-place update)은 적절하지 못하다. [19]의 PDOM 구현 예를 보면 변경된 비트 세그먼트는 PDOM 화일의 끝에 첨부된다. 따라서 기법 PD는 보통 캐쉬 문서의 일부분만 변경하게 되는 점진적 갱신에 매우 효율적이다.

그러나 질의 결과를 반환할 때, DOM 트리 상에서 qdocuemnt 엘리먼트의 start/end 태그와DID 속성을 모두 삭제하는 것은 DOM 트리의 심한 구조적 변화를 초래하게 되는데, 이진 포맷으로 된 PDOM 화일은 이와 같은 광범위한 변경 연산의 효율적인 지원에는 적합하지 않다. 이 문제는 IT 기법에서와 비슷한 방법으로 해결될 수 있다.

캐쉬 문서를 PDOM 화일에 저장하게 되면 텍스트 화일 기반의 기법 TD, IT, 그리고 ILT에 비해 공간 부담이 적다. 왜냐하면 이진 포맷으로 이루어진 PDOM 화일은 매우 컴팩트한 구조로 되어있기 때문이다. 그러나 변경이 누적되면 PDOM 화일의 크기가 ILT 기법의 경우에서 처럼 점차 커지게 된다. 이와 같이 커진 PDOM 화일의 재구성은 간단하다. DOM 트리 전체를 메모리에 적재한 후, 텍스트로 변환시키고, 그것을 다시 파싱하여 PDOM 화일로 저장하면 된다.

4.2.2 기법 IPD(Indexed PDOM)

이 기법의 아이디어는 IT 기법의 그것과 비슷하다. 그림 2의 형식으로 된 캐쉬 문서는 파싱되어 두 개의 화일 Xp 와 i 로 변환되어 저장된다. Xp 화일은 결과 문서의 PDOM 형식 화일이고 i 화일은 Xp 화일의 DID 값에 대한 색인이다. Xp 화일이 이진 포맷으로 되어 있기 때문에 qdocument 엘리먼트를 가리킬 때 IT 기법에서 처럼 바이트 오프셋 또는 비트 오프셋을 사용하는 것은 불가능하다. 색인 화일 i의 각 엔트리는 DID 및 OL 두 값으로 구성된다. OL(ordinal location)은 가리키는 qdocument 엘리먼트에 대한 서브 DOM 트리가 루트의 몇번째 자식인지 그 순서 또는 위치를 나타내는 값이다. 즉, 모든 qdocument 엘리먼트는 캐쉬 문서 루트 엘리먼트의 (즉, 그림 2의 cache 엘리먼트) 자식 노드인데 첫 번째 qdocument 노드의 경우의 i 화일의 해당 엔트리의 OL 값은 1이다. 2번째 qdocument 노드의 OL 값은 2, 세번째는 3 등이다. i 화일의 엔트리는 DID 값에 의해 정렬된다. 만약 i 화일의 크기가 디스크 블록의 크기보다 커지는 경우에는 기법 IT에서처럼 DID 값에 대한 B+ 트리를 이용할 수 있다.

Xp 화일이 결과 문서의 PDOM 화일이기 때문에 질의 결과를 반환하는 과정은 그것을 텍스트로 변환하는 과정만 요한다. 점진적 갱신을 위하여 Xp 화일을 변경할 때는, IT 기법에서와 마찬가지로 i 화일의 해당 엔트리를 우선 검색한다. 그리고 그것의 OL 필드를 보고 DOM 트리의 해당 qdocument 서브트리만을 메모리에 적재한다. 그 이후의 과정은 PD 기법에서와 같다. i 화일은 qdocument 엘리먼트 노드의 위치 정보를 저장하고 있기 때문에 새로운 qdocument 엘리먼트가 삽입되거나 존재하는 qdocument 엘리먼트가 삭제되지 않는한 바뀌지 않는다. 이러한 삽입 또는 삭제가 발생하면 그로 인해 영향을 받는 엔트리의 OL 필드를 모두 1씩 증가시키거나 1씩 감소시키면 된다. PD 기법과 마찬가지로 변경 연산이 누적 됨에 따라 Xp 화일의 크기가 증가하게 된다. 따라서, PD 기법에서와 비슷하게 Xp 화일의 재구성이 필요하다.

4.2.3 기법 PDF(PDOM Fragments)

이 기법에서는 그림 2의 형식으로 된 캐쉬 문서를 파싱한 후 각각의 qdocument 엘리먼트를 별도의 PDOM 화일로 변환하여 저장한다. 이들 각각의 PDOM 화일은, 해당 qdocument 엘리먼트에서 start/end 태그와 DID 속성을 제거한 후 더미(dummy) 루트 엘리먼트로 qdocument 엘리먼트의 서브 엘리먼트들을 묶은 XML 조각(fragment)을 PDOM 화일로 변환한 것이다.

