□ 기술해설 □

주기억 장치 데이타베이스 시스템에서 새도우 갱신 방법을 사용한 회복 기법

서강대학교 김재철* · 박 석**

● 목 차 ●

1. 서 론

주기억 장치 데이타베이스 시스템(main memory database system)이란 전체 데이타베이스를 주기억 장치(MM : Main Memory)에 상주시킨 후 데이타베이스 연산(operation)을 수행하는 시스템을 말한다[3, 5, 10]. 주기억 장치 데이타베이스 시스템은 모든 데이타가 주기억 장치에 저장되므로, 데이타베이스 연산을 수행할 때 디스크 입출력을 유발하지 않는다. 따라서, 사용자가 요구하는 응답시간(response time)을 단축시키고, 시스템의 전체적인 성능을 높일 수 있다[3, 7, 10].

트랜잭션의 빠른 처리를 보장하므로, 실시간 처리를 요구하는 데이타베이스의 여러 응용 분야에서 주기억 장치 데이타베이스 시스템을 사용하고 있다. 실시간 처리를 요구하는 데이타베이스 응용 분야로는 공장 감시 시스템(plant monitoring system), 컴퓨터 통합 제조(computer integrated manufacturing)나 공장 자동화 시스템(factory automation system) 등이 있다[6].

주기억 장치 데이타베이스 시스템은 디스크를 기반으로 하는 데이타베이스 시스템과는 달리 데이타베이스 전체가 주기억 장치에 상주하므로 시스템 전원(system power)이나 주기억 장치 칩(chip) 등에 예기치 않은 문제가 발생하는 경우 데이타베이스의 내용 전체가 손실된다. 따라서, 주기억 장치 데이타베이스 시스템에서 시스템이 파손되었을 때, 백업 디스크로부터 데이타베이스를 빠른 시간 내에 주기억 장치로 재적재 시키고, 재적재된 데이타베이스를 일관된 상태로 복구해주는 회복 작업이 매우 중요하다[3].

일반적으로 주기억 장치 데이타베이스 시스템의 회복 방법으로 갱신 방법이 상당히 중요하다. 갱신 방법이란 주기억 장치 데이타베이스를 접근하는 트랜잭션이 데이타베이스를 갱신하는 방법을 말한다. 갱신 방법으로는 트랜잭션이 데이타를 갱신하고자 할 때, 직접 데이타베이스를 고치는 직접 갱신(immediate updating)방법과 새도우 장치에서 트랜잭션의 수행을 하였다가 완료 후 주기억 장치 데이타베이스를 갱신하는 새도우 갱신(shadow updating)방법이 있다[4]. 현재는 주로 새도우 갱신 방법을 사용하고 있는데, 새도우 장치를 사용하므로 하드웨어 비용이 증가되지만 완료된 트랜잭션이 갱신한 데이타만을 주기억 장치에 반영하므로 회복 수행을 완료된 트랜잭션이 반영한 값만을 재수행(redo)하는 것으로 마칠 수 있기 때문이다. 갱신 방법 외에도 트랜잭션이 데이타베이스에 갱신한 기록을 유지하는 로깅

*비 회 원
**종신회원

방법도 회복에 있어서 상당히 중요하다. 본 논문에서는 섀도우 갱신 방법을 사용하는 주기억 장치 데이타베이스의 회복 방법을 로깅 방법에 의해 설명하고자 한다. 제2장에서는 섀도우 갱신 방법을 사용하는 데이타베이스 시스템에서 로깅 방법에 따른 회복 방법에 대해서 살펴보겠다. 즉, 로깅 후 갱신(WAL) 방법을 사용하는 회복 방법과 갱신 후 로깅(LAW) 방법을 사용하는 회복 방법을 설명하겠다. 그리고, 로그를 이용한 회복 방법의 문제점을 살펴본 후, 제3장에서는 문제점을 개선하는 재수행 버퍼를 이용한 회복 방법에 대해 설명하고, 제4장에서 결론을 맺는다.

2. 섀도우 갱신 방법을 이용한 회복 방법

주기억 장치 데이타베이스 시스템의 회복 기법에 관한 많은 연구가 있었는데, 대부분의 연구는 디스크를 기반으로 하는 일반 데이타베이스 시스템에서 사용하는 회복 기법을 주기억 장치 데이타베이스 시스템에 적합하게 한 것이다. 최근에는 효과적인 회복 수행을 위해 회복 프로세서(recovery processor)나 배터리 백업 램(battery backup RAM) 등의 추가적인 하드웨어를 사용하기도 한다[3, 4, 9].

회복에 있어서 필요한 방법은 로깅과 검사점 수행 방법, 데이타베이스 갱신 방법이 있다. 최근의 연구에서는 트랜잭션이 수행될 때, 주기억 장치 데이타베이스를 직접 갱신하지 않고, 트랜잭션의 수행을 위한 섀도우 장치를 따로 두어 완료한 트랜잭션이 갱신한 데이타만을 주기억 장치 데이타베이스에 반영하는 섀도우 갱신 방법을 주로 사용하고 있다[4, 9]. 섀도우 갱신 방법을 사용하면 취소된 트랜잭션이 수행한 값이 주기억 장치 데이타베이스를 갱신하지 않기 때문에 취소수행(undo)이 간단해지고, 회복을 재수행(redo)만으로 할 수 있다는 장점이 있다.

