□ 기술해설 □

주기억 장치 데이타베이스를 위한 저장 시스템†

한국과학기술원 황규영* · 장지웅** · 이영구** · 김원영*

● 목 차 ●

1. 서 론
2. 주기억 장치 저장 시스템의 특성
2.1 주기억 장치 관리 기법
2.2 엑세스 기법
2.3 동시성 제어 기법
2.4 파손 회복 기법
3. 주기억 장치 데이타베이스 시스템의 소개
3.1 설계만 수행된 시스템
3.2 구현된 시스템
4. 결 론

1. 서 론

최근들어 PCS(personal communication system)에서의 HLR(home location register), 이동 컴퓨팅(mobile computing), 공장제어를 위한 CIM (computer integrated manufacturing), FA(factory automation) 등 신속한 처리를 요구하는 응용 분야가 점차 확대되고 있다. 현재 널리 사용되고 있는 디스크 기반 데이타베이스 시스템(disk resident database system)에서는 데이타를 다루기 위한 디스크 액세스의 오버헤드가 지나치게 크므로 빠른 처리 속도를 요구하는 응용에 적합하지 않다. 더구나 주기억 장치의 용량이 커지고 가격이 많이 하락함에 따라 컴퓨터 시스템 내의 주기억 장치 용량은 점점 증가하는 추세에 있다. 이에 따라 데이타베이스 분야에서는 늘어나는 주기억 장치의 용량을 최대한 활용하여 디스크 내에 저장된 데이타를 모두 주기억 장치로 상주시켜 데이타베이스 시스템의 성능을 개선하는 주기억 장치 데이타베이스 시스템(memory resident database system)에 관한 연구가 활발히 진행 중에 있다[4, 20, 21].

주기억 장치 저장 시스템은 주기억 장치 데이타베이스 시스템의 하부 구조에 해당하며, 데이타를 저장하고 검색하는 부분을 직접 관리함으로써 데이타베이스의 성능에 직접적인 영향을 준다. 주기억 장치 저장 시스템에서 대용량의 주기억 장치를 활용하는 방법으로는 크게 두가지가 있다. 첫째는 대용량 주기억 장치를 기존의 디스크 기반 저장 시스템의 커다란 버퍼 영역으로 활용하는 방법이고[4], 둘째는 모든 데이타베이스를 주기억 장치에 저장하는 방법이다[1]. 첫번째 방법은 기존의 디스크 기반 저장 시스템을 큰 수정없이 사용할 수 있다는 장점이 있으나, 디스크 기반 저장 시스템은 기본적으로 디스크 액세스를 최소화는데 중점을 두어 설계되었으므로 주기억 장치 저장 시스템의 특성을 최대한 활용할 수 없다는 단점이 있다. 두번째 방법은 주기억 장치의 특성을 최대한 활용할 수 있도록 주기억 장치와 CPU의 효율적인 사용에 중점을 두어 저장 시스템 자체를 새로이 설계하고 구현하는 방법이다[7].

주기억 장치는 디스크와 비교할 때 다음과 같은 특성을 지닌다[7]. 첫째, 주기억 장치에 대한 액세스는 디스크 액세스에 비하여 속도가 매우 빠르다. 그러므로 디스크 기반 저장 시스템에서는 별로 문제가 되지 않았던 로킹과 같

† 본 연구는 '95 현대전자(주) 연구과제와 인공지능 연구센터의 지원을 받은 결과임.
*종신회원
**학생회원

은 연산이 큰 오버헤드로 작용할 수 있다. 둘째, 디스크는 영구적인 저장 구조인데 반하여 주기억 장치는 일반적으로 휘발성이다. 그러므로 주기억 장치 저장 시스템에서는 백업(back-up) 및 파손 회복 기능이 매우 중요하다. 셋째, 디스크는 액세스되는 데이타의 크기에 크게 구애받지 않으며, 한번의 액세스에 크지만 일정한 비용을 필요로 한다. 그러나 주기억 장치는 액세스 비용이 데이타의 크기에 비례한다. 넷째, 디스크는 데이타의 배치 상태가 액세스 속도에 큰 영향을 미친다. 그러므로 클러스터링(clustering)이 매우 중요하다. 그러나 주기억 장치는 데이타의 배치 상태가 액세스 속도에 영향을 미치지 않으므로 클러스터링이 불필요하다.

본 논문에서는 위와 같은 주기억 장치의 특성을 활용하여 모든 데이타베이스를 주기억 장치에 저장하고 관리하는 주기억 장치 저장 시스템에 대하여 개괄적으로 고찰한다. 제2장에서는 주기억 장치 저장 시스템의 개발에 관련된 여러 기법들을 살펴보고, 제3장에서는 기존의 주기억 장치 데이타베이스 시스템에 대하여 간략하게 소개하며, 제4장에서 결론을 맺는다.

