변경 집합을 이용한 XML 문서의 버전 관리를 위한 저장 기법

윤 홍 원†

요 약

데이터 마이닝을 요구하는 전자상거래, 전자정부와 관련된 문서 처리 시스템 등에서 XML 문서의 버전 관리에 대한 관심이 높아지고 있다. 본 논문에서는 대량의 XML 문서를 장기적으로 유지하면서 XML 문서의 이력 정보를 효율적으로 관리하기 위하여, XML 버전을 생성하는 변경 집합을 정의하고 변경 집합을 이용한 XML 문서의 저장 방법을 제안한다. 변경 집합은 변경 연산집합과 시간지원 차원을 포함하고 있으며, 변경 연산 집합은 스키마 변경 연산과 데이터 변경 연산으로 구성된다. 변경 집합을 이용한 세가지 XML 문서의 저장 방법을 제안한다. 세 가지 저장 방법은, ① 변경 집합을 모두 저장하는 방법, ② 변경 집합과 버전을 주기적으로 저장 방법, 그리고 ③ 저장 적합 시점에 변경 집합 모음과 버전을 저장하는 방법이 있다. 또한, 기존의 저장 방법과 제안한 저장 방법들 사이에 성능을 비교한다. 성능 평가를 통하여 저장 적합 시점에 변경 집합모음과 버전을 저장하는 방법의 성능이 다른 저장 방법보다 우수함을 보인다.

Storage Policies for Versions Management of XML Documents using a Change Set

Hong Won Yun†

ABSTRACT

The interest of version management is increasing in electronic commerce requiring data mining and documents processing system related to digital governmentapplications. In this paper, we define a change set that is to manage historicalinformation and to maintain XML documents during a long period of time and propose several storage policies of XML documents using a change set. A change set includes a change operation set and temporal dimensions and a change operation set is composed with schema change operations and data change operations. We propose three storage policies using a change set. Three storage policies are ① storing all the change sets, ② storing the change sets and the versions periodically, ③ storing the aggregation of change sets and the versions at a point of proper time. Also, we compare the performance between the existing storage policy and the proposed storage policies. Through the performance evaluation, we show that the method to store the aggregation of change sets and the versions at a point of proper time outperforms others.

키워드 : XML, 버전 관리(Version Management), 저장 기법(Storage Polic)

1. 서 론

XML(eXtensible Markup Language)은 인터넷을 통한 데이터 교환의 새로운 표준으로 자리잡고 있다. XML 문서의 양은 시간이 지나면서 급속히 증가하고 있으며, 문서의 구조와 내용이 변화하고 있다. 정보 제공자와 협업자들은 문서를 재활용하고 과거 문서를 검색하기 위하여 XML 문서의 버전 관리에 많은 관심을 가지고 있다[1, 2]. 대량의 멀티미디어 문서 데이터, 전자도서관 등에서 문서의 버전 관리가 필요하며, 특히, 데이터 마이닝을 요구하는 전자상거래와 전자정부와 관련된 문서 처리 시스템에서 XML 문서의 버전 관리에 대한 관심이 높아지고 있다.

이러한 XML 문서는 그 내용의 일부가 새로 만들어지기도 하고, 이미 있던 내용이 삭제 또는 변경되기도 한다. 전자 도서관이나 전자정부의 사용자들은 지난 XML 문서들에 대하여 어떤 시점에 유효했던 정보를 찾기도 하고, 문서에 들어있는 내용이 어떻게 변경되었는지 문서의 이력 정보를 검색하기도 한다[3-5]. XML로 저장된 문서에서 과거의 이력 정보를 검색하기 위해서 대량의 XML 문서는 그 버전들을 효율적으로 관리할 필요가 있다.

1990년대 후반까지 객체, HTML, 텍스트 문서 모델등을 포함한 다양한 데이터 모델에 대하여 버전을 관리하기 위한 방법들이 개발되었으며[6-9], 최근에는 XML 문서의 버

† 정 회 원 : 신라대학교 컴퓨터정보공학부 교수
논문접수 : 2004년 4월 22일, 심사완료 : 2004년 9월 7일

전을 관리하는 방법들이 제안되고 있다[4, 12, 14]. XML 문서의 버전을 관리하기 위하여 제안된 이들 방법은, 저장 공간이나 검색 시간이라는 두 측면 중에서 어느 하나에 초점을 맞추어 설계되었다. 또한, 이들 방법은 현재로부터 가까운 과거의 버전을 관리하기에 적합하도록 고안되어 있으며 시간적으로 오래된 과거 정보와 대량의 XML 데이터를 관리하기에는 적합하지 않다.

최근에는 전자정부의 출현으로 대량의 XML 문서를 장기적으로 보존하면서 과거 어떤 시점에 유효했던 정보나 문서의 이력 정보를 효율적으로 검색할 수 있는 XML 문서의 버전 관리시스템을 필요로 하고 있다. 하지만, XML을 이용한 문서 관리의 필요성이 제기되고 있으며, 대량의 XML 문서를 관리하는 방법에 대한 연구도 시작 단계에 있다. 대량의 XML 문서에서 과거에 유효했던 정보를 검색하기 위해서는 XML 문서의 버전을 관리해야 하는데, 저장 공간을 효율적으로 사용하면서 과거 정보에 대한 검색을 지원하기 위해서 XML문서에서 변경된 내용을 저장하는 방법을 제안한다. 이 논문에서는, XML 문서에서 변경된 내용을 저장하는 방법으로써, XML 문서의 버전을 생성하게 한 변경 집합을 정의하고, 이 변경 집합을 이용한 XML 문서의 저장 방법을 제안한다.