본 논문에서는 이 분리된 화일을 PDOM 조각(화일)(PDOM fragement(file))이라고 부른다. 만약 캐쉬 문서에 n 개의 qdocument 엘리먼트가 있다면 n 개의 PDOM 조각 화일이 생성된다. DID 값이 주어졌을 때 그것의 해당 qdocument 엘리먼트를 저장하고 있는 PDOM 조각을 찾아내는 방법은 여러 가지가 있다. DID 값을 PDOM 조각 화일 이름의 일부분으로 사용하거나 그러한 사상을 나타내는 화일 디렉토리를 사용할 수 있다.

PDOM 조각이 변경될 때는 메모리의 변경된 DOM 트리를 텍스트로 변환한 후 그것을 다시 PDOM 화일로 변환하여 저장한다. 이런 방식으로 PD 및 IPD 기법에서 변경 연산의 누적에 따라 화일의 크기가 증가하는 문제를 해결할 수 있는데, qdocument 엘리먼트의 크기가 아주 크지 않는 한 이러한 방식은 적절하다. 결론적으로 이 기법은 qdocument 엘리먼트의 변경에 매우 효율적이다. 변경에 걸리는 시간은 캐쉬 문서에 크기에 전혀 좌우되지 않는다. 그러나 질의 결과를 반환할 때는, 모든 PDOM 조각 화일이 접근되어야 하고 각각에 대해 더미 루트가 제외된 부분을 텍스트로 변환한 것을 병합

하여 결과 문서를 생성해야 한다. 즉, n개의 화일들이 처리되어야 하는데, n의 크기 및 화일을 열고 닫는 등의 화일 처리 부담에 따라 큰 비용을 초래할 수 있다.

4.2.4 기법 PDTF(PDOM/Text Fragments)

이 기법은 PDF 기법의 변종이다. PDF 기법의 각 PDOM 조각 화일에 대해 그것을 텍스트로 변환한 텍스트 화일(이때 더미 루트 엘리먼트는 제외됨)을 함께 저장한다. 이 텍스트 화일은 텍스트 조각(화일)(text fragment(file))이라고 부른다. 만약 캐쉬 문서에 n 개의 qdocument 엘리먼트가 있다면 모두 2n 개의 화일이 생성된다(n개의 PDOM 조각 화일과 n개의 텍스트 조각 화일). PDF 기법에서 PDOM 조각 화일이 변경될 때 해당 DOM 트리의 텍스트 변환이 어차피 일어나는데 PDTF 기법에서는 그 점을 활용할 수 있다. PDTF 기법은 PDF 기법에서 질의 결과를 반환할 때 PDOM 조각 화일들을 텍스트로 변환하는 시간을 절약함으로써 성능을 향상시킨다.

4.3 RDBMS를 사용하는 기법

RDBMS가 널리 사용되어 왔기 때문에, RDBMS에 XML 문서를 저장하고 질의 처리를 수행하는 기법은 실용적으로 그 중요성이 몹시 크고 따라서 많이 연구되었다. 본 논문의 캐쉬 XML 문서 역시 RDBMS에 저장될 수 있다. 본 절에서는 XML 질의 캐쉬를 RDBMS에 저장하는 기법을 기술한다.

4.3.1 관계형 테이블 스키마와 변경에 강건한 XML 번호 체계

많은 연구들이 XML-관계 사상(XML to relational mapping)을 제안하였다. 본 논문에서는 XML 캐쉬 문서를 저장하기 위한 관계 테이블 스키마의 설계를 XRel [17]을 기반으로 하였다. XRel은 DTD와 같은 스키마 정보가 없는 XML 문서도 저장할 수 있다. 따라서 다양한 구조를 갖는 캐쉬 문서가 저장되고 삭제됨에 따라 테이블 스키마를 변경해 줄 필요가 없다. 대부분의 XML-관계 사상과 마찬가지로 XRel은 단지 효율적인 XML 질의 처리의 지원만을 염두에 두고 고안된 XML 저장 체계이다. 본 논문에서는 캐쉬 문서의 점진적 갱신을 위한 변경이 지원되어야 하므로 변경도 지원하도록 XRel을 확장하는 것이 필요하다.

XML 질의 처리 기법들에서는 각각의 XML 엘리먼트에(그것이 XML 문서의 단말 노드이든 아니든) 엘리먼트 식별자(EID)라고 부르는 고유의 번호를 부여하는 XML 번호 체계가 매우 중요한 역할을 한다. XML 문서는 보통 노드에 레이블을 부여하는 순서가 있는 트리로 모델링되며, 이때 XML 엘리먼트(단말 이든 아니든)가 트리의 노드에 해당된다. XRel의 번호 체계를 포함하여 가장 널리 사용되고 있는 XML 번호 체계는 XML 트리를 루트부터 단말 노드까지 preorder로 탐색하면서 매번 증가되는 번호를 부여하는 것이다.