2. 주기억 장치 저장 시스템의 특성

주기억 장치 저장 시스템도 디스크 기반 저장 시스템과 같이 객체의 저장 및 관리와 객체에 대한 액세스 기능을 제공하는 것이 필요하며, 다사용자가 동시에 사용할 때에 데이타의 일관성을 제공하기 위한 동시성 제어 기능과 시스템 파손시를 대비한 회복 기능이 필요하다. 본 장에서는 주기억 장치 저장 시스템의 특성에 대하여 가능별로 구분하여 설명한다.

2.1 주기억 장치 관리 기법

주기억 장치 저장 시스템은 디스크 액세스가 없으므로 기존의 디스크 기반 저장 시스템과는 달리 디스크 액세스 횟수를 최소화하기 위한 노력이 필요하지 않다. 즉, 클러스터링과 같은 작업이 더이상 의미가 없다. 그러나 대용량 주기억 장치의 사용이 가능하다 할지라도 아직은 주기억 장치의 용량에 한계가 있고, 로킹(locking)과 같은 오버헤드의 비중이 디스크 기반 저장 시스템에 비하여 상대적으로 매우 높으므로 주기억 장치를 효율적으로 관리해야 한다. 특히, 가변적인 길이를 갖는 데이타를 주기억 장치 내에서 효율적으로 저장하고 관리할 수 있는 방법이 중요하다.

본 절에서는 가변길이 데이타를 효율적으로 관리할 수 있는 주기억 장치 관리 기법들에 대하여 간략하게 소개하고 장단점을 비교한다.

2.1.1 Starburst에서의 주기억 장치 관리 방법

Starburst[14]는 디스크 기반 저장 시스템과 주기억 장치 저장 시스템을 겸용하고 있는 프로토타입 DBMS이다. 본 절에서는 Starburst의 주기억 장치 릴레이션 관리자(MMM : main memory relation manager)에서 주기억 장치를 관리하고 객체를 저장하는 방법에 대하여 설명한다. Starburst에 대한 소개는 제 3장에 기술하였다.

Starburst에서는 주기억 장치를 할당하고 반환하는 기본 단위로 파티션(partition)을 사용한다. 파티션은 일정한 크기의 연속적인 주기억 장치 공간으로서 디스크 기반 저장 시스템에서의 디스크 페이지와 같은 역할을 한다. 그림 1은 파티션의 구조를 나타낸다. 파티션은 끝부분에 record slot array를 두어 파티션 내에서 객체의 위치를 가리키도록 하여 파티션

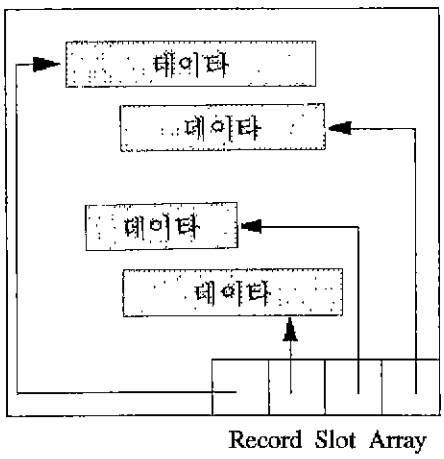


그림 1 Starburst의 파티션 구조

내에서 객체의 위치와 크기를 자유로이 변경할 수 있다.

이 방법의 장점은 크기가 가변적인 객체를 저장할 때에도 높은 저장효율(storage utilization)을 가진다는 것이다. 그러나, 파티션 내에서의 단편화 현상(fragmentation)을 해결하기 위하여 파티션을 컴팩션(compaction)해야 하고, 새로운 객체를 생성하는 경우 충분한 크기의 자유공간(free space)을 가진 파티션을 빠르게 찾을 수 있도록 파티션 내의 자유공간의 크기별로 파티션들을 연결하여 관리해야 하므로 많은 오버헤드가 따른다.

2.1.2 버디 시스템을 이용한 주기억 장치 관리 방법

가변길이 데이타를 효율적으로 관리하는 방법으로 널리 알려져있는 방법 중의 하나가 버디 시스템(buddy system)[9]이다. 본 절에서는 가장 간단한 형태인 이진 버디 시스템(binary buddy system)에서의 주기억 장치 구조 및 주기억 장치의 할당과 반환 방법에 대하여 기술한다.

버디 시스템은 주기억 장치 블럭(block)을 정해진 크기로 정해진 위치에서만 할당하고, 이 블럭을 버디라고 불리우는 이웃한 블럭과 합하여 더 큰 블럭을 생성함으로써 주기억 장치를 동적으로 관리하는 방법이다. 이진 버디 시스템에서는 2의 거듭제곱인 크기의 블럭만 할당될 수 있다. 주기억 장치의 크기가 $N=2^m$이라고 하면, 블럭의 가능한 크기는 $2^m, 2^{m-1}, \ldots, 2^1, 2^0$가 된다. 그리고 크기가 2^k인 각 블럭은 2^k의 배수가 되는 주소(address)에서만 시작될 수 있다.

