필터링에 기반한 고차원 색인구조의 동시성 제어기법의 설계 및 구현

이 용 주[†] · 장 재 우[††] · 김 학 영[†††] · 김 명 준[††††]

요 약

최근 이미지, 비디오와 같은 멀티미디어 데이터에 대한 효율적인 검색을 위해 많은 다차원 및 고차원 색인 구조들에 대한 연구가 활발히 진행되고 있다. 하지만 기존의 색인 구조의 연구 방향은 검색의 효율을 극대화 하는데 초점을 맞추어 왔으며 최근의 멀티미디어 데이터베이스 나 데이터 마이닝 분야와 같은 다수 사용자 환경을 요구하는 환경에서는 부적합한 실정이다. 이에 본 논문에서는 기존의 제시된 차원이 증가하면서 급속하게 성능이 저하되는 문제를 특징 벡터의 시그니처를 구성하여 완화시킨 필터링에 기반한 고차원 색인 구조에 동시성 제어기법을 설계 및 구현하여 위스콘신 대학에서 개발한 지속성 객체 저장 시스템인 SHORE 하부저장 시스템과 밀결합 방식으로 통합하였다. 확장된 SHORE 하부저장 시스템은 고차원 데이터에 대한 효율적인 검색 뿐만 아니라 레코드 레벨의 색인 데이터에 대한 동시성 제어를 지원하며 시그니쳐 파일을 모두 메모리에 로딩하는 구조를 개선하여 페이지 레벨의 관리가 가능하다. 아울러 본 논문에서 제시한 확장된 SHORE 하부저장 시스템을 실제 응용 시스템에 적용하기 위해 플랫폼 독립적인 환경을 지원하는 자바 언어를 사용하여 미들웨어 구축 방안을 제시한다. 또한 구축된 미들웨어를 통해 쓰레드 별로 대표적인 내용기반 질의 형태인 포인트질의, 범위질의, k-최근접 질의에 대한 다수 사용자 환경에서의 성능 평가를 수행하였다.

Design and Implementation of High-dimensional Index Structure for the support of Concurrency Control

YongJu Lee[†] · JaeWoo Chang[††] · HagYoung Kim[†††] · MyungJoon Kim[††††]

ABSTRACT

Recently, there have been many indexing schemes for multimedia data such as image, video data. But recent database applications, for example data mining and multimedia database, are required to support multi-user environment. In order for indexing schemes to be useful in multi-user environment, a concurrency control algorithm is required to handle it. So we propose a concurrency control algorithm that can be applied to CBF (cell-based filtering method), which uses the signature of the cell for alleviating the dimensional curse problem. In addition, we extend the SHORE storage system of Wisconsin university in order to handle high-dimensional data. This extended SHORE storage system provides conventional storage manager functions, guarantees the integrity of high-dimensional data and is flexible to the large scale of feature vectors for preventing the usage of large main memory. Finally, we implement the web-based image retrieval system by using the extended SHORE storage system. The key feature of this system is platform-independent access to the high-dimensional data as well as functionality of efficient content-based queries. Lastly, We evaluate an average response time of point query, range query and k-nearest query in terms of the number of threads.

키워드 : 동시성 제어(Concurrency Control), 고차원 색인구조(High-dimensional Index Structure), 멀티미디어 데이터(Multimedia Data), SHORE 하부저장 시스템(SHORE Storage System)

1. 서 론

최근 데이터베이스 응용에서 내용 기반 검색에 대한 많은 관심이 대두되고 있다. 대용량의 멀티미디어 데이터베이스나 데이터 마이닝 분야에서 사용자의 다양한 요구를 보다 효율적으로 지원하기 위해서는 효과적인 내용기반 멀티미디어 검색 기법이 요구된다. 멀티미디어 검색 기법은 멀티미디어 객체로부터 특징벡터(feature vector)를 추출하여 데이터베이스에 저장한 후, 사용자가 검색하려고 하는 멀티미디어 객체의 특징 벡터와 가장 유사한 값을 찾는 것이다. 이러한 내용 기반 검색을 효율적으로 수행하기 위한 색인구조로 X-TREE[1], TV-TREE[2], R*-TREE[3] 등과 같은 고차원 색인 구조들에 대한 많은 연구가 이루어졌다. 하지만 이러한

† 정 회 원 : 한국전자통신연구원 컴퓨터시스템연구부 연구원
†† 종신회원 : 전북대학교 전자정보공학부(컴퓨터공학과) 교수
††† 정 회 원 : 한국전자통신연구원 컴퓨터시스템연구부 책임연구원
†††† 종신회원 : 한국전자통신연구원 컴퓨터소프트웨어 연구소 책임연구원
논문접수 : 2001년 12월 31일, 심사완료 : 2002년 11월 21일

고차원 색인 구조들은 차원이 증가하면 급격히 성능이 저하되는 dimensional curse[4, 5] 문제가 야기되며 이를 해결하고자 특징 벡터의 요약 정보를 순차 탐색하는 필터링에 기반한 고차원 색인 구조(CBF)[6-8]가 제안되었다. 하지만 필터링에 기반한 고차원 색인 구조는 파일 시스템에서 적용되는 형태이며 이는 단일 사용자 환경만을 고려하여 구현되었기 때문에 데이터베이스 관리 시스템의 하부저장 구조에 직접적으로 적용하기가 어렵다. 아울러 시그니쳐 파일을 메인 메모리에 모두 로딩하는 구조로써 대용량의 시스템에는 부적합한 실정이다.

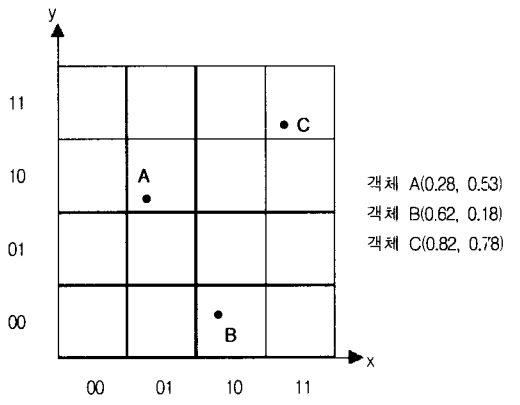
따라서 본 논문에서는 다중 사용자 환경에서 사용할 수 있도록 필터링에 기반한 고차원 색인 구조의 동시성 제어기법을 설계 및 구현한다. 아울러 동시성 제어기법을 지원하는 필터링에 기반한 고차원 색인 구조를 일반적인 파일 시스템에 적용하기 보다는 하부저장 시스템에 통합함으로써 보다 일반적인 어플리케이션에 적용할 수 있게 한다. 이를 위해 이질적인 데이터에 대한 검색을 용이하게 할 수 있으며 객체지향적 스토리지 성격을 가진 미국 Wisconsin 대학에서 개발한 SHORE 하부저장 시스템을 사용하였다. SHORE 하부저장 시스템은 Layered Architecture를 채택하고 있으며 인덱스를 위한 Index Layer가 존재하므로 여기에 밀결합 방식으로 통합하여 구현하였으며 페이지 레벨로 시그니쳐 데이터에 대한 검색이 가능하게 하였다. 구축한 하부저장 시스템의 검증을 위해 실제 웹 상에서 유사성에 기반한 이미지 검색 시스템을 구축하였다. 이를 통해 자바 언어의 특성과 SHORE 서버 환경을 고려하여 자바의 장점인 플랫폼 독립성을 유지하면서 응용 프로그램과 애플릿에서 모두 SHORE 서버를 사용할 수 있도록 SHORE 패키지 모델의 특성을 지닌다. 아울러 제시한 시스템을 통해 쓰레드 별로 다수 사용자 환경을 재연하여 고차원 색인 구조의 대표적인 질의 형태인 포인트 질의, 범위질의, k-최근접 질의에 대한 성능 평가를 차원에 따라 수행하였다.

