대용량 멀티미디어 객체를 위한 객체저장엔진의 설계 및 구현

(Design and Implementation of Object Storage Engine for Large Multimedia Objects)

진 기 성 † 장 재 우 ††
(Ki-Sung Jin) (Jae-Woo Chang)

요 약 최근 멀티미디어 객체를 다루는 연구는 국내외적으로 활발하게 진행되고 있으나, 이러한 멀티미디어 객체들을 효율적으로 저장 및 검색하기 위한 하부저장 시스템에 대한 연구는 미흡한 실정이다. 본 연구에서는 이러한 대용량 멀티미디어 객체들을 효율적으로 저장 및 검색하기 위한 객체 저장 엔진을 구현한다. 이를 위해, 비정형 멀티미디어 객체의 저장을 위한 객체 관리자와, 비정형 텍스트 객체의 색인을 위한 역화일 관리자를 설계한다. 아울러, 설계된 객체 관리자와 역화일 관리자를 기존의 하부저장 구조인 SHORE 저장시스템에 통합하여 DBMS 측면에서 제공하는 동시성 제어, 회복기법 등을 지원할 수 있는 객체 저장 엔진을 구현한다. 마지막으로, 구현된 객체저장엔진의 유용성을 검증하기 위해 논문검색시스템 TIROS(Thesis Information Retrieval system using Object Storage engine)를 구축한다.

키워드 : 객체저장엔진, 멀티미디어 데이타, 하부저장시스템

Abstract Recently, although there are strong requirements to manage multimedia data, there are a few researches on efficient storage and retrieval of multimedia data. In this paper, we design an object storage engine which can store and retrieve various multimedia objects efficiently. For this, we design an object manager for storing a variety of multimedia data and an inverted file manager for indexing unformatted text objects. In addition, we implement the objects storage engine which can support concurrency control and recovery schemes of DBMS by integrating the object manager and the inverted file manager with the SHORE low-level storage system. Finally, we develope a TIROS(Thesis Information Retrieval using Object Storage engine) system in order to verify the usefulness of our object storage engine.

Key words : Object Storage engine, multimedia data, low-level storage system

1. 서 론

최근 정보통신 기술의 급속한 발달로 인해 이미지, 오디오, 비디오와 같은 다양한 미디어로 구성된 대용량 멀티미디어 자료를 효율적으로 저장하고 관리해야 하며, 이러한 환경에서 사용자들은 다양한 형태의 멀티미디어 서비스를 요구하는 추세이다. 현재 데이타베이스 시스템들은 텍스트 또는 수치정보 서비스에 적합한 구조로서 관계형 DBMS 또는 파일시스템을 기반으로 데이타베이스를 관리하고 있으나, 멀티미디어 응용을 다루기 위해서는 기존 관계형 DBMS를 멀티미디어 자료를 다룰 수 있도록 확장하는 것이 필요하다.

멀티미디어 객체를 위한 하부 저장 시스템을 구축하는 방법으로는 첫째, 기존의 저장 시스템 내부에 멀티미디어 객체를 위한 기법을 구현하는 밀결합 방법과, 둘째, 시스템이 제공하는 인터페이스를 이용하여 외부에서 구현하는 소결합 방법이 있다[1]. 밀결합 방식의 경우 DBMS측면에서 제공하는 동시성 제어, 무결성 유지, 회복기법 등을 효율적으로 지원하는 반면, 확장된 시스템의 정적인 구조로 인해 다양한 종류의 멀티미디어 객체

† 정 회 원 : 한국전자통신연구원 연구원
ksjin@dblab.chonbuk.ac.kr
†† 종신회원 : 전북대학교 전자정보공학부 교수
전자정보신기술연구센터 연구원
jwchang@dblab.chonbuk.ac.kr
논문접수 : 2000년 12월 20일
심사완료 : 2002년 5월 4일

에 능동적으로 대처하지 못하는 단점을 지닌다. 반면 소결합 방식은 동시성 제어, 무결성 유지, 회복기법 등을 보장하지는 못하지만 다양한 종류의 멀티미디어 객체들에 대해 유연성있는 저장 및 접근 기법을 제공할 수 있는 장점이 있다. 따라서 텍스트, 이미지, 비디오와 같은 멀티미디어 객체를 위한 하부저장 시스템을 구축하기 위해서는, 비정형의 다양한 종류의 객체를 다루기에 적합한 소결합 방식을 사용하는 것이 바람직하다.

본 논문에서는 멀티미디어 객체를 다루기 위한 객체 저장 엔진을 설계하고, 이를 위해 이미지, 오디오, 비디오와 같은 다양한 비정형 객체를 위해 객체 관리자를, 대용량 텍스트를 위해 역화일 관리자를 미국 위스콘신 대학에서 개발한 효율적인 SHORE(Scalable Heterogeneous Object Repository) 하부 저장 시스템에 소결합하여 구현한다. 객체 저장 관리자는 가변적 길이의 비정형 멀티미디어 객체를 효율적으로 관리하기 위해서 데이타베이스, 파일, 오브젝트, 인덱스 모듈로 분류하여 각각의 기능을 독립적으로 수행하도록 설계한다. 또한, 역화일 관리자는 대용량 텍스트에 대한 효율적 관리를 위하여 벌크로딩 기법을 적용한다. 역화일 관리자는 벌크로딩을 통해 하나의 키 값에 해당하는 포스팅 레코드의 내용들을 버퍼 관리자에 담아두고 해당 키의 작업이 끝났을 경우 일괄적으로 저장함으로써 성능을 향상시킬 수 있도록 설계한다.

본 논문의 구성은 다음과 같다. 제2장에서는 관련연구로서 SHORE 하부 저장 시스템 및 확장된 SHORE 저장 시스템에 대하여 소개한다. 제3장에서는 대용량 멀티미디어 객체를 지원하는 객체 저장 엔진의 설계를 위해 SHORE 하부저장 시스템을 사용하여 소결합된 객체 관리자와 역화일 관리자를 제시한다. 아울러 제4장에서는 구현된 객체 저장 엔진의 성능을 평가하고, 제5장에서는 구현된 객체 저장 엔진을 사용하여 논문 정보검색 시스템인 TIROS(Thesis Information Retrieval Using Object Storage engine)를 구축한다. 마지막으로 제6장에서는 본 논문에 대한 결론 및 향후 연구방향을 제시한다.

2. 관련연구

2.1 SHORE 하부 저장 시스템

SHORE[2, 3]는 미국의 위스콘신 대학에서 개발한 지속성 객체 저장 시스템으로 기존의 OODB(Object-Oriented Database)의 단점인 파일 시스템과의 결합의 어려움, 트랜잭션(Transaction)관리의 어려움, Peer-to-Peer 구조의 부적절성을 해결하기 위해 개발되었다. SHORE는 객체지향 데이타베이스 기술과 파일시스템 기술을 합성하여 기존의 UNIX 파일 시스템 기반 응용 프로그램을 쉽게 접목시킬 수 있다. [그림 1]은 SHORE 하부 저장 시스템의 구조를 나타낸다.

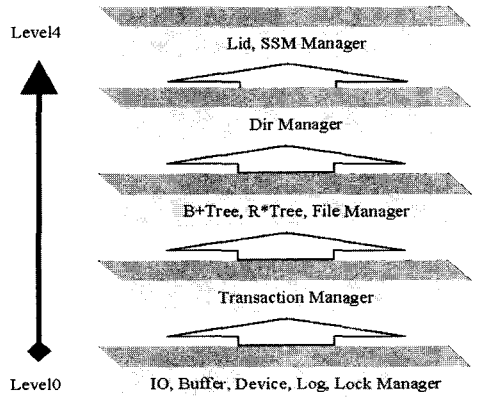

그림 1 SHORE 하부 저장 시스템의 구조

SHORE는 UNIX 시스템에서 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. 또한 각 계층의 모든 작업들에 대해 쓰레드 단위의 작업 할당을 함으로써 다수 사용자에 대한 트랜잭션 처리에 용이한 구조를 가지고 있으며, 외부 서버와의 데이타 교환을 위한 통로로서 SVAS(Shore Value-Added Server)와 SSM(Shore Storage Manager)로 구성되어 있다. SVAS는 원격 시스템간에 객체들을 서로 원활하게 통신할 수 있도록 관리하며, SSM은 지역 시스템에서 생성되고 처리되는 모든 객체들을 관리한다.

SHORE를 이용하여 구현된 시스템으로는 GIS응용을 위해 개발된 PARADISE[4]와 Cornell 대학에서 수행한 PREDATOR[5]가 있다. PARADISE는 지속성 객체(persistent object) 관리자를 위해 SHORE를 기반으로 2D 지도상에 공간적인 속성 객체들을 이용한 Client-Server 구조로 구현되었고, PREDATOR는 객체-관계(Object-Relational) DBMS로서 SHORE의 SVAS를 이용하여 구현한 시스템이다.

