□ 기술해설 □

멀티미디어 데이타베이스 시스템에서의 대형 멀티미디어 객체 관리 기법

강원대학교 김상욱*
한국과학기술원 이영구** · 김원영*** · 장지웅** · 황규영***

● 목 차 ●

1. 서 론

최근 컴퓨터 기술의 눈부신 발전으로 인하여 텍스트, 그래픽스, 이미지, 애니메이션, 오디오, 비디오 등의 새로운 멀티미디어 데이타들을 컴퓨터 시스템내에서 다룰 수 있게 되었다. 이러한 멀티미디어 데이타의 가장 큰 특성은 저장을 위하여 매우 큰 공간을 요구한다는 것이다. 예를 들어, 400dpi 비트 맵 이미지의 경우 약 2 메가 바이트, 5 분 가량의 오디오의 경우 약 50 메가 바이트, 그리고 초당 12 프레임으로 구성된 5 분 가량의 고품질 비디오인 경우 약 850 메가 바이트의 저장 공간이 각각 요구된다 [14, 16]. 이것은 대략 수 바이트 크기의 저장 공간을 요구하는 기존의 수치, 문자 데이타와 비교할 수 없을 정도의 엄청난 차이를 보이는 것이다.

은행 업무, 일반 사무 업무 등 기존의 전통적인 데이타베이스 응용 분야에서는 주로 수치나 문자 위주의 데이타만을 다루었으므로 기존의 상용 데이타베이스 관리 시스템(DataBase Management System : DBMS)의 개발자들은 각 데이타가 하나의 디스크 페이지내에 저장 가능하다는 가정하에 DBMS를 개발하여 왔다. 그러나 대형 멀티미디어 데이타의 새로운 등장으로 이러한 가정은 더 이상 실제 상황과 맞지 않는 것이 되었다[13]. 이와 같이 크기가 커서 하나의 페이지내에 저장이 불가능한 데이타를 대형 객체(large object)라 정의한다[1, 2]. 따라서 DBMS가 멀티미디어 응용을 지원하기 위해서는 이러한 대형 객체를 효과적으로 관리하기 위한 별도의 기법이 요구된다. 최근 이러한 요구에 부응하기 위한 몇몇 연구가 고급 응용 분야(advanced applications)를 위한 DBMS 및 저장 시스템 개발을 통하여 수행된 바 있다. 본 논문에서는 이러한 대형 객체 관리에 관한 최근의 연구 결과에 관하여 고찰하고자 한다.

본 논문에서 다루고자 하는 내용은 다음과 같다. 먼저, 제 2장에서는 대형 객체 관리 기법의 설계시 고려 사항에 관하여 논의한다. 제 3

*정 회 원
**학생회원
***종신회원

장에서는 최근 많은 상용 DBMS에서 지원하는 기법과 WiSS[3], Exodus[6], Starburst [10], EOS[1] 등 최근에 개발된 고급 저장 시스템에서 지원하는 각각의 기법에 대하여 기본 자료 구조, 각종 연산 알고리즘, 성능상의 특성 등을 비교 및 분석한다. 제 4장에서는 대형 객체 관리 기법에 관한 국내의 연구 동향에 대하여 소개한다. 끝으로 제 5장에서는 결론을 내리고, 본 주제와 관련된 향후의 연구 이슈들을 제시한다.

2. 대형 객체 관리 기법의 설계시 고려 사항

본 장에서는 대형 객체 관리 기법의 설계시 효율성을 위하여 고려해야 할 사항에 관하여 논의한다. 이들은 제 3장에서 기존의 기법들의 성능을 평가하는 기준으로서 사용된다.

(1) 무제한 크기의 지원 : 멀티미디어 데이타는 수 메가에서 수 기가에 이르는 상당한 크기의 저장 공간을 요구한다. 또한, 하드웨어 및 소프트웨어 기술이 급속도로 발전함에 따라 더욱 많은 저장 공간을 요구하는 고품질의 데이타 처리가 가능해지고 있다. 따라서 대형 객체 관리 기법은 이러한 추세를 감안하여 물리적인 저장 공간이 충족되는 범위에서는 무제한 크기의 객체를 지원할 수 있어야 한다.

(2) 저장 공간 이용률 : 저장 공간 이용률(storage utilization)이란 대형 객체를 위하여 할당된 페이지들의 총 바이트 수 중 실제로 데이타 저장을 위해 사용하는 바이트 수의 비율이 얼마인가를 의미한다. 저장 공간 이용률이 낮은 기법의 경우, 할당되어야 할 페이지 수가 많아지므로 저장 공간의 오버헤드가 커지며, 연속 액세스시 발생되는 페이지 액세스 수가 많아지므로 처리 성능이 저하된다. 따라서 저장 공간 이용률은 가능한 100 %에 가깝도록 유지되어야 한다.

(3) 연속 액세스시의 성능 : 연속 액세스(sequential access)란 하나의 대형 객체를 처음부터 끝까지 순차적으로 액세스하는 연산을 의미한다. 비디오를 처음부터 끝까지 상영하는 것이 연속 액세스의 대표적인 예가 된다. 만일, 이러한 연속 액세스시의 성능이 낮다면 매우 부자연스러운 영상이 될 것이다. 따라서 성능 극대화를 위하여 페이지들을 디스크에서 개별적으로 할당하지 않고, 연속적인 페이지들의 집합 단위로 할당함으로써 연속 액세스시 디스크 헤드의 이동 시간(seek time)을 최소화 할 수 있는 방안이 고려 되어야 한다.

(4) 랜덤 액세스시의 성능 : 랜덤 액세스(random access)란 대형 객체의 임의의 바이트를 갖는 페이지의 위치를 찾는 연산을 의미한다. 이것은 특정 위치로부터 일정한 크기의 데이타를 연속적으로 액세스하는 부분 액세스(partial access) 연산의 일부로서 사용된다[1]. 예를 들어, 클래식이 수록된 오디오의 일부만을 듣기 위해서는 이러한 부분 액세스가 유용하다. 특히, 데이타의 뒷부분에 수록된 내용을 찾고자 하는 경우, 랜덤 액세스시의 성능은 응답 시간에 큰 영향을 미친다.

(5) 부분 변경시의 성능 : 부분 변경(partial update)이란 대형 객체의 임의의 위치에 일정 크기의 데이타를 삽입하거나, 삭제하는 연산을 의미한다[2]. 예를 들어, 비디오의 편집시에는 중간에 프레임을 삽입 및 삭제할 수 있다. 이와 같이 변경이 객체의 극히 일부에서만 발생하는 경우, 나머지 부분에는 영향을 미치지 않고 해당 부분만 변경할 수 있어야 한다.

