XML 실체뷰 갱신 기법의 성능 평가

성 호 상[†] · 문 찬 호[†] · 강 현 철[††]

요 약

실체뷰는 질의 처리의 성능 향상을 위한 수단으로 널리 연구되어 왔다. 실체뷰는 하부 데이타가 변경되었을 경우 일관성을 유지해야 하는데, 그 기법으로는 뷰를 하부 데이타로부터 재생성하는 방법과 변경 내용 중 뷰와 관련 있는 것만 반영하는 점진적 갱신이 있다. 최근 XML이 웹 데이타 교환의 표준으로 대두되면서 XML 문서를 효율적으로 저장하고 검색하는 기법에 관한 연구가 활발히 수행되고 있다. 본 논문은 XML 문서를 기존의 관계 DBMS에 저장할 경우, 자주 제기되는 질의의 결과를 XML 실체뷰로 지원하고 그것을 점진적으로 갱신하는 기법의 성능 평가에 관한 것이다. XML 실체뷰 및 그것의 점진적 갱신을 지원하는 XML 저장 시스템의 구현에 대해 기술하고, XML 실체뷰의 점진적 갱신 기법의 성능을 실험을 통해 평가하였다. 실험 결과, 자주 제기되는 질의에 대해 매번 그 결과를 재생성하는 것보다 그 결과를 실체뷰로 유지하면서 뷰의 하부 데이타에 대한 변경 중 뷰와 연관성이 있는 것만을 점진적으로 반영하여 일관성을 유지하는 실체뷰 기법이 XML 데이타의 검색에도 효율적이라는 것을 확인하였다.

Performance Evaluation of XML Materialized View Refresh

Hosang Sung[†] · ChanHo Moon[†] · Hyunchul Kang[††]

ABSTRACT

Materialized views have received much attention for query performance improvement. They need to be refreshed whenever their underlying data sources are updated. They could be recomputed from scratch or they could be incrementally refreshed by reflecting only those portions of updates that affect them. With emergence of XML as the standard for data exchange on the Web, active research is under way on effectively storing and retrieving XML documents. In this paper, we describe a performance study on the incremental refresh of XML materialized views for the case where XML documents are stored in a relational DBMS, and XML materialized views are maintained with incremental refresh. We describe implementation of a prototype XML storage system that supports XML materialized views and their incremental refresh, and report the performance results obtained with the implemented system through a detailed set of experiments on the incremental refresh of XML materialized views. The results show that the XML view maintenance with incremental refresh outperforms the ordinary view recomputation.

키워드 : XML, 실체뷰(materialized view)

1. 서 론

XML(Extensible Markup Language)[1] 이 인터넷 상에서 데이타 교환의 표준으로 널리 확산되고 있는 가운데 전자상거래를 비롯한 여러 응용 분야에서 XML 데이타의 저장과 검색 기능이 필요하게 되었다. 이에 XML 문서의 저장과 검색, 특히 기존의 관계 DBMS를 이용한 저장과 검색에 관한 연구가 활발히 수행되고 있다[2-4]. 또한 기존의 관계 데이타베이스에 저장되어 있는 데이타를 전자상거래 등의 웹 응용에서 XML 데이타로 요구하고 있는데 이를 충족시키기 위한 XML 출판에 관한 연구도 활발히 수행되고 있다[5-9]. 이러한 이유로 여러 DBMS 벤더들도 그들의 제품이 XML을 지원할 수 있도록 확장하고 있다[10-12]. 실제 eXcelon[13]이나 Tamino[14] 같은 XML 전용 데이타베이스가 이미 존재한다. 이러한 XML 저장 시스템들에서 중요한 이슈 중의 하나는 많은 양의 XML 문서들에 대한 고성능의 검색 기법이다.

뷰는 데이타 통합과 데이타 여과 기능을 통하여 사용자가 요구하는 데이타를 제공한다. 따라서 XML 문서에 대해서도 유용한 개념이다[15-19]. 뷰는 질의 처리의 성능 향상을 위해 실체뷰(materialized view)로 유지할 수 있는데[20], 일관성 유지를 위해서는 뷰를 하부 데이타로부터 재생성하는 방법과 변경 내용 중 뷰와 관련 있는 것만 반영하는 점진적 갱신 기법이 있다.

본 논문은 XML 실체뷰의 점진적 갱신 기법의 성능 평가에 관한 것이다. XML 문서를 저장하기 위하여 기존의 관계 DBMS를 사용할 경우, [21]의 연구에서 제시된 XML 실체뷰의 점진적 갱신 알고리즘을 바탕으로 실체뷰를 지원하는 XML 저장 시스템을 구현하고, 이 시스템에서 XML 뷰 검색을 위하여 기존에 생성된 실체뷰를 점진적으로 갱

※ 본 연구는 한국과학재단 목적기초연구(R01-2000-000-00272-0)지원으로 수행되었음.
† 준 회 원 : 중앙대학교 대학원 컴퓨터공학과
†† 종신회원 : 중앙대학교 컴퓨터공학과 교수
논문접수 : 2002년 8월 22일, 심사완료 : 2003년 1월 6일

신하는 기법의 성능을 해당 뷰를 재생성할 때의 성능과 실험을 통하여 비교 평가하였다.

본 논문의 구성은 다음과 같다. 2절에서는 [21]을 기반으로 XML 실체뷰의 점진적 갱신 알고리즘을 기술한다. 3절에서는 실체뷰를 지원하는 XML 저장 시스템의 구현에 대해 기술하고, 4절에서는 구현된 시스템을 이용한 여러 가지 다양한 실험 결과를 기술한다. 5절에서는 결론을 맺고 향후 연구 과제를 제시한다.

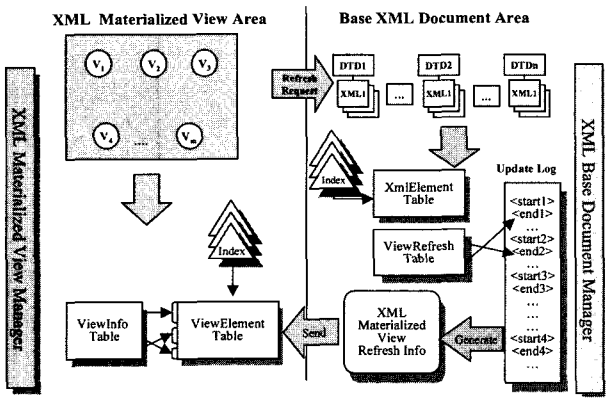
2. XML 실체뷰 점진적 갱신 알고리즘

본 절에서는 3절에서 기술할 실체뷰를 지원하는 XML 저장 시스템의 구현을 위한 자료구조 및 실체뷰의 점진적 갱신 알고리즘의 개요를 기술한다(이들은 [21]에서 제시된 것을 바탕으로 하였으므로 보다 상세한 내용은 [21]을 참조).

2.1 XML 뷰 검색 모델

실체뷰를 지원하는 XML 저장 시스템은 (그림 1)과 같이 하부 XML 문서 영역과 XML 실체뷰 영역으로 구성되고 이들 영역은 XML 하부 문서 관리 모듈과 XML 실체뷰 관리 모듈에 의해 각각 관리된다[21]. 하부 XML 문서 영역에는 다수의 DTD와 XML 문서, 이들의 검색을 위한 인덱스, XML 문서의 변경을 기록하기 위한 로그 화일과 각 실체뷰의 점진적 갱신을 위해 필요한 정보가 저장된다. 그리고 실체뷰 영역에는 뷰에 대한 정보, 실체뷰 및 이들의 검색을 위한 인덱스가 저장되어 있다.

XML 실체뷰의 점진적 갱신은 지연(deferred) 수행된다. 즉, 하부 XML 문서의 변경이 발생하면 실체뷰에 즉각 반영되지 않고, 변경 내용만을 로그 화일에 기록한다. 이후 사용자로부터 뷰가 요청되었을 때, 실체뷰 관리 모듈은 하부 문서 관리 모듈에게 해당 뷰를 갱신하기 위한 갱신정보를 요청한다. 하부 문서 관리 모듈은 변경로그로부터 해당 실체뷰에 대한 갱신정보를 생성하고 이를 반환한다. 갱신정보를 전송 받은 실체뷰 관리 모듈은 해당 실체뷰에 대한 점진적 갱신을 수행한 후 사용자에게 뷰를 제공한다.



(그림 1) XML 실체뷰 관리 기법

2.2 XML 문서 저장 구조

XML 문서를 현재 가장 널리 사용되고 있는 관계 DBMS에 XML 문서를 저장하려는 시도는 많이 연구되어 왔다[2-4]. 본 논문의 XML 문서 저장 구조도 관계 DBMS를 기반으로 하였으며 XML 문서 정보를 저장하는 Doc_Info테이블, DTD를 저장하고 있는 Dtd_Info테이블, 그리고 엘리먼트 단위로 XML 문서를 저장하고 있는 XmlElement라는 세 개의 테이블로 구성되어 있다. (그림 2)는 이들 테이블의 스키마를 나타낸 것으로 [2]에서 분류한 edge-inlining 접근법을 바탕으로 한 것이다.

