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 애플리케이션 객체들을 위한 객체 관리자 및 대용량 텍스트를 위한 항목관리 기반을 설계하기 위한 문제를 문헌고찰을 수행하였다. 이연구, 기존의 하부저장 구조인 SHORE 저장 시스템에 통합하여 DBMS 측면에서 제공하는 동시성 제어, 희귀기법 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 이는 다양한 환경에서 사용자들은 다양한 형태의 멀티미디어 서비스를 요구하는 추세이다. 현재 데이터베이스 시스템들은 텍스트 또는 수치정보 서비스에 적합한 구조로 판계형 DBMS 또는 파일시스템을 기반으로 데이터베이스를 관리하고 있으나, 멀티미디어 용량을 위해서는 기존 판계형 DBMS를 확장하여 멀티미디어 자료를 다루는 초보적인 단계로도 많은 관심을 드러내고 있다.

이를 위하여 본 논문에서는 다양한 객체들을 효율적으로 저장 및 검색하기 위한 구조를 분석하고, 미국 워렌스 대학에서 개발한 SHORE 하부저장 시스템을 확장한다. SHORE에서 사용되는 대용량 텍스트, 이미지, 비디오 등 매우 큰 비정형 객체를 위한 저장 방법으로 4GB까지 확장 가능한 가변길이 레코드를 제공하며, 이미지 쓰기 데이터를 기반으로 한 페이지를 향한다. 그러나 멀티미디어 객체의 경우 크기가 매우, 가변적이기 때문에 단순한 삽입연산을 통해 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 객체들을 효율적으로 저장 및 검색할 수 있는 객체 관리자 설계하고, 이를 위한 사용자 인터페이스로서 객체관리 시스템을 구현한다. 이는 메모리, 텍스트 검색을 위한 센터기로는 볼크로딩을 응용한 역할의 기함[1]을 SHORE level2에 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 연구계획을 제시한다.

2. 관련연구


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 관리하기 위해 기존의 SSM API를
개선된 객체 관리자 설계 및 구현하고, 테스트 검색을 위해
역할임 기법을 SHORE Level 2에 확장한다. 설계된 객체 관리자와 역할임 관리자를 통합한 SHORE 저장시스템의 전체적
규모는 그림 1과 같다.

그림 1 확장된 SHORE 저장시스템

3.1 멀티미디어 객체를 위한 객체 관리자

다양한 크기의 영상적인 멀티미디어 객체들을 저장 및 관리하기 위해 SHORE를 이용하여 객체 관리자를 설계 및 구현한다. 이를 위해 OBJECT_m이라는 관리자 클래스를 설계하고 ss_m 클래스로부터 상속 받아 구현한다. OBJECT_m 클래스는 크게 다음과 같은 4가지 모듈 및 API로 구성된다.

* Device Management
  - CreateDB() // 볼륨의 생성
  - OpenDB() // 볼륨의 개방
  - CloseDB() // 볼륨의 사계

* 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 읽기
  - InsertIntoObject() // 오브젝트 부분 삽입
  - AppendToObject() // 오브젝트 부분 삭제
  - 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

또한 다음은 제공된 객체 관리 API를 이용하여 파일을 생성하고 객체를 생성하는 예시이다.

(1) W_DO(OBJECT->begin_xct());
(2) REQUEST_W_DO(OBJECT->CreateFile(DBid, "News96", Fid));
(3) W_DO(OBJECT->CreateObject(Dbid, Fid, buf, buffer, Oid));
(4) REQUEST_W_DO(OBJECT->commit_xct());

(1)과 (6)는 트랜잭션의 시작과 종료를 의미하며, (3)와 같이 "News96"이라는 파일을 생성하고 (4)는 오브젝트 생성하여 비파의 내용을 삽입한다.