그림 2는 이진 버디 시스템의 한 예이다. 그림에서 점선으로 연결된 블럭들은 서로가 버디 관계 임을 나타낸다. 빗금친 블럭은 사용 중인 블럭을 나타내고, 그렇지 않은 블럭은 비사용 중 블럭을 나타낸다. 각 블럭의 가능한 크기에 대해서 비사용중 블럭들은 프리 리스트(free list)라고 불리우는 양방향 연결 리스트로 연결되어 있다. 그림 2는 크기가 1인 블럭 하나와 크기가 2인 블럭 하나가 할당된 상태를 나타낸다.

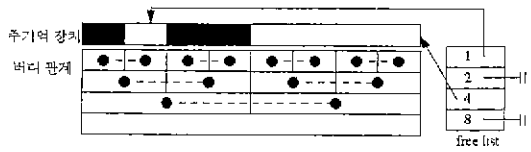


그림 2 이진 버디 시스템

버디 시스템에서는 일정 크기의 블럭에 대한 할당 요청이 들어오면 먼저 요청된 블럭의 크기에 대한 프리 리스트에서 할당해줄 블럭을 찾는다. 만일 해당 프리 리스트에 할당해 줄 블럭이 없으면 요청된 크기보다 큰 블럭을 반복적으로 두 개의 버디로 나누어 블럭을 할당한다. 어떤 블럭이 반환되었을 때 그 블럭의 버디가 사용 중이 아니면 버디 관계인 두 블럭을 합쳐서 커다란 블럭을 만들어 나간다. 이 방법은 실제 객체의 크기에 비하여 큰 주기억 장치 영역을 할당하여야 하므로 Starburst의 주기억 장치 관리 기법에 비하여 저장효율이 떨어지는 단점이 있다.

2.2 액세스 기법

저장 시스템은 저장된 객체들 중에서 특정한 내용을 가진 객체를 빠르게 액세스할 수 있는 방법을 제공해야 한다. 이러한 액세스 방법으로 널리 쓰이는 것이 색인(index)을 이용한 방법과 해싱(hashing)을 이용한 방법이다. 본 절에서는 대표적인 주기억 장치 색인 구조인 T-트리와 여러가지 해싱 기법에 대하여 간략하게 소개한다.

2.2.1 T-트리

디스크 기반 저장 시스템에서는 디스크에 있는 객체를 적은 수의 디스크 액세스로 검색하기 위해 fan-out이 크고, 깊이(depth)가 작은 B^+-트리[10, 3]를 주로 사용한다. 그러나, 주기억 장치 저장 시스템에서는 디스크 액세스를 하지 않으므로 높은 저장 효율과 빠른 수행 속도가 중요하며, 이러한 요구 사항을 만족시키는 대표적인 주기억 장치 색인 구조로 T-트리[11]가 있다.

T-트리는 AVL트리[9]와 마찬가지로 몇가지 제한이 가해진 균형된 이진 탐색 트리(balanced binary search tree)이다[11]. AVL 트

리는 갱신 연산이 재균형 알고리즘(rebalance algorithm)에 의해 빠른 시간에 이루어지는 반면 노드 하나당 키(key)가 한개씩만 저장되어 저장 공간 효율이 좋지 않다. T-트리는 AVL 트리의 단점을 보완하여 B^+-트리처럼 하나의 노드에 여러개의 키를 저장하고 노드 내에서는 이진탐색(binary search)을 이용하여 데이타를 찾는 방법으로, AVL트리에 비하여 저장 효율이 높고, 삽입과 삭제가 하나의 노드 안에서 행해지는 경우가 많으므로 AVL트리보다 트리의 재균형 연산이 덜 발생한다.

2.2.2 해쉬(Hashing) 기법

디스크에 존재하는 화일에서 해쉬에 기반한 탐색 방법에 대한 많은 연구가 수행되어져 왔으며, 화일의 크기가 동적으로 변하는 경우에도 데이타에 대한 빠른 엑세스를 제공하고자 하는 방법들이 연구되어 왔다. 대표적인 해쉬 기법으로는 선형해쉬 기법(linear hashing)[15], 확장해쉬 기법(extendible hashing)[6] 등이 있다.

디스크 기반으로 고안된 해쉬 방법들은 쉽게 주기억 장치 환경에 적용할 수 있으나 주기억 장치의 특성을 보다 효율적으로 이용하기 위해 기존의 해쉬 방법을 수정한 방법들도 제안 되었다. 그 예로는 버켓연결 해쉬 기법(CBH : chained bucket hashing)[10], 다단계 디렉토리 해쉬 기법(multi-directory hashing) [2] 등을 들 수 있다.