Doc_Info(XML 문서 정보 테이블)

DID	Doc_Name	DTDID	createdate
문서 ID	문서명	DTD ID	저장날짜

Dtd_Info(DTD정보 테이블)

DTDID	Doc_type	content	contsize	createdate
DTD ID	문서형식	DTD내용	DTD크기	저장날짜

XmlElement(XML 문서 저장 테이블)

DID	DTDID	EID	Ename	Content
문서 ID	DTD ID	엘리먼트 ID	엘리먼트이름	내 용

(그림 2) XML 하부 문서 저장 스키마

2.3 XML 실체뷰 저장 구조

본 논문에서는 XML 질의어의 구문으로 표현 가능한 뷰 중에서 단일 DTD를 준수하는 XML 문서들을 대상으로 생성되는 XML 뷰 만을 고려하였다. 뷰에 대한 정보와 실체뷰는 ViewInfo 테이블과 ViewElement 테이블에 각각 저장된다. ViewInfo 테이블에는 각 XML 실체뷰마다 한 개의 레코드가 존재하며 각 XML 실체뷰에 부여되는 뷰의 ID(ViewID), 뷰의 생성 조건을 정의한 뷰 정의(ViewDef), 뷰가 참조하는 DTD의 ID 등으로 구성된다. ViewElement 테이블은 View ID, DID, BaseEID, Ename, ViewEID, Content 컬럼으로 구성된다. ViewID는 ViewInfo 테이블의 ViewID를 참조하는 외래키이다. DID는 뷰를 생성한 하부 XML 문서의 ID를 나타낸다. ViewEID는 실체뷰를 구성하는 엘리먼트의 ID를

ViewInfo(실체뷰 정보테이블)

ViewID	ViewDef			DTDID
	CE	Cond	TE	
뷰 ID	조건 엘리먼트	뷰의 조건	추출 엘리먼트	뷰 참조DTDID

ViewElement(실체뷰 엘리먼트테이블)

ViewID	DID	BaseEID	Ename	ViewEID	Content
뷰 ID	하부문서 ID	엘리먼트 ID	엘리먼트 이름	뷰 엘리먼트 ID	내 용

(그림 3) XML 실체뷰 저장 스키마

나타낸다. BaseEID는 DID가 가리키는 하부 XML 문서의 해당 엘리먼트 ID를 나타낸다. Content에는 실제 XML 뷰를 구성하는 데이타가 저장된다.

2.4 XML 문서 변경 모델

XML 문서는 문서 단위, 엘리먼트 단위, 속성 단위로 변경될 수 있다. 본 논문에서는 문서 및 엘리먼트 단위의 변경 만을 고려하였으며, 수정은 XML 문서 내에서 값을 갖는 엘리먼트에 대해 엘리먼트 단위로 수행되고, 삽입과 삭제는 문서 단위로 수행된다고 가정하였다.

XML 하부 문서 관리 모듈은 XML 문서가 변경될 때마다 XML 실체뷰의 점진적 갱신을 지원하기 위해서 그 내용을 로그 화일에 기록한다. 변경로그 레코드의 자료구조는 (그림 4)와 같다.

```
<StartUpdateLog, DTDID, DID, ObjType, OpType>
[<BaseEID, Ename, Content>[,< BaseEID, Ename, Content>, ···]]
<EndUpdateLog>
```

(그림 4) 변경로그 레코드 자료구조

변경로그 레코드는 <StartUpdateLog> 필드로 시작해서 <EndUpdateLog> 필드로 끝나는 블록구조를 갖는다. DTD ID와 DID는 변경이 발생한 XML 문서의 DTD와 문서 ID를 각각 나타낸다. ObjType은 변경의 단위가 엘리먼트('ELEMENT')인지 문서('DOCUMENT')인지를 나타낸다. OpType은 XML 문서에 대한 변경의 종류를 나타낸다. <BaseEID, Ename, Content>는 변경된 엘리먼트에 대한 정보로서 ObjType과 OpType에 따라 생략되거나 한번 또는 그 이상 기록될 수 있다. BaseEID는 변경되는 엘리먼트의 ID를 나타내고, Ename은 변경된 엘리먼트 명을 나타낸다. Content는 엘리먼트의 값을 나타내며, Optype = 'DELETE'인 경우 Content는 NULL로 설정된다.

2.5 XML 실체뷰의 점진적 갱신

실체뷰의 점진적 갱신은 첫째, 변경로그의 검색, 둘째, 그를 통한 갱신정보의 생성, 그리고 셋째, 갱신정보의 실체뷰로의 반영으로 구성된다. 변경로그의 검색은 이전의 뷰 갱신 이후에 기록된 변경로그 레코드들을 대상으로 수행된다. 새로이 검색해야할 첫 번째 변경로그 레코드의 위치는 XML 실체뷰의 점진적 갱신을 위해 필요한 정보를 저장하고 있는 ViewRefresh 테이블의 시작옵셋(LastULOffset) 컬럼에 기록되어 있다. (그림 5)는 ViewRefresh 테이블의 구조를 나타낸 것이다. 실체뷰 갱신정보의 생성은 변경로그의 검색을 통해 각 로그 레코드별로 해당 뷰에 영향을 미치는지 여부를 판단하고 <표 1>과 같은 연관성이 있을 경우 실체뷰에 반영하는데 필요한 내용으로 갱신정보를 생성한다. 이때 갱신정보는 RefType, DID, BaseEID, Ename, Vi ewEID, Content로 구성된다. RefType은 실체뷰의 갱신 유형(refresh type)을 나타내는 것으로 'MODIFY', 'INSERT', 그리고 'DELETE'의 세 가지 유형이 있다. DID는 변경된 문서의 ID, BaseEID는 변경된 하부 문서 엘리먼트의 ID, Ename은 엘리먼트 이름, ViewEID는 삽입될 뷰 엘리먼트의 ID, 그리고 Content는 실체뷰에 반영될 엘리먼트의 값을 나타낸다. 이들 중 해당 사항이 없는 것은 NULL로 설정된다. 이와 같이 뷰와 연관성이 있는 각 변경로그 레코드별로 생성된 갱신정보는 ViewElement 테이블의 해당 뷰 레코드들에 반영된다.

ViewRefresh(갱신정보 생성을 위한 정보테이블)

ViewID	DTDID	LastULOffset	DIDList
뷰 ID	뷰 관련 DTDID	시작 옵셋 값	관련 문서번호 리스트

(그림 5) 실체뷰의 점진적 갱신정보를 위한 스키마

<표 1> 연관성 유형

연관성 유형	내 용
INSERT	뷰의 조건을 만족하는 문서 삽입
DELETE	뷰의 조건을 만족하던 문서 삭제
MODIFY-M	뷰의 조건을 만족하는 문서의 추출 엘리먼트 수정
MODIFY-I	뷰의 조건을 만족하지 않던 문서의 조건 엘리먼트 값이 수정되어 뷰의 조건을 만족하게 됨
MODIFY-D	뷰의 조건을 만족하던 문서의 조건 엘리먼트 값이 수정되어 뷰의 조건을 만족하지 않게 됨

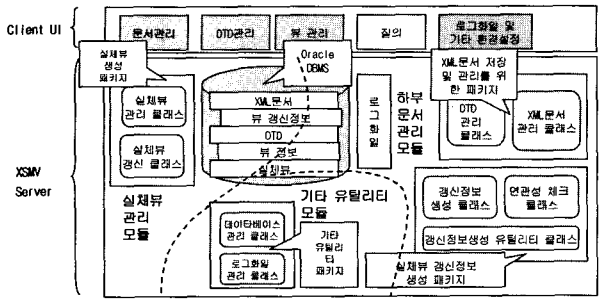
3. 실체뷰 지원 XML 저장 시스템의 구현

본 논문의 XML 저장 시스템은 DBMS로 Oracle8*i*를 사용하였고, Java로 구현하였다. 실체뷰 지원 XML 저장 시스템은 (그림 6)과 같이 크게 클라이언트 UI(User Interface) 부분과 서버측 기능을 수행하는 XSMVServer(XML Storage System Supporting Materialized Views Server)로 구성되어 있다. XSMVServer는 하부문서 관리 모듈과 실체뷰 관리 모듈 그리고 기타 유틸리티 모듈로 구성되어 있으며, 각 해당 관리 모듈은 기능별로 패키지와 클래스로 구성되어 있다. 하부 문서 관리 모듈은 XML 문서를 관리하는 패키지를 가지고 있으며, 이는 XML 문서를 저장하고, 엘리먼트 단위의 수정을 하며, 문서를 삽입하고 삭제하는 등의 기능을 수행한다. 또한, DTD를 관리하는 기능과 변경로그 기록을 담당한다. 실체뷰 갱신정보 생성 패키지는 갱신정보를 생성하기 위한 연관성 검사를 수행하며 생성한 갱신정보를 실체뷰 관리 모듈에게 반환하는 기능을 수행한다. 실체뷰 관리 모듈은 실체뷰 생성 패키지를 가지고 있으며, 뷰의 질의 요청에 대한 응답을 담당하고 해당 뷰에 대한 갱신을 하부 문서 관리 모듈에게 요청하는 기능을 수행한다. 기타 유틸리티 모듈은 데이타베이스 연결에 대한 관리와 로그 화일에 대한 관리를 수행하는 클래스 등으로 이루어져 있