3.2 테스트 문서를 위한 역할임 관리자

대형의 테스트 문서를 효율적으로 저장 및 관리하기 위해 백로그를 응용한 역할임 기법을 SHORE level2에 확장한다. 일반적인 역할임 기법의 경우 B+트리와 포스트링 파일을 구성되지만, 각각의 기 값을 포스트링 레코드를 생성 또는 추가하는 작업이 이루어지기 때문에 수많은 I/O가 발생할 뿐만 아니라 부가저장 공간의 측면에서도 상당한 성능의 개선을 가져오게 된다. 따라서 하나의 기 값에 해당하는 포스트링 레코드의 내용을 백로그를 통해 SHORE의 비록 관리자에 담아두고 해당 기의 작업이 끝났을 때 일반적으로 저장함으로써 성능을 향상시킬 수 있다. 이를 위해 본 논문에서는 SHORE를 확장하기 위한 개선된 역할임 관리자 클래스인 invt_m 클래스를 정의한다. B+트리와 포스트링 파일은 SHORE에서 제공하는 메소드를 이용하여 동시성을 제공하기 위해 트랜잭션 관리자와 임시 관리자를 이용하여 설계한다. 또한 포스트링 파일의 경우 백로그에서 가장 우수한 성능을 가질 수 있도록 Extent의 수를 적절히 설정하여 설계한다. 한편 포스트링 파일의 구조도 매우 종종한 B+트리의 구조로 백로그를 이용하여 저장할 수 있도록 하여 이 B+트리는 SHORE에서 제공하는 메소드를 이용한다. 한편 확장된 역할임의 검색을 위해 본 논문에서는 서래개념을 이용한 검색 API를 설계한다. 개

<table>
<thead>
<tr>
<th>표 1. SHORE에 확장된 역할임 관리자 API</th>
</tr>
</thead>
<tbody>
<tr>
<td>확장된 역할임 API</td>
</tr>
<tr>
<td>CreateInvtIndex()</td>
</tr>
<tr>
<td>DestroyInvtIndex()</td>
</tr>
<tr>
<td>InsertDocInvt()</td>
</tr>
<tr>
<td>InsertInvtBulkInvt()</td>
</tr>
<tr>
<td>DeleteDocInvt()</td>
</tr>
<tr>
<td>GetCursor()</td>
</tr>
</tbody>
</table>

4. 구현 및 성능고찰
볼거리미디어 객체의 효율적 저장 및 검색을 위한 객체 저장 엔진을 구현 및 성능 고찰을 위해 최신 버전인 SHORE2.0을 사용하고, 구현환경은 CPU 650 MHz dual, 메모리 512MB의 리눅스 사례에서 수행하였다. 본 논문에서 제시하는 객체 저장 엔진은 비정형 문서들을 저장하기 위한 논리적 단위로서 Store를 사용한다. 하나의 Store는 기본적으로 두개의 Extent로 구성되어지며 하나의 Extent는 8개의 페이지로 구성된다. 이때 첫번째 Extent는 Small Object를 위한 공간이며 두번째 Extent는 Large Object를 위한 공간으로 예약된다. 각각의 Extent는 64Kb 크기로 새로운 객체를 위한 공간이 없을 경우 빈 Extent를 병합하여 Store를 확장하여 사용하고 이론적으로 4Gb까지 확장이 가능하다. 그러나 하나의 Store를 확장하여 사용하는 경우 새로운 Extent를 찾아 데이터를 삽입하는 과정에서 많은 계산시간으로 인해 성능이 감소하게 되고, 각각의 삽입 연산에 대해서 개별적인 Store를 생성하여 사용할 경우 이미 예약된 두개의 Extent에 저장되지 않은 빈 공간들이 무수히 많아지기 때문에 저장공간의 낭비를 초래한다. 따라서 대량의 문서에 대해 효율적인 저장 및 검색을 위해서는 삽입성능과 부가 저장공간의 두 가지 측면을 만족시킬 수 있는 방법이 필요하다.