2.2 확장된 SHORE 하부 저장 시스템

멀티미디어 응용을 위해 확장된 SHORE 하부 저장

시스템은[6] 텍스트 검색을 위해 역화일을, 이미지 및 비디오 특징벡터 검색을 위해 고차원 색인구조인 x-tree[7]를 SHORE에 밀결합 방식으로 통합하였다. 확장된 SHORE 하부 저장 시스템에서는 역화일 구조를 위해 SSM의 레벨 2에서 invertedfile_m 클래스를 제공한다. 역화일 구조는 단어 색인 파일로는 SSM에서 제공하는 B+트리를, 포스팅 파일로는 SSM 파일을 사용하였고, 각각 btree_m 클래스와 file_m클래스로부터 상속받아 구현되었다.

또한, 확장된 SHORE 하부 저장 시스템에서는 X-트리 인덱스를 위해 SSM의 레벨 2에 xtree_m 클래스를 제공한다. 특징벡터 색인 관리자인 xtree_m 클래스는 특징벡터들의 저장, 검색에 관한 메쏘드를 제공하며, 페이지 관리를 위한 xtree_head_p, xtree_dir_p, xtree_data_p 클래스를 이용하여 구현되었다. xtree_head_p 클래스는 x-트리의 헤더로서 루트페이지 식별자, 부모 페이지가 데이타 페이지인지의 여부, 차원의 수, 중간 노드의 수 등의 정보를 관리한다. xtree_data_p 클래스는 데이타 노드를 관리하며, xtree_dir_p 클래스는 x-트리의 중간 노드를 저장 및 관리하며, 수퍼노드인 경우 여러 페이지가 하나의 노드로 구성되어 있으므로 연관된 페이지들의 식별자들을 유지한다. 또한 x-트리의 확장된 검색을 위해 scan_xd_i 클래스를 제공한다. scan_xd_i 클래스는 포인트 질의, 범위 질의, k-최근접 질의[8]를 제공하며, 생성자에 의해 질의 방법과 질의 벡터를 지정한다.

3. 대용량 객체를 위한 객체 저장 엔진의 설계

3.1 객체 저장 엔진 설계시 고려사항

멀티미디어 응용에 적합한 객체 저장 엔진을 처음부터 개발하는 것은 매우 어려우며 또한 많은 시간이 걸리는 작업이기 때문에, 기존 하부저장 시스템을 이용하여 구축하는 방법이 바람직하다. 기존 하부저장 시스템을 이용하여 대용량 멀티미디어 객체들을 위한 객체 저장 엔진을 구축하는 방법으로는 밀결합 방식과 소결합 방식이 있다. 밀결합 방식의 경우 DBMS측면에서 제공하는 동시성 제어, 무결성 유지, 회복기법 등을 효율적으로 지원하는 반면, 확장된 시스템의 정적인 구조로 인해 다양한 종류의 멀티미디어 객체에 능동적으로 대처하지 못하는 단점을 지닌다. 반면 소결합 방식의 경우 동시성 제어, 무결성 유지, 회복기법 등을 완전히 보장하지는 못하지만 다양한 종류의 멀티미디어 객체들에 대해 유연성 있는 접근 기법을 제공할 수 있는 장점이 있다. 따라서 텍스트, 이미지, 비디오와 같은 멀티미디어 객체를 위한 하부저장 시스템을 구축하기 위해서는, 다양한 비정형 객체를 다루기에 적합한 소결합 방식을 사용하는 것이 바람직하다. 한편 기존 하부 저장 시스템으로는 안정성과 효율성을 갖춘 미국 위스콘신 대학에서 개발한 SHORE 하부 저장 시스템이 적합하다. 아울러 기존의 SHORE 저장 시스템을 이용하여 대용량 멀티미디어 객체를 위한 객체 저장 엔진을 구축하는데 다음과 같은 설계상의 고려사항이 필요하다.

첫째, 멀티미디어 객체들의 효율적인 관리를 위해 설계된 기법들은 하나의 객체 속성에 종속되지 않게 범용성을 지녀야 한다. 따라서 설계된 객체 관리자 및 역화일 관리자는 객체 지향 언어를 이용하여 객체들을 관리할 수 있어야 하며, 응용 프로그램에서는 시스템에서 제공하는 메소드를 이용하여 각 객체의 식별자로서 접근할 수 있어야 하고, 각 객체의 식별자를 통해 데이타베이스에 저장된 객체의 내용을 관리할 수 있어야 한다. 이때, 멀티미디어 문서를 관리하기 위한 모든 엔터티는 각 객체로 모델링되며, 각 객체는 시스템 내에서 유일한 식별자를 가진다. 따라서 사용자는 객체의 속성에 관계없이 객체의 식별자를 통해 접근이 가능해야 하며, 획득한 정보에 대한 신뢰성을 줄 수 있도록 객체 지향 모델에 적합한 시스템을 설계해야 한다.

둘째, SHORE 저장 시스템이 제공하는 자원들을 효율적으로 이용하여 객체들의 안정성을 보장해야 한다. 기존의 SHORE 저장 시스템은 모든 객체들의 안정성을 위해 고유의 논리적 식별자를 사용하고 이를 통한 물리적 디바이스에 접근시에 잠금 및 회복 기능을 지원한다. 이때 잠금의 경우에는 하위 계층에 구현된 잠금관리자를 사용하고, 회복의 경우에는 회복관리자를 사용하여 설계해야 한다. 또한 임의의 객체에 관련된 모든 작업들은 쓰레드 단위로서 트랜잭션을 수행할 때 가장 높은 성능을 발휘하므로, 다수의 사용자의 접근시 효과적인 동시성 제어가 가능하도록 트랜잭션 단위의 관리가 용이해야 하고, 각 트랜잭션들의 동기를 유지할수 있도록 관리되어져야만 한다.

셋째, 멀티미디어 객체를 위한 시스템의 구축시 저장 및 검색의 효율을 최대한 보장할 수 있어야 한다. 기존 SHORE 저장 시스템의 경우 데이타의 물리적인 저장 단위로서 store, extent, page와 같은 개념을 사용하며, 이때 각 store가 담당하는 데이타의 크기에 따라 저장 및 검색 성능이 달라지게 된다. 따라서 삽입하고자 하는 객체의 개별 문서 크기 및 전체 문서 량에 따라 적합한 관리 형태를 찾아내는 것이 중요하다.

넷째, 구현된 객체 관리자 및 역화일 관리자의 효율적

이며 편리한 사용자 접근을 위해 일관된 사용자 API를 제공해야 한다. 이를 위해 기존 저장 시스템의 가장 상위 계층에 확장된 인터페이스를 제공하여 텍스트 및 바이너리 객체들에 대한 접근 방법을 제시해야 하고, 사용자는 객체 저장 엔진의 하부 구조를 고려하지 않고도 접근이 용이하도록 일관되고 편리한 접근 형태를 제공할 수 있도록 설계되어야 한다.

이와 같은 고려사항을 통해, 비정형 멀티미디어 객체를 위한 객체 관리자와 대용량 텍스트를 위한 역화일 관리자를 통합한 객체 저장 엔진을 설계한다. 객체 저장 엔진은 기존의 SHORE 하부 저장 시스템에 객체 관리자와 역화일 관리자를 소결합 방식을 통해 설계하였으며, 설계된 객체 저장 엔진의 구조는 [그림 2]와 같다.

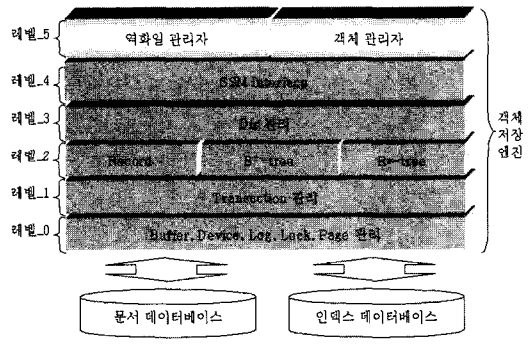

그림 2 설계된 객체 저장 엔진의 구조

3.2 객체 관리자

객체 관리자는 이미지, 오디오, 비디오와 같이 길이에 제한이 없고 매우 가변적인 멀티미디어 객체를 저장 및 검색하기 위한 관리자이다. 이를 위해 cOBJECT_m 이라는 관리자 클래스를 설정하고 기존 ss_m 클래스로부터 상속받아 설계한다. 통합적인 객체 관리자 클래스인 cOBJECT_m 클래스는 [표 1]과 같은 4가지 모듈로 구성되며 데이타베이스, 파일, 오브젝트, 인덱스를 효율적으로 제어하고, 동시에 다수의 사용자 요구를 처리할 수 있는 트랜잭션 기능을 제공한다.

