ORDBMS를 사용한 XML 질의 캐쉬의 효율적인 지연 갱신

황 대 현[†] · 강 현 철[††]

요 약

XML 문서의 생성 및 활용도가 증가함으로 인해 XML 문서를 데이터베이스에 저장하여 관리하는 기법이 활발히 연구되고 있다. 관계형 또는 객체 관계형 데이터베이스 관리 시스템(RDBMS/ORDBMS)을 XML 문서의 저장소로 사용하는 것은 현재 가장 실용적인 방법으로 받아들여지고 있다. 데이터베이스에 저장된 XML 문서에 대한 빠른 질의 처리를 위하여 XML 질의 캐쉬를 사용할 수 있다. 그러나 XML 질의 캐쉬의 사용은 하부 자료의 변경에 대한 일관성 유지 비용이 든다. 본 논문에서는, ORDBMS를 XML 질의 캐쉬와 하부 XML 자료의 저장소로 사용하고 변경로그를 사용하여 XML 질의 캐쉬를 지연 갱신할 때, XML 질의 캐쉬에 대한 비효율적인 갱신의 원인인 변경로그에 저장된 동일한 XML 문서에 대한 중복 변경을 제거 또는 여과하는 알고리즘을 제시한다. 또한 이를 바탕으로 XML 질의 캐쉬의 갱신에 사용할 최적화된 SQL 문을 생성한다. 실험을 통해 본 논문에서 제안한 XML 질의 캐쉬의 지연 갱신 방법의 효율성을 보인다.

키워드 : XML, XML 질의 캐쉬, 지연 갱신, 객체 관계형 데이터베이스 관리 시스템

Efficient Deferred Incremental Refresh of XML Query Cache Using ORDBMS

Dae Hyun Hwang[†] · Hyunchul Kang[††]

ABSTRACT

As we are to deal with more and more XML documents, research on storing and managing XML documents in databases are actively conducted. Employing RDBMS or ORDBMS as a repository of XML documents is currently regarded as most practical. The query results out of XML documents stored in databases could be cached for query performance though the cost of cache consistency against the update of the underlying data is incurred. In this paper, we assume that an ORDBMS is used as a repository for the XML query cache as well as its underlying XML documents, and that XML query cache is refreshed in a deferred way with the update log. When the same XML document was updated multiple times, the deferred refresh of the XML query cache may get inefficient. We propose an algorithm that removes or filters such duplicate updates. Based on that, the optimal SQL statements that are to be executed for XML query cache consistency are generated. Through experiments, we show the efficiency of our proposed deferred refresh of XML query cache.

Key Words : XML, XML Query Cache, Deferred Incremental Refresh, ORDBMS

1. 서 론

XML은 웹상에서 이기종 시스템 및 응용 간의 데이터 교환의 표준으로 대두되고 있다. 사용자가 쉽게 작성할 수 있는 유연한 그래프 형태의 데이터 모델로 표현되는 XML은 반구조적(semistructured) 특성과 태그(tag)를 사용한 자기 기술적(self-descriptive) 특성을 갖는다. 현재 통신, 전자상거래, 전자서적, 디지털 도서관 등에서 XML을 사용하여 정보를 나타내고 있고, 그 활용 분야와 사용량은 점점 증가하고 있다. 이와 같이 생성된 XML 문서를 데이터베이스에 저장하고 관리하는 연구가 활발히 수행되고 있다[1-3].

XML의 중첩된 계층적 구조는 전자상거래에서 사용되는 상품 목록과 같이 여러 XML 문서를 통합할 경우 쉽게 이용할 수 있는 장점이 있는 반면, 자료를 저장할 테이블 스키마가 먼저 결정되는 RDBMS나 ORDBMS로는 처리하기 까다로운 단점이 있다. 이와 같은 이유로 XML의 반구조적인 형태를 그대로 저장하는 Software AG의 Tamino[4], X-Hive사의 X-Hive/DB[5]와 같은 XML 전용(native) 데이터베이스가 연구 및 개발되고 있다. 그러나 현재 RDBMS 또는 ORDBMS가 널리 사용되고 있고, 주요 RDBMS나 ORDBMS 개발 업체들도 XML 관련 특성을 자사의 제품이 지원하도록 계속 연구 및 개발하고 있다[6, 7]. 또한, XML 전용 데이터베이스는 RDBMS나 ORDBMS가 제공하는 표준 비절차 언어인 SQL, 동시성 제어(concurrency control),

※ 본 논문은 한국과학재단 특정기초연구사업(R01-2003-000-10395-0) 지원으로 수행되었음.
† 준 회 원 : 중앙대학교 컴퓨터공학부 박사과정
†† 종신회원 : 중앙대학교 컴퓨터공학부 교수
논문접수 : 2005년 8월 10일, 심사완료 : 2005년 12월 29일

복구(recovery)와 같은 기능성, 그리고 대용량 자료를 저장 및 관리하는 안전성 등의 면에서 RDBMS나 ORDBMS와 비교할 때 부족한 점이 많다. 따라서 현재 RDBMS나 ORDMBS를 사용하여 XML 문서를 저장하고 관리하는 것이 가장 실용적인 것으로 간주되고 있다.

뷰(view)는 자료의 여과와 통합에 효율적으로 사용된다. 의사결정에 필요한 정보를 제공하기 위해 분산된 하부 데이터베이스들로부터 자료를 수집, 분석하는 데이터웨어하우스에서 뷰는 사용자의 질의에 빠르게 응답할 수 있도록 실제 자료를 저장한 실체뷰(materialized view)의 형태로 사용된다. 한편, 웹상의 XML 문서를 대상으로 XML 웨어하우스를 구축한 후 사용자가 원하는 정보를 가공하여 제공하려는 연구도 수행되고 있다[8, 9]. 본 논문은 XML 질의 캐쉬를 실체뷰로 지원하는 시스템에 관한 것이다. 질의 캐쉬는 자주 사용하는 질의의 결과를 캐쉬하여 동일 질의가 다시 제기되면 결과를 바로 사용자에게 제공하기 위한 것이다. 그러나 캐쉬를 사용할 경우, 하부 데이터베이스에 저장된 자료가 변경되었을 때, 그 변경을 반영하여 하부 자료와 동일한 내용을 유지하기 위한 질의 캐쉬의 일관성 유지 비용이 발생한다. 데이터웨어하우스 환경에서는 예를 들어 낮에 수행된 하부 자료의 변경에 대해 업무 부하가 줄어든 밤에 캐쉬의 일관성 유지를 위해 일괄 처리하는 지연 갱신이 사용될 수 있다[10].

본 논문에서는 ORDMBS를 사용하여 XML 실체뷰와 하부 XML 자료를 저장하고 변경로그를 사용하여 XML 질의 캐쉬를 지연 갱신할 때, XML 질의 캐쉬에 대한 비효율적인 갱신의 원인인 변경로그에 저장된 동일한 XML 문서에 대한 중복 변경을 제거 또는 여과하는 알고리즘을 제시한다. 이를 바탕으로 XML 질의 캐쉬의 갱신에 사용할 최적화된 SQL 문을 생성한다. 이는 XML 실체뷰 기법을 제시한 기존 연구 [11]의 결과를 확장한 것이다. [11]에서 지연 갱신을 처리하는 방식은, 시간 순으로 변경 내용을 변경로그에 기록하고 그것을 순차적으로 처리함으로써 동일한 하부 XML 문서에 대한 중복 변경을 제거 및 여과 처리하지 못하고 따라서 XML 질의 캐쉬의 일관성을 유지하기 위한 SQL 문도 최적화되어 있지 않다는 문제점을 갖는다. 또한, [11]의 변경로그 레코드의 자료구조는 단순 텍스트로 되어 있어 그 처리가 효율적이지 못한 단점도 있다. 따라서 본 논문에서는 이러한 문제점들을 해결하기 위해 동일 XML 문서의 중복 변경을 제거, 여과하고 최적의 SQL 문을 생성한다. 또한, 클래스로 변경로그를 구성하여 XML 질의 캐쉬의 일관성을 유지할 경우 효율성을 높이도록 하였다. 실험 결과, 본 논문에서 제시한 방법이 [11]의 방법 및 XML 질의 캐쉬를 사용하지 않는 재생성(recomputation) 방법과 비교하여 큰 성능 향상을 보였다.

