□ 기술해설 □

주기억 장치 데이타베이스 시스템에서의 회복 기법에 대한 고찰

한국과학기술원 우승균* · 이윤준** · 김명호**

목 차

1. 서 론
2. 회복 기법의 분류
2.1 안정된 메모리
2.2 별도의 하드웨어 사용
2.3 commit 방식
2.4 갱신 방식
2.5 체크포인팅 방식
3. Shadow 갱신 방식에 기반한 회복 기법
3.1 특성
3.2 시스템 구성
3.3 트랜잭션 처리 과정
3.4 체크포인팅 수행 방식
3.5 회복 과정
4. 결론 및 앞으로의 연구 방향

1. 서 론

주기억 장치 데이타베이스(Main Memory Database : MMDB)는 주기억 장치가 데이타의 주된 기억장소(primary store)로, 데이타를 영구적으로 주기억 장치에 보관하는 데이타베이스를 말한다. 반도체의 가격이 계속해서 떨어지고 있고 집적도는 계속 증가함에 따라 MMDB의 실현 가능성은 점점 높아지고 있고[7], 이미 상용 데이타베이스에 일부분으로 사용하거나 시제품으로 구현되고 있다[4].

MMDB는 메모리에 있는 데이타를 직접 다룰 수 있으므로 기존의 디스크를 기반으로 하는 데이타베이스보다 더 빠른 응답시간과 더 높은 트랜잭션 처리능력(throughput)을 장점으로 가진다. 또한 MMDB에서 데이타 입출력 시간의 예측이 가능함에 따라 실시간 데이타베이스 시스템에서의 적합성도 많이 연구되고 있다[20, 21, 1]. 그러나, 메모리의 휘발성(volatile)이라는 성질에 의해서 시스템이 파손(crash)되면 메모리 내의 데이타를 모두 잃어버리게 되는 문제점이 있다. 이를 해결하기 위해서 MMDB에서는 파손 뒤 다시 일관된 상태로 회복시키는 기법이 필요하다.

기존의 디스크를 기반한 데이타베이스에서의 회복 기법을 MMDB에 적용하는 것은 여러 가지 문제를 발생시킨다. 이런 문제로는 메모리를 사용함으로써 얻을 수 있는 빠른 처리 등의 이점을 떨어뜨리게 되는 효율성의 문제[14]와 데이타베이스 저장의 주된 매체가 메모리라는 특성에 의한 회복 과정의 차이와 commit 및 checkpoint 과정의 차이[7] 등이 있다. 따라서 MMDB에서의 회복 기법에 관한 많은 연구가 활발하게 진행되어 왔다. 본 고에서는 먼저 회복 기법의 분류 기준과 각 기준에 따른 연구 내용을 소개한다. 그리고 하나의 회복 기법에 대한 소개로 안정된 메모리를 사용하지 않는 표준의 기계 구성 환경에서 shadow 갱신 방식에 기반한 회복 기법에 대하여 설명을 하고, 마지막으로 앞으로의 연구 방향과 결론에 관하여 기술을 한다.

2. 회복 기법의 분류

지금까지 MMDB 회복 기법에 대한 연구들을 분류하는 기준은 [4]에 따르면 디스크외

*비 회 원
**종신회원

battery-backup RAM과 같은 안정된 메모리(stable memory)와 로그 저장을 위한 특별한 하드웨어, 체크포인트를 위한 별도의 프로세서 등의 사용 여부와 commit 방식에 따르고 있다. 이 외에 데이타베이스 갱신 방식과 체크포인팅 방식[3] 등의 기준이 있다. 이 장에서는 이들 분류 기준의 의미와 적용 방식, 관련 연구 등에 대하여 기술한다.

2.1 안정된 메모리

이들 기준에 대하여 살펴보면, 먼저 안정된 메모리는 전원이 차단되더라도 그 내용을 보관할 수 있는 것으로 로그를 저장하는 부분이나 shadow 방식에서 shadow 메모리에 해당되는 부분에 사용한다. 로그 버퍼를 안정된 메모리로 사용하면 일반 메모리에 비해서 가격이 좀 비싸고 접근 속도가 떨어지는 점은 있으나 로그 데이타들을 즉시 반영할 필요가 없어 트랜잭션의 commit시에도 디스크 액세스를 유발시키지 않으므로 응답 시간을 단축시킬 수 있다. 로그 버퍼의 재사용을 위하여 주기적으로 로그 버퍼의 내용을 로그 디스크로 반영시켜야 한다. 이 수행은 트랜잭션의 수행과 별도로 동작할 수 있으므로 응답 시간에 미치는 영향은 거의 없다. 이러한 장점으로 인하여 안정된 메모리를 사용한 회복 기법에 관한 연구가 많이 되고 있고[15, 14, 4, 18, 9, 16, 12] MMDB에 적합한 안정된 메모리에 대한 연구도 이루어지고 있다[5].

