□ 기술해설 □

M²RTSS : 주메모리 실시간 저장 시스템

서울대학교 차상균* · 박장호** · 박병대** · 이성직**

목 차

1. 서 론
2. 주메모리 DBMS의 특징
 2.1 저장 관리
 2.2 동시성 제어
 2.3 회복 관리
3. 관련 시스템
 3.1 System M
 3.2 Starburst
 3.3 Dali
 3.4 SmallBase
 3.5 ClustRa
4. M²RTSS : 주메모리 실시간 저장 시스템
 4.1 시스템 개요
 4.2 저장 관리
 4.3 동시성 제어
 4.4 회복 관리
5. 결 론

1. 서 론

컴퓨터 하드웨어 기술의 발전은 소프트웨어 측면에서 이제까지는 불가능하다고 생각되었던 기능과 구조를 가능하게 하고 있다. 대표적 예로써 컴퓨터 메모리와 관련된 기술을 들 수 있는데, DRAM 부문에서는 현재 256 MB 칩이 개발되어 있고, 앞으로 GB 혹은 그 이상의 대용량 칩도 개발될 것으로 기대되고 있다. 비휘발성 반도체 메모리 부문에서는 블록 단위로 접근 가능한 플래시 메모리가 상품화되면서 자기 기록 매체의 역할을 부분적으로 대신하고 있다. 한편, 현재의 보편적인 32 비트 주소 체계 CPU에서 4 GB로 제한되던 주메모리 주소 영역은 최근 64 비트 주소 체계의 CPU가 출시됨에 따라 조만간 거의 무한의 영역으로 확장될 것이다.

위에서 언급한 하드웨어 기술의 발전은 데이타베이스 기술에도 큰 영향을 미치고 있는데, 대부분의 상용 DBMS는 버퍼를 확장하는 방식으로 증가된 메모리를 활용하고 있다. 그러나, 컴퓨터 메모리 용량이 최대 수 MB이던 시기에 개발된 전통적 DBMS 구조와 이러한 국부적인 수정으로 얻을 수 있는 성능의 한계가 인식되면서 최근 주메모리 실시간 DBMS에 대한 연구 개발이 활성화되고 있다[1, 2]. 주메모리 실시간 DBMS는 디스크에 비하여 상대적으로 빠른 데이타 접근 시간과 균일한 성능 분포 특성으로 인하여 실시간 고성능을 요하는 응용 분야에서 각광을 받고 있으며, 여타 분야에서도 데이타베이스 응용 프로그램의 응답 시간을 단축시켜 더 많은 서비스를 제공할 수 있을 것으로 기대된다.

본 논문에서는 주메모리 실시간 DBMS의 특징 및 연구 동향을 소개하고, 서울대학교에서 연구 개발 중인 주메모리 실시간 저장 시스템 M²RTSS(Main-Memory Real-Time Storage System)에 대하여 기술한다. M²RTSS의 데이타베이스는 가상 메모리 내에 탑재되며, T-트리와 ECBH 인덱스를 이용하여 접근할 수 있다. 동시성 제어는 잠금 기반 방식을 채용하고 잠금 충돌 발생시 우선 순위 상속 방식을 이용한

*종신회원
**학생회원

다. 회복 관리의 경우에는 ARIES 방식[3]을 주메모리 구조에 맞게 변경하였으며, 로깅시 성능 향상을 위하여 비휘발성 메모리를 사용할 수 있도록 한다. M²RTSS 서버 프로세스는 시스템 성능 향상을 위하여 다중 쓰레드 구조를 갖는데, 트랜잭션 처리와 회복 관리를 위한 디스크 접근을 독립적으로 수행함으로써 병렬 하드웨어 구조로 쉽게 확장할 수 있도록 했다.

논문의 구성은 다음과 같다. 2장에서는 주메모리 실시간 DBMS를 저장 관리, 동시성 제어, 회복 관리의 측면에서 디스크 기반 DBMS와 비교하고, 주메모리 DBMS 설계시 고려할 사항을 기술한다. 3장에서는 관련 시스템을 기술하고, 4장에서는 M²RTSS의 구체적 설계 내용과 구현 현황에 대하여 기술한다. 마지막으로 5장에서 결론을 맺는다.

2. 주메모리 DBMS의 특징

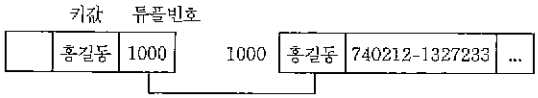
2.1 저장 관리

디스크 I/O 속도가 CPU 연산 속도나 주메모리에의 접근 속도에 비하여 현저하게 느리기 때문에, 디스크 기반 DBMS에서는 데이타를 클러스터링하여 I/O를 줄이는 것이 필수적이다. 반면 주메모리 상에서는 데이타의 분산이 접근 속도에 큰 영향을 미치지 않기 때문에 데이타의 클러스터링이 상대적으로 중요하지 않다.

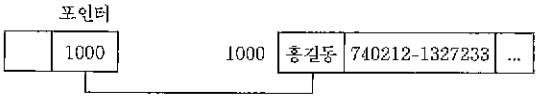
디스크 기반 DBMS는 B-트리와 같이 디스크 I/O를 최소화하기 위해 설계된 인덱스를 사용하지만, 주메모리 DBMS에서는 메모리 공간을 효율적으로 사용하고 접근 비용이 저렴한 인덱스가 요구된다. 주메모리 DBMS를 위한 인덱스로는 수정된 선형 해싱 기법과 T-트리 인덱스가 효율적인 것으로 알려져 있다[4].