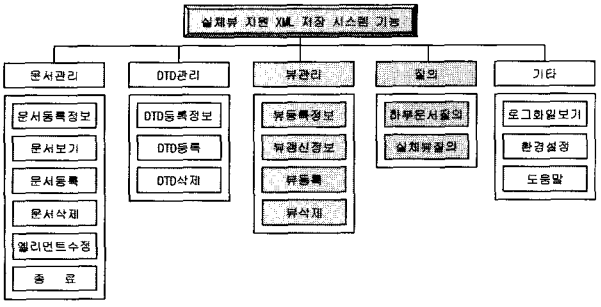
〈표 2〉 시스템의 모듈 및 주요 클래스의 기능

모 듈 명	관련 패키지 명	주요 클래스	기 능
하부 문서 관리 모듈	dblab.BXD.XMLstroage	DtdMgr	DTD 문서 관리 기능
		XmlStroageMgr	XML 문서 저장
		DocMgr	XML 문서 관리 기능
		LogMgr	변경로그 관리 기능
	dblab.BXD.refreshinfo	CreateRefreshInfo	갱신정보 생성 기능
		RelevanceCheck	뷰와의 연관성 체크 기능
		RefreshInfoUtility	갱신정보 생성 관련에 필요한 유틸리티
실체뷰 관리 모듈	dblab.MV.MaterializedView	MateralizedViewApp	실체뷰 관리 기능
		RefreshMV	생성된 갱신 정보를 반영하는 기능
		ViewMgr	실체뷰 관리에 필요한 주요 메소드 모음
기타 유틸리티 모듈	dblab.Utility	DBOpenConnect	데이타베이스 연결 기능
		ResultsModel	요청 결과 자료형 모델
		LogFileIO	변경로그 I/O 기능

다. <표 2>는 구현된 시스템의 모듈 및 패키지, 주요 클래스, 이들의 기능들을 정리하여 나타낸 것이다. 클라이언트 UI는 (그림 7)처럼 다섯 가지 메뉴를 가지고 있으며 XSM VServer의 해당 메소드를 호출함으로써 각각의 기능을 수행한다. (그림 7)은 이들을 기능 별로 정리한 것이다.



(그림 6) 실체뷰 지원 XML 저장 시스템의 구성



(그림 7) 실체뷰를 지원하는 XML 저장 시스템의 기능

4. 성능 평가

본 절에서는 XML 뷰를 검색하기 위하여 뷰를 재생성하는 방법과 하부 문서의 변경 내용 중 뷰와 관련있는 것만 반영하여 실체뷰를 점진적으로 갱신하는 방법의 성능을 3절에서 기술한 XML 저장 시스템을 사용하여 여러 가지 성능 파라미터들을 변화시키며 평가한 결과를 기술한다.

4.1 실험 환경 및 데이타

성능 평가를 위한 실험 환경과 실험 데이타는 <표 3>과 같다. 실험에 사용된 XML 문서는 그 크기 면에서 두 종류이다. 첫째는 평균 엘리먼트의 개수가 20개 정도로 크기가 작은 "영화" XML 문서(이하, "영화" 문서)이고 둘째는 평균 엘리먼트의 개수가 7,000개 이상인 크기가 아주 큰 "The Play of Shakespeare" XML 문서[22] (이하, "Play" 문서)이다.

4.2 실험의 유형

본 실험의 가장 큰 목적은 실체뷰를 재생성하는 방법과 점진적으로 갱신하는 기법의 성능이 교차하는 지점이 어디인가를 알아보는 것이다. 즉, 어떠한 조건에서 실체뷰 기법이 뷰 재생성보다 나은 성능을 보이는지 그 조건을 찾고자 하였다. 이를 위해 크게 두 가지 유형의 실험을 수행하였다. 첫째, 기존에 정의된 XML 뷰가 다시 요청되었을 시점에 변경로그에 기록되어 있는 (즉, 쌓여 있는) 하부 문서에 대한 변경 비율의 증가에 따라 실체뷰 기법이 어떤 성능을 보이는가 하는 것에 대한 실험이다. 이때 변경로그에 기록된 변경들은 해당 뷰가 마지막으로 검색된 이후에 발생한 것들로서 아직 실체뷰에 반영되지 않은 것들이다. 따라서 이들이 많아질수록 실체뷰 기법의 성능은 저하되어 갈 것이다. 둘째, 뷰의 도출 대상 소스인 하부 XML 문서의 개수의 증가에 따라 실체뷰 기법이 어떤 성능을 보이는가 하는 것에 대한 실험이다. 하부 문서량이 증가할수록 뷰 재생성에 비해 실체뷰 기법의 성능은 더 좋아질 것이다. 최근 웹 환경에서 XML 문서의 양이 급증해 감에 따라 XML 웨어하우스 서비스에 대한 연구가 수행되고 있다. 프랑스 INRIA에서 수행된 Xyleme 프로젝트[23]는 동일 명의 회사 설립을 낳았고 그 상용 서비스 준비가 진행되고 있다[24]. 기존 관계 데이타베이스에서도 그랬지만 실체뷰 기법은 데이타 웨어하우스를 지원하기 위한 핵심 기술 중의 하나이다.

위 두 가지 유형의 실험 외에, 주어진 하나의 변경로그 레코드에 대하여 해당 변경과 뷰와의 연관성 유형 별로 갱

신정보를 생성하는데 걸리는 시간을 측정하였다. 또한 실체뷰의 점진적 갱신 과정에서 수행되는 여러 단계 별 소요 시간을 측정하였다. 이들은 본 논문의 실험의 정확성을 검증하기 위한 것이다.

〈표 3〉 실험 환경 및 데이타

항 목		환경 또는 내용
실험 데이타		• "영화" XML 문서 XML 문서의 평균 엘리먼트 개수 : 20개 • "The Play of Shakespeare" XML 문서 XML 문서의 평균 엘리먼트 개수 : 7000개 출처 : http://www.ibiblio.org/bosak/
뷰 정의		• "감독" 엘리먼트 내용이 "장이모" 일때 영화 문서의 "제목", "상영시간", "수입배급사"를 추출 • "TITLE" 엘리먼트가 "Coriolanus"이란 단어를 포함하고 있는 PLAY 문서의 "TITLE", "SCNDESCR", "PLAYSUBT"를 추출
OS		Windows 2000 Server
XML 저장 시스템		3절 참조
DBMS		Oracle 8*i*
시스템	CPU	Pentium Ⅲ 1GHz
	RAM	512 MB

4.3 성능 척도 및 파라미터

본 실험의 성능 척도는 XML 질의 (즉, 뷰의 요청)를 제기한 시점부터 질의 결과 (즉, 뷰의 내용)를 생성 완료하여 디스플레이하기 전까지 걸린 시간이다. <표 4>는 실험을 위한 성능 파라미터와 내용을 나타낸 것이다. 본 실험에서는 첫째, 뷰의 대상 소스인 하부 XML 문서의 개수는 항상 일정한 값을 유지하도록 하였다. 이는 동일한 뷰를 재생성하는 데 걸리는 시간을 일정하게 유지하기 위한 것으로서 한번 뷰를 생성한 후 그 다음 번에 같은 뷰를 재생성할 때까지 발생한 문서 삽입과 삭제의 횟수를 동일하도록 제어하고 삽입되는것 중 뷰의 조건을 만족하는 것의 비율과 삭제되는 것 중 뷰의 조건을 만족하는 것의 비율을 동일하게 유지함으로써 가능하다. 둘째, 검색된 뷰 (즉, 재생성되거나 실체뷰를 점진적으로 갱신한 결과)의 크기도 항상 일정하도록 하였다. 이는 연속된 두 번의 뷰 검색 사이에 발생한 변경 중 뷰와 연관성이 있는 변경의 종류와 비율을 다음과 같이 제어함으로써 가능하다. ① <표 1>의 연관성 유형 중 INSERT와 DELETE이 같은 비율로 발생한다. 그리고 ② MODIFY 유형 중에서 MODIFY-I와 MODIFY-D 유형이 같은 비율로 발생한다.

본 실험에서 가장 중요한 파라미터는 하부 문서에 대한 변경의 양이다. 본 논문에서는 XML 실체뷰의 지연 갱신 모델을 채택하고 있으므로, 뷰의 갱신 시점에서 변경로그에 기록되어 있는 변경로그 레코드가 많을수록 점진적 갱신 기법의 성능 저하가 초래된다. 이러한 점을 고려해서 본 실험에서는 변경로그 레코드의 개수와 그들의 뷰와의 연관성 여부를 결정하는 파라미터들을 상세히 제어하였다. <표 5>는 이들 파라미터의 설정치를 실험 유형별로 나타낸 것이다. 그리고 <표 6>은 <표 5>의 파라미터 설정치에 의해 실험에서 발생하게 되는 하부 문서중 뷰와 연관성이 있는 문서의 개수, 연관성이 없는 문서의 개수, 그리고 변경로그에 기록된 변경 중 연산 및 연관성 유형 별로 뷰와 연관성이 있는 것의 개수, 없는 것의 개수를 구하는 공식을 나타낸 것이다. <표 7>과 <표 8>은 두 가지 실험을 예로 들어 <표 6>의 파라미터 값의 예를 보인 것이다.