본 논문의 구성은 다음과 같다. 먼저, 2절에서 관련 연구를 살펴보고, 3절에서는 ORDBMS를 하부 XML 문서와 XML 질의 캐쉬의 저장소로 사용할 경우 XML 질의 캐쉬의 일관성 유지를 위한 지연 갱신의 비용 모델을 기술한다. 4절에서는 개선된 자료 구조의 변경로그를 사용하여 동일한 하부 XML 문서에 대한 중복 변경을 제거, 여과하는 알고리즘을 실제 예를 통해 살펴보고, XML 질의 캐쉬의 갱신을 위한 최적화된 SQL 문을 제시한다. 5절에서는 제시하는 기법의 효율성을 검증하는 실험 결과를 기술한다. 마지막으로 6절에서 결론을 맺고 향후 연구를 제시한다.

2. 관련 연구

뷰는 일반적으로 하부 자료에 대한 뷰의 정의만을 저장하여 사용하기 때문에 사용될 때마다 하부 자료로부터 재생성된다. 데이터웨어하우스 환경에서는 사용자의 질의에 대한 빠른 응답을 위해 자주 사용하는 자료를 저장한 실체뷰를 사용한다[12]. 그러나 실체뷰는 하부 자료의 변경에 대해 실체뷰의 일관성을 유지해야하는 부담이 있다. 일관성을 유지하기 위한 방법으로는 실체뷰를 하부 자료로부터 재생성하는 방법과 점진적으로 갱신(incremental refresh)하는 방법이 있다. 일반적으로 점진적 갱신 방법을 사용하고 있고 RDBMS에서 관계 실체뷰의 일관성 유지에 대한 많은 연구가 있었다[13].

실체뷰의 일관성 유지에 대한 연구는 반구조적 데이터 모델에 대해서도 연구되었고[14, 15] XML에 대해서도 연구되었다[11, 16-18]. [15]에서는 반구조적 실체뷰의 일관성 유지 기법을 제안하였다. 이는 [14]의 제한적인 변경에 대한 실체뷰의 일관성 유지 기법을 일반화한 것이다. 객체뿐만 아니라 구조까지 저장하고 하부 자료의 변경과 실체뷰의 연관성을 빠르게 확인할 수 있도록 부가 정보인 연관된 객체 ID (RelevantOids)를 사용하였다. XML 실체뷰의 일관성 유지는 연관성 검사를 위한 부가 정보인 APIX(Aggregate Path IndeX)를 제시한 연구가 있다[16]. APIX를 사용하여 XML 실체뷰를 지원하는 Argos 시스템[17]은 PDOM(Persistent DOM)을 사용하여 XML 문서를 저장하였다. 그러나 하부의 XML 자료를 지역 시스템에 저장하고 APIX라는 보조 인덱스 구조를 생성하여 질의를 처리함으로 인해 질의의 소스 자료가 방대할 경우 비효율적이라는 단점이 있다. [11]은 ORDBMS에 저장된 XML 자료에 대한 실체뷰의 지연 갱신 기법을 제시하였고 [18]은 실험을 통하여 그것의 성능을 평가한 것이다.

XML 실체뷰는 XML의 유연한 구조를 사용하여 사용자가 원하는 자료를 저장한 여러 XML 실체뷰의 내용을 통합하여 제공할 수 있는 장점이 있다[19]. 또한 질의 캐쉬는 질의 재작성(query rewriting)을 이용하여 XML 실체뷰를 사용하여 관련 질의의 처리에 이용될 수 있다[20]. 이러한 기법은 질의 최적화, 물리적 자료의 독립성 보장, 자료 통합, 데이터웨어하우스 디자인 등을 위해 현재 많은 주목을 받는 연구 주제이다[21].

RDBMS나 ORDBMS로 XML 문서를 저장, 관리하는 것은 서론에서 언급한 것과 같이 실용적인 방법이기 때문에 많은 연구가 수행되었다[1-3]. 그러나 본 논문에서 제시하는 XML 질의 캐쉬의 일관성 유지 기법은 이러한 연구들에서 제시된 것과는 다른 기법을 필요로 한다.

3. 지연 갱신의 비용 모델

지연 갱신은 하부 자료의 변경 시 변경로그에 기록하고 이를 사용하여 하부 자료의 여러 변경 내용을 한 번에 XML 질의 캐쉬에 반영하여 일관성을 유지하는 기법이다. 지연 갱신은 하부 자료와 XML 질의 캐쉬의 저장소가 웹 상에서 분리된 위치에 있을 경우, 하부 자료의 변경이 발생할 때마다 XML 질의 캐쉬에 반영하는 즉각 갱신보다 효율적이다. XML 질의 캐쉬의 지연 갱신을 위한 XML 질의 캐쉬의 변경은 일관성 유지의 주체에 따라 다음과 같은 두 가지 방법으로 분류할 수 있다. 첫째 방법은 하부 자료의 관리 시스템이 주기적으로 하부 자료의 변경 사항을 XML 질의 캐쉬에 반영하는 것이고, 둘째 방법은 사용자의 요구에 의해 하부 자료의 변경 사항을 XML 질의 캐쉬에 반영하는 것이다. 이 중 첫째 방법은 임의의 시간에 XML 질의 캐쉬와 관련된 자료를 변경하기 위해 잠금(locking)을 사용하여 이를 사용하는 다른 사용자의 접근을 막는다는 단점이 있다. 따라서 본 논문에서는 사용자의 요구에 의하여 일정한 시간(특히 시스템의 작업 부하가 낮은 야간 시간대)에 XML 질의 캐쉬의 일관성을 유지하는 둘째 방법을 고려하였다.

지연 갱신을 사용하여 최신의 XML 질의 캐쉬를 사용자에게 제공하려면 다음과 같은 일련의 방법을 사용한다. 첫째, 변경로그 파일을 메모리로 적재한다. 둘째, ORDBMS의 테이블에 저장된 XML 질의 캐쉬와 관련된 부가 정보를 메모리로 적재한다. 셋째, 메모리에 적재된 변경로그의 내용과 해당 XML 질의 캐쉬의 부가 정보를 사용하여 연관성 검사를 한다. 넷째, 연관성 검사의 결과를 사용하여 XML 질의 캐쉬를 점진적으로 갱신한다. 다섯째, 변경된 XML 질의 캐쉬와 관련된 부가 정보를 갱신한다. 마지막으로 사용자에게 갱신된 실체뷰를 반환하기 위해 갱신된 XML 질의 캐쉬의 해당 튜플들을 검색하여 XML 태깅을 수행, XML 파일로 저장한다. 본 논문에서는 이와 같은 일련의 방법에서 갱신된 실체뷰를 XML 파일로 변환하여 저장하는 마지막 작업을 뺀 나머지 작업에 걸린 시간을 XML 질의 캐쉬의 지연 갱신 시간이라고 정의한다.

XML 질의 캐쉬의 지연 갱신 시간 T_c는 (식 1)과 같다. 하부 자료의 변경과 XML 질의 캐쉬의 연관성 검사를 하기 위한 전처리에 걸린 시간은 T_{log}과 T_{aug}이다. T_{log}는 파일에 저장된 변경로그를 메모리에 적재하는 시간이고, T_{aug}는 ORDBMS의 테이블에 저장된 XML 질의 캐쉬와 관련된 부가 정보를 메모리에 적재하는 시간이다.

$$T_c = T_{log} + T_{aug} + T_{rel} + T_{exe} + T_{sav} \quad \text{(식 1)}$$

T_{rel}은 하부 자료의 변경과 XML 질의 캐쉬의 연관성을 검사하는 연관성 검사 시간이다. 이는 다음과 같이 3가지 작업으로 이루어진다.

$$T_{rel} = T_{chk} + T_{info} + T_{sql} \quad \text{(식 2)}$$

T_{chk}는 변경로그에 기술된 하부 자료의 변경 내용과 XML 질의 캐쉬와의 연관성을 검사하여 5개의 연관성 유형(INSERT, DELETE, MODIFY-I, MODIFY-M, MODIFY-D)[11] 중 해당되는 것 하나 또는 연관이 없음을 결정하는 데 걸린 시간이다. 즉, 해당 XML 질의 캐쉬에 대한 변경 정보의 생성 여부를 구분하는 시간이다. T_{info}는 확정된 연관성 유형을 사용하여 XML 질의 캐쉬의 갱신 정보를 생성하는 시간이고, T_{sql}은 생성된 갱신 정보를 사용하여 ORDBMS에 저장된 XML 질의 캐쉬를 변경하기 위해 변경 SQL문을 생성하는 시간이다. (식 1)에서 연관성 검사 시간 T_{rel}의 다음 항인 T_{exe}는 생성된 변경 SQL문을 사용하여 XML 질의 캐쉬를 실제로 갱신하는 시간이고, T_{sav}는 하부 자료의 변경을 반영한 XML 질의 캐쉬 관련 부가 정보를 저장하는 시간이다. <표 1>은 이들 지연 갱신의 비용 파라미터를 정리한 것이다.