---

1) 부분 액세스의 성능은 액세스의 대상이 되는 데이타 부분의 크기에 영향을 많이 받는다. 이것은 연속 액세스의 성능에서 유사한 형태로 다루어지므로 본 논문에서는 랜덤 액세스시의 성능만을 설계시의 고려 대상으로 선정한다.

2) 임의의 위치로부터 일정한 크기의 기존 데이타를 새로운 데이터로 대치시키는 갱신 연산(overwrite)도 이 범주에 속하나, 이것은 부분 액세스와 거의 같은 성능 특성을 가지므로 본 논문에서는 별도로 언급하지 않는다.

3. 대형 객체 관리를 위한 기존의 접근 기법들

본 장에서는 대형 객체를 관리하기 위한 기존의 기법들에 관하여 논의한다. 즉, 상용 DBMS와 WiSS[3], Exodus[6], Starburst [10], EOS[1] 등 고급 저장 시스템에서 제공하는 각 기법에 대하여 자료 구조, 연산 알고리즘, 각 알고리즘의 성능을 제시한다[3]

3.1 연결 리스트를 이용한 BLOB 관리 기법

연결 리스트를 이용한 BLOB(binary large object) 관리 기법은 가장 단순한 기법이며, 이러한 단순성으로 인하여 Informix[5], Sybase [15] 등 상용 DBMS에서 주로 채택하고 있다.

(1) 자료 구조

BLOB은 페이지들을 연결 리스트(linked list)의 형태로 연결해 놓은 단순한 구조이다 (그림 1 참조). 데이타는 첫 페이지로부터 빈 공간 없이 차례로 저장되며, 마지막 페이지에서는 데이타를 저장하고 남은 뒷부분이 빈 공간으로 남을 수 있다. 페이지의 크기는 일반 페이지와 같도록 정의함으로써 디스크 공간을 공유할 수도 있으나, 응용의 특성에 따라 별도의 디스크 공간을 할당함으로써 최적화된 크기의 페이지를 사용할 수도 있다. 그림 1은 페이지의 크기가 100 바이트라는 가정하에 943 바이트 크기를 갖는 데이타가 BLOB 형태로서 저장된 것을 나타낸 것이다.[4] 검은 부분은 데이타가 저장된 부분을 의미하며, 숫자는 저장된 바이트 수를 나타낸다.

그림 1 연결 리스트를 이용한 BLOB 관리 기법

3) 어떤 기법에서는 성능상의 이유로 특정 연산을 허용하지 않는 경우가 있으나, 본 논문에서는 우선 해당 기법의 자료 구조를 기반으로 그 연산을 지원하기 위한 알고리즘을 제시하고, 그 성능 저하의 발생 원인에 대해서도 언급한다.

(2) 연산 알고리즘

A. 생성 : 페이지들을 할당하여 이들을 연결 리스트의 형태로 연결시키고, 첫 페이지로부터 끝 페이지까지 데이타를 차례로 채워 넣는다.

B. 연속 액세스 : BLOB의 첫 페이지를 찾고, 이후의 연결 리스트상에 나타난 페이지들을 차례로 액세스하여 각 페이지내에 있는 데이타를 읽어들인다.

C. 랜덤 액세스 : 연속 액세스에서와 같이 BLOB의 첫 페이지 이후의 연결 리스트상에 나타난 페이지들을 차례로 액세스하여 해당 바이트를 가지는 페이지를 찾아낸다.

D. 삽입 : 랜덤 액세스를 이용하여 새로운 데이타의 첫 바이트가 삽입될 위치 P를 가지는 페이지를 찾아낸다. P로부터 데이타를 차례로 삽입하게 되는데, 이 과정에서 새로운 페이지들이 할당될 수 있으며, 이 페이지들도 연결 리스트 형태로 연결된다. 삽입전 P이후에 존재하던 모든 데이타는 삽입된 새로운 데이타의 마지막 바이트 이후로 밀리게 된다.

E. 삭제 : 삭제될 데이타의 첫 바이트가 존재하는 위치 P를 가지는 페이지를 찾아낸다. P로부터 연결 리스트상에 나타나는 페이지들을 따라가며 삭제를 원하는 데이타의 마지막 바이트 이후에 존재하는 데이타를 P 직후로 끌어 당김으로써 해당 데이타를 삭제시킨다. 이 과정 중 완전히 비게 된 페이지들은 연결 리스트상에서 제거되어 반환된다.

(3) 성능 특성

A. 무제한 크기의 지원 : 얼마든지 많은 페이지들을 리스트에 연결하여 사용할 수 있으므로 무제한의 크기를 지원한다.

B. 저장 공간 이용률 : 마지막 페이지 이외의

4) 앞으로 사용하는 모든 예제에서 페이지내의 제어 정보의 크기는 설명의 편의를 위하여 무시한다. 또한, 대형 객체의 위치(여기서는 BLOB의 첫 페이지 주소)와 실제 데이타 크기는 대형 객체와는 별도로 관리된다고 가정한다.

모든 BLOB 페이지들은 빈 공간 없이 데이타로 채워지므로 대부분의 경우 100%에 가깝게 나타난다.

C. 연속 액세스 : 100%에 가까운 높은 저장 공간 이용률은 연속 액세스시의 성능을 높이는 긍정적인 요소로서 작용한다. 그러나 각 페이지가 개별적으로 할당되므로 페이지 액세스마다 디스크 헤드의 이동이 발생하게 되는데, 이것이 성능 저하의 원인이 된다.[5]

D. 랜덤 액세스 : 특정 바이트를 찾기 위해서는 항상 첫 페이지로부터 페이지들을 차례로 액세스 해야 하므로 해당 데이타의 크기와 원하는 바이트의 위치에 따라 성능이 크게 좌우된다.

E. 부분 변경 : 삽입 및 삭제시, 삽입 혹은 삭제될 데이타 이후에 나타나는 모든 데이타들을 이동시켜야 한다. 이것은 상당한 오버헤드이며, 이러한 이유로 인하여 BLOB을 사용하는 상용 DBMS에서는 부분 변경 연산을 지원하지 않고 있다.

3.2 WiSS에서의 긴 데이타 아이템 관리 기법

WiSS(Wisconsin Storage System)[3]는 Wisconsin 대학에서 DBMS 구축을 위하여 개발한 실험적인 저장 컴포넌트이며, 대형 객체의 저장을 위하여 긴 데이타 아이템(long data item : LDI)을 제공한다.