엘리먼트 삽입과 삭제 같은 XML 변경은 XML 트리의 구조 변화를 초래하므로 XML 트리를 preoder로 탐색하면서 부여했던 EID의 일부 또는 경우에 따라서 대부분을 변경해 주어야 하는 것이 필요하다. 이는 경우에 따라 아주 큰 부담을 초래하게 된다. 따라서 XML 변경에도 불구하고 기존에 부여된 EID를 일체 변경하지 않거나 극히 일부만 변경하면 되는 번호 체계가 바람직한데 그러한 번호체계를 변경에 강건하다고(update robust) 한다.

XRel에서는 XML 문서에 대하여 PATH, TEXT, ATTRIBUTE 그리고 ELEMENT 라고 이름 붙인 4개의 테이블이 각각 Path 정보, 텍스트 값(PCDATA), 속성, 그리고 엘리먼트에 대한 기타 정보를 저장한다. 그림 4는 본 논문에서 상기 XRel의 4개 테이블 스키마를 변경에 강건한 번호 체계를 사용하여 확장한 스키마를 나타낸 것이다. 이 스키마를 구성하는 XML-관계 사상과 변경에 강건한 XML 번호 체계는 본 논문의 범위를 벗어나므로 기술하지 않는다.

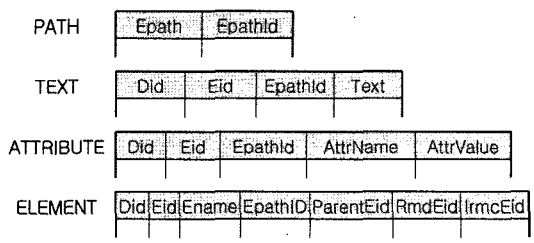

그림 4 기법 RDB의 테이블 스키마

4.3.2 기법 RDB

본 절에서는 캐쉬 XML 문서를 RDBMS에 저장하는 기법 RDB를 기술한다. 캐쉬 문서의 점진적 갱신을 위한 변경은 그림 4의 4개 테이블에 대한 일련의 SQL 문으로 변환되어(즉, INSERT, DELETE, UPDATE 문) 수행된다.

질의 결과를 반환하기 위해서는, 캐쉬 문서를 저장하고 있는 TEXT, ATTRIBUTE, 그리고 ELEMENT 테이블의 튜플들을 검색하여 [20]에서 제안된 outer-union 연산에 의해 하나의 결과 셋으로 병합하는 SQL 문을 수행한 후 해당 결과 셋에 대한 XML 태깅을 수행하여 결과 문서를 생성한다. 이때, qdocument 엘리먼트와 그것의 DID 속성은 배제된다.

5. 구현 및 성능 평가

본 논문에서는 전 절에서 기술한 여덟 개의 기법을

Windows 2000 Server 상에서 Java로 모두 구현하였다. 일반 텍스트 화일에 기반을 둔 기법들에서 DOM 파서로는 Apache Xerces Java Parser 1.4.3. Release [21]가 사용되었다. PDOM 화일에 기반을 둔 기법들을 위해서는GMD-IPSI XQL Engine Version 1.0.2[19]의 PDOM 구현을 사용하였다. RDBMS를 사용하는 기법을 위해서는 Oracle 9*i*를 JDBC를 통해 사용하였다. 실험을 통해서 각각의 기법마다 캐쉬 문서를 변경하는 작업과 질의 결과를 반환하는 작업에 대한 성능을 평가하였다. 또한 캐쉬 문서를 저장하는 시간 및 요구되는 저장 공간 부담도 측정하였다.

TPC-W benchmark[22]의 order XML 문서를 실험에서 XML 소스 데이타로 사용하였다. 많은 수의 이들 문서가 전자 상거래와 같은 웹 응용을 지원하기 위하여 XML 웨어하우스 에 저장되어 있다고 가정하였다. order 문서는 고객, 날짜, 지불, 주문 상품 등의 주문에 대한 정보를 저장하고 있다. 실험에 사용된 캐쉬 XML 문서들은 다음과 같이 구하였다. 우선 8,000 개의 order 문서를 XML benchmark XBench의 XML 자동 생성기인 ToXgene[23]을 사용하여 생성하였다. 그들을 XML 전용 데이타베이스 시스템인 X-Hive에 저장한 후(이때 각 order 문서의 식별자로는 order 문서의 루트 엘리먼트인 order 엘리먼트의 ID 속성 값을 사용), 이들 order 문서들에 대하여 모든 order/order_lines 엘리먼트를 검색하는 XQuery 질의를 제기하여 그림 2의 형식으로 된 XML 문서를 그 결과로 얻었다. (즉, 각 order 문서로부터 검색된 order_lines 엘리먼트가 qdocument 엘리먼트의 서브 엘리먼트가 된다.) 이와 같이 하여 모두 8개의 캐쉬 문서를 생성했는데 각각의 qdocument 엘리먼트의 수는 1,000에서 8,000으로 문서마다 1,000씩 차이가 나게 하였다. 1,000개의 qdocument 엘리먼트로 구성된 가장 작은 캐쉬 문서의 크기는 공백 문자를 제외하고 약 736KB이고, 8,000개의 qdocument 엘리먼트로 구성된 가장 큰 것은 약 6MB이다. 본 실험은 Pentium IV 1.8GHz CPU와 256MB 메모리가 장착된 시스템에서 수행되었다.