표 1 cOBJECT_m 클래스 모듈

cOBJECT_m 클래스	
데이타베이스 관리 모듈	데이타의 물리적 저장소 관리
파일 관리 모듈	오브젝트들의 클러스터된 집합 관리
오브젝트 관리 모듈	각각의 오브젝트 관리
인덱스 관리 모듈	인덱스 데이타 관리

3.2.1 데이타베이스 관리 모듈

SHORE 저장 시스템은 물리적인 데이타베이스로서 device_m 클래스인 디바이스 관리자를 제공하며, 디바이스 관리자는 사용자 요청에 해당하는 데이타베이스 위치와 크기에 따라 볼륨이라는 논리적인 공간을 생성한다. 이때 각각의 볼륨은 각각 시스템 카달로그 정보인 루트 인덱스를 생성하고 볼륨내의 모든 물리적 변화를 관리하게 된다. 그러나 멀티미디어 객체의 경우 기본적으로 대용량을 차지하고 각각의 객체들에 대한 인덱스 정보가 존재하기 때문에, 하나의 볼륨에서 모든 데이타를 저장하는 것은 많은 자원의 낭비 및 비효율성을 초래한다. 따라서 삽입하고자 하는 멀티미디어 객체들을 형태에 따라 분류하고, 각각의 볼륨을 생성하여 통합적으로 관리할 필요가 있다. 이를 위해 데이타베이스 관리 모듈에서는 각각의 사용자 요구에 맞는 물리적 공간을 생성, 삭제, 개방, 종료할 수 있는 매소드를 설계한다. 데이타베이스 생성의 경우 생성할 데이타베이스 이름과 크기를 이용하여 생성할 데이타베이스를 각 Extent 단위로 분할하고, 새로운 볼륨을 위한 식별자를 생성한 후 시스템 카달로그 정보인 루트 인덱스를 생성한다. 데이타베이스 삭제의 경우 먼저 데이타베이스 이름에 해당하는 볼륨 식별자를 획득후 데이타베이스 내의 논리적 공간을 삭제한다. 그러나 논리적 공간이 삭제되어도 물리적인 공간은 존재하므로 최종적으로 물리적 공간을 삭제하기 위한 작업을 통해 데이타베이스를 삭제한다. 데이타베이스 개방 및 종료의 경우 데이타베이스의 크기는 고려하지 않고 단지 데이타베이스의 이름을 통하여 이미 생성된 데이베이이스의 식별자를 획득 및 반환한다.

3.2.2 파일 관리 모듈

객체 저장 엔진에서 파일은 일반 DBMS의 테이블과 같은 역할을 하며 각각의 객체들에 대한 클러스터링의 기능을 수행한다. 따라서 객체의 형태에 따라서 각각의 파일이 생성되어야 하며, 이때 각 객체의 크기에 상관없이 동일한 성능을 발휘할 수 있도록 설계되어야 한다. 한편 파일은 사용자가 인식하는 논리적인 단위이며, 시스템 측면에서는 store란 물리적 개념으로 변환된다. 이때 store는 [그림 3]과 같이 1 페이지 미만인 작은 객체를 위한 extent와 1 페이지 이상인 대용량 객체를 위한 extent로 이루어지고, 각각의 extent는 인접한 8개의 페이지로서 구성되어진다. 따라서 처음 생성되었을 때 파일의 크기는 128KB 로서 고정되며, 삽입되는 데이타들의 합이 이를 초과했을 경우 새로운 extent를 병합하여 store를 확장하게 된다. 그러나 store가 무한 확장될 경우 데이타의 저장 위치를 찾기 위한 계산시간의 증가

로 시스템의 성능이 저하되게 되므로 이를 위한 방법이 필요하다. 이를 위해 파일 관리 모듈에서는 각각의 사용자 요구에 맞는 물리적 공간을 생성, 삭제, 개방, 종료할 수 있는 방법을 설계한다.

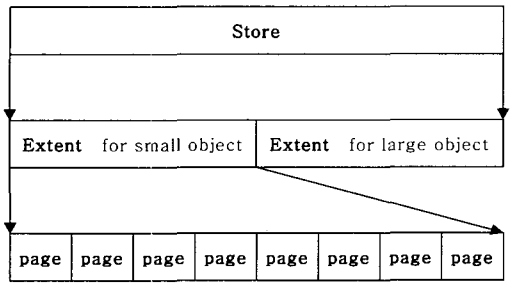

그림 3 파일의 물리적 구조

파일 생성의 경우 생성하고자 하는 데이타베이스 식별자를 통해 생성할 위치를 결정하고, file_p 클래스에서 128Kb 의 공간을 store로 전환하여 해당 store의 첫 번째 페이지를 할당받아 논리적 식별자 관리기인 lid_m 클래스에 전달한다. 한편 파일 관리 모듈의 상위에서 시스템이 어떻게 동작하는지 독립적이기 위해서 생성한 파일의 이름과 식별자를 각각 키와 엘리먼트로 하여 루트 인덱스에 저장한다. 따라서 사용자는 파일 관리 모듈의 동작을 고려하지 않고 단지 파일의 이름만을 가지고 작업을 수행할 수 있다. 파일 삭제의 경우 삭제할 파일의 이름을 가지고 루트 인덱스를 검색하여 실제 물리적 공간인 store를 먼저 삭제하고 논리적 식별자와 함께 루트인덱스에서 파일의 정보를 삭제한다. 파일 개방 및 종료의 경우 파일의 이름을 키로서 변환하여 루트인덱스를 검색하고, 검색 결과인 엘리먼트를 파일 식별자로 변환하여 사용자에게 반환한다.

3.2.3 오브젝트 관리 모듈

오브젝트 관리 모듈에서는 각각의 객체들에 대한 효율적 저장 및 검색 방법을 설계한다. 사용자 관점에서 각 객체에 해당하는 오브젝트는 시스템 관점에서 레코드로서 변환되며, 각각의 레코드는 페이지의 슬롯에 해당한다. 따라서 이러한 페이지내의 각각에 레코드에 대한 효율적인 관리가 필요하다. 이를 위해, 오브젝트 관리 모듈은 [표 2]와 같은 역할을 수행한다.

오브젝트 생성을 위해서는 삽입하고자 하는 객체의 크기를 이용하여 삽입할 객체가 1 페이지 보다 클 경우에는 store의 첫 번째 extent, 1 페이지 보다 작을 경우에는 두 번째 extent에 위치하여 객체가 삽입될 빈 슬

표 2 오브젝트 관리모듈의 기능

오브젝트 관리 모듈	
오브젝트 생성 및 삭제	객체들의 저장 및 삭제
오브젝트 검색	객체들의 검색
오브젝트 수정	객체들의 부분삽입(삭제),갱신,첨가
객체 식별자 검색	객체들의 식별자 전, 후위 검색

롯을 찾는다. 이때 store 내에 모든 페이지를 검색하여 빈 슬롯이 존재할 경우에는 삽입과 함께 생성된 식별자를 반환하고, 빈 슬롯이 존재하지 않을 경우에는 새로운 extent를 할당받아 첫 번째 페이지의 시작위치에 객체를 삽입한다. 오브젝트 삭제의 경우는 생성된 오브젝트 식별자로부터 검색된 슬롯을 삭제하고 해당 페이지의 슬롯정보를 수정한다. 오브젝트의 검색을 위해서는 pin_i 클래스를 이용한다. pin_i 클래스는 오브젝트 식별자를 통해 오브젝트의 크기와 내용을 검색하는 역할을 하며 1 페이지 이상의 경우에는 커서 개념을 적용하여 1페이지 단위로 오브젝트의 내용을 반환한다. 또한 사용자의 경우 전체 오브젝트의 내용이 아닌 부분적인 내용만을 검색하고자 할 경우 검색할 시작위치와 종료위치를 통해서 부분 검색이 가능하다.

저장된 객체의 수정을 위해서는 부분삽입, 부분삭제, 갱신, 첨가, 첨삭 등의 기능이 필요하다. 부분삽입의 경우 삽입할 시작위치와 크기를 이용하여 해당 오브젝트가 저장된 슬롯을 획득하고 시작위치에 데이타를 삽입 후 기존 시작위치 뒤의 데이타를 첨가한다. 부분삭제 및 갱신은 오브젝트의 일정 부분만을 변경하는 것으로서, 시작위치로부터 일정 크기만큼의 데이타를 삭제 또는 수정한다. 첨가 및 첨삭의 경우는 오브젝트의 후위에 새로운 데이타를 삽입하거나, 일정 크기만큼을 제거하는 역할을 한다. 상위 단계에서 하나의 파일내에 클러스터링된 객체들을 일괄적으로 관리하기 위해서는 각각의 객체들에 대한 식별자를 효율적으로 관리하기 위한 방법이 필요하다. 따라서 오브젝트 관리 모듈에서는 커서 개념을 이용하여 파일내의 오브젝트 식별자들을 검색할 수 있는 방법을 설계한다. 이를 위해서 scan_file_i 클래스를 사용하며, 주어진 파일이름을 인덱스의 키로 변환한 후 커서를 초기화하면, 커서는 파일내 페이지들의 첫 번째 슬롯을 가리킨다. 이때 사용자의 요구에 따라 커서를 이동하며 가장 마지막 식별자 또는 이전, 이후의 식별자들을 사용자에게 반환한다.