CBH는 해쉬 테이블의 크기가 고정되어 있고, 충돌이 발생하는 경우 오버플로우 체인을 연결하는 방법으로서 정적인 화일에 대한 빠른 엑세스를 제공한다. 그러나 동적인 화일에 대하여 적용하기에는 적당하지 않다. 이 방법은 특별히 주기억 장치 환경에 적합하도록 개발된 방법은 아니지만 버켓의 크기를 키우는 정도로 주기억 장치 환경에 적합하게 수정할 수 있다.

확장해쉬 기법이나 선형해쉬 기법과 같이 동적인 화일에 적합한 해쉬 기법들은 해쉬 테이블의 크기가 급격하게 증가하는 문제점이 있다. 이러한 문제점을 보완한 방법이 다단계 디렉토리 해쉬 기법이다. 다단계 디렉토리 해쉬 기법에는 고속탐색 다단계 디렉토리 해쉬 기법(FSMH : fast search multi-directory hashing)과 탐색제어 다단계 디렉토리 해쉬(CSMH : controlled search multi-directory hashing)[2] 등이 있다. FSMH은 키값의 일부분으로 루트 디렉토리에 대해 해쉬을 적용하고, 디렉토리의 각 엔트리는 다른 해쉬 테이블을 가르킨다. 이 경우 키값의 나머지 부분을 이용하여 다시 해쉬을 하게 된다. 이렇게 해쉬을 반복적으로 적용함으로써 마치 해쉬 테이블을 하나의 노드로 하는 트리 구조를 탐색하는 듯한 효과를 갖는다. 이 방법은 해쉬 테이블의 크기가 2배씩 증가하는 것이 아니라 국소적으로 증가하며, 해쉬 테이블이 연속된 주기억 장치 공간에 위치할 필요가 없다. CSMH는 FSMH와 비슷하며, 단지 충돌 발생 시 탐색 횟수의 평균 값, 저장효율 등을 사용자가 조정할 수 있도록 함으로써 평균적으로 일관성 있는 엑세스를 제공할 수 있다.

2.3 동시성 제어 기법

현재까지 개발된 대부분의 주기억 장치 데이타베이스 시스템에서는 로킹 기법을 사용하여 동시성을 제어하고 있다. 본 절에서는 주기억 장치 저장 시스템에서 로킹 기법에 관련된 내용에 대하여 설명한다.

2.3.1 주기억 장치 저장 시스템에서 로킹 기법의 특징

디스크 기반 저장 시스템에서는 로크를 획득하는 연산이 디스크 액세스에 비하여 매우 빠른 연산이므로 로크로 인한 오버헤드가 별로 고려되지 않았다. 그러므로 로킹의 횟수가 증가하더라도 로크의 단위를 작게 함으로써 로크의 충돌을 줄이는 방법이 많이 사용되었다.

그러나, 주기억 장치 저장 시스템에서는 트랜잭션들이 빠른 속도로 데이타를 액세스하여 비교적 빨리 끝나게 되므로 로킹 연산의 오버헤드가 상대적으로 크게 된다. 즉, 로크의 충돌이 발생하는 경우가 적어지므로 로크의 단위를 작게 함으로써 얻어지는 이익의 비중이 줄어들게 된다. 심지어 로크의 단위를 데이타베이스 전체로 함으로써 트랜잭션들을 하나씩 순서적으로 수행하는 경우도 생각할 수 있다[7]. 이

렇게 할 경우 로킹으로 인한 오버헤드가 감소할 뿐만 아니라 교착 상태(deadlock)를 방지하기 위한 작업도 할 필요가 없다. 그러나, 이러한 방법은 트랜잭션이 오랜 시간에 걸쳐 수행되는 경우에는 적당하지 않으므로 응용 프로그램의 성격에 따라 적절한 로크의 단위를 설정하여 사용하는 것이 필요하다.

2.3.2 로크 테이블의 구조

디스크 기반 저장 시스템에서는 로크할 객체에 대한 로크 정보를 액세스하기 위하여 해쉬 기법을 사용한다. 그 이유는 모든 객체에 대한 로크 정보를 주기억 장치내에 둘 수 없기 때문이다. 그러나 주기억 장치 저장 시스템에서는 모든 객체가 주기억 장치에 저장되므로 객체가 직접 자신의 로크 정보를 가지고 있는 방법을 생각할 수 있다. 즉, 객체의 헤더 부분에 그 객체가 로킹되어 있는지 여부를 나타내는 정보를 기록하고, 로크로 인하여 대기해야 하는 일이 발생하는 경우에만 해쉬 기법을 사용한다[7]. 주기억 장치 저장 시스템의 경우 제 2.3.1절에서 언급한 바와 같이 로킹으로 인한 충돌이 발생할 확률이 적으므로 대부분의 로킹 연산은 객체 헤더(header)의 정보를 수정하는 정도로 충분하다. 이렇게 함으로써 주기억 장치 저장 시스템의 특성을 살려 로킹 연산의 오버헤드를 줄일 수 있다.