안정된 메모리의 장점에도 불구하고 안정된 메모리를 사용하지 않는 연구도 이루어지고 있다[3, 11, 22]. 안정된 메모리를 사용하기 위해서는 특별한 하드웨어를 구성하여야 하지만 안정된 메모리를 사용하지 않는 회복 기법은 시스템의 표준 구성에도 적용시킬 수 있고 안정된 메모리의 사용할 때 MMDB에 효율적으로 적용시킬 수도 있다.

2.2 별도의 하드웨어 사용

로그 저장을 위한 특별한 하드웨어 사용과 체크포인트를 위한 별도의 프로세서 사용은 CPU의 부담을 줄이기 위해 사용하는 방법들이다. 로그 저장을 위한 하드웨어는 초기의 회복 기법 연구[18]에서 소개되었으나 지금은 거의 사용하지 않는다. 요즘은 다중 프로세서 환경에서 프로세서의 역할 분리에 의하여 데이타베이스를 위한 프로세서와 회복 프로세서로 분리하여 회복 프로세서에서 로그 저장 및 체크포인트 작업을 담당하게 하고 있다. 따라서 요즘 시스템이 다중 프로세서를 고려하고 있기 때문에 이들 분류의 기준은 큰 의미가 없다고 볼 수 있다.

2.3 commit 방식

Commit 방식에 따른 분류는 트랜잭션의 commit 시에 로그를 디스크에 안전하게 저장하는 방식에 대한 것이다[4, 23]. 안정된 메모리를 사용하는 회복 기법에서는 트랜잭션의 commit 시에 로그를 디스크에 저장할 필요가 없으므로 이 기준은 안정된 메모리를 사용하지 않는 회복 기법에 적용된다. commit 방식으로 세 가지가 있다.

Immediate commit은 트랜잭션이 commit하기 전에 그 트랜잭션의 모든 로그 레코드를 디스크에 반영하는 방식이다. 반영이 완료되는 시점이 트랜잭션의 commit 시점이 된다. 이 방식은 모든 트랜잭션들이 각각의 commit 시점에서 로그 레코드를 디스크에 반영시키기 때문에 디스크 입출력에 대한 부담으로 인하여 트랜잭션의 응답 시간이 길어지는 단점과 로그 레코드가 한 페이지(디스크 입출력 단위)가 차지 않은 partial 페이지의 기록에 따르는 문제점이 있다.

Immediate commit의 문제점을 해결하기 위해서 제시된 방식으로 group commit가 있다. 이 방식은 로그 버퍼의 페이지가 가득 차는 경우에 디스크에 반영을 한다. 이렇게 함으로써 디스크 입출력을 줄이고 partial 페이지의 기록을 없애고 동시에 여러 트랜잭션이 commit을 할 수 있게 된다. Group commit 방식으로 전체 시스템의 성능(throughput)은 높일 수 있으나 group 내의 모든 트랜잭션은 해당 로그 버퍼가 기록될 때까지 기다려야 하기 때문에 응답 시간은 immediate commit 방식보다 오히려 길어진다. 또한 트랜잭션의 commit 시점까지 접근한 데이타에 대한 잠금(lock)을 가지고

있어야 하기 때문에 동시성이 떨어지게 된다.

Group commit에서 트랜잭션이 commit 시점까지 잠금을 유지하는 문제점은 트랜잭션의 commit 로그가 로그 버퍼에 기록된 후 바로 잠금을 풀어버리는 방식으로 해결할 수 있다. 이 방식을 pre-commit라 한다. 즉, 트랜잭션은 로그 버퍼가 디스크에 기록되는 것을 기다리지 않고 미리 잠금을 풀어 다른 트랜잭션에서 그 잠금에 해당하는 데이타를 이용할 수 있게 하여 동시성을 높이는 것이 목적이다. 이런 방식으로 수행하여도 데이터베이스의 일관성이 유지되는 이유는 엄격한 2단계 잠금(rigorous 2 phase locking) 프로토콜을 사용하고 로그 버퍼에 로그가 순차적으로 기록되므로 앞서 잠금을 푼 트랜잭션(precommit된 트랜잭션)이 그 다음 해당 잠금을 획득한 트랜잭션보다 먼저 commit되기 때문이다. 또한 일단 precommit된 트랜잭션의 commit를 못하게 하는 경우는 오직 시스템의 파손에 의해서만 발생하고 사용자나 시스템에서 발생한 철회로는 commit을 멈출 수가 없기 때문이다[2].