이 논문의 구성은 다음과 같다. 2장에서는 기존의 XML 문서의 버전 관리 방법을 살펴보고, 3장에서 시간지원 차원을 지원하는 XML 문서 버전의 변경 집합을 정의한다. 4장에서 XML 문서 버전의 저장 방법을 제안하고, 5장에서는 실험을 통하여 제안한 저장 방법의 성능을 평가한다. 마지막으로, 6장에서 결론을 맺는다.

2. 관련 연구

시간이 지나면 실세계의 규칙이나 지식이 바뀌게 되며, 이러한 데이터를 반영하는 XML 문서도 그 내용이 변경되거나 문서의 구조가 바뀌게 된다. XML 문서의 버전 관리는 이와 같이 시간이 지남에 따라 변화하는 문서를 관리하는 것을 말한다. 전통적인 텍스트 라인 중심의 버전 관리 방법은 SCCS(Source Code Control System)와 RCS(Revision Control System)가 있다[8, 9]. SCCS는 소스 코드파일의 버전을 관리할 목적으로 개발되었으며, 소스 코드의 변경이 발생하면 변경 과정을 기록하고 관리할 수있다. RCS는 가장 최근 파일과 역방향 편집 스크립트를 저장하며, 역방향 편집 스크립트를 적용하여 최신 파일을 제외한 다른 버전을 생성한다. SCSS와 RCS는 텍스트 라인 중심의 문서를 관리하기에 적합하지만 XML과 같이 구조적인 문서의 버전을 관리하기에는 적합하지 않다.

XML 문서의 버전을 관리하는 방법으로써 UBCC(Usefulness Based Copy Control)가 있다[12]. UBCC는 현재 유효한 개체들을 중심으로 클러스터링한다. 한 페이지 안에서 현재 유효한 개체가 임계값보다 작으면 유효한 개체를 다른 페이지에 복사하고, 편집 스크립트를 유지한다. 이 방법은 XML 문서의 단기적인 버전 관리에 초점을 맞추었다. 또한, XML 문서의 내용 변경을 중심으로 개발되었으며 구조 변경에 대한 버전 관리가 쉽지 않은문제점을 가지고 있다.

XML 문서의 버전을 저장하는 간단한 방법으로써 모든 버전을 저장하는 방법이 있다[14]. 이것은 문서의 이력 정보를 요구하는 질의가 많은 응용에 적합하며 가장 단순한 저장 방법이다. 이 방법은 저장 공간을 많이 필요로 하는 문제점을 가지고 있다. Marian 등은 가장 최근버전과 역방향 델타를 저장하는 방법을 제시하였다[14]. 이 방법은 저장 공간을 적게 차지하지만, 델타를 이용하여 과거 버전을 생성하므로 이력 정보를 검색하는 데 시간이 많이 걸리게 된다.

대량의 XML 문서의 버전을 관리하기 위하여 저장 공간을 적게 차지하고 검색 시간을 줄일 수 있으면서, 과거 어떤 시점에 유효했던 XML 문서를 찾거나 문서의 이력 정보 검색할 수 있는 저장방법이 필요하다. UBCC는 XML 문서의 내용에서 최근에 변경된 버전을 저장하고 관리할 수 있는 방법이지만, 대량의 XML 문서를 장기적으로 관리할 수 없고 또한 이력 정보를 검색할 수 없는 문제점을 가지고 있다. Marian이 제안한 방법은 과거 버전을 생성하기 위해서 델타를 이용하며 역 방향으로 과거 버전을 생성하기 때문에 이력 정보를 검색하는 경우에는 검색 시간이 많이 걸리는 단점이 있다.

XML 문서의 버전 관리와 관련된 연구로써 UBCC와 Marian이 제안한 방법이 가지고 있는 문제점을 해결하기 위하여, XML 문서에서 이력 정보를 검색하는 시간을 줄이고 대량의 XML문서의 버전을 관리하는 방법의 연구가 필요하다. 과거에 유효했던 정보의 검색을 지원하기 위해서 XML 문서의 변경집합을 정의하고, 변경집합을 이용하여 장기적으로 XML 문서를 저장하는 방법을 연구한다. 우리는 시간지원 질의를 지원하는 XML 문서 버전의 변경 집합을 정의하고, 변경 집합에서는 XML 문서의 내용 변경과 구조 변경 연산을 정의한다. 많은 양의 XML 문서 버전을 효율적으로 관리하기 위하여, 변경 집합을 이용한 XML 문서 버전의 저장 방법을 제안한다. 제안하는 저장 방법에서는 장기적인 관점에서 문서의 버전을 유지할 수 있도록 한다.

3. 변경 집합

2장에서 XML 문서의 버전 관리와 관련된 기존의 연구들을 살펴 보았다. 기존 연구에서는 버전과 버전 사이의 변화를 델타(또는편집 스크립트)로 표현하였다[12, 14]. 이 장에

서는 XML 문서의 버전과 버전 사이의 변화를 변경 집합으로 나타내며, 이 변경 집합은 구조 변경 연산과 내용 변경 연산을 모두 포함한다. XML 문서의 버전은 하나의 초기 문서 M에서 시작하여 첫 번째 변화 뒤에 $V1$이 되고, 그 뒤를 이어 변화의 과정을 거치면서 전체적으로 V_1, V_2, …, V_n의 순서를 가지게 된다. 여기서 마지막 V_n은 현재 유효한 버전이거나 최종 버전이 된다.