〈표 1〉 지연 갱신 비용 파라미터

파라미터	내 용	
T_{log}	변경로그를 메모리에 적재하는 시간	
T_{aug}	부가 정보를 메모리에 적재하는 시간	
T_{rel}	T_{chk}	변경로그에 기술된 하부 자료의 변경 내용과 XML 질의 캐쉬의 연관성을 검사하는 시간
	T_{info}	XML 질의 캐쉬와 연관된 변경에 대하여 갱신 정보를 생성하는 시간
	T_{sql}	갱신 정보를 사용하여 XML 질의 캐쉬를 변경하기 위한 SQL 문을 생성하는 시간
T_{exe}	생성된 SQL 문으로 XML 질의 캐쉬를 갱신하는 시간	
T_{sav}	변경된 부가 정보를 저장하는 시간	

4. 지연 갱신의 최적화

본 절에서는 지연 갱신을 이용한 XML 질의 캐쉬의 일관성 유지의 최적화 기법을 제안한다. 먼저 4.1절에서 기존 연구 [11]의 문제점을 기술하고, 4.2절에서 지연 갱신 과정을 최적화하는 방법을 제시하고 예를 사용하여 설명한다.

4.1 RX-S[11]의 문제점

[11]에서 제안한 XML 질의 캐쉬의 갱신 기법은 변경로그 레코드들을 시간 순으로 저장된 순서대로 순차(sequence) 처리한다. 따라서 본 논문에서는 이후 이 기법을 "**RX-S(Sequence)**"로 부른다. 본 논문에서는 RX-S 및 [11]의 문제점을 다음과 같이 크게 세 가지로 지적한다. 첫째, 텍스트 형태로 변경로그를 구성하였기 때문에 일관성 유지를 위해 변경로그의 분석 과정을 필요로 한다. 둘째, 동일한 하부 XML 문서의 중복된 변경으로 인해 불필요한 SQL 문의 생성 및 실행과 같은 비효율성이 발생한다. 셋째, 변경로그 레코드 각각에 대해 별도의 SQL문을 생성하여 실행하므로 매우 비효율적이다.

4.2 개선된 방법

본 논문에서는 4.1절에서 살펴본 RX-S의 문제점을 해결

〈표 2〉 개선된 방법 RX의 내용과 관련 단축 시간

단축 시간	개선 사항
T_{chk}	클래스를 이용한 변경로그 저장
T_{sql}, T_{exe}	동일한 하부 XML 문서 중복 변경의 제거 및 여과
T_{exe}	ORDBMS에 저장된 XML 질의 캐쉬의 갱신을 위한 SQL 문의 최적화

하고자 다음과 같이 개선된 방법을 제안한다. 제안되는 개선 기법은 "RX"로 부른다. 변경로그를 클래스 형태로 저장한 후 리스트로 연결하고, 동일한 하부 XML 문서에 대한 중복 변경의 제거 및 여과를 처리하는 알고리즘을 적용하고, ORDBMS에 저장된 질의 캐쉬에 하부 XML 문서의 변경된 내용을 반영하기 위한 SQL 문을 최적화하여 생성한다. 이를 통해 XML 질의 캐쉬의 일관성 유지 시간을 크게 단축하였다. <표 2>는 XML 질의 캐쉬의 최적화된 지연 갱신을 위해 본 논문에서 제시하는 개선된 사항과 개선된 내용으로 인해 <표 1>의 지연 갱신 비용 파라미터 중 시간이 단축되는 것을 정리한 것이다.

4.2.1 클래스를 이용한 변경로그 저장

(그림 1)은 본 논문에서 사용하는 변경로그의 구조를 나타낸 것이다. LSN(Log Sequence Number)은 시간 순으로 저장되는 변경로그에 부여되는 식별자로서 순차적인 번호를 나타낸다. U-Type은 하부 자료의 변경 유형을 나타내는 것으로, "I", "D", "M"의 세 가지 유형 중 하나를 기록한다. "I"는 INSERT를, "D"는 DELETE를, "M"은 MODIFY를 뜻한다. DID는 변경 문서의 ID를 나타내고, EECP는 엘리먼트 ID(EID), 엘리먼트 이름(Ename), 내용(Content), 부모 엘리먼트 ID(ParentID)와 같이 하부 XML 문서의 변경 내용을 저장한다. EID는 변경 XML 문서의 엘리먼트 ID를 나타내고, Ename은 변경되는 엘리먼트 이름을 나타낸다. 그리고 Content는 변경된 엘리먼트의 내용을 나타내고 ParentID는 해당 엘리먼트의 부모 엘리먼트의 ID를 나타낸다.

U-Type이 "I"일 때 EECP는 여러 개 기록될 수 있다. 이는 하부 자료에 대한 접근 없이 XML 질의 캐쉬의 일관성 유지를 위해 필요한 정보를 모두 포함하는 변경로그 레코드를 작성하기 위함이다. U-Type이 "D"일 때 EECP는 없다. 그 이유는 본 논문에서 문서 단위의 삽입, 삭제와 엘리먼트 단위의 수정을 고려하기 때문이다. 따라서, 변경 유형이 "D"인 경우 문서 단위의 삭제를 나타내는 문서 ID만이 필요하다. 변경 유형이 "M"일 경우, 엘리먼트의 변경을 나타내기 위해 하나의 EECP를 사용하여 해당 엘리먼트의 변경 내용을 저장한다((그림 5) 참조). 이와 같은 변경로그의 구조를

LSN	U-Type	DID	EECP {{EID, ENAME, CONTENT, PARENTID}, ...,}
로그번호	변경유형	변경문서 ID	변경 내용

(그림 1) 변경로그의 구조

클래스로 구현하면 XML 질의 캐쉬의 일관성 유지를 위해 변경로그의 내용을 분석하는 비용 없이 저장된 내용을 사용할 수 있다는 장점이 있다.

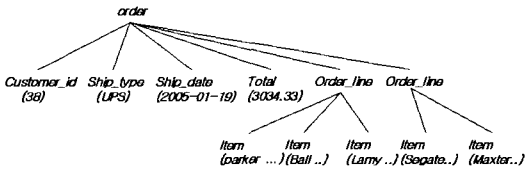
4.2.2 동일한 하부 XML 문서의 중복 변경 제거 또는 여과

본 절에서는 동일한 하부 XML 문서의 중복 변경을 제거 또는 여과하는 알고리즘을 제시하고 실제 예를 사용하여 처리되는 과정을 살펴본다.

(그림 2)와 같은 DTD를 갖는 order 문서들을 ORDBMS에 분할 저장하면 (그림 3)과 같이 저장된다. 본 논문에서 사용할 XML 실체뷰의 예 C_1을 정의하는 XML 질의로, [11]

```
<!ELEMENT order(customer_id, ship_type, ship_date, total, order_line*)>
<!ELEMENT customer_id(#PCDATA)>
<!ELEMENT ship_type(#PCDATA)>
<!ELEMENT ship_date(#PCDATA)>
<!ELEMENT total(#PCDATA)>
<!ELEMENT order_line(item*)>
<!ELEMENT item(#PCDATA)>
```

[Order1.xml]



(그림 2) order XML 문서의 DTD와 문서 트리의 예

DID	EID	Ename	Content	ParentID
1	1	order	-	0
1	2	customer_id	38	1
1	3	ship_type	UPS	1
1	4	ship_date	2005-1-19	1
1	5	total	3034.33	1
1	6	order_line	-	1
1	7	item	Parker pencil	6
1	8	item	Ball point pen	6
1	9	item	Lamy fountain pen	6
1	10	order_line	-	1
1	11	item	Seagate st9808211A HDD	10
1	12	item	Maxter 6b300s0 HDD	10
3	1	order	-	0
3	2	customer_id	27	1
3	3	ship_type	HAN	1
3	4	ship_date	2005-2-17	1
3	5	total	2325.69	1
3	6	order_line	-	1
3	7	item	Girl applique album	6
3	8	item	Morning system 2500-b12	6
3	9	order_line	-	1
3	10	item	Samsung ata-133 HDD	9
3	11	item	Notebook optical mouse	9
...	...	...	...	...