5.1 캐쉬 XML 문서의 점진적 갱신을 위한 변경

그림 5와 그림 6은 캐쉬 문서의 크기가 증가함에 따라 ILT, PD, IPD, PDF, PDTF, 그리고 RDB 기법이 qdocument 엘리먼트 내에서 하나의 단말 엘리먼트의 삽입과 단말 엘리먼트의 삭제를 처리하는 데 걸린 시간을 각각 비교하여 나타낸 것이다. TD 기법과 IT 기법은 경쟁력이 없는 것으로 나타나 표시하지 않았다.

TD 기법과 IT 기법의 성능이 아주 나쁘게 나타난 이유는 문서의 전체가 아닌 일부분이 변경되는데도 불구하고 캐쉬 문서 전체를 처리해야 하기 때문이다. 반면에

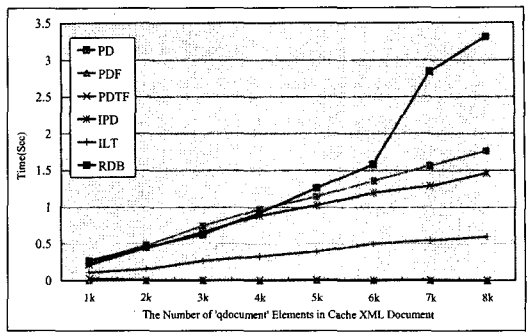

그림 5 엘리먼트 삽입 처리 시간

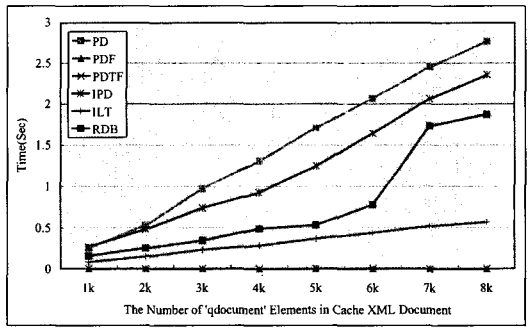

그림 6 엘리먼트 삭제 처리 시간

다른 기법들은 캐쉬 문서 중 변경된 부분만 처리되기 때문에 비교적 좋은 성능을 보였다.

PDF 기법과 PDTF 기법은 거의 유사한 성능을 보였으며 예상대로 다른 기법들과 비교하여 가장 효율적이었다. PDF가 최고의 성능을 보였으며 PDTF 보다는 약간 더 좋은 성능을 보였다. 이는 PDTF에서 PDOM 조각에 대한 텍스트 조각을 유지하는 부담이 거의 무시할 정도인 것을 의미한다. PDF 및 PDTF 기법을 제외한 다른 기법들은 모두 캐쉬 문서의 크기가 증가함에 따라 성능이 저하되었지만, PDF 및 PDTF 기법의 성능은 캐쉬 문서의 크기에 영향을 받지 않았다. 기법 IPD는 PD에 비해 성능이 개선된 것으로 나타났다. 이는 IPD의 경우 색인(즉, i 화일)을 통하여 변경이 요구되는 qdocument 엘리먼트의 해당 서브 트리를 바로 검색하여 처리할 수 있기 때문이다. 기법 RDB의 성능은 캐쉬 문서의 크기가 qdocument 엘리먼트의 수가 6,000개일 때 까지는 캐쉬 문서의 크기 증가에 따라 완만한 성능 저하를 보였으나 그보다 커지면서 급격히 저하되는 것으로 나타났다.

5.2 XML 질의 결과 반환

그림 7은 캐쉬된 XML 질의 결과를 반환하기 위해서 캐쉬 문서로부터 텍스트로 된 결과 문서를 생성하는 데 걸린 시간을 기법 TD, IT, ILT, IPD, PDTF에 대해

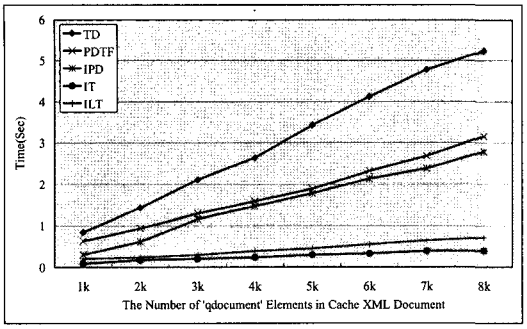

그림 7 질의 결과 반환 시간

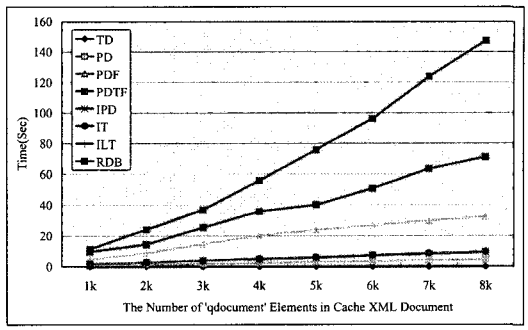

그림 8 캐쉬 문서의 저장 시간

캐쉬 문서의 크기가 증가함에 따라 나타낸 것이다. 기법 PD, RDB, PDF는 성능이 현저히 떨어져 나타내지 않았다.

