울산대학교 공학연구논문집 제33권 1호 pp. 265~274, 울산대학교.
Journal of Engineering Research Vol. 33, No. 1, pp. 265~274, 2002
University of Ulsan.

안전한 실시간 데이터베이스 시스템의 복원 관리자 설계*

고재진
컴퓨터정보통신공학부

<요 약>

실시간 데이터베이스 시스템은 데드라인을 갖는 트랜잭션들을 처리하는 시스템이다. 많은 실시간 데이터베이스 응용시스템은 안전이 중요한 기관이나 군사 시스템에 적용이 되고 안전을 확보하는 것이 기관의 성패에 중요한 요소가 된다. 안전을 확보하면서 데드라인을 맞추는 것은 상호간에 상충되는 면이 있다. 본 연구에서는 확고한 데드라인을 갖는 실시간 응용 시스템에 대한 이러한 문제들을 고찰하고, 안전한 실시간 데이터베이스 시스템의 복원 관리자를 설계하였다.

Designing a recovery manager of a secure real-time database system

Jae Jin Koh
School of Computer Engineering & Information Technology

<Abstract>

A real-time database system is a transaction processing system where transactions have deadlines. Many real-time database applications arise in safety-critical installations and military systems where enforcing security is crucial to the success of the organization. The mechanisms for achieving security and for meeting transaction deadlines often work at cross-purposes. In this paper we have addressed this problem

* 본 연구는 2001년도 울산대학교 연구비 지원에 의해 수행되었음.

for real-time applications with firm deadlines and have designed a recovery manager of the secure real-time database system.

1. 서 론

안전한 실시간 데이터베이스 응용시스템은 보통 세 가지 면의 문제에 직면하게 된다. 첫째 많은 양의 데이터를 처리하면서 데이터베이스의 정확성을 유지하여야 하고, 둘 째 제한된 시간 내에 업무를 처리하여야 하고. 세 째 업무의 각 단계마다 데이터 보안이 준수되어야 한다. 본 연구에서 이러한 문제점들을 갖고 있는 안전한 실시간 데이터베이스 시스템에서 복원 관리자를 설계하였다. 아울러 확고한 데드라인(firm deadline)을 갖는 실시간 응용시스템에 보안성을 제공하는 문제를 해결하려고 한다[5]. 확고한 데드라인을 갖는 실시간 응용시스템에서 데드라인이 지난 후에 완결되는 트랜잭션은 전혀 효용가치가 없고, 오히려 해로울 수도 있기 때문에 시스템으로부터 제거된다. 트랜잭션 처리시스템이 안전하기 위해서는 그것의 구성요소들이 안전해야 한다. 그것의 구성요소에는 동시성 제어 관리자, 버퍼 관리자, 그리고 복원 관리자가 있다. 복원 관리자는 트랜잭션이나 시스템의 실패 후에 데이터베이스를 일관성 있는 상태로 만드는 역할을 수행한다. 안전한 실시간 데이터베이스 시스템에서 모든 이러한 구성 요소들은 안전성을 확보하면서 좋은 실시간 효율성을 갖도록 설계되어야 한다.

실시간 데이터베이스 시스템에 비밀 데이터와 일반 데이터가 있다고 가정하자. 만약 비밀 데이터가 부패한 고순위 보안성을 갖는 트랜잭션에 의해서 일반 데이터베이스로 전송되어서 부정을 꾀하는 저순위 보안성을 갖는 트랜잭션에 의해서 읽혀진다면, 이런 시스템에서 보안성 위배가 발생할 수 있다. 그런 직접적인 보안성 위배는 Bell-LaPadula 보안성 모델[8]의 구현에 의해서 방지될 수 있다. 그러나 Bell-LaPadula 모델은 비밀 채널(covert channel)로부터 데이터를 보호할 수 없다. 비밀 채널은 고순위 보안성 트랜잭션이 어떤 정보를 저순위 보안성 트랜잭션으로 전송할 수 있는 간접적인 수단을 말한다[7]. 예를 들면 저순위 보안성 트랜잭션이 어떤 자원에의 접근을 요청할 때, 그 자원이 고순위 보안성 트랜잭션에 의해서 이미 사용되고 있다면, 저순위 보안성 트랜잭션은 지연될 것이다. 지연이 있고 없음이 고순위 보안성 트랜잭션이 비밀 정보를 저순위 보안성 트랜잭션으로 전송하는 신호 내지는 인코딩 절차로 이용될 수 있다. 저순위 보안성 트랜잭션이 고순위 보안성 트랜잭션을 볼 수 없도록 함으로써 비밀 채널을 방지할 수 있다. 이런 현상은 불간섭(non-interference)[4]이라는 개념으로 정립되었는데 이것은 저순위 보안성 트랜잭션이 고순위 보안성 트랜잭션의 있고 없음을 분간할 수 없도록 하는데 있다. 예를 들면 저순위 보안성 트랜잭션과 고순위 보안성 트랜잭션 사이에 갈등(conflicts)이 있을 때 마다 저순위 보안성 트랜잭션에 더 높은 우선권을 주는 방법으로 불간섭을 구현할 수 있다. 본 연구에서는 그런 비밀 채널을 방지하는 복원 관리자를 설계하려고 한다.