임의 XML 문서의 버전(이하 버전)에서 바로 다음 버전으로 변경된 내용을 다음과 같이 튜플 U로 표현하며 U를 버전 변경 집합(이하 변경 집합)이라고 한다.

$$U = (d, v, i, p, C(d))$$

변경 집합 U안의 d는 XML 문서의 식별 번호, v는 XML 문서 버전의 번호, 그리고 i는 각 버전에서 몇 번째 버전인가를 나타낸다. XML 문서의 식별 번호 d는 최초 문서의 고유번호이고, 최초 문서로부터 버전이 생성되면 각 버전을 구분하기 위하여 버전 번호 v를 부여한다. i는 최초 버전에서 몇 번째 버전인지를 나타내며 임의 버전에 빠른 접근을 위하여 사용된다. $C(d)$는 하나의 버전에서 발생한 변경 연산의 집합을 말하며, 이미 저장되어 있는 버전에 변경 연산을 적용하여 다음 버전을 생성할 수 있는데, 이 때 p는 저장되어 있는 버전의 식별 번호를 나타낸다. 변경 집합 U안에 들어 있는 변경 연산의 집합 $C(d)$는 다음과 같다.

$$C(d) = (S \cup D) \circ T$$

S는 XML 문서의 스키마 변경 연산집합을 나타내고, D는 XML문서의 데이터 변경 연산 집합이다. 스키마 변경 연산은 XML 문서에서 구조 변경을 표현하고, 데이터(값)가 변경되면 그 변경 내용을 데이터 변경 연산으로 나타내는데, 각각은 <표 1>과 <표 2>와 같으며 기본적인 연산을 나타낸다. T는 시간지원 데이터베이스에서 시간지원 질의 처리를 위해서 사용하는 시간지원차원이며 트랜잭션 시간과 유효 시간을 포함한다. 아래의 S는 스키마 변경 연산 집합을 표현한 것이다.

$$S = \{e, a, n, i\}$$

XML 문서의 스키마에서, e는 엘리먼트의 스키마 변경 연산이며 a는 애트리뷰트의 스키마 변경 연산을 나타낸다. n은 네임 스페이스의 변경 연산이고, I는 스키마 식별자의 변경 연산을 뜻한다. 이러한 스키마 변경 연산은 <표 1>과 같다.

$$D = \{ l, t \}$$

XML 문서의 데이터 변경 연산 집합 D에서, l은 데이터 엘리먼트의 변경 연산이고 t는 데이터 애트리뷰트의 변경 연산을 나타낸다. 시간의 연속을 t_1, t_2, …, t_n이라고 하면, 어떤 시점 t_i에서 생성된 버전의 정보가 실세계에서 유효한 시간을 유효 시간으로 나타내고, 버전이 시스템에 기록된 시간을 트랜잭션 시간으로 나타낸다. 시간지원 차원[13, 15]을 나타내는 T는 유효 시간과 트랜잭션 시간을 가지며 다음과 같다.

$$T = \{ vt, tt \}$$

유효 시간 vt는 유효 시작 시간과 유효 종료 시간의 쌍으로 나타내고, 트랜잭션 시간 tt는 트랜잭션 시작 시간과 트랜잭션 종료 시간의 쌍으로 나타내며, 다음과 같다.

vt = [*valid_start_time, valid_end_time*]
tt = [*transaction_start_time, transaction_end_time*]

〈표 1〉 스키마 변경 연산

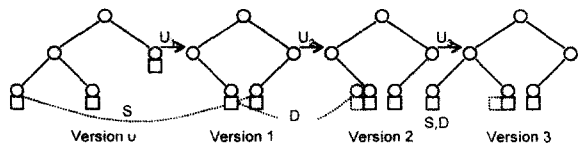
스키마 변경 연산	설 명
InsertElement(e, n, p)	엘리먼트 e를 위치 p에 부모 노드 n의 자식 노드로 삽입한다
DeleteElement(e)	엘리먼트 e를 삭제한다
MoveElement(e, n, p)	엘리먼트 e를 위치 p에 부모 노드 n의 자식 노드로 옮긴다
RenameElement(e, ne)	엘리먼트 e의 이름을 새 이름 ne로 바꾼다
CreateElement(e, n, p)	위치 p에 노드 n의 루트 엘리먼트 e를 만든다.
UnionElement(e, p)	위치 p에 엘리먼트 e의 모든 하위 노드를 e의 자식 노드로 만든다
AddAttribute(a, e, t)	엘리먼트 e에 타입 t를 가지는 애트리뷰트 a를 삽입한다
DeleteAttribute(a, e)	엘리먼트 e에서 애트리뷰트 a를 삭제한다
ChangeAttributeType(a, e, t)	엘리먼트 e의 애트리뷰트 a의 타입을 t로 바꾼다
ChangeNamespace(s)	타겟네임스페이스를 s로 바꾼다
ChangeIdentifer(oid, nid)	파일이름/위치 oid를 새로운 이름/위치 nid로 바꾼다

〈표 2〉 데이터 변경 연산

데이터 변경 연산	설 명
InsertDataElement(e, p)	위치 p인 엘리먼트 e에 새로운 값을 삽입한다
DeleteDataElement(e)	엘리먼트 e의 값을 삭제한다
UpdateDataElement(e, v)	엘리먼트 e의 값을 v로 바꾼다
InsertDataAttribute(a, v, p)	위치 p의 애트리뷰트 a의 값을 v로 바꾼다
DeleteDataAttribute(a)	애트리뷰트 a의 값을 삭제한다
UpdateDataAttribute(a, v, e)	엘리먼트 e의 애트리뷰트 a의 값을 v로 바꾼다.

