대용량 멀티미디어 객체를 위한 객체 저장 엔진의 설계 및 구현

진기성° 장재우
전북대학교 컴퓨터공학과
{ksjin, jwchang}@dblab.chonbuk.ac.kr

Design and Implementation of Object Storage Engine for Large Multimedia Objects

Ki-Sung Jin° Jae-Woo Chang
Dept. of Computer Engineering, Chonbuk National University

요 약

최근 텍스트, 이미지, 오디오, 비디오와 같은 멀티미디어 객체를 다루는 연구는 국내외적으로 활발하게 진행되고 있으나, 이러한 멀티미디어 객체들을 효율적으로 저장 및 검색하기 위한 하부 저장 시스템에 대한 연구는 미흡한 실정이다. 본 연구에서는 이러한 대용량 멀티미디어 객체들을 효율적으로 저장 및 검색하기 위한 구조를 분석하고 다양한 이질적 객체들을 위한 객체 관리자 및 대용량 텍스트를 위한 역화일 관리자를 설계한다. 아울러, 기존의 하부저장 구조인 SHORE 저장시스템에 통합하여 DBMS 측면에서 제공하는 동시성 제어, 회복기법 등을 지원할 수 있는 객체 저장 엔진을 구현한다.

1. 서론

최근 정보통신 기술의 급속한 발달로 인해 이미지, 오디오, 비디오와 같은 다양한 미디어로 구성된 대용량의 멀티미디어 자료를 효율적으로 저장하고 관리해야 하며 이러한 환경에서 사용자들은 다양한 형태의 멀티미디어 서비스를 요구하는 추세이다. 현재 데이터베이스 시스템들은 텍스트 또는 수치정보 서비스에 적합한 구조로서 관계형 DBMS 또는 파일시스템을 기반으로 데이터베이스를 관리하고 있으나, 멀티미디어 응용을 위해서는 기존 관계형 DBMS 를 확장하여 멀티미디어 자료를 다루는 초보적인 단계로서 많은 한계성을 드러내고 있다.

이를 위하여 본 논문에서는 다양한 객체들을 효율적으로 저장 및 검색하기 위한 구조를 분석하고, 미국 위스콘신 대학에서 개발한 SHORE 하부저장 시스템을 확장한다. SHORE 에서는 대용량 텍스트, 이미지, 비디오 등 매우 큰 비정형 객체를 위한 저장 방법으로 4GB 까지 확장 가능한 가변길이 레코드를 제공하며, 아무리 작은 데이터라도 기본적으로 한 페이지를 할당한다. 그러나 멀티미디어 객체의 경우 크기가 매우 가변적이기 때문에 단순한 삽입연산을 통하여 저장하는 경우 많은 저장 공간의 낭비를 초래할 뿐만 아니라 성능의 감소를 가져온다. 따라서 가변적인 대용량 문서들에 대한 효율적 저장 방법이 필요하다. 이를 위해, 본 논문에서는 이미지, 비디오와 같은 대용량의 비정형 객체를 효율적으로 저장 및 검색할 수 있는 객체 관리자를 설계하고, 이를 위한 사용자 인터페이스로서 객체관리 API 를 구현한다. 아울러, 텍스트 검색을 위한 색인 기법으로는 벌크로딩을 응용한 역화일 기법[1]을 SHORE level_2 에 확장한다.

본 논문의 구성은 다음과 같다. 제 2 장에서는 관련연구로서 SHORE 저장 시스템을 소개한다. 제 3 장에서는 대용량 객체를 지원하는 객체 저장 엔진의 설계 및 구현에 대해 설명하고, 제 4 장에서는 결론 및 향후 연구과제를 제시한다.