구체적인 회복 방법을 설명하기 전에 먼저 본 논문에서 가정하고 있는 시스템을 살펴보면 그림 1과 같다. 프로세서로 일반적인 데이타베이스 처리를 담당하는 데이타베이스 프로세서(DP : Database Processor)와 트랜잭션의 완료처리(commit processing) 및 회복 수행을 담당하는 회복 프로세서(RP : Recovery Processor)가 있다. 그리고, 주 데이타베이스(primary database)를 저장하는 주기억 장치(MM : Main Memory)와 완료처리 이전의 데이타베이스 상태를 유지하는 섀도우 장치(SM : Shadow Memory)가 있다. 또한, 트랜잭션이 데이타베이스에 갱신한 모든 기록을 유지하는 로그(Log)가 있고, 시스템 파손에 대비한 데이타베이스의 백업인 백업 디스크(AM : Archive Memory)가 있다. 로그는 배터리 백업 램으로 안정된 장치로 구성된다.

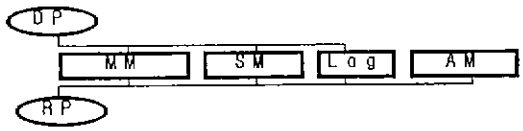


그림 1 주기억 장치 데이타베이스 시스템

트랜잭션이 갱신한 모든 페이지에 대한 정보는 시스템 파손시 회복 수행과 트랜잭션의 일치성 보장을 위해 안정된 장치에 기록되어야 한다[1]. 트랜잭션이 수행한 정보와 검사점 수행 정보 등을 기록하는 장치가 로그이다.

로그는 로그 레코드의 순차적인 집합이다. 로그 레코드는 트랜잭션이 행한 모든 기록을 유지한다[9]. 즉, 트랜잭션의 시작(BT : Begin-Transaction)과 트랜잭션의 끝(ET : End-Transaction), 그리고, 갱신 이후 값(AFIM : After Image), 검사점 수행의 시작(BC : Begin Checkpoint)과 끝(EC : End Checkpoint)의 정보를 담고 있다. 본 논문에서 사용하고 있는 로그 레코드의 형식은 표 1과 같다.

표 1 로그 레코드의 여러 형식

Type	TID	LAddr	Value
AFIM	tid	laddr	afim
BT	tid	ϕ	ϕ
ET	tid	ϕ	commit or abort
BC	chkptid	ϕ	ϕ
EC	chkptid	ϕ	ϕ

AFIM은 트랜잭션이 데이타베이스에 갱신한 값을 나타내는데, 트랜잭션 식별자(tid)와 갱신한 페이지의 주소(laddr), 그리고 갱신한 값(afim)을 저장한다. BT는 트랜잭션의 시작으로 트랜잭션 식별자만을 저장하면 된다. ET는 트랜잭션이 끝났음을 알려주는데 그 트랜잭션이 성공적으로 수행을 마쳤으면 완료(commit)를 표시하고, 그렇지 않으면 취소(abort)를 표시한다. BC는 검사점 수행의 시작을 나타내므로 검사점 수행 식별자(chkptid)만을 저장하고, EC는 검사점 수행의 끝을 나타낸다. 트랜잭션이 수행되면, 모든 정보가 로그에 기록된다.

섀도우 갱신 방법을 사용하는 데이타베이스 시스템은 로깅 방법에 따라 재수행의 시점이 달라진다. 왜냐하면, 회복 수행은 로그에 기록되어 있는 값을 가지고 하게되는데, 로그의 기록되어 있는 시점에 따라 재수행을 할 수도 있고, 안할 수도 있기 때문이다. 그림 2는 섀도우 갱신 방법을 따르는 시스템에서의 일반적인 회복 수행을 보여준다.

BC1은 시스템이 파손된 시점에서 가장 최근의 완료된 검사점 수행의 시작이고, BC2는 파손 시에 진행 중이던 검사점 수행의 시작이다. Oldest-BT란 BC1에서 진행 중이던 트랜잭션 중 가장 일찍 시작한 트랜잭션의 시작 시점이다. 로깅 방법으로 로깅 후 갱신을 따르는 알고리즘에서는 그림 2의 (1)과 같이 Oldest-BT로부터 파손 시점까지 재수행을 하고, 갱신 후 로깅을 사용하는 알고리즘에서는 (2)와 같이 BC1에서부터 재수행하면 된다[4, 9].

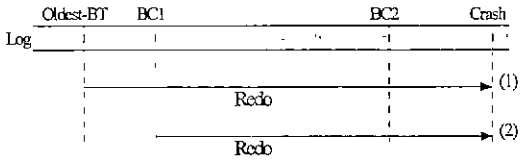


그림 2 섀도우 갱신 방법을 사용한 회복 수행

2.1 로깅 후 갱신 방법을 사용한 회복 방법

로깅 후 갱신 방법은 [정의 1]과 같이 정의할 수 있다[4, 9].

로깅 후 갱신 방법을 사용한 알고리즘에서는 그림 2에서 볼 때, 데이타베이스를 갱신할 내용은 BC1 이전에 로그에 기록하였지만 실제로 로그에 기록한데로 데이타베이스를 갱신한 시점은 BC1 이 후일 수 있다. 따라서, 회복 수행 시에 문제가 발생할 수 있는데, 그림 3에서 그러한 문제를 나타내었다.

> [정의 1] 로깅 후 갱신(WAL : Write Ahead Logging)
> 트랜잭션이 수행한 내용을 로그에 먼저 기록한 후 데이타베이스를 갱신하는 방법

BC1 시점에서 볼 때, 트랜잭션이 갱신한 데이타의 값은 BC1 이전에 기록(logging)되어 있고, 실제로 주기억 장치 데이타베이스는 BC1 이 후에 갱신(AFIM)되는 경우가 발생한다. 검사점 수행시에는 데이타베이스에 갱신된 페이지만이 백업 디스크로 백업되므로, 이와 같은 경우 BC1 이 후에 갱신된 데이타베이스 페이지는 BC2 이 후에 백업 디스크로 백업을 하게된다. 따라서, AFIM가 시스템 파손 시에 백업 디스크로 정확히 백업되었는지는 알 수 없다.