<table>
<thead>
<tr>
<th>문서 크기</th>
<th>구분</th>
<th>방법 1</th>
<th>방법 2</th>
<th>방법 3</th>
<th>방법 4</th>
<th>방법 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.7K 20만건</td>
<td>삽입 (sec)</td>
<td>2501</td>
<td>2427</td>
<td>2413</td>
<td>2442</td>
<td>2874</td>
</tr>
<tr>
<td></td>
<td>검색 (sec)</td>
<td>0.04</td>
<td>0.04</td>
<td>0.04</td>
<td>0.04</td>
<td>0.04</td>
</tr>
<tr>
<td></td>
<td>SO (%)</td>
<td>224</td>
<td>134</td>
<td>123</td>
<td>120</td>
<td>118</td>
</tr>
<tr>
<td>1-3K 20만건</td>
<td>삽입 (sec)</td>
<td>2651</td>
<td>2523</td>
<td>2511</td>
<td>2595</td>
<td>3116</td>
</tr>
<tr>
<td></td>
<td>검색 (sec)</td>
<td>0.04</td>
<td>0.04</td>
<td>0.04</td>
<td>0.04</td>
<td>0.04</td>
</tr>
<tr>
<td></td>
<td>SO (%)</td>
<td>289</td>
<td>144</td>
<td>124</td>
<td>118</td>
<td>116</td>
</tr>
</tbody>
</table>

이를 위해 본 논문에서는 1.7Kb 크기의 문서 20만 건과, 1-3Kb의 가변적 크기를 가진 문서 20만 건을 이용하여 다음 표 2와 같은 실험을 수행한다. 각각의 실험을 위해 문서 20만 건과 문서의 포지션을 이용하여 100개 문서를 구현하였다. 이에 범용 1은 하나의 Store가 2개의 Extent만을 가지고 공간이 적을 경우 새로운 Store를 할당하는 방식이고, 방법 2에서는 범용 5까지 Store는 각각 5개, 10개, 15개, 20개의 Extent를 할당하는 방법이다. 검색시간 측면에서 보면 하나의 문서 검색이 0.004sec로 동일한 결과를 보이고 있다. 삽입시간은 가변적 크기의 문서에 대해서 새로운 페이징을 할당하는 복잡성 때문이며, 방법 5는 20개의 Extent에서 데이터를 삽입할 공간을 찾는 계산시간의 증가에 성능이 감소한다. 또한, 부가저장 공간 측면에서는 방법 5가 가장 우수하다. 이는 모든 새로운 Store를 생성한 뒤 발생하는 빈공간의 낭비를 최소화 하기 때문이다. 따라서 삽입시간과 부가저장 공간의 효율성을 만족시키는 방법을 찾는 것은 머신의 1과 같다.

식 1. Weight = (1 * Wi) + (SO * Wso)

위 식에서 Wi는 삽입시간의 중요도, Wso는 부가저장 공간의 중요도이다. 먼저 삽입시간과 부가저장 공간의 수치를 0에서 1까지 정규화하여 각각의 실험에 적용한다. 그림 2는 Wi와 Wso의 각각의 가중치를 주었을 때의 결과로, 10개의 Extent를 사용한 방법 3의 경우가 가장 효율적인 성능을 보인다.

5. 결론
본 논문에서는 대용량의 백스트, 이미지, 비디오와 같은 멀티미디어 객체를 효율적으로 관리하기 위해 기존의 SHORE 하둡저장 시스템을 확장하여 다양한 객체들을 위한 객체 저장 엔진을 설계 및 구현하였다. 이를위한 SHORE 저장 시스템의 내부에서 각 객체들이 저장되는 방식을 분석하고 하나의 Store가 확장될 수 있는 최적의 환경을 찾아내어 이를 통해 객체 저장 엔진을 구현하였다. 먼저 이미지, 비디오와 같은 이질적인 멀티미디어 객체를 위해서 분류, 파일, 오브젝트, 인덱스를 통합 관리할 수 있는 객체 관리자를 설계 및 구현하였고, 대량의 백스트 문서를 위해서 볼크로밍을 통용한 개선된 역학적인 인덱스는 SHORE의 level 2에 삽입함으로써 효율적인 문서 관리와 함께 DBMS 측면의 동시성 제어, 복구기법 등을 지원할 수 있도록 하였다.

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

6. 참고 문헌