효율적 XML 문서 변경 및 검색을 위한 페이징 기법*

연제원°, 이강찬, 이규철, *나중찬, *이미영
충남대학교 컴퓨터공학과 데이터베이스 연구실
*한국전자통신연구원

Paging Mechanism for Efficient XML Document Updates and Retrieval

Jewon Yeon°, Kangchan Lee, Kyuchul Lee, *Jungchan Na, *Miyoung Lee
Department of Computer Engineering, Chungnam National University
*Electronics and Telecommunications Research Institute

요 약

최근 들어 XML에 대한 연구가 늘어나면서, XML(eXtensible Markup Language)문서에 대한 저장/검색에 대한 다양한 방법들이 제시되고 있다. 특히 XML 문서의 구조적인 특성을 살리면서 문서의 저장 및 변경을 원활하게 지원할 수 있는 방안에 대한 요구가 늘고 있다. 기존의 저장관리 시스템을 위한 저장 모델로는 크게, XML문서의 빠른 검색을 지원할 수 있는 가상분할모델(Virtual Fragmentation Model)과, 문서에 대한 변경을 빠르게 지원해 줄 수 있는 분할모델(Decomposition Model)로 나누어 볼 수 있는데, 본 연구에서는 이 두 가지 모델의 장점을 취합하여 문서의 검색 속도는 가상분할모델정도로, 문서의 변경 속도는 분할모델정도로 빠르게 지원 해 줄 수 있는 페이징(Paging)기법에 대해 설계하였다. 본 페이징 기법은 XML문서뿐만 아니라, 문서 구조상 여러 개의 노드(또는 엘리먼트)로 구성되는 SGML(Standard Generalized Markup Language)이나 HTML(HiperText Markup Language)문서의 저장관리 시스템에서도 똑같이 적용 될 수 있다. 본 연구의 후반부에서는 페이징기법과 다른 기법에 대한 비교를 통하여 페이징 기법의 성능을 분석하였다.

1. 서론

1990년대 컴퓨터업계에 가장 큰 영향을 미친 것은 인터넷이라고 할 수 있다. 그리고 인터넷상의 대부분의 정보는 지금까지 HTML(HyperText Markup Language)로 작성 및 유지되어 왔다. 최근 들어서는 HTML의 단점을 극복하고 SGML의 장점을 취한 XML[1] 표준이 등장하면서 인터넷 상에서 XML문서의 활용에 대한 관심이 대폭 고조되고 있는 추세이며, 이에 대한 연구도 활발히 진행되고 있는 실정이다.

국내에서는 충남대와 테크노2000프로젝트 연구소에서 개발한 XDMS, 한국정보공학의 ReposiWare, K4M의 TheoCMS 등의 저장/검색 시스템이, 외국의 경우는 eXcelon, POET CMS등의 XML문서 저장관리 시스템등이 선을 보이고 있다. 이러한 문서 저장시스템의 경우, XML문서의 특성을 살려, 구조검색과 내용검색이 원활히 지원될 수 있도록 XML문서를 저장하여야 하며, 일반적인 텍스트문서와는 다르게 XML문서를 구성하는 단위인 노드(또는 엘리먼트)에 대한 의미를 살려 저장하고 검색할 수 있어야 한다. 지금까지 이러한 저장관리 시스템을 구현하기 위해 많은 연구가 있어왔다[2][3][4]. 본 연구에서는 지금까지 연구된 XML문서 저장기법에 대해 논의하고 이의 발전된 형태인 페이징기법에 대해 설명하며, 객체지향 데이터베이스 시스템에서의 페이징 기법에 대한 적용예를 보이도록 한다.

* 본 논문은 한국전자통신연구원의 "멀티미디어 문서 모델링 도구개발 (위탁계약번호:99-44)"과제의 일부로 수행된 결과임.

2. 관련연구

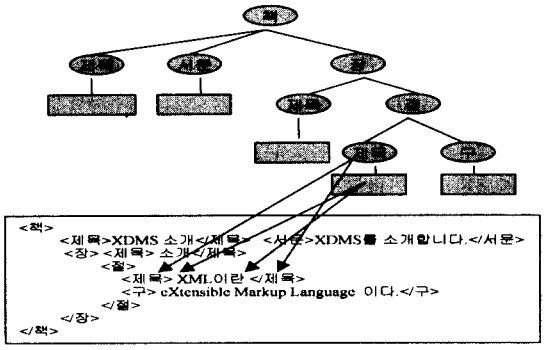
2.1. 기존의 XML 문서 저장모델

기존의 XML 문서 저장방법은 여러 가지가 있을 수 있으나 문서를 구성하는 노드의 저장방식을 기준으로 크게 두 가지로 분류할 수 있다. 첫 번째는 여러 개의 노드(또는 엘리먼트)로 구성된 XML문서의 내용을 하나의 저장공간(LOB등)에 저장한 후, 노드에 대한 검색이 발생하였을 때 노드가 가르키는 저장공간의 위치(offset)로 문서의 내용을 검색해 오는 가상분할모델(Virtual Fragmentation Model)이 있으며, 두번째는 문서를 구성하는 노드를 실제 노드별로 쪼개어 저장하는 분할모델(Decomposition Model)이 있다[5].