3.2.4 인덱스 관리 모듈

인덱스 관리 모듈에서는 각각의 객체들에서 추출되는 인덱스 정보에 대한 효율적 저장 및 검색 방법을 설계한

다. 인덱스는 기본적으로 B+-트리를 사용하며, 키의 형태로는 char, int, float 뿐만 아니라 이들을 조합한 형태도 제공하고, 엘리먼트의 길이는 페이지내의 슬롯에 따라서 결정된다. 따라서 이러한 비정형 키와 엘리먼트를 갖는 인덱스를 관리하기 위해 [표 3]과 같은 기능들이 필요하다.

표 3 인덱스 관리모듈의 기능

인덱스 관리 모듈	
인덱스 생성 및 삭제	인덱스의 생성 및 삭제
하나의 데이타 삽입	인덱스에 하나의 데이타를 삽입
벌크 데이타 삽입	인덱스에 벌크로딩을 통한 데이타 삽입
데이타 수정	인덱스내의 엘리먼트 수정
인덱스 순행	인덱스내의 키 순차 검색
인덱스 검색	인덱스를 통한 키 검색

인덱스의 생성 및 삭제를 위해서는 btree_m 클래스를 이용한다. 먼저 생성할 인덱스의 식별자를 lid_m 클래스에서 획득한 후, 인덱스가 생성될 store를 검색하며, 생성할 인덱스에 대한 조건이 없을 경우 기본적으로 하나의 페이지만으로 인덱스를 생성 후 데이타가 1 페이지를 초과할 때 store로 전환을 한다. 그러나 다량의 데이타가 삽입되는 멀티미디어 객체의 경우 1 페이지 만으로 인덱스 정보를 저장할 가능성이 적으므로, 초기 생성의 단계에서 하나의 store로서 공간을 할당받는다. 따라서 페이지에서 store로 전환되는 오버헤드를 줄임으로써 성능을 향상시킬 수 있다. 인덱스에 데이타를 삽입하기 위해서는 모든 데이타들에 대해서 하나씩 인덱스에 삽입하는 방법과, 모든 데이타들을 임시 파일에 담아서 일괄적으로 삽입하는 벌크로딩(bulkloading) 방법이 존재한다. 벌크로딩의 경우 일반 삽입에 비해 트리를 구성하는데 우수한 성능을 나타내지만 인덱스의 생성 후에 단 한번만 수행할 수 있는 단점이 존재한다. 따라서 인덱스 관리 모듈에서는 일반적인 삽입과 벌크로딩을 통한 삽입을 함께 제공한다. 일반 삽입의 경우에는 상위 단계에서 넘겨준 키와 엘리먼트를 vec_t 클래스로 변환하여 인덱스의 노드를 검사하고 삽입될 단말노드를 찾아서 저장한다. 반면에 벌크로딩을 사용한 삽입의 경우에는 삽입하고자 하는 모든 키와 엘리먼트를 임시화일에 순차적으로 저장한후 btree_m 클래스의 벌크로딩 메소드를 호출하면, 일차적으로 임시화일을 정렬한후 단말노드부터 상위노드 순으로 인덱스를 일괄적으로 구축한다.

상위 단계에서 인덱스의 단말노드에 저장된 키들을 일괄적으로 관리하기 위해서는 각각의 키를 효율적으로 관리하기 위한 방법이 필요하다. 따라서 인덱스 관리 모듈에서는 커서 개념을 이용하여 인덱스 내의 키들을 검색할 수 있는 방법을 설계한다. 이를 위해서 scan_index_i 클래스를 사용한다. 그러나 커서를 초기화할 때 키의 형태에 따라서 커서의 형태가 달라지게 되므로 키의 속성 정보를 사용하는게 바람직하다. 따라서 사용자는 키의 형태를 고려하지 않고 인덱스의 시작, 마지막, 이전, 이후의 키들을 검색할 수 있다.

인덱스에 저장된 키와 엘리먼트를 검색하기 위해 정확검색, 우절단 검색, 범위검색을 위한 방법을 설계한다. 이를 위해서 { LE, LT, EQ, GT, GE }와 같은 5 가지의 검색조건을 사용하며, LE 의 경우는 '키보다 작거나 같은', EQ의 경우는 '키와 동일한'의 의미이다. 정확검색을 위해서는 키와 EQ 모드로서 검색된 결과를 반환한다. 우절단 검색은 '정보*'와 같은 질의 형태로서, 주어진 키 뿐만아니라 우절단을 위한 키를 구해야만 한다. 이를 위해서는 질의로 주어진 키인 '정보'를 스트림에 저장한후, 스트림의 가장 후위 코드를 하나 증가하여 두 번째 키로 설정하고 { GE, key1, LT, key2 }와 같은 조건으로 커서를 설정하여 사용자에게 검색 결과를 반환한다. 범위 질의는 사용자가 두 개의 키로서 질의를 수행하는 경우이다. 따라서 범위 질의의 경우에는 { GE, key1, LE, key2 }와 같은 조건을 수행하여 커서를 설정하고, 커서의 움직임에 따라서 검색 결과를 사용자에게 반환한다.

3.2.5 객체 관리자 API

설계된 5가지의 객체 관리 모듈을 위해서 다음과 같은 객체 관리자 API를 설계한다. 객체 관리 API는 ss_m 클래스를 상속받도록 하였으며, 같은 계층의 클래스인 scan_index_i, scan_file_i, pin_i 클래스들을 이용하여 검색이 이루어질 수 있도록 한다. 설계된 객체 관리자 API는 [표 4]와 같다.

표 4 객체 관리자 API

구분	객체 관리자 API
데이터베이스 관리	CreateDB(), DestroyDB(), OpenDB(), CloseDB()
파일 관리	CreateFile(), DestroyFile(), OpenFile(), CloseFile()
오브젝트 관리	CreateObject(), DestroyObject(), GetObjectLen(), ReadObject(), InsertIntoObject(), DeleteFromObject(), AppendToObject(), TruncateObject(), FirstObjectID(), LastObjectID(), NextObjectID(), PrevObjectID()
인덱스 관리	CreateIndex(), DestroyIndex(), OpenIndex(), CloseIndex(), InsertEntity(), DeleteEntity(), BeginLoadEntity(), LoadEntity(), EndLoadEntity(), GetFirstEntity(), GetLastEntity(), GetPrevEntity(), GetNextEntity(), GetExactEntity(), SetEntityRange(), SetEntityRange(), GetEntity(), ReleaseEntityRange()
유틸리티	PrintVolume(), PrintIndex(), BeginXct(), CommitXct()

3.3 역화일 관리자

대량의 텍스트 문서를 효율적으로 저장 및 관리하기 위해 벌크로딩을 응용한 역화일 관리자를 설계한다. 이를 위해 역화일 관리자 클래스인 cINVERT_m 클래스를 설정하고, file_m, btree_m 클래스를 이용하여 설계한다. 또한 확장된 역화일 구조의 사용을 위해 사용자 API를 ss_m 클래스에 확장하고, 확장된 검색을 위해 SetCursor 클래스를 설계한다.

3.3.1 역화일 구조

일반적인 역화일 기법[9]의 경우 B+tree와 포스팅 파일로 구성되어 지며, 각각의 키 값마다 포스팅 레코드를 생성 또는 추가하는 작업이 이루어지기 때문에 수많은 I/O가 발생할 뿐만 아니라, 부가저장 공간의 측면에서도 상당한 성능의 감소를 가져온다. 따라서 하나의 키 값에 해당하는 포스팅 레코드의 내용들을 벌크로딩을 통해 버퍼관리자에 담아두고 해당 키의 작업이 끝났을 경우 일괄적으로 저장 함으로써 성능을 향상시킬 수 있다. 설계된 역화일의 구조는 [그림 4]와 같다. 여기서 PSW는 각각 문단(Paragraph), 문장(Sentence), 단어(Word) 번호를 나타낸다. Rlen는 예약필드 길이이며 예약필드는 응용에 따라 미리 할당된 필드이다.

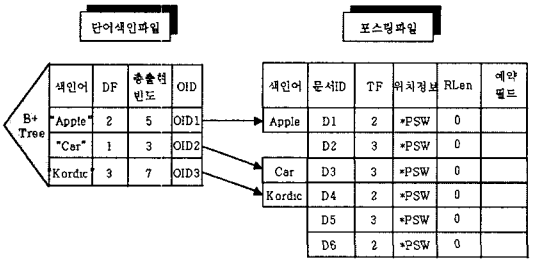

그림 4 역화일 구조

단어 색인 파일을 위해서는 SHORE에서 제공하는 btree_m 클래스를 이용한다. 일반적으로 역화일의 경우 단어 색인 파일내에 중복되는 키를 허용하지 않으므로 't_uni_btree' 모드를 사용하여 비중복 키들만을 사용할 수 있도록 하며, 단어 색인 파일의 키로서는 색인어가 삽입되고 엘리먼트로서는 문서 출현 빈도, 전체 키 출현 빈도, 포스팅 레코드 식별자를 조합하여 구성한다. 또한 단어 색인파일의 삽입은 포스팅 파일의 벌크로딩이 끝난 후 인덱스의 벌크로딩을 통해 이루어지며, 이때 인덱스에 EXCLUSIVE 잠금을 설정하여 벌크로딩 수행중에는 다른 사용자의 접근을 배제하도록 한다.

