주기억 상주형 XML DBMS 저장 구조 설계

임혁수, 노현종, 이규철
충남대학교 컴퓨터공학과
e-mail : maverick@cnu.ac.kr

Design of a Storage Structure for Main Memory XML DBMS

HyuckSoo Lim, HyunJong Noh, KyuChul Lee
Dept. of Computer Engineering, ChungNam National University

요 약

최근 인터넷을 중심으로 한 첨단 기술 산업이 발달함에 따라 고성능 트랜잭션 처리가 요구되고, XML 관련 기술들 역시 확대 적용되고 있다. 이에 따라 디스크 기반 DBMS 의 한계를 극복하고, 점차 사용 범위가 확대되고 있는 XML 기술을 지원하기 위한 시스템의 필요성이 대두되고 있다. 기존의 XML 저장 방법은 대부분 XML DOM 구조 정보만을 저장하는 것이었으나, 이 방법은 XQuery 를 처리하는 속도면에 한계가 있었다. 본 논문에서는 이 점을 해결하기 위해 2 중의 인덱스 구조를 두어, XQuery 처리 속도를 향상시킬 수 있는 주기억 상주형 XML DBMS 의 저장 구조를 설계 한다.

1. 서론

최근 이동통신 기술과 모바일 기술, 인터넷 쇼핑몰, 콜 센터, 게임 등의 첨단 기술 산업이 발달함에 따라 고성능 트랜잭션 처리가 요구되고 있다.

기존의 디스크 기반 DBMS 는 하드 디스크에서 데이터를 읽어 메모리로 적재한 후, 메모리에서 클라이언트에게 정보를 제공하도록 하는 구조를 취해, 구조적으로 성능이 저하가 될 수 있다는 문제점을 안고 있다. 반면, 주기억 상주형 DBMS 는 디스크에 대한 접근 없이 직접 메인 메모리 접근을 통해 데이터를 관리함으로써 고성능 트랜잭션 처리가 가능하다.

이와 함께 최근 첨단 기술 산업에서는 XML 의 장점으로 인해 그 사용이 빈번해지고 있다. 현재 많은 연구가 진행되고 있는 유비쿼터스 환경에서 사용되는 각종 센서나 기기들간의 통신 메시지가 그 대표적인 예라 하겠다. 그러나 기존에 주로 사용되던 관계형 DBMS 는 계층 구조를 가진 XML 데이터를 나누어 저장하는 형식을 취하기 때문에 XML 데이터를 효율적으로 관리 하기 어렵다. 이에 따라 부각 되고 있는 것이 XML DBMS 인데, 이것은 XML 구조를 DB 에서 그대로 유지하게 함으로서 XML 데이터를 훨씬 효과적으로 관리 할 수 있도록 해준다.

본 논문에서는 점차 증가하고 있는 상기의 두가지 필요성, 즉 고성능의 트랜잭션 처리와 XML 데이터 처리를 동시에 만족시킬수 있도록 주기억 상주형 DBMS 를 기반으로 하는 XML DBMS 의 저장 구조를 설계 한다.

2. 관련 연구

XML 데이터의 사용 범위가 확대됨에 따라 XML 문서를 효과적으로 저장하기 위한 연구도 매우 활발하게 진행되고 있다. 연구 초기에는 기존의 관계형 DBMS 에 XML 문서를 저장하는 방법에 대한 연구가 활발했으나, 점차 XML 문서 정보를 그대로 유지할수 있는 저장 구조에 대한 연구로 옮겨가고 있다. 아파치(Apach e)의 Xindice 와 슬리피캣(Sleepycat)의 버클리 DB XML 등과 같은 XML DBMS 가 그 대표적인 성과물이라 하겠다. 또한 고성능의 트랜잭션 처리를 위해 주기억 상주형 DBMS 에 대한 연구도 활발하게 진행되고 있으며, 국내에도 알티베이스(Altibase), 카이로스(KAIROS)와 같은 상용화 제품이 나와 있는 상태이다.

3. 시스템의 구조

3.1. 시스템 요구사항

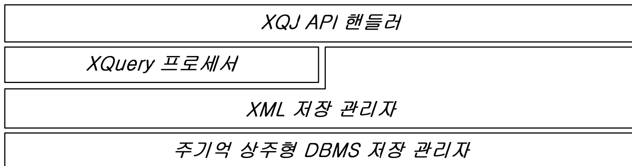
본 논문에서 설계하는 XML 저장구조는 기존의 주기억상주형 DBMS 의 하부 시스템인 저장 관리자 위에 적용될 예정이다. 이 시스템에서는 XML 문서의 저장/삭제/검색 기능을 제공하는데, 수정 기능은 문서단위의 수정을 원칙으로 하고 있다. XML 문서에 대한

질의어로 사용되는 W3C 의 XQuery 에서 수정 기능을 지원하고 있지 않을 뿐 아니라, 노드 단위의 수정을 지원할 경우 복잡한 트랜잭션에 의해 XML DBMS 의 주요 기능인 검색 성능에 부정적인 영향을 줄 수 있기 때문이다..