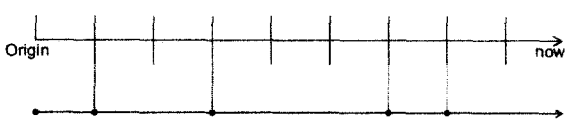
모든 변경 집합 $C(d)$는 유효 시간과 트랜잭션 시간을 가

짐으로써 XML문서 버전의 이력 정보를 검색할 수 있도록 지원한다.



(그림 1) XML 문서의 버전과 변경 집합의 관계

(그림 1)은 XML 문서의 버전과 변경 집합의 관계를 나타낸다. (그림 1)에서 원형은 엘리먼트이고, 사각형은 텍스트 노드를 뜻한다. XML 문서의 초기 버전인 Version 0에서 스키마의 변경이 일어나서 Version 1이 생성되었다. 스키마 변경은 스키마 변경 연산 S에 의하여 이루어졌으며 이 스키마변경 연산은 U_1으로 표현된다. Version 0에서 Version 1에 생성된 과정은, *Version* $0 \circ U_1$ = *Version* 1으로 나타낼 수 있다. 마찬가지로, Version 1에서 데이터 변경 연산 D가 발생하여 Version 2가 되었으며, 이것은 *Version* $1 \circ U_2$ = *Version* 2로 나타낸다. 따라서, 어떤 버전 V_i에서 다음 버전 V_{i+1}로 버전이 바뀐 것은 $V_i \circ U_{i+1} = V_{i+1}$가 된다.



(그림 2) XML 문서의 버전과 변경 집합 사이의 순서

(그림 2)에서와 같이 버전의 생성 순서가 V_1, V_2, …, V_n이고, 변경 집합이 만들어진 순서가 U_1, U_2, …, U_n이라고 하면 전체 버전과 전체 변경 집합 사이의 생성 순서는 다음과 같이 된다.

$$U_1, V_1, U_2, V_2, \cdots, V_n, U_n$$

보기를 들어서, V_2가 저장되어 있는 버전이라고 하면 V_2에 U_3을 적용하여 버전 V_3을 재생성할 수 있다. 이 때, XML 문서의 버전을 모두 저장한다면 대량의 저장 공간을 필요로 할 것이고, 변경 집합을 저장한다면 모든 문서의 버전을 저장하는 것보다 적은 저장 공간을 차지한다. 저장된 버전에 변경 집합을 적용하여 다른 버전을 재생성할 수 있다. 어떤 버전을 저장하고 변경 집합은 어떻게 저장할 것인가에 대해서 4장에서 살펴본다.

4. 저장 방법

앞 장에서 XML 문서의 버전 변경 집합에 대해서 살펴보았다. 4장에서는 이 논문에서 제안하는 XML 문서의 세 가지 저장 방법에 대해서 살펴 본다. 앞으로 다루게 될 세 가지 저장 방법들은 3장에서 살펴 본 변경 집합을 이용한다. 세 가지 저장 방법들은 기본적으로 질의 빈도가 가장 높은 가장 최근 버전과, 특정 버전을 재생성 하는데 사용되는 최초 버전을 저장해 둔다. 앞으로 살펴 볼 XML 문서의 세 가지 저장 방법은 다음과 같다.

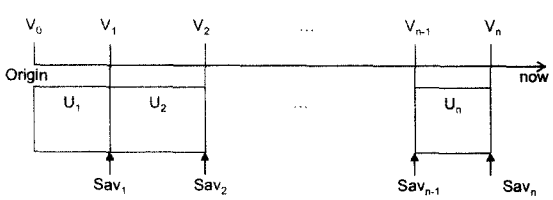
- 스킴 1 : 변경 집합을 저장한다.
- 스킴 2 : 변경 집합과 버전을 주기적으로 저장한다.
- 스킴 3 : 가장 적합 시점에 변경 집합 모음과 버전을 저장한다.

스킴 1은 버전이 생성되면 버전을 저장하지 않고 변경 집합을 저장한다. 최초 버전 이후부터 가장 최근 버전사이에 만들어진 모든 변경 집합을 저장한다. 스킴 2는 변경 집합을 주기적으로 모아서 저장하고 버전을 주기적으로 저장한다. 스킴 3은 저장 적합 시점을 찾아서 변경 집합의 모음과 버전을 저장한다.

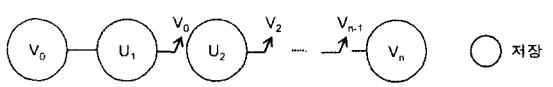
4.1 스킴 1

스킴 1은 최초 버전과 가장 최근 버전, 그리고 변경 집합을 저장한다. (그림 3)은 변경 집합을 저장하는 스킴 1의 개념을 그림으로 나타낸 것이다. 이 저장 방법은 버전이 생성되면 버전을 저장하지 않고 변경 집합만 저장한다. (그림 3)에서 V_0, V_1, …, V_n은 버전을 나타내고, U_1, U_2, …, U_n은 변경 집합을 의미한다. Sav_1, Sav_2, …, Sav_{n-1}은 각 버전이 생성된 시점이고, 또한 각 변경 집합이 저장되는 시점이다. (그림 3)에 의하면, Sav_1 시점에 V_1이 생성되었으며 변경 집합 U_1이 저장된다. 스킴 1에서는, 버전 V_0와 V_n이 저장되고 U_1에서 U_n 사이(U_n 포함)의 변경 집합이 저장된다 (그림 3) 참조.