정확히 백업되었는지를 알 수 없으므로 정확한 회복을 위해서 재적재된 주기억 장치 데이타베이스에 대해 BC1이 아닌 Oldest-BT로부터 재수행을 해주어야 한다.

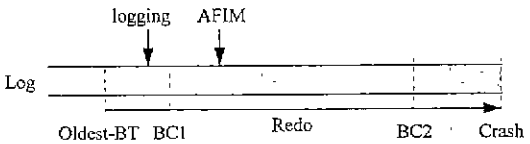


그림 3 로깅 후 갱신 방법에서의 회복 수행

만약, 그림 3과 같이 Oldest-BT로부터 재수행을 하지 않고 BC1에서부터 재수행을 한다면, AFIM에 대한 로그의 기록이 BC1 이전에 기록되어 있기 때문에 AFIM의 값은 재수행되지 않는다. 이러한 경우, 시스템 파손 전에 다행히 AFIM가 백업 디스크에 백업되었다면 아무런 문제가 발생하지 않지만 그렇지 않고 백업 디스크로 백업되지 않은 상태라면 회복 수행을 한 데이타베이스의 상태는 비일관된 상태

가 되므로 시스템을 일관된 상태로 다시 복구시킨 후 시스템을 재 가동하여야 한다. 그러나, 그림 3과 같이 Oldest-BT로부터 재수행을 한다면 이와 같은 문제는 해결된다. 그리고, Oldest-BT는 트랜잭션 진행 중에 알 수 있는 기록이므로 회복 수행시에 다시 찾는 오버헤드는 없다.

로깅 후 갱신방법을 사용하는 데이타베이스 시스템에서 사용하는 회복 알고리즘은 다음과 같다[4].

```
〈Algorithm SH_NO1〉
a initiate CL list.
b read in log block starting from end of
  log.
c WHILE BC1 is not reached DO
  c.1 FOR each log record in the block DO
      if ET and value is Commit-ET, then
      put its TID into CL.
  c.2 read in the next BFIM log block.
d WHILE end of log is not reached DO
    FOR each log is after Oldest-BT DO
      if TID is in CL, then copy AFIM to
      shadow memory.
```

SH_NO1은 섀도우 장치로 안정된 장치를 사용하지 않기 때문에 시스템이 파손되면, 로그의 섀도우 장치에서 수행되고 있는 내용까지 모두 손실된다. 따라서, 갱신되고 있는 내용은 안정된 로그에 먼저 기록하게 되므로 로깅 후 갱신방법을 따르는 것이다. 알고리즘 SH_NO1을 자세히 설명하면 다음과 같다.

먼저, 완료 리스트(CL)를 초기화한다. 그리고, 로그의 끝으로부터 순차적으로 검색하여 트랜잭션의 끝(ET : commit)이 있으면 완료한 트랜잭션이므로 트랜잭션 식별자(TID)를 완료 리스트에 추가한다. 이러한 완료 리스트를 만드는 이유는 재수행을 할 때에 필요 없는 부분을 미리 제거하기 위함이다. 완료 리스트의 작성을 마쳤으면, Oldest-BT로부터 로그를 순차적으로 검색하여 완료 리스트에 들어있는 트랜잭션이 기록한 AFIM만을 섀도우 장치에 반영한다. 주기억 장치에 직접 반영하지 않고 섀도우 장치에 반영하는 것은 섀도우 장치를 사용하는 시스템이므로 직접 데이타베이스를 고치지 않고, 섀도우 장치에서 모든 부분이 반영된 후에 완료처리로 데이타베이스를 고치기 위함이다. 이러한 일련의 모든 과정을 수행하므로 회복을 마치게된다.

위의 SH_NO1은 아주 간단하지만, 실제로 재수행하는 작업에 비해 완료 리스트를 구성하는데 드는 비용이 상당하다. 따라서, 회복 수행을 보다 빠르고 효율적으로 하기 위해서 완료 리스트를 트랜잭션 수행 중에 구성하는 방법을 개발하였다. 다음의 SH_NO2가 그것이다.

SH_NO2는 트랜잭션의 수행이 시작되면, 트랜잭션 식별자를 취소 리스트(AL)에 추가한다. 트랜잭션이 성공적으로 마치면 그 트랜잭션 식별자를 취소 리스트에서 제거한다. 따라서, 시스템 파손 시에 취소 리스트에 남아 있는 트랜잭션은 수행이 완료되지 않은 트랜잭션이므로 회복시 재수행할 필요가 없다. SH_NO2는 회복 수행에서 트랜잭션이 완료되었는지 아니면, 취소되었는지에 대한 검사를 하지 않으므로 SH_NO1보다 빠른 회복을 보장하고 있다.

```
〈Algorithm SH_NO2〉
a read in log block starting from Oldest-
  BT.
b WHILE end of log is not reached DO
  b.1 FOR each log record in the block DO
      if TID is not in AL or transaction
      table, then copy AFIM to shadow
      memory.
  b.2 read in the next AFIM log BLOCK.
```

2.2 갱신 후 로깅 방법을 사용한 회복 방법

앞 절에서는 섀도우 장치로 휘발성 장치를 사용하기 때문에 로깅 후 갱신 방법을 사용하여 데이타베이스에 갱신 이전에 그 기록을 로그에 유지하여 회복 수행을 하는 회복 알고리즘에 대해 살펴보았다. 본 절에서는 섀도우 장치로 안정된 장치를 사용하므로 갱신 후 로깅 방법을 사용하는 회복 알고리즘을 살펴보겠다.