(그림 3) ORDBMS에 분할 저장된 order XML 문서

CacheID	DID	EID	Ename	Content	ParentID
C_1	3	2	customer_id	27	1
C_1	3	4	ship_date	2005-2-17	1
C_1	3	5	total	2325.69	1

(그림 4) XML 질의 캐쉬의 초기 내용

의 용어에 따라 조건 엘리먼트(Condition Element) CE = "ship_type", 조건 엘리먼트의 내용 P = "HAN", XML 질의 캐쉬에 저장할 목적 엘리먼트(Target Element) TE ∈ { "customer_id", "ship_date", "total"}이라 하면, XML 질의 캐쉬에 저장된 초기 내용은 (그림 4)와 같다. 한편, (그림 3)과 같이 하부 XML 문서를 저장하는 테이블 구조는 다음과 같다. DID는 문서의 ID를 저장하고, EID는 XML 문서의 엘리먼트 ID를 깊이 우선 탐색(Depth First Search) 순서로 부여한 것을 저장한다. ParentID는 각 엘리먼트의 부모 엘리먼트의 식별자(ID)를 저장한다. Ename은 각 엘리먼트의 태그명을 저장하고, Content는 각 단말(leaf) 노드의 내용을 저장한다. XML 질의 캐쉬의 테이블 구조는 XML 질의 캐쉬의 ID인 CacheID를 첨가한 것을 제외하고 하부 자료를 저장한 테이블과 같은 구조를 갖는다((그림 4) 참조).

하부 자료의 변경이 변경 로그에 기록된 후, 일관성 유지를 위하여 XML 질의 캐쉬를 지연 갱신하는 작업에 대해 설명하면 다음과 같다. 먼저 부가 정보의 사용에 대한 설명이 필요하다. XML 질의 캐쉬의 일관성 유지를 위한 부가 정보는 하부 자료의 변경 내용과 XML 질의 캐쉬에 저장된 자료의 연관성 여부를 빨리 알아내기 위해 사용된다[15, 16]. [11]의 RX-S에서는 부가 정보를 XML 질의 캐쉬 갱신 테이블이라는 이름의 테이블에 저장한다. (그림 4)의 테이블에 저장된 XML 실체뷰 C_1을 (그림 5)와 같은 변경로그에 대하여 지연 갱신한다고 하자. C_1의 관련 DIDList [11]를 DL_1이라 하면 DL_1= {3}이다. <표 3>은 지연 갱신 최적화 알고리즘인 (그림 9)의 checkRelevance() 함수를 사용하여 각 변경로그 레코드와 C_1의 연관성을 검사한 결과와 연관성 검사에

LSN	U-Type	DID	EECP { (EID, ENAME, CONTENT,PARENTID), . }
1	I	5	{(2, customer_id, 11, 1), (3, ship_type, HAN, 1), (4, ship_date, 2005-05-12, 1), (5, total, 2028.4, 1), ...}
2	M	8	{(3, ship_type, HAN, 1)}
3	M	3	{(7, item, samsung HDD, 6)}
4	M	5	{(4, ship_date, 2003-10-16, 5)}
5	I	7	{(2, customer_id, 28, 1), (3, ship_type, HAN, 1) (4, ship_date, 2005-04-28, 1), (5, total, 5421.4, 1), ...}
6	M	9	{(2, ship_type, HAN, 0)}
7	M	7	{(2, ship_type, UPS, 1)}
8	M	3	{(4, ship_date, 2005-03-10, 1)}
9	I	11	{(2, customer_id, 23, 1), (3, ship_type, HAN, 1), (4, ship_date, 2005-03-08, 1), (5, total, 6721.4, 1), ...}
10	M	9	{(2, ship_type, UPS, 0)}
11	M	5	{(4, ship_date, 2003-10-20, 1)}
12	M	11	{(4, ship_date, 2005-03-15, 1)}
13	D	5	{}

(그림 5) 변경로그의 예

따라 갱신되는 DL_1의 값을 나타낸 것이다. checkRelevance() 함수는 각 변경로그 레코드의 연관성 유형을 찾고, 연관성 유형에 따라 DL_1을 갱신한다. 이때 클래스를 사용하여 구현된 변경로그는 분석 없이 바로 메모리로 적재되어 연관성 검사에 사용될 수 있다.

이와 같이 기록된 순서대로 변경로그 레코드를 처리하면서 (그림 9)의 알고리즘은 다음과 같이 동일한 하부 XML 문서에 대한 중복 변경을 제거 또는 여과한다. NTT(New Tuple Table)는 연관성 유형 중 INSERT 및 MODIFY-M 유형의 튜플들을 저장하는 메모리 내 테이블이며 구조는 (그림 4)의 XML 질의 캐쉬의 구조와 동일하다. nTTList는 NTT를 리스트 형태로 메모리에 저장한 것이다. (그림 5)의 변경로그의 예에서 사용된 LSN이 2와 3인 변경로그 레코드는 해당 XML 질의 캐쉬와 관련이 없으므로 반영하지 않는

〈표 3〉 연관성 검사

LSN	연관성 검사 점검 내용	연관성 검사 결과	DL_1
1	U-Type = I & (CE = "Ship_type", P = "HAN")	INS	{3,5}
2	DID ∉ DL_1	irrelevant	{3,5}
3	"item" ∉ TE	irrelevant	{3,5}
4	U-Type = M & DID ∈ DL_1 & "ship_date" ∈ TE	MODIFY/M	{3,5}
5	U-Type = I & CE = "ship_type", P = "HAN"	INSERT	{3,5,7}
6	U-Type = M & DID ∉ DL_1 & (CE = "ship_type", P = "HAN")	MODIFY-I	{3,5,7,9}
7	U-Type = M & DID ∈ DL_1 & (CE = "ship_type", P ≠ "HAN")	MODIFY-D	{3,5,9}
8	U-Type = M & DID ∈ DL_1 & "ship_date" ∈ TE	MODIFY-M	{3,5,9}
9	U-Type = I & (CE = "ship_type", P = "HAN")	INSERT	{3,5,9,11}
10	U-Type = M & DID ∈ DL_1 & (CE = "ship_type", P ≠ "HAN")	MODIFY-D	{3,5,11}
11	U-Type = M & DID ∈ DL_1 & "ship_date" ∈ TE	MODIFY-M	{3,5,11}
12	U-Type = M & DID ∈ DL_1 & "ship_date" ∈ TE	MODIFY-M	{3,5,11}
13	U-Type = D & DID ∈ DL_1	DELETE	{3,11}

다. 연관성 검사 결과를 R이라 할 때 변경로그 레코드 별로 처리 내용을 살펴보면 다음과 같다.

• LSN = 1
R = INSERT : DID 값인 5를 sI에 저장하고 변경로그 레코드의 EECP 중 XML 질의 캐쉬 정의에서 사용된 TE를 genInsTuple() 함수를 사용하여 nTTList에 저장한다 ((그림 9)의 (가) 참조).

• LSN = 4
R = MODIFY/M : DID 값인 5가 이미 sI에 속하므로 sMM에 저장하지 않는다. genMMTuple() 함수를 사용하여 변경할 엘리먼트의 내용을 nTTList에 저장한다 ((그림 9)의 (다) 참조).

• LSN = 5
R = INSERT : DID 값인 7을 sI에 저장하고 변경로그 레코드의 EECP중 XML 질의 캐쉬 정의에서 사용된 TE를 genInsTuple() 함수를 사용하여 nTTList에 저장한다 ((그림 9)의 (가) 참조).

• LSN = 6
R = MODIFY/I : DID 값인 9를 sMI에 저장한다 ((그림 9)의 (나) 참조).

• LSN = 7
R = MODIFY/D : LSN이 5인 변경로그 레코드의 처리에서 sI에 저장된 7을 sI에서 제거한다 ((그림 9)의 (마) 참조).