2. 관련연구

SHORE(Scalable Heterogeneous Object Repository)는 미국의 위스콘신 대학에서 개발한 지속성 객체 저장 시스템으로 기존의 OODB(Object-Oriented Database)에서의 단점인 파일 시스템과의 결합의 어려움, 트랜잭션(Transaction)관리의 어려움, Peer-to-Peer 구조의 부적절성을 해결하기 위해 개발되었다[2]. SHORE 는 객체지향 데이터베이스 기술과 파일시스템 기술을 합성하여, 기존의 UNIX 파일 시스템 기반 응용 프로그램을 쉽게 접목시킬 수 있는 장점을 지닌다. SHORE 는 기존의 EXODUS[3] 저장 관리자를 확장한 것으로 5 단계의 계층구조를 가진다. 또한 보다 우수한 성능을 얻기 위해 Raw disk 방식을 이용한 물리적 공간을 관리하여, 디스크의 일정 영역을 운영체제의 버퍼를 거치지 않고 원하는 시점에 동기적으로 기록할 수 있게 한다. 다량의 자료가 저장된 파일에 대하여 특정 키를 통해 범위 검색 접근을 제공하기 위해 B+트리 형태의 인덱스를 제공하고, 파일의 크기가 동적으로 변하는 환경에서 원하는 레코드의 빠른 접근을 위해 논리적 식별자 관리기, 캐쉬 관리기를 제공한다. SHORE 를 이용하여 구현된 시스템으로는 GIS 응용을 위해 개발된 PARADISE[4]와 Cornell 대학에서 수행한 PREDATOR[5]가 있다. PARADISE 는 지속성 객체(persistent object) 관리자를 위해 SHORE 를 기반으로 2-D 지도상에 공간적인 속성 객체들을 이용한 Client-Server 구조로 구현되었고, PREDATOR 는 객체-관계(Object-Relational) DBMS 로써 SHORE 의 SVAS 를 이용하여 구현한 시스템이다.

3. 대용량 객체를 위한 객체 저장 엔진의 설계

본 장에서는 멀티미디어 객체를 효율적으로 저장 및 검색하기 위한 객체저장 엔진을 설계한다. 이미지, 비디오와 같은 비정형 객체들을 효율적으로 관리하기위해 기존의 SSM API 를

개선한 객체 관리자를 설계 및 구현하고, 텍스트 검색을 위해 역화일 기법을 SHORE Level_2 에 확장한다. 설계된 객체 관리자와 역화일 관리자를 통합한 SHORE 저장 시스템의 전체적인 구조는 그림 1 과 같다.

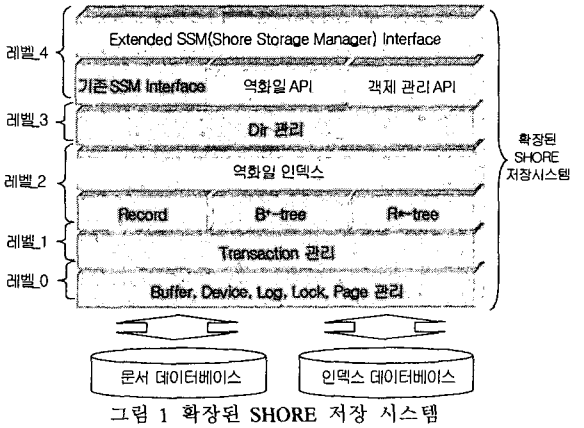

그림 1 확장된 SHORE 저장 시스템

3.1 멀티미디어 객체를 위한 객체 관리자

다양한 크기의 이질적인 멀티미디어 객체들을 저장 및 관리하기 위해 SHORE 를 이용하여 객체 관리자를 설계 및 구현한다. 이를 위해 cOBJECT_m 이라는 관리자 클래스를 설정하고 ss_m 클래스로부터 상속 받아 구현한다. cOBJECT_m 클래스는 크게 다음과 같은 4 가지 모듈 및 API 로 구성된다.