본 논문의 구성은 다음과 같다. 2장에서는 필터링에 기반한 고차원 색인 구조에 대한 연구와 이를 하부저장 구조에 확장하기 위한 연구에 대해 기술한다. 3장에서는 필터링에 기반한 고차원 색인 구조의 동시성 제어기법의 설계 및 SHORE 하부저장 시스템에 통합된 고차원 색인 관리자의 구현과 사용자 API에 대해서 기술한다. 4장에서는 구현된 시스템에 대한 성능평가를 제시하고, 5장에서는 구현된 고차원 색인 관리자를 이용하여 웹 기반 응용을 위한 자바 미들웨어의 구축 방안을 기술한다. 마지막으로 6장에서는 결론 및 향후 연구 방향을 기술한다.

2. 관련 연구

2.1 필터링에 기반한 고차원 색인 구조

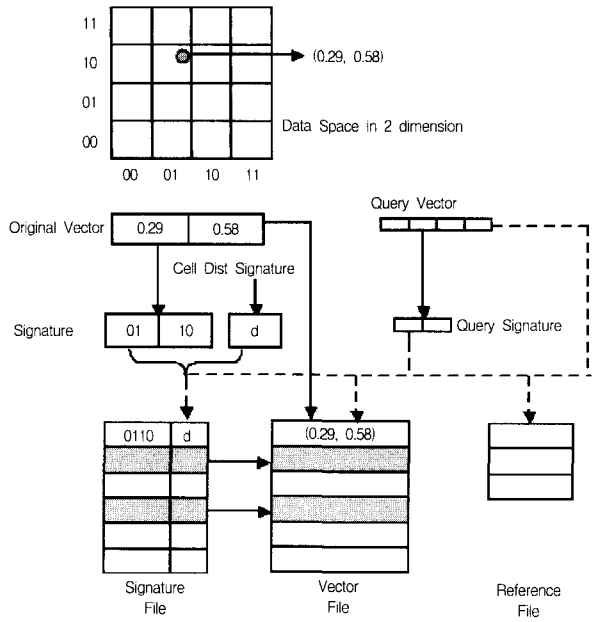
고차원 데이터에 대한 시그니쳐는 데이터 각각의 차원에 따른 특징 벡터에 대한 요약들의 집합이므로, 각 차원에 해당하는 벡터의 성질을 유지하도록 표현할 수 있어야 한다. 또한, 질의를 처리하기 위해서, 질의 벡터와 시그니쳐들 사이의 관계를 표현할 수 있어야 한다. 따라서, 특징 벡터를 시그니쳐로 변환하는 방법은 데이터 공간을 셀(cell) 단위로 나누고, 셀에 대한 시그니쳐를 사용하는 것이다. 이렇게 만들어진 시그니쳐는 각각의 셀을 대표하는 값이 되며, 셀에 대한 정보를 포함하게 된다. 또한, 차원에 따라 데이터 공간을 균등하게 분할함으로써, 셀에 대한 시그니쳐는 데이터 공간에서의 어떤 영역의 범위를 가지게 되고, 시그니쳐를 통해 쉽게 이 범위값을 구할 수 있다. (그림 1)은 각각의 차원에 대해서 2-비트 시그니쳐를 사용할 경우, 2차원 공간에서의 시그니쳐를 만들어내는 과정을 나타낸다.



(그림 1) 2차원 공간에서의 시그니쳐 생성과정

그림에서 각 차원에 따른 특징 벡터의 시그니쳐 크기를 2비트로 할당하므로 각 차원에 대한 데이터 공간은 4개의 부분으로 나뉘어져, 2차원 공간에서 총 16개의 셀을 가지게 된다. 차원에 따른 데이터 공간에서 표현할 수 있는 객체들의 실제 값이 0에서 1사이인 경우, 각각의 셀을 나누는 점의 좌표는 0, 0.25, 0.50, 0.75, 1이 된다. 따라서, 객체의 벡터가 0에서 0.25 사이에 있는 값은 시그니쳐 '00'으로 표현되고, 0.25 ; 0.50 사이의 값은 '01'로, 0.50 ; 0.75 사이의 값은 '10', 0.75 ; 1 사이의 값은 '11'로 각각 표현된다. 이러한 고차원 데이터에 대한 시그니쳐를 이용한 셀-기반 필터링 기법(Cell-Based Filtering Method ; CBF)은 셀의 시그니쳐 정보뿐만 아니라, 미리 계산된 셀의 중심에서 객체까지의 거리 정보를 이용함으로써 필터링 효과를 증대시킨 방법이다. (그림

2)는 셀-기반 필터링 기법을 사용하여 주어진 객체의 특징 벡터를 저장 및 검색하는 과정을 나타낸다.



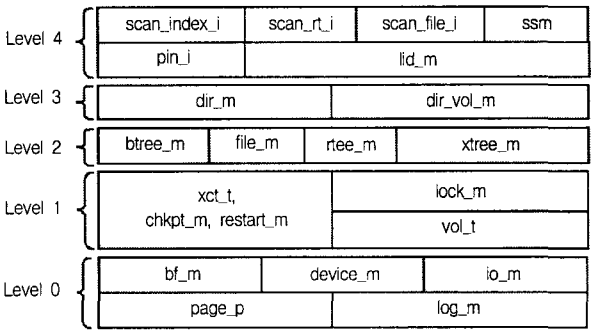
(그림 2) 셀기반 필터링 기법의 저장 및 검색

(그림 2)에서 셀-기반 필터링 기법의 저장 단계는 객체의 특징 벡터에 대한 시그니처와 객체가 포함된 셀의 중심으로부터 객체까지의 거리를 계산하여 함께 시그니처 파일에 저장한다. 또한, 참조 테이블(Reference Table) 파일을 조사하여 파일의 알맞은 위치를 찾고 이를 저장하게 된다. 즉, 참조 테이블에는 시그니처 파일이나 데이터 파일로부터 현재까지 삭제된 레코드의 위치 정보를 유지하고 있어서, 새로운 데이터를 추가할 때는 우선적으로 현재 비어있는 레코드 위치에 데이터를 삽입하게 된다. 검색을 수행할 때는 우선 질의 벡터에 대한 질의 시그니처를 생성하고, 시그니처 파일을 탐색함으로써 필터링을 수행하여 후보 셀들을 얻고, 최종적으로 데이터 파일을 접근함으로써 질의에 해당하는 결과를 얻게 된다.

2.2 SHORE 하부저장 시스템

SHORE(Scalable Heterogeneous Object REpository)는 미국 위스콘신 대학에서 개발한 지속성 객체 저장 시스템으로 기존의 OODB(Object-Oriented DataBase)에서의 단점인 파일 시스템과의 결합의 어려움, 트랜잭션 관리의 어려움, Peer-to-Peer 구조의 부적절성 등을 해결하기 위해 개발되었다[9]. SHORE는 객체 지향 데이터 베이스 기술과 파일 시스템 기술을 합성하여, 기존의 UNIX 파일 시스템 기반 응용 프로그램을 쉽게 접목시킬 수 있는 장점을 지닌다. SHORE가 가지는 특징은 객체 지향 자료 모델링 지원, 멀티미디어 자료 저장 관리 지원, 전문 정보 색인 및 검색 지원, 트랜잭션 처리 및 회복 기능 지원 등이다.

SHORE 하부저장 시스템의 SSM은 (그림 3)과 같이 기본적으로 5단계의 계층으로 이루어진 Layered-Architecture를 채택하고 있다



(그림 3) 확장된 SHORE 하부저장 시스템의 계층 구조

각 계층마다 독립적인 역할 수행을 하는 객체지향 기법을 사용한다. 이때 시스템의 동작은 제일 하위 계층인 Leve l0에서부터 Level 4까지 각각의 독립적인 기능들을 상속 받아 수행된다. 보다 세부적인 사항을 살펴보면 Level 0에서는 시스템의 하위저장 부분을 담당하는 Device, I/O, Buffer와 데이터베이스의 회복기법인 Log 관리자가 담당하고, Leve l1에서는 트랜잭션 관리자와 잠금 관리자가 존재한다. Level 2에서는 파일과 B+트리, R*트리에 대한 관리자를 제공해주며, Level 3에서는 디렉토리에 관한 기능을 수행한다. 마지막으로 가장 상위 레벨인 Level 4에서는 논리적인 ID에 관한 관리자와 SSM을 위해 실제 사용자 레벨의 API를 제공해준다[9, 10]. 이를 통해 사용자는 상위레벨의 API를 가지고 SHORE 하부저장 시스템에 접근하여 데이터를 관리한다.