복원 관리자의 역할은 트랜잭션의 원자성과 지속성을 보장하는데 있다. 복원 과정이 각 보안 수준에서 독립적으로 진행될 수 있지만, 모든 중요한 보안 수준에서 복원이 완료될 때 까지, 새로운 트랜잭션은 시작할 수 없다. 이 문제를 해결하기 위해서 Undo/NoRedo 프로토콜[6]이 제안되었다. 그러나 이 방법은 각 보안 수준별로 별개의 log가 유지되어야 한

다. 분산 시스템에서 복원은 중앙집중 시스템에서 보다 더욱 복잡하다. 이 문제를 해결하는 프로토콜로서 Secure Early Prepare(SEP)[1]라는 프로토콜이 제안되었다. 이것은 준비된 상태를 포함하여, 높은 보안성을 갖는 트랜잭션 수행의 어느 단계에서도 낮은 보안성을 갖는 트랜잭션에 의한 로크 선취를 제공한다.

실시간 복원 처리 분야의 연구에서 centralized timed 2PC 프로토콜이 [3]에 서술되어 있는데, 이것은 프로세서나 통신이나 클럭에 결함이 없을 때, 트랜잭션의 상태(commit 또는 abort)를 데드라인 전에 트랜잭션의 모든 참여자에게 알릴 수 있도록 보장한다. 개별 사이트들이 일반적으로 commit할 수 있도록 허용하는 기법은 논문 [9]에 기초하고 있다. 이 생각은 일방적 commit가 활동의 시의적절성을 결과로 낸다는 것이다. 본 논문의 구성은 다음과 같다. 2 절에서 안전한 실시간 데이터베이스 시스템의 개요를 서술하고, 3절에서는 복원 관리자의 개요를 서술한다. 4절에서는 복원 관리자의 설계에 대해서 서술하고, 5절에서는 결론을 기술한다.

2. 안전한 실시간 데이터베이스 시스템의 개요

안전한 실시간 트랜잭션 처리 분야는 데이터베이스 기술, 실시간 처리 기술, 정보 보안 기술 등이 결합된 것이다. 트랜잭션 처리는 ACID(Atomicity, Consistency, Isolation, Durability) 성질을 만족하도록 제공되어야 한다. 데이터베이스 시스템은 동시성 제어 관리자, 버퍼 관리자, 복원 관리자의 상호 협동에 의해서 ACID 성질을 제공한다. 복원 관리자는 commit된 트랜잭션의 모든 효과가 데이터베이스에 반영되고, abort된 트랜잭션의 효과는 전혀 반영되지 않도록 보장하여야 한다[2]. 이런 보장성은 트랜잭션의 실패나 시스템의 실패 시에도 적용되도록 하여야 한다.

복원 목적을 달성하기 위한 두 가지의 잘 알려진 기법으로 shadow paging과 write-ahead-logging이 있다. 분산 데이터베이스에서 트랜잭션의 분산된 실행에 참여하는 모든 사이트가 commit인지 abort인지를 정하는 최종 결정에 동의할 수 있도록 보장하는 commit 프로토콜이 구현되어야 한다. 그러한 commit 프로토콜에는 Two Phase Commit, Presumed Commit, Presumed Abort, Three Phase Commit등이 있다.

실시간 데이터베이스 시스템은 시간적 제한을 갖는 트랜잭션을 처리하는 시스템이다. 시간적 제한이란 데드라인을 말하며, 트랜잭션이 데드라인 전에 완료되어야 함을 말한다. 실시간 데이터베이스 시스템은 데이터 무결성 제한조건과 실시간 제한조건등 두 가지의 제한조건을 동시에 만족시켜야 한다. 실시간 시스템은 데드라인의 종류에 따라서 Hard Deadline, Firm Deadline, Soft Deadline으로 나눌 수 있다. 본 논문에서 적용한 Firm Deadline의 의미는 트랜잭션을 데드라인이 지난 후에 완료한다는 것은 전혀 소용이 없고 해로울 수도 있다는 것으로 데드라인이 지난 트랜잭션은 취소해서 버리는 것이 필요하다.