2.4 파손 회복 기법

주기억 장치 저장 시스템을 구축하는데 있어서 반드시 고려해야 하는 문제점의 하나는 파손시의 회복이다[16, 13, 19]. 주기억 장치 저장 시스템에서도 디스크 기반 저장 시스템에서 발생되는 트랜잭션 오류, 시스템 오류, 미디어 오류가 발생할 수 있으므로 이에 대처할 수 있는 회복 기능이 요구된다.

주기억 장치 저장 시스템에서는 데이타를 저장하는 매체가 주기억 장치의 특성을 따르게 되므로 디스크 기반 저장 시스템에서 사용되는 회복 알고리즘을 그대로 사용하는 경우에는 그 효율성에 큰 문제가 있다[13]. 그러므로 주기억 장치의 특성을 충분히 활용할 수 있는 새로운 회복 기법이 필요하다. 본 절에서는 트랜잭션 종료시의 동작과 체크포인트시의 동작 및 회복을 위한 재시동 시의 동작에 대하여 설명한다.

2.4.1 트랜잭션 종료시의 동작

트랜잭션의 종료시 로그 레코드(log record)를 안전한 저장 장치에 보존시키는 기법으로는 다음의 네가지가 연구되었다.

Immediate Commit

Immediate commit[16]은 디스크 기반 데이타베이스 시스템에서 사용되는 WAL(write ahead logging)기법의 트랜잭션 종료와 같은 방식으로 수행된다. 즉, 트랜잭션이 수행되는 동안 모든 변경은 주기억 장치 내의 데이타베이스에 그대로 반영되며, 로그 레코드를 로그 버퍼에 기록한다. 트랜잭션의 종료 시점에 이르면 commit 로그 레코드를 기록하고 이 시점까지의 모든 로그 레코드를 디스크에 반영시킨다. 이 방식은 트랜잭션의 응답시간(response time)이 길어진다는 단점이 있다. 또한 트랜잭션의 종료시에 디스크 액세스가 발생하므로 트랜잭션 종료시의 오버헤드가 상대적으로 더욱 크게 부각된다. 그러므로 주기억 장치 저장 시스템 고유의 장점을 살리는데에는 적합하지 못하다[21].

Group Commit

Group commit[16]은 commit 로그 레코드가 존재하는 로그 버퍼의 페이지가 가득 찰 때까지 트랜잭션의 commit를 보류하였다가 그 페이지가 가득 차면 디스크에 반영하고 보류중이던 트랜잭션들을 모두 commit하는 방법이다. 이 방법은 로그 버퍼 페이지가 디스크에 반영되는 횟수를 줄임으로써 전체 시스템의 성능을 개선시킬 수 있으나, 트랜잭션 commit 로그 레코드가 존재하는 로그 버퍼 페이지가 가득찰 때까지는 그 트랜잭션의 commit를 유보해야 하므로 응답시간은 Immediate commit 기법보다 오히려 길어진다. 또한 트랜잭션이 액세스한 모든 데이타에 대하여 commit시점까지 로크를 유지해야 하므로 동시성이 저하된다.

Precommit

Group commit의 문제점 중 하나는 트랜잭

션의 commit 로그 레코드가 존재하는 로그 버퍼 페이지가 가득 찰 때까지 트랜잭션의 commit를 유보함으로써 그 트랜잭션이 획득한 로크도 그대로 유지해야 한다는 것이었다. 이러한 문제점을 해결하기 위한 방법이 Precommit이다[13].

Precommit은 commit 로그 레코드가 일단 로그 버퍼에 기록되면 그 로그 버퍼가 디스크에 반영되지 않더라도 트랜잭션이 획득했던 모든 로크를 반환한다. 이 방법은 실제적으로는 엄정 이단계 로킹 규약(strict 2-phase locking protocol)을 준수하는 것이므로 데이타의 일관성을 유지하는데에는 문제가 없다. 예를 들어, 트랜잭션 A가 로크를 반환한 데이타를 트랜잭션 B가 액세스한다고 하더라도 B의 commit 로그 레코드는 A의 commit 로그 레코드 이후에 기록된다. 따라서 A의 commit 로그 레코드가 디스크에 반영되기 전에 시스템 오류가 발생되면 A뿐만 아니라 B도 모두 철회되므로 데이타를 일관성 있게 복구할 수 있다. 이 방법은 동시성을 높임으로써 전체 시스템의 성능을 개선시킬 수 있다.