PD 기법의 성능이 좋지 않은 이유는 전체 DOM 트리에 걸쳐 qdocument 엘리먼트의 start/end 태그와 DID 속성을 삭제하는 부담 때문이다. 이진 포맷으로 되어 있는 PDOM은 그러한 방대한 규모의 변경에 적합하지 않다. RDB 기법의 성능이 좋지 않은 것은 XML로 된 결과 문서를 생성하기 위하여 관계 튜플로 분할 저장된 엘리먼트들을 테이블로부터 검색하여 XML 태깅하는 과정이 필요하기 때문이다. 이는 XML 문서를 RDB에 저장하고 질의 처리를 지원하는 여러 기법들이 본 논문의 XML 질의 캐쉬의 저장 기법으로 원용하는 데는 비효율적임을 의미한다. PDF 기법의 성능이 좋지 않은 이유는 여러 개의 PDOM 조각 화일을 읽어 메모리에 적재한 후 텍스트로 변환하는 작업이 수행되어야 하기 때문이다. 반면 미리 텍스트 조각들을 유지하고 있는 PDTF는 좋은 성능을 나타냈다.

결과 문서를 미리 텍스트로 저장하고 있는 IT 기법의 성능이 최적이다. ILT 기법은 읽고 색인 화일 i를 읽고 그것을 전처리한 이후 그를 통해 최신의 텍스트 조각을 Xt 화일로부터 검색하는 과정 만큼의 시간이 더 걸리므로 IT 보다 약간의 시간이 더 걸렸다. PDOM 화일을 사용하는 기법인 IPD와 PDTF 기법은 ILT 기법보다는 성능이 떨어지지만 TD 기법보다는 개선된 성능을 보였다.

5.3 저장 시간

그림 8은 각각의 기법에서 그림 2의 형식으로 된 캐쉬 XML 문서를 XML 데이타 서버로부터 받아 응용 서버의 캐쉬 공간에 저장하는 데 걸리는 시간을 나타낸 것이다. TD 기법이 최적이다. 이는 캐쉬 문서에 대해 저장 전에 아무런 전처리를 하지 않기 때문이다. PD 기법이 그 다음 적은 시간을 소요하는 것으로 나타났으며, 캐쉬 문서에서 qdocument 엘리먼트의 태그와 DID 속성을 없앤 후 두개의 화일 Xp와 i로 변환하여 저장하는 IPD는 PD보다 조금 더 시간이 걸리는 것으로 나타났다. IPD와 유사한 전처리 과정을 거치는 기법 IT와 ILT는 IPD와 거의 동일한 시간이 걸렸다. (ILT와 IT는 초기 저장 시간은 동일하다.)

이들 기법들에 비하여 PDTF, RDB 그리고 PDF 기법은 캐쉬 문서의 저장에 많은 시간이 소요되는 것으로 나타났다. PDF의 경우 캐쉬 문서를 여러 PDOM 조각 화일로 나누어서 저장하는 부담이 크며, PDTF는 조각 화일 수가 PDF의 두배이므로 훨씬 더 오랜 시간이 걸렸다. RDB 기법의 경우, PDTF 보다는 적은 시간이 걸렸으나 PDF 보다는 오래 걸리는 것으로 나타났다.

5.4 저장 공간 요건

그림 9는 캐쉬 XML 문서를 저장하는 데 기법 별로 요구되는 저장 공간을 나타낸 것이다. 텍스트 화일 기반의 기법들에서는 공백 문자를 포함시키지 않았다. 콤팩트한 저장 구조로 된 PDOM 화일을 사용하는 기법 IPD와 PD가 가장 적은 공간을 요하는 것으로 나타났다. IPD에서는 색인 화일 i를 저장해야 하지만 화일 Xp에 qdocument 엘리먼트의 태그와 DID 속성이 저장되지 않는데 색인 화일 없이 qdocument 엘리먼트를 명시적으로 나타내고 있는 PD에 비해 저장 공간이 다소 줄어드는 것으로 나타났다. TD와 PDF는 거의 비슷한 용량을 필요로 하는 것으로 나타났고 IT는 TD보다 약간