주기억 장치와 새도우 장치로 모두 휘발성 장치를 사용하면 시스템 파손시, 주기억 장치의 내용과 새도우 장치의 내용 모두를 손실한다. 따라서, 앞 절에서와 같이 이미 기록되어 있는 로그의 내용으로만 회복을 하여야 한다. 그러나, 방대한 로그의 내용을 검색하여 재수행 하여야하므로 회복 수행을 빠르고 효과적으로 수행하지 못한다. 이러한 단점을 극복하고 회복 수행을 보다 효과적으로 하기 위해 새도우 장치로 비휘발성 장치를 사용하는 시스템을 개발하게 되었는데, 그림 1에서 새도우 장치를 로그와 같은 안정된 배터리 백업 램을 사용한다.

안정된 새도우 장치를 사용하면, 안정된 장치를 많이 사용하므로 시스템의 가격이 높아지는 단점이 발생하지만, 로그에 기록한데로 진행 중이던 트랜잭션이 갱신했던 값이 그대로 유지되기 때문에 보다 최근의 값으로 회복할 수 있고, 로그에 미리 기록하지 않아도 되므로 모든 갱신을 데이타베이스에 한 후에 로그에 기록하는 갱신 후 로깅 방법을 사용할 수 있는 장점이 있다.

위에서 설명한데로 안정된 새도우 장치를 이용하면, 갱신 후 로깅 방법을 사용할 수 있다. 갱신 후 로깅 방법을 정의하면 [정의 2]와 같다[4,9].

갱신 후 로깅 방법을 사용하면, 그림 4와 같이 BC1에서부터 파손 시점까지 재수행 만으로 회복 수행이 가능하다.

> [정의 2] 갱신 후 로깅
> (LAW : Logging After Writing)
> 트랜잭션이 새도우 장치에서 수행한 내용을 데이타베이스에 모두 갱신한 후 그 내용을 로그에 기록하는 방법.

BC1 시점에서 볼 때, 데이타베이스에는 이미 갱신한 값(AFIM)을 기록하였지만, 로그에는 BC1 이후에 기록되어 있는 경우가 발생한다. 이 경우에 데이타베이스는 BC1 이전에 갱신되었기 때문에 검사점 수행에 의하여 이미 갱신된 내용이 백업 디스크에 기록되었다. 그러나, 로그에는 BC1 이후에 기록되어 있으므로 회복 수행시 재수행을 하게 된다. 이렇게 이미 재적재되어 있는 페이지에 대해 필요없는 재수행을 하게되는 경우도 있지만, 로깅 후 갱신 방법에서와 같이 Oldest-BT를 유지하거나 그 시점부터 재수행할 필요가 없으므로 좋은 성능을 보이게 된다.

갱신 후 로깅방법을 사용하는 시스템에서의 회복 방법은 많이 연구되어 왔는데, [4,9]에서 적절한 알고리즘을 제시하고 있다. 갱신 후 로깅 방법을 사용한 알고리즘 SH_CO를 설명하면 다음과 같다.

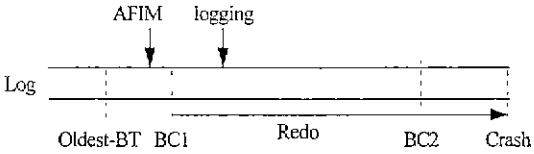


그림 4 갱신 후 로깅 방법에서의 회복 수행

시스템 파손 시에 완료 처리 중이던 모든 트랜잭션에 대해 갱신된 값이 새도우 장치에 있다면 그것을 참조하여 로그를 재구성한다. 아무리 갱신된 값이 새도우 장치에 있다 하더라도 재수행은 로그를 가지고 하기 때문에, 그 값을 로그에 추가하는 것이다. 이 작업을 모두 마치면, 진행 중이던 트랜잭션이 갱신하던 모든 값에 대해 로그가 그 기록을 유지하게 된다. 위의 작업을 모두 마친 후에는 앞 절에서 설명한 알고리즘과 같이 로그를 검색하여 재수행하면 된다. 이 알고리즘 역시 직접 데이타베이스에 대해 재수행하는 것이 아니라, 새도우 장치에 반영한 후에 완료처리로 데이타베이스를 갱신하게 된다.

> 〈Algorithm SH_CO〉
> a. For each committing transaction DO
> While there is updated item in shadow DO write log record to the log.
> b. read in log block starting from BC1
> c. While end of the log is BT reached DO
> c.1 FOR each log record in the block DO
> copy AFIM to shadow memory.
> c.2 read in the next log block

안정된 새도우 장치를 이용한 회복 알고리즘 SH_CO는 시스템 파손시 진행 중이던 트랜잭션이 갱신하고 있는 페이지에 대해서도 회복을 수행하므로 SH_NO1이나 SH_NO2 보다 더 최근의 값으로 회복하게 된다. 따라서, 새로운 트랜잭션을 수행할 때, 이전의 갱신된 값을 읽지 않으므로 보다 나은 성능을 보인다. 그러나, 안정된 새도우 장치를 사용하므로 부가적인 하드웨어의 비용이 소모되고, 진행중인 트랜잭션에 대한 로그를 세로 구성하므로 회복 시간이 SH_NO2보다 느리다는 단점이 있다.

2.3 로그를 이용한 회복의 문제점

2.1절과 2.2절에서 각각 로깅 후 갱신 방법과 갱신 후 로깅 방법을 사용하는 회복 알고리즘에 대해 살펴보았다. 본 절에서는 로그를 이용한 회복이 가지고 있는 근본적인 문제점에 대해 살펴보기로 하겠다.

앞에서 설명한 회복 방법들은 모두 시작하는 위치만 차이가 있을 뿐, 로그를 순차적으로 검색하여 재수행하게 되는데, 로그는 데이타베이스를 갱신한 모든 내용을 유지하고 있으므로 근본적으로 회복 수행을 지연시킨다. 예를 통해 회복이 지연되는 경우를 살펴보겠다.