(1) 자료구조

LDI는 슬라이스(slice)를 가지는 페이지들의 집합과 디렉토리로 구성된다(그림 2 참조). 각 슬라이스는 실제 데이타를 가지는 일종의 가변 길이 레코드로서 하나의 페이지내에서 관리된다. 디렉토리는 각 슬라이스의 크기와 위치의 쌍으로 구성된 엔트리들의 리스트이며, 슬라이스와 마찬가지로 최대 한 페이지의 크기를 갖는다. 그림 2는 페이지의 크기가 100 바이트라는 가정하에 943 바이트 크기를 갖는 데이타가 LDI로 저장된 예를 나타낸 것이다.

5) 페이지의 크기를 상향 조정함으로써 어느 정도의 성능 향상은 기대할 수 있다

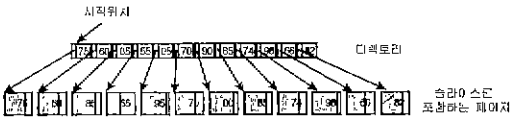


그림 2 WiSS에서의 긴 데이타 아이템 관리 기법

(2) 연산 알고리즘

A. 생성 : 디렉토리를 하나 생성한 후, D 항의 삽입 연산을 통하여 현존하는 데이타의 마지막 부분에 새로운 데이타를 계속 삽입함으로써 LDI를 단계적으로 생성한다.

B. 연속 액세스 : 디렉토리의 각 엔트리가 가리키는 슬라이스들을 차례로 액세스하여 각 슬라이스내에 있는 데이타를 읽어들인다.

C. 랜덤 액세스 : 디렉토리 엔트리들의 길이 필드를 이용하여 원하는 바이트를 가지는 슬라이스와 대응되는 페이지를 찾아낸다.

D. 삽입 : 삽입될 위치 P를 가지는 슬라이스를 랜덤 액세스를 이용하여 찾아낸다. P로부터 데이타를 차례로 삽입한다. 이 과정에서 새로운 슬라이스들이 할당될 수 있다. 끝으로, 슬라이스의 길이 확장으로 인한 기존 엔트리의 길이 필드 변화와 슬라이스의 할당으로 인한 새로운 엔트리의 추가 등을 디렉토리에 반영시킨다.

E. 삭제 : 삭제될 첫 바이트 P를 가지는 슬라이스를 찾아, P로부터 데이타를 차례로 삭제한다. 이 과정에서 슬라이스들이 반환될 수 있으며, 삭제된 데이타의 첫 부분과 끝 부분을 가졌던 두 슬라이스가 한 페이지보다 작은 경우에는 하나로 병합된다. 끝으로, 슬라이스의 길이 축소로 인한 엔트리의 길이 필드 변화, 슬라이스의 반환으로 인한 엔트리의 삭제, 그리고 슬라이스 병합으로 인한 대응되는 해당 엔트리의 병합 등을 디렉토리에 반영시킨다.

(3) 성능 특성

A. 무제한 크기의 지원 : 디렉토리와 슬라이스의 최대 크기가 모두 한 페이지로 고정

되어 있기 때문에 제한된 크기의 데이타만을 저장할 수 있다. 4 킬로 바이트의 페이지를 가정할 때, LDI가 지원할 수 있는 최대 크기는 약 1.6 메가 바이트이다[3]. 이것은 멀티미디어 데이타 저장시의 큰 제한점으로 지적된다.

B. 저장 공간 이용률 : 삽입과 삭제 연산을 통하여 각 페이지내에서 슬라이스가 차지하고 있지 않는 부분은 빈 공간으로 남게 된다. 삽입 및 삭제 알고리즘은 최소 50 % 이상이 되도록 보장하고 있으나 설계 요건에서의 목표인 100%와는 상당한 차이가 있다.[6]

C. 연속 액세스 : 저장 공간 이용률이 낮으므로 액세스 해야 할 페이지 수가 많고, 각 페이지가 개별적으로 할당되므로 각 페이지 액세스마다 디스크 헤드의 이동이 발생된다. 따라서 연속 액세스시의 성능이 떨어진다.[7]

D. 랜덤 액세스 : 디렉토리만을 액세스 함으로써 원하는 데이타가 저장된 슬라이스의 위치를 직접 알아낼 수 있으므로 성능이 매우 뛰어나다.

E. 부분 변경 : 삽입시 디렉토리 페이지와 삽입될 데이타를 가지는 페이지들만이 액세스되며, 삭제시 디렉토리 페이지와 삭제될 데이타를 가지는 페이지들 중 맨 앞과 뒤의 두 페이지만이 액세스된다. 즉, 변경되지 않는 다른 데이타의 이동은 발생되지 않는다.

3.3 Exodus에서의 대형 객체 관리 기법

Exodus[6]는 Wisconsin 대학에서 개발한 확장형 DBMS이다. 하부 구조인 저장 관리자는 대형 데이타 관리를 위하여 대형 객체(large object : LO)[2]를 제공한다.

(1) 자료 구조

LO는 리프 페이지(leaf page)와 내부 페이지(internal page)로 구성되는 대형 객체 트리를 사용한다(그림 3 참조). 이것은 디렉토리를 다단계로 확장함으로써 WiSS LDI의 크기 제한의 문제점을 극복하고자 한 것이다. 리프 페이지는 데이타 저장을 위하여 사용되고, 내부 페이지는(count, position)의 쌍으로 구성되는 엔트리들의 리스트를 가진다. Position은 해당 엔트리와 대응되는 다음 단계의 서브 트리의 루트 페이지를 가리키며, count는 그 서브 트리내에 저장된 마지막 바이트가 현재 내부 페이지를 루트로 하는 서브 트리내에서 몇번째 바이트에 해당되는가를 나타낸다. 그림 3은 리프 페이지의 크기가 100 바이트이고, 내부 페이지는 최대 세 개의 엔트리들을 가진다는 가정하에 943 바이트 크기의 데이타가 LO로 저장된 예를 나타낸 것이다.

(2) 연산 알고리즘

A. 생성 : 루트 페이지를 생성한 후, 첨가 연산(append)을 통하여 현존하는 데이타의 마지막 부분에 데이타를 반복적으로 삽입함으로써 LO를 단계적으로 생성한다. 별도의 첨가 연산을 사용하는 이유는 D 항의 삽입 연산과는 달리 가장 오른쪽에 나타나는 두 페이지를 제외한 모든 리프 페이지들을 빈 공간 없이 데이타로 채울 수 있기 때문이다.