2.4 갱신 방식

데이타베이스 갱신 방식에 의한 분류는 직접 갱신(in-place update) 방식과 shadow 갱신 방식이 있다. 직접 갱신 방식은 데이타베이스의 내용을 직접 갱신하는 것이고[2, 18, 14, 11], shadow 갱신 방식은 일반 메모리[3, 12, 22]나 안정된 메모리[16]로 구성된 shadow 메모리에 갱신된 내용을 기록하고 트랜잭션의 commit 시점에 shadow 메모리의 내용을 데이타베이스로 반영하는 방식이다. 직접 갱신 방식은 속도가 빠르다는 장점이 있다. Shadow 갱신 방식의 장점으로는 redo 로그만 기록하기 때문에 로그의 양이 작고 회복이 간단하고 빠른 재적재 처리 등이 있다[6]. 그러나 단점으로 갱신할 데이타의 두번 복사 과정으로 직접 갱신 방식보다 시간이 많이 걸리고 shadow 메모리를 위한 여분의 메모리가 필요로 하다는 것이 있다. 직접 갱신 방식의 단점은 shadow 방식의 장점에 상반된 것으로, redo/undo를 위한 로그가 필요하므로 로그의 양이 많고 회복 과정이 shadow 방식보다 복잡하다는 것이 된다.

직접 갱신 방식과 shadow 방식을 비교한 연구[13, 16]가 수행되었는데, [13]에서는 두 가지 방식의 회복 기법을 하나씩 선정하여 성능을 비교하였고 [16]에서는 갱신 방식이 파손 후 회복의 재적재 과정의 성능에 미치는 영향에 대한 분석을 하였다. 이 두 연구에서의 결과를 살펴보면, [13]에서는 shadow 갱신 방식이 직접 갱신 방식보다 commit하는 시간을 좀 느리지만 일반적인 트랜잭션 처리 환경에서는 더 빠른 응답 시간을 가지고 로그 기록이나 철회의 경우 shadow 갱신 방식이 적합한 조건을 가지고 있다고 결론을 내렸다. [16]에서는 fuzzy 체크포인트 방식을 사용하였는데, 결론으로 재적재 과정에서 shadow 갱신 방식을 사용한 경우 더 좋은 성능을 나타낸다고 보이고 있다.

2.5 체크포인팅 방식

다음으로 기술한 회복 기법의 분류 기준은 체크포인팅 방식이다. MMDB에서의 체크포인트는 데이타베이스 중에서 변경이 된 부분은 모두 디스크에 반영시킨다. 이렇게 하는 이유는 가능한 최근의 데이타를 디스크에 기록해두므로써 회복 과정의 부담을 줄이기 위한 것이다[14]. 대표적인 체크포인트 방식에는 행위 일관(action consistent) 방식[14, 11], 트랜잭션 일관(transaction consistent) 방식[15], fuzzy 체크포인트 방식[10, 16, 12]이 있다. 행위 일관과 트랜잭션 일관 방식은 트랜잭션 수행과 동기를 맞추는 방식으로 디스크에 기록되는 데이타베이스의 상태가 일관된 상태에 있다는 것을 의미한다. 따라서 체크포인팅을 할 때 일관된 상태를 가지기 위해서는 갱신이 일어나지 않아야 하기 때문에 트랜잭션의 수행을 멈추어야 한다.

일관 방식과는 달리 fuzzy 방식은 비동기 방식으로 트랜잭션의 수행에 영향을 주지 않고 체크포인팅을 수행한다. Fuzzy 방식의 원리는 체크포인팅 중에 디스크로 기록된 image에 체크포인팅 중에 발생한 log를 적용하면 행위 일관 방식이나 트랜잭션 일관 방식과 같은 image를 얻을 수 있다는 것이다[8].

동기 방식은 데이타베이스의 수행을 정지시키기 때문에 성능이 떨어질 수 있고 비동기 방식은 데이타베이스를 정지시킬 필요가 없기 때문에 성능에 영향을 미치지 않는다. 그러나 동기 방식은 논리적 로그 기록(logical logging) 방식을 적용할 수 있어 로그를 간단히 기록할 수 있으나 fuzzy 방식과 같은 비동기 방식은 물리적 로그 기록(physical logging)만 지원되므로 로그의 양이 많아질 수 있다[19].

체크포인트 방식의 성능을 높이기 위해 여러 가지 다른 기법들을 적용하는데, 행위 일관 방식을 사용하는 회복 기법[14, 11]의 경우에는 데이타베이스의 분할(segment)하여 분할된 단위로 체크포인팅을 하므로써 동기에 따르는 부담을 줄이고 있다. 그리고 일관 체크포인팅 방식을 page 단위로 적용하여 트랜잭션 수행에 대한 영향을 최소화하는 방식도 있다[19]. 한 가지 주목할 만한 연구는 [19]에서 수행한 여러 가지 체크포인팅 방식의 비교 연구인데, 이 연구에서는 fuzzy 방식이 비록 물리적 로그 기록 방식을 사용한다할지라도 가장 효율적인 방식이라는 것을 제시하고 있다.