XML DBMS 에 어떤 방식으로 XML 문서를 저장/검색할 것인가의 문제도 중요한 문제이다. XQuery 에서 질의하는 대상이 개별적인 XML 문서이고, 구조상 하나의 XML 문서가 기존의 관계형 DBMS 정보나 또 다른 XML 문서의 정보 모두를 포함 할 수도 있기 때문이다. 본 연구에서는 문서철에 해당하는 컬렉션(Collection)을 저장 단위로 사용한다. 컬렉션은 XML 문서를 보관하는 논리적 구조로 각각의 XML 문서는 문서이름으로 구별되며, 저장할 수 있는 XML 문서의 수에는 제한이 없다.

3.2. 전체 시스템의 구조

전체 시스템은 [그림 1]과 같은 구조를 가진다.



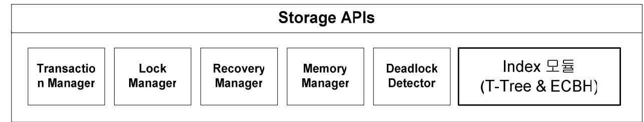
[그림 1] 전체 시스템 구성

애플리케이션으로부터의 요청은 XQJ API 핸들러를 통해 XQuery 프로세서, XML 저장 관리자를 거쳐 주기억 상주형 DBMS 의 저장 관리자에 있는 데이터에 접근하게 된다.

XQJ API 핸들러는 XQJ(XML Query API for Java)를 이용해 검색/저장/삭제/수정을 할수 있도록 API 를 제공하고, 외부로부터 받은 요청을 하위 블록에 전달한다. XQuery 를 통해 이루어지는 검색 명령은 하부의 XQuery 프로세서로 전달되고, 별도의 API 를 통해 이루어지는 저장/삭제/수정에 대한 요청은 XML 저장 관리자에 직접 전달 된다.

XQuery 프로세서는 XQJ API 핸들러로부터 오는 검색 요청을 처리하기 위한 블록이다. 내부에는 XQuery 파서, 질의 최적화기, 질의 코드 생성기 등이 포함되어 있다.

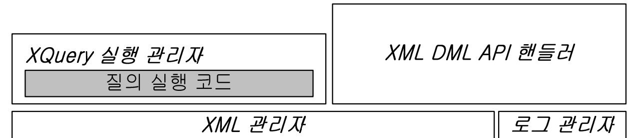
XML 저장 관리자는 상위 블록으로부터 전달된 XML 처리 요청을 받아들여, 하위의 주기억 상주형 DBMS 저장 관리자가 실제로 질의를 처리할 수 있도록 중간자 역할을 한다. 즉 XML 의 저장 및 논리적 질의 작업이 XML 저장 관리자에서 이루어지게 되는 것이다. XML 저장 관리자는 논리적인 XML 문서 저장소를 제공하며, [그림 2]와 같은 구조의 주기억 상주형 DBMS 의 저장 관리자는 그 문서가 물리적으로 저장/삭제/검색되는 부분이다.



[그림 2] 주기억 상주형 DBMS 의 저장 관리자 구조

3.3. XML 저장 관리자의 구조

[그림 3]은 XML 저장 관리자의 내부 구조를 도시한 것이다. XML 관리자와 로그 관리자 이외의 요소들은 상위의 요청을 실제로 수행 할수 있는 형태로 처리하기 위한 것들이다.



[그림 3] XML 저장 관리자 구조

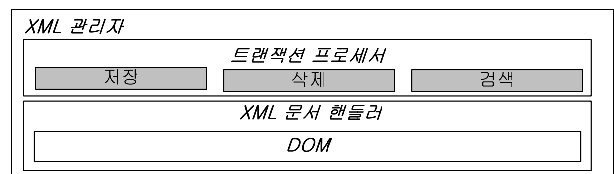
[그림 1]에서와 같이 XQJ API 핸들러가 저장/삭제에 대한 요청을 했을 경우, XML 저장 관리자는 XML DML API 핸들러를 통해 그 요청을 받아들인다. XML DML API 핸들러는 다시 XML 관리자에 해당 질의에 대한 처리를 요청하고, 로그 관리자를 통해 요청 내용과 결과에 대한 로그 정보를 기록하게 된다.