B. 연속 액세스 : 루트 페이지에서 시작하여

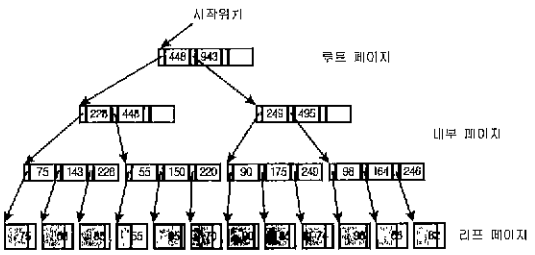


그림 3 Exodus에서의 대형 객체 관리 기법

6) WiSS에서는 슬라이스를 위한 각 페이지내의 빈 공간을 인접한 슬라이스내의 데이타로 채우고, 불필요한 페이지를 반환하는 함수를 제공한다[3]. 이 함수를 수행하면 BLOB 관리 기법에서와 같이 저장 공간 이용률을 100%에 가깝게 유지시킬 수 있으나 LDI 전체 구조를 재구성해야 하므로 수행 시간의 오버헤드는 매우 크다.

7) 다음절에서 언급할 Exodus에서와 같이 페이지의 크기를 상향 조정함으로써 어느 정도의 성능 개선을 기대할 수 있으나, WiSS에서 이러한 방법을 제공하고 있지는 않다.

LO내의 모든 페이지들을 깊이 우선 탐색(depth first search) 방식으로 액세스함으로써 리프 페이지내의 데이타를 차례로 읽어들인다.

C. 랜덤 액세스 : 루트 페이지내의 엔트리들의 count 정보를 이용하여 원하는 바이트를 가진 서브 트리와 대응되는 엔트리를 찾아낸다. 그 엔트리가 가리키는 서브 트리의 루트 페이지에 대하여 같은 방식을 재귀적으로 적용함으로써 최종적으로 원하는 바이트가 저장된 리프 페이지를 찾아낸다.

D. 삽입 : 데이타가 삽입될 리프 페이지 P를 랜덤 액세스를 이용하여 찾아낸다. P내에 공간이 충분한 경우에는 데이타를 삽입하고, 루트 페이지에서 P까지의 경로를 구성하는 각 엔트리의 count 필드에 데이타의 삽입을 반영시킨다. 그렇지 않은 경우에는 리프 페이지들을 할당 받은 뒤 P내에 있던 데이타와 삽입될 데이타를 P와 새로 할당된 리프 페이지내에 균등하게 저장시킨다. 그리고 새로 할당된 각 리프 페이지들에 대하여 위치와 저장된 데이타 크기의 쌍을 바로 상위 내부 페이지내에 삽입한다. 또한, 루트 페이지에서 P까지의 경로를 구성하는 각 엔트리의 count 필드에 이러한 변경을 반영시킨다. 만일, 내부 페이지내에 충분한 공간이 없는 경우에는 이러한 작업을 재귀적으로 반복하여 처리한다.

E. 삭제 : 데이타를 삭제하는 작업은 서브 트리의 삭제를 유발할 수 있다. 즉, 루트 페이지로부터 그 데이타의 첫 바이트를 가지는 리프 페이지까지의 경로와 마지막 바이트를 가지는 리프 페이지까지의 경로 사이에 존재하는 서브 트리가 삭제되는 것이다. 삭제 연산은 다음의 두 단계로 진행된다. 첫 번째 단계에서는 두 경로 사이에 존재하는 서브 트리를 제거하면서 이러한 변화를 두 경로상에 나타나는 엔트리들의 count 필드에 반영시킨다. 이 때, 두 경로상의 페이지들은 내부 데이타의 삭제로 인하여 50% 미만의 점유율을 가질 수 있다. 두 번째 단계에서는 이들에 대하여 이웃하는 페이지와의 병합(merging) 혹은 재분배(redistribution)를 통하여 저장 공간 이용률을 높여주는 작업을 수행한다.

(3) 성능 특성

무제한의 크기 지원을 제외한 나머지 성능 평가 항목에 있어서 WiSS의 LDI와 유사한 성능을 보인다.

A. 무제한 크기의 지원 : 트리의 깊이를 증가시킴으로써 실제 데이타를 저장하는 리프 페이지들을 얼마든지 많이 가질 수 있으므로 무제한의 크기를 지원할 수 있다.

B. 저장 공간 이용률 : 최초의 생성시에는 100%에 가까운 값을 가지지만, 이후에 발생되는 삽입과 삭제 연산에 의하여 리프 페이지내에서의 빈 공간의 생성은 불가피하므로 LDI와 같이 50%에서 100% 사이의 값을 갖는다.

C. 연속 액세스 : 낮은 저장 공간 이용률, 내부 및 리프 페이지의 개별적인 할당, 그리고, 깊이 우선 탐색으로 인한 내부 페이지의 추가 액세스 등의 성능 저하 요소들을 갖는다. 응용 분야에 따라 리프 페이지의 크기를 상향 조정함으로써 어느 정도의 성능 개선은 가능하다.

D. 랜덤 액세스 : 루트 페이지에서 리프 페이지까지의 경로상에 나타나는 페이지들만을 액세스 함으로써 전체 데이타의 크기와 원하는 바이트 위치에 영향을 받지 않는다.

E. 부분 변경 : 삽입 및 삭제시 해당 연산과 연관된 데이타를 가지는 서브 트리내의 페이지들만이 액세스되며, 그 이외의 불필요한 데이타의 이동은 발생되지 않는다.

3.4 Starbrust에서의 긴 필드 관리 기법

Starburst[10]는 IBM Almaden 연구소에서 새로운 DBMS 기술을 시험하기 위하여 개발한 실험용 DBMS이며, 대형 객체를 저장하기 위하여 긴 필드(long field : LF)[9]를 제공한다.

(1) 자료 구조

LF는 버디 세그먼트(buddy segment)들의 집합과 LF 기술자(long field descriptor)로 구성된다(그림 4 참조). 버디 세그먼트(간략히, 세그먼트)는 데이타의 저장 단위로서 물리적으로 인접한 페이지들의 집합이다. 각 세그먼트를 구성하는 페이지의 수는 이진 버디 시스템(binary buddy system)의 할당 방식에 따라 2^n개로 한정된다.