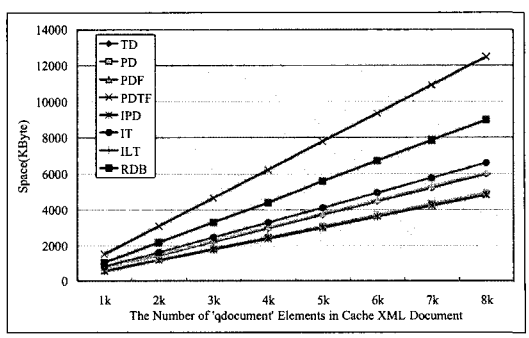

그림 9 캐쉬 문서의 저장 공간

의 공간을 더 요하는 것으로 나타났다. (ILT의 초기 저장 공간은 IT와 동일하다.) 이는 PD와 IPD 간의 비교와 반대되는 결과이지만 두 경우 모두 양자 간의 차이는 미미하다. RDB는 캐쉬 문서를 관계 튜플로 사상하여 분할 저장하므로 상기 기법들에 비해 현저히 많은 양의 공간을 필요로 했고, PDTF는 조각 화일 수가 많아 그보다 더 많은 공간을 요하는 것으로 나타났다.

5.5 성능 평가 결과 요약

성능 평가 결과를 요약하면 다음과 같다.

(1) 기법 TD와 IT는 점진적 갱신을 위하여 캐쉬된 질의 결과를 변경하는 작업에 아주 적합하지 못했다. 이는 캐쉬 문서의 일부분을 변경하기 위해서 캐쉬 문서 전체를 처리해야 하기 때문이다.

(2) 기법 RDB와 PDF는 캐쉬된 질의 결과를 반환하는 데 상대적으로 비효율적이었다. RDB의 경우 관계 튜플로 사상되어 분할 저장된 캐쉬 문서의 엘리먼트들을 검색하여 결과 문서를 구성하는 부담이 다른 저장 구조에 비해 크기 때문이고, PDF는 PDOM 화일 조각들을 텍스트로 변환하여 병합하는 부담이 크기 때문이다. 반면 PDTF는 텍스트 조각 화일들을 유지하기 때문에 결과 반환에서 적절한 성능을 보였다.

(3) 기법 PDTF, PDF, 그리고 RDB는 분할 저장 체계를 사용하므로 다른 기법에 비해 캐쉬 문서를 초기 저장하는 데 더 오랜 시간을 소요했다.

(4) PDTF를 제외한 PDOM 화일 기반의 기법들의 저장 공간 효율이 가장 좋았다.

(5) 점진적 갱신을 위해 캐쉬된 질의 결과를 변경하는 연산 및 캐쉬된 질의 결과를 반환하는 연산 양자 간에 존재하는 성능 트레이드오프를 가장 잘 절충하는 기법은 ILT였다. IPD도 ILT 만은 못하지만 양면에서 모두 만족할만한 성능을 보였다. PDTF도 그러한데 다만 공간 효율이 좋지 못했다.

기법 ILT, IPD, 그리고 PDTF를 제외한 나머지 기법들도 주어진 웹 응용의 특성 및 캐쉬된 질의 결과의 변경 및 검색 두 연산의 상대적 빈도에 따라 경쟁력 있는 저장 구조일 수 있다. 예를 들어, 기법 PDF는 캐쉬 문서의 일부 변경에 아주 효율적이므로 캐쉬의 변경은 잦으나 최신의 질의 결과 대신 대략적인 스냅샷을 질의 결과로 반환해도 되는 응용의 경우 아주 효율적이다. 이런 환경에서는 웹 응용에게 반환되는 질의 결과를 적절한 주기로 생성해주면 된다. 기법 IT와 TD도 소스의 변경이 거의 없는 응용의 경우에는 아주 효율적인 저장 기법이라 할 수 있다.

5.6 기존 기법과의 비교

이상의 성능 평가는 본 논문에서 제시한 여덟 개의 XML 질의 캐쉬 저장 기법 간의 비교를 위한 것이었다. 그 결과 캐쉬된 질의 결과의 일부분을 점진적으로 갱신하는 연산과 캐쉬된 질의 결과 전체를 반환하는 두 연산 간에 존재하는 성능 트레이드오프를 가장 잘 절충하는 기법은 ILT로 나타났다. 본 절에서는 ILT와 기존 기법과의 비교 평가 결과를 기술한다.

기존의 질의 캐쉬 연구에서는 본 논문에서 제시한 것과 같은 디스크 기반의 캐쉬 저장 구조를 고려하지 않았다. 따라서 기존의 기술로써 캐쉬된 질의 결과를 디스크에 저장하는 방법으로 생각할 수 있는 것은 현재까지 XML 문서 저장 기법으로 가장 많이 연구된 RDBMS를 이용하는 것이다. 즉, XML 소스 문서가 관계 튜플로 사상되어 관계 데이타베이스에 저장되어 있을 때 그에 대해 수행된 XML 질의 결과를 구성하기 위해 필요한 관계 튜플들을 캐쉬 즉, 관계 테이블에 저장하는 것이다.