〈표 4〉 성능 파라미터

파라미터	내 용
U	XML 저장 시스템에 저장된 것으로 동일 DTD를 준수하는 하부 문서 개수
L	하부 문서에 대한 변경 비율
S	뷰 선택율(view selectivity : 하부 문서 중 뷰의 조건을 만족하는 것의 비율)
R	연관성 비율(로그 레코드 중 뷰와 연관성 있는 로그 레코드 비율)
I	변경로그 레코드 중 문서 INSERT 연산 비율
D	변경로그 레코드 중 문서 DELETE 연산 비율
M	변경로그 레코드 중 엘리먼트 MODIFY 연산 비율
MI	연관성 있는 MODIFY 연산 중 MODIFY-I 연관성 비율
MD	연관성 있는 MODIFY 연산 중 MODIFY-D 연관성 비율
MM	연관성 있는 MODIFY 연산 중 MODIFY-M 연관성 비율

〈표 5〉 파라미터 설정치

파 라 미 터	설 정 치					
	"영화" XML 문서				"The Play of Shakespeare" XML 문서	
	L 변화 실험		U 변화 실험		L 변화 실험	U 변화 실험
U	20000		5000, 10000, 15000, 20000, 25000, 30000		1000	500, 1000, 1500, 2000, 2500, 3000
L	0, 0.1, 0.2, 0.3, 0.4, 0.5		0.2		0, 0.1, 0.2, 0.3, 0.4, 0.5	0.2
S	0.2, 0.3		0.2, 0.3		0.2,	0.2
R	0.2, 0.3		0.2, 0.3		0.2,	0.2
I	0.2	0.4	0.2	0.4	0.2	0.2
D	0.2	0.4	0.2	0.4	0.2	0.2
M	0.6	0.2	0.6	0.2	0.6	0.6
MI	0.2	0.4	0.2	0.4	0.2	0.2
MD	0.2	0.4	0.2	0.4	0.2	0.2
MM	0.6	0.2	0.6	0.2	0.6	0.6
실험 결과	(그림 8, 15, 16)	(그림 9)	(그림 11, 14)	(그림 12)	(그림 10, 15, 17)	(그림 13)

〈표 6〉 연관성 관련 파라미터와 그 값을 구하는 공식

변 수	내 용
Rel_Doc	하부 문서 중 뷰 조건과 연관성 있는 문서 개수 [U * S]
UnRel_Doc	하부 문서 중 뷰 조건과 연관성 없는 문서 개수 [U - Rel_Doc]
Rel_I	뷰 조건과 연관성 있는 Insert 연산 개수 [(U * L * I) * R]
Rel_D	뷰 조건과 연관성 있는 Delete 연산 개수 [(U * L * D) * R]
Rel_M	뷰 조건과 연관성 있는 Modify 연산 개수 [(U * L * R) - (Rel_I + Rel_D)]
Rel_M_I	Rel_M 연산 개수 중 MODIFY_I 연관성 유형 개수 [Rel_M * MI]
Rel_M_D	Rel_M 연산 개수 중 MODIFY_D 연관성 유형 개수 [Rel_M * MD]
Rel_M_M	Rel_M 연산 개수 중 MODIFY_M 연관성 유형 개수 [Rel_M - (Rel_M_I + Rel_M_D)]
UnRel_I	뷰 조건과 연관성 없는 Insert 연산 개수 [(U * L * I) - Rel_I]
UnRel_D	뷰 조건과 연관성 없는 Delete 연산 개수 [(U * L * D) - Rel_D]
UnRel_M	뷰 조건과 연관성 없는 Modify 연산 개수 [((U * L) - (U * L * S)) - (UnRel_I + UnRel_D)]

〈표 7〉 하부 문서 변경 비율에 따른 실험에서 연관성 관련 파라미터 값의 예

L	0% (0)	10% (2000)	20% (4000)	30% (6000)	40% (8000)	50% (10000)
Fixed Parameter	S = 20%, U = 20000, R = 20%, I = 20%, D = 20%, M = 60%, MI = 20%, MD = 20%, MM = 60%					
Rel_I	0	80	160	240	320	400
Rel_D	0	80	160	240	320	400
Rel_M	0	240	480	720	960	1200
Rel_M_I	0	48	96	144	192	240
Rel_M_D	0	48	96	144	192	240
Rel_M_M	0	144	288	432	576	720
UnRel_I	0	320	640	960	1280	1600
UnRel_D	0	320	640	960	1280	1600
UnRel_M	0	960	1920	2880	3840	4800

〈표 8〉 하부 문서 개수에 따른 실험에서 연관성 관련 파라미터 값의 예

U	5000	10000	15000	20000	25000	30000
L	20% (1000)	20% (2000)	20% (3000)	20% (4000)	20% (5000)	20% (6000)
Fixed Parameter	S = 20%, R = 20%, I = 20%, D = 20%, M = 60%, MI = 20%, MD = 20%, MM = 60%					
Rel_I	40	80	120	160	200	240
Rel_D	40	80	120	160	200	240
Rel_M	120	240	360	480	600	720
Rel_M_I	24	48	72	96	120	144
Rel_M_D	24	48	72	96	120	144
Rel_M_M	72	144	216	288	360	432
UnRel_I	160	320	480	640	800	960
UnRel_D	160	320	480	640	800	960
UnRel_M	480	960	1440	1920	2400	2880

4.4 실험 결과

4.4.1 하부 문서 변경 비율의 변화에 따른 뷰 검색 성능 비교

본 실험에서는 하부 문서 개수(U)가 일정할 때, 하부 문서 변경 비율(L)의 변화에 따라, 뷰를 재생성하는 방법에 비해 점진적으로 갱신하는 기법의 성능이 더 효율적일 때의 조건을 찾고자 하였다. 또한, ① 뷰 선택율(S)의 변화, ② 연관성 유형들간의 비율 변화, 그리고 ③ 하부 문서의 크기 변화가 두 방법의 상대적 성능 비교에 미치는 영향을 분석하였다.

(그림 8)은 "영화" 문서를 대상으로 측정한 뷰 재생성 방법과 실체뷰 갱신 방법의 성능 비교를 나타낸 것이다. 하부 문서 개수는 문서의 삽입과 삭제가 같은 비율로 발생하도록 했기 때문에 항상 일정한 개수를 유지한다. 따라서 뷰를 재생성하는 방법에서는 검색해야 할 전체 하부 문서의 개수가 일정하게 유지됨으로써 뷰 검색 시간이 거의 일정하게 유지되는 것으로 나타났다. 반면에 실체뷰의 점진적 갱신에서는 INSERT와 DELETE 연관성 유형, 그리고 MODIFY-I와 MODIFY-D 연관성 유형이 각각 같은 비율로 발생하도록 했기 때문에 갱신이 완료된 실체뷰의 크기는 일정하지만, 하부 문서 변경의 비율이 높아질수록 변경로그 레코드 개수가 증가하므로 이들을 처리하여 갱신정보를 생성하고 다시 이를 실체뷰에 반영하는 시간의 증가로 인해 뷰 검색 시간이 계속 증가하는 것으로 나타났다.

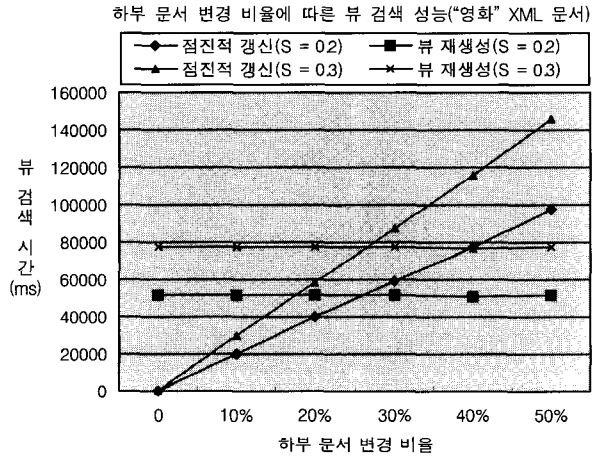
(그림 8)의 실험은 성능 파라미터들의 기본 설정을 <표 5>와 같이 하되, I : D : M의 비율은 2 : 2 : 6으로 하고, 뷰 선택율(S)의 값은 0.2일 때와 0.3일 때 각각 수행한 것이다. S = 0.2일 때와 S = 0.3일 때 모두 하부 문서 변경 비율이 약 27% 이하일 때는 뷰를 재생성하는 방법에 비하여 실체뷰를 점진적으로 갱신하는 기법이 더 효율적인 것으로 나타났다. 이는 하부 XML 문서에 대한 변경이 빈번하지 않은 응용의 경우, XML 문서 검색을 위한 실체뷰 기법이 아주 효율적이라는 것을 의미한다.