LF 기술자는 세그먼트들을 관리하기 위한 정보로서 총 세그먼트 수, 첫 세그먼트내의 페이지 수, 마지막 세그먼트내의 페이지 수 등으로 구성되는 제어 데이타와 각 세그먼트에 대한 포인터의 리스트로 구성되는 디렉토리로 이루어진다. 디렉토리의 i번째 엔트리가 가리키는 세그먼트의 크기는 (i-1)번째 엔트리가 가리키는 세그먼트 크기의 두배이다. 단, (i-1)번째 엔트리가 가리키는 세그먼트가 미리 정한 최대 크기라면, 이후에 나타나는 세그먼트는 최대 크기로 고정된다. 마지막 세그먼트내에서 실제 사용하지 않는 페이지들은 반환함으로써 저장 공간의 낭비를 방지한다. 데이타는 첫 세그먼트로부터 마지막 세그먼트까지 빈 공간 없이 차례로 저장된다. 그림 4는 페이지의 크기가 100 바이트라는 가정하에 943 바이트 크기를 갖는 데이타가 LF로 저장된 예를 나타낸 것이다.

(2) 연산 알고리즘

A. 생성 : LF 기술자를 생성한 후, 이진 버디 할당 방식을 이용하여 세그먼트들을 할당한다.[8] 디렉토리내의 각 엔트리가 할당된 해당 세그먼트를 가리키도록 하고, 제어 데이타내의 각 필드에 생성된 LF의 특성을 반영시킨 후, 데이타를 저장한다.

B. 연속 액세스 : 디렉토리내의 첫 엔트리로부터 시작하여 마지막 엔트리까지 각 엔트리가 가리키는 세그먼트를 차례로 액세스하여 데이타를 읽어들인다.

C. 랜덤 액세스 : LF 기술자의 제어 데이타 부분에 저장된 첫 세그먼트의 크기와 세그먼트 할당 방식을 고려하여 원하는 바이트를 가지는 세그먼트를 직접 찾을 수 있다.

D. 삽입 : 랜덤 액세스를 이용하여 삽입될 위치 P를 가지는 세그먼트를 찾아낸다. P로부터 데이타를 차례로 삽입하는데, 삽입전 P이후에 존재하던 모든 데이타는 삽입된 새로운 데이타의 마지막 바이트 이후로 밀리게 된다. 이 과정에서 세그먼트들이 할당될 수 있으며, 이와 대응되는 엔트리들이 LF 기술자내의 디렉토리에 삽입된다. 끝으로, 총 세그먼트 수, 마지막 세그먼트를 구성하는 페이지의 수 등의 변화를 제어 데이타에 반영시킨다.

E. 삭제 : 삭제될 첫 바이트가 존재하는 위치 P를 가지는 세그먼트를 찾아낸다. 삭제될 마지막 바이트 이후에 존재하는 데이타를 P 직후로 끌어 당긴다. 이 과정에서 뒷부분에 존재하는 세그먼트들이 반환될 수 있으며, 이와 대응되는 엔트리들이 디렉토리로부터 삭제된다. 삭제 과정으로 인한 변화를 제어 데이타에 반영시킨다.

(3) 성능 특성

A. 무제한 크기의 지원 : LF 기술자와 세그먼트의 최대 크기가 고정되어 있기 때문에 제한된 크기의 데이타만을 저장할 수 있다. 이것은 전술한 WiSS LDI의 경우와 유사하나 LF의 엔트리 크기가 LDI보다 작으므로 많은 수의 세그먼트들을 가질 수 있고, 또한 각 세그먼트는 다수의 페이지들로 구성되므로 LDI에 비해서는 훨씬 큰 데이타를 지원할 수 있다.[9]

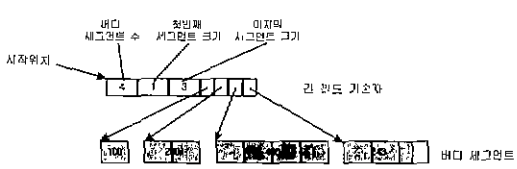


그림 4 Starburst에서의 긴 필드 관리 기법

8) 전체 크기를 미리 알고 있는 경우에는 첫 세그먼트를 최대 크기로 시작하도록 허용하지만, 그렇지 않은 경우에는 하나의 페이지로 시작하도록 한다.

9) 참고문헌[9]에서 언급한 구현에서는 약 400메가 바이트까지의 데이타를 저장할 수 있다.

응용 분야의 특성을 분석하여 세그먼트의 최대 크기를 조정함으로써 큰 제한 없이 대부분의 데이타를 수용할 수 있다.

B. 저장 공간 이용률 : 마지막 세그먼트 이외의 모든 세그먼트들은 빈 공간 없이 데이타로 채워지므로 대부분의 경우 100%에 가깝게 나타난다.

C. 연속 액세스 : 다음과 같은 두가지 이유로 인하여 현재 제안된 기법들 중 가장 뛰어난 성능을 보인다. 첫째, 높은 저장 공간 이용률로 인하여 액세스해야 할 페이지의 수가 최소화 된다. 둘째, 세그먼트가 물리적으로 인접한 페이지들의 집합이므로 한 번의 디스크 헤드 이동으로 한 세그먼트내의 모든 페이지들을 액세스할 수 있다.

D. 랜덤 액세스 : LF 기술자만을 액세스 함으로써 데이타의 크기와 원하는 바이트의 위치에 전혀 영향을 받지 않고, 해당 바이트가 저장된 세그먼트의 위치를 직접 알아낼 수 있으므로 매우 뛰어난 성능을 보인다.

E. 부분 변경 : BLOB 관리 기법에서와 마찬가지로 삽입 또는 삭제되는 데이타 이후에 나타나는 모든 데이타들을 이동시켜야 하므로 부분 변경시의 성능은 매우 떨어진다. 이러한 비효율성으로 인하여 LF에서는 실제로 부분 변경 연산을 지원하지 않고 있다.

3.5 EOS에서의 대형 동적 객체 관리 기법

EOS(Experimental Object Store)[1]는 Boston 대학에서 개발한 실험용 저장 시스템이며, 대형 객체의 관리를 위하여 대형 동적 객체(large dynamic object : LDO)[1]를 제공한다.

(1) 자료 구조

LDO는 Exodus LO의 연속 액세스시의 약점과 Starburst LF의 부분 변경시의 약점을 보완하자는 설계 동기에서 출발하였다. LDO는 세그먼트들의 집합과 디렉토리로 구성된다(그림 5 참조). 세그먼트는 데이타 저장 단위이며, 물리적으로 인접한 페이지들의 집합이다. EOS에서도 이진 버디 시스템 기반의 세그먼트 할당 방식을 사용하지만, 불필요한 페이지들을 미리 반환함으로써 세그먼트를 구성하는 페이지들의 수를 2^n개로 한정시키지 않고 자유로운 크기를 갖도록 허용한다. 데이타는 세그먼트의 시작 위치부터 연속적으로 저장되므로 세그먼트의 마지막 페이지에 한해서만 빈 공간이 존재할 수 있다. 내부 페이지들로 구성되는 디렉토리는 세그먼트들을 관리하기 위한 것이며, LO의 디렉토리와 똑같은 트리 형태의 구조를 갖는다. 그림 5는 페이지의 크기가 100 바이트라는 가정하에 943 바이트 크기를 갖는 데이타가 LDO로 저장된 예를 나타낸 것이다. 내부 페이지는 최대 세 개의 엔트리들을 가질 수 있다고 가정한다.