디스크 기반 DBMS의 인덱스 노드에는 레코드의 키값과 논리적 포인터가 저장되므로 키를 구성하는 필드가 달라지면 인덱스 노드의 구조도 달라진다. 반면 주메모리 DBMS의 경우에는 레코드를 가리키는 포인터만 저장해도 키값을 구할 수 있기 때문에 인덱스 노드의 구조가 키의 구성에 관계없이 동일하다.

• 디스크 기반 DBMS의 인덱스 노드



• 주메모리 DBMS의 인덱스 노드



2.2 동시성 제어

2.2.1 주메모리 DBMS에서의 동시성 제어

주메모리에 대한 접근은 디스크에 대한 접근에 비하여 매우 빠르기 때문에 트랜잭션의 수행이 빠르게 완료될 수 있다. 따라서, 잠금의 단위를 크게 잡는 것이 바람직하다. 잠금의 단위가 크면 충돌의 발생 가능성은 크게 되지만, 이로 인한 blocking 지연이 짧기 때문에 큰 문제가 되지 않는다. 일부 시스템에서는 잠금의 단위를 전체 데이타베이스로 삼아 트랜잭션을 순차적으로 수행할 것을 제안하는 경우도 있다[1]. 이와 같이 할 경우 잠금의 설정 및 해제, 데드락의 검출 및 처리 등과 같은 오버헤드가 제거되므로 오히려 병행 수행하는 경우보다 실행 효율을 높일 수 있다. 하지만, 이러한 장점은 긴 트랜잭션이 존재할 경우 더 이상 장점이 되지 못한다. 긴 트랜잭션의 수행과 함께 짧은 트랜잭션이 수행될 수 있도록 하기 위해서는 동시성 제어가 필요하게 된다. 또한 다중 프로세서 시스템의 경우에는 모든 트랜잭션이 짧다 하더라도 동시에 수행될 수 있기 때문에 동시성 제어가 요구된다.

주메모리 DBMS에서는 잠금의 대상이 되는 데이타가 모두 메모리 상에 존재하기 때문에, 잠금의 실제 구현도 디스크 기반 DBMS와는 다른 방식이 가능하다. 기존의 디스크 기반 DBMS에서는 잠금에 대한 정보를 저장하기 위하여 별도로 해쉬 테이블을 만들어 정보를 관리한다. 주메모리 DBMS의 경우에는 각각의 데이타에 몇 비트를 할당하여 잠금 설정 상태를 표시할 수 있다. 따라서, 해쉬 테이블에 대한 검색 없이 작은 수의 연산만으로 잠금의 설정 및 해제를 수행할 수 있게 된다.

[1]은 지금까지 제안되거나 혹은 실제 구현

된 주메모리 데이타베이스 시스템에 대하여 요약, 기술하고 있다. 이들 시스템의 동시성 제어 측면을 살펴보면, MM-DBMS, MARS, System M등은 릴레이션 단위로 2단계 잠금(2PL)을 지원하며, TPK는 트랜잭션의 순차적 수행을 가정한다. System M의 경우에는 트랜잭션의 동시 수행을 지원하지만, 동시에 수행되는 트랜잭션의 수를 작게 함으로써 충돌이 일어날 가능성을 줄인다. IMS/VS Fast Path의 경우에는 앞의 시스템들과는 달리 자주 접근되는 데이타만을 주메모리에 상주시키기 때문에, 잠금을 레코드 단위로 설정하여 데이타에 대한 충돌을 피하도록 하고 있다.

2.2.2 실시간 DBMS에서의 동시성 제어

주메모리 DBMS와는 달리 실시간 DBMS에서의 동시성 제어에 대한 연구는 활발히 진행되어 왔다. 특히, 실시간 DBMS에서의 동시성 제어는 트랜잭션 스케쥴링과 관련되어 함께 연구되는 경우가 많다. 실시간 트랜잭션 스케쥴링의 목적은 최대한 많은 트랜잭션이 각각의 종료시간(deadline) 안에 그 수행을 완료하도록 하는 것이므로, 기존의 동시성 제어 방식들도 이를 위하여 수정되어야 한다. 현재의 연구 동향을 살펴보면, 기존의 잠금 기반 및 낙관적 제어 방식(optimistic concurrency control)을 실시간 특성에 맞도록 수정한 방식이 많이 적용되고 있다[2, 5, 6].

본 절에서는 잠금 기반 및 낙관적 제어 방식에 대한 조사 연구 내용을 중심으로 기술한다. 특히, 구현의 간단함으로 인하여 기존의 DBMS에 많이 적용되고 있는 잠금 기반 방식에 대하여 자세히 설명한다.

(1) 잠금 기반 동시성 제어

2PL은 기존의 DBMS에서 가장 일반적인 동시성 제어 방식이다. 이를 채용할 경우 트랜잭션 T_H가 잠금을 설정한 데이터를 트랜잭션 T_R이 접근하려고 할 때, 함께 잠금을 설정할 수 없는 경우에는 잠금이 해제될 때까지 기다려야 한다.

2PL을 실시간 DBMS의 동시성 제어 방법으로 채용한 연구들에서는 2PL이 낙관적 제어 방식에 비하여 실시간 데이타베이스에 더 적합하다고 주장한다[5]. 낙관적 제어 방식을 이용할 경우, 일단 각 트랜잭션이 수행된 후 데이타의 충돌이 있는지 여부를 판단하고 충돌이 있는 경우 그 트랜잭션을 abort하게 되므로, 충돌이 자주 발생한다면 시스템 자원의 낭비를 초래한다. 반면, 2PL의 경우 데이타에 대한 충돌을 트랜잭션 수행 초기에 감지하여 해소시키기 때문에 시스템 자원 낭비를 줄일 수 있게 된다.