(그림 9)와 (그림 10)도 (그림 8)과 같이 하부 문서 변경 비율(L)의 변화에 따라 뷰를 재생성하는 방법과 점진적으로 갱신하는 기법의 성능을 비교한 것이다. (그림 9)는 (그림 8)의 실험에서 I : D : M의 비율을 4 : 4 : 2로 변경했을

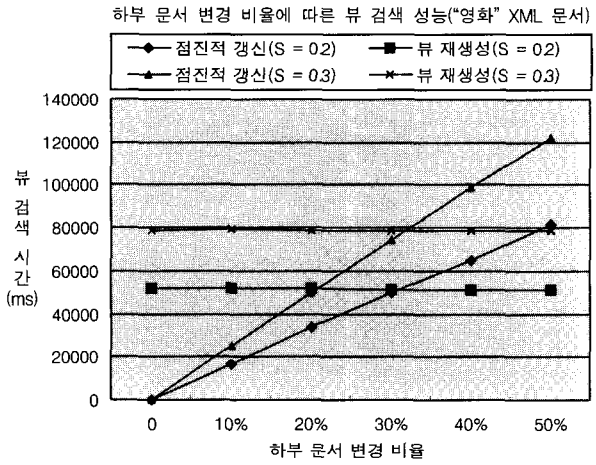
때의 실험 결과이다. (그림 10)은 "영화" 문서 대신에 크기가 큰 "Play" 문서를 대상으로 실험한 결과이다. (그림 9)에서는 하부 문서 변경 비율이 약 31%~32% 이하일 때는 뷰를 재생성하는 방법에 비하여 실체뷰를 점진적으로 갱신하는 기법이 더 효율적인 것으로 나타났다. (그림 10)에서는 성능 교차 지점이 하부 문서 변경 비율 약 42%까지 올라간 것으로 나타났다. 그밖에 이들 결과에서 몇가지 주요 파라미터 값의 변화가 미치는 영향에 대한 분석은 다음과 같다.

● 뷰 선택율(S)의 변화

(그림 8)과 (그림 9)에 나타난 성능은 S = 0.2일 때와 S = 0.3일 때 모두 측정된 것이다. S가 0.2에서 0.3으로 증가하면 하부 문서 중 뷰의 조건을 만족하는 문서의 비율이 증가한 것이므로 뷰를 재생성하는 방법과 실체뷰를 점진적으로 갱신하는 기법 모두에서 뷰 검색 시간이 증가하는 것으로 나타났다. (그림 8)에서는, 뷰 재생성의 경우 약 51%의 증가가 있었는데 반해 실체뷰 갱신의 경우 하부 문서



(그림 8) 하부 문서 변경 비율의 변화에 따른 실험 결과 연산 유형별 개수(I : D : M=0.2 : 0.2 : 0.6)



(그림 9) 하부 문서 변경 비율의 변화에 따른 실험 결과 연산 유형별 개수(I : D : M=0.4 : 0.4 : 0.2)

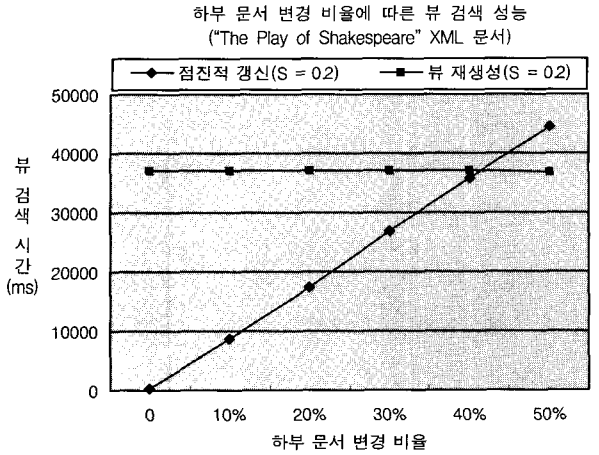
변경 비율이 약 27% 이하일 때는 증가폭이 그것보다 적고 그 이상일 때부터는 증가폭이 그것보다 커지기 시작했다. (그림 9)에서도 동일한 양상을 보였다. 뷰 재생성의 경우 약 51%의 증가가 있었고 실체뷰 갱신의 경우 하부 문서 변경 비율 약 32% 지점을 기준으로 그 이하일 때는 증가폭이 뷰 재생성의 그것보다 적었고, 그 이상일 때는 커지기 시작했다. 이러한 결과 역시 뷰 선택율의 증가시 하부 문서 변경 비율이 적절한 값 아래일 경우에는 뷰 재생성에 비해 실체뷰의 갱신이 더 효율적이라는 것을 의미한다.

● 연관성 유형들간의 비율 변화

(그림 8)과 (그림 9)에 나타난 성능은 실체뷰의 점진적 갱신시 변경로그의 레코드 중 뷰와 연관성이 있는 것들 간의 유형별 비율이 I : D : M = 2 : 2 : 6 그리고 I : D : M = 4 : 4 : 2 일 때 각각 측정된 것이다. (그림 8)에 비해 (그림 9)에서는 연관성 유형 MODIFY의 비율이 INSERT/DELETE에 비해 상대적으로 낮아졌는데 이는 MODIFY 유형 중 MODIFY-I 유형의 비율을 낮추는 효과를 갖는다. <표 1>의 다섯 가지 연관성 유형 중에서 MODIFY-I 유형의 처리 시간이 단연 가장 길다. 이는 MODIFY-I 유형의 변경에 대한 실체뷰 갱신정보를 생성하기 위해서는 변경로그 외에 하부 XML 문서에 대한 검색이 필요하기 때문이다. (자세한 사항은 4.4.3절을 참조.) 따라서 (그림 8)에서의 실체뷰 갱신 성능에 비해 (그림 9)의 그것은 하부 문서 변경 비율이 10%, 30%, 50% 일 때 모두 약 16% (S = 0.2 및 S = 0.3 경우 모두) 정도의 성능 향상을 보였다. 또한 이로 인해 뷰 재생성과 실체뷰 갱신 간의 성능 교차 지점도 하부 문서 변경 비율 기준으로 약 4%~5% 올라갔다. 즉, MODIFY-I 유형이 줄어들수록 뷰 재생성에 비해 실체뷰를 점진적으로 갱신하는 기법의 성능이 더욱 향상됨을 알 수 있다.

● XML 하부 문서 크기의 변화

(그림 8)과 (그림 9)에 나타난 성능은 크기가 작은 "영화" 문서를 대상으로 한 실험에서 측정된 것인데 반하여 (그림 10)의 성능은 크기가 아주 큰 "Play" 문서를 대상으로 측정된 것이다. 이 실험은 하부 문서 개수(U)를 1000개, 뷰 선택율(S)을 0.2, 그리고 I : D : M을 2 : 2 : 6으로 한 것이다. ("Play" 문서의 경우, 하부 문서의 용량 증대로 인해 뷰 재생성의 계산 시간이 많이 걸려 실험 대상 문서 수를 "영화" 문서의 경우처럼 크게 할 수 없었다.) (그림 8)과 (그림 9)의 결과에 비하여 주목할 점은 뷰 재생성과 실체뷰 갱신 간의 성능 교차 지점이 약 42%까지 올라갔다는 것이다. 이는 (그림 8)에서의 성능 교차 지점인 약 27%의 하부 문서 변경 비율과 비교했을 때, 뷰의 소스인 하부 문서의 크기가 커질수록, 한번 생성된 실체뷰를 점진적으로 갱신하는 기법의 효율성이 뷰 재생성에 비해 더욱 증대된다는 것을 의미한다.



(그림 10) 하부 문서 변경 비율의 변화에 따른 실험 결과
("The Play of Shakespeare" XML 문서)

4.3.2 하부 문서 개수의 변화에 따른 뷰 검색 성능 비교

본 실험에서는 하부 문서 변경 비율(L)이 일정할 때, 하부 문서 개수(U)의 변화에 따라, 뷰를 재생성하는 방법에 비해 실체뷰를 점진적으로 갱신하는 기법의 성능이 더 효율적일 때의 조건을 찾고자 하였다. 이는 Xyleme 프로젝트[23]에서 연구된 것과 같은 대용량 XML 웨어하우스에서 XML 실체뷰 기법의 효용성 여부를 평가하기 위한 것이다.

(그림 11)은 "영화" 문서를 대상으로 측정한 뷰 재생성 방법과 실체뷰 갱신 방법의 성능 비교를 나타낸 것이다. 뷰 선택율(S)을 0.2 또는 0.3으로 고정시켰기 때문에 하부 문서의 개수가 증가할수록 재생성되는 뷰의 크기는 커지는 것으로 나타났다. 또한 실체뷰의 점진적 갱신에서도, 하부 문서의 변경 비율(L)을 20%로 고정시켰기 때문에 하부 문서의 개수가 증가할수록 변경로그에 기록된 변경로그 레코드 개수가 증가하므로 이들을 처리하여 갱신정보를 생성하고 다시 이를 실체뷰에 반영하는 시간의 증가로 인해 뷰 검색 시간이 계속 증가하는 것으로 나타났다.