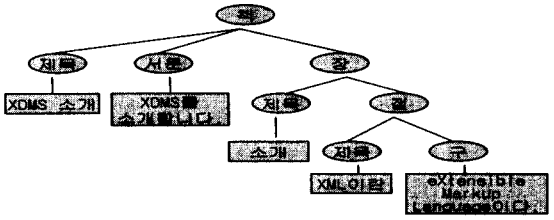
이 두 가지 모델의 경우 각각 다음과 같은 장/단점을 가진다. 가상분할 모델은, <그림 1>과 같이 검색시 시작위치와 길이 등의 정보로 저장공간에 대한 단 한번의 접근으로 검색대상을 가져올 수 있으나, 문서의 변경 즉, 문서를 구성하는 노드의 추가나 삭제의 경우 문서를 구성하는 대부분의 노드에 대한 위치정보가 변경되게 되어, 많은 수의 저장 노드에 대한 정보를 바꾸어 주어야 하는 단점이 있다.

이에 비해 분할 모델은, <그림 2>와 같이 문서 변경이 발생하였을 경우 저장된 다른 노드에 대한 정보를 바꿀 필요없이, 직접 변경된 노드(추가된 노드나 삭제된 노드)만을 변경해 주면 되기 때문에, 변경연산에 대해서는 가상분할 모델에 비해 유리하나, 검색의 경우 하위 노드에 대한 모든 내용

을 취합하는 과정을 거치기 때문에, 검색시간은 가상분할 모델에 비해 길어진다는 단점이 있다.



< 그림 1. 가상분할 모델의 예 >



< 그림 2. 분할 모델의 예 >

2.2. 기타 저장모델의 응용 형태

이러한 분할 및 가상분할 모델 이외에, 복합된 형태로 여러 가지 방법이 있을 수 있다. 예를 들면 가상분할 모델을 사용하면서 문서의 변경 기능을 지원하기 위해 따로 로그파일등을 두어 변경된 사항만 로그파일에 기록하고, 검색이 발생한 경우 가상분할 모델에 대한 내용검색을 실시하고, 후에 로그파일을 참조하는 방법이 있을 수 있다.

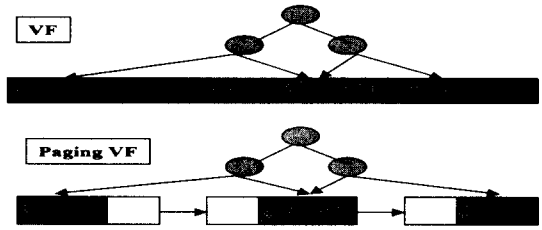
또 한가지는 분할 모델을 적용하되 최하위 터미널 노드까지 분할하지 않고 중간단계의 적정선에서 분할을 멈추고 일정크기의 노드를 저장하는 방식이 있을 수 있다. 물론 이렇게 될 경우 분할이 멈춘 이후의 노드내에 포함된 하위노드를 검색하기 위해서는 오프셋등을 활용하여, 어느 정도 가상분할 모델 형태를 취해야 한다.

3. 페이징 기법에 대한 설계

3.1 페이징 기법의 개요

<그림 3> 에서 보는바와 같이 페이징 기법은 기존의 가상분할 모델을 변형한 형태이다. 순수한 가상분할 모델이 하나의 페이지(즉 문서전체)를 사용함에 대해 페이징 기법은 문서를 여러 페이지로 쪼개어 저장하는 형태이다.

이런 방법을 사용하게 될 경우, 문서 검색 시 취합해야 하는 객체(또는 튜플)의 수가 노드의 수에서 페이지 수만큼 감소하게 되므로 분할 모델에 비해 검색은 빠르게 되고, 변경연산의 경우 바뀌어야 되는 객체(또는 튜플)의 수가 페이지내의 노드들로 제한되기 때문에 가상분할 모델에 비해 빠르게 된다.