본 절에서는 기존의 대표적인 XML-관계 사상인 [24]의 관계 테이블 스키마를 이용한 XML 저장 기법으로 질의 캐쉬를 저장하는 경우와 본 논문에서 제안한 ILT를 이용하는 경우를 비교한다. XML 질의 캐쉬 모델은 3절의 것을 사용하되 XML 소스에 대한 변경으로는 문서 단위의 삽입 및 삭제 그리고 문서의 트리 구조를 변경하지 않는 텍스트 값 (즉, PCDATA)의 수정만 고려하였다. 이러한 제약은 [24]의 XML 저장 기법이 변경에 강건한 XML 번호 체계를 지원하지 않아 XML 소스에 대한 엘리먼트 단위의 변경을 처리할 수 없기 때문이다.

실험은 TPC-W 벤치마크의 order XML 문서 10,000개를 대상으로 RDBMS로는 Oracle 9i를 사용하여 Celeron 2GHz CPU 와 512MB 메모리가 장착된 시스템에서 수행되었다. 주요 실험 파라미터로는 질의 선택률(S)과 소스 변경률(U)을 들 수 있다. S는 전체 소스 문서 중 3절의 질의 캐쉬 모델에 따라 질의를 만족하는 문서의 비율이다. U는 소스 문서 중 변경이 발생한 문서의 비율이다. 본 실험에서는 S를 0.2 및 0.3, U를 1% 및 10%로 설정하였다.

그림 10과 그림 11은 [24]의 기법 기반과 ILT 기반의 XML 질의 캐쉬 저장 기법을 사용하였을 때의 성능을 비교한 것이다. 측정된 시간은 질의 선택률 S의 질의 결과가 캐쉬되어 있을 때 XML 소스에 대한 비율 U 만큼의 변경을 그것에 점진적으로 반영하는 시간과 갱신된 질의 캐쉬로부터 질의 결과를 반환하는 시간을 모두 합한 것이다. 비교 결과 본 논문에서 제시한 ILT 기반의 성능이 [24]의 기법 기반의 성능보다 약 17%에서 29%까지 우수한 것으로 나타났다.

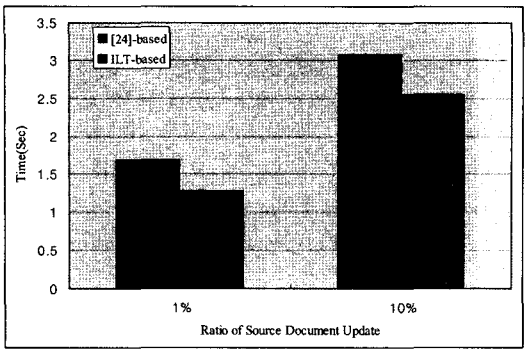

그림 10 기존 기법과의 성능 비교 (S=0.2)

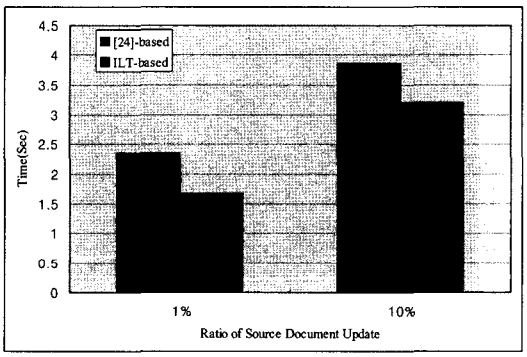

그림 11 기존 기법과의 성능 비교 (S=0.3)

6. 결 론

본 논문에서는 XML 질의 캐쉬의 저장 기법을 다루었다. 일반적인 텍스트 화일에 기반을 둔 기법으로 TD (Text/DOM), IT(Indexed Text), 그리고 ILT(Indexed Log-structured Text), 영속성 있는 DOM(PDOM) 화일에 기반을 둔 기법으로 PD(PDOM), IPD(Indexed PDOM), PDF(PDOM Fragments), 그리고 PDTF (PDOM/Text Fragments), RDBMS를 사용하는 기법 RDB의 총 여덟 개의 XML 질의 캐쉬 저장 기법을 제시하였다. 이들을 모두 구현하여 성능을 비교 평가하였다. 또한 기존의 XML 저장 기술에 기반을 둔 질의 캐쉬 저장 기법과도 비교 평가하였다.

최근에 많은 주목을 받기 시작한 XML 질의 캐쉬 기법에 대한 연구들에서 캐쉬된 질의 결과를 어떻게 저장하는 것이 효율적인가 하는 문제는 다루지 않았다. 캐쉬에 대한 주요 두 연산인 캐쉬 전체의 검색과 점진적 갱신을 위한 캐쉬 일부의 변경 간에는 명백히 성능 트레이드오프가 존재하므로 캐쉬의 저장 구조를 설계하는 것은 단순하지 않다. 본 논문의 기여는 디스크 기반의 다양한 캐쉬 저장 구조를 제시하고 상기 두 연산 간의 성능 트레이드오프를 실험을 통해 검토함으로써 XML 질의 캐쉬 기술의 한 요소를 제공하였다는 데 있다.