(그림 11)의 실험은 성능 파라미터들의 기본 설정을 <표 5>와 같이 하되, I : D : M의 비율은 2 : 2 : 6으로 하고, 뷰 선택율(S)의 값은 0.2일때와 0.3일때 각각 수행한 것이다. S = 0.2일때와 S = 0.3일때 모두 뷰를 재생성하는 방법에 비하여 실체뷰를 점진적으로 갱신하는 기법이 항상 더 효율적인 것으로 나타났다. 뿐만 아니라 하부 문서 개수가 증가할 때 두 방법의 뷰 검색 시간 증가폭이 뷰 재생성 쪽이 실체뷰 갱신 쪽보다 더 크게 나타났다. 이는 대용량 XML 웨어하우스를 구축하여 운용하는 데 있어, XML 문서 검색을 위한 실체뷰 기법이 아주 효율적이라는 것을 의미한다.

(그림 12)와 (그림 13)도 (그림 11)과 같이 하부 문서 개수(U)의 변화에 따라 뷰를 재생성하는 방법과 점진적으로 갱신하는 기법의 성능을 비교한 것이다. (그림 12)는 (그림 11)의 실험에서 I : D : M의 비율을 4 : 4 : 2로 변경했을 때의 실험 결과이다. (그림 13)은 "영화" 문서 대신에 크기가 큰 "Play" 문서를 대상으로 실험한 결과이다. (그림 12)와 (그림 13) 모두에서 뷰를 재생성하는 방법에 비하여 실체뷰를 점진적으로 갱신하는 기법이 항상 더 효율적인 것으로 나타났으며, 하부 문서 개수가 계속 증가해 갈때 뷰 재생성의 성능 저하 폭에 비해 실체뷰 갱신의 그것이 적은 것으로 나타났다. 그밖에 이들 결과에서 몇 가지 주요 파라미터 값의 변화가 미치는 영향에 대한 분석은 다음과 같다.

- 뷰 선택율(S)의 변화

(그림 11)과 (그림 12)에 나타난 성능은 S = 0.2일때와 S = 0.3일때 모두 측정된 것이다. S가 0.2에서 0.3으로 증가하면 하부 문서중 뷰의 조건을 만족하는 문서의 비율이 증가한 것이므로 뷰를 재생성하는 방법과 실체뷰를 점진적으로 갱신하는 기법 모두에서 뷰 검색 시간이 증가하는 것으로 나타났다. (그림 11)에서는, 뷰 재생성의 경우 하부 문서 개수가 10000, 20000, 30000일때 약 12초, 25초, 384초의 증가가 각각 있었는데 반해 실체뷰 갱신의 경우 약 10초, 19초, 29초의 증가가 각각 있었다. (그림 12)에서도 비슷한 양상을 보였다. 뷰 재생성의 경우 하부 문서 개수가 10000, 20000, 30000일때 약 12초, 26초, 39초의 증가가 각각 있었고, 실체뷰 갱신의 경우 약 8초, 17초, 25초의 증가가 각각 있었다.

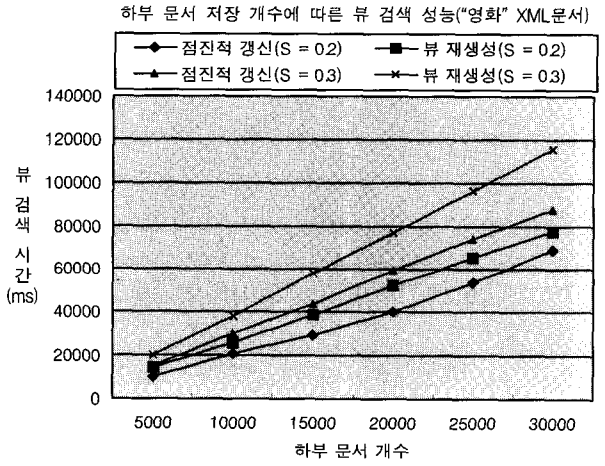
- 연관성 유형들 간의 비율 변화

(그림 11)과 (그림 12)에 나타난 성능은 실체뷰의 점진적 갱신시 변경로그의 레코드중 뷰와 연관성이 있는 것들간의 유형별 비율이 I : D : M = 2 : 2 : 6 그리고 I : D : M = 4 : 4 : 2 일 때 각각 측정된 것이다. (그림 11)에 비해 (그림 12)에서는 연관성 유형 MODIFY의 비율이 INSERT/ DELETE에 비해 상대적으로 낮아졌는데 이는 MODIFY 유형 중 MODIFY-I 유형의 비율을 낮추는 효과를 갖는다. 4.3.1 절에서 언급하였듯이 <표 1>의 다섯 가지 연관성 유형 중에서 MODIFY-I 유형의 처리 시간이 단연 가장 길다. 따라서 (그림 11)에서의 실체뷰 갱신 성능에 비해 (그림 12)의 그것은 하부 문서 개수가 10000, 20000, 30000일 때 각각 약 18%, 16%, 15%(S = 0.2의 경우) 그리고 약 17%, 16%, 14% (S = 0.3의 경우) 정도의 성능 향상을 보였다. 또한 이로 인해 뷰 재생성에 대한 실체뷰 갱신의 성능 향상의 정도도 (그림 11)의 경우에 비해 (그림 12)에서 더 커졌다.

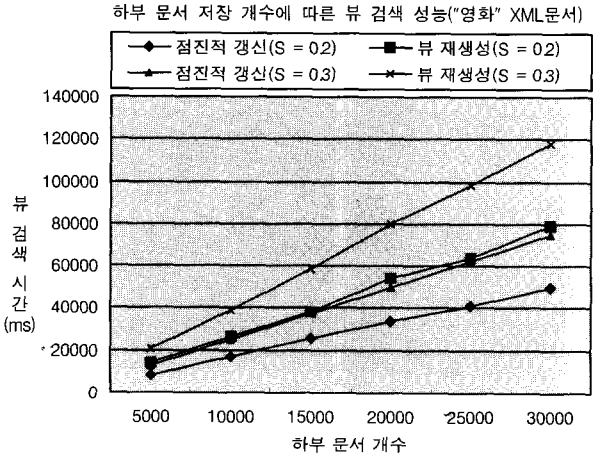
- XML 하부 문서 크기의 변화

(그림 11)과 (그림 12)에 나타난 성능은 크기가 작은 "영화" 문서를 대상으로 한 실험에서 측정된 것인데 반하여 (그림 13)의 성능은 크기가 아주 큰 "Play" 문서를 대상으로 측정된 것이다. 이 실험은 하부 문서 개수(U)를 1000개, 뷰 선택율(S)을 0.2, 그리고 I : D : M을 2 : 2 : 6으로 한 것이다. 역시 뷰의 재생성에 비해 실체뷰의 갱신이 항상 더 나은 성능을 보였는데, (그림 11)의 결과에 비하여 주목할

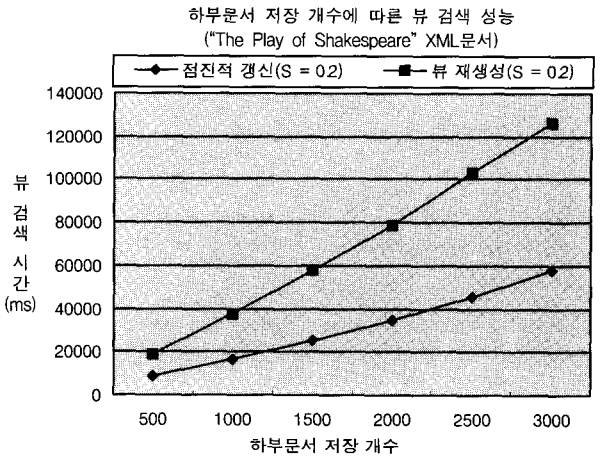
점은 ① 뷰 재생성에 대한 실체뷰 갱신의 성능 향상 폭이 더 커졌다는 것 그리고 ② 하부 문서 개수의 증가에 따른 뷰 검색 시간의 증가폭이 뷰 재생성에 비해 실체뷰 갱신 쪽이 더 줄어들었다는 것이다. 뷰 재생성에 대한 실체뷰 갱신의 성능 향상은 하부 문서 개수가 1000, 2000, 3000일 때 약 56%, 56%, 54% 정도로 나타났다. 하부 문서 개수 증가에 따른 뷰 검색 시간의 증가는, 하부 문서 개수가 1000에서 2000으로 증가할 때 하부 문서 개수 단위 증가 당 뷰 검색 시간이 뷰 재생성에서는 약 42ms 정도의 증가를 보인 반면, 실체뷰 갱신에서는 약 18ms 정도의 증가를 보였다. 하부 문서 개수가 2000에서 3000으로 증가할 때에는 각각 약 48ms와 23ms의 증가를 보였다. 이러한 결과를 (그림 11)에서의 성능 변화 추이와 비교하면, 뷰의 소스인 하부 문서의 크기가 커질수록, 한번 생성된 실체뷰를 점진적으로 갱신하는 기법의 효율성이 뷰 재생성에 비해 더욱 증대된다는 것을 의미한다.



(그림 11) 하부 문서 저장 개수의 변화에 따른 실험 결과
연산 유형별 개수(I : D : M = 0.2 : 0.2 : 0.6)



(그림 12) 하부 문서 저장 개수의 변화에 따른 실험 결과
연산 유형별 개수(I : D : M = 0.4 : 0.4 : 0.2)



(그림 13) 하부 문서 저장 개수의 변화에 따른 실험 결과
("The Play of Shakespeare" XML 문서)

4.4.3 연관성 유형별 갱신정보 생성 시간

<표 1>에는 변경로그에 기록된 하부 XML 문서에 대한 변경과 뷰와의 연관성 유형 다섯 가지가 정리되어 있다. 본 절에서는 4.4.1절과 4.4.2절의 실험을 수행하는 과정에서 이들 다섯 가지 연관성 유형 별로 하나의 변경로그 레코드로부터 해당 갱신정보를 생성하는 데까지 걸리는 시간을 측정한 결과를 기술한다.