한편 포스팅 파일의 경우는 일반적인 역화일의 삽입 기법을 개선하여 설계한다. 하나의 단어 색인 파일의 키에 해당하는 포스팅 레코드의 문서 정보는 수백건에서 수만건까지 많은 양의 정보를 지니기 때문에 일반적인 역화일 삽입 방식을 이용했을 경우 해당 문서 건수 만큼의 I/O가 일어나게 된다. 따라서 하나의 색인어에 대한 포스팅 레코드를 버퍼에 담아두고 모든 문서정보의 삽입이 종료한후 일괄적으로 삽입을 수행하면 단 몇 번의 I/O 만으로 작업을 수행하여 I/O 수를 최소화하고 우수한 성능을 얻을 수 있다. [그림 5]는 벌크로딩을 사용한 역화일 삽입 방법을 보이고 있다.

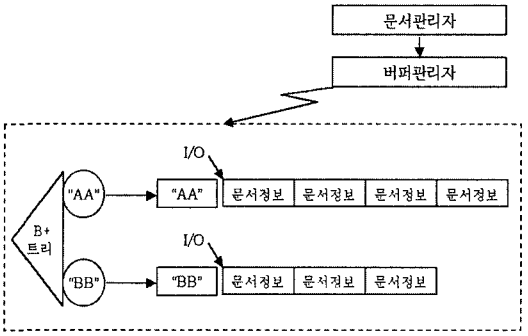

그림 5 벌크로딩을 사용한 역화일 삽입 방법

아울러 역화일 구조에 대한 동시성을 유지하기 위해, 버퍼에 담긴 문서 정보가 포스팅 파일에 저장될 때는 EXCLUSIVE 잠금을 설정하여 다른 사용자의 접근을 배제한다. 또한 벌크로딩을 위한 문서정보 파일의 정렬을 위해 SHORE에서 제공하는 sorted_stream을 이용하는 방법과 새롭게 확장된 cUNIXSORT를 사용하는 두가지의 정렬 기법을 함께 사용한다. sorted_stream은 SHORE에서 제공하는 정렬 기법으로서 정렬하고자 하는 모든 데이타를 메모리와 임시파일에서 수행하는 방법이다. 그러나 sorted_stream은 많은 메모리의 낭비로 인해 다량의 데이타에는 적합하지 않은 방법이다. 따라서 퀵(Quick)정렬과 머지(Merge)정렬을 혼합한 cUNIX SORT클래스를 함께 제공한다.

3.3.2 역화일 관리자 모듈

역화일 관리 모듈에서는 원 문서와 문서로부터 추출된 키워드 및 문서정보들을 효율적으로 관리하기 위한 방법을 설계한다. 역화일 관리 모듈에서는 삽입으로서 일반적인 방법을 사용하는 경우와 벌크로딩(bulkloading)을 이용한 삽입 방법을 제공하고, 문서의 검색을 위해서는 커서를 이용한 확장 검색을 지원한다. 이러한 역화일 관리자 모듈의 기능은 [표 5]와 같다.

표 5 역화일 관리자 모듈

역화일 관리자 모듈	
역화일 생성 및 삭제	단어색인 파일 및 포스팅 화일 생성 및 삭제
문서 삽입	일반적입 삽입 및 벌크로딩을 통한 삽입
문서 검색	커서를 이용한 확장검색

역화일 구조의 생성를 위해서는 btree_m, file_m 클래스를 이용한다. 역화일 구조는 B+트리와 포스팅 파일로 이루어지며, SHORE에서 제공하는 serial_t 타입의 식별자를 이용한다. 따라서 B+트리 식별자와 포스팅 파일 식별자를 혼합하여 역화일 구조 식별자로 활용한다. 역화일 인덱스의 생성을 위해서는 B+트리와 포스팅 파일을 위한 store 2개를 할당받고 io_m 클래스에서 생성한후에 하나는 B+트리를 위한 공간으로 또하나는 포스팅 파일을 위한 공간으로 설정한후 생성된 식별자들을 lid_m 클래스에 등록한다. 또한 생성된 물리적 정보는 dir_m 클래스를 통하여 논리적 식별자들과의 연결을 유지한다. 한편 생성한 역화일 구조를 삭제하기 위해 역화일 식별자를 통해 B+트리와 포스팅 파일의 식별자를 획득한후 btree_m, file_m 클래스를 통해 각각의 store 정보를 삭제한다.

생성된 역화일을 통한 문서의 삽입을 위해 일반적인 문서 삽입 방법과 벌크로딩을 이용한 문서 삽입 방법을 설계한다. 일반적인 문서 삽입 방법의 경우, 원문서를 삽입후 생성된 문서 식별자 정보와 문서에서 추출된 정보를 이용하여 B+트리에 색인어가 존재하는지 확인한다. 색인어가 존재하는 경우에는 기존의 포스팅 레코드에 문서정보를 삽입하고 B+트리의 엘리먼트 정보를 수정한다. 색인어가 존재하지 않는 경우에는 새롭게 포스팅 레코드를 할당받아 이때 생성된 레코드 식별자와 함께 B+트리에 삽입한다. 아울러 벌크로딩을 이용한 문서 삽입의 경우 모든 원 문서의 삽입이 끝난후에 벌크로딩을 통하여 역화일이 구성한다. 먼저 모든 원 문서를 삽입하고 이때 각각의 문서 식별자와 문서에서 추출된 정보를 cUNIXSORT 클래스를 통하여 색인어 순으로 정렬하고, 정렬된 색인어 리스트에서 동일한 색인어의 문서 추출결과만을 버퍼에 저장한다. 하나의 색인어 정보가 버퍼에 모두 담겨지면 버퍼의 내용을 포스팅 레코드를 통해 삽입하고, 모든 포스팅 파일의 구성이 끝나게되면 B+트리의 키와 엘리먼트 정보들을 btree_m 클래스의 벌크로딩을 통해 수행한다. 벌크로딩은 역파일 인덱스 초기 구축 시에 한번 사용되며, 벌크로딩 후에 하나의 문서의 삽입은 일반적인 문서 삽입 방법을 통해 문서의 추가가 가능하다.

벌크로딩 후에 하나의 문서의 삭제를 위해서는 포스팅 파일의 순차탐색을 통한 삭제를 수행한다. 이를 위해, 포스팅 파일의 모든 레코드를 순회하며 포스팅 레코드에 저장된 문서식별자와 삭제할 문서의 식별자가 동일한 경우 해당 포스팅 레코드에 'INVALID' 표시를 한다. 'INVALID'는 물리적인 공간의 삭제가 아니라 논리적으로 레코드가 삭제되었음을 명시하는 값으로 역파일 인덱스 구조의 빈번한 변경이 발생하여 포스팅 파일의 압축(Compaction)을 수행하게 될 때 물리적인 삭제가 이루어진다. 문서의 삭제를 위해 순차탐색을 수행하는 이유는 포스팅 파일 내에서 삭제할 문서의 정보가 저장된 레코드의 위치를 알 수 없기 때문이다. 문서의 식별자와 색인어를 저장하기 위한 별도의 부가공간을 생성하여 효율적인 문서 삭제를 지원할 수 있으나, 이는 문서의 삽입 시마다 삭제를 위한 부가적인 비용이 너무 큰 단점이 있다.

벌크로딩 후에 하나의 문서의 갱신을 위해서는 보다 많은 고려사항이 필요하다. 예를 들어 하나의 문서에 대해 저장된 색인어의 수가 10개이고, 갱신된 문서의 색인어 수가 6개라면 나머지 4개의 색인어에 대해서는 문서 삭제의 경우처럼 포스팅 레코드를 순차 검색하여 해당되는 레코드에 'INVALID' 표시를 해야 한다. 또한 갱신된 문서의 6개의 색인어에 대해 문서 내 출현빈도가 달라지는 경우, 포스팅 레코드의 길이가 확장(또는 축소)되며, 이로 인한 포스팅 파일 내의 구조가 변경되어야 한다. 이를 위해, 기존 위치에 'INVALID' 표시를 한 후 해당 포스팅 파일의 뒤에 새롭게 추가하는 방법이 고려될 수 있다. 본 논문에서는 문서의 갱신을 위해서 새로운 알고리즘을 제공하는 대신, 저장된 기존의 문서를 삭제한 후 수정된 문서를 재삽입 하는 방법을 사용한다.

역화일을 이용한 확장 검색을 위해서는 정확질의, 우절단, 범위질의를 제공하며, 세가지의 형태에 대해서 SetCursor() API를 오버라이딩에서 사용한다. 각각의 SetCursor()는 주어진 조건에 대해서 scan_index_i, pin_i 클래스를 통해 커서를 초기화하고, GetDocument() API를 통해 사용자에게 결과를 반환한다.