[예 1]

먼저, 간단한 트랜잭션의 수행을 가정하자. 트랜잭션 T_1은 x와 y를 각각 90,000과 210,000으로 갱신하는 연산을 가지고 있고, 수행이 완료(commit)되었다. 반면에, T_2는 a와 b를 각각 5,000과 50,000으로 갱신하는 연산을 수행하였는데, 취소(abort)되었다. 이때, 일반 로그를 사용한다고 가정하면, 표 2와 같이 T1과 T2의 모든 내용이 로그에 기록된다. 실제로 회복에 필요한 내용은 색칠되어 있는 부분이지만 재수행시 다른 모든 부분까지도 검색하므로 회복 시간이 지연되고 있다.

그러한 점을 개선하여 로그의 양을 줄이고, 회복 수행에 걸리는 시간을 단축시키기 위해 지역 로그와 전역 로그를 구분하여 사용하는 방법이 제시되었다[2,4,7,8,9]. 전역 로그와 지역 로그를 구분하여 사용하면 지역 로그의 내용은 표 2와 같지만 전역 로그에는 표 3과 같이 T_1에 관한 내용만이 기록되어 있으며, 레코드의 아이템도 3개로 줄어들고 있다. 회복 수행시에는 지역 로그는 참조하지 않고 전역 로그만을 이용한다.

표 2 일반 로그의 예

BT	1	ϕ	ϕ
AFIM	1	x	90,000
AFIM	1	y	210,000
ET	1	ϕ	commit
BT	2	ϕ	ϕ
AFIM	2	a	5,000
AFIM	2	b	50,000
ET	2	ϕ	abort

표 3 전역 로그의 예

BT	1	ϕ
1	x	90,000
1	y	210,000
ET	1	commit

그러나, 전역 로그에도 회복에 필요 없는 내용을 가지고 있으므로 실제로 회복 시간이 지연되고 있다.

[예 1]에서와 같이 로그를 이용한 회복 방법은 로그가 회복에 필요한 내용 외에 다른 내용을 함께 가지고 있으므로 회복시간을 지연시킴을 알 수 있다. 따라서, 주기억 장치 데이타베이스 시스템의 특징인 빠른 트랜잭션 처리를 보장하기 위해, 로그를 이용하지 않는 새로운 회복 방법의 연구가 필요하다.

3. 재수행 버퍼(Redo-Buffer)를 이용한 회복 방법

앞 절에서 제시한 문제점을 해결하기 위한 방법으로 본 절에서는 회복 수행에서 로그를 이용하지 않고 재수행 버퍼를 이용한 방법에 대해 살펴보겠다. 재수행 버퍼는 로그와 같이 안정된 장치에 기록되며, 실제로 로그의 일부분을 사용한다. 그리고, 로그의 기록 중에서 회복에 필요한 부분인 데이타 아이템과 데이타 값만을 유지한다.

재수행 버퍼는 재수행 레코드의 집합으로 구성되며, 재수행 레코드는 회복에 필요한 정보인 데이타 아이템과 데이타 값의 순서쌍이다. 여기서, 데이타 아이템은 주기억 장치의 데이타베이스 페이지를 나타낸다.

[정의 3] 재수행 버퍼(Redo-Buffer)
재수행 버퍼 R은 재수행 레코드의 집합인데, 재수행 레코드는 데이타 아이템(data item) D와 데이타 값(data value) V의 순서쌍이다.
$R=\{(D_1,V_1), (D_2,V_2), ...,(Dm,Vm)\}$
이 때, $D_n(1\leq m\leq n)$은 데이타베이스의 데이타 아이템을 나타내고, $Vm(1\leq m\leq n)$은 데이타 값을 나타낸다.

재수행 버퍼는 트랜잭션의 완료 처리 시에 갱신되는데, 트랜잭션이 처리되는 과정은 다음과 같다.

〈트랜잭션 처리 알고리즘〉
a. T_i의 시작(BT_i)을 로그에 기록한다.
b. T_i를 구성하는 각 연산에 대해,
  b.1 새도우 장치에서 연산을 수행한다.
  b.2 AFIM를 로그에 기록한다.
c. T_i의 모든 연산이 모두 수행되었다면,
  c.1 완료 처리를 수행한다.
  c.2 commit을 로그에 기록한다.
  c.3 새도우 장치의 내용을 해제한다.
d. T_i의 연산이 도중에 취소되었다면,
  d.1 abort를 로그에 기록한다.
  d.2 새도우 장치의 내용을 해제한다.

트랜잭션 T_i가 생성되어 수행을 요청하면, 트랜잭션의 시작을 로그에 기록한다. 트랜잭션을 구성하는 각 연산은 새도우 장치에서 수행된다. 새도우 장치에서 수행된 후, 그 기록을 로그에 기록한다. 모든 연산의 수행이 성공적으로 마친다면, 그 트랜잭션은 완료 처리를 하게 되는데, 완료 처리를 한 후에 트랜잭션의 완료를 로그에 기록하고 모든 수행을 끝낸다. 만약 연산의 수행이 성공적이지 못하였다면, 트랜잭션의 취소를 로그에 기록하고, 모든 수행을 끝낸다.

트랜잭션이 데이타베이스에 수행한 내용을 기록하는 로깅 기법으로는 〈트랜잭션 처리 알고리즘〉 b.1, b.2 에서 보듯이 갱신 후 로깅 방법을 사용하고 있는데, 새도우 장치에 기록한 값만을 로그에 유지하여 로그의 양을 최대한 줄이고자 함이다.