하지만 2PL을 실시간 DBMS에 바로 적용할 수는 없는데, 이는 우선순위 역행 현상과 데드락의 가능성이 존재하기 때문이다. 앞에서 T_R의 우선순위가 T_H의 우선순위보다 높다면, T_R은 우선순위가 높음에도 불구하고 T_H의 수행이 완료될 때까지 기다려야 한다. 이와 같이 우선순위가 높은 트랜잭션이 우선순위가 낮은 트랜잭션의 수행을 기다리는 현상을 우선순위 역행 현상이라 한다. 데드락의 경우에는 데드락 발생시 어느 트랜잭션을 abort시킬 것인가가 문제가 된다. 가능한 방법을 몇 가지 나열하면 다음과 같다.

- 종료시간을 넘긴 트랜잭션을 abort시킨다.
- 가장 긴 종료시간을 가진 트랜잭션을 abort시킨다.
- 가장 중요하지 않은(least critical) 트랜잭션을 abort시킨다.

우선순위 상속(PI : Priority Inheritance)

우선순위 역행 현상이 발생했을 때 이를 해결하기 위한 방안으로 T_R의 우선순위 $pr(T_R)$가 T_H의 우선순위 $pr(T_H)$보다 높을 때, T_H의 우선순위를 T_R의 그것과 같게 만들어 주는 방법을 고려할 수 있다[7]. 이와 같이 함으로써 T_H의 수행이 빨라질 수 있고, 따라서 잡고 있던 잠금을 빨리 해제할 가능성이 높아진다. 이는 결과적으로 T_R의 수행 시작을 앞당길 수 있는 효과를 갖는다. 하지만 PI 방식은 여전히 우선순위가 낮은 트랜잭션에 의해 우선순위가 높은 트랜잭션이 기다려야 한다는 단점을 가진다.

High Priority(HP)

HP 방식은 우선순위가 낮은 트랜잭션이 잠금을 잡고 있을 때 우선순위가 높은 트랜잭션이 잠금 설정을 시도하면, 우선순위가 낮은 트

랜잭션을 abort함으로써 충돌을 해소한다[5]. 이는 트랜잭션을 abort시키는 비용이 높지 않을 때 특히 유용하며, 알고리듬이 간단하다는 측면과 항상 우선순위가 높은 트랜잭션부터 수행될 수 있다는 장점을 갖는다. 반면 우선순위가 낮은 트랜잭션은 계속적으로 abort되어 재실행을 반복하다가 결국 수행되지 못하는 문제가 발생할 수 있다. 이를 개선하기 위하여 abort되는 트랜잭션의 우선순위를 높여 주는 방안을 고려할 수 있는데, abort되는 트랜잭션의 새로운 우선순위가 이를 abort시킨 트랜잭션의 우선순위보다 높게 부여될 경우, 반복적 재실행이 발생하게 된다는 점에 주의해야 한다.

Conditional Restart(CR)

HP 방식은 우선순위가 낮은 트랜잭션을 무조건 abort시킴으로써 시스템 자원을 낭비하게 되는 측면이 있다. 만약 T_R의 slackness가 T_H의 잔여 수행시간보다 크다고 하면, T_H를 abort시키지 않고 수행한 후 T_R을 시작하더라도 종료시간 내에 수행이 가능하다. 이 방식을 CR이라고 한다[5]. 이 방식의 문제점으로는 먼저, T_R의 slackness와 T_H의 잔여 수행시간을 알아야만 알고리듬을 적용할 수 있다는 점이다. 둘째로, 알고리듬에 의해 T_R의 slackness 내에 T_H의 수행을 마칠 수 있다고 판단되어 계속 수행하였는데, 우선순위가 높은 또 다른 트랜잭션과 충돌이 발생하게 되면, 앞의 판단은 무의미하게 된다는 문제가 있다.

(2) 낙관적 동시성 제어

실시간 DBMS의 동시성 제어를 위하여 낙관적 방식을 제안하는 연구들은 잠금 기반 방식과는 상반된 주장을 편다[6]. 잠금 기반 방식은 잠금을 설정하기 위하여 기다려야 하지만, 낙관적 방식에서는 이러한 지연 현상이 없기 때문에 실시간 DBMS에 더 적합하다고 한다. 우선순위가 높은 트랜잭션에 의해 우선순위가 낮은 트랜잭션이 연속적으로 abort되어 재실행되는 일이 없으므로, 시스템 자원 측면에서도 유리하다고 한다. 잠금 기반 방식을 제안하는 논문에서는 낙관적 방식이 충돌이 일어날 수 있는 트랜잭션을 일단 수행하고 충돌 검사를 나중에 하기 때문에 시스템 자원을 낭비할 수 있다고 하는데, 이에 대해 데이타의 충돌이 많지 않은 경우에는 문제가 되지 않는다고 한다.

낙관적 동시성 제어 방식을 채용할 경우 트랜잭션의 수행은 세 단계 — (1) 읽기 단계, (2) validation 단계, (3) 쓰기 단계-로 나누어진다. 읽기 단계 동안에는 데이타를 메모리로 읽어 들여 연산이 수행된다. 그 결과 새로운 값이 얻어지지만 쓰기 단계가 되기 전에는 데이타베이스에 반영되지 않는다. 연산이 모두 수행된 후 validation 단계에서는 다음과 같은 조건이 만족하는지 조사함으로써 트랜잭션 T_i와 T_j가 순차적인지 체크한다.

- R/W rule : T_i에 의해 갱신된 데이타를 T_j가 그 전에 읽어서는 안된다.
- W/W rule : T_i의 갱신이 T_j가 갱신한 내용을 덮어 쓰면 안된다.

Validation 결과 rule이 위배된 경우에는 충돌 해소를 위하여 적당한 트랜잭션이 선택되어 abort된다. 충돌 해소 방식은 다수 제안되었는데[8], 대표적인 것으로 Haritsa 등에 의해 제안된 priority Wait 방식과 WAIT-50 방식 등이 있다.