- Device Management
  - CreateDB() // 볼륨의 생성
  - OpenDB() // 볼륨의 개방
  - CloseDB() // 볼륨의 삭제
- File Management
  - CreateFile() // 파일의 생성
  - DestroyFile() // 파일의 삭제
  - OpenFile() // 파일의 개방
  - CloseFile() // 파일의 종료
- Object Management
  - CreateObject() // 오브젝트 생성
  - DestroyObject() // 오브젝트 삭제
  - GetObjectLen() // 오브젝트 길이 획득
  - ReadObject() // 오브젝트 검색
  - InsertIntoObject() // 오브젝트 부분 삽입
  - DeleteFromObject() // 오브젝트 부분 삭제
  - AppendToObject() // 오브젝트 끝에 첨가
  - TruncateObject() // 오브젝트 끝에서부터 삭제
  - FirstObjectID() // 첫번째 오브젝트 ID 반환
  - LastObjectID() // 마지막 오브젝트 ID 반환
  - NextObjectID() // 다음 오브젝트 ID 반환
  - PrevObjectID() // 이전 오브젝트 ID 반환
- Index Management
  - CreateIndex() // B+트리 인덱스 생성
  - DestroyIndex() // B+트리 인덱스 삭제
  - OpenIndex() // B+트리 인덱스 개방
  - CloseIndex() // B+트리 인덱스 종료
  - InsertEntity() // B+트리 Entity 삽입
  - DeleteEntity() // B+트리 Entity 삭제
  - BeginLoadEntity() // B+트리 벌크로딩 시작
  - LoadEntity() // B+트리 벌크로딩
  - EndLoadEntity() // B+트리 벌크로딩 종료

또한 다음은 제공된 객체 관리 API 를 이용하여 파일을 생성하고 객체를 생성하는 예제이다.

```
(1)W_DO(OBJECT->begin_xct());
(2){
(3)    W_DO(OBJECT->CreateFile(DBid, "News96", Fid));
(4)    W_DO(OBJECT->CreateObject(DBid,Fid, buf, buflen, Oid));
(5)}
(6)W_DO(OBJECT->commit_xct());
```

(1)과 (6)는 트랜잭션의 시작과 종료를 의미하고, 트랜잭션의 내부에서 (3)과 같이 "News96"이란 파일을 생성하고 (4)는 오브젝트를 생성하여 버퍼의 내용을 삽입한다.

3.2 텍스트 문서를 위한 역화일 관리자

대량의 텍스트 문서를 효율적으로 저장 및 관리하기 위해 벌크로딩을 응용한 역화일 기법을 SHORE level2 에 확장한다. 일반적인 역화일 기법의 경우 B+트리와 포스팅 파일로 구성되어지는데, 각각의 키 값마다 포스팅 레코드를 생성 또는 추가하는 작업이 이루어지기 때문에 수많은 I/O 가 발생할 뿐만 아니라 부가저장 공간의 측면에서도 상당한 성능의 감소를 가져오게 된다. 따라서 하나의 키 값에 해당하는 포스팅 레코드의 내용들을 벌크로딩을 통해 SHORE 의 버퍼 관리자에 담아두고 해당 키의 작업이 끝났을 때 일괄적으로 저장함으로써 성능을 향상시킬 수 있다. 이를 위해 본 논문에서는 SHORE 를 확장하기위한 개선된 역화일 관리자 클래스인 invert_m 클래스를 정의한다. B+트리와 포스팅 파일은 SHORE 에서 제공하는 메소드를 이용하며, 동시성을 제공하기 위해 트랜잭션 관리자와 잠금 관리자를 이용하여 설계한다. 또한 포스팅 파일의 경우 벌크로딩시에 가장 우수한 성능을 가질 수 있도록 Extent 의 수를 적절히 선정하여 설계한다. 한편 포스팅 파일의 구성이 모두 종료한후 B+트리의 구성도 벌크로딩을 이용하여 저장할수 있도록 하며 이때 B+트리는 SHORE 에서 제공하는 메소드를 이용한다. 한편 확장된 역화일의 검색을 위해 본 논문에서는 커서개념을 이용한 검색 API 를 설계한다. 커서는 다수의 질의 결과가 있을 때 현재 커서의 위치를 이동해 가면서 결과를 반환하는 방식으로, 질의 형태로서는 색인어의 정확검색과 우절단 검색을 지원한다. 이러한 확장된 역화일의 API 는 표 1 와 같다.