아울러, 기존의 하부저장 시스템에 고차원 색인 구조를 통합하기 위한 방법으로는 소결합방식(Loosely Coupled Approach)과 밀결합방식(Tightly Coupled Approach)이 있다. 구현은 하기 어렵지만 데이터베이스 측면에서 가장 적합한 밀결합 방식으로 구현한 확장된 SHORE 하부저장 시스템(XTREE_M)이 존재한다[11]. XTREE_M은 멀티미디어 응용을 위해 이미지 및 비디오 특징벡터 검색을 위해 고차원 색인구조인 X-tree를 기존 SHORE 하부저장 시스템에 밀결합 방식으로 통합한 것이다. 또한 XTREE_M는 대표적인 잠금 결합기법(Lock-Coupling Technique)[12]으로 구현되었다. 하지만 기존의 제시된 XTREE_M은 여전히 차원이 증가하면 급격히 성능이 저하되는 dimensional curse 문제를 안고 있다. 따라서 본 논문에서는 dimensional curse 문제의 해결책으로 제시된 셀기반 필터링에 기반한 고차원 색인구

조에 대한 동시성 제어기법을 적용하여 이를 SHORE 하부저장 시스템에 밀결합 방식으로 통합하여 내용 기반 질의를 요구하는 일반 어플리케이션의 하부저장 시스템으로 제공하고자 한다.

3. 필터링에 기반한 고차원 색인 구조의 동시성 제어 기법의 설계 및 구현

필터링에 기반한 고차원 색인 구조를 SHORE 하부저장 시스템에 통합하여 멀티미디어에 적합한 시스템을 제공하기 위해서 두 가지 측면에서의 고려사항이 필요하다. 첫째, 색인 구조 측면에서는 기존의 제시된 필터링에 기반한 고차원 색인 구조의 파일 형태와 I/O의 형태를 하부저장 시스템에 맞게 설계해야 하며, 둘째, 하부저장 시스템 측면에서는 설계한 방법에 적합하게 SHORE 하부저장 시스템에 제공하는 페이지, 레코드, 동시성 제어, 회복 기법에 대한 연구가 선행되어야 하고 이를 통해 제공되는 기능과 설계 사항을 고려하여 구현이 이루어져야 한다.

3.1 인덱스 구조적인 측면에서의 설계사항

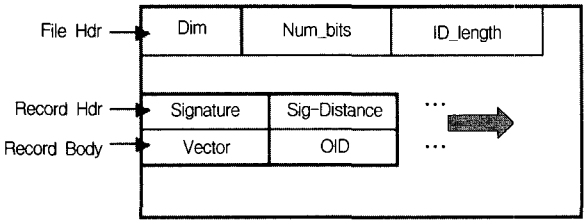
고차원색인 구조에서의 설계시 고려사항은 기존의 파일시스템에서 제시된 필터링에 기반한 고차원 색인구조를 하부저장 시스템으로 통합하기 위한 몇 가지 고려사항이 있으며 이는 다음과 같다.

① 파일 시스템의 고차원 색인 구조를 하부저장 시스템으로 확장하면서 I/O를 최소화 해야 한다.
② 기존의 파일시스템에서의 구조에서는 검색의 성능을 높이기 위해 시그니쳐 정보를 모두 메모리에 로딩하는 구조이며 이는 대용량의 멀티미디어 데이터에는 부적합하다.

첫째, 파일 시스템에서의 검색 성능의 향상에 중점을 두어 파일을 여러개 두는 구조는 하부저장 시스템으로 확장하면 불필요한 I/O 발생의 원인이 된다. 이를 위해서 여러 파일을 통합할 필요성이 제기되었다. 기존에 제시된 CBF 구조에서 먼저 삭제를 위해 삭제되는 위치의 파일 포인터를 저장하는 참조 파일은 하부저장 구조로 확장을 하면 하부저장 구조 측면에서 페이지 관리가 이루어지므로 고려사항에서 제외된다. 또한 시그니쳐 정보와 벡터 정보를 하나로 통합하여 하부저장 구조의 논리적 파일 하나로 통합함으로써 I/O를 줄일 수 있는 계기를 마련하였다. 이러한 하부저장 구조의 논리적 파일을 HD 파일이라 명한다.

둘째, 데이터베이스의 사이즈가 증가하면서 좀더 안정적인 시스템으로 확대되기 위해서는 버퍼를 사용하여 임의의 블록 사이즈 만큼씩 읽는 형태로 바뀌어야 한다. 이를 위해 SHORE가 제공하는 기본적인 페이지 사이즈인 8K를 고려하여 8K 만큼씩 읽어서 Pruning을 수행하는 절차를 가져야 한다. 이러한 방식은 실제 시그니쳐 정보가 매우 작으므로 상당히 많은 양의 시그니쳐를 한 번에 불러올 수 있다. 또한 기존의 파일시스템에서는 전체 데이터의 개수를 메모리 상에서 유지하기 때문에 동시에 여러 쓰레드가 삽입을 하면 일관성을 잃어버리게 된다. 이를 해결하기 위해 파일의 맨 처음 레코드에 전체 데이터에 대한 헤더 정보를 유지한다.

이를 통해 설계한 SHORE 하부저장 시스템에서 필터링에 기반한 고차원 색인구조 HD의 파일 구조는 (그림 4)와 같다.



(그림 4) SHORE 상의 HD파일의 내부 구조

HD 파일의 파일 헤더에는 기존의 인덱스를 관리하기 위한 정보인 차원정보, 비트정보, ID 길이 정보를 저장하고 레코드부분에 기존의 파일 시스템의 시그니쳐 정보와 벡터 정보를 저장하는 구조를 채택하였다. 레코드의 헤더 부분에는 시그니쳐 파일에는 특징벡터의 요약 정보인 시그니쳐와 셀의 중심 점까지의 거리 정보는 r_dist정보를 저장한다. 아울러 레코드의 바디 부분에는 원 벡터 정보를 저장한다. 그 이유는 실제 특징 벡터의 요약 정보인 시그니쳐 정보가 상당히 작으므로 레코드의 헤더에 저장하였으며 이에 비해 벡터 정보가 크므로 레코드의 바디 부분에 저장하는 구조이다.

3.2 하부저장 시스템 측면에서의 설계사항

다음으로 SHORE 하부저장 시스템에 고차원 색인 구조를 밀결합 방식으로 통합하기 위해서는 기본적으로 SHORE가 제공해주는 동시성 제어기법, 페이지 관리 기법, 레코드 관리 기법을 분석하고 이를 고차원 색인 구조에 적합하게 설계해야 하므로 이를 위해 필요한 고려사항은 다음과 같다.

① SHORE가 제공하는 동시성을 위한 Lock 관련 사항을 고려해야 한다.
② SHORE 상의 Page의 특성을 고려하여 색인 구조 설계가 이루어져야 한다.
③ SHORE의 Record의 형태를 고려해야 한다.

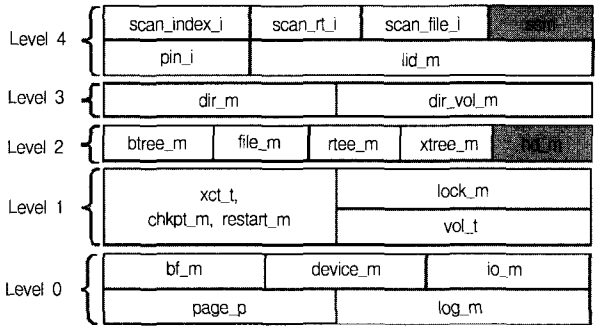
첫째, Lock 관련 부분에서 SHORE는 기본적으로 페이지 레벨 잠금을 지원하며 사용자 레벨에서 사용할 수 있는 논리적 ID를 통한 잠금 API 들과 Level 1에서 물리적 ID를 통해 잠금을 다룰 수 있는 API 들로 이루어져 있다. SHORE가 제공하는 기본적인 잠금 관련 API는 잠금을 획득하기 위한 lock( ) 메쏘드와 잠금을 해제하는 unlock( ) 메쏘드, 현재 노드에 잠금의 형태를 알아 볼 수 있는 query_lock( ) 메쏘드가 존재한다.