XQuery 실행 관리자는 XQuery 프로세서로부터의 요청을 받아들인다. XQuery 프로세서는 자체적인 처리를 통해 XQuery 실행 관리자에게 질의 실행 코드를 전달하게 되는데, XQuery 실행 관리자는 그 실행 코드를 이용해 XML 관리자에게 실제 정보의 검색을 요청하고 그 결과를 재가공 하게 된다.

XML 관리자는 상위 블록으로부터의 요청을 처리하고 하위의 주기억 상주형 DBMS 저장 관리자에 물리적인 정보 처리를 요청하는 부분이다. 이 부분은 XML 저장 관리자의 핵심이 되는 부분으로 입력 받은 XML 문서와 하부의 물리적인 구조를 연결한다. XML 문서의 저장/삭제/수정 작업의 트랜잭션 처리를 위해 하부의 주기억 상주형 DBMS 저장 관리자의 트랜잭션 관련 기능을 이용하게 된다. 본 논문에서는 주로 XML 문서와 하부의 물리적인 구조를 어떻게 연결할지에 대해 다룬다.

3.4. XML 관리자의 구조

[그림 4]는 XML 저장 관리자의 핵심인 XML 관리자의 구조이다.



[그림 4] XML Manager 구조

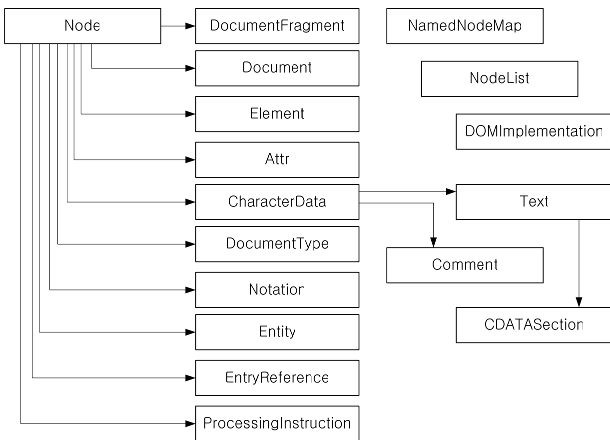
트랜잭션 프로세서는 저장/삭제/검색 요청을 트랜잭

션이 고려된 상태로 처리하기 위한 부분으로 주기억 상주형 DBMS 저장 관리자와 가장 긴밀하게 상호작용하게 된다. XML 문서 핸들러는 XML 관리자로 들어온 XML 문서를 DOM 형태를 유지한 상태로 물리적 저장구조로 변환하는 기능을 하게 된다. 결국 XML DBMS 가 제공하는 모든 기능은 XML 문서 핸들러로부터 제공되는 기능들이 그 기반이 되는 것이다.

4. XML 저장 구조의 설계

4.1. XML 문서의 구조

본 논문에서 설계하는 XML 저장 구조에서는 XML 의 DOM 구조를 그대로 유지하게 된다. DOM 구현에는 여러가지 형식이 존재한다. 기본 문서 작업을 위한 인터페이스의 핵심인 DOM Core 와 선택적 모듈인 DOM HTML, DOM CSS 가 대표적인데, 본 저장 구조에서는 DOM Core 만을 사용한다. DOM Core 의 구조는 [그림 5]와 같다.



[그림 5] DOM Core 의 구조

실제 저장 구조에서는 [그림 5]와 같은 구조를 기존의 MMDBMS 에 저장하게 된다.