(2) 연산 알고리즘

LDO를 위한 트리 구조는 리프 페이지를 세그먼트로 대체한 것을 제외하고는 Exodus LO와 동일하므로 LDO의 각종 알고리즘도 LO의 것과 유사하다.

A. 생성 : 내부 페이지를 하나 생성한 후, 첨가 연산(append)을 통하여 현존하는 데이타의 마지막 부분에 빈 공간 없이 새로운 데이타를 삽입함으로써 LDO를 단계적으로 생성한다.[10] 이 결과, 세그먼트들은 LF와 같게 되고, 디렉토리는 LO와 같게 된다.

B. 연속 액세스 : 루트 페이지에서 시작하여 트리를 깊이 우선 탐색 방식으로 방문함으로써 각 세그먼트내의 데이타를 차례로 읽어들인다.

C. 랜덤 액세스 : LO에서와 같은 트리 탐색 방법을 이용하여 원하는 바이트를 가지는 세그먼트를 찾아낸다.

D. 삽입 : LO의 삽입 과정과 유사하나, 다양한 크기의 세그먼트가 할당될 수 있다는 것과 세그먼트가 세 개의 세그먼트로 분

10) 할당되는 첫 세그먼트의 크기는 Starburst의 LF에서와 같은 방식을 사용한다.

할될 수 있다는 것이 큰 차이점이다. 후자는 데이타의 첫 바이트가 삽입될 세그먼트 S내에 충분한 공간이 없는 경우에 발생한다. S를 데이타가 삽입될 위치 P (P를 가지는 페이지를 P'이라 하자)를 기준으로 S내에서 P 직전까지의 데이타를 포함하는 왼쪽 페이지들의 집합(P' 포함)과 P'의 오른쪽 페이지들의 집합으로 각각 구성되는 두 개의 세그먼트로 분할한다. 또한, 삽입될 데이타와 P'내에서 P 이후의 데이타를 저장할 수 있는 크기의 새로운 세그먼트를 할당하여 두 데이타를 이곳에 저장한다. 이로 인하여 S가 세 개의 세그먼트로 되었으므로 상위 단계 내부 페이지에서 S와 대응되는 엔트리를 새로운 세 개의 엔트리로 대치시킨다. 만일, 내부 페이지내에 충분한 공간이 없는 경우에는 유사한 작업을 상위 단계로 올라가며 재귀적으로 반복하여 처리한다.

E. 삭제 : LO에서와 마찬가지로 삭제될 데이타를 가지는 서브 트리를 제거한다. 이러한 작업이 종료되면, 삭제될 첫 바이트 P1을 가지는 세그먼트 S1은 P1이전의 데이타까지만을 가지는 페이지들로 세그먼트로 축소되고, 마지막 바이트 P2를 가지는 세그먼트 S2도 P2를 포함하는 페이지 이후의 페이지들만을 가지도록 축소된다. 또한, 축소된 S2에서 P2를 포함하는 한 페이지는 새로운 세그먼트로 독립된다.[11] 이 결과, S1, S2가 세 개의 새로운 세그먼트로 되었으므로 상위 단계 내부 페이지에서 기존의 두 엔트리들을 새로운 세 개의 엔트리로 대치시키는 작업을 재귀적으로 반복한다.

(3) 성능 특성

A. 무제한 크기의 지원 : 트리의 특성상 세그먼트들을 얼마든지 많이 가질 수 있으므로 무제한의 크기를 지원한다.

11) 이러한 삭제 연산의 빈번한 경우, 대부분의 세그먼트들이 하나의 페이지로 구성될 수 있다. EOS에서는 일정한 임계치(threshold) T를 두어 세그먼트의 크기가 가능한 T 이상이 되도록 하고 있다[1].

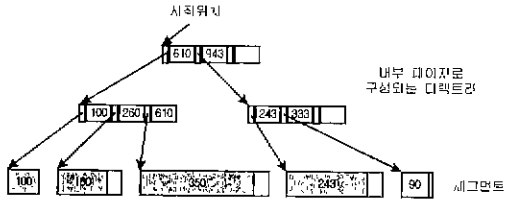


그림 5 EOS에서의 대형 동적 객체 관리 기법

B. 저장 공간 이용률 : 최초의 생성 이후에 발생하는 삽입과 삭제 연산으로 인하여 세그먼트에서 빈 공간이 생성될 수 있으나 세그먼트내에서는 데이타를 연속적으로 저장하므로 세그먼트의 마지막 페이지에 한하여 빈 공간이 발생한다. 따라서 BLOB이나 LF에서와 같이 100%에 가까운 값을 나타내지는 않으나 LO나 LDI에 비해서는 훨씬 높은 값을 갖는다.

C. 연속 액세스 : 높은 저장 공간 이용률과 물리적으로 연속된 페이지들로 구성되는 세그먼트의 특성으로 인하여 연속 액세스시 좋은 성능을 보인다. 그러나 LF와 비교하면, 깊이 우선 탐색으로 인한 내부 페이지 액세스의 오버헤드가 있고, 저장 공간 이용률도 상대적으로 낮으므로 성능이 떨어진다.

D. 랜덤 액세스 : LO에서와 마찬가지로 루트 페이지에서 해당 세그먼트까지의 경로상에 나타나는 페이지들만을 액세스 하므로 랜덤 액세스시의 성능은 뛰어나다.

E. 부분 변경 : LO에서와 같이 부분 변경과 연관되지 않는 다른 데이타의 이동은 발생되지 않으므로 좋은 성능을 보인다.

4. 국내의 연구 현황

국내에서는 90년대에 이르러 멀티미디어에 대한 관심이 고조되기 시작하였으며, 대형 멀티미디어 저장에 관한 연구도 이와 함께 시작되었다. 본 장에서는 이러한 국내의 연구 현황에 관하여 간략히 논의하고자 한다.