앞서 설명한 기준에 의해서 회복 기법들의 특성을 그림 1에 정리하였다.

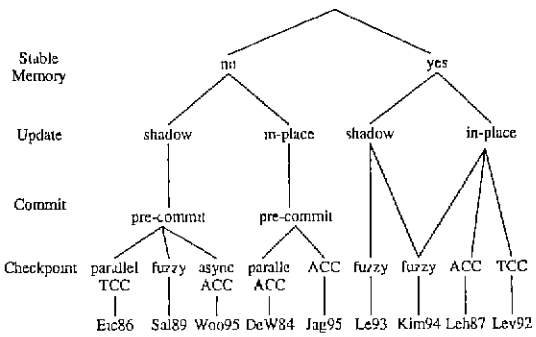

그림 1 회복 기법들의 분류

3. Shadow 갱신 방식에 기반한 회복 기법

이 장에서는 회복 기법에 대한 소개로 안정된 메모리를 사용하지 않는 시스템 환경에서 MMDB에서의 회복 기법을 제안한다. 먼저, 이 회복 기법의 특성을 설명하고, 시스템 구성과 트랜잭션 처리 과정, 체크포인팅 수행 방식, 회복 과정에 대하여 설명한다.

3.1 특 성

여기서 설명하는 회복 기법은 앞장에서 설명한 여러 기준에서의 장점을 가진 다음과 같은 특성을 지닌다.

- shadow 갱신 방식을 사용
- commit 방식은 pre-commit
- fuzzy 체크포인트
- 트랜잭션별 전용 로그 버퍼(private log buffer) 고려
- 안정된 메모리 사용을 고려하지 않음

안정된 메모리를 고려하지 않은 것은 안정된 메모리와 같은 특별한 기계 구성에 구애받지 않고 표준 기계 구성에서도 쉽게 사용할 수 있게 하고자 하는 이유이다. Shadow 갱신 방식을 사용하는 것은 2.4절에서 언급한 것과 같이 [13]와 [16]의 연구 결과에서 shadow 갱신 방식이 직접 갱신 방식보다 좋은 성능을 보이기 때문에 채택하였다. 또한 shadow 갱신 방식은 shadow 메모리에 대한 부담은 있지만 redo 로그만을 저장하여 로그의 용량을 줄일 수 있고 이에 따른 로그 기록에 의한 디스크 입출력의 감소, 회복이 간단한 점 등의 장점을 가지고 있다.

Pre-commit 방식은 트랜잭션 처리 성능을 높일 수 있기 때문에 안정된 메모리를 사용하지 않는 시스템에서 많이 사용하는 방식이다. fuzzy 체크포인트는 [19]의 연구에 의하여 MMDB 환경에서 가장 효율적인 방식이기 때문에 이를 사용한다. 전용 로그 버퍼의 사용은 MMDB 로그 버퍼의 잠금 획득에 대한 경쟁을 줄이기 위해서 사용한다. 단일 프로세서 환경에서는 시스템에서 한 순간에 하나의 트랜잭션만 수행되고 있기 때문에 잠금에 대한 의미가 없으나 다중 프로세서 환경에서는 시스템에서 단 하나인 MMDB 로그 버퍼에 대한 잠금 획득에 부담은 높아질 것이다. 예로 [9]에서 성능 측정을 위해서 사용한 매개변수의 값을 보면, lock을 얻는데 걸리는 시간은 0.025ms, 메

모리에서 하나의 단어의 입출력 시간은 0.0001ms로 되어있다. 이들 매개변수의 값을 보면 MMDB 환경에서는 잠금이 상당한 부담이 되는 것을 알 수 있다. 따라서 이에 대한 부담을 줄이기 위해서 트랜잭션별로 전용 로그 버퍼를 가지고 있고 commit 시점에 MMDB 로그 버퍼로 해당 트랜잭션의 로그 데이타를 옮기는 방식을 사용한다.

3.2 시스템 구성

시스템 구조는 그림 2와 같은 구조를 가진다. 메모리는 시스템 영역과 데이타베이스를 위한 영역으로 나누어진다. 시스템 영역은 운영체제가 사용하는 영역과 사용자가 사용하는 영역을 나타내며 데이타베이스 영역은 DBMS, 로그 버퍼, MMDB, shadow 영역으로 나누어진다. DBMS는 데이타베이스 관리 시스템이 적재되어 수행되는 영역이고 로그 버퍼는 트랜잭션이 commit하기 전에 갱신에 의해서 발생한 로그를 모아두는 곳이다.