• LSN = 8
R = MODIFY/M, sMM에 DID 값인 3을 저장하고 genMMTuple() 함수를 사용하여 변경할 엘리먼트의 내용을 nTTList에 저장한다 ((그림 9)의 (다) 참조).

• LSN = 9
R = INSERT, DID 값인 11을 sI에 저장하고 변경로그 레코드의 EECP중 XML 질의 캐쉬 정의에서 사용된 TE를 nTTList에 저장한다 ((그림 9)의 (가) 참조).

• LSN = 10
R = MODIFY/D, LSN이 6인 변경로그 레코드의 처리에서 sMI에 저장된 9를 제거한다 ((그림 9)의 (바) 참조).

• LSN = 11
R = MODIFY/M, sMM에 DID 값인 5를 저장하고 genMMTuple() 함수를 사용하여 변경할 엘리먼트의 내용을 nTTList에 저장한다 ((그림 9)의 (다) 참조).

• LSN = 12
R = MODIFY/M, sMM에 DID 값인 11을 저장하고 genMMTuple() 함수를 사용하여 변경할 엘리먼트의 내용을 nTTList에 저장한다 ((그림 9)의 (다) 참조).

• LSN = 13
R = DELETE, LSN이 1인 로그 레코드의 처리에서 sI에 저장된 5를 sI에서 제거한다 ((그림 9)의 (라) 참조).

변경로그 레코드를 저장된 순서로 처리할 때 nTTList에 저장된 삽입 및 수정 사항에 대한 불필요한 내용은 연관성 검사를 모두 끝낸 후, 저장된 sI와 sMM을 사용하여 제거한다. 예를 들면, LSN이 5인 변경로그 레코드의 처리에서 nTTList에 저장된 DID 값이 7인 XML 문서의 내용은 연관성 검사가 끝난 후, sI와 sMM에 속하지 않으므로 삭제한다.

또한, 동일한 하부 XML 문서의 중복된 변경에 의해 같은 엘리먼트의 변경사항이 nTTList에 저장될 경우, 가장 최근에 변경된 내용을 XML 질의 캐쉬에 반영해야 한다. 즉, DID와 EID가 같은 엘리먼트가 nTTList에 저장되었을 경우, LSN이 가장 큰 변경 사항을 XML 질의 캐쉬의 일관성 유지에 반영한다. 예를 들면, 중복 변경된 DID 값이 11이고 EID 값이 4인 "ship_date"의 변경은 변경로그의 LSN이 9(입력)와 LSN이 12(수정)일 때, 두번 발생하지만 XML 질의 캐쉬의 갱신에 사용될 변경은 LSN이 큰 12의 "ship_date"이 선택된다. 동일한 하부 XML 문서의 중복된 변경을 포함하는 (그림 5)의 변경로그의 예에 대해 변경로그의 레코드를 하나씩 순서대로 처리하여 최적화 작업을 마친 최종 nTTList는 (그림 6)과 같다. (그림 6)은 XML 질의 캐쉬의 CacheID 저장 컬럼에 LSN을 임시로 저장하고 비교하여 가장 최근에 변경된 내용이 XML 질의 캐쉬에 반영될 수 있도록 함을 보여준다. 전체 13개의 변경로그 레코드 중 XML 질의 캐쉬와 연관된 11개의 변경로그 레코드가 SQL 문으로 변경되어 XML 질의 캐쉬의 일관성 유지에 반영되는 RX-S와 달리 RX에서는 7개만 반영되기 때문에 3절에서 살펴본 T_{rel}, T_{exe} 이 감소하게 된다. 이와 같은 차이는 실제로 많은 하부 자료의 변경을 저장한 변경로그를 처리할 때 더욱 크게 나타날 수 있다.

동일한 하부 XML 문서의 중복 변경에 대한 XML 질의 캐쉬의 일관성 유지에 대해 RX-S 는 시간 순으로 저장된 변경로그의 내용 중 XML 질의 캐쉬와 관련된 내용을 처음부터 차례로 SQL 문으로 변경한다. 따라서, 변경로그로부터 XML 질의 캐쉬와 관련된 변경로그 레코드의 개수만큼

LSN	DID	EID	Ename	Content	ParentID
8	3	2	ship_date	2005-03-10	1
9	11	2	customer_id	23	1
12	11	4	ship_date	2005-03-15	1
9	11	5	total	6721.4	1

(그림 6) 동일한 하부 XML 문서의 중복 변경이 제거, 여과된 최종 nTTList

SQL 문을 생성한다. 그러나, 본 논문의 RX에서는 동일한 하부 XML 문서의 중복 변경에 대한 최적화를 수행하여 중복된 변경을 제거 또는 여과하기 때문에 최종적인 변경 효과만을 XML 질의 캐쉬에 반영할 수 있다. 이렇게 해도 질의 캐쉬의 일관성 유지의 정확성에는 아무 문제가 없다.

본 논문에서는 XML 질의 캐쉬 즉 XML 실체뷰의 형태로 질의 조건을 만족하는 XML 문서들로부터 검색된 결과 엘리먼트들을 하나의 루트 엘리먼트로 묶는 XML 문서를 채택하였다. ((그림 7) 참조). 'qcache' 엘리먼트는 XML 질의 캐쉬의 루트 엘리먼트가 된다. 'qdocument' 엘리먼트는 XML 질의의 정의를 만족하는 XML 문서로부터 검색한 내용을 나타내고, 't_i'(i = 1, 2, 3, 4, ..., n) 엘리먼트는 질의의 결과 엘리먼트를 나타낸다. (그림 8)은 (그림 7)의 XML 질의 캐쉬 문서 틀을 사용하여 생성한 결과 XML 문서이다.

```
<qcache>
    <qdocument>
        <t1> ... </t1>
        <t2> ... </t2>
        ...
    </qdocument>
    ...
    <qdocument>
        <t1> ... </t1>
        <t2> ... </t2>
        ...
    </qdocument>
....
</qcache>
```

(그림 7) XML 질의 캐쉬 문서 틀

```
<qcache>
    <qdocument>
        <customer_id> 27 </customer_id>
        <ship_date> 2005-03-10 </ship_date>
        <total> 2325.69 </total>
    </qdocument>
    <qdocument>
        <customer_id> 23 </customer_id>
        <ship_date> 2005-03-15 </ship_date>
        <total> 6721.4 </total>
    </qdocument>
</qcache>
```

(그림 8) 결과 XML 문서

4.2.3 갱신을 위한 SQL문 생성의 최적화

ORDBMS에 저장된 XML 질의 캐쉬를 갱신하기 위해 사용되는 SQL문의 개수를 줄이는 것은 성능 향상에 많은 도움을 준다. XML 질의 캐쉬를 갱신하는 SQL문의 최적화를 위해 본 논문에서 제안하는 방법은 다음과 같다.

첫째, 대용량 삽입(bulk insert)문을 사용한다. 대용량 삽입문은 입력할 테이블의 튜플을 파일에 저장한 후, 파일에 저장된 튜플들을 목적 테이블에 일괄 입력한다. 파일은 사용자가 정의하는 구획 문자 사이에 입력할 내용을 넣는 구조를 갖는다. 이때, 파일의 구조는 삽입할 목적 테이블의 구조와 같아야 한다. 따라서 본 논문에서 대용량 삽입에 사용할 파일의 구조는 XML 질의 캐쉬 테이블의 구조와 같다.

둘째, 병합(MERGE)문을 사용한다. 병합문은 SQL-1999에 첨가된 표준 SQL의 기능으로 두 개의 테이블을 병합할 때 사용한다. 파일이나 테이블에서 기본 키(primary key)에 해당하는 키를 등록한 후, 목적 테이블에 이미 해당 자료가 존재하여 해당 키가 목적 테이블의 키와 일치하면 원천 테이블(source table)의 내용을 목적 테이블(target table)의 내용으로 수정하고, 기본 키가 일치하지 않으면 원천 테이블의 내용을 목적 테이블에 삽입한다. 원천 테이블로 사용할 수 있는 자료는 연관된 INSERT와 MODIFY/M 관련 변경을 저장하는 nTTList와 같은 테이블이나 MODIFY/I 관련 변경을 처리할 경우 하부 자료를 검색한 SQL의 결과 값이다. 대용량 삽입문에서 SQL문의 결과 값을 원천 테이블로 사용할 경우 SQL 검색 결과 튜플을 사용하여 바로 목적 테이블을 변경할 수 있다.