한국과학기술원에서는 관계 DBMS는 물론 객체지향 DBMS, 멀티미디어 DBMS, CAD/CAM 등과 같은 새로운 데이타베이스 응용들을 효율적으로 지원하기 위하여 다사용자용 다

목적 저장 시스템인 KAOSS(KAIST Object Storage System)[8]를 개발하였다. KAOSS의 객체 관리자는 각종 데이타를 객체 단위로 저장하며, 사용자가 크기에 관계 없이 객체를 자유롭게 조작할 수 있도록 통일된 인터페이스를 제공한다.

KAOSS에서는 대형 객체를 관리하기 위하여 Exodus의 LO와 유사한 형태의 자료 구조를 이용하였다.

서울대학교에서는 복합 객체와 멀티미디어 객체 저장을 위하여 MOSS(Multimedia Object Storage System)[7]를 개발하였다. MOSS의 대용량 레코드 관리자는 대형의 데이타를 관리하며, WiSS의 LDI와 유사한 자료 구조를 사용한다. LDI의 크기 제한의 문제점을 극복하기 위하여 Unix 파일 시스템에서 i-노드를 관리하는 기법과 유사한 다중 단계 디렉토리를 사용하였다.

충남대학교에서는 대형의 데이타 저장을 위하여 기 개발된 '바다' 관계 DBMS를 확장하는 연구를 수행하였다[12].

기본 자료 구조는 Exodus의 LO와 유사하나 오버플로우 페이지 개념을 추가로 도입하였다. 도입 목적은 삽입과 삭제가 빈번히 발생하는 동적인 환경에서 트리를 재구성하는 과정을 제거하고, 연속 액세스의 성능도 일부 보완하고자 한 것이다.

중앙대학교에서는 멀티미디어 데이타의 특성이 응용에 따라 매우 다양하다는 것에 착안하여 기존의 각 방식으로는 이러한 다양한 특성을 모두 최적으로 맞출 수 없음을 보였다[4]. 또한, 이를 극복하기 위하여 여러 타입의 대형 객체 관리 구조를 함께 제공함으로써 응용 프로그래머가 해당 분야에 가장 적합한 타입을 선택하여 사용할 수 있도록 하는 기법을 제안하였다.

끝으로, 대우 통신에서는 '바다' 관계 DBMS를 '한바다'라는 이름으로 상용화하면서 대형 멀티미디어 저장을 위한 기능을 추가하였다. 자료 구조는 Sybase, Informix 등 국외의 상용 DBMS와 마찬가지로 연결 리스트를 기반으로 하는 BLOB 구조를 사용하는 것으로 알려져 있다[11].

5. 결 론

최근 컴퓨터 기술의 눈부신 발전으로 인하여 다양한 멀티미디어 데이타들을 컴퓨터 시스템 내에서 다룰 수 있게 되었다. 이러한 멀티미디어 데이타의 가장 큰 특성은 저장을 위하여 수십 메가 바이트에서 수 기가 바이트에 이르는 매우 큰 저장 공간을 요구한다는 것이다. 본 논문에서는 대형 데이타 관리를 위한 최근의 연구 결과에 관하여 논의하였다. 먼저, 대형 객체 관리 기법의 설계시 고려 사항에 관하여 논의하고, 현재 상용 DBMS에서 지원하는 기법과 WiSS, Exodus, Starburst, EOS 등 최근에 개발된 대표적인 저장 시스템들에서 지원하는 각각의 기법에 대하여 (1) 기본 자료 구조, (2) 각종 연산 알고리즘, (3) 성능상의 특성 및 장단점 등을 비교, 검토하였다. 또한, 본 주제와 연관된 국내의 연구 동향에 관하여 아울러 언급하였다.

대형 멀티미디어 데이타 관리와 관련된 그밖의 연구 이슈들은 다음과 같다.

첫째, 대형의 데이타를 일반 데이타와 구분하여 별도로 저장할 것인가 혹은 함께 저장할 것인가 하는 문제이다[13][1]. 전자는 대형 데이타를 별도의 대형 속성으로서 따로 저장하고, 이에 대한 기술자(descriptor)만을 연관된 일반 속성들과 함께 저장하는 방식이다. 일반 속성만을 액세스하는 응용의 경우, 같은 소형 객체로부터 연관된 속성들을 함께 액세스할 수 있으므로 최적의 성능을 낼 수 있다. 또한, 사용 빈도가 높지 않는 대형 속성은 저가의 저장 장치내에 관리함으로써 유지 비용을 절감할 수 있다[13]. 한편, 후자는 대형 데이타를 일반 데이타와 함께 저장하여, 이들의 집합인 객체 자체를 대형화 하는 방식이다. 이러한 경우, 전자의 장점들을 얻을 수는 없으나 대형 객체와 소형 객체에 대한 통일된 형태의 인터페이스를 제공할 수 있으며, 일반 속성들의 수가 많아 한 페이지를 초과하는 객체도 다룰 수 있다.

둘째, 버퍼링을 어떻게 제공할 것인가 하는 문제이다. 대형 데이타는 페이지 보다 훨씬 크고, 가변길이를 가지므로 페이지를 단위로 하는 일반 버퍼링 기법을 그대로 적용하는 경우

많은 문제점이 발생한다[2]. 대형 데이타를 위하여 버퍼링 기능을 제공해야 하는가를 규명하는 근본적인 문제. 적절한 대치 알고리즘을 고안하는 문제, 그리고 일반 데이타와 버퍼를 공유할 것인가 혹은 별도로 유지해야 하는가 하는 문제 등이 있다.

셋째, 동시성 제어 문제이다. 물론, 액세스시 대형 데이타 전체에 락(lock)을 걸어둠으로써 간단히 상호 배제를 구현할 수 있다. 그러나 같은 대형 데이타의 서로 다른 부분을 액세스하고자 하는 두 트랜잭션이 동시에 수행될 수 없으므로 동시성이 떨어진다. 특히, 멀티미디어 응용에서는 트랜잭션이 장기간 수행되는 경향이 많으므로 이러한 방식은 바람직하지 않다. 따라서 동시성의 향상을 위하여 액세스되는 부분만을 대상으로 하는 새로운 상호 배제 기법이 요구된다.

넷째, 파손 회복 문제이다. 현재, 회복을 위하여 가장 널리 사용되는 기법은 WAL (Write-Ahead Logging)이다. 그러나 대형 데이타의 경우 변경되는 부분의 크기도 상당히 크므로 WAL을 그대로 사용하는 경우 로깅의 오버헤드가 상당히 크게 나타난다[13]. 이를 극복하기 위한 방법으로서 쉐도우(shadow) 기법의 도입이 제시되고 있으나[2, 13], 구현에 관한 구체적인 연구 논문은 아직 발표되지 않고 있다.