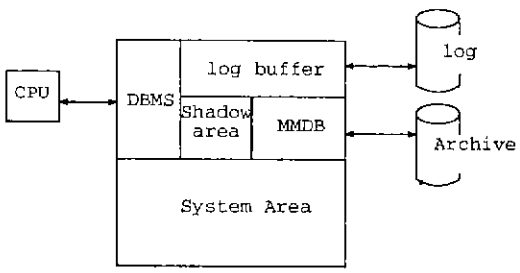

그림 2 시스템 구조

MMDB는 실제 데이타베이스의 내용이 저장되어 트랜잭션에 의해 처리되는 부분이다. 트랜잭션이 MMDB 내의 데이타를 변경하고자 할 때, 직접 MMDB의 내용을 변경하지 않고 해당 데이타의 변경된 내용을 shadow 영역에 일정 부분을 할당하여 기록한다. 트랜잭션 내에서 계속되는 변경 작업도 shadow 영역에 있는 데이터에 대해서 처리하고 트랜잭션이 commit되는 시점에 변경된 데이타를 MMDB 영역으로 반영한다.

3.3 트랜잭션 처리 과정

본 시스템에서는 전용 로그 버퍼를 가지고 있고 shadow 갱신 방식을 사용하므로 그림 3와 같은 방식으로 처리된다.

잠금은 엄격한 2단계 기법을 사용한다. 즉, 트랜잭션의 수행(1)에서 획득한 잠금들은 (5)에서 한꺼번에 모든 잠금을 해제한다. 트랜잭션이 수행을 마치면 pre-commit 상태가 되고 여기서는 먼저 트랜잭션의 전용 로그 버퍼에 있는 데이타들을 MMDB 로그 버퍼로 옮긴다(2). 그리고 나서 shadow 메모리에 있는 변경된 데이타들을 MMDB에 반영을 하고(3) 사용한 shadow 메모리를 다른 트랜잭션에서 사용할 수 있게 해제한다(4). 다음으로 트랜잭션은 잠금을 획득한 데이타에 대해서 잠금을 해제하고(5), (2)에서 로그 버퍼로 옮긴 로그 데이타가 디스크에 기록되기를 기다린다(6).

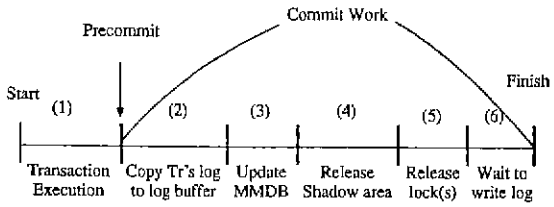

그림 3 트랜잭션 수행 과정

Shadow 메모리의 효율적인 사용에 대한 사항은 [6]에 언급되어 있는 방식을 따르며 본고에서는 자세하게 언급하지는 않는다. (6)에서 트랜잭션이 자신의 로그가 디스크에 기록되었는지를 알 수 있는 방법은 디스크에 기록되는 로그 페이지마다 LSN(log sequence number)를 유지하여 (2)에서 로그 버퍼의 현재 LSN을 알아두고 (6) 시점에서의 현재 LSN과 먼젓번 것을 비교하는 것이다. 두 LSN이 다를 경우는 해당 트랜잭션의 로그 데이터가 디스크에 기록된 것으로 트랜잭션은 commit하게 되고, 두 LSN이 같을 경우는 로그가 디스크에 반영이 안된 상태이므로 반영되기를 기다린다. 이 때 트랜잭션은 semaphore 기법[8]을 이용하여 기다린다.

트랜잭션의 철회는 shadow 갱신 방식을 사용하므로 쉽게 처리된다. 트랜잭션을 철회하고자 할 때는 shadow 메모리에서 해당 트랜잭션의 갱신 내용을 제거하고 사용하던 shadow 메

모리는 해제하고 전용 로그 버퍼를 사용하므로 트랜잭션의 로그 데이타를 제거한다. 그리고 2.3절에서 설명하였듯이 precommit 상태에 들어간 트랜잭션은 철회할 수 없으므로 철회에 의한 데이타베이스의 일관성은 유지된다.

3.4 체크포인팅 수행 방식

체크포인팅 방식은 fuzzy 체크포인트 방식을 사용한다. Fuzzy 방식은 트랜잭션의 수행과는 비동기로 수행되기 때문에 데이타베이스의 정지가 일어나지 않는다. 체크포인팅 중에는 MMDB 상의 변경된 페이지만 디스크에 기록을 한다. 수행 과정은 체크포인터가 수행이 되면 로그 버퍼에 체크포인팅 시작 로그(begin/_chkpt)를 넣고 MMDB 상의 변경된 페이지를 디스크에 기록한다.