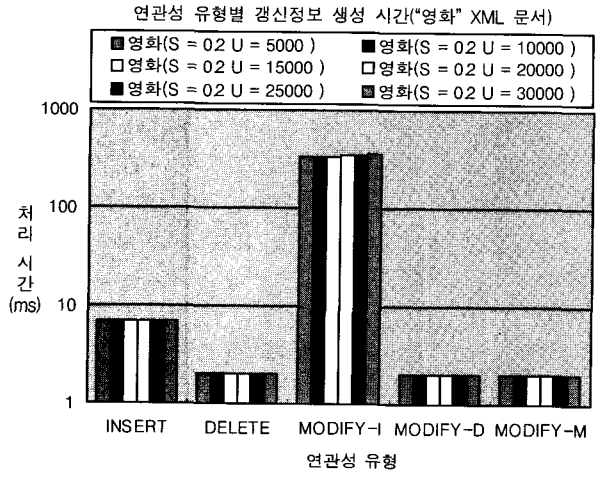
(그림 14)는 4.4.2절의 실험에서 하부 XML 문서로 "영화" 문서의 개수(U)를 5,000개에서 30,000개까지 증가시켜 가는 동안 각 U의 값마다 나타나는 연관성 유형 별 갱신정보 생성 시간의 평균을 구하여 나타낸 것이다. 각 유형별 시간은 하나의 로그 레코드를 처리하는데 걸린 평균 시간을 구한 것이므로 U의 값에 상관없이 거의 동일한 값이 나와야 한다. (그림 14)에서 먼저 이점이 확인되며 이는 구현된 본 논문의 XML 저장 시스템이 대용량의 XML 문서를 저장하였을 때 정확하게 동작한다는 하나의 반증이다. 이보다 중요한 것은 연관성 유형 별 갱신정보 생성 시간인데 MODIFY-I 유형이 가장 오랜 시간을 소요하는 것으로 나타났다. 그 다음으로 INSERT 유형이 많은 시간을 소요했고 나머지 세 유형들은 비슷한 시간이 걸리는 것으로 나타났다.

MODIFY-I 유형이 단연 가장 많은 시간을 요하는 이유는, 어떤 하부 문서가 엘리먼트 값 수정으로 뷰의 조건을 만족하게 되었는데 해당 문서에서 뷰의 결과로 검색되어야 할 추출 엘리먼트의 값이 변경로그에 기록되어 있지 않으므로 갱신정보를 생성하기 위하여 하부 XML 문서를 저장하고 있는 XmlElement 테이블의 검색이 요구되기 때문이다. 다른 유형는 모두 갱신정보를 생성하는데 필요한 정보를 해당 변경로그 레코드가 다 가지고 있으므로 XmlElement 테이블의 검색을 필요로 하지 않는다(이에 대한 자세한 사항은 [21]을 참조).

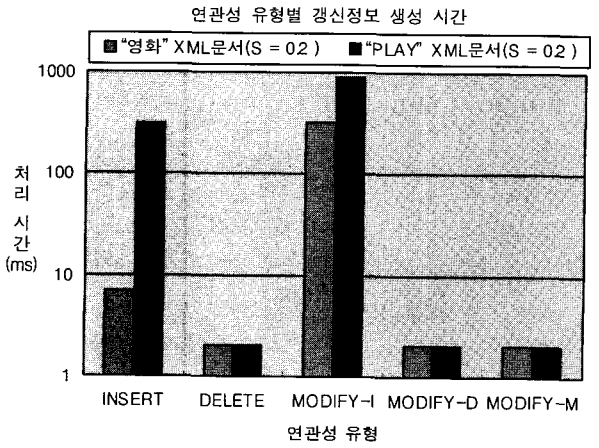
MODIFY-I 유형 다음으로 INSERT 유형의 처리 시간이

많이 드는 이유는 나머지 다른 세 유형 DELETE, MODIFY-I, 그리고 MODIFY-D와 달리 갱신정보 생성시 데이타 (즉, 엘리먼트의 값)를 처리하는 부담이 크기 때문이다. 즉, 변경로그에 기록되어 있는 새로 삽입된 문서의 내용 중 추출 엘리먼트를 검색하는 등의 작업을 필요로 하기 때문이다.

이상의 두 유형에 대해서는, "Play" 문서처럼 크기가 큰 XML 문서의 경우 해당 갱신정보의 생성에 소요되는 시간이 더 늘어나게 된다. 왜냐하면 갱신정보 생성에 필요한 추출 엘리먼트의 검색 시간이 엘리먼트의 수가 많은 문서일수록 더 걸리기 때문이다. (그림 15)는 4.4.1절의 실험에서 하부 XML 문서로 "영화" 문서와 "Play" 문서를 각각 사용하면서 하부 문서 변경 비율(L)을 0%에서 50%까지 증가시켜 가는 동안 나타나는 연관성 유형 별 갱신정보 생성 시간의 평균을 구하여 나타낸 것이다. (그림 15)에서와 같이 MODIFY-I, INSERT 순으로 가장 많은 시간이 소요되었고 나머지 세 유형이 소요한 시간은 비슷하게 나타났다. 하부 XML 문서가 "영화" 문서일때에 비해 "Play" 문서로 바뀌면 MODIFY-I와 INSERT 유형은 그 시간이 대폭 증가하는 것으로 나타



(그림 14) 연관성 유형별 갱신정보 생성 시간

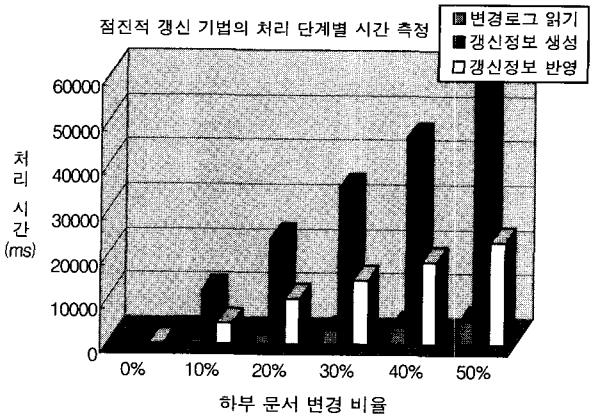


(그림 15) 연관성 유형별 갱신정보 생성 시간

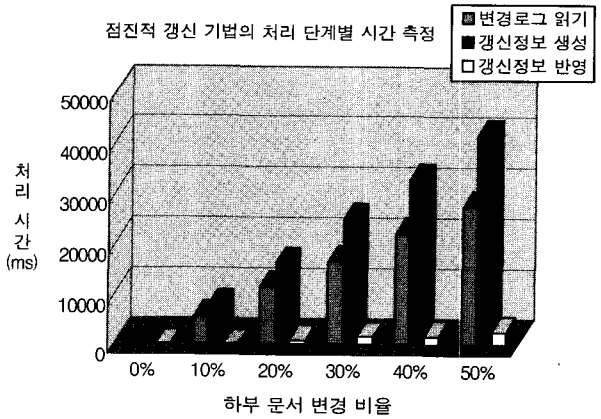
났다. 반면 나머지 세 유형은 갱신 정보 생성 과정이 하부 문서의 엘리먼트 수에 영향을 받지 않으므로 두 문서간에 거의 차이를 보이지 않는 것으로 나타났다.

4.4.4 실체뷰의 점진적 갱신 처리 단계별 시간

본 절에서는 4.4.1절의 실험에서 실체뷰의 점진적 갱신의 경우 처리 단계별 시간을 측정한 결과를 기술한다. 실체뷰를 점진적으로 갱신하는 데 걸리는 시간은 크게 ① 변경로그를 읽는 시간, ② 변경로그 레코드에 기록된 변경과 뷰와의 연관성을 판별하고 연관성이 있을시 해당 갱신정보를 생성하는 시간, 그리고 ③ 생성된 갱신정보를 실체뷰에 반영하는 시간으로 나눌 수 있다. (그림 16)은 20000개의 "영화" 문서를 대상으로 뷰 선택율(S)은 0.2이고, 연관성 유형간의 비율(I : D : M)은 2 : 2 : 6일때 하부 문서 변경 비율(L)의 변화에 따른 이들 단계별 시간의 총합을 나타낸 것이다.



(그림 16) 점진적 갱신 기법의 처리 단계별 시간
("영화" XML 문서)



(그림 17) 점진적 갱신 기법의 처리 단계별 시간
("The Play of Shakespeare" XML 문서)

하부 문서 변경 비율이 0%인 경우에는 읽어야 할 변경로그가 없으므로 이 경우 모든 단계의 처리 시간은 없다. L이 10%에서 50%까지 증가해 가면 실체뷰를 갱신하기 위한

부담이 커지는 것이므로 각 단계별 처리 시간은 모두 증가하였다. 세 단계간에 비교해 보면 두 번째 단계인 갱신정보 생성이 가장 긴 처리 시간을 요하는 것으로 나타났다. 이는 변경로그 읽기와 갱신정보를 실체뷰에 반영하는 작업은 상대적으로 단순 작업인데 반해 변경과 뷰와의 연관성 여부를 판단하고 그로부터 갱신정보를 생성하는 과정이 가장 복잡도가 높으며 실체뷰 기법의 핵심임을 의미하는 것이다.