이 방법은 가장 최근 버전이 저장되어 있다가 새로운 버전이 만들어지면 새로 만들어진 버전이 현재 버전이 되고, 이 때 직전 버전은 삭제되며 변경 집합만 저장된다. 보기를 들어서, (그림 3)에서 V_{n-1}이 가장 최근 버전으로 저장되어 있다가 새로운 버전 V_n이 만들어지면 저장되어 있던 V_{n-1}은 삭제하고 변경 집합인 U_n을 저장한다.



(그림 3) 스킴 1 : 변경 집합을 저장하는 방법

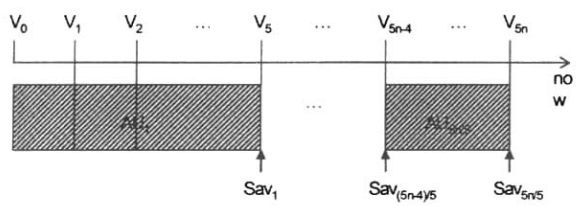


(그림 4) 스킴 1에서 저장되는 버전과 변경 집합

스킴 1에서 저장되는 버전과 변경 집합을 (그림 4)에서 나타낸다. 스킴 1에서 저장되는 버전과 변경 집합의 순서는 V_0, U_1, U_2, …, U_n, V_n과 같다. V_0에 U_1을 적용하여 V_1을 재생성 할 수 있다. 이것을 $V_1 = V_0 \circ U_1$으로 나타낸다. 마찬가지로, V_1에 U_2를 적용하여 V_2을 재생성 할 수 있으며, $V_2 = V_1 \circ U_2$로 표현할수 있다. 스킴 1에서는 최초 버전과 가장 최근 버전을 제외한 모든 버전을 최초 버전으로부터 다시 만들어야 한다. 따라서, V_2는 최초 버전 V_0에서 시작하여 다시 만들어야 하며, 이 과정은 $V_2 = (V_0 \circ U_1) \circ U_2$가 된다. n-1번째 버전을 재생성하는 과정을 식으로 나타내면, $V_{n-1} = ((V_0 \circ U_1) \circ U_2) \circ U_3) \circ U_{n-1}$가 된다.

4.2 스킴 2

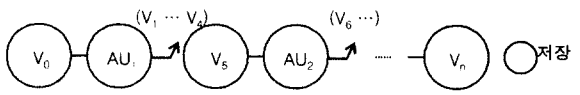
스킴 2는 변경 집합을 주기적으로 모아서 저장하고 변경 집합을 저장하는 주기에 버전을 저장한다. 저장 주기는 시간 단위가 된다. 여기서는 시간 단위로써 네 개의 수준, 연, 월, 주, 일을 사용한다. 연, 월, 주, 그리고 일 가운데에서 하나를 저장 단위로 선택할 수 있다.



(그림 5) 스킴 2 : 변경 집합과 버전을 주기적으로 저장하는 방법

변경 집합과 버전을 주기적으로 저장하는 방법의 개념은 (그림 5)와 같다. 이 그림에서 저장 주기는 5로 가정하였으며, AU_1, …, $AU_{5n/5}$ 각각은 저장 주기마다 변경 집합의 모음을 나타낸다. AU_1 은 첫 주기에 다섯 개의 변경 집합을 모은 것이며 ($AU_1 = U_1 + \cdots + U_5$), 첫 주기의 저장 시점인 Sav_1에 AU_1이 저장되고, 또한 V_5가 저장된다. 이 스킴에서 저장되는 버전은 V_0, V_5, …, V_{5n}이고, 저장되는 변경 집합은 AU_1, …, $AU_{5n/5}$이 된다.

스킴 2에서는 가장 최근 버전을 사용하고 있다가, 새로운 버전이 생성되면 새 버전이 만들어진 시점을 저장 주기인 시점과 비교한다. 새로운 버전이 생성된 시점이 저장 주기 시점보다 크거나 같으면 변경 집합의 모음과 직전 버전을 저장하고, 저장 주기 시점보다 작으면 변경 집합은 임시 저장소에 계속 모으고 직전 버전은 삭제한다. 보기를 들어 (그림 5)에서, 가장 최근 버전을 V_2, 첫 번째 저장 주기인 시점을 Sav_1이라고 하면, 버전 V_2를 사용하고 있다가 새로운 버전 V_3이 생성되면 새 버전 V_3이 생성된 시점을 첫번째 저장 주기인 시점 Sav_1과 비교한다. (그림 5)에서는, V_3이 생성된 시점이 Sav_1보다 작으므로 변경 집합을 임시 저장소에 계속 모으고 직전 버전 V_2를 삭제한다. 시간이 지나서, 가장 최근 버전 V_{5n}를 사용하고 있다가, 새로운 버전 V_{5n+1}이 만들어졌다고 하면, V_{5n+1}이 생성된 시점과 저장 주기시점 $Sav_{5n/5}$와 비교한다. V_{5n+1}이 생성된 시점이 저장 주기 시점 $Sav_{5n/5}$ 보다 크므로 임시 저장소에 들어있는 변경 집합의 모음 $AU_{5n/5}$를 저장하고, 직전 버전 V_{5n}을 저장한다.