3.3.3 역화일 관리자 API

설계된 역화일 관리 모듈을 위해서 다음과 같은 역화일 관리자 API를 설계한다. 역화일 관리자 API는 file_m, btree_m, scan_index_i, scan_file_i, pin_i 클래스들을 이용하여 저장 및 검색이 이루어질 수 있도록 한다. 설계된 역화일 관리자 API는 [표 6]과 같다.

표 6 역화일 관리자 API

구분	역화일 관리자 API
인덱스 생성, 삭제	CreateInvtIndex(), DestroyInvtIndex()
문서삽입	InsertDocInvt(), InsertBulkInvt()
문서갱신 및 삭제	UpdateDocInvt(), DeleteBulkInvt()
문서검색	SetCursor(), NextCursor(), GetDocument()

4. 구현 및 성능 고찰

객체 저장 엔진은 가변적인 크기를 가지는 비정형 객체들을 효율적으로 저장 및 검색할 수 있도록 설계 되었으며, 객체 저장 엔진의 구현은 기존의 하부저장 시스템인 SHORE ver2.0을 사용하고, 구현환경은 Intel Pentium-III 750 Dual CPU 상에서 Linux 6.1을 운형체제로 하여 구현하였다. 컴파일러는 G++2.7.2.3 및 Make ver 3.74를 사용하였고, SHORE 저장 시스템을 위한 내부 파싱을 위해 Perl ver5.0, Flex ver2.5.3, Bison ver1.25를 사용하였다.

4.1 성능평가 고려사항

본 논문에서 제시하는 객체 저장 엔진은 비정형 문서들을 저장하기 위한 논리적 단위로서 store를 사용한다. 하나의 store는 기본적으로 두 개의 extent로 구성되어지며 하나의 extent는 8개의 페이지로 이루어진다. 이때 첫 번째 extent는 1 페이지 미만의 작은 객체를 위한 공간이며, 두 번째 extent는 1페이지 이상의 큰 객체를 위한 공간으로 예약된다. 각각의 extent는 64KB 크기로서 새로운 객체를 위한 공간이 없을 경우 빈 extent를 병합함으로써 store를 확장하여 사용하고 이론적으로 4GB까지 확장이 가능하다. 그러나 하나의 store를 계속해서 확장하여 사용하는 경우 새로운 extent를 할당받아 데이타를 삽입하는 과정에서 많은 계산시간으로 인해 성능이 감소하게 되고, 각각의 삽입연산에 대해서 개별적인 store를 생성하여 사용할 경우 이미 예약되어진 두 개의 extent에 저장되지 않은 빈 공간들이 무수히 많아지기 때문에 저장공간의 낭비를 초래한다. 따라서 대량의 문건에 대해 효율적인 저장 및 검색을 위해서는 삽입성능과 부가저장 공간의 두 가지 측면을 모두 만족시킬 수 있는 방법이 필요하다. 이를 위해 본 논문에서는 다음 [표 7]과 같은 5 가지의 평가 방법을 이용하여 성능 평가를 수행한다.

<방법 1>은 하나의 store에 문서가 삽입될 공간이 없을 경우 새로운 store를 생성하여 다음 문서를 삽입하는 경우이며 하나의 store내에 포함되는 평균 삽입 문서수는 32개이고, <방법 5>는 하나의 store에 문서가 삽입될 공간이 없을 경우 새로운 store를 생성하지 않고 비어있는 extent를 병합하여 문서를 삽입하다가 store내의 extent수가 21개가 되면 새로운 store를 생성하는 방법이다.

표 7 객체 저장 엔진의 성능 평가 방법

평가방법	store당 extent수	store당 포함 평균 문서수
방법 1	2	30
방법 2	6	160
방법 3	11	320
방법 4	16	480
방법 5	21	640

4.2 객체 관리자의 성능평가

객체 관리자의 성능 평가를 수행하기 위해서 1KB ~ 8KB의 크기를 가지는 국외논문초록 Small 객체 100만건과 10KB ~ 30KB의 크기를 가지는 랜덤하게 생성한 Large 객체 10만건을 이용하였으며, [표 8]은 성능평가를 위한 실험인자를 보여준다.

표 8 객체 관리자의 성능평가를 위한 실험인자

평균 문서 크기	1KB ~ 30KB의 가변적인 비정형 객체
전체 문서 수	Small 객체 100 만건, Large 객체 10만건
데이타 종류	국외논문초록, 랜덤하게 생성
비교 지수	삽입시간, 부가저장 공간

또한 Linux 운영체제의 경우 생성할 수 있는 최대 데이타베이스 크기가 2GB를 초과할 수 없으므로, 2개의 비정형 객체 데이타베이스를 생성하여 성능 평가를 수행하였다. 성능평가를 위해 객체 저장 엔진의 물리적 페이지 크기는 8KB로 설정하였고, 이에 대한 객체 관리자의 성능은 [표 9]와 같다.

Small 객체의 삽입 시간의 경우 <방법 3>이 437초로서 우수한 성능을 보이고 <방법 1>, <방법 5>는 각각 785초, 451초로서 많은 시간이 소요됨을 나타낸다. <방법 1>의 경우는 소량의 문서마다 새로운 store를

표 9 객체 관리자 성능평가

문서	평가방법	실험 방법				
		방법1	방법 2	방법 3	방법 4	방법 5
Small 객체	전체객체삽입	785초	458초	437초	443초	451초
	문서당객체삽입	0.0007초	0.0004초	0.0004초	0.0004초	0.0004초
	부가저장공간	198 %	125 %	119 %	115 %	112 %
Large 객체	전체객체삽입	1080초	680 초	513 초	590 초	664 초
	문서당객체삽입	0.0108초	0.0068초	0.0051초	0.0059초	0.0066초
	부가저장공간	168 %	120 %	117 %	115 %	114 %

빈번히 생성하기 때문이며, <방법 5>는 20개의 extent에서 객체를 삽입할 공간을 찾는 계산시간의 증가로 성능이 감소된다. 한편, 부가저장 공간의 측면에서는 <방법 5>가 112%로서 가장 우수하다. 이는 매번 새로운 store를 생성할 때 발생하는 빈공간의 낭비가 최소화되기 때문이다. 한편 Large 객체의 경우에도 삽입 시간의 경우 <방법 3>이 513초로서 가장 우수한 성능을 보이며, 부가 저장 공간의 경우 <방법 5>가 114%로서 가장 우수한 성능을 보인다. 따라서 삽입시간과 부가저장 공간의 효율성을 동시에 만족시키는 방법을 찾는 식은 다음 (식 1)과 같다.

$$Ei = WI * \frac{1}{Ii/Imin} + WS * \frac{1}{Si/Smin} \quad (식1)$$

Ei = 방법 i 의 객체관리자 효율성
WI = 삽입시간의 가중치
Ii = 방법 i 에 대한 삽입시간
Imin = MIN(I1, I2, I3, I4, I5)
WS = 부가저장공간의 가중치
Si = 방법 i 에 대한 부가저장공간
Smin = MIN(S1, S2, S3, S4, S5)

먼저 삽입 시간과 부가저장 공간의 수치를 0에서 1까지 정규화하여 각각의 실험에 적용한다. 삽입 시간만을 고려할 경우에는 <방법 3>의 경우가 가장 우수한 성능을 보이는 반면 부가 저장 측면에서 비효율적이고, 부가저장 공간만을 고려할 경우에는 <방법 5>의 경우가 가장 우수한 성능을 보이는 반면 삽입 시간 측면에서 비효율적이다. 따라서 삽입 시간과 부가저장 공간을 동일한 가중치로 고려할 경우(WI=WS=0.5), <방법 3>의 경우가 가장 우수한 성능을 나타낸다. [그림 6]은 WI와 WS를 각각 0.5의 가중치로 하였을 때의 결과이다.

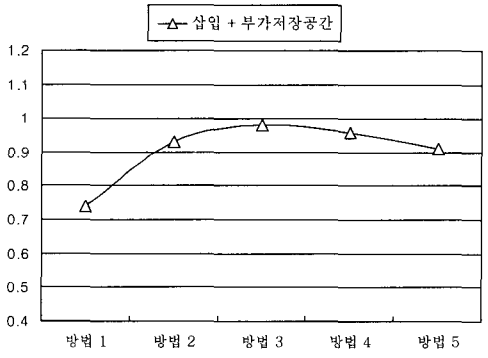

그림 6 가중치에 따른 객체 관리자 성능비교

4.3 역화일 관리자의 성능평가

역화일 관리자의 성능 평가를 수행하기 위해 [표 10]과 같이 평균 1KB ~ 3KB 의 크기를 가지는 100 만건의 텍스트 객체를 이용하였으며, 이를 위한 저장 및 검색 기법으로 역화일기법을 이용하였다.