둘째, SHORE의 잠금 유지는 기본적으로 페이지의 설정에 영향을 받는다. 처음 레코드가 삽입되면 8K의 빈 페이지를 할당 받는다. 삽입이 계속 이루어지면 페이지의 슬롯에 잠금을 유지하며 이를 상위 계층에 영향을 준다. 페이지는 기본적으로 8K 중 페이지 유지 정보와 레코드를 가리키는 슬롯들로 구성되어 있다. 또한 로깅을 위해 LSN이 두 개 존재하며 링크드 리스트 구조를 채택하고 있어서 next와 prev 포인터와 page와 store 종류를 나타내는 여러 가지 플래그 정보가 존재한다. 레코드가 할당되는 slot[0] 부터 점점 페이지에 할당되는 형태이며 SHORE는 내부적으로 LATCH와 LOCK을 혼용하면서 슬롯에 상호 배제를 유지하는 형태이다.

셋째, SHORE 상에 존재하는 Record 형태는 크게 SMALL과 LARGE를 지원하며 실제 특징 벡터의 정보는 대체적으로 8K 미만이므로 SMALL RECORD를 채택하기로 한다. SMALL RECORD는 한 페이지 내에 레코드를 삽입하고 그 페이지에 슬롯 정보도 존재한다. 레코드를 할당할 때 기존의 레코드 사이즈를 초과하면 새로운 페이지를 할당하는 형태이다. LARGE RECORD는 레코드의 바디 부분이 또 다른 페이지를 가리키는 포인터를 가지고 있다. 2 레벨로 구현한 이 페이지 형태는 크게 2G까지의 BLOB 정보를 저장할 수 있으며 최대 SHORE 상에는 3 레벨까지 구현이 가능하다.

3.3 동시성 제어기법의 구현

필터링에 기반한 고차원 색인구조의 SHORE 하부저장 시스템에서 확장을 위해 사용되어진 클래스 형태는 다음과 같다. 먼저 사용자 레벨의 클래스인 ss_m 클래스에는 사용자가 사용할 수 있는 논리적 ID를 가진 API가 존재하며 이 API를 수행함으로써 삽입과 검색 측면에서 동시성을 보장받는다. 다음 2 레벨에 XTREE, RTEE와 같이 HD 클래스가 존재하며 이는 물리적 ID를 사용하는 클래스이며 밀결합 방식으로 구현되었다. 또한 기존의 시그니쳐 관리 클래스는 2 레벨의 HD 클래스에 통합하였으며 레코드 삭제 관리 클래스인 GARBAGE 클래스는 제외하였다. 확장된 SHORE 하부저장 구조는 (그림 5)와 같이 크게 상위레벨 API와 2 레벨의 hd_m 클래스로 나타내어진다.



(그림 5) 확장된 SHORE 하부저장 시스템의 구조

클래스의 상속 관계와 구현한 클래스의 이름은 다음과 같다.

- **class ss_m : public smlevel_top**
  - 기존의 사용자 레벨의 클래스에 사용자 API를 추가하였다.
- **class hd_m**
  - 레벨 2에 밀결합 방식으로 구현된 클래스이다.
- **class scan_file_i**
  - 기존의 파일 관련 검색에 사용되는 scan_file_i 클래스를 사용하였다.

3.3.1 삽입 알고리즘

삽입과정은 크게 4단계로 이루어진다.

첫 번째 단계는 SHORE 상에서 로깅을 ON 했는지 OFF 했는지 체크하여 ON한 경우 트랜잭션 레벨에서 로깅을 배제시킨다. 이를 위해 트랜잭션 클래스에서 제공되어지는 xct_log_switch_t log_off(OFF)를 사용하여 삽입 메쏘드에서 구간을 정해 이를 구현하였다.

두 번째 단계는 인덱스 측면에서 주어진 특징 벡터를 시그니쳐로 변환하는 루틴과 구해진 시그니쳐의 중심 셀 까지의 거리 정보를 얻는 루틴으로 이루어진다. 특징 벡터를 시그니쳐로 변환하는 루틴은 초기 인덱스 생성 단계에서 주어진 비트 수를 가지고 이를 요약정보로 추출해 낸다. 기본적으로 제공되어지는 비트 수는 2비트, 4비트, 8비트 시그니쳐를 지원한다. 구해진 시그니쳐의 중심 셀 까지의 거리는 유클리언트 거리를 사용하여 계산하였으며 구해진 거리 정보는 검색시 MINDIST와 MAXDIST를 계산할 때 사용되어진다. 인덱스의 거리정보는 실제 부가 저장 공간이 작기 때문에 어느 정도의 오버헤드는 증가하지만 Pruning을 최대한 보장할 수 있는 이점이 있다.

세 번째 단계는 구해진 시그니쳐 정보와 거리정보를 SHORE 상의 논리적 파일의 맨 마지막에 삽입하는 것이다. 내부적으로 마지막 페이지를 찾고 이 페이지의 마지막 슬롯에 데이터를 넣는 구조인데 마지막 페이지에 레코드 삽입시 빈 공간

이 부족하면 새로운 페이지를 할당한다. 또한 삽입 이전에 레코드를 위한 고유 ID를 generation하여 이를 전역 BTREE에 삽입한다. 그런 다음 레코드의 슬롯에 잠금을 유지하고 페이지, 스토어, 볼륨까지 잠금 관련 정보를 LOCK_M에서 관리한다. 이러한 잠금 정보는 실제 잠금 해시 테이블 사이즈가 한정되어 있으므로 장기 트랜잭션을 수행 할 때는 문제가 발생한다. 이를 위해 COMMIT 시점을 적절히 선택하였다.

네 번째 단계는 삽입된 레코드의 잠금을 해제하기 전에 인덱스에 들어간 모든 레코드의 개수를 체크하는 루틴을 거치게 된다. 주어진 논리적 파일의 맨처음 레코드는 전체 레코드 개수를 체크하는 레코드로 이 레코드에 잠금을 획득하여 전체 레코드 개수를 하나 증가시키는 루틴을 가진다. 이러한 네 가지 단계를 통해 하나의 특징 벡터의 시그니처 정보, 거리정보, 원 벡터를 삽입하며 Lock Escalation 임계 값을 조절하여 최대한의 동시성을 보장할 수 있다.

제시하는 삽입 알고리즘은 다음과 같다.

```
Procedure hd_insert                    /* insert algorithm */
Input : query_vector, ID
Output : Return Code
sig_vector ;                           /* signature vector */
objdist ;                              /* center distance */
if (not transaction logging off)       /* Logging Check */
    xct_log_switch_t log_off (OFF) ;
end
makeSig (query_vector, sig) ;          /* calculate the signature */
extractSig (sig, sig_vector) ;
getObjectDist (sig_vector, objdist) ;  /* calculate cell dist */

p = start ;                            /* signature start page */
loop
 if (p is last page )
    latchmode = X-mode ;
    if (not enough space in page)
       new page allocated
    else
       insert [sig_vector, ID, query_vector] on last record
    unlatch (p) ;
 else
    search last page
end
if (total_num record is locked)
    wait for release
else
    update_total_num (first record serial number) ;
End Procedure
```

3.3.2 검색 알고리즘

검색 알고리즘은 크게 3단계로 이루어진다.