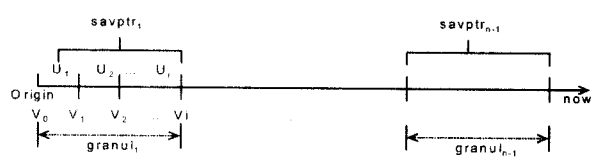


(그림 6) 스킴 2에서 저장되는 버전과 변경 집합

(그림 6)은 스킴 2에서 저장되는 버전과 변경 집합의 보기를 나타낸다. 스킴 2에서 버전과 변경 집합이 저장되는 일련의 순서는 V_0, AU_1, V_5, AU_2, …, V_n이 된다. 최초 버전 V_0에 첫 번째 변경의 집합의 모음 AU_1을 적용하여 V_1, …, V_4를 재생성할 수 있으며, 이것을 $V_1 \sim V_4 = V_0 \circ AU_1$로 표현한다. V_5에 변경 집합의 모음 AU_2를 적용하여 V_6, …, V_9를 다시 만들 수 있고, $V_6 \sim V_9 = V_5 \circ AU_2$로 표현한다. 스킴 2에서 어떤 버전을 재생성하는 과정을 식으로 나타내면, $V_{qn+1} \sim V_{qn+(q-1)} = V_{qn} \circ AU_{n+1}$ ($n \geq 0$, 저장주기 : q)이 된다.

4.3 스킴 3

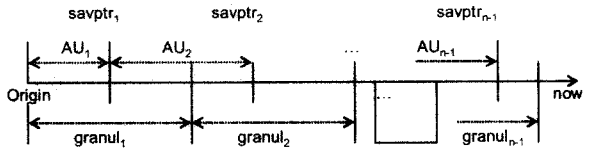
스킴 3은 저장 주기 안에서 변경 연산이 가장 많았던 버전을 저장하고, 저장하는 버전의 직전까지 변경 집합을 모아서 저장한다. (그림 7)에서 $granul_i$은 저장 주기를 나타내고, 이 저장 주기 안에 버전 V_0, V_1, V_2, …, V_i가 있고 변경 집합 U_1, U_2, …, U_i가 있다. 각 변경 집합 안에는 변경 연산의 집합 $C(d)$가 포함되어 있다. 변경 연산의 집합 $C(d)$를 읽어서 변경 연산이 가장 많은 버전을 찾는다 ($max(count\_of\_\ UpdateOp(C_1(d)), \cdots, C_i(d))$). 변경 집합 U_4에 들어있는 $C_4(d)$ 가 가장 많은 변경 연산을 가지고 있다면 V_4를 저장하고 V_4까지 변경 집합의 모음 (AU_1=U_1, U_2, U_3, U_4)을 저장한다. V_4와 AU_1을 저장하는 시점이 $savptr_1$이 된다.



(그림 7) 스킴 3 : 저장 적합 시점을 찾아 저장하는 방법

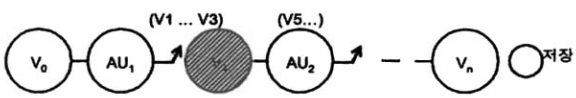
스킴 3에서는 가장 최근 버전을 사용하고 있다가 새로운 버전이 생성되면 새 버전이 만들어진 시점과 저장 주기 시점을 서로 비교한다. 새로운 버전이 만들어진 시점이 저장 주기 시점보다 작으면 변경 집합과 직전 버전을 임시 저장

소에 저장하고, 크거나 같으면 직전에 저장된 버전의 변경 집합 앞까지 읽어서 변경 연산이 가장 많았던 버전을 찾아서 저장한다. 변경 연산이 가장 많았던 버전의 변경 집합에서부터 직전에 저장된 버전의 변경 집합까지 변경 집합의 모음을 저장한다.



(그림 8) 저장 주기, 저장 적합 시점, 그리고 변경 집합 모음과의 관계

(그림 8)에서 저장 주기, 저장 적합 시점, 그리고 변경 집합모음 사이의 관계를 나타낸다. 첫 번째 저장 주기 $granul_1$ 안의 시간에서 변경 연산이 가장 많았던 변경 집합을 찾은 뒤에, 첫번째 변경 집합과 버전을 저장한 시점을 $savptr_1$이라고 하면, 그 다음 저장 적합 시점 $savptr_2$는 $granul_2$안의 시간에서 찾는다. $granul_2$안의 시간에서 찾은 저장 적합시점이 $savptr_2$라고 하면, 변경 집합의 모음은 직전 저장 적합 시점 $savptr_1$에서부터 $savptr_2$ 직전까지 변경 집합의 모음을 만들고 저장한다.



(그림 9) 스킴 3에서 저장되는 버전과 변경 집합

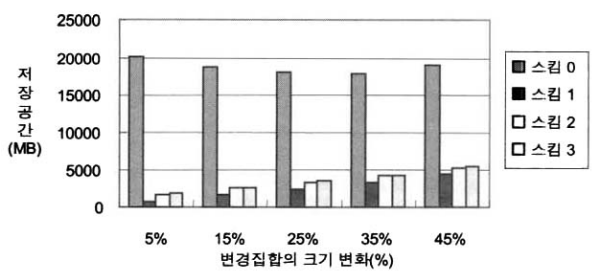
(그림 9)는 스킴 3에서 저장되는 버전과 변경 집합을 나타낸 것이다. 스킴 3에서 저장되는 버전과 스킴의 순서는 V_0, AU_1, V_4, AU_2, …, V_n이 된다. 스킴 2에서는 저장될 버전이 미리 결정되어 있으나 스킴 3에서는 저장 시점이 미리 결정되어 있지 않으며, 저장 적합 시점을 찾아서 저장한다. 최초 버전 V_0에 첫 번째 변경의 집합의 모음 AU_1을 적용하여 V_1, …, V_3를 다시 만들 수 있으며, 이것을 $V_1 \sim V_3 = V_0 \circ AU_1$로 표현한다. 저장된 버전 V_4에 AU_2를 적용하여 AU_2에 들어있는 변경 집합의 순서대로 버전을 재생성한다.