4.2. XQuery 처리를 위한 요구 사항

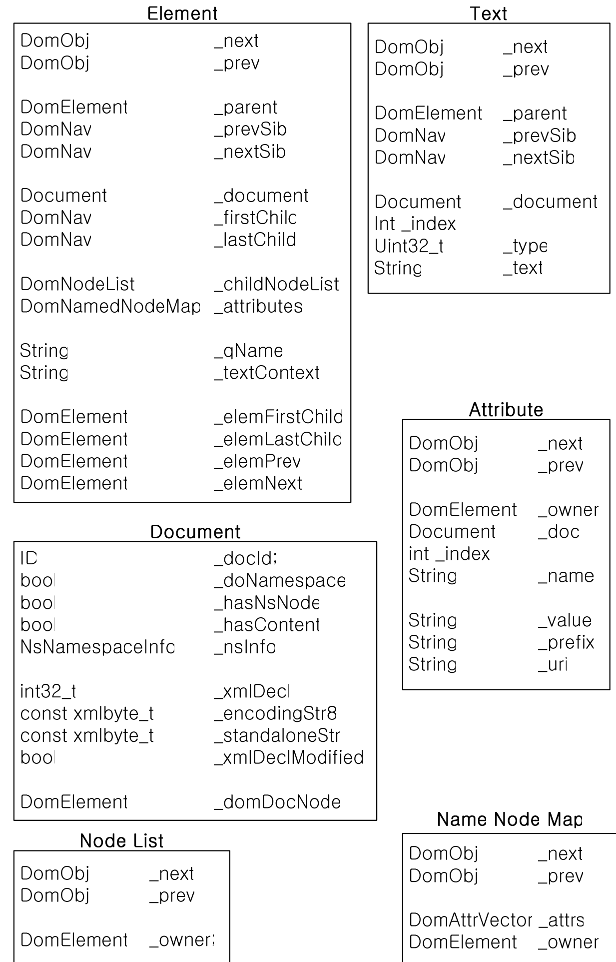
[그림 1]에서와 같이 본 논문에서 설계하고 있는 저장 구조에 검색을 위한 질의를 할 때는 XQuery 를 사용하게 된다. XQuery 는 W3C 에서 제정한 XML 질의 언어이다. XQuery 는 여러가지 유형의 표현식으로 구분되어 정의 되어 있으나, 그중 XQuery 질의의 전형적인 형태를 보여주는 표현식은 FLWOR 표현식으로, 조인과 재구조화 기능을 제공한다. FLWOR 표현식은 For/Let 절, Where 절, OrderBy/Return 절로 구성된다. 하부의 XML 저장 구조에서는 FLWOR 에서 사용될 경로 표현식을 빠르게 지원할 수 있어야 하는데, XQuery 에서 지원하는 주요 표현식은 다음과 같다.

```
PathExpr ::=          ("/" RelativePathExpr?)
                      | ("//" RelativePathExpr)
                      | RelativePathExpr
RelativePathExpr ::= StepExpr (("/" | "//") StepExpr)*
```

이 경로 표현식을 보면, XQuery 에서는 특정 노드의 하위 노드나, 상대 경로상의 다른 노드를 찾는 기능을 사용한다고 판단할 수 있다.

4.3. 저장 구조 설계

4.1에서 이미 제시한 DOM Core 를 저장하기 위해서는 [그림 6]과 같은 구조의 정보를 유지해야 한다.



[그림 6] DOM Core 정보 저장을 위한 구조

하지만, 이 정보만 가지고는 효과적으로 XQuery 를 처리하지는 못한다. 특정 문서를 찾아가거나, 특정 노드의 상대 경로를 찾아 가기 위해서는 각 노드의 ID 를 순차적으로 따라가야만 하기 때문이다. 그 문제를 해결하기 위해서 본 논문에서는 DOM Core 정보를 이용해, 노드 경로 검색 속도를 높일 수 있는 인덱스 구조를 제안한다.

본 논문에서 제안하는 인덱스는 두 단계의 구조를 가지고 있다. 첫번째 단계는 XML 문서의 이름을 인덱싱 하는 것이다. 여기서는 B-Tree 보다 주기억 상주형 DBMS 에서 메모리나 성능 효율이 뛰어난것으로 알려진 T-Tree 를 사용한다. 두번째 단계의 인덱스는 XML 문서의 각 노드들에 대한 인덱싱을 하는 것으로, 다음과 같은 정보들로 구성된다.

- 문서 번호 : 문서에 할당된 일련번호로, 첫번째 단계의 인덱스와의 연결 고리가 된다.
- 노드 이름 : 해당 노드의 이름으로, XML 문서의 요소와 속성이 모두 포함된다.
- 경로 정보 : XML 문서 내에서 각 노드가 어떤 경로에 위치해 있는지에 대한 정보가 저장된다.
- ID : 해당 노드에 대한 정보가 실제로 DOM Core 정보를 저장한 구조의 어느 곳에 위치하고 있는지를 가리킨다.
- 속성 여부 : 각 노드는 요소와 속성을 모두 포함하는데, XQuery 에서는 속성을 별도의 경로 표기법으로 접근한다. 여기서는 해당 노드가 속성인지 요소인지를 구별한다.

여기서 XML 문서가 입력됐을 경우, 그 문서를 저장하는 방법은 다음과 같다.