셋째, DID OR-ing을 사용한다. XML 질의 처리 결과와 연관된 XML 문서의 DID들을 OR로 묶어 해당 SQL문의 where절에서 사용한다. 이렇게 함으로써 여러 DID와 관련된 SQL문들을 하나의 문장으로 처리한다.

이와 같이 XML 질의 캐쉬의 일관성을 유지하기 위한 SQL문의 생성을 최적화하면 최대 3개의 SQL문만 생성하면 된다. 이러한 최적화를 요약하면 <표 4>와 같다. 이와 같이 생성된 SQL문의 예는 (부록 1)에 수록하였다. 연관성 검사의 결과로 nTTList에 저장된 INSERT와 MODIFY/M 관련 변경은 대용량 삽입문을 사용하여 처리되고, DELETE와 MODIFY/D 관련 변경은 DID를 OR-ing한 하나의 DELETE 문을 사용하여 처리된다. 하부 자료를 검색하여 XML 질의 캐쉬에 저장될 새로운 결과를 얻는 MODIFY/I의 경우도 하나의 SQL문을 사용하여 처리된다.

<표 4> XML 질의 캐쉬의 일관성 유지를 위한 SQL문 생성 최적화

연관성 유형	사용된 특성	관련 SQL 문
연관된 INSERT & MODIFY/M	Table merge & Bulk Insert	INSERT & UPDATE
MODIFY/I	Table merge & DID OR-ing	INSERT
연관된 DELETE & MODIFY/D	DID OR-ing	DELETE

5. 실험 및 분석

본 절에서는 실험을 통해 본 논문에서 제안한 RX가 기존 연구 [11]의 RX-S에 비해서 많은 성능 향상이 있고, 재생성 기법과의 비교에서도 큰 성능 향상이 있었음을 보인다.

5.1 구현 환경

실험을 위하여 ORDBMS를 사용하여 XML문서를 저장하

```
procedure RemoveDuplicateUpdate(Set sI, Set sD, Set sMI, Set sMM, Set sMD){
  /* NTT(New Tuple Table) : sI 및 sMM 유형의 튜플들을 저장하는 메모리 내 테이블이며 구조는 XML 질의 캐쉬의 구조와
  동일하다. */
   nTTList = NULL;
   uLogList = readLogFromFile();

   /* Ui 는 uLogList에 저장된 i 번째의 로그 레코드를 뜻한다. */
   while(Ui ∈ uLogList) {
         /* checkRelevance 함수는 질의 캐쉬와 관련된 DIDList를 변경한다. */
         Ui.Type = checkRelevance(Ui, DIDList, Definition);
         SWITCH (Ui.Type) {
            CASE (INSERT) :
                  sI = sI ∪ { Ui.did };                                    (가)
                  nTTList = genInsTuple();
            CASE (MODIFY-I) :
                  sMI = sMI ∪ { Ui.did };                                  (나)
            CASE (MODIFY-M) :
                  IF (Ui.did ∈ sMI) break;
                  IF(Ui.did ∉ sI) sMM = sMM ∪ { Ui.did };                  (다)
                  nTTList = genMmTuple();
            CASE (DELETE) :
                  IF (Ui.did ∈ sI) sI = sI - { Ui.did };                   (라)
                  ELSE IF (Ui.did ∈ sMI) THEN sMI = sMI - { Ui.did };
                                          ELSE sD = sD ∪ { Ui.did };
            CASE (MODIFY-D) :
                  IF (Ui.did ∈ sI) sI = sI - { Ui.did };                   (마)
                  ELSE IF (Ui.did ∈ sMI) THEN sMI = sMI - { Ui.did };      (바)
                                          ELSE sMD = sMD ∪ { Ui.did };
         } /* end of SWITCH */
         Ui ++;
   }
}
```

(그림 9) 동일 문서 중복 변경의 제거, 여과를 지원하는 변경로그 처리 알고리즘

고 XML 질의 캐쉬를 지원하는 시스템을 기법 RX-S 및 RX 그리고 재생성에 대해 각각 구현하였다. 본 논문의 구현은 Windows 2000 운영체제를 사용한 Intel Celeron 2.0 GHz 컴퓨터에서 수행되었으며 이 메인 메모리의 크기는 512MB이고 E-IDE 방식의 80GB의 하드디스크가 장착되었다. 하드디스크의 속도는 7200rpm이다. JDBC와 SAX 파서[22]를 사용하여 XML 문서를 ORDBMS로 사용된 Oracle9*i*에 분할 저장하였고 자바1.4로 구현하였다. 본 논문의 성능 평가에 사용된 실험 데이타는 order 문서로 이는 TPC-W benchmark[23]에서 사용된 XML 문서이다. <표 5>는 실험에서 사용된 XML 데이터와 성능 파라미터 및 파라미터 값을 나타낸 것이다[18]. 본 논문에서 측정한 질의 처리 시간은 연관성 검사를 위한 변경로그와 XML 질의 캐쉬의 부가 정보를 메모리에 적재하는 것에서부터 XML 질의 캐쉬를 갱신한 질의 결과를 XML 형태로 변경하여 파일로 저장하는 데까지 걸린 총 시간이다.

5.2 실험 결과

<표 5>에서 기술한 성능 파라미터를 사용하여 성능을 평가하는 방법은 크게 다음과 같이 구분하였다. S를 0.2와 0.3으로 설정하여 XML 질의 캐쉬에 저장되는 자료의 양을 조절하였으며, 문서 삽입과 삭제를 많이 하는 I : D : M=4 : 4 : 2와 엘리먼트의 수정을 많이 하는 I : D : M=2 : 2 : 6의 설정을 사용하여 하부 XML 문서의 변경을 제어하였다. 본 논문과 관련된 XML 질의 캐쉬의 부가 정보나 연관성 유형 등은 RX-S에 기초하여 사용된 것들이기 때문에, 본 실험에서 제안된 RX와 기존의 RX-S를 서로 비교하였고, XML 질의 캐쉬를 사용하지 않는 재생성과도 비교하여 RX의 성능 향상을 보였다.

〈표 5〉 성능 파라미터 및 값 설정

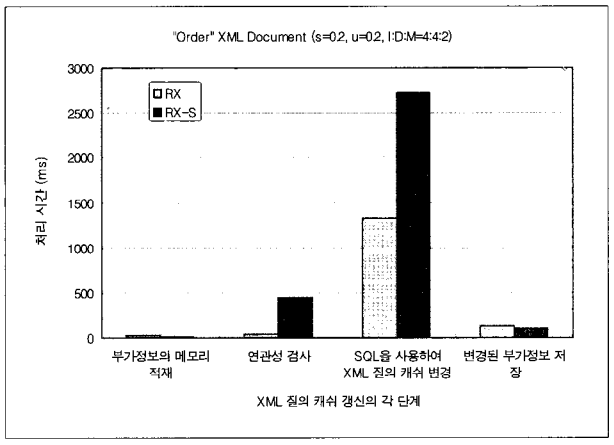
파라미터 이름	내 용	값
DN	하부 데이터에 저장된 XML 문서의 개수	10,000
U	질의의 하부 XML 문서의 변경율 (%)	1, 10, 20, 30, 40, 50
S	질의 선택율 : 질의 조건에 맞는 하부 문서 비율	0.2, 0.3
R	질의에 맞는 변경로그 레코드의 비율	0.2, 0.3
I	INSERT 변경의 비율	0.4, 0.2
D	DELETE 변경의 비율	0.4, 0.2
M	MODIFY 변경의 비율	0.2, 0.6
MI	모든 MODIFY 관련 로그 중 MODIFY-I의 비율	0.4, 0.2
MD	모든 MODIFY 관련 로그 중 MODIFY-D의 비율	0.4, 0.2
MM	모든 MODIFY 관련 로그 중 MODIFY-M의 비율	0.2, 0.6

(그림 10)은 XML 질의 캐쉬의 일관성 유지를 위한 각 단계별 처리 시간에 대해 RX와 RX-S를 비교한 것이다. order 문서 10,000개를 분할 저장한 하부 XML 문서를 사용하여 그 중 20%를 XML 질의 캐쉬에 저장하였고, 하부 자료의 변경을 20%의 문서에 대해 발생시켜 성능을 측정하였다. 단계별 처리 시간에서 XML 질의 캐쉬와 연관된 부가 정보의 메모리 적재와 변경된 부가 정보의 저장 시간은 차이가 거의 없었지만 연관성 검사와 SQL문을 사용하여 ORDBMS에 저장된 XML 질의 캐쉬를 변경하는 시간은 많은 차이를 보였다. 이러한 성능 향상의 결과는 RX-S에 비해 개선된 자료구조의 변경로그를 사용하였고, 동일 XML 문서에 대한 중복 변경을 제거 또는 여과하여 처리하였기 때문이다. 또한, XML 질의 캐쉬의 갱신에 최적화된 3개의 SQL문만 사용하기 때문에 RX의 SQL문을 사용하여 갱신하는 시간이 RX-S에 비해 상당히 빠름을 알 수 있다. (그림 11)은 RX의 경우 XML 질의 캐쉬 갱신을 위한 SQL문의 처리 시간을 연관성 유형 별로 나타낸 것이다.