이 과정이 끝나면 로그 버퍼에 체크포인팅의 끝 로그(end/_chkpt)를 넣고 로그 버퍼를 강제로 디스크에 기록한다. 이런 수행에서 한가지 문제점 발생하는데, 앞에서 제시한 트랜잭션 수행 과정에서 fuzzy 체크포인팅을 하게 되면 체크포인트된 결과가 완전하지 않게 되는 경우가 발생한다. 하나의 트랜잭션이 수행이 끝나서 발생한 로그를 MMDB 로그 버퍼에 옮기고 난 뒤, 체크포인터가 수행되어 시작 로그를 로그 버퍼에 기록한 경우가 발생할 수 있다.

이 경우 그 트랜잭션에 의해서 변경되는 MMDB의 페이지들이 체크포인팅 중에 디스크로 기록될 수 있을 것이다. 그러면 변경된 페이지들이 디스크에는 기록이 되어 있는데, 체크포인트의 시작과 끝 로그 사이에는 변경된 페이지에 대한 로그의 기록이 없기 때문에 2.5절에서 설명한 fuzzy 방식의 원리에 어긋나서 회복을 할 수 없게 된다. 이 경우의 해결은 위와 같은 트랜잭션들이 MMDB에 변경된 내용을 반영한 후에 변경된 MMDB 내 페이지들에 대한 작업을 수행하면 된다. 이렇게 되면 그 트랜잭션들의 commit된 후에 체크포인팅 작업이 수행되는 것이므로 회복을 할 수 있는 상태로 기록이 된다.

이 방법은 로그와 관련된 두 개의 변수를 두어서 적용할 수 있다. curr_tr은 현재 로그 버퍼에 로그 데이타를 기록한 트랜잭션의 수이고, prev_tr은 이전 로그 버퍼에 로그 데이타를 기록한 트랜잭션 수의 합이다. 현재 로그 버퍼가 디스크에 기록되고 나면 curr_tr은 0이 되고 prev_tr에 curr_tr 값이 더해진다. MMDB의 갱신이 끝난 뒤 트랜잭션은 이들 변수에 대한 값을 변경시켜는데, 현재 로그 버퍼의 LSN이 자신이 로그를 기록한 로그 버퍼의 LSN과 같으면 curr_tr에서 하나를 감소하고 그렇지 않으면 prev_tr에서 하나를 감소시킨다.

체크포인터는 이들 값을 보고 체크포인팅 시점을 결정하게 되는데, 시작 로그를 버퍼 로그에 기록하고 난 뒤 prev_tr과 curr_tr이 0이 되면 현재 MMDB를 갱신하는 트랜잭션이 없는 것이므로 작업을 수행한다. 이들 값이 0이 되지 않고 현재 버퍼가 디스크에 기록되었다면 그 다음부터는 prev_tr 값만을 고려한다. 시작 로그 이후에 로그를 기록한 트랜잭션에 대해서는 문제가 발생하지 않기 때문이다. prev_tr이 0이 되면 그 때 체크포인트 작업을 수행한다. 보통 한 번의 디스크 기록을 하고 나면 체크포인트 작업을 수행할 수 있다. 왜냐하면, 보통 한번의 디스크 기록이 있고 나면 디스크 기록이 수행되는 시간 동안 해당 트랜잭션은 MMDB 갱신 작업을 끝낼 충분한 시간을 가질 수 있기 때문이다.[9]에서 제시된 매개변수를 보면 하나의 로그 페이지를 디스크에 기록하는데 필요한 시간은 12ms이고, 메모리 상에서 하나의 페이지를 복사하는 시간은 약 0.4ms로 충분한 시간이 있음을 알 수 있다.

3.5 회복 과정

파손 후 회복 과정은 shadow 갱신 방식의 사용으로 redo만 고려해서 쉽게 할 수 있다. 먼저 가장 최근에 완료된 체크포인트를 찾아 디스크에 저장된 데이타베이스를 메모리에 적재한다. 그리고 로그 파일에서 해당 체크포인트의 시작 로그를 찾기 위해서 역방향으로 탐색을 하면서 redo할 트랜잭션들의 list를 구성한다. 시작 로그부터 로그를 MMDB에 적용하고 체크포인트의 끝 로그까지 수행되면 체크포인팅 당시의 일관된 상태까지 복원이 된 것이

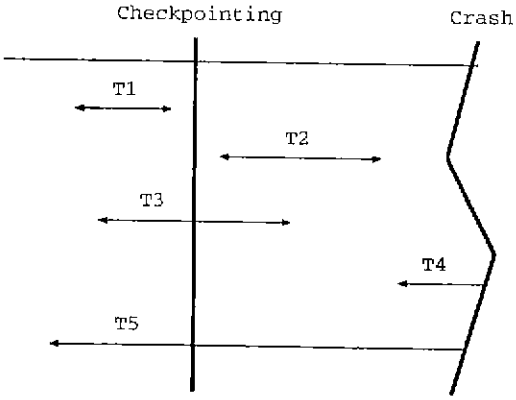

그림 4 파손 후 트랜잭션의 형태

다. 계속해서 로그 화일을 탐색하면서 조금전 구성한 redo할 트랜잭션에 대해서 redo를 수행한다.