안정된 주기억 장치(stable main memory)의 사용

이 방법은 안정된 주기억 장치[4]를 로그 버퍼로 사용하여 디스크 액세스로 인한 오버헤드를 피하는 방법이다. 안정된 주기억 장치는 일반 주기억 장치에 비하여 가격이 비싸고 속도가 떨어지지만 시스템 오류가 발생한 경우에도 저장된 내용이 보존되는 장점이 있다. 따라서 트랜잭션의 종료시에도 디스크 액세스를 유발하지 않으므로 응답시간을 크게 단축시킬 수 있다[16, 13].

안정된 주기억 장치를 로그 버퍼로 사용하는 경우 로그 버퍼의 크기에 제한이 있으므로 일반 트랜잭션과는 별도로 로그 버퍼의 내용을 디스크에 반영하여야 하지만 이에 수행되는 시간은 응답시간에 영향을 미치지 않는다. 이러한 장점으로 인하여 많은 주기억 장치 데이타베이스 시스템에서 안정된 로그 버퍼를 사용하고 있다.

2.4.2 체크포인트시(checkpoint)의 동작

주기억 장치 저장 시스템에서는 체크포인트시에 로그 레코드 뿐만아니라 변경이 가해진 주기억 장치내의 데이타 부분을 모두 디스크에 반영한다[13]. 그 이유는 가능한 디스크내에 최신의 데이타베이스를 유지함으로써 회복을 위한 재시동시의 오버헤드를 줄이기 위한 것이다[21]. 전체 시스템을 정지시킨 상태에서 체크포인트 동작을 수행하는 방법도 있을 수 있으나 이러한 경우 다른 트랜잭션들이 대기해야 하므로 일반적으로 디스크 기반 데이타베이스 시스템의 WAL 기법에서 사용된 퍼지 체크포인트를 사용한다[17].

2.4.3 회복을 위한 재시동시의 동작

주기억 장치 저장 시스템에서 회복을 위한 재시동은 안전하게 보존된 로그 레코드들과 가장 최근에 체크포인트된 디스크내의 데이타베이스를 기반으로 수행된다[16]. 수행순서는 먼저 디스크내의 가장 최근에 체크포인트 되었던 데이타베이스를 주기억 장치에 로드한 후 체크포인트 레코드 이후에 기록된 로그 레코드들을 적용함으로써 디스크 기반 저장 시스템 회복 기법의 재시동 동작과 유사하게 진행된다. 단지 회복 동작 중에 다시 시스템 오류가 발생하는 경우에는 디스크 내의 백업용 데이타베이스에는 영향을 미치지 않으므로 첫 시스템 오류가 발생하였을 때와 같은 재시동 동작을 수행하면 된다[21].

이외에도 재시동 시에 모든 데이타베이를 복구하는 것이 아니라 실제로 데이타가 액세스되는 순간에 로그 레코드를 읽어 데이타를 복구하는 방법[7], 디스크 배열 장치(disk array)를 사용하여 디스크 액세스 시간을 단축하는 방법[7] 등이 제안되어 있다.

3. 주기억 장치 데이타베이스 시스템의 소개

주기억 장치 데이타베이스 시스템은 세계적으로도 아직 연구의 초기 단계에 있으며, 구현된 시스템도 많지 않다. 본 장에서는 이미 구현되었거나 설계만 수행된 여러 주기억 장치 데이타베이스 시스템의 특징에 대하여 간략하

게 소개한다.

3.1 설계만 수행된 시스템

MM-DBMS[12]

미국 Wisconsin 대학에서 설계된 시스템으로 관계형 데이타 모델을 지원한다. 색인 구조로 T-트리를 사용하며, 변형된 선형해싱 기법을 사용한다. 동시성 제어를 위하여 릴레이션에 대한 엄정 이단계 로킹 기법을 사용하고, 로깅을 위하여 안정된 주기억 장치를 사용한다. 백업을 전담하는 프로세서를 따로 두어 일반 트랜잭션과는 독립적으로 백업을 수행한다.

MARS MMDB[5]

Southern Methodist 대학에서 설계한 시스템으로 동시성 제어를 위해 릴레이션에 대한 엄정 이단계 로킹 기법을 사용하며, 회복기능을 위해 안정한 주기억 장치를 이용한 쉐도우 기법을 사용한다. 즉, 데이타베이스는 일반 주기억 장치와 안정한 주기억 장치에 각각 존재하며 트랜잭션이 데이타베이스를 갱신할 때에는 안정한 주기억 장치에 존재하는 데이타베이스를 갱신한다. 그리고 트랜잭션이 commit하는 시점에 갱신된 데이타가 안정한 주기억 장치로부터 일반 주기억 장치로 복사된다. 만일 트랜잭션이 abort하는 경우에는 안정한 주기억 장치에서 발생한 갱신 연산은 무시된다.