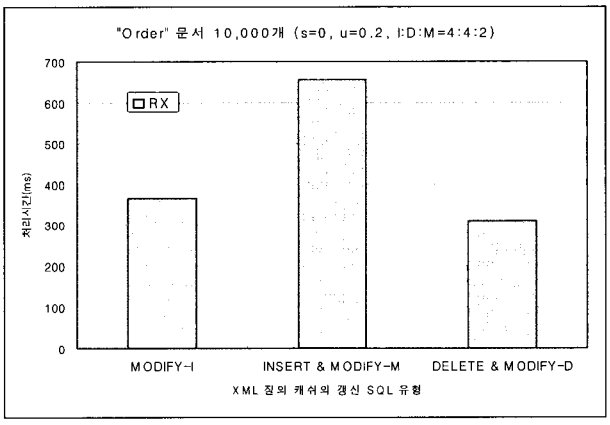
(그림 12)는 하부 문서 변경 비율 중 엘리먼트의 수정이 많은 경우를 실험한 결과이고, (그림 13)은 하부 문서 변경 비율 중 문서의 삽입과 삭제가 많은 경우를 실험한 결과이다. XML 문서의 삽입이 발생하면 4.2.1절에서 기술한 것과 같이 변경로그가 커진다. 따라서 엘리먼트의 수정이 많은 경우와 비교할 때, 변경로그의 메모리 적재 시간과 변경로그의 내용에 대한 연관성 검사의 시간이 상대적으로 증가한다. 따라서, (그림 12)와 (그림 13)의 (a)에서 볼 수 있듯이, I : D : M=2 : 2 : 6의 경우에 RX는 재생성과 비교할 때 변경율이 약 38%까지 증가할 때까지 빠른 처리 속도를 보이는 반면, S=0.2의 동일한 선택율을 갖지만 I : D : M=4 : 4 : 2인 경우에는 RX는 재생성과 비교할 때 하부 문서 변경율이 약 25%까지 증가하는 동안 향상된 처리 결과를 보였다.

(그림 12)와 (그림 13)에서 하부 문서의 변경율이 5% 이하로 매우 작은 경우 RX-S는 RX보다 빠른 처리 결과를 나타낸다. 이는 RX의 최적화 처리가 부담으로 작용하여 RX의 처리 시간이 늦어지는 원인이 된다는 것을 보여준다. 그러나, 변경율이 증가할수록 RX의 성능은 RX-S에 비해 월등히 향상됨을 나타낸다. 이것은 본 논문에서 제안한 RX가 하부 자료의 변경에 대해 XML 질의 캐쉬의 지연 갱신 시간을 효율적으로 단축시킬 수 있었기 때문이다. 또한 RX는 재생성에 비해서도 탁월한 성능을 보였다.

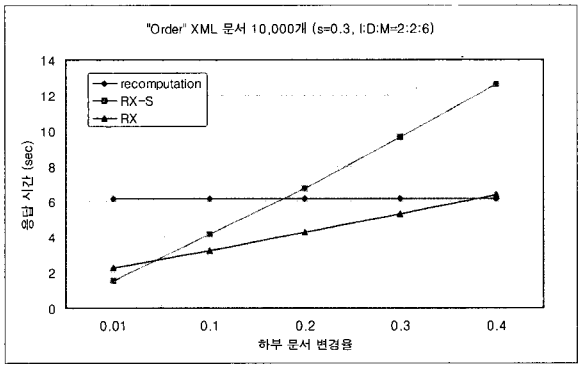
이상의 성능 평가 분석에서 보았듯이 본 논문이 제안하는 효율적인 지연 갱신 방법을 사용한 RX 기법의 XML 질의 캐쉬 일관성 유지는 RX-S 기법을 사용한 경우와 비교하여



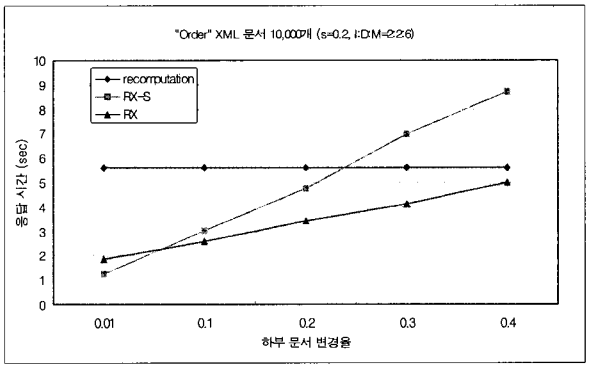
(그림 10) XML 질의 캐쉬 갱신의 각 단계별 처리 시간 비교



(그림 11) XML 질의 캐쉬 갱신을 위한 SQL 문의 연관성 유형별 처리 시간

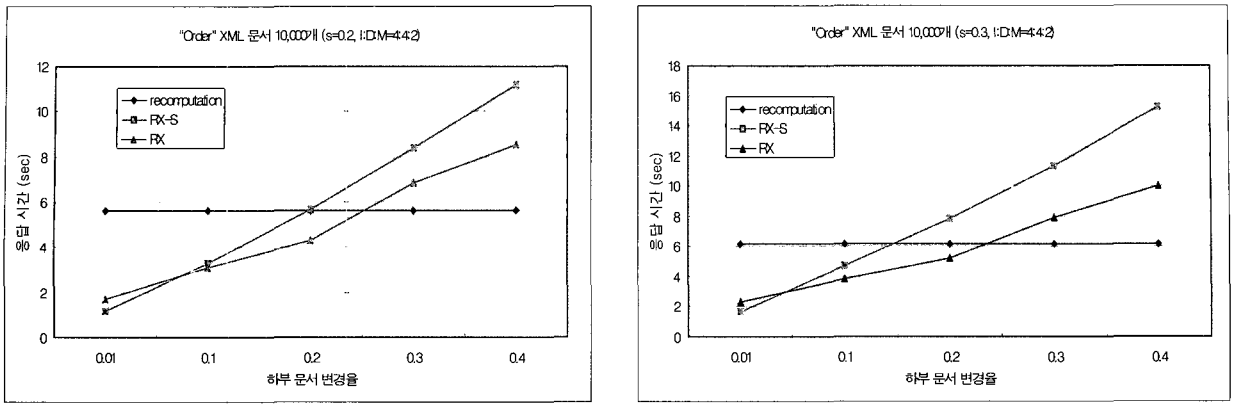


(a) s=0.2



(b) s=0.3

(그림 12) 하부 문서 변경율의 증가에 따른 비교 (I:D:M=2:2:6)



(a) s=0.2 (b) s=0.3

(그림 13) 하부 문서 변경율의 증가에 따른 비교 (I:D:M=4:4:2)

연관성 검사와 XML 질의 캐쉬의 갱신 측면에서 모두 나은 성능을 나타냈다. 그러나 RX는 변경로그 레코드의 처리를 최적화하기 위해 중간 결과를 생성해야 하고, 동일한 하부 XML 문서가 중복되었는지 검사하는 추가 비용을 지불한다. 만약 하부 XML 문서의 변경율이 극히 낮고, 동일한 하부 XML 문서의 중복된 변경이 거의 없는 경우라면, 오히려 RX-S 기법을 사용하는 것이 더 바람직하다. 그러나 XML 문서는 변경 가능하고 자주 사용되는 XML 문서는 중복 변경될 확률도 높다는 점에서 본 논문에서 제안하는 RX 기법의 효용성을 확인할 수 있다.