그림 4의 파손이 일어났을 때 트랜잭션 형태에 따라서 회복 과정에서의 처리를 살펴보면, 먼저 T1은 체크포인트에 반영이 되어 있으므로 아무런 조치를 하지 않아도 되고, T2는 반영이 되지 않았으므로 redo 작업을 해야한다. T4는 MMDB에 반영이 안되어 있는 상태이고 shadow 갱신 방식을 사용하므로 아무런 조치를 취하지 않는다. T3과 T5는 체크포인트에 반영되었는지 안되었는지 알 수 없으므로 로그 화일에 해당 트랜잭션의 commit 로그가 있으면 redo 작업을 하고 없으면 아무런 조치를 취하지 않는다.

4. 결론 및 앞으로의 연구 방향

본 고에서는 MMDB의 회복 기법에 관한 고찰로서 지금까지 회복 기법의 종류에 대한 분류에 대한 기준을 설명하였고 그에 따른 연구 기법을 간략하게 살펴보았다. 분류에 대한 기준은 여기에서 소개한 것외에 다른 것이 더 있을 수 있으나 대부분의 연구에서 특징으로 내세우고 있는 몇 가지를 소개하였다. 그리고, 회복 기법의 예로 안정된 메모리를 사용하지 않는 환경에서 적용할 수 있는 shadow 갱신 방식을 이용하여 제안된 MMDB 회복 기법 소개하였다. 이 기법은 [22]에서 제시된 회복 기법의 문제점인 체크포인팅 방식을 개선한 것으로 행위 일관 체크포인트 방식을 비동기적으로 수행할 때의 shadow 메모리 한계에 의해서 발생할 수 있는 트랜잭션 수행의 정지를 fuzzy 체크포인트 방식의 도입으로 해결하였다.

MMDB의 회복 기법에서 앞으로 더 연구되어야 할 점들이 있다면 현재 대부분의 기법들이 데이타베이스 전체가 메모리 내에 적재된다고 가정을 하고 있다. 몇몇 연구에서 제시한 데이타베이스의 분할을 통한 방법이 있으나 뚜렷한 해결책은 아직 제시되지 않고 있는 실정이다. 다중 프로세서 환경에서의 회복 기법과 그들 간의 성능 측정도 고려되어야 할 부분이라고 생각된다. 지금까지 연구된 회복 기법들도 다중 프로세서 환경에서 잠금 획득에 대한 경쟁, 역할 분담 등을 고려해서 효율적인 개선이나 성능 측정을 할 수 있을 것이다.

최근의 연구에서는 유동적인 데이타베이스의 분할을 이용한 새로운 회복 기법도 제시되고 있다. 데이타베이스의 분할은 앞선 몇몇 연구 [14, 11]에서도 제안되었으나 이들은 미리 데이타베이스를 분할한 방식이고 [17]에서는 사용량과 빈도에 따라서 유동적으로 데이타베이스를 분할하여 체크포인팅 등 회복에 대처하는 방법들이 제안되었다.

지금까지 MMDB에 대한 많은 연구가 이루어지고 있으며 국외에서는 10여년부터 MMDB에 대한 연구가 시작되어 상당한 연구 활동이 진행되었고, 국내에서는 최근에 MMDB에 대한 관심이 높아지고 있다. 앞으로는 MMDB는 실시간 데이타베이스 시스템과 같은 한정된 분야에서뿐만 아니라 일반적인 데이타베이스 분야에서도 기본적으로 사용될 것이다. 따라서 이에 관한 지속적인 연구가 수행되어야 할 것이다.

참고문헌

[1] Abbott, Robert., and Garcia-Molina, Hector., "Scheduling Real-Time Transactions : A Performance Evaluation," ACM Transactions on Database Systems, Vol. 17, No. 3, pp.513-560, Sep. 1992.

[2] DeWitt, David J., Katz, Randy H., and Olken, Frank., "Implementation Techniques for Main Memory Database Systems," In Proc. of Intl. Conf. on Management of Data, pp.1-8, 1984.

[3] Eich, Margaret H., "MMDB Recovery," Southern Methodist University, Technical Report 86-CSE-11, March 1986.

[4] Eich, Margaret H., "A Classification and Comparision of Main Memory Database Recovery Techniques," In Proc. of Intl. Conf. on Database Engineering, pp.332-339, 1987.

[5] Eich, Margaret H., and Sun, Wei-Li., "Nonvolatile Main Memory : An Overview of Alternatives," Southern Methodist University, Technical Report 88-CSE-6, Jan. 1988.