이 외에도 Prinston 대학에서 설계한 HALO (HArdware LOgging)[7]과 같은 주기억 장치 데이타베이스 시스템이 있다.

3.2 구현된 시스템

OBE(Office-By-Example)[20]

IBM에서 개발된 시스템으로 IBM370에서 수행되도록 설계되었다. 이 시스템에서 릴레이션은 튜플들의 연결 리스트로 구성되며, 튜플은 그 튜플이 갖는 속성 값을 가리키는 포인터들의 배열로 이루어져 있다. 그리고 색인 구조로 역색인(inverted index)를 제공한다. 역색인은 튜플을 가리키는 포인터를 그 튜플이 갖는 속성의 값에 따라 정렬한 배열구조를 제공한다.

TPK[7]

TPK는 Prinston 대학에서 개발된 멀티프로세서 주기억 장치 트랜잭션 처리 시스템 프로토타입으로 debit/credit와 같은 유형의 트랜잭션을 빠르게 처리할 수 있는 것으로 알려져 있다. 이 시스템은 트랜잭션들을 하나씩 순서적으로 수행하며, group commit을 사용하여 로그를 디스크에 반영하는 횟수를 줄이고, precommit도 제공한다.

Starburst[14]

Starburst는 IBM 알마덴 연구소에서 개발된 프로토타입 DBMS로서 주기억 장치 저장 구조를 제공한다. 색인 구조로 T-트리를 사용하며, 동시성 제어를 위하여 기본적으로는 릴레이션에 대한 로크를 사용하나 경우에 따라 튜플 단위의 로킹을 할 수 있도록 구현되었다.

이 외에도 Prinston 대학에서 개발한 System M[18], ETRI에서 연구 프로토타입으로 개발한 FLASH, IBM에서 개발되어 상용화된 IMS/VS Fast Path[8] 등이 있다.

4. 결 론

본 논문에서는 기존의 주기억 장치 데이타베이스 시스템과 주기억 장치 저장 시스템에 대하여 고찰하였다. 먼저, 주기억 장치 저장 시스템이 갖추어야할 기능과 그 구현 방법들을 살펴보았다. 주기억 장치를 관리하는 방법으로 Starburst에서의 주기억 장치 관리 기법과 버디 시스템에 대하여 기술하였고, 주기억 장치 색인 구조로 널리 사용 중인 T-트리를 설명하였다. 해싱 기법으로는 CBH, MLH, FSMH 등의 방법들을 간단하게 소개하였으며, 동시성 제어 기법과 회복 기법에 대하여 설명하였다. 다음으로 주기억 장치 데이타베이스 시스템으로써 설계만 수행된 시스템과 이미 구현된 시스템들에 관하여 간단히 살펴보았다.

본 논문은 주기억 장치 저장 시스템을 설계할 경우에 필요한 주요 기법들을 간단하게 소개하는 수준이었으며, 실제로 구현을 함에 있어서는 많은 문제점들이 대두될 것이다. 뿐만 아니라 주기억 장치 저장 시스템은 그 특성상 범용 시스템으로 설계하는 것보다는 응용 프로그램의 성격에 맞도록 특화하여 설계하는 것도 신중히 고려해보아야 할 것이다. 최근에는 국

내에서도 주기억 장치 데이타베이스 시스템에 대한 관심이 고조되고는 있으나 아직 미비한 단계이며, 보다 과학적이고 체계적인 연구가 수행되어야 한다. 국내에서도 빠른 시일 내에 훌륭한 주기억 장치 데이타베이스 관리 시스템이 출현할 것을 바라며, 많은 분들의 관심과 노력을 기대한다.

참고문헌

[1] Ammann, A, Hanrahan, M., and Krishnamurthy, R., "Design of a Memory Resident DBMS," In *Proc. IEEE COMPCON*, San Francisco, Feb. 1985.

[2] Analyti, A. and Pramanik, S., "Fast Search in Main Memory Databases," In *Proc. Intl. Conf. on Management of Data, ACM SIGMOD*, pp. 215-224, 1992.

[3] Comer, D., "The Ubiquitous B-tree," *ACM Computing Surveys*, Vol. 11, No. 2. pp. 121-137, 1979.

[4] DeWitt, D., Katz, R., Olken, F., Shapiro, L., Stonebraker, M., and Wood, D., "Implementation Techniques for Main Memory Database Systems," In Proc. *Intl. Conf. on Management of Data, ACM SIGMOD*, pp. 1-8, 1984.

[5] Eich, M., "MARS : The Design of a Main Memory Database Machine," In *Proc. 4th Intl. Workshop on Database Machines*, Oct. 1985, pp. 468-481.