표 1. SHORE 에 확장된 역화일 관리자 API

확장된 역화일 API	기능
CreateInvtIndex()	역화일 인덱스 생성
DestroyInvtIndex()	역화일 인덱스 삭제
InsertDocInvt()	역화일 구조에 하나의 문서삽입
InsertBulkInvt()	역화일 구조에 벌크문서 삽입
DeleteDocInvt()	역화일 구조에서 하나의 문서삭제
SetCursor()	확장 검색을 위한 커서 초기화
NextCursor()	다음 위치로 커서를 이동
GetCursor	현재 위치의 검색 결과를 반환

4. 구현 및 성능고찰

멀티미디어 객체의 효율적 저장 및 검색을 위한 객체 저장 엔진을 구현 및 성능 고찰을 위해 최신 버전인 SHORE2.0 을 사용하고, 구현환경은 CPU 650 MHz dual, 메모리 512MB 의 리눅스 서버에서 수행하였다. 본 논문에서 제시하는 객체 저장 엔진은 비정형 문서들을 저장하기 위한 논리적 단위로서 Store 를 사용한다. 하나의 Store 는 기본적으로 두개의 Extent 로 구성되어지며 하나의 Extent 는 8 개의 페이지로 구성된다. 이때 첫번째 Extent 는 Small Object 를 위한 공간이며 두 번째 Extent 는 Large Object 를 위한 공간으로 예약된다. 각각의 Extent 는 64Kb 크기로서 새로운 객체를 위한 공간이 없을 경우 빈 Extent 를 병합하여 Store 를 확장하여 사용하고 이론적으로 4Gb 까지 확장이 가능하다. 그러나 하나의 Store 를 확장하여 사용하는 경우 새로운 Extent 를 찾아 데이터를 삽입하는 과정에서 많은 계산시간으로 인해 성능이 감소하게 되고, 각각의 삽입 연산에 대해서 개별적인 Store 를 생성하여 사용할 경우 이미 예약되어진 두개의 Extent 에 저장 되지 않은 빈 공간들이 무수히 많아지기 때문에 저장공간의 낭비를 초래한다. 따라서 대량의 문건에 대해 효율적인 저장 및 검색을 위해서는 삽입성능과 부가저장공간의 두 가지 측면을 만족시킬 수 있는 방법이 필요하다.

표 2. Store 별 성능 비교

문서 크기	구분	방법 1	방법 2	방법 3	방법 4	방법 5
1.7 K 20 만건	삽입 (sec)	2501	2427	2413	2442	2874
	검색 (sec)	0.04	0.04	0.04	0.04	0.04
	SO (%)	224	134	123	120	118
1 ~ 3K 20 만건	삽입 (sec)	2651	2523	2511	2595	3116
	검색 (sec)	0.04	0.04	0.04	0.04	0.04
	SO (%)	289	144	124	118	116

이를 위해 본 논문에서는 1.7 Kb 크기의 문건 20 만 건과, 1 ~ 3Kb 의 가변적 크기를 가진 문건 20 만 건을 이용하여 다음 표 2 과 같은 실험을 수행한다. 각각의 실험을 위해 문서 20 만건과 문서의 파싱결과를 이용하여 역화일을 구성한다. 이때 방법 1 은 하나의 Store 가 2 개의 Extent 만을 가지고 공간이 꽉 찼을 경우 새로운 Store 를 할당하는 방식이고, 방법 2 에서 방법 5 까지 Store 마다 각각 5 개, 10 개, 15 개, 20 개의 Extent 를 할당 받는 방법이다. 검색시간 측면에서 보면 하나의 문서 검색시 0.004sec 로 동일한 결과를 보이고 있다. 삽입시간의 경우 방법 3 이 가장 우수한 성능을 보이고 방법 1, 방법 5 은 많은 시간이 소요됨을 알 수 있다. 방법 1 의 경우는 소량의 문서마다 새로운 Store 를 빈번히 생성하기 때문이며, 방법 5 는 20 개의 Extent 에서 데이터를 삽입할 공간을 찾는 계산시간의 증가로 성능이 감소된다. 한편, 부가저장 공간 측면에서는 방법 5 가 가장 우수하다. 이는 매번 새로운 Store 를 생성할 때 발생하는 빈공간의 낭비를 최소화 하였기 때문이다. 따라서 삽입시간과 부가저장 공간의 효율성을 만족시키는 방법을 찾는 식은 식 1 과 같다.