〈트랜잭션 처리 알고리즘〉 c.1의 완료 처리는 트랜잭션이 새도우 장치에 반영한 값을 주기억 장치로 옮기고, 재수행 버퍼를 갱신하는 것이다. 완료 처리 알고리즘은 다음과 같다.

〈완료 처리 알고리즘〉
a. 새도우 장치에 갱신한 페이지들에 대해,
  a.1 갱신한 값을 주기억 장치에 반영한다.
  a.2 페이지 비트를 '1'로 셋팅한다.
  a.3 재수행 버퍼를 갱신한다.

트랜잭션이 새도우 장치에 반영한 모든 데이타 값을 주기억 장치로 옮기고, 그 페이지가 검사점 수행시에 백업 디스크로 쓰여질 수 있도록 페이지 비트를 '1'로 셋팅한다. 마지막으로, 회복 수행에 필요한 재수행 버퍼를 갱신한다.

갱신하고자 하는 데이타 아이템이 재수행 버퍼에 존재하면, 그 데이타 값을 갱신하면 된다. 재수행 버퍼에 그 데이타 아이템이 존재하지 않는다면, 재수행 버퍼에 갱신하고자 하는 데이타 아이템을 등록하고 그 데이타 값을 갱신한다. 그림 5는 이러한 재수행 버퍼 갱신 알고리즘을 나타낸 것이다. 데이타베이스 x를 90,000으로 갱신하였을 때, 재수행 버퍼를 갱신하여야 한다. x가 이미 재수행 버퍼에서 사용된 데이타이면, (ㄱ)과 같이 기존의 값을 90,000으로 갱신하면 되고, 그렇지 않을 경우에는 x의 데이타를 재수행 버퍼에 등록한 후에 (ㄴ)과 같이 갱신한다.

재수행 버퍼는 검사점 수행시에 구성되는데, 이전 검사점 수행시에 구성된 재수행 버퍼와 현 검사점 수행시에 구성되는 재수행 버퍼가 독립되어 존재한다. 따라서, 재수행 버퍼는 항상 2개를 유지하고 있다.

하나의 재수행 버퍼는 검사점 수행이 진행되는 동안에 구성되는데, 이전 검사점 수행 중에

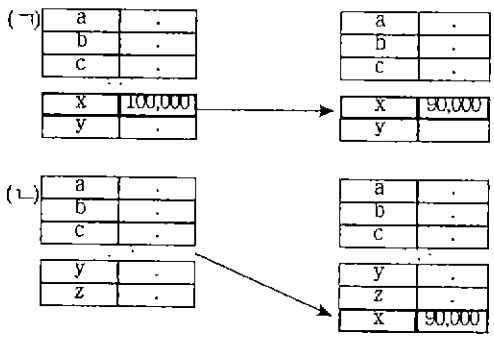


그림 5 재수행 버퍼 갱신 알고리즘 수행 예

재수행 버퍼가 구성되었다면, 현재 진행 중인 검사점 수행 동안에는 새로운 재수행 버퍼가 구성되고, 검사점 수행이 끝나면, 이전에 구성된 재수행 버퍼는 해제된다. 다음 검사점 수행 시에는 새로운 재수행 버퍼를 구성한다. 따라서, 재수행 버퍼는 항상 2개를 유지하게 된다. 항상 2개를 유지하는 이유는, 시스템 파손 시에 정확한 회복을 위해 이전 검사점 수행시에 구성된 재수행 버퍼로 재수행을 해야 하기 때문이다.

재수행 버퍼는 항상 2 개를 유지하는데, 시스템 파손 시점에서 보면, 그림 6과 같이 하나는 BC1부터 BC2까지 구성된 재수행 버퍼이고, 다른 하나는 BC2부터 Crash 시점까지 구성된 재수행 버퍼이다. BC2 이전에 구성된 재수행 버퍼를 재수행 버퍼1이라 하고, 그 이후에 구성된 것을 재수행 버퍼2라 하겠다.

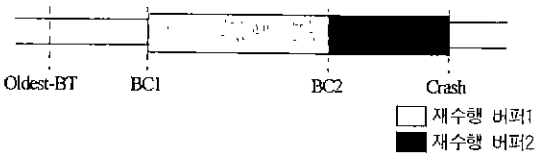


그림 6 시스템이 파손되었을때 검사점 수행 기록

실제로 시스템 파손이 생겼을 때, 재수행 버퍼를 이용하여 회복하는 방법에 대해 살펴보자.

주기억 장치와 섀도우 장치는 휘발성이기 때문에 시스템이 파손되면, 그 내용을 모두 손실한다. 그러나, 로그와 백업 디스크의 내용은 손실되지 않으므로 트랜잭션이 행한 BC1 이 후의 모든 작업의 내용은 로그와 백업 디스크에 있는 내용을 분석하여 알 수 있다. 그림 6에서와 같이 BC1 이전의 검사점 수행은 모두 완료하였으므로 백업 디스크는 BC1 이전의 주기억 장치의 내용을 정확하게 반영하고 있다.

또한, 재수행 버퍼의 정의와 재수행 버퍼의 성질에 의해 재수행 버퍼는 BC1 이 후의 주기억 장치 데이타베이스의 내용을 정확히 반영하고 있다. 이상과 같은 내용을 규칙으로 정하여 재수행 버퍼를 이용한 회복 알고리즘의 정확성을 증명할 수 있다. 따라서, 다음과 같은 규칙을 사용할 수 있다.

[규칙]
가. 백업 디스크는 BC1 이전의 주기억 장치 상태를 정확하게 반영하고 있다.
나. 로그는 BC1 이후에 갱신된 페이지에 대한 정보를 모두 가지고 있다.
다. 재수행 버퍼1은 BC1과 BC2 사이의 완료된 페이지와 완료 중인 페이지를 정확히 반영한다.
라. 재수행 버퍼2는 시스템 파손 시점까지의 완료된 페이지의 내용을 정확하게 반영하고, 완료 중인 페이지의 일 부분을 반영하고 있다.