향후 연구로는 다음을 들 수 있다. 첫째, 캐쉬된 질의 결과를 반환하는 보다 효율적인 과정에 대한 연구가 필요하다. 이는 소스로부터 질의 결과를 구성하는 XML 문서 조각을 캐쉬하면서 부수적으로 같이 저장되는 캐쉬와 소스 간의 사상 정보의 구성 및 표현과 관련된 문제로 캐쉬 문서 구조의 보다 효율적인 모델링을 검토해야 하는 문제이다. 둘째, 기법 ILT와 IPD에서 각각 Xt 화일과 Xp 화일의 재구성 기법과 정책에 대한 검토가 필요하다. 셋째, 본 논문에서 고려된 저장 기법들 이외의 추가 기법들을 생각할 수 있다. 예를 들어, 기법 PDTF는 PDOM 화일과 텍스트 화일을 다 사용하므로 일종의 기초적인 합성(hybrid) 기법이라 할 수 있다. 본 논문의 실험 결과에 나타난 성능 트레이드오프를 바탕으로 보다 복잡하지만 더 효율적인 합성 기법의 설계가 필요하다.

참 고 문 헌

[1] A. Gupta and I. Mumick, "Materialized Views: Techniques, Implementations, and Applications," 1999, MIT Press.

[2] A. Levy et al., "Answering Queries Using Views," Proc. of ACM Int'l Symp. on PODS, 1995.

[3] L. Chen and E. Rundensteiner, "Aggregate Path Index for Incremental Web View Maintenance," Proc. 2nd Int'l Workshop on Advanced Issues of E-Commerce and Web-based Information Systems, 2000.

[4] L. Quan et al., "Argos: Efficient Refresh in an XQL-Based Web Caching System," Proc. Workshop on the Web and Databases, 2000, pp. 23-28.

[5] K. Dimitrova, M. El-Sayed, E. Rundensteiner, "Order-sensitive View Maintenance of Materialized XQuery Views," Tech. Rep. WPI-CS-TR-03-17, Computer Science Department, Worcester Polytechnic Institute, May 2003.

[6] A. Sawires, J. Tatemura, O. Po, D. Agrawal, and K. Candan, "Incremental Maintenance of Path-Expression Views," Proc. ACM SIGMOD Int'l Conf. on Management of Data, 2005.

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

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

[9] P. Marron and G. Lausen, "Efficient Cache Answerability for XPath Queries," Proc. the 2nd Int'l Workshop on Data Interation over the Web (DIWeb), 2002, pp. 35-45.

[10] L. Yang, M. Lee, and W. Hsu, "Efficient Mining of XML Query Patterns for Caching," Proc. Int'l

Conf. on VLDB, 2003.
[11] A. Balmin, F. Özcan, K. Beyer, R. Cochrane, and H. Pirahesh, "A Framework for Using Materialized XPath Views in XML Query Processing," Proc. Int'l Conf. on VLDB, 2004.
[12] W. Xu and Z. Özsoyoglu, "Rewriting XPath Queries Using Materialized Views," Proc. Int'l Conf. on VLDB, 2005.
[13] B. Mandhani and D. Suciu, "Query Caching and View Selection for XML Databases," Proc. Int'l Conf. on VLDB, 2005.
[14] G. Huck, I. Macherius, and P. Fankhauser, "PDOM: Lightweight Persistency Support for the Document Object Scheme," Proc. OOPSLA Workshop "Java and Databases: Persistence Options," 1999.
[15] Schmidt, A. et al., "XMark: A Benchmark for XML Data Management," Proc. Int'l Conf. on VLDB, 2002.
[16] XBench: http://db.uwaterloo.ca/~ddbms/projects/xbench/.
[17] M. Yoshikawa, T. Amagasa, T. Shimura, S. Uemura, "XRel: A Path-Based Approach to Storage and Retrieval of XML Documents Using Relational Databases," ACM Trans. on Internet Technology, Vol. 1, No. 1, Aug. 2001, pp. 110-141.
[18] J. Kohl et al., "HighLight: Using a Log-structured File System for Tertiary Storage Management," Proc. Winter USENIX, 1993, pp. 435-447.
[19] G. Huck, and I. Macherius, "GMD-IPSI XQL Engine," http://xml.darmstadt.gmd.de/xql/.
[20] J. Shanmugasundaram et al., "Efficiently Publishing Relational Data as XML Documents," Proc. Int'l Conf. on VLDB, 2000, pp. 65-76.
[21] Xerces Java Parser: http://xml.apache.org/xerces-j/.
[22] TPC-W: Transaction Processing Performance Council. http://www.tpc.org/tpcw/.
[23] ToXgene: http://www.cs.toronto.edu/tox/toxgene/.
[24] D. Florescu and D. Kossmann, "Storing and Querying XML Data Using an RDBMS," IEEE Data Engg. Bulletin, Sep. 1999, pp. 27-34.

김 영 현
정보과학회논문지 : 데이타베이스
제 33 권 제 3 호 참조

강 현 철
정보과학회논문지 : 데이타베이스
제 33 권 제 3 호 참조