첫 번째 단계는 검색을 위해 SCAN 클래스를 인스턴스화하여 HD 파일의 첫 번째 레코드를 읽는다. 이 첫 번째 레코드에서 전체 데이터베이스에 들어있는 데이터의 개수가 존재한다. 그런 다음 두 번째 레코드 부터 블록 단위로 읽어서 시그니처를 메모리에 로딩한다. 메모리에 로딩하는 이유는 Pruning 단계에서 좀더 적은 I/O를 위해 메모리에 로딩하여 처리한다. 두 번째 단계는 메모리에 로딩된 시그니처를 순차적으로 검색하면서 적절한 후보 시그니처를 리스트 형태로 저장한다. 저장된 시그니처는 3차 Pruning에서 I/O를 발생시키면서 결과 벡터를 얻는다. 세 번째 단계는 필터링에 기반한 고차원 색인 구조의 필터링 단계이다. 이 단계에서는 이전 단계에서 얻어진 후보 셀들을 lower bound의 오름차순으로 정렬한다. 그 이유는 lower bound 값이 작은 셀을 먼저 탐색하기 위함이다. 즉 질의 포인트와 가장 가까운 셀들을 먼저 탐색함으로써 검색 성능을 높일 수 있기 때문이다. 3차 Pruning에서는 필터링을 수행하기 위해 후보 셀의 lower bound값과 현재 얻어진 객체거리 값을 비교하여 적절한 후보 셀을 찾고 부적절한 후보 셀은 리스트로부터 제거된다. 얻어진 결과를 사용자에게 리스트 형태로 반환하며 반환되는 값은 벡터와 벡터의 ID 정보가 된다.

k-최근접 검색 알고리즘은 다음과 같다.

```
Procedure hd_knn_search                /* knn search algorithm */
Input : query_vector, k_value
Output : Object ID
sig_vector ;                           /* signature vector */
objdist ;                              /* center distance */
candidate_vector ;                     /* candidate vector */
result_set ;                           /* searched result set */

makeSig (query_vector, sig) ;          /* calculate the signature */
extractSig (sig, sig_vector) ;
getObjectDist (sig_vector, objdist) ;  /* calculate cell dist */
p = start                              /* start signature page */
loop
 if (p is not last page)
    latchmode = S-mode ;
    Read One page ;
    candidate_vector = knn_first_pruning (sig_vector) ;
    if (candidate_vector is adjusted)
       add entry to first pruning result set ;
    unlatch (p) ;
 end
 loop
 knn_second_pruning (candidate_vector) ;
 loop
    for each candidate record entry
       compare object vector to query_vector
       if (consistent (k_value) )
          attach search predicate to result set
 end
end
 End Procedure
```

사용자에게 제공되는 API는 크게 생성, 파괴 API, 개방, 폐쇄 API 삽입, 삭제 API, 검색 API로 구성되어 있다.

① 생성, 파괴 API : 인덱스를 만들거나 파괴할 때 사용하는 API이며, hd_create( ) 메쏘드는 데이터베이스 볼륨 ID가 주어지고 HD 파일의 기본적인 Description 정보

를 가지고 파일을 생성한다. 생성된 파일의 헤더에는 차원정보, 비트정보, ID 길이 정보를 가지고 있으며 파일의 첫 번째 레코드에 전체 개수를 0으로 세팅한다. SHORE 상에서는 기본적으로 모든 ID를 serial_t 타입을 사용하므로 hdid 역시 serial_t 타입의 구조체로 선언하였다. 이 정보들은 디렉터리 관리자를 통해 식별자들과의 맵핑을 이룬다. hd_destroy( ) 메쏘드는 데이터베이스 볼륨 ID, HD 파일의 description 정보를 가지고 파일을 삭제한다. HD 파일의 식별자를 가지고 디렉토리 관리자에서 물리적 ID를 얻고 이를 BTREE에서 삭제하여 물리적 STORE 또한 폐쇄한다. 이를 통해 기존의 삽입되어있는 데이터 역시 파괴된다. 하지만 기존의 삽입된 데이터가 많은 경우 물리적 ID를 관리하는 디렉토리 관리자에서 ID를 지워야 하므로 다소 시간이 소요된다.

```
static rc_t hd_create (const lvid_t & DB_id,          // DB ID
                       hdid_t & hd_id,                // HD File ID
                       const char * hd_desc,          // HD File Description
                       store_property_t property,     // Store Type
                       int      dim,                  // Dimension Information
                       int      nbits,                // Bit Information
                       int      id_length)            // ID Length
static rc_t hd_destroy (const lvid_t & DB_id,         // DB ID
                       const char* hd_desc)           // HD File Description
```

② 개방, 폐쇄 API : 인덱스를 열어서 삽입이나 검색을 위한 정보를 가져오는 API이며, 먼저 hd_open( ) 메쏘드는 HD 파일의 description정보를 가지고 논리적 HD 파일의 ID를 얻는 과정이다. 아울러 hd_close( ) 메쏘드는 실제 내부적으로 SHORE 상에서는 데이터베이스가 인스턴스화 되었을 때 이를 닫는 API는 불필요하다. 그 이유는 SHORE에서 사용하는 모든 ID 정보는 전역 BTREE에 들어 있으며 이는 항상 활성화되어 있어야 하기 때문이다.

```
static rc_t hd_open (const lvid_t & DB_id,    // DB ID
                     hdid_t & hd_id,          // HD File ID
                     const char * hd_desc)    // HD File Description
static rc_t hd_close( )
```

③ 삽입, 삭제 API : 인덱스에 데이터를 삽입하거나 삭제하는 API이며, hd_insert( ) 메쏘드는 N 차원의 특징 벡터와 ID가 주어졌을 때 이를 HD 파일에 삽입하는 루틴으로 동시성을 보장하기 위해 페이지 레벨의 잠금을 유지하며 이를 상위 레벨에 반영한다. 또한 hd_delete( ) 함수 역시 주어진 ID를 가지고 HD 파일에 존재하는 벡터와 시그니쳐 정보를 삭제하는 것이다.

```
static rc_t hd_insert (const lvid_t & DB_id,      // DB ID
                       hdid_t & hd_id,            // HD File ID
                       void * D_id,               // Document ID
                       float * vectors)           // Input Vector
static rc_t hd_delete (const lvid_t & DB_id,      // DB ID
                       hdid_t &  hd_id,           // HD File ID
                       void * D_id)               // Document ID
```

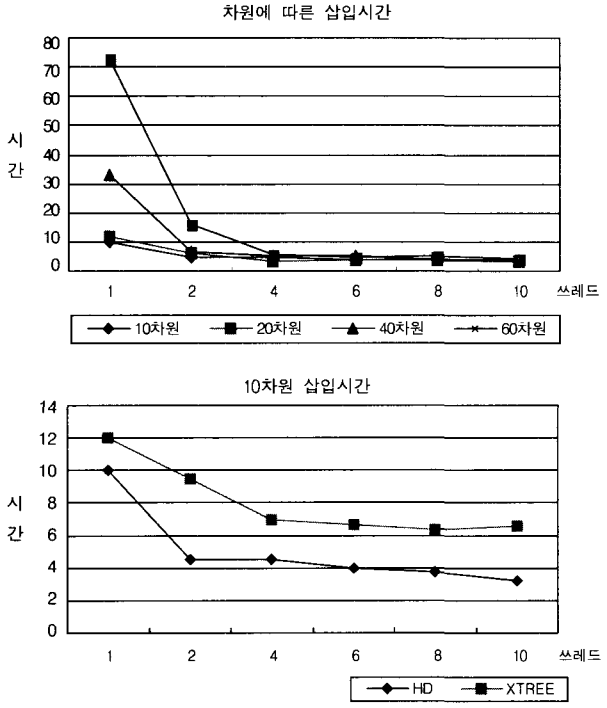
④ 검색 API : 유사성에 기반한 질의인 포인트, 범위, k-최근접 질의를 수행하는 API이며, 멀티미디어 데이터베이스 시스템에서는 이러한 유사성 질의를 효율적으로 지원하는 것이 매우 중요하다. 가장 먼저 포인터 질의의 메쏘드인 hd_point_query( ) 볼륨 ID, HD 파일 ID를 가지고 질의 벡터를 주었을 때 주어진 질의벡터와 일치하는 문서의 ID 리스트를 반환한다. 여기서 ID List 타입은 void 형 포인터이므로 어떤 형태의 문서나 이름이 가능하게 된다. 다음으로 일정한 영역(radius)를 가지고 그 범위에 포함되는 범위 질의인 hd_range_query( ) 메쏘드 역시 볼륨 ID, HD 파일 ID를 가지고 질의 벡터를 주었을 때 결과를 문서 리스트로 반환하게 된다. 마지막으로 가장 유사한 K개의 데이터를 찾는 k-최근접 질의 역시 다른 API와 유사하며 K 값이 매개변수와 주어지고 결과는 문서 리스트에 반환하게 된다.

```
static rc_t hd_point_query (const lvid_t & DB_id,  // DB ID
                            hdid_t & hd_id,        // HD File ID
                            float * query_vector,  // Query Vector
                            void * D_ids)          // Document List
static rc_t hd_range_query (const lvid_t & DB_id,  // DB ID
                            hdid_t & hd_id,        // HD File ID
                            float * query_vector,  // Query Vector
                            void * D_ids,          // Document List
                            float radius,          // Radius
                            int & result_num)      // Number of Result
static rc_t hd_knn_query (const lvid_t & DB_id,    // DB ID
                            hdid_t & hd_id,        // HD File ID
                            float * query_vector,  // Query Vector
                            void * D_ids,          // Document List
                            int knum,              // K Value
                            int & result_num)      // Number of Result
```