<그림 3 페이징 기법의 개요 >

3.2 페이징 기법의 활용 예

페이징 기법에 대한 활용 예로, 바다III 객체지향 데이터베이스 시스템 상의 XML문서 저장을 위한 스키마중 페이징 기법과 관련된 부분예를 들어보면 <그림 4>와 같다.

```
class Page{
   OM_REF <Page> next:     OM_LIST <OK_VCHAR>  content;
   OK_INT contentstart:    OK_INT contentend;
}

class Document{
  OK_VCHAR docname:                 OM_REF <DocumentType> doctype;
  OM_REF <Element> rootelement:     OM_REF <Page> startpage;
}

class Node{
   OK_INT nodeType;                 OK_VCHAR nodename;
   OK_VCHAR nodevalue;              OM_REF <Page> start_page;
   OK_INT startslot;                OM_REF <Page> end_page;
   OK_INT endslot:                  OM_REF <Document> ownerDocument;
   OM_REF <Node> parent;            OM_REF <Node> next;
   OM_REF <Node> prev;              OM_LIST <Node> childlist;
   OM_LIST <NamedNodeMap> attributes;
 }
```

<그림 4. XML 저장 스키마의 부분 예>

그림에서 보듯이 저장되는 노드는 페이지 중 자신의 내용(content)이 포함되어 있는 페이지와 페이지내의 오프셋(슬롯번호)을 가지고 있다. 여기서 각 페이지의 content부분은 바다III에서 지원하는 collection클래스로 구성을 해 놓았는데, 만일 collection 클래스를 지원하지 않는 시스템에서라면 CLOB등으로 content를 구성하고, Node의 startslot과 endslot을 CLOB내에서의 offset으로 표현해도 된다. 이 경우 노드의 내용변경연산에 대해 같은 페이지내의 다른 노드가 영향을 받으므로, collection클래스로 내용을 저장하는 경우에 비해 구현이 좀 더 복잡해지는 단점이 있다.

위와 같은 구조에서 특정 노드에 대한 검색이 발생하였을 경우에는 Node의 시작 페이지와 slot위치부터, 계속 연결되는 페이지를 따라 끝 페이지의 slot위치까지 내용을 계속 추출해 오면 된다.

노드 추가의 경우에는, 노드의 내용을 페이지 내에 삽입한

후에(collection클래스), 페이지 내에서 영향을 받은 다른 노드들의 slot위치만을 변경해주면 된다. 만일 삽입되는 노드로 인해 페이지의 수가 증가하게 되는 경우에는 새로운 페이지를 할당하여 페이지 중간에 삽입하면 되고, 이로 인해 다른 페이지의 노드들은 영향을 받지 않는다.

삭제의 경우는 삭제되는 노드가 속해 있는 페이지를 수정하고, 중간의 노드가 삭제됨으로 인해 변경되는 페이지내의 노드들의 slot위치만을 바꾸어 주면 된다.

변경 연산의 경우 실제 노드의 slot위치 정보는 바꿀 필요없이, 페이지내의 collection클래스 문자열만을 변경하면 된다.

4. 분석 및 결론

본 논문에서 제안한 페이징 기법의 성능을 알아보기 위해 객체지향 데이터베이스 시스템을 기준으로, 기존의 분할 모델과 가상분할 모델의 성능에 영향을 미치는 파라미터를 분류 정의하고, 이를 수식화 하여 페이징기법의 성능에 대해 분석하였다. 성능결정 파라미터와 노드 추출(fetch) 및 변경시간에 대한 수식은 <표1>과 <그림5>와 같다.

수식에서 사용된 XML문서는 차수(degree)가 Nc이고 깊이가 Ntd인 완전트리(complete tree)의 노드로 구성된다고 가정하였다. 또한 문서의 내용부분(content)은 전부 터미널 노드로 구성된다고 가정 하였다.(엘리먼트의 태그부분은 내용에서 제외). <그림 6>의 성능비교 그래프는 Tor=1ms, Tow=3ms, Nc=10, Ntd=6, Sn=20, Sp=4000, Sps=1000으로 놓고 계산한 결과이다.