2.3 회복 관리

회복 관리 시스템은 트랜잭션 오류나 시스템 이상이 발생했을 경우 데이타베이스의 일관성(consistency)과 지속성(durability)을 보장해야 한다[9, 10]. 이를 위해 데이타베이스의 변경에 대한 로그가 기록되어야 하고, 시스템 재실행의 시간을 줄이기 위해 주기적으로 체크포인팅을 수행해야 한다.

주메모리 DBMS의 회복 관리 시스템은 디스크와는 다른 주메모리의 특성에 맞게 설계되어야 한다. 시스템 이상이 발생하면 주메모리 상의 모든 데이타를 잃어버리게 되므로 디스크 기반 시스템과는 다른 회복 기법을 필요로 한다. 체크포인팅시 변경된 페이지를 디스크에 옮길 필요가 없는 디스크 기반 시스템과 달리, 주메모리 구조에서는 변경된 페이지를 반드시 비휘발성 저장 매체에 옮겨야 한다[11, 12].

본 절에서는 회복 관리 기법을 몇가지 기준에 의해 분류하고, 이러한 분류가 주메모리

DBMS에서 어떻게 변경되어야 하는지 기술한다.

2.3.1 회복 관리 기법 분류

회복 관리 기법은 데이타 변경 방식, 버퍼 관리 방식, commit 과정, 체크포인팅 방식에 따라 다음과 같이 분류할 수 있다[9, 13].

- 데이타 변경 방식에 따라
  - Update In Place : 데이타 변경시 기존의 데이타 위에 새로운 내용을 덮어 쓴다. Undo를 위해 변경 이전의 데이타를 로그 레코드에 기록해야 한다.
  - Shadowing : 데이타 변경시 기존의 데이타 페이지가 아니라 shadow 페이지를 변경한다. Undo를 할 필요가 없다는 장점이 있지만, 메모리를 많이 사용하는 단점이 있다.
- 버퍼 관리 방식에 따라
  - Steal : 변경된 데이타가 해당 트랜잭션이 commit하기 이전에 디스크에 옮겨질 수 있다. 트랜잭션이 abort할 경우 undo를 수행해야 한다.
  - No Steal : 변경된 데이타는 해당 트랜잭션이 commit한 이후에만 디스크에 옮겨질 수 있다. 트랜잭션 undo가 필요 없으나, 버퍼의 사용이 비효율적이기 쉽다.
- Commit 과정에 따라
  - Force : commit시 해당 트랜잭션에 의해 변경된 모든 데이타 페이지를 디스크에 옮긴다. 트랜잭션 redo가 필요 없으나, commit과정이 길어진다.
  - No Force : commit시 해당 트랜잭션에 의해 변경된 페이지를 디스크에 옮기지 않는다.
- 체크포인팅 방식에 따라
  - 퍼지 체크포인팅 : 체크포인팅은 일반 트랜잭션의 수행과 독립적으로 수행된다. 데이타베이스의 일관성을 보장할 수 없지만, Write Ahead 로깅 프로토콜을 보장하면 시스템 재실행시 이를 복구할 수 있다[3],[13].
  - 트랜잭션 Consistent 체크포인팅 : 체크포인팅시에 일반 트랜잭션의 수행을 멈춘다. 데이타베이스의 일관성이 보장되지만 시스템의 성능이 저하되기 쉽다. 체크포인팅과 동기 시키는 단위가 트랜잭션이 아니라 트랜잭션의 일부인 action일 경우 이를 action consistent 체크포인팅이라고 한다.

2.3.2 주메모리 DBMS에서의 회복 관리 기법

디스크 기반 데이타베이스에서 주데이타베이스가 디스크에 존재하고 필요에 따라 해당 페이지를 메모리 버퍼에 옮기는 반면, 주메모리 구조에서는 주데이타베이스가 메모리에 상주한다. 이러한 구조상의 차이로 인해 앞에서 살펴본 회복 기법들은 그대로 주메모리 DBMS에 적용될 수 없다.

디스크 기반 회복 기법을 분류하는데 적용한 기준을 주메모리 DBMS 입장에서 살펴보면 다음과 같다.

- 데이타 변경 방식 : 마찬가지로 구분할 수 있다.
- 버퍼 관리 방식에 따라 : 데이타베이스가 주메모리에 상주하므로 Steal, No Steal의 구분은 의미가 없다. Shadowing을 사용하지 않는 한 모든 변경은 직접 메모리 상의 주데이타베이스에서 이루어지므로 undo를 위한 정보가 페이지 변경 이전에 로그에 기록되어야 한다.
- Commit 과정에 따라 : 데이타베이스가 주메모리에 상주하므로 Force, No Force의 구분도 의미가 없다. 이 경우 해당 로그 레코드를 어느 시점에 stable 로그에 옮기는가에 따라 다음과 같이 구분할 수는 있다.
  - Force-Log-At-Commit : commit시에 해당 트랜잭션의 모든 로그 레코드를 stable 로그에 옮긴다. 디스크 IO가 빈번하게 발생하므로 주메모리 DBMS의 성능을 크게 저하시킬 수 있다.
  - Group Commit : 로그를 페이지 단위로 stable 로그에 옮기는 방법이다. I/O 횟수가 감소하는 장점이 있으나, 개별 트랜잭션의 commit이 늦춰질 수

있다.

- 체크포인팅 방식에 따라 : 세가지 구분은 유효하다. 그러나, 디스크 기반 시스템에서 체크포인팅시에 데이타베이스의 변경된 페이지를 디스크에 옮길 필요가 없는 반면에 주메모리 구조에서는 반드시 변경된 페이지를 디스크에 옮겨야 한다.