Acknowlegment

본 연구는 95년도 인공지능연구센터 위탁과제 연구비와 96년도 한국과학재단 핵심전문과제 연구비(과제 번호 : 961-0903-019-1)의 부분적인 지원에 의한 결과임.

참고문헌

[1] Billris. A., "An Efficient Database Storage Structure for Large Dynamic Objects," In *Proc. Intl. Conf. on Data Engineering, IEEE,* pp. 301-308, Feb. 1992.

[2] Carey, M. et al., "Storage Management for Objects in Exodus," In Book *Object-Oriented Concepts, Databases, and Applications*(Eds. Kim, W. and Lochovsky, F. H), ACM Press, pp. 341-369, 1989.

[3] Chou, H-T., et al., "Design and Implementation of the Wisconsin Storage System," *Software-Practice and Experience,* Vol. 15, No. 10, pp. 943-962, Oct. 1985.

[4] 최기호, 강은지, 강현철, "멀티미디어 데이타의 효율적 저장을 위한 BLOB 타입의 다양화," *한국정보과학회 논문지(B)*, 제 22권, 제 10호, pp. 1404-1415, 1995년 10월.

[5] 다우 기술, Informix-OnLine Mannual, 1992.

[6] Exodus Implementation Team, Exodus Storage Manager : Architectural Overview, 1991.

[7] 김범수, 이상돈, 권택근, 최유용, 이석호, "멀티미디어 객체 저장 시스템 MOSS의 개발," 데이타베이스 연구회지, 제 11권, 제 2호, pp. 47-62, 1995년 7월.

[8] 이영구, 김상욱, 김원영, 장지웅, 황규영, "KAOSS : 새로운 데이타베이스 응용을 지원하는 다사용자용 다목적 데이타베이스 저장 시스템," *데이타베이스 연구회지*, 제 11권, 제 2호, pp. 91-110, 1995년 7월.

[9] Lehman, T. J. and Lindsay, B. G., "The Starburst Long Field Manager," In *Proc. Intl. Conf. on Very Large Data Bases*, VLDB, Amsterdam, pp. 375-383, 1989.

[10] Lohman, G. M., et al., "Extensions to Starburst : Objects, Types, Functions, and Rules," *Comm. of the ACM*, Vol. 34, No. 10. pp. 94-109, Oct. 1991.

[11] 박상현, 대우통신 한바다 개발팀, *Private Communication.*

[12] 류은숙, 마경호, 이강찬, 조옥자, 이규철, "바다 관계 DBMS의 멀티미디어 기능 지원을 위한 확장 설계 및 구현," *한국정보과학회 논문지(C)*, 게재 예정.

[13] Shelter, T., "Birth of the BLOB." *BYTE*, pp. 221-226, Feb. 1990.

[14] 송일렬, "91 컴퓨터 과학 하계 세미나 : 멀티미디어 데이타베이스 시스템을 위한 정보 검색, 한국과학기술원 인공지능연구센터, 1991년 7월.

[15] Sybase SQL Server Release 10 - Transact-

SQL Commands.

[16] 황규영, 박종목, 노응기, 박병권, "멀티미디어 데이타베이스," *정보과학회지*, 제 12권, 제 7호, pp. 59-69, 1994년 8월.

김 상 욱

1989 서울대학교 컴퓨터공학과 졸업(학사)
1991 한국과학기술원 전산학과 졸업(석사)
1994 한국과학기술원 전산학과 졸업(박사)
1991 미 스탠포드 대학 방문 연구원
1994~95 한국과학기술원 정보전자연구소 Post-Doc.
관심분야 : 데이타베이스 시스템, DBMS, 트랜잭션 프로세싱, 다차원 동적 화일, 공간데이타베이스/GIS, 객체지향 데이타베이스, 멀티미디어 데이타베이스

이 영 구

1992 과학기술대학 전산학과 졸업(B.S.)
1994 한국과학기술원 전산학과 졸업(M.S.)
1994~현재 한국과학기술원 전산학과 박사과정
관심분야 : 데이타베이스 시스템, 다중 키 액세스, 객체지향 데이타베이스, 공간 데이타베이스

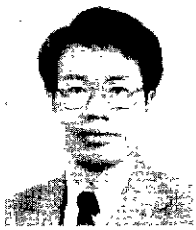
김 원 영

1989 이화여자대학교 전자계산학과 졸업(B.S.)
1991 한국과학기술원 전산학과 졸업(M.S.)
1991~현재 한국과학기술원 전산학과 박사과정
관심분야 : 데이타베이스 시스템, 동시성 제어, 분산 데이타베이스

장 지 웅

1993 연세대학교 전산학과 졸업(B.S.)
1995 한국과학기술원 전산학과 졸업(B.S.)
1995~현재 한국과학기술원 전산학과 박사과정
관심분야 : 데이터베이스 시스템, 다중 키 액세스, 객체지향 데이타베이스, 동시성 제어, 주기억 장치 데이타베이스

황 규 영



1973 서울대학교 전자과 졸업(B.S.)
1975 한국과학기술원 전기 및 전자공학과 졸업(M.S.)
1982 Stanford University(전산학과, M.S.)
1983 Stanford University(전산학과, Ph.D.)
1975~78 국방과학연구소 선임연구원
1983~90 IBM T. J. Watson Research Center, Research Staff Member
1991 Visiting Professor, Stanford University
1992 Visiting Associate Professor, Georgia Institute of Technology
1993 Hewlett-Packard Laboratories 기술자문 (Palo Alto)
1992~94 한국정보과학회 데이타베이스연구회(SIGDB) 운영위원장
1993~현재 체신부 통신진흥협의회 DB산업육성분과 위원장
1995 한국정보과학회 이사 겸 논문지 편집위원장
Editor : The VLDB Jorunal
Editor : Distributed and Parallel Databases : An International Jorunal
Associate Editor : The IEEE Data Engineering Bulletin (1990-1993)
Editor : International Jorunal of Geographical Information Systems
1990~현재 인공지능연구센터 데이타베이스 및 멀티미디어 연구실장
1990~현재 한국과학기술원 전산학과 교수
관심분야 : 데이타베이스 시스템, 멀티미디어 데이타베이스, 객체지향 데이타베이스 하이퍼미디어, GIS, 분산 데이타베이스 및 Client/Server 기술, 연역 데이타베이스, 공학 데이타베이스, 사무자동화, CASE, 전문가 시스템