[규칙 다]와 [규칙 라]에 의해 BC1 이 후에 갱신된 모든 페이지에 대한 정보를 재수행 버퍼가 가지고 있다. 따라서, 재수행 버퍼를 이용하여 정확한 회복을 할 수 있다. 재수행 버퍼를 이용한 회복 알고리즘은 다음과 같다.

〈재수행 버퍼를 이용한 회복 알고리즘〉
a. 완료 중이던 모든 트랜잭션을 가지고,
  a.1 재수행 버퍼2를 재구성한다.
b. 재수행 버퍼1을 가지고 재수행한다.
c. 재수행 버퍼2를 가지고 재수행한다.

우선 재수행 버퍼2를 재구성한다. 재수행 버퍼2를 재구성하는 이유는 시스템 파손 시에 구성 중이던 재수행 버퍼는 완료 중인 트랜잭션이 갱신한 페이지의 일 부분만을 반영하고 있다. 그러한 경우, 반영된 일 부분을 해제하거나 갱신 중인 모든 부분을 완전히 반영하여야 데이타베이스의 일관성이 유지된다. 본 논문에서

는 반영된 일 부분을 해제하지 않고, 완료 중인 트랜잭션이 갱신하고 있는 모든 페이지를 재수행 버퍼2에 다시 반영하는 방법을 사용하고 있다[9]. 이 방법을 사용하므로, 좀 더 최근의 갱신된 값으로 회복할 수 있다.

재수행 버퍼2를 완전히 재구성 한 후, 실질적인 회복 수행을 하게 된다. 실질적인 회복은 데이타베이스를 재수행 하는 것인데, 재수행이란 데이타베이스의 내용을 파손 이전 값으로 재 갱신하는 것을 말한다.

예를 들어, 시스템이 파손되기 전의 데이타베이스 상태는 x=1, y=2 이였다고 가정한다. 그러나, 시스템 파손으로 주기억 장치 데이타베이스의 내용은 모두 사라지고, 백업 디스크의 내용만이 남아 있다. 백업 디스크는 주기억 장치 데이타베이스의 이전 값을 반영하고 있으므로, x=3, y=1 의 값으로 파손 이전의 주기억 장치 데이타베이스와는 다른 값을 갖는다. 따라서, 회복 수행시에 로그에 기록되어 있는 값에 따라, 주기억 장치 데이타베이스의 값을 x=1, y=2로 되돌려야 한다. 이렇게 파손 이전 값으로 데이타 값을 재 갱신하는 작업을 재수행이라 한다.

재수행 버퍼를 이용한 회복 수행에서는 재수행 버퍼2보다 이전의 갱신된 값을 유지하는 재수행 버퍼1을 가지고 먼저 재수행하고, 다시 재수행 버퍼2를 가지고 재수행 한다. 재수행 버퍼1과 재수행 버퍼2의 순서로 재수행 하는 것은 유지하는 것은 순차적인 재수행을 하므로 데이타베이스의 일관성을 유지하기 위함이다. 재수행 버퍼를 이용한 회복 알고리즘은 이상의 규칙과 회복 알고리즘에 의해 주기억 장치 데이타베이스를 일관된 상태로 회복한다.

재수행 버퍼를 이용한 회복 알고리즘과 로그를 사용한 방법을 비교하여 보면 다음과 같다. 재수행 버퍼를 사용하지 않고 로그를 사용하여 회복을 한다면, 로그를 검색하는데 드는 비용이 상당하므로 재수행 버퍼를 이용한 회복 수행보다 회복시간이 길어진다. 로그의 내용은 트랜잭션이 데이타베이스에 행한 모든 기록을 유지한다. 따라서, [규칙 가]에 의해 로그가 BC1 이 후에 갱신된 모든 내용을 가지고 있다 하더라도 로그의 많은 부분이 회복과는 상관없는 정보이다. 그러나, 재수행 버퍼는 [정의 3]과 재수행 버퍼 갱신 알고리즘에 의해 회복에 필요한 값만을 유지하고 있다. 따라서, 재수행 버퍼를 이용한 회복 방법은 로그를 이용한 방법보다 훨씬 적은 양의 정보를 가지고 재수행 하므로 빠른 회복을 보장한다. [예 1]에서 로그가 회복에 필요 없는 부분을 가지고 있음을 보였는데, 실제로 재수행 버퍼를 이용한 회복에서는 재수행 버퍼가 표 4와 같이 나타나므로 회복 수행에 있어서 지연이 발생하지 않는다.

표 4 재수행 버퍼의 예

x	90,000
y	210,000

재수행 버퍼를 이용한 회복 수행의 장점은 빠른 회복을 수행하는 것뿐만 아니라, 다중 재수행을 제거한다는 점이다. 다중 재수행이란, 로그를 사용하여 회복 수행을 할 때 발생하는 것으로, 회복을 수행할 때 한 번 재수행된 페이지에 대하여 여러 번 재수행이 일어나는 것을 말한다. 예를 통해, 재수행 버퍼를 이용한 회복 방법이 다중 재수행을 제거함을 보이겠다.

[예 2]

다음의 스케줄이 있을 때

시간 / 트랜잭션	1	2	3	4	5	6	7	8	9
T_1	w(x:1)	w(y,9)		c					
T_2			w(z 2)		w(x 3)		c		
T_3						w(y:8)		w(x:4)	c

로그의 기록은 표 5와 같다.