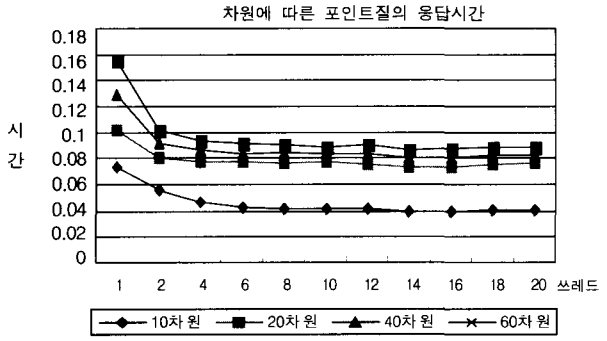
4. 성능 평가

본 장에서는 필터링에 기반한 고차원 색인 구조를 SHORE 하부저장 시스템에서 확장하여 성능 평가를 수행한다. 시스템 환경은 운영체제는 리눅스 환경이며, 650MHz Dual CPU, 128MB×4 메모리에서 측정하였다. 실험 데이터는 10차원의 synthetic data를 사용하였으며 평가 형태는 삽입시간, 포인트질의 검색시간, 범위질의 검색시간, k-최근접 질의 검색시간을 통해 쓰레드의 개수를 증가시키면서 일정량의 작업량을 가지고 응답시간의 감소율을 측정하였다. 먼저 삽입시

간의 경우 10차원, 20차원, 40차원, 60차원의 경우 각각의 쓰레드별 응답시간과 10차원의 경우 XTREE와의 성능을 측정하였다. (그림 6)에서 차원에 따른 삽입시간은 차원이 증가할수록 쓰레드별 응답시간이 급격히 감소한다. 그 이유는 차원이 증가하면서 한 페이지에 들어갈 수 있는 벡터의 수가 감소하며 이는 곧 Lock Escalation에 따른 경합을 최소화 할 수 있기 때문이다. 아울러 10차원의 경우 XTREE와의 삽입 시간 성능 비교는 한 예로 쓰레드의 개수가 4일때 약 30%의 응답시간 감소를 보인다.



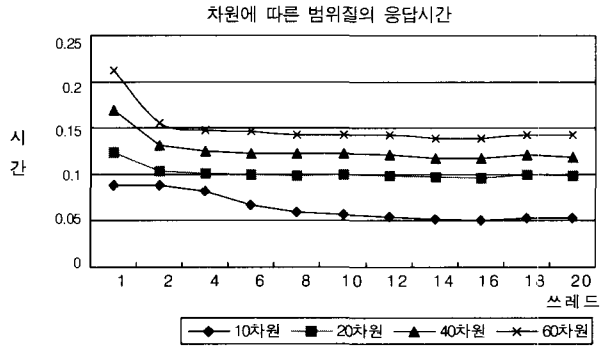
(그림 6) 삽입시간



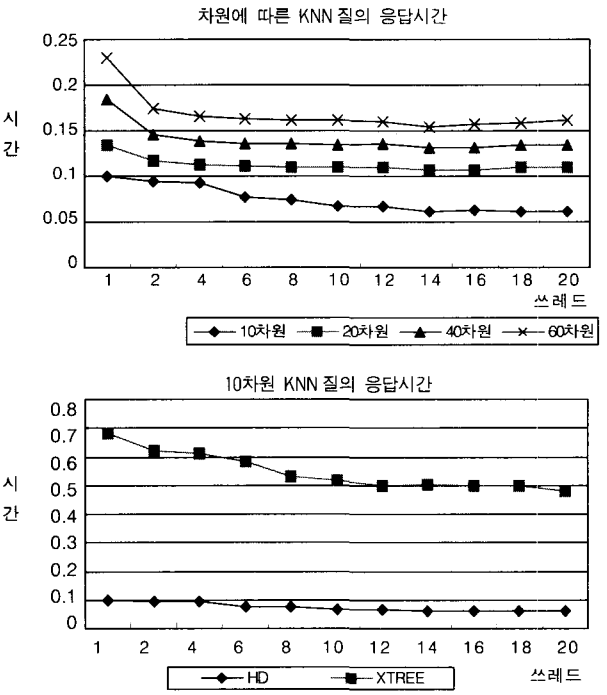
(그림 7) 차원에 따른 포인트 질의 검색 시간

주어진 벡터와 DB 내에 일치하는 벡터를 찾는 포인트질의의 경우 10차원의 100건의 데이터를 인덱스 파일에서 찾았을 경우 쓰레드의 개수가 1에서 10까지 증가하면서 2배 이상 감소되는 것을 보였다. 포인트 질의의 경우 DB 내에 일치되는 벡터를 찾으므로 최대한의 동시성을 보장받을 수 있다. (그림 7)에서는 본 논문에서 구현된 HD의 각각의 차원에 따른 포인트 질의 응답시간을 보인 것이다. 쓰레드 개수가 6개 이상 증가하면 응답시간 감소가 뚜렷해지는데 그 이유는 한 페이지에서 벡터를 찾는 쓰레드 개수가 증가하며 Lock Escalation 임계값이 설정을 기본적으로 5개로 했을 경우는 거의 직렬화 되는 것을 확인할 수 있다.

범위질의는 주어진 질의 포인트로부터 일정한 거리 영역 안에 포함된 객체를 찾는 질의이다. (그림 8)은 synthetic data에 대한 범위질의 검색시간을 쓰레드의 개수를 1에서 20개까지 증가시키면서 응답시간을 감소되는 것을 나타낸 것이다. 범위값은 전체 데이터 중에서 0.1%의 데이터를 검색할 수 있는 값을 범위 값으로 사용하였다. 각각의 범위값은 10차원인 경우 0.4, 20차원인 경우 0.7, 40차원인 경우 1.0, 60차원의 경우 1.8로 하였다.이는 약 100개 정도의 결과셋을 유도할 수 있다.



(그림 8) 차원에 따른 범위 질의 검색 시간



(그림 9) k-최근접 질의 검색 시간

k-최근접 질의는 주어진 질의 포인트로부터 가장 가까운 k개의 객체를 찾는 질의이다. (그림 9)에서는 k값을 100일 때의 차원에 따른 KNN 질의응답 시간과 10차원의 경우의 XTREE와의 성능 비교를 보여준다. 차원에 따른 KNN 질의 응답시간은 주어진 차원에 따라 응답시간의 감소를 보이고 있으며, 특히 10차원의 경우 XTREE 보다 월등히 응답시간이 감소하는 것을 볼수 있다.

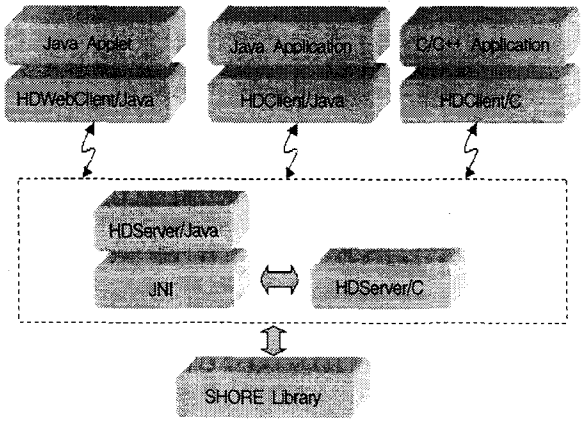
5. 자바 미들웨어를 통한 이미지 검색 시스템 구축

SHORE 하부저장 구조는 C++언어로 구현된 시스템으로 웹과의 연동시 웹 서버에서 CGI 형태로 SHORE 서버를 호출한다. 하지만 CGI는 서버의 부담을 증가시키며 플랫폼 독립적인 환경에 부적합하다. 위와 같은 기존 시스템의 단점을 해결하기 위한 방안으로 SHORE 상에 다음과 같은 특성을 지닌 미들웨어를 구축한다.