3. 관련 시스템

3.1 System M

System M[11]은 Princeton 대학에서 개발한 주메모리 DBMS testbed이다. System M은 응용 프로그램을 시뮬레이션하는 메시지 서버, 트랜잭션의 관리를 담당하는 트랜잭션 서버, 회복 관리를 위한 로그 서버와 체크포인트 서버, 동시성 제어를 위한 잠금 서버로 이루어져 있다. 데이타베이스는 공유 가상 메모리에 적재되며, 디스크 I/O의 성능을 높이기 위하여 raw 디바이스 인터페이스를 사용한다.

로그 서버와 체크포인트 서버는 주메모리 구조에 적합한 회복 알고리듬을 찾기 위하여 다양한 옵션을 지원한다. [11]에서는 퍼지 체크포인팅, black/white 체크포인팅, copy-on-update 체크포인팅 기법을 비교하여 퍼지 체크포인팅 기법이 가장 우수한 성능을 나타냄을 보였다.

3.2 Starburst

Starburst[14]는 IBM Almanden 연구소에서 개발한 extensible 관계형 DBMS이다. Starburst Memory-Resident Storage Component는 기존의 디스크 기반 Starburst 시스템을 확장한 것으로 주메모리상의 테이블을 관리하는 MMM(Main Memory Relation Manager)와 T-트리 인덱스, 선형 해싱 인덱스로 구성된다. [14]에 의하면 MMM은 기존의 디스크 기반 Starburst에서 모든 데이타를 메모리 버퍼에 적재하였을 때보다 우수한 성능을 나타낸다.

MMM은 주메모리상의 데이타베이스를 세그먼트와 파티션으로 나누어 관리하며, 세그먼트 번호, 파티션 번호, 파티션 내의 슬롯 번호로 이루어진 RID를 이용하여 레코드에 접근한다. 동시성 제어를 위해서는 별도의 잠금 테이블을 만들지 않고 직접 데이타에 잠금 필드를 두어 관리한다. 회복 관리는 기존의 디스크 기반 Starburst에 사용된 기법을 사용하였다.

3.3 Dalí

Dalí [15]는 AT&T에서 개발한 주메모리 저장 시스템이다. 데이타베이스는 공유 가상 메모리상에 적재되며 여러 개의 파티션으로 구분된다. Dal 는 회복 관리자, 메모리 관리자, 동시성 제어 관리자, 트랜잭션 관리자, 아이템 관리자로 이루어진다. 아이템은 저장 시스템이 관리하는 최소 저장 단위로써 헤더와 몸체로 이루어지며 관계형 DBMS의 개별 레코드에 해당한다.

Dalí는 [16]의 회복 기법을 사용하여, redo 로그만을 디스크에 기록함으로써 로그의 양과 디스크 접근 시간을 줄이도록 했다. 이를 위해 개별 트랜잭션은 별도의 로그 버퍼를 관리해야 하고 재실행 시간을 줄일 수 있는 1단계 알고리듬을 사용한다. 동시성 제어는 2PL을 기반으로 하며, 새로운 잠금 모드를 추가할 수 있는 인터페이스를 제공한다.

3.4 SmallBase

SmallBase[17]는 HP에서 개발하고 있는 주메모리 관계형 DBMS이다. SmallBase는 SQL을 입력 받아 실행 가능한 코드로 변환해 주는 컴파일러, 컴파일된 코드를 실행하는 수행 엔진, 주메모리상의 데이타를 관리하는 저장 관리자로 구성되며, 이들 모듈에 대한 응용 프로그램 인터페이스를 제공한다. 데이타에 대한 빠른 접근을 위해 해쉬 인덱스와 T-트리 인덱스를 제공하고, 다중 사용자 환경을 지원하기 위해 공유 메모리를 사용한다.

3.5 ClustRa

ClustRa[18]는 이동통신 서비스에 사용하기 위한 주메모리 DBMS이다. 동시성 제어를 위해 2PL을 사용하고 2-level 로깅을 수행한다. 주메모리 구조의 장점을 살리기 위해 T-트리 인덱스 등을 지원하는 다른 주메모리 DBMS

와는 달리 기존의 B-트리 인덱스를 지원한다. 또한 모든 데이타베이스가 주메모리에 상주하는 것이 아니라 실시간 제한 조건을 갖는 부분만 주메모리에 상주시키도록 하고 있다.

4. M²RTSS : 주메모리 실시간 저장 시스템

본 장에서는 주메모리 실시간 저장 시스템인 M²RTSS의 객체지향 설계 및 구현에 대하여 기술한다. M²RTSS는 한국전자통신연구소에서 개발한 주메모리 저장 시스템인 Mr. RT 1.0과 본 연구실의 주메모리 객체지향 DBMS 개념 설계의 경험을 바탕으로 설계 및 구현되고 있다. M²RTSS는 새로운 연구 결과를 수용하고 응용 분야의 변화에 쉽게 적응할 수 있도록 하기 위해 객체지향 기법을 사용하여 설계 및 구현하였다. 예를 들어, 비휘발성 메모리의 사용은 로깅 방식에 변화를 초래할 수 있는데, 이러한 변화는 상속과 캡슐화 등의 객체지향 개념을 적용하여 쉽게 수용할 수 있을 것으로 기대된다.

4.1 시스템 개요

M²RTSS는 데이타의 물리적 저장, 동시성 제어, 회복 관리 등과 같은 DBMS의 기본 기능과 함께 실시간 트랜잭션을 지원하기 위한 스케쥴링 기능을 지원한다. 이들 각각의 기능은 C++의 객체로 구현되는 관리자(manager)들이 담당하게 되며, 시스템의 성능을 향상시키기 위하여 다중 쓰레드 구조를 갖는다.