<표 1. 성능분석을 위한 파라미터>

- 영구객체 Load시간 : Tor
- 영구객체 Write 시간 : Tow
- 페이지 크기 : Sp
- 평균 페이지 여백 크기 : Sps
- 문서의 크기 : Sd (bytes)
- 문서의 터미널 노드 개수 : Nn
- 터미널 노드의 평균 길이 : Sn(bytes)
- 엘리먼트의 평균 차일드 노드 개수 : Nc
- 문서의 평균 깊이 (트리구조상 depth) : Ntd
- 페이지당 평균 터미널 노드 개수 : Nnpp (= (Sp-Sps) / Sn)
- 노드 fetch시 평균 참조 터미널 노드 개수 : Nrn
- 노드 fetch시 평균 페이지 참조 횟수 : Nrp (= Nrn / Nnpp)
- 노드 변경(터미널 노드)시 평균 변경 터미널노드 개수 : Nu (= (1/2)xNnpp)
- 평균 변경 시간 : Tu
- 평균 노드 fetch 시간 : Tf

❑ **Decomposition**

Tf = 터미널/관련조상노드 fetch시간

$$= \sum_{i=1}^{Nrp}\left[(Nrn/Nc^{(n-i)})\right] \times Tor$$

Tu = Tor + Tow

=> 어느 노드를 수정하더라도 변경 대상 객체(Nu)는 하나

❑ **Virtual Fragmentation**

Tf = 노드 객체 가져오는 시간 + 페이지 객체 가져오는 시간

= Tor + Tor

Tu = (전체노드의 절반) x (Tor+Tow) + 페이지 객체 변경

$$= \sum_{i=1}^{Nrp}\left[((1/2)Nn/Nc^{(n-i)})\right] \times (Tor+Tow) + (Tor+Tow)$$

$$= (\sum_{i=1}^{Nrp}\left[((1/2)Nn/Nc^{(n-i)})\right] + 1) \times (Tor+Tow)$$

❑ **Paging VF 기법**

Tf = 노드 객체 가져오는 시간 + 페이지 객체 가져오는 시간

= 1 x Tor + Nrp x Tor

= 1 x Tor + ((Nrn x Sn) / (Sp-Sps)) x Tor

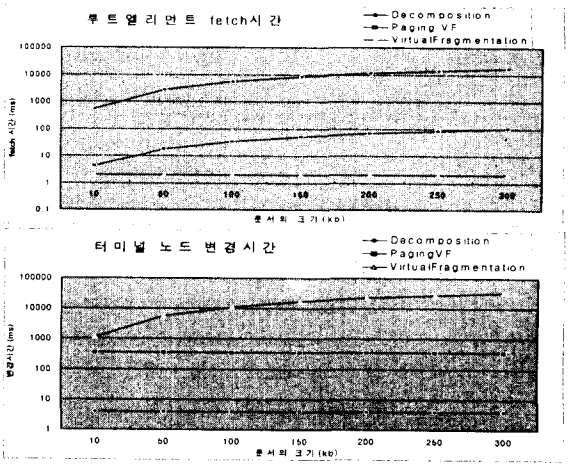
= Tor (1 + (Nrn x Sn) / (Sp-Sps))

Tu = 터미널/관련조상노드 변경 + 페이지 변경

$$= \sum_{i=1}^{Nrp}\left[(Nu/Nc^{(n-i)})\right] \times (Tor+Tow) + (Tor+Tow)$$

$$= (\sum_{i=1}^{Nrp}\left[\ (1/2 \times ((Sp-Sps)/Sn) / Nc^{(n-i)})\right] + 1) \times (Tor+Tow)$$

<그림 5. 각 기법에서의 노드fetch 및 변경 성능>



<그림 6. fetch 및 변경 성능 그래프>

<그림 6>에서 보듯이, 분할모델이나 가상분할모델은 각각, 검색 및 변경 문서가 커지는데 따라, 상당히 큰 추출/변경 시간(수십~수백sec)을 가지게 되나, 페이징 기법을 활용하여 문서를 추출/변경하게 되면 수초 이내에 원하는 작업을 수행할 수 있다. 이러한 페이징 기법을 사용하여 앞으로는 좀 더 효율적인 XML문서 저장관리 시스템을 개발할 수 있을 것이다.

5. 참고 문헌

[1] Extensible Markup Language(XML),"http://www.w3.org/TR/PR-xml-971208"

[2] 연제원, 조정수, 이강찬, 이규철 "XML 문서 구조검색을 위한 저장 시스템 설계", 한국정보과학회 학술발표 논문집(B), 26권 1호, 1999

[3] 이용석, 손기락, "XML문서 저장 시스템 설계 및 구현", 정보과학회 학술 발표 논문집(I), 25권 2호, 1998

[4] 장식원, 신용태, "XML문서 저장과 편집 어플리케이션 구현", 한국정보과학회 학술발표 논문집(B), 26권 1호, 1999

[5] 연제원, 장동준, 김용훈, 이강찬, 이규철, 김현기, 노대식, 강현규 "효율적인 검색지원 SGML 저장관리기의 설계 및 구현", 한국 데이터베이스 학술대회 논문집, 1999