6. 결론 및 향후 연구

본 논문에서는 XML 문서를 ORDBMS에 분할 저장하고 빠른 질의 처리를 위해 XML 질의 캐쉬를 역시 ORDBMS에 저장하는 환경에서 하부 자료의 변경을 기록한 변경로그를 사용하여 XML 질의 캐쉬의 효율적인 지연 갱신을 수행하는 기법을 제시하였다. 이를 위하여 첫째, 지연 갱신의 비용 모델을 정의하고, 이를 기반으로 각 지연 갱신의 처리 단계마다 효율성을 제고할 수 있는 방법을 사용하였다. 둘째, 파일에 저장된 변경로그를 메모리에 바로 적재하여 연관성 검사에 사용할 수 있도록 하였다. 셋째, 하부 자료의 변경 중 동일 XML 문서의 중복 변경이 지연 갱신을 비효율적으로 만드는 문제점을 해소하기 위해 동일한 하부 XML 문서의 중복 변경을 제거, 여과하는 알고리즘을 제시하였다. 넷째, 이를 통해 ORDBMS에 저장된 XML 질의 캐쉬 내용을 변경하기 위한 최적화된 SQL 문을 생성하여 사용하였다. 마지막으로 본 논문에서 제시한 지연 갱신의 기법의 효율성을 검증하기 위해, 실험을 통하여 본 논문에서 제시하는 기법과 기존 연구 [11]의 기법을 비교할 때 성능이 향상됨을 보였고, 재생성에 비해서도 성능이 크게 향상됨을 보였다.

향후 연구로는 하부 XML 문서의 변경으로 엘리먼트 단위의 삽입 및 삭제를 지원하는 기법 및 복잡한 XML 질의의 캐쉬를 지원하는 기법으로 본 논문의 결과를 확장하는 연구가 필요하다. 또한 XML 질의 캐쉬의 저장을 ORDBMS가 아닌 텍스트 파일이나 PDOM(persistent DOM) 파일에 저장함으로써 캐쉬의 갱신 및 질의 결과의 반환의 효율성을 제고하는 기법의 연구도 필요하다.

(부록 1) XML 질의 캐쉬의 갱신에 사용된 SQL문의 예

```
// bulk insert 시 해당 디렉토리를 변수에 선언한다.
CREATE OR REPLACE DIRECTORY dat_dir AS
\'c:/oracle/Oradata/Data
CREATE OR REPLACE DIRECTORY log_dir AS
\'c:/oracle/Oradata/Log
CREATE OR REPLACE DIRECTORY bad_dir AS
\'c:/oracle/Oradata/Bad

// 사용자 ID인 hwang에 해당 디렉토리의 접근을 허용한다.
GRANT READ ON DIRECTORY dat_dir TO hwang
GRANT WRITE ON DIRECTORY log_dir TO hwang
GRANT WRITE ON DIRECTORY bad_dir TO hwang

// 위에 선언한 변수를 사용하여 'bulk'라는 이름을 갖는 table을
만든다.
create table bulk (viewid number(2), did number(10), eid
number(10),
ename varchar2(32), parentid number(10), content varchar2(4000))
ORGANIZATION EXTERNAL
( TYPE ORACLE_LOADER
DEFAULT DIRECTORY dat_dir
ACCESS PARAMETERS
( records delimited by newline
badfile bad_dir:'bulk%a_%p.bad'
logfile log_dir:'bulk%a_%p.log'
fields terminated by '|'
missing field values are null
(viewid, did, baseEid, ename, parentid, content)) LOCATION
('bulk.insert') )
PARALLEL 4
REJECT LIMIT UNLIMITED
```

(그림 14) bulk insert 준비

```
MERGE INTO viewelem v
USING bulk b ON(v.eid = b.eid and v.did = b.did)
WHEN MATCHED
THEN UPDATE SET v.content = b.content
WHEN NOT MATCHED
THEN INSERT VALUES (b.viewid, b.DID, b.eid, b.Ename,b.parentid, b.content)
```

(그림 15) INSERT와 MODIFY-M 연관성 유형과 관련된 SQL문의 예

```
MERGE INTO viewelem v
USING (select * from xmlelem where ((did = 6160) or (did = 7539) or (did = 5382) or .....) and ((ename =
'customer_id') or (ename = 'order_date') or (ename = 'subtotal'))) b
ON (v.eid = b.eid and v.did = b.did)
WHEN MATCHED
THEN UPDATE SET v.content = b.content
WHEN NOT MATCHED
THEN INSERT VALUES (viewID,b.DID,b.eid, b.Ename, b.parentId, b.content)
```

(그림 16) MODIFY-I 연관성 유형과 관련된 SQL문의 예

```
delete from viewelem where did = 1242 or did = 2498 or did = 427 or did = 92 or did = 1951 or did = 115 or did
= 1927 or did = 2665.....
```

(그림 17) MODIFY-D와 DELETE 연관성 유형과 관련된 SQL문의 예

참 고 문 헌

[1] J. Shanmugasundaram et al., "Relational Databases for Querying XML Documents : Limitations and Opportunities," Proc. Int'l Conf. on VLDB, 1999.

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

[3] 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, pp.110-141, Aug., 2001.

[4] Tamino : http://www2.softwareag.com/corporate/products/tamino/default.asp.

[5] X-Hive: http://www.x-hive.com/.

[6] Oracles XML SQL Utility, http://www.oracle.com/technology/tech/xml/index.html.

[7] DB2 XML Extender, http://www-4.ibm.com/software/data/db2/extenders/xmlext/.

[8] L. Xyleme, "A dynamic warehouse for XML data of the Web," IEEE Data Eng. Bulletin Vol.24, No.2, pp.40-47, 2001.

[9] Xyleme, http://www.xyleme.com, 2005.

[10] I. Mumick, D. Quass and B. Mumick, "Maintenance of Data Cubes and Summary Tables in a Warehouse," Proc. ACM SIGMOD Int'l Conf. on Management of Data, 1997.

[11] 임재국 외, "점진적 갱신에 기반을 둔 XML 형성뷰 관리 프레임워크," 정보처리학회논문지D, 제8-D권 제4호, pp.327-338, 2001.

[12] J. Hammer, Hector Garcia-Molina, J. Widom, W.J. Labio, Y, Zhuge, "The Stanford Data Warehousing Project." IEEE Data Engineering Bulletin, June 1995.

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

[14] Y. Zhuge, H. Garica-Molina. "Graph Structured Views and Their Incremental Maintenance," Proc. Int'l Conf. on Data Engineering, 1998.

[15] S. Abiteboul et al., "Incremental Maintenance for Materialized Views over Semistructured Data," Proc. Int'l Conf. on VLDB, pp.38-49 1998.

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

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

[18] 성호상 외, "XML 실체뷰 갱신 기법의 성능 평가," 정보처리학회논문지D, 제10-D권 제3호, pp.387-398, 2003.

[19] S. Abiteboul, "On Views and XML," Proc. ACM Symp. on Principle of Database System, pp.1-9, 1999.

[20] L. Chen and E. Rundensteiner, "ACE-XQ : A CachE-aware XQuery Answering System," Proceedings of the 5th Int'l Workshop on the Web and Databases (WebDB), pp.31-36, 2002.

[21] A. Halevy, "Answering Queries Using Views : A Survey," The VLDB Journal, Vol.10, No.4, pp.270-294, Dec., 2001.

[22] SAX (Simple API for XML) http://sax.sourceforge.net/.
[23] http://www.tcp.org.

황 대 현

e-mail : dhhwang@dblab.cse.cau.ac.kr
1992년 중앙대학교 컴퓨터공학과(학사)
1995년 중앙대학교 대학원 컴퓨터공학과 (석사)
1995년~2000년 ㈜한전KDN
2000년~2002년 ㈜뉴스디자인
2003년~현재 중앙대학교 컴퓨터공학부 박사과정
관심분야 : XML 실체뷰, XML 변경, 웹 데이터베이스 등

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 컴퓨터공학부 교수
관심분야 : XML 및 웹 데이터베이스, 이동 데이터베이스, Stream 데이터 관리 등