4.1.1 구성 요소

M²RTSS는 데이타베이스마다 하나의 서버 프로세스를 통해 사용자 프로세스에 대한 서비스를 제공한다. M²RTSS에서 트랜잭션은 트랜잭션 begin으로부터 commit 또는 abort에 이르는 일련의 action(a sequence of actions)으로 정의되는데, 데이타에 대한 잠금 시도, 데이타 검색, 갱신 등 데이타베이스에 대한 모든 연산들이 action으로 정의된다. 그림 1은 M²RTSS 서버 프로세스의 구성 요소를 보여 준다.

트랜잭션 관리자는 다른 관리자들의 기능을

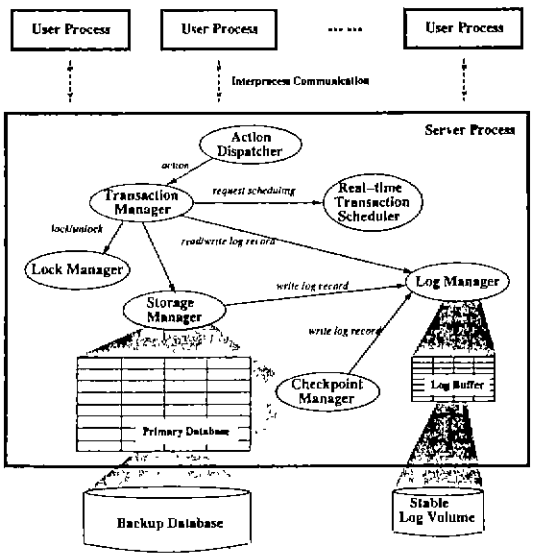


그림 1 M²RTSS 구성 요소

사용하여 트랜잭션의 begin, commit, abort를 처리한다. 실시간 트랜잭션 스케쥴러는 트랜잭션간의 실행 순서를 결정하고, 저장 관리자는 메모리 상의 데이타베이스에 대한 생성/변경/삭제 등의 기능을 제공한다. 회복 관리는 로그 관리자와 체크포인트 관리자에 의해 이루어지며, 잠금 관리자는 동시성 제어를 담당한다.

4.1.2 쓰레드 구조

그림 2는 서버 프로세스의 쓰레드 구성을 보여 준다. 쓰레드 생성시의 오버헤드를 줄이기 위해 시스템 시작시에 Action 수행 쓰레드 Pool을 생성하고, 이들 쓰레드가 동시에 여러 action을 처리할 수 있도록 한다. 그림에서 Action Queue는 Action Dispatcher 쓰레드와 Action 수행 쓰레드의 통신을 위해 사용된다.

Action Dispatcher 쓰레드는 응용 프로그램으로부터 메시지를 받아 이를 action 객체로 변환하고, 이를 사용 가능한 Action 수행 쓰레드에 분배하는 일을 담당한다. 이때, Input Queue가 비어 있거나, Action 수행 쓰레드 Pool 내에 사용 가능한 쓰레드가 없을 경우에는 sleep하게 된다.

Action 수행 쓰레드는 개별 action을 처리하는 역할을 한다. 이때, 쓰레드의 개수가 너무

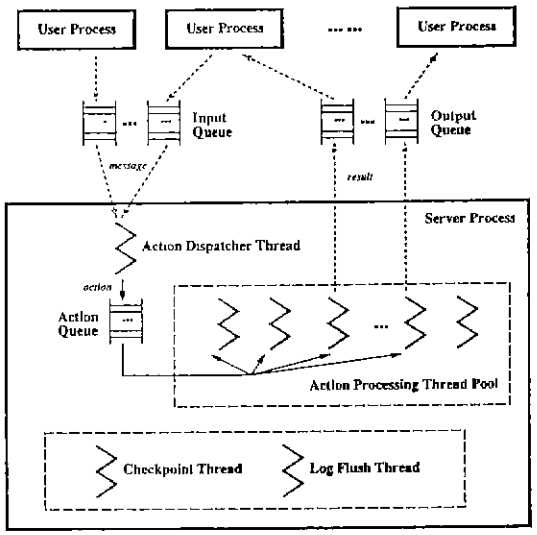


그림 2 쓰레드 구조

많아지면 시스템 전체의 성능이 저하될 수 있으므로 동시에 수행될 수 있는 Action 쓰레드의 개수를 제한한다.

4.2 저장 관리

저장 관리 시스템은 데이타베이스에 저장되는 객체들을 관리하는 데이타베이스 관리자, 데이타베이스의 메모리 영역을 관리하는 메모리 관리자, 그리고 디스크 백업을 관리하는 백업파일 관리자로 이루어진다.

M2RTSS의 다른 시스템들은 데이타베이스 관리자를 통해서 데이타베이스에 접근할 수 있다. 시스템 오류시 회복을 담당하는 회복 관리 시스템은 저장 시스템의 메모리 관리자와 백업파일 관리자를 통해서 트랜잭션의 redo, undo, 체크포인팅 등을 수행한다.

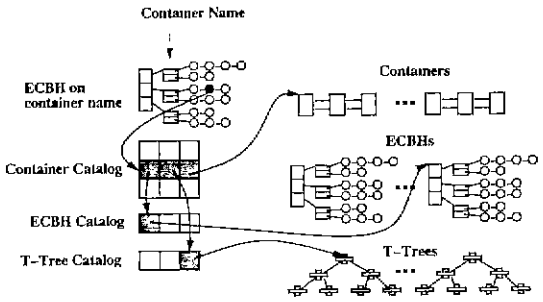


그림 3 데이타베이스 구성 요소

4.2.1 데이타베이스 관리자

데이타베이스 관리자는 데이타베이스를 구성하는 컨테이너와 인덱스, 카탈로그 객체들을 관리하며, 데이타베이스에 접근하려는 action 객체들에게 일관되고 사용이 간단한 인터페이스를 제공한다. 그림 3은 데이터베이스의 구성 요소를 보여 준다.