[6] Eich, Margaret H., "Main Memory Database Research Directions," Southern Methodist University, Technical Report 88-CSE-35, 1988.

[7] Garcia-Monlia, Hector., and Salem, Kenneth., "Main Memory Database Systems : An Overview," IEEE Trans. on Knowledge and Data Engineering, Vol. 4, No. 6, pp.509-516, Dec. 1992.

[8] Gray, Jim., and Reuter, Andreas., Transaction Processing : Concepts and Techniques, Morgan Kaufmann, 1993.

[9] Gruenwald, Le., and Eich, Margaret H., "MMDB Reload Algorithms," In Proc. of ACM SIGMOD, pp.397-405, 1991.

[10] Hagmann, Robert B., "A Crash Recovery Scheme for a Memory-Resident Database System," IEEE Trans. on Computers, Vol. C-35, No. 9, pp.839-843, 1986.

[11] Jagadish, H. V., Silberschatz, Avi., and Sudarshan, S.. "Recovering from Main-Memory Lapses," In Proc. of VLDB Conf., pp.391-404, 1993.

[12] 김성혜, 강종규, 진성일, "In-Place 갱신과 Shadow 갱신 기법을 혼용한 주기억장치 데이터베이스 시스템," 한국정보과학회 봄 학술발표논문집, Vol. 21, No. 1, pp. 51-54, 1994.

[13] Kumar, Vijay., and Burger, Albert., "Performance Measurement of Some Main Memory Database Recovery Algorithms," In Proc. of Intl. Conf. on Data Engineering, pp.436-443, 1991.

[14] Lehman, Tobin J., and Carey, Michael J., "A Recovery Algorithm for A High-Performance Memory-Resident Database System," In Proc. of Intl. Conf. on Management of Data, pp.104-117, 1987.

[15] Levy, Eliezer., and Silberschatz, Avi., "Incremental Recovery in Main Memory Database Systems," IEEE Trans. on Knowledge and Data Engineering, Vol. 4, No. 6, pp.529-540, 1992.

[16] Li, Xi., and Eich, Margaret H., "Post-crash Log Processing for Fuzzy Checkpointing Main Memory Databases," In Proc. of Intl. Conf. on Data Engineering, pp.117-124, 1993.

[17] Li, Xi., et. al., "Partition Checkpointing in Main Memory Databases," Technical Report, 93-CSE-23, Department of Computer Science and Engineering, Southern Methodist University, Oct. 1993.

[18] Salem, Kenneth., and Garcia-Molina, Hector., "Crash Recovery Mechanisms for Main Memory Storage Database Systems," Princeton University, CS-TR-034-86, April 1986.

[19] Salem, Kenneth., and Garcia-Molina, Hector., "Checkpointing Memory-Resident Databases", In Proc. of Intl, Conf. on Data Engineering, pp. 452-462, 1989.

[20] Singhal, Mukesh., "Issues and Approaches to Design of Real-Time Database Systems," Sigmod Record, Vol. 17, No. 1, pp.19-33, March 1988.

[21] Son, Sang Hyuk., and Kang, Hyunchul., "Approaches to Design of Real-Time Database Systems," In Proc. of Intl. Symp. on Database Systems for Advanced Applications, pp.274-281, April 1989.

[22] 우승균, 홍석희, 이윤준, 김명호, "Shadow

방식을 이용한 주기억장치 데이타베이스에서의 회복 기법," 한국정보과학회 가을 학술발표논문집, Vol. 22, No. 2, pp. 293-296, 1995.

[23] 황규영, 김상욱, "주기억 장치 데이타베이스 시스템에서의 파손 회복 기법," 정보과학회지, 제11권, 제1호, pp. 47-60, 1993. 2.

우 승 균

1990 연세대학교 이과대학 전산과학학과 학사
1992 한국과학기술원 전산학과 석사
1995~현재 한국과학기술원 전산학과 박사과정 재학중
1992~현재 LG 전자 연구원
관심분야 : 실시간 데이타베이스 시스템, 메모리 데이터베이스 시스템

이 윤 준

1977 서울대학교 계산 통계학과 학사
1979 한국과학기술원 전산학과 석사
1983 프랑스 INPG-ENIMAG 전산학과 박사
1984~1994 한국과학기술원 전산학과 부교수
1995~현재 한국과학기술원 정교수
관심분야 : 분산 데이터베이스, 정보 검색, 트랜잭션처리, 데이터베이스 응용

김 명 호

1982 서울대학교 컴퓨터 공학과 학사
1984 서울대학교 컴퓨터 공학과 석사
1989 Michign State Unversity 전산학과 박사
1989~현재 한국과학기술원 전산학과 부교수
관심분야 : 트랜잭션 처리, Mobile computing, 멀티미디어 정보 처리