5. 성능 평가

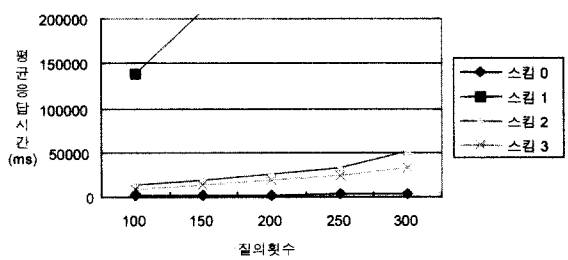
4장에서는 본 논문에서 제안하는 세 가지 저장방법을 살펴 보았다. 이 장에서는 기존의 저장 방법과 세 가지 저장 방법들 사이의 성능을 평가한다. 성능 평가는 각 저장 방법이 차지하는 저장 공간과 사용자 질의에 대한 평균 응답 시간을 측정하여 서로 비교하였다. 실험에서는 XML 문서의 크기를 100메가바이트로 하였고, 모의 실험 기간(lifespan)을 365로 하였다. 모의 실험 기간동안 변경 횟수는 랜덤하게 발생하였으며, 사용자 질의의 비율은 현재 질의 50%, 과거 질의 50%로 하였다. 변경이 발생할 때 마다 각 저장 방법에 맞게 버전과 변경 집합을 저장하였다. C언어로 작성한 모의 실험 프로그램을 펜티엄4 프로세서의 Win2000 Server에서 수행하였으며 Seagate사의 ST3x 시리즈의 디스크 탐색시간인 8.5ms를 사용하였다.

성능 평가에서 사용한 기존의 저장 방법은 모든 버전을 저장하는 방법을 채택하였다[14]. 모든 버전을 저장하는 방법(아래 차트에서 '스킴 0'로 표기)은 최초 XML 문서로부터 새로운 버전이 생성될 때마다 생성된 버전을 모두 저장하는 방법이며, 각 버전의 변경 집합은 없다. (그림 10)은 각 저장 방법들이 차지하는 저장 공간을 나타낸다. 이 실험에서는 XML 문서에서 변경 집합이 차지하는 비율을 5%에서 45%까지 바꾸어가면서 각 저장 방법이 차지하는 저장 공간의 크기를 평가하였다. 질의에 대한 응답 시간은 변경 집합 탐색 시간, 버전 재생성 시간, 그리고 해당 레코드 탐색 기간의 합으로 산출하였다.

(그림 10)에서 보인 것처럼, 스킴 1이 저장 방법들 가운데 가장 적은 저장 공간을 차지하는 것으로 나타난다. 스킴 1은 최초 버전과 가장 최근 버전, 그리고 변경 집합만을 저장하기 때문에 저장 공간을 적게 차지한다. 스킴 0은 모든 버전을 저장하므로 가장 많은 저장 공간을 차지하는 것으로 나타나며, 다른 저장 방법보다 지나치게 많은 저장 공간이 필요함을 알 수 있다. 스킴 2와 스킴 3은 저장 공간을 차지하는 비율이 비슷하게 나타나며, 저장 공간이 가장 적게 필요한 스킴 0와 큰 차이가 없음을 알 수 있다.



(그림 10) 저장 방법들 사이의 저장 공간 비교



(그림 11) 저장 방법들 사이의 평균응답시간 비교

(그림 11)은 네 가지 저장 방법의 사용자 질의에 대한 평균응답 시간을 나타낸다. 스킴 1이 가장 큰 응답 시간을 보이는데, 이것은 과거 버전에 대한 질의가 발생하는 경우에, 버전을 재생성하는 시간이 많이 걸리기 때문이다. 스킴 0은 가장 작은 응답 시간을 보이고 있다. 과거 버전을 모두 저장하고 있는 스킴 0은 과거 버전을 다시 생성하는 시간이 들지 않기 때문에 가장 작은응답 시간을 보인다. 스킴 2는 비교적 적은 응답 시간을 보이고 있으며, 스킴 3은 스킴 0보다 많은 응답시간을 보이며, 스킴 2보다 우수한 성능을 보이고 있다.

(그림 10)과 (그림 11)에서 보듯이, 스킴 0은 버전을 모두 저장하고 있으므로 과거 질의에 대하여 버전을 재생성할 필요가 없으므로 평균 응답 시간은 우수하나, 모든 버전을 저장하고 있으므로 대량의 저장 공간을 필요로 하는 문제점을 가지고 있다. 스킴 1은 과거 버전을 모두 저장하는 것이 아니라 변경 집합만을 저장하므로 가장 적은 저장 공간이 필요하지만, 과거 버전을 재생성하는 시간이 많이 소요되므로 사용자 질의에 대하여 지나치게 큰 응답 시간을 보이고 있다. 스킴 2와 3은 최초 버전이 생성된 이후부터 현재까지의 시간상에 여러 개의 버전을 저장하면서 변경 집합을 보관하고 있으므로 저장 공간 측면에서 격차가 거의 없다. 조건없이 주기적으로 버전과 변경 집합을 저장하는 스킴 2보다, 저장 적합 시점을 찾아서 동적으로 저장하는 스킴 3이 응답 시간에서 우수한 성능을 보이고 있다. 저장 적합 시점을 찾아서 동적으로 변경 집합의 모음과 버전을 저장하는 방법(스킴 3)이 저장 공간측면에서 양호한 성능을 보이며, 응답 시간이 가장 작게 나타났다.