4.2.2 메모리 관리자

데이타베이스를 구성하는 전체 메모리 공간은 SEGMENT_SIZE의 길이를 갖는 MAX_SEGMENT_NO 개의 세그먼트로 구성되고, 각 세그먼트는 다시 PAGE_PER_SEGMENT 개의 페이지로 나누어진다. 메모리 관리자는 메모리 공간을 관리하면서 데이타베이스의 컨테이너와 인덱스, 카탈로그 객체들에게 페이지 단위로 메모리를 할당하고 반납 받는다. 메모리 관리자는 회복 관리자, 백업파일 관리자와 함께 상호 작용하여 메모리 페이지 위에 저장된 객체들에게 지속성을 부여한다.

메모리 페이지는 그것을 사용하는 객체에 따라서 다른 포맷을 가진다. 컨테이너의 경우에는 페이지를 다시 일정한 길이의 슬롯으로 나누고, 각 슬롯에 레코드를 저장한다. 이 때 슬롯은 세그먼트 번호-페이지 번호-슬롯 번호로 구성되는 식별자에 의하여 구별된다. 이 식별자는 데이타베이스 범위에서 유일하고, ECBH와 T-트리의 각 노드에서는 이 식별자를 레코드에 대한 포인터로서 사용한다.

4.3 동시성 제어

M²RTSS의 동시성 제어는 잠금 기반 방식을 채용하여 설계되었다. 이는 잠금 기반 방식이 구현이 용이하다는 장점과 함께, 약점으로 지적되는 blocking 문제도 데이타베이스가 주메모리 상에 상주하게 됨으로써 상당 부분 완화될 수 있기 때문이다. 잠금 충돌 발생시 해소 방안으로는 현재 PI 방식이 구현되어 있으며, 향후 다른 알고리듬을 수용할 수 있게 할 예정이다.

현재 M²RTSS의 구현은 C++를 이용하여 이루어지고 있으므로, 상속, 다형성 등의 객체지향 개념을 적절히 활용함으로써 확장이 용이

할 것으로 기대된다. 동시성 제어에 관한 모든 정보는 잠금 관리자에 의해 관리되는데, 잠금 충돌 해소 방안에 무관하게 잠금의 설정 및 해제에 대하여 동일한 인터페이스를 제공한다.

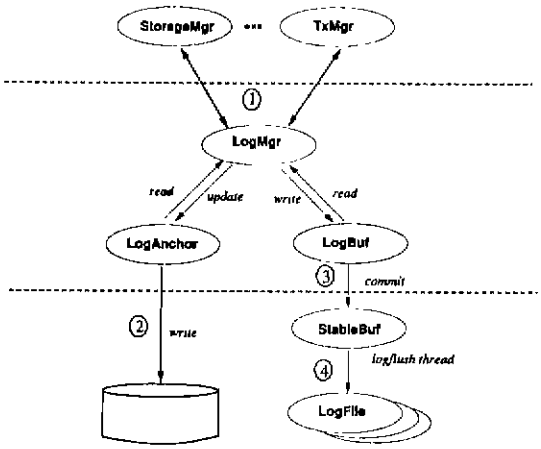


그림 4 회복 관리

4.4 회복 관리

M²RTSS의 회복 관리 시스템은 로그 관리자와 체크포인트 관리자로 이루어진다. 저장 시스템의 다른 관리자들은 그림 4에서와 같이 로그 관리자를 통해서만 로그에 접근할 수 있으며, 로그 관리자는 로그 레코드의 읽기/쓰기, 트랜잭션 undo 등의 인터페이스를 제공한다. 체크포인트 관리자는 다른 저장 시스템과는 독립적으로 체크포인트 쓰레드에 의해 수행된다.

M²RTSS의 회복 관리 기법은 ARIES 알고리듬을 기반으로 하고 있다. 그러나, 앞에서 기술한 것처럼 주메모리 DBMS는 디스크 기반 시스템과는 다른 요구 조건을 갖고 있기 때문에 이에 따른 변경이 필요하다. 주데이타베이스가 모두 메모리에 상주하기 때문에 체크포인팅시 메모리 상의 변경 사항을 디스크의 백업 데이타베이스에 반영해야 하고, 따라서 체크포인팅 로그 레코드의 내용이 바뀌게 된다. 또한 3단계 알고리듬을 사용하는 ARIES와는 달리 analysis 단계 없이 2단계만으로도 회복이 가능하게 된다.

앞에서 기술한 바와 같이 M²RTSS에서는 컨테이너 단위의 잠금만을 지원한다. 이는 주메모리 DBMS의 경우 빠른 트랜잭션 처리 속도로 인해 잠금 단위가 큰 쪽이 시스템 전체의 성능에 도움이 되기 때문이다. 따라서 레코드 단위의 잠금을 지원하기 위한 ARIES의 기능들은 포함되지 않았다.

5. 결 론

본 논문에서는 주메모리 실시간 DBMS의 특징 및 연구 동향을 소개하고 서울대학교에서 연구개발 중인 M²RTSS에 대하여 기술하였다. M²RTSS는 그 자체로서 실시간 고성능 처리가 요구되는 분야의 하부 저장 시스템으로 활용될 수 있다. 개발될 시스템은 국내에서는 아직 구현된 바가 없는, 트랜잭션 개념과 회복 관리, ECBH, T-트리 등을 지원하는 주메모리 상주형 실시간 DBMS로서, 이를 바탕으로 주메모리 상주형 데이타베이스 및 실시간 데이타베이스 분야의 고급 연구 기반을 확보할 수 있을 것으로 기대된다.