표 10 역화일 관리자의 성능평가에 대한 실험인자

평균 문서 크기	1KB ~ 3KB의 가변적인 텍스트 문서
전체 문서 수	100 만건
데이타 종류	국외 학술대회 논문 초록
검색 방법	키워드 빈도가 1000 이상인 문서 검색
비교 지수	삽입시간, 검색시간, 부가저장 공간

역화일 관리자를 위한 성능평가의 경우 삽입시간 및 부가저장 공간의 측정을 위해서 [표 7]에서 제시된 방법들을 적용하였다. 검색방법으로는 키워드 빈도가 1000 이상인 문서를 검색하는데 소요되는 시간을 측정하였으며, 이에 대한 역화일 관리자의 성능은 [표 11]과 같다.

표 11 역화일 관리자 성능평가

문서	평가방법	실험 방법				
		방법1	방법 2	방법 3	방법 4	방법 5
텍스트 문서	원문서삽입	785초	458초	437초	443초	451초
	전체역화일구성	1482초	1480초	1481초	1482초	1485초
	문서당역화일구성	0.0014초	0.0014초	0.0014초	0.0014초	0.0014초
	부가저장공간	242 %	145 %	133 %	129 %	127 %
	문서검색	0.004초	0.004초	0.004초	0.004초	0.004초

검색시간 측면에서는 하나의 문서 검색시 0.004초로 동일한 결과를 보이고 있다. 한편 삽입시간 측면에서는, 전체 역화일을 구성하는 시간은 약 1480초로 유사한 반면, 원 문서를 삽입하는 시간은 <방법 3>이 437초로서 가장 우수한 성능을 보이고 <방법 1>, <방법 5>는 각각 785초, 451초가 소요됨을 나타낸다. 한편 부가저장 공간 측면에서는 <방법 5>가 127%로서 가장 우수하다. 따라서 역화일 관리자의 성능평가를 위해 삽입시간, 부가저장 공간, 검색시간 모두의 효율성을 동시에 만족시키는 다음 (식 2)가 필요하다.

$$Ei = WI * \frac{1}{Ii/Imin} + WR * \frac{1}{Ri/Rmin} + WS * \frac{1}{Si/Smin} \quad (식2)$$

Ei = 방법i의 역화일관리자 효율성
WI = 삽입시간의 가중치
Ii = 방법 i 에 대한 삽입시간
Imin = MIN(I1, I2, I3, I4, I5)
WR = 검색시간의 가중치
Ri = 방법 i 에 대한 검색시간
Rmin = MIN(R1, R2, R3, R4, R5)
WS = 부가저장공간의 가중치
Si = 방법 i 에 대한 부가저장공간
Smin = MIN(S1, S2, S3, S4, S5)

먼저 삽입 시간과 부가저장 공간 및 검색 시간의 수치를 0에서 1까지 정규화하여 각각의 실험에 적용한다. 삽입 시간만을 고려할 경우에는 <방법 3>의 경우가 가장 우수한 성능을 보이는 반면, 부가저장 공간만을 고려할 경우에는 <방법 5>의 경우가 가장 우수한 성능을 나타낸다. 한편 문서 검색 시간의 경우에는 모두 동일한 성능을 보인다. 따라서 삽입 시간, 부가 저장 공간, 검색

시간 모두를 동일한 가중치로 고려할 경우(WI=WS=WR=0.33), <방법 3>의 경우가 가장 우수한 성능을 나타낸다. [그림 7]은 WI, WS, WR을 각각 0.33의 가중치로 하였을 때의 결과이다.

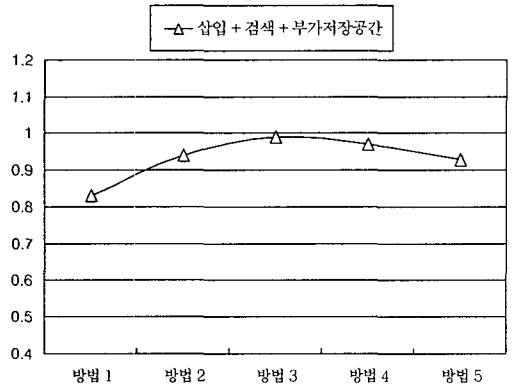

그림 7 가중치에 따른 역화일 관리자 성능비교

한편, 본 논문에서 제시하는 역화일 관리자는 기존의 SHORE 하부 저장 시스템에 소결합 되었으며, 문서 관리의 기법으로 벌크로딩을 사용한다. 따라서 확장된 SHORE 하부 저장 시스템에서 제시하는 밀결합된 역화일 관리자와[6], 본 논문에서 제시하는 소결합된 역화일 관리자의 일반적인 문서 삽입 성능과, 아울러 벌크로딩을 사용했을 경우의 문서 삽입 성능을 [그림 8]에 제시한다. 일반적인 방법을 사용하는 경우는 문서로부터 추출된 각 키워드마다 역화일을 구성하는 방식이며, 벌크로딩 방법을 사용하는 경우는 문서로부터 추출된 키워드 및 문서정보를 정렬한 후 버퍼관리자를 통해 일괄적으로 구성하는 방법이다. 실험을 위한 인자로는 표 8과 같이 100만건의 국외 논문 초록을 이용하였으며, 삽입시간, 부가저장 공간, 검색시간의 가중치를 각각 0.33으로 동일하게 설정하였을 때 가장 우수한 성능을 보이는 <방법 3>을 이용하여 수행하였다.

확장된 SHORE 저장 시스템에서 제공하는 밀결합된 역화일의 문서 삽입은 약 0.0025 초, 본 논문에서 제안하는 소결합된 역화일의 일반삽입은 약 0.0024초로서 유사한 성능을 보인다. 반면 벌크로딩을 이용한 방법의 경우 약 0.0002초로서, 약 13배의 성능 향상을 보인다. 이는 일반삽입의 경우 각각의 색인어에 대해서 포스팅 파일을 검색하는 I/O와 단어 색인 파일을 검색하는 I/O가 발생하는 반면, 벌크로딩의 경우 버퍼 관리자를 통한 일괄 작업으로 I/O 횟수가 현저하게 감소하기 때문이다.

한편 벌크로딩을 한 후 하나의 문서의 삽입은 일반 문서 삽입과 같은 시간(약 0.0025 초)이 소요된다. 아울러 본 논문에서는 삭제 문서에 대한 정보를 특별히 관리하지 않기 때문에,벌크로딩을 한 후 하나의 문서의 삭제는 역화일 구조의 포스팅 파일을 순차 탐색하는 시간만큼이 소요된다. 아울러 벌크로딩을 한 후 하나의 문서의 갱신은 본 논문에서 제시하는 방법이 문서 삭제 후 재삽입 방법을 사용하므로, 하나의 문서의 삽입과 삭제를 합산한 시간이 요구된다.

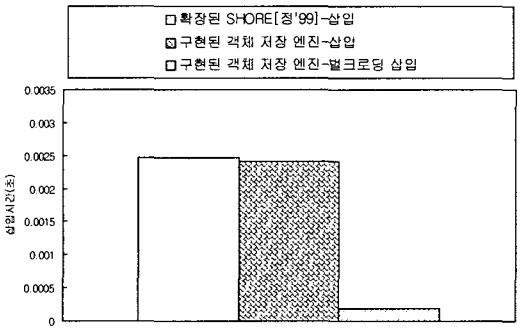

그림 8 확장된 SHORE vs 객체 저장 엔진

5. 객체 저장 엔진을 이용한 TIROS 구축

본 장에서는 구현된 객체 저장 엔진의 유용성을 검증하기위해 논문 정보검색 시스템인 TIROS(Thesis Information retrieval using Object Storage engine)를 본 논문에서 제시한 객체 저장 엔진을 사용하여 구축한다. 이를 위해, *Apache Web Server*를 이용하였고, 웹과 객체 저장엔진의 통신을 위해서 CGI를 사용하였다.

5.1 문서의 구성 및 데이타베이스 구조

구현된 객체 저장 엔진을 사용하여 구축된 논문 정보검색 시스템인 TIROS를 위하여 200 만건의 국외 논문 초록을 추출하여 각각 6개의 세션으로 분류한다. 각각의 세션들은 <저자명>, <키워드>, <발행년도>, <원문수록처>, <주제>, <본문>으로 구성되며, 문서처리기에서 입력받아 각각의 세션에 따른 데이타베이스가 구성된다.

한편 정보검색을 위해서 하나의 문서 데이타베이스와 하나의 데이타베이스가 존재한다. 문서 데이타베이스에는 원문서가 저장되고, 이때 생성된 문서 식별자를 이용하여 각 세션에서 파싱된 결과를 인덱스 데이타베이스에 저장한다. 그러나 Linux 운영체체의 경우 생성할 수 있는 최대 파일 크기가 2GB를 초과할 수 없으므로, 50만건마다 각각의 Store를 생성한다. 인덱스 데이타베이

스에는 문서처리기에서 파싱된 결과를 저장하며 문서의 색인 및 검색을 위해 역화일 관리자를 사용하고, 각 세션에 대한 Store를 생성하여 효율적인 관리가 이루어질 수 있도록 한다. 이에대한 TIROS 시스템의 저장 및 검색 구조는 [그림 9]와 같다.