① SHORE 하부저장 구조를 이용하는 자바 응용프로그램에 자바의 장점인 플랫폼 독립성을 유지한다.
② 일반 응용프로그램 뿐만 아니라 애플릿에서도 SHORE를 이용할 수 있다.

5.1 시스템 구성

자바 기술을 이용한 SHORE/JAVA의 구조는 크게 C/C++ 응용 프로그램에서 접근을 허용하기 위한 HDServer/C를 두고 Java Application이나 Java Applet에서 SHORE/JAVA를 접근하기 위해 JNI 기법을 이용한 HDServer/JAVA를 제공하는 형태이다. 클라이언트 입장에서는 C/C++/JAVA등 여러 가지의 언어를 이용하여 프로그램을 작성하여 SHORE 라이브러리에서 제공하는 하부저장 시스템의 여러 가지 특성을 이용할 수 있다.

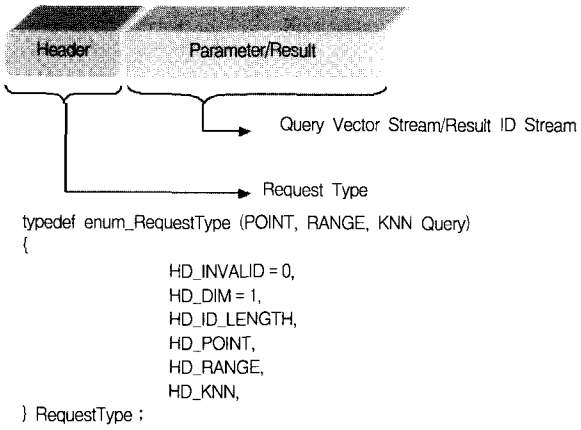


(그림 10) SHORE/JAVA의 구조

구현하는 시스템은 고차원 색인 구조의 검색 API에 한정하기 때문에 먼저 검색 API의 특성을 분석한다. 이에 대한 검색 API의 형태는 다음과 같다.

- **hd_retrieve_point**
  - parameter : 질의벡터
  - return : 검색된 객체 IDs
- **hd_retrieve_range**
  - parameter : 질의벡터, Radius
  - return : 검색된 객체 IDs
- **hd_retrieve_knn**
  - paramter : 질의벡터, KNum
  - return : 검색된 객체 IDs

HDServer/C는 소켓 서버이기 때문에 HDClient/C와의 전달되는 데이타는 스트림형태로 취하게 된다. 따라서 클라이언트에서 전달되는 요청(Request)나 서버로부터 전달되는 결과(Result)를 팩킹(packing)과 언팩킹(unpacking) 작업을 요구하게 되고 이의 정확한 전달을 위해서는 서버와 클라이언트의 정의된 프로토콜을 요구하게 된다. 이의 정의된 형태는 (그림 11)과 같다. 데이타 스트림의 Header 부분은 클라이언트에서 요구하는 Request Type을 의미하게 되는데 포인트질의, 범위질의 그리고 k-NN 질의 등을 갖는다. 데이타 스트림의 Parameter/Result 부분은 요청시에는 질의벡터를 갖고 결과 반환시에는 검색된 객체 ID를 갖게 된다.



(그림 11) SHORE/JAVA의 데이터 스트림 구조

다음은 정의된 프로토콜에 의한 데이터 스트림의 예를 보이고 있다.

- ***Point* 질의 예(10차원)**

C → S : 3 # 0.1 # 0.2 # 0.3 # 0.4 # 0.5 # 0.6 # 0.7 # 0.8 # 0.9 # 1.0 # $

S → C : 3 # 5 # img001.gif # img002.gif # img003.gif # img004.gif # img005 # $

• **Range 질의 예(10차원)**

C → S : 4 # 0.1 # 0.2 # 0.3 # 0.4 # 0.5 # 0.6 # 0.7 # 0.8 # 0.9 # 1.0 # 0.1 # $

S → C : 4 # 5 # img001.gif # img002.gif # img003.gif # img004.gif # img005 # $

• **KNN 질의 예(10차원)**

C → S : 5 # 0.1 # 0.2 # 0.3 # 0.4 # 0.5 # 0.6 # 0.7 # 0.8 # 0.9 # 1.0 # 5 # $

S → C : 5 # 5 # img001.gif # img002.gif # img003.gif # img004.gif # img005 # $

한 예로 KNN질의의 경우 K값은 5이고 질의 벡터는 (0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0)인 경우의 클라이언트와 서버 사이의 데이터 스트림의 구조를 보인 것이다. 스트림의 맨 처음 5는 Request Type의 HD_KNN을 가리키며 각각의 벡터는 delimiter(#)으로 구분하였다. 마지막의 5는 k값을 가리키며 스트림의 마지막은 delimiter($)을 사용하였다.

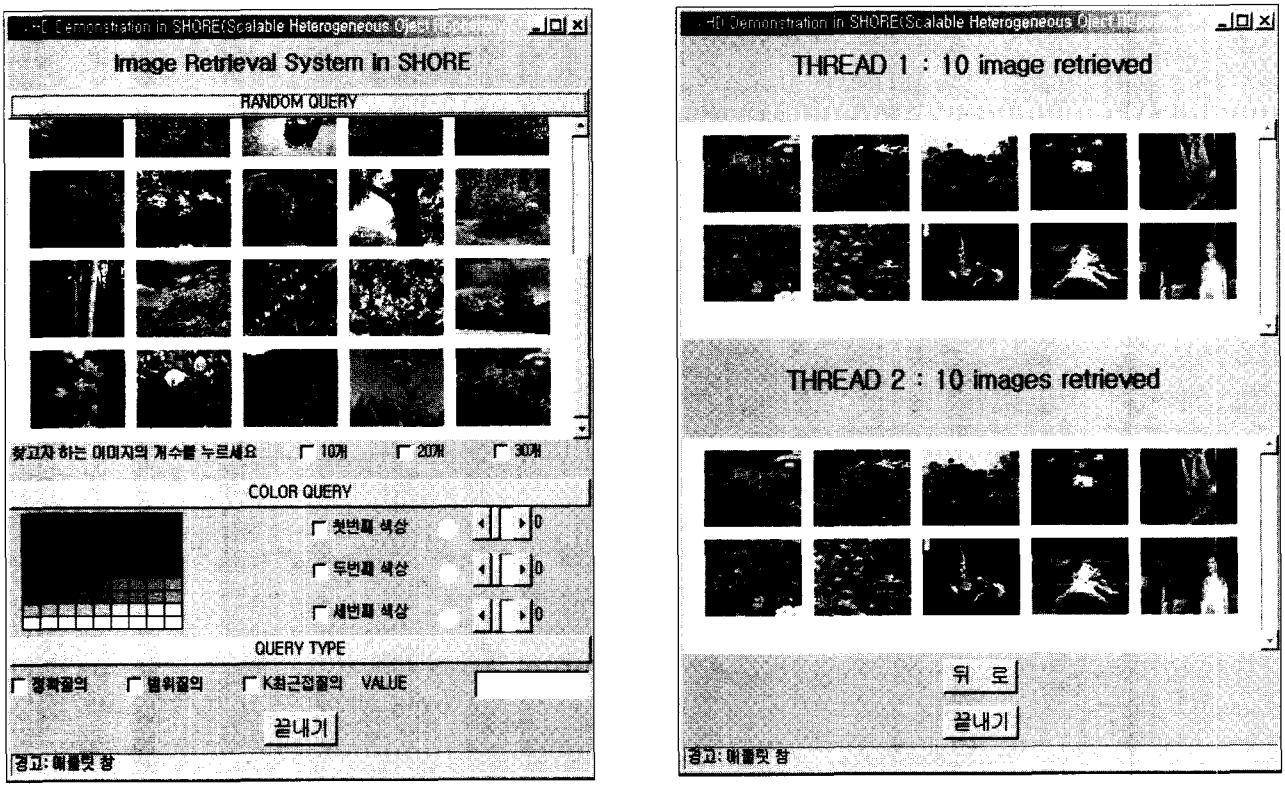
5.2 질의 유형 및 인터페이스

사용자 인터페이스에서 처리할 수 있는 질의의 유형을 분류하면 랜덤이미지 질의, 칼라맵을 이용한 질의로 나눌 수 있으며, 랜덤 이미지 질의는 애플릿에 초기화 될 때 SHORE 서버에 접속하여 랜덤이미지를 화면에 디스플레이 한다. 이 질의를 통해 원하는 이미지와 색상이 유사한 결과 이미지를 찾을 수 있다. 칼라맵을 이용한 질의는 원하는 이미지의 색상을 최대 3가지를 지정하여 이를 비율로 지정할 수 있으며 이를 통해 좀더 세밀한 색상에 대한 질의를 가능하게 한다. 이러한 두 가지 형태의 색상 질의를 기반으로 원하는 색상을 선택하고 이와 유사한 결과 이미지를 찾는 유사성에 기반한 질의형태는 크게 포인트 질의, 범위질의, k-최근접 질의를 지원한다.