식 1. $Weight = (I * Wi) + (SO * Wso)$

위 식에서 Wi 는 삽입시간의 중요도, Wso 는 부가저장 공간의 중요도이다. 먼저 삽입시간과 부가저장 공간의 수치를 0 에서 1 까지 정규화하여 각각의 실험에 적용한다. 그림 2 는 Wi 와 Wso 를 각각 1 의 가중치를 주었을 때의 결과로서, 10 개의 Extent 를 사용한 방법 3 의 경우가 가장 효율적인 성능을 보인다.

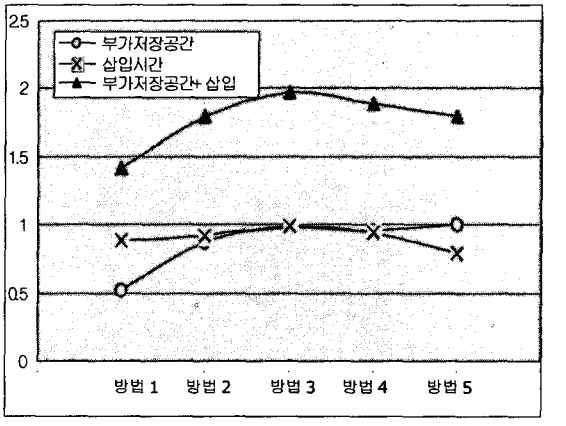

그림 2. 가중치에 따른 성능비교

5. 결론

본 논문에서는 대용량의 텍스트, 이미지, 비디오와 같은 멀티미디어 객체를 효율적으로 관리하기 위해 기존의 SHORE 하부저장 시스템을 확장하여 다양한 객체들을 위한 객체 저장 엔진을 설계 및 구현하였다. 이를위해 SHORE 저장 시스템의 내부에서 각 객체들이 저장되는 방식을 분석하고 하나의 Store 가 확장될 수 있는 최적의 환경을 찾아내어 이를 통해 객체 저장 엔진을 구현하였다. 먼저 이미지, 비디오와 같은 이질적인 멀티미디어 객체를 위해서 볼륨, 파일, 오브젝트, 인덱스를 통합 관리할 수 있는 객체 관리자를 설계 및 구현하였고, 대량의 텍스트 문서를 위해서 벌크로딩을 응용한 개선된 역화일 인덱스를 SHORE 의 level 2 에 삽입함으로써 효율적인 문서 관리와 함께 DBMS 측면의 동시성 제어, 회복기법 등을 지원할 수 있도록 하였다.

향후 과제로는 본 논문에서 설계 및 구현된 객체 저장 엔진을 기반으로 다양한 멀티미디어 응용에 적용하여 제안된 시스템의 성능을 평가하는 것이다.

6. 참고 문헌

[1] W.B. Frakes and R. Baeza-Yates, "Information Retrieval data Structures & Algorithms",Pretice Hall, 1992

[2] M.J. Carey, et al, "Shoring Up persistent Applications", in Proc. Of ACM SIGMOD, VOL.23, No.2, pp.383-394, 1994

[3] M. J. Carey, et al., "The architecture of the EXODUS Extensible DMBS," in Proc. VLDB, 1986.

[4] DeWitt, D., Luo, J., Patel, J., and Yu, J., "Paradise – A Parallel Geographic Information System." In Proc. of the ACM Workshop on Advances in Geographic Information Systems, November 1993.

[5] Praveen Seshadri, Mark Paskin, "PREDATOR: An OR-DBMS with Enhanced Data Types", SIGMOD, 1997