[6] Fagin, R., Nievergelt, J., Pippenger, N., and Strong, H. R., "Extendible Hashing : A Fast Access Method for Dynamic Files," *ACM Trans. on Database Systems*, Vol. 4, No. 3, pp. 315-344, 1979.

[7] Garcia-Molina, H., "Main Memory Database Systems : An Overview," *IEEE Trans. on Knowledge and Data Engineering*, Vol. 4, No. 6, pp. 509-516, 1992.

[8] Gawlick. D. and Kinkade. D. "Varieties of Concurrency Control in IMS/VM Fast Path," *IEEE Data Engineering Bull.*, Vol. 8, No. 2, pp. 3-10, June 1985.

[9] Harry, R. L. and Larry Denenberg, *Data Structure and Their Algorithm*, Harper Collins Publishers, 1991.

[10] Knuth, D., *The Art of Programming-Sorting and Searching*, Addison-Wesley Publishing Co. Inc., 1973.

[11] Lehman, T. J. and Carey, M. J., "A Study of Index Structures for Main Memory Database Management Systems," In *Proc. Intl. Conf. on Very Large Data Bases*, pp. 294-303, May 1986.

[12] Lehman, T. J. and Carey, M. J., "Query Processing in Main Memory Database Management Systems," In *Proc. Intl. Conf. on Management of Data, ACM SIGMOD*, pp. 239-250, 1986.

[13] Lehman, T. J. and Carey, M. J., "A Recovery Algorithm for a High-Performance Memory-Resident Database System," In *Proc. Intl. Conf. on Management of Data, ACM SIGMOD*, pp. 104-107, 1987.

[14] Lehman, T. J. et al., "An Evaluation of Starburst's Memory Resident Storage Component," *IEEE Trans. on Knowledge and Data Engineering*, Vol. 4, No. 6, pp. 555-566, Dec. 1992.

[15] Litwin, W., "Linear Hashing : A New Tool for File and Table Addressing," In *Proc. 6th Intl. Conf. Very Large Data Bases*, Oct. 1980.

[16] Salem, K. and Garcia-Molina, H., Crash Recovery Machanisms for Main Storage Database Systems, CS-TR-034-86, Dept. of Computer Science, Prinston Univ. 1986.

[17] Salem, K. and Garcia-Molina, H., "Checkpointing Memory-Resident Databases," In *Proc. Intl. Conf. on Data Engineering, IEEE*, pp. 452-462, 1989.

[18] Salem, K. and Garcia-Molina, H., "System M : A Transaction Processing Testbed for Memory Resident Data," *IEEE Trans. on Knowledge and Data Engineering*, Vol. 2, No. 1, pp. 161-172, Mar. 1990.

[19] Whang K.-Y. et al., "Office-by-Example : An Integrated Office System and Da-

tabase Manager," *ACM Trans. on Office Information Systems*, Vol. 15, No. 4, pp. 393-427, Oct. 1987.

[20] Whang K.-Y. and Krishnamurthy, R., "Query Optimization in a Memory Resident Domain Relational Calculus Database Systems," *ACM Trans. on Database Systems*, Vol. 15, No. 1, pp. 67-95, Mar. 1990.

[21] 황규영, 김상욱, "주기억 장치 데이타베이스 시스템에서의 파손 회복 기법," 정보과학회지, 제 11권, 제 1호, pp. 47-60, 1993년 2월.

황 규 영

1973 서울대학교 전자과 졸업(B.S.)
1975 한국과학기술원 전기 및 전자공학과 졸업(M.S.)
1982 Stanford University(전산학, M.S.)
1983 Stanford University(전산학, Ph.D.)
1975～1978 국방과학연구소 선임연구원
1983～1990 IBM T.J. Watson Research Center, Research Staff Member
1990～현재 인공지능연구센터 데이타베이스 및 멀티미디어 연구실장
1990～현재 한국과학기술원 전산학과 교수
1991 Visiting Professor, Stanford University
1992 Visiting Associate Professor, Georgia Institute of Technology
1993 Hewlett-Packard Laboratories 기술자문(Palo Alto)
1992～1994 한국정보과학회 데이타베이스연구회(SIGDB) 운영위원장
1993～현재 체신부 통신진흥협의회 DB산업육성분과 위원장
1995 한국정보과학회 이사 겸 논문지 편집위원장
관심분야 : 데이타베이스 시스템, 멀티미디어 데이타베이스, 객체지향 데이타베이스, 하이퍼미디어, GIS, 분산 데이타베이스 및 Client/Server 기술, 연역 데이타베이스, 공학 데이타베이스, 사무자동화, CASE, 전문가 시스템

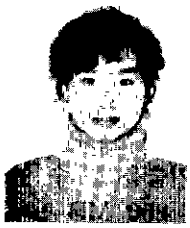
장 지 웅

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