(그림 17)은 1000개의 "Play" 문서를 대상으로 뷰 선택율(S)은 0.2이고, 연관성 유형간의 비율(I : D : M)은 2 : 2 : 6일 때 하부 문서 변경 비율(L)의 변화에 따른 이들 단계별 시간의 총합을 나타낸 것으로 (그림 16)과 비슷한 결과를 나타냈다.

5. 결론 및 향후 연구 과제

본 논문에서는 관계 DBMS를 기반으로 하는 XML 저장 시스템에서 XML 실체뷰의 점진적 갱신 기법의 성능 평가에 관해 연구하였다. 이를 위해 [19]에서 제안된 XML 실체뷰 갱신 알고리즘을 바탕으로 실체뷰를 지원하는 XML 저장 시스템을 구현하고, 구현한 시스템을 대상으로 XML 뷰의 검색을 위해 해당 뷰를 하부 XML 문서들로부터 재생성하는 방법과 기존에 생성한 실체뷰를 점진적으로 갱신하는 기법의 성능을 실험을 통해 측정하고 비교하였다. 점진적 갱신 기법은 실체뷰의 일관성을 유지하기 위한 오버헤드가 있음에도 불구하고 실체뷰에 반영해야 할 변경의 양이 아주 많지 않을 경우 전자에 비해 훨씬 나은 성능을 나타냈다. 이는 자주 제기되는 질의에 대해 매번 그 결과를 재생성하는 것보다 그 결과를 실체뷰로 유지하면서 뷰의 하부 데이타에 대한 변경 중 뷰와 연관성이 있는 것만을 점진적으로 반영하여 일관성을 유지하는 실체뷰 기법이 XML 데이타의 검색에서도 효율적이라는 것을 입증하는 것이다.

본 논문에서 수행한 실험에서는, ① 뷰 재생성과 해당 실체뷰의 점진적 갱신의 성능이 교차하는 지점을 찾아 어떤 조건에서 실체뷰의 갱신이 뷰 재생성보다 성능이 우수한지를 검토하였고, ② 한편으로는 본 실험을 위하여 구현된 실체뷰를 지원하는 XML 저장 시스템의 안정적이고 정확한 작동 여부를 파악하였다. 실험 결과, ① 실체뷰를 생성한 대상 소스인 하부 XML 문서의 20% 대 후반에서 40% 대 초반까지에 이르는 양의 변경에 대해 뷰에 대한 연관성 검사 및 실체뷰의 점진적 갱신을 수행하는 것이 해당 뷰를 재생성하는 것보다 성능이 우수하며, 실체뷰의 갱신을 위해 고려되어야 하는 변경의 양이 그 이상을 넘어서면 해당 뷰를 재생성하는 것이 더 나은 것으로 나타났다. ② 뷰의 소스인 하부 XML 문서의 수가 증가할수록 실체뷰 기법의 성능 우위는 더 확고해지는 것으로 나타났다. 그리고 ③ 연관성 유형 별 갱신정보의 생성 시간과 실체뷰 기법의 처리 단계별 시간을 측정한 결과 논리적이고 일관된 결과를 얻을 수 있었다.

본 논문의 성능 평가에 사용된 XML 실체뷰의 점진적 갱신 알고리즘은 성능 및 기능 향상을 위하여 여러 가지 측면에서 확장 가능하다. 첫째, 변경로그의 자료구조 확장이다. 현재 새로운 문서의 삽입시 변경로그에도 그 문서가 기록되고 하부 문서 영역에도 문서가 저장되는데 변경로그 레코드중 문서 삽입 레코드의 자료구조 변경으로 이와 같은 중복을 피할 수 있다. 둘째, 뷰와 연관성이 있는 변경로그 레코드별로 갱신정보를 생성한 후 이들간의 관계를 비교함으로써 후최적화(post-optimization)를 수행할 수 있다. 예를 들어 삽입된 문서가 이후 삭제되면 두 갱신정보를 모두 없앨 수 있다. 셋째, XML 하부 문서의 변경 단위를 삽입과 삭제의 경우에도 문서 단위에서 엘리먼트 단위로 할 경우 그리고 뷰 정의를 위한 XML 질의의 조건을 복잡화할 경우의 연관성 검사 및 갱신정보 생성 알고리즘의 확장이 필요하다. 본 논문에서는 이러한 확장은 고려하지 않고 성능 평가를 위한 실험을 수행하였다. 향후 연구 과제로는 이러한 확장을 3절에서 기술한 XML 저장 시스템에 구현하고 그에 따른 성능 평가를 수행하는 것이다.

참 고 문 헌

[1] T. Bray et al., "Extensible Markup Language(XML) 1.0," http://www.w3.org/TR/1988/REC-xml-19980210, 1998.

[2] D. Florescu and D. Kossmann, "Storing and Querying XML Data Using an RDBMS," IEEE Engg. Bulletin, pp.27-34, Sep., 1999.

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

[4] A. Deutsch et al., "Storing Semistructured Data with STORED," Proc. ACM SIGMOD Int'l Conf. on Management of Data, 1999.

[5] J. Shanmugasundaram et al., "Efficiently Publishing Relational Data as XML Documents," Proc. Int'l Conf. on VLDB, pp.65-76, 2000.

[6] M. Carey et al., "XPERANTO : Publishing Object-relational Data as XML," Proc. Workshop on the Web and Databases, May, 2000.

[7] J. Shanmugasundaram et al., "Querying XML Views of Relational Data," Proc. Int'l Conf. on VLDB, pp.261-270, 2001.

[8] M. Fernandez et al., "Efficient Evaluation of XML Middleware Queries," Proc. ACM SIGMOD Int'l Conf. on Management of Data, pp.103-114, 2001.

[9] M. Fernandez, et al., "SilkRoute : Trading Between Relations and XML," Proc. the 9th WWW Conf., 2000, pp723-746, 2000.

[10] K. Williams et al, XML Databases, Wrox, 2000.

[11] S. Banerjee et al., "Oracle 8i-The XML Enabled data management System," Proc. Int'l Conf. on Data Engineering,

pp.561-568, 2000.
[12] M. Rys, "State-of-the-art XML Support in RDBMS : Microsoft SQL Server's XML Features," IEEE Bulletin of TCDE, Vol.24, No.2, pp.3-11, Jun., 2001.
[13] eXcelon Inc., http://www.exceloncorp.com/.
[14] H. Schoning, "Tamino-A DBMS Designed for XML," Proc. Int'l Conf. on Data Engineering, pp.149-154, 2001.
[15] S. Abiteboul, "On Views and XML," Proc. ACM Symp. on Principles of Database System, pp.1-9, 1999.
[16] S. Abiteboul et al., "Active Views for Electronic Commerce," Proc. Int'l Conf. on VLDB, pp.138-149, 1999.
[17] S. Kim and H. Kang, "XML Query Processing Using Materialized Views," Proc. Int'l Conf. on Internet Computing, pp.111-117, Jun., 2001.
[18] Y. Papakonstantinou and V. Vianu, "DTD Inference for Views of XML Data," Proc. of 19th ACM SIGACT-SIG MOD-SIGART Symp. on PODS, 2000.
[19] S. Cluet et al., "Views in a Large Scale XML Repository," Proc. Int'l Conf. on VLDB, pp.271-280, 2001.
[20] A. Gupta and I. Mumick, "Maintenance of Materialized Views : Problems, Techniques, and Applications," IEEE Bulletin of TCDE, Vol.18, No.2, pp.3-18, Jun., 1995.
[21] 임재국 외, "점진적 갱신에 기반을 둔 XML 형성뷰 관리 프레임워크", 정보처리학회논문지D, 제8-D권 제4호, pp.327-338, 2001.
[22] http://www.ibiblio.org/bosak/.
[23] L. Xyleme, "A Dynamic Warehouse for XML Data of the Web," IEEE Data Eng. Bulletin, Vol.24, No.2, pp.40-47, June, 2001.
[24] Xyleme S. A., http://www.xyleme.com/.

성 호 상

e-mail : hssung@dblab.cse.cau.ac.kr
2001년 군산대학교 컴퓨터정보과학과 (공학사)
현재 중앙대학교 컴퓨터공학과 석사과정 재학중
관심분야 : 웹 데이타베이스, XML, Java

문 찬 호

e-mail : moonch@dblab.cse.cau.ac.kr
1997년 중앙대학교 컴퓨터공학과(공학사)
1999년 중앙대학교 컴퓨터공학과 대학원 (공학석사)
현재 중앙대학교 컴퓨터공학과 박사과정 재학중
관심분야 : 웹 데이타베이스, XML, 질의 변환

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 : 이동 데이타베이스, 웹 데이타베이스, DBMS 저장 시스템