1. 첫번째 인덱스에 해당 문서의 이름을 추가 한다. 첫번째 인덱스에서 관리되는 정보는 XML 문서의 논리적 저장 단위인 컬렉션 정보에 해당된다.
2. XML 문서의 각 노드를 레이블링 해 두번째 인덱스에 정보를 추가한다.
3. 첫번째 인덱스의 문서 이름과 두번째 인덱스에 저장된 노드 중 해당 문서의 최상위 노드(root node)를 연결한다.
4. XML 문서를 파싱해 DOM Core 저장 구조에 추가한다.
5. 두번째 인덱스와 DOM Core 저장 구조를 ID 를 통해 연결한다.

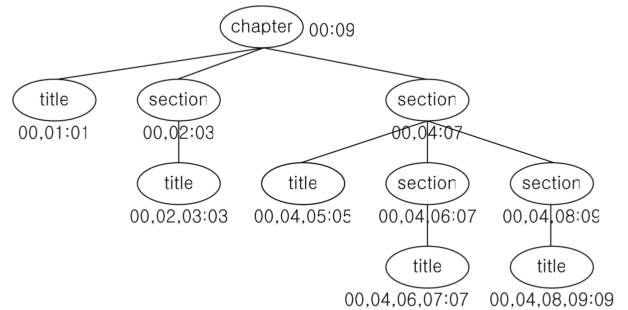
두번째 인덱스에서 각 노드에 레이블링할 때는 접두사 기반 레이블링과 구간 기반 레이블링을 혼합해 사용한다. 아래의 XML 문서를 레이블링하면 [그림 7]과 같다.

```
<chapter>
    <title>Data Model</title>
    <section>
        <title>Syntax For Data Model</title>
    </section>
    <section>
        <title>XML</title>
        <section>
            <title>Basic Syntax</title>
        </section>
        <section>
            <title>XML and Semistructured Data</title>
        </section>
    </section>
</chapter>
```

[그림 7]을 통해 레이블링 방식을 살펴보면 다음과 같다.

1. 각각의 노드에 전위순회 방식으로 번호를 매긴다.
2. 최상위 노드인 <chapter>의 경우 첫번째 노드이기 때문에 번호는 0 으로 표기하고, 뒤에는 <chapter>의 마지막 자손 노드인 <title>의 번호를 병기해 <chapter>의 범위를 알수 있게 해준다.
3. 같은 방법으로 <chapter>의 첫번째 자식 노드인 <title>의 번호는 1 이고, <title>이 단말 노드이기 때문에 마지막 범위 역시 자신의 번호인 1 로 표기 된다. 또, <title>의 경로를 표기 하기 위해, 자신의 조상 노드를 앞에 병기한다.

이와 같은 규칙은 트리의 깊이가 깊어질수록 자신의 노드 값 앞에 오는 경로 값이 길어지는 특성을 가진다. 또 이 경우, 노드의 추가/삭제가 발생할 경우, 다시 레이블링과 인덱싱을 해야 한다는 문제가 발생하지만, 본 논문의 대상 시스템이 노드 단위의 수정을 지원하지 않고, 문서 단위의 수정을 하기 때문에 영향을 받지 않게 된다.



[그림 7] XML 문서의 노드 레이블링

이와 같은 라벨링과 인덱싱 과정을 거쳐 두번째 인덱스 정보를 완성하게 된다. 이 인덱스 정보는 실제 XML 정보를 포함하고 있는 DOM Core 정보를 가리켜, 검색을 속도를 높일수 있게 된다.

5. 결론

기존의 XML DBMS 는 대부분 XML DOM 정보만을 저장하는 것이었다. 이 경우, XQuery 를 효과적으로 처리하는데는 한계가 있을 수밖에 없었으나, 본 논문에서는 기존의 XML DOM 구조 이외에도, XQuery 의 효율적 처리를 위한 인덱스 구조를 추가했다.

인덱스의 구조상 본 논문에서 설계한 저장 구조는 저장/검색/삭제 기능에서는 빠른 성능을 보일 수 있으나, 수정 기능에는 한계를 보일 수 밖에 없다. 그렇기 때문에 본 논문에서 설계한 저장구조는 수정이 빈번하게 발생하는 분야 보다는, 대량의 XML 문서가 입력된 후, 검색이 빈번하게 발생하는 분야에 적용되는 것이 적합할 것이다.

참고 문헌

[1] E.Cohen, H.Kaplan, and T.Milo. Labeling dynamic XML trees. In Proceedings of ACM Symposium on Principles of Database Systems(PODS'02), 2002

[2] N.Santoro and R.Khatib. Labeling and implict routing in networks, In The Computer J., 28:5-8, 19

[3] T. J. Lehman, M. J. Carey, "A Study of Index Structures for Main Memory Database Management Systems", in proceedings 12th Int. Conf. on very large Database, Kyoto, August 1986, pp.294-303.