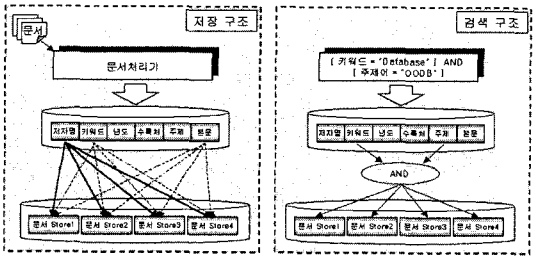

그림 9 각 세션별 저장 및 검색 구조

5.2 질의 유형 및 사용자 인터페이스

사용자 인터페이스에서 처리할 수 있는 질의의 유형을 분류하면 단순질의, 복합질의로 나눌수 있으며, 단순질의에는 각 세션에 따른 질의형태로서 특성 정보중 한 가지만을 사용하는 질의로서 <저자명>, <키워드>, <발행년도>, <수록처>, <주제>, <본문>에 대한 정확질의 및 확장검색을 위한 우절단 질의를 제공한다. 또한 복합질의의 경우는 불리언 질의를 이용한 다수 세션이 결합된 형태이며, {<저자명> AND <수록처>}와 같다.

TIROS 시스템의 질의 입력 부분은 <저자><제목><본문><키워드><원문수록처><발행일자> 각각에 대한 질의 중 필요한 정보를 입력받으며, 다수의 질의에 대해서 AND, OR 연산을 통한 복합질의를 제공한다. 질의에 대한 세부 형태로는 정확 질의와 함께 확장된

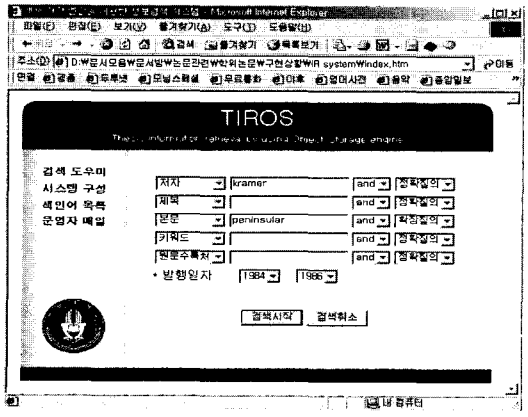

그림 10 TIROS 시스템 질의 인터페이스

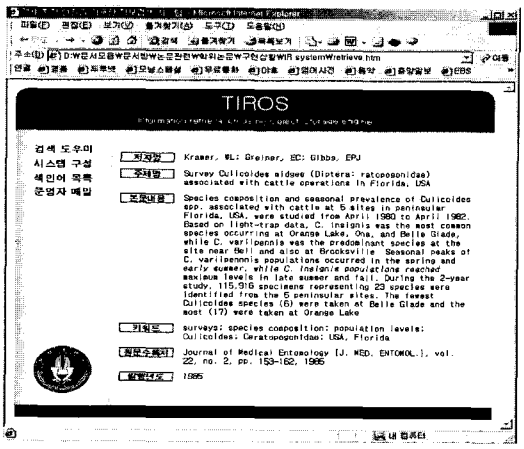

그림 11 TIROS 시스템 질의 결과 인터페이스

질의인 우절단 질의를 제공하고, 색인어 목록을 지원하여 검색의 편리성을 보장하도록 한다.

[그림 10]은 사용자로부터 질의를 처리하는 예제이다. 저자명은 ('Kramer', '정확질의')를, 본문내용은 ('peninsular', '확장질의')를 포함한 문서중 발행일자가 1984년부터 1986년까지인 문서를 검색하도록 하였으며, 이 질의에 대한 최종 처리 결과는 [그림 11]과 같다.

6. 결 론

본 논문에서는 텍스트, 이미지, 오디오, 비디오와 같은 대용량의 멀티미디어 객체를 효율적으로 저장하고 관리하기 위해서 비정형 멀티미디어 객체를 위한 객체 관리자와 대용량 텍스트를 위한 역화일 관리자를 통합한 객체 저장 엔진을 설계 및 구현하였다. 객체 저장 엔진은 기존의 SHORE 하부저장 시스템에 객체 관리자와 역화일 관리자를 소결합하여 구현하였으며, 따라서 다양한 종류의 멀티미디어 객체들에 대해 유연성있는 접근 기법을 제공한다.

객체 저장 관리자는 이미지, 오디오, 비디오와 같은 가변적 길이의 비정형 멀티미디어 객체를 효율적으로 관리하기 위해서 데이타베이스, 파일, 오브젝트, 인덱스 모듈로 분류하여 각각의 기능을 독립적으로 수행하도록 하였으며, cOBJECT_m 클래스를 이용하여 통합 관리할 수 있도록 구현하였다. 데이타베이스 관리 및 파일 관리 모듈의 경우 객체들의 물리적인 저장소에 대한 효율적인 관리가 이루어지도록 하였으며, 각각의 개별적인 객체들을 관리하기 위한 오브젝트 관리 모듈의 경우 생성, 삭제, 부분수정, 객체 식별자 순차 탐색 등의 기능을 지원한다. 인덱스 관리모듈은 객체로부터 파싱된 정보를

색인하기 위하여 생성, 삭제, 삽입, 벌크로딩을 통한 삽입, 엘리먼트 수정, 키 순차 탐색등의 기능을 지원한다.

역화일 관리자는 대용량 텍스트에 대한 효율적 관리를 위하여 벌크로딩을 이용하여 구현하였다. 이를 위해 역화일 관리자를 위한 cINVERT_m 클래스와 원 문서로부터 파싱된 결과를 정렬하기 위한 cUNIXSORT 클래스를 설계 및 구현하였다. 역화일 관리자는 하나의 키 값에 해당하는 포스팅 레코드의 내용들을 벌크로딩을 통해 버퍼 관리자에 담아두고 해당 키의 작업이 끝났을 경우 일괄적으로 저장함으로써 성능을 향상시킬 수 있도록 하였다. 한편 역화일 관리자를 통한 확장된 검색을 위해 SetCursor 클래스를 설계하여 정확 질의, 우절단 질의, 범위 질의를 지원한다.

구현된 객체 저장 엔진은 다수의 사용자에 대한 접근을 효율적으로 관리 위해 모든 관리 모듈들에 대하여 잠금과 트랜잭션을 통한 동시성을 보장할 수 있도록 하였으며, 시스템 손상을 위한 회복 기능을 제공하였다. 이를 통해, 일반 텍스트 및 비정형 멀티미디어 객체에 대한 효율적인 관리를 수행할 수 있으며, 이러한 객체 저장 엔진의 유용성을 검증하기 위해 Web과의 연동을 통한 논문 정보검색 시스템인 TIROS를 구축하였다.

향후 과제로는 본 논문에서 설계 및 구현된 객체 저장 엔진을 기반으로 다양한 멀티미디어 응용분야에 적합한 멀티미디어 정보검색 시스템을 구축하는 것이다.

참 고 문 헌

[1] 황규영, "객체지향 멀티미디어 DBMS-오디세우스 개발사례", 동계 데이타베이스 학술대회 튜토리얼, pp. 131-156, 1998.

[2] M.J. Carey, et al. "Shoring Up Persistent Applications," Proc. of ACM SIGMOD, Vol23, No.2, pp.383-394, 1994.

[3] H-T Chou, et al, "Design and Implementation of the Wisconsin Storage System," Software Practice and Experience, Vol. 15, No. 10, 1985.

[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.

[6] 정재욱, 장재우, "멀티미디어 응용을 위한 SHORE 하부저장 시스템의 확장", 추계정보과학회 학술대회, 제 26-2호, pp6-8, 1999.

[7] Berchtold S., Keim D., Kriegel H.-P., "The X-tree: An Index Structure for High-Dimensional Data," 22nd Conf. on Very Large Databases, 1996, Bombay, India.

[8] Arya S., Mount D.M., Narayan O., "Accounting for Boundary Effects in Nearest Neighbor Searching," Proc. 11th Annual Symp. on Computational Geometry, Vancouver, Canada, pp. 336-344, 1995

[9] C. Faloutsos, "Access Methods for Text," ACM Computing Surveys, Vol17, No.1, 1985.

진 기 성

1999년 전북대학교 컴퓨터공학과(공학사). 2001년 전북대학교 컴퓨터공학과(공학석사). 2001년 ~ 현재 한국전자통신연구원 연구원. 관심분야는 멀티미디어 데이타베이스, 멀티미디어 정보검색, 하부저장구조, 클러스터 데이터베이스

장 재 우

1984년 서울대학교 전자계산기공학과(공학사). 1986년 한국과학기술원 전산학과(공학석사). 1991년 한국과학기술원 전산학과(공학박사). 1996년 ~ 1997년 Univ. of Minnesota, Visiting Scholar. 1991년 ~ 현재 전북대학교 전자정보공학부 정교수. 관심분야는 멀티미디어 데이타베이스, 클러스터 데이타베이스, 하부저장구조