(그림 12)에서 찾고자 하는 이미지의 개수를 체크하고 RANDOM QUERY 부분의 임의의 이미지를 클릭하면 SHORE 서버에 접속하여 질의 유형에 맞는 검색 결과를 동시에 쓰레드 2개가 작업을 하여 결과를 보여주는 것이다. 또한 COLOR MAP을 이용하여 색상을 선택하고 각각의 색상의 정도를 스크롤 바로 조절하면 역시 검색 결과를 얻어볼 수 있다. 이러한 검색 유형은 좀더 세밀한 검색에 사용되어지며 특징 벡터의 색상을 추출한 데이터 이외에 다른 SHAPE 같은 것으로의 확장도 가능하다.

6. 결 론

최근 들어 멀티미디어 데이터베이스, 데이터 마이닝 분야에서 널리 사용되는 멀티미디어 데이터는 기존의 텍스트 정



(그림 12) 이미지 검색 시스템의 구현 인터페이스

보 뿐만 아니라 이미지, 비디오 등과 같은 다양한 형태의 자료가 복합적으로 구성되어 있다. 이러한 멀티미디어 데이터에 대한 효율적인 검색을 위해 많은 다차원 및 고차원 색인 구조에 대한 연구가 이루어졌으나 이는 검색의 효율에 대한 향상에 초점을 맞추어 왔다. 이에 본 논문에서는 기존의 제시된 특징 벡터의 요약 정보인 시그니쳐 기반의 고차원 색인 구조를 실제 응용시스템에 적용하기 위해 동시성 제어기법을 설계 및 구현하여 SHORE 하부저장 시스템과의 통합을 수행하였다. 확장된 SHORE 하부저장 시스템은 기존의 텍스트 기반의 전문정보 검색 뿐만 아니라 멀티미디어 데이터에 대한 내용 기반 검색을 가능하게 한다. 아울러 레코드 레벨의 동시성 제어기법이 가능하며, 시그니쳐 정보에 대한 페이지 관리를 가능하게 하였다. 확장된 SHORE 하부저장 시스템은 또한 멀티미디어 응용 분야에 적합한 응용시스템으로의 확장성을 용이하게 하기 위해 자바 언어를 사용하여 플랫폼 독립성을 유지하면서 응용프로그램과 애플릿에서 모두 SHORE 서버를 사용할 수 있도록 하는 SHORE 패키지 모델을 제시하였다. 또한 쓰레드 별로 다수 사용자 환경을 재연하여 고차원 색인 구조의 대표적인 내용기반 질의의 형태인 포인트 질의, 범위질의, k-최근접 질의에 대해서 차원에 따른 성능평가를 보였다.

향후 연구에서는 필터링에 기반한 고차원 색인 구조의 SHORE 하부저장 시스템에서의 회복에 대한 연구가 필요하다.

참 고 문 헌

[1] Berchtold S., Keim D., and Kriegel H.-P., "The X-tree : An Index Structure for High-Dimensional Data," 22nd Conf. on Very Large Databases, pp.28-39, 1996.

[2] K. I. Lin, H. Jagadish, and C. Faloutsos, "The TV-tree : An Index Structure for High Dimensional Data," VLDB Journal, Vol.3, pp.517-542, 1994.

[3] Beckmann N., Kriegel H. P, Schneider R., and Seeger B., "The R*-tree : An Efficient and Robust Access Method for Points and Rectangles," Proc. ACM SIGMOD Int. Conf. on Management of Data, pp.322-331, 1990.

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

[5] Berchtold S., Bohm C., Keim D., Kriegel H. P, "A Cost Model fro Nearest Neighbor Search in High-Dimensional Data Space," ACM PODS Symposium on Principles of Databases Systems, Tucson, Arizona, 1997.

[6] 한성근, 이용주, 장재우, 김현진, "효율적인 고차원 데이터색인을 위한 셀-기반 필터링 기법", Proceedings of Korean Database Conference, pp.26-35, 2000.

[7] Sung-Geun Han and Jae Woo Chang, "A New High-Dimensional Index Structure Using a Cell-Based Filtering Technique," Proc. of International Conf. on ADBIS-DASFAA, pp.79-92, 2000.

[8] Kwang Taek Song, Hwa-Jin Nam, and Jae-Woo Chang, "A cell-based index structure for similarity search in high-dimensional feature spaces," Proc. of ACM SAC, pp.264-268, 2001.

[9] Carey, M., DeWitt, D., Naughton, J., Solomon, M., et. al, "Shoring Up Persistent Applications," Proc. of International Conf. on ACM SIGMOD, pp.383-394, 1994.

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

[11] 정재욱, 장재우, "멀티미디어 응용을 위한 SHORE 하부저장 시스템의 확장", 한국정보과학회 가을학술발표논문집(1), pp. 6-8, 1999.

[12] M. Kornacker, C. Mohan and J. M. Hellerstein, "Concurrency and Recovery in Generalized Search Trees," Proc. of International Conf. on ACM SIGMOD, pp.62-72, 1997.

이 용 주

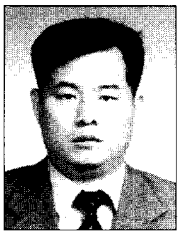
e-mail : yongju@etri.re.kr

1999년 전북대학교 컴퓨터공학과(공학사)

2001년 전북대학교 대학원 컴퓨터공학과 (공학석사)

2001년~현재 한국전자통신연구원 컴퓨터시스템연구부 연구원

관심분야 : 하부저장시스템, 네트워크스토리지, 분산파일시스템, 멀티미디어 서버



장 재 우

e-mail : jwchang@dblab.chonbuk.ac.kr

1984년 서울대학교 전자계산기공학과 (공학사)

1986년 한국과학기술원 전산학과(공학석사)

1991년 한국과학기술원 전산학과(공학박사)

1996년~1997년 Univ. of Minnesota, Visiting Scholar.

1991년~현재 전북대학교 전자정보공학부(컴퓨터공학과)

관심분야 : 멀티미디어 데이터베이스, 멀티미디어 정보검색, 클러스터 DBMS, 데이터베이스 하부저장구조, 고차원 색인기법

김 학 영

e-mail : h0kim@etri.re.kr

1983년 경북대학교 전자공학과 전자계산 전공(공학사)

1985년 경북대학교 대학원 전자공학과 전자계산전공(공학석사)

2001년 충남대학교 대학원 컴퓨터공학과 (박사)

1988년 현재 한국전자통신연구원 컴퓨터시스템연구부 책임연구원

관심분야 : 인터넷 서버, 컨텐츠 스트리밍, 컨텐츠 분배, 시스템 소프트웨어, 분산파일시스템, GRID

김 명 준

e-mail : joonkim@etri.re.kr

1978년 서울대학교 계산통계학과 학사

1980년 한국과학기술원 전산학과 이학석사

1986년 프랑스 Nancy 제1대학교 응용수학 및 전산학과 이학박사

1980년~1986년 아주대학교 종합연구소 연구원

1981년~1986년 프랑스 Nancy 전산학 연구소(CRIN) 연구원

1993년 프랑스 Univ. of Nice Sophia-Antipolis 방문교수

1986년~현재 한국전자통신연구원 컴퓨터소프트웨어 연구소 책임연구원

관심분야 : 데이터베이스, 분산시스템, 인터넷서비스