6. 결 론

대량의 XML 문서를 장기적으로 유지하면서 XML 문서의 이력 정보를 효율적으로 관리하는 방법의 필요성이 대두되고 있다. 이 논문에서는 XML문서를 관리하기 위하여, XML 버전을 생성하는 변경 집합을 정의하고 변경 집합을 이용한 XML 문서의 저장 방법을 제안하였다.

버전 변경 집합에는 변경 연산 집합과 시간지원 차원이 들어있고, 변경 연산 집합은 스키마 변경 연산과 데이터 변경 연산으로 구성된다. 이 논문에서 제안한 XML 문서의 저장 방법은 스킴 1, 2, 3이 있다. 스킴 1은 변경 집합을 중심으로 저장하며, 스킴 2는 변경 집합과 버전을 주기적으로 저장하고, 스킴 3은 저장 적합 시점을 찾아서 변경 집합과 버전을 저장한다. 기존의 저장 방법을 포함하여 이 논문에서 제안한 방법들 사이에 실험을 통하여 성능을 평가하였다. 스킴 0은 평균 응답 시간은 우수하나 지나치게 많은 저장 공간을 차지하였으며, 스킴 1은 가장 적은 저장 공간이 필요하지만 사용자 질의에 대하여 지나치게 큰 응답 시간을 보였다. 스킴 2와 3은 적은 저장 공간을 차지하였으며, 스킴 3은 응답 시간에서 우수한 성능을 보였다.

참 고 문 헌

[1] Shu-Yao Chien, Vassilis J. Tsotras, Carlo Zaniolo, "XML Document Versioning," SIGMOD Records, Vol.30, No.3, pp.46-53, Sep., 2001.

[2] J. Shanmugasundaram, K. Tufte, G. He, C. Zhang, D. DeWitt and J. Naughton, "Relational databases for querying xml documents : Limitations and opportunities," In Proc. of the 25th VLDB Conf., pp.302-314, Sep., 1999.

[3] Kjetil Nrvg, "Temporal query operators in XML databases," The Symposium of Applied Computing 2002, pp.402-406, 2002.

[4] B. Benatallah, M. Mahdavi, P. Nguyen, Q. Z. Sheng, L. Port, B. McIver, "An Adaptive Document Version Management Scheme," The 15th Conference on Advanced Information Systems Engineering (CAiSE'03), pp.16-20, Austria, 2003.

[5] Toshiyuki Amagasa, Masatoshi Yoshikawa, and Shunsuke Uemura, "Realizing Temporal XML Repositories using Temporal Relational Databases," The Third International Symposium on Cooperative Database Systems for Advanced Applications (CODAS'2001), Beijing, China, pp.23-24, April, 2001.

[6] Sommerville, I., Rodden, T., Rayson, P., Kirby, A., Dix, A. "Supporting information evolution on the WWW," World Wide Web 1, pp.45-54, 1998.

[7] Benatallah, B., "A Unified Framework for Supporting Dynamic Schema Evolution in Object Databases," 18th Int. Conf. on Conceptual Modeling - ER'99, Paris, France, Springer-Verlag (LNCS series), 1999.

[8] Rochkind, M. J., "The Source Code Control System," IEEE Transactions on Software Engineering 1, pp.255-265, 1975.

[9] Tichy, W. F., "RCS - A System for Version Control," Software Practice and Experience 15, pp.637-654, 1985.

[10] Chien, S. Y., Tsotras, V. J., Zaniolo, C., "Copy-Based versus Edit-Based Version Management Schemes for Structured Documents," RIDE-DM'2001, Heidelberg, Germany, 2001.

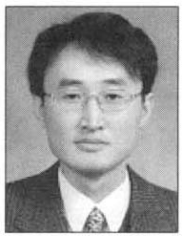
[11] Chawathe, S. S., Abiteboul, S., Widom, J., "Representing and Querying Changes in Semistructured Data," Proc. of Int. Conf. on Data Engineering (ICDE), pp.4-13, 1998.

[12] Chien, S. Y., Tsotras, V. J., Zaniolo, C., "Version Management of XML Documents," WebDB (Informal Proceedings), pp.75-80, 2000.

[13] G. Ozsoyoglu and R. T. Snodgrass, "Temporal and Real-Time Databases : A Survey," IEEE Transactions on Knowledge and Data Engineering, Vol.7, No.4, pp.511-532, August, 1995.

[14] A. Marian, S. Abiteboul, G. Cobena, L. Mignet, "Change-Centric Management of Versions in an XML Warehouse," In Proc. of 27th Int. Conf. on Very Large Data Bases (VLDB). pp.581-590, 2001.

[15] T. Amagasa, M. Yoshikawa and S. Uemura. "Realizing temporal XMLrepositories using temporal relational databases," In CODAS, pp.63-68, 2001.



윤 홍 원

e-mail : hwyun@silla.ac.kr

1986년 부산대학교 계산통계학과 졸업 학사

1990년 한국외국어대학교 경영정보대학원 전자계산학과 이학사

1998년 부산대학교 대학원 전자계산학과 이학박사

2003년~2004년 North Carolina State University 객원교수

1996년~현재 신라대학교(구.부산여자대학교) 컴퓨터정보공학부 부교수

관심분야 : 데이터베이스 시스템, 시맨틱 웹, 시간 데이터베이스