표 5 로그의 예

AFIM	1	x	1
AFIM	1	y	9
AFIM	2	z	2
ET	1		commit
AFIM	2	x	3
AFIM	3	y	8
ET	2		commit
AFIM	3	x	4
ET	3		commit

하지만, 실제로 회복시 필요한 데이타와 데이타 값인 재수행 버퍼의 내용은 표 6과 같다.

표 6 재수행 버퍼의 예

x	4
y	8
z	2

따라서, 로그를 이용한 회복에서는 x에 대해 3번, y에 대해서는 2번, z에 대해서는 1번의 재수행을 한다. 하지만, 재수행 버퍼에는 위의 표에서 보는 바와 같이 x,y,z에 대해 한 번의 재수행만을 하면 된다. 따라서, 재수행 버퍼를 사용한 회복 방법은 추가적인 비용 없이 다중 재수행을 제거한다.

결론적으로, 주기억 장치 데이타베이스 시스템에서 재수행 버퍼를 이용한 회복 수행은 로그를 이용한 회복 수행과 마찬가지로 주기억 장치를 파손 이전의 상태로 회복하여 준다. 재수행 버퍼는 회복에 필요한 정보만을 가지고 회복 수행을 하기 때문에, 상대적으로 정보가 많은 로그를 이용한 회복 수행 보다 빠르게 수행할 수 있는 장점이 있다. 그리고, 로그를 이용한 회복 수행의 문제점인 다중 재수행의 문제도 추가 비용없이 해결함을 알 수 있다.

4. 결 론

주기억 장치 데이타베이스는 시스템 파손시 주기억 장치의 모든 데이타베이스를 손실하므로 손실된 데이타를 빠른 시간 내에 복구시키는 회복 방법이 아주 중요하다.

본 논문에서는 새도우 장치를 사용하는 주기억 장치 데이타베이스 시스템에 기초한 회복 방법들을 살펴보았는데, 로그를 이용한 방법으로 로깅 후 갱신 방법을 사용한 회복 방법과 갱신 후 로깅 방법을 사용한 회복 방법을 살펴보았다. 그리고, 로그를 사용함으로써 발생하는 회복시간의 지연을 줄이고, 효율적이고 빠른 회복을 수행하는 재수행 버퍼를 이용한 방법을 살펴보았다. 재수행 버퍼를 이용한 회복 방법은 로그를 검색하지 않으므로 회복 수행의 지연이 발생하지 않고, 한 데이타 아이템에 대하여 여러 번 재수행을 하는 다중 재수행도 제거됨을 알 수 있었다.

앞으로는 주기억 장치 데이타베이스 시스템이 더욱 광범위하게 사용될 것인데, 기억 장치의 특성상 회복 기법에 대한 많은 연구가 계속되어야 하겠다. 특히, 회복을 효율적으로 수행하고 빠른 시간 안에 트랜잭션이 수행될 수 있도록 부분 재적재(partial reloading)를 이용한 회복 알고리즘을 개발하는 것이 필요하다.

참고문헌

[1] P. A. Bernstein, V. Hadzilacos, and N. Goodman, Concurrent Control and Recovery Systems, pp.167-216, Addison-Wesley, 1987.

[2] M. H. Eich, "MARS : The Design of a Main Memory Machine," IEEE Trans. Computer. V61. c-33, No.5, pp. 391-399, 1984.

[3] M. H. Eich, "Main-Memory Database Research Directions," Tech. Rep., 88-USE -35, Pomp. Ski. Dep., Southern Methodist University, 1988.

[4] X. Li and M. H. Eich, "Post-crash Log Processing for Fuzzy Checkpointing Main Memory Databases," Proc. 9th International Conference on Data Engineering, pp. 117-124, 1993,

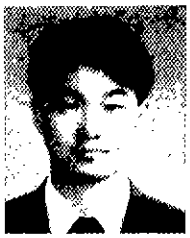
[5] K. Salem, and H. Garcia Molina, "Checkpointing Mamory-resident Databases," Proc. 8th International Conference on Data Engineering, pp. 452-462, 1989.

[6] R. Abott and H. Garcia Molina, "Scheduling Real-Time Transactions : a Performance Evaluation," Proc. 14th international Conference on VLDB, pp. 1-12, September, 1988.

[7] X. Li and M. H. Eich, "Partition Checkpointing in Main Memory Databases," Tech. Rep. 93-CSE-23, Dept. of Computer Science and Engineering, South Methodist University, Dallas.

[8] H. Jágadish, A. Silberschatz, and S. Sudarshan, "Recovering from Main-Memory Lapses." Proc. 19th international Conference on VLDB, pp. 391-404, 1993.
[9] M. H. Eich and X. Li, "Logging in Main Memory Databases," accepted to appear at the 1995 Australasian Database Conference, January 1995.
[10] 황규영, 김상옥, "주기억 장치 데이타베이스 시스템에서의 파손 회복 기법," 정보과학회지, 제 11권 1호, pp. 47-60, 1993.

김 재 철



1994 서강대학교 전자계산학과 학사
1994~현재 서강대학교 대학원 전자계산학과 석사
관심분야 : 실시간 데이타베이스의 병행수행 제어, 주기억 장치 데이타베이스의 회복, 보안 데이타베이스

박 석

1978 서울대학교 계산통계학과 학사
1980 한국과학기술원 전산학과 석사
1983 한국과학기술원 전산학과 박사
1983~현재 서강대 전자계산학과 교수
1989~1991 University of Virginia 방문교수
1992년~현재 한국정보과학회 데이타베이스 연구회 부위원장
1993년~1994년 한국정보과학회 논문지 편집위원
1995년~현재 통신정보보호학회 논문지 편집위원
관심 분야 : 실시간 데이타베이스, 보안 데이타베이스, 주기억 장치 데이타베이스, 멀티미디어 데이타베이스, 트랜잭션 관리, 병행수행 제어