한편, M²RTSS 상에 객체지향 모델을 탑재하여 객체지향 DBMS(MMOODBMS : Main Memory Object-Oriented DBMS)를 연구개발하는 것도 고려할 수 있다. 이러한 객체지향 DBMS는, 관계형 DBMS에 비해서 복잡한 데이타 모델을 요구하는 응용 분야-GIS, CAD/CAM 등-에서의 활용이 기대된다.

참고문헌

[1] Garcia-Molina, H., and Salem, K., "Main Memory Database Systems : An Overview," *IEEE Transactions on Knowledge and Data Engineering*, Vol. 4, No. 6, 1992.

[2] Kao, B. and Garcia-Molina, H., "An Overview of Real-Time Database Systems," Advances in *Real-Time Systems*, pp. 463-486, Prentice Hall, 1995.

[3] Mohan, C., Haderle, D., Lindsay, B., Pirahesh, H., and Schwarz, P., "ARIES : A Transaction Recovery Method Supporting Fine-Granularity Locking and Partial Rollback Using Write-Ahead Logging," *ACM Transactions on Database*

Systems, Vol. 17, No. 1, 1992.

[4] Lehman, T. J. and Carey, M. J., "A Study of Index Structure for Main Memory Database Management Systems," *Proceedings of the 12th VLDB Conference*, 1986.

[5] Abbott, R. and Garcia-Molina, H., "Scheduling Real-Time Transactions : A Performance Evaluation," *Proceedings of the 14th VLDB Conference*, 1988.

[6] Huang, J., Stankovic, J. A., and Ramamritham, K., "Experimental Evaluation of Real-Time Optimistic Concurrency Control Schemes," *Proceedings of the 17th VLDB Conference*, 1991.

[7] Sha, L., Rajkumar, R., and Lehoczky, J. P., "Priority Inheritance Protocols : An Approach to Real-Time Synchronization," *IEEE Transactions on Computers*, Vol. 39, No. 9, pp.1175-1185, 1990.

[8] Haritsa, J. R., Carey, M. J, and Linvy, M., "On Being Optimistic about Real-Time Constraints," *Proceedings of ACM PODS Symposium*, 1990.

[9] Bernstein, P. A., Hardzilacos, V., and Goodman, N., Concurrency *Control and Recovery in Database Systems*, Addison Wesley, 1987.

[10] Gray, J. and Reuter, A., Transaction Processing : *Concepts and Techniques*, Morgan Kaufmann, 1993.

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

[12] Li, X. and Eich, M. H., "Post-crash Log Processing for Fuzzy Checkpointing Main Memory Database," *Proceedings of IEEE Conference on Data Engineering*, 1993.

[13] Haerder, T. and Reuter, A., "Principles of Transaction-Oriented Database Recovery," *ACM Computing Surveys*, Vol. 15, No. 4, pp.287-318, 1983.

[14] Lehman, T. J., Shekita, E. J., and Cabreara, L., "An Evaluation of Starburst's Memory Resident Storage Component," *IEEE Transactions on Knowledge and Data Engineering*, Vol. 4, No. 6, pp. 555-565, 1992.

[15] Jagadish, H. V., Lieuwen, D., Rastogi, R., and Silberschatz, A., "Dal : A High Performance Main Memory Storage Manager," *Proceedings of the 20th VLDB Conference*, 1994.

[16] Jagadish, H. V., Silberschatz, A., and Sudarshan, S., "Recovering from Main-Memory Lapses," *Proceedings of the 19th VLDB Conference*, 1993.

[17] Heytens, M., Listgarten, S., Neimat, M.-A., and Wilkinson, K., "Smallbase : A Main-Memory DBMS for High-Performance Applications(release 3.7)," HP Lab., Palo Alto, CA, USA, 1994.

[18] Hvasshovd, S.-O., Torbjørnsen, Ø., and Bratsberg, S. E., "The ClustRa Telecom Database : High Availability, High Throughput, and Real-Time Response," *Proceedings of 21st VLDB Conference*, 1995.

[19] "Non - Volatile Memory SIMM (NVSIMM)," the specification for Axil's NVSIMM for Axil workstation, 1994

[20] Hagmann, R. B., "A Crash Recovery Scheme for a Memory-Resident Database System," *IEEE Transactions on Computer*, Vol. 35, No. 9, pp. 839-843, 1986.

차 상 균

1980 서울대학교 전기공학과, 학사
1982 서울대학교 제어계측공학과, 석사
1991 미국 Stanford 대학교 전기공학과(컴퓨터 시스템), 박사 데이콤, Texas Instruments, IBM Palo Alto, HP Lab. 근무
1992~현재 서울대학교 전기공학부 조교수
관심분야 : 주메모리 DBMS, 객체지향 DBMS, 지능 시스템, GIS 시스템, 분산 가상현실 시스템, CIM

박 장 호

1992 서울대학교 제어계측공학과, 학사
1994 서울대학교 제어계측공학과, 석사
1994~현재 서울대학교 전기공학부 박사과정
관심분야 : 엔지니어링 데이타베이스, 주메모리 DBMS, CIM

박 병 대

1995 서울대학교 제어계측공학과, 학사
1995~현재 서울대학교 전기공학부 석사과정
관심분야 : 주메모리 DBMS, 실시간 시스템, 객체지향 DBMS

이 성 직

1994 서울대학교 제어계측공학과, 학사
1994~현재 서울대학교 전기공학부 석사과정
관심분야 : 주메모리 DBMS, 가상현실 시스템, 하이퍼텍스트

● 제15회 정보과학논문경진대회 논문모집 ●

- 논문마감 : 1996년 2월 24일(토)
- 주 최 : 한국정보과학회
- 제 출 및 : 한국정보과학회 사무국

문 의 처 137-063 서울시 서초구 방배 3동 984-1(머리재빌딩 401호)
T. 02-588-9246/7 F. 02-521-1352