안전한 복원 절차의 역할은 비밀 채널의 가능성을 열어 줌이 없이 트랜잭션의 원자성과 지속성을 보장하는데 있다. 각 활동적인 보안 수준에 대한 별개의 복원 로그를 유지하는 것이 가능하고, 각 수준에서 복원은 독립적으로 나아갈 수 있다. 그럼으로 간섭 채널을 쉽게 배제할 수 있다. 분산 시스템에서 한 사이트가 부분 트랜잭션을 국지적으로 commit하려고 정한 후에라도, 즉 한 사이트가 commit 준비된 상태로 들어간 후에라도 국지 트랜잭

션 관리자는 전체 트랜잭션의 전반적인 결정이 가능할 때까지 그것들의 log를 유지하여야 한다. 이것은 트랜잭션의 원자성을 제공하는데 필요하다. 그러한 시나리오에서 한 사이트의 부분 트랜잭션이 높은 보안성 수준의 부분 트랜잭션에 의해서 lock된 데이터 요소에 대해서 갈등적인 lock를 요청할 때, 그 요청자는 lock 소유자가 abort되거나 commit될 때 까지 기다려야 한다. 그러나 이 절차는 악의의 높은 우선권을 갖는 트랜잭션에 의해서 신호 보내는 메카니즘으로 악용될 수 있다. 그러므로 분산된 commit 프로토콜의 보안성을 보장하는 특수한 메카니즘이 필요하게 된다.

안전한 실시간 데이터베이스 시스템의 구성도는 그림 1. 에 표시되어 있다.

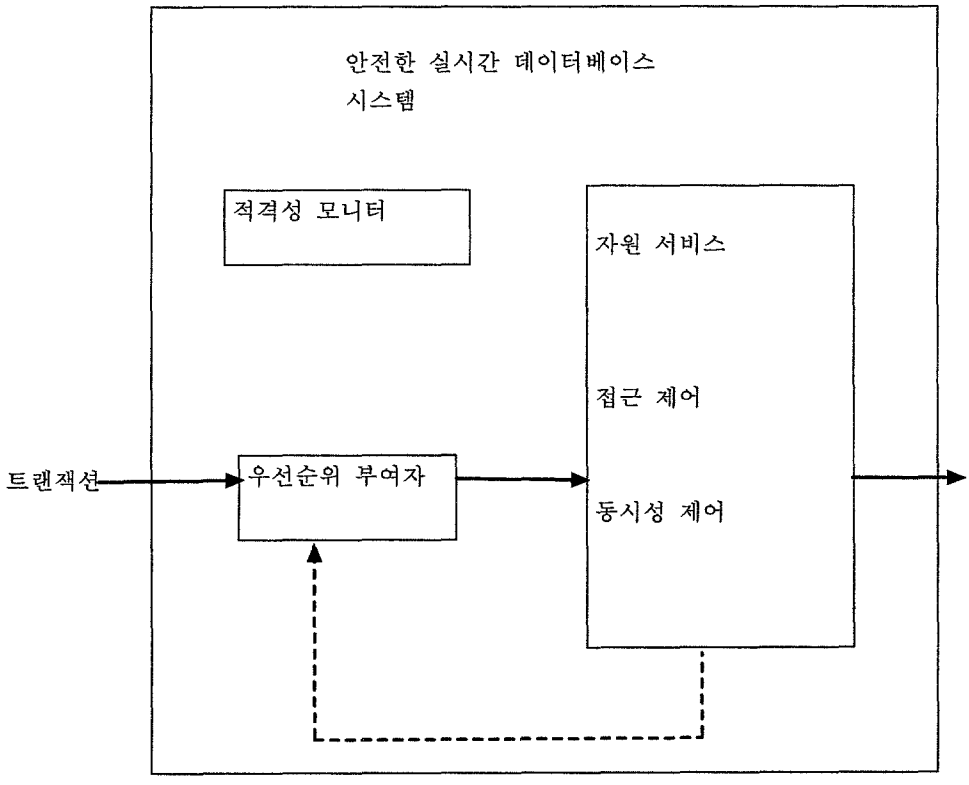

그림 1. 안전한 실시간 데이터베이스 시스템 구성도

그림 1.에서 입력은 도착시간과 데드라인과 보안 허가 수준을 갖는 트랜잭션들이다. 출력은 두 가지로 나눌 수 있는데, 제 시간에 완료한 트랜잭션과 늦게 처리된 트랜잭션이다. 적격성 모니터는 시스템에서 수행중인 모든 트랜잭션들의 적격성 상태를 추적한다. 우선순위 부여자는 도착하는 트랜잭션에 우선순위를 부여한다. 우리의 연구는 firm deadline을 갖는 트랜잭션에 한정해서 수행했다. 또 다른 전제 조건은 이 시스템이 트랜잭션 처리 요구사항이나, 트랜잭션 데이터 접근에 대해서 전혀 사전 지식을 갖고 있지 않다는 것이다. 이것은 안전한 실시간 데이터베이스 시스템의 입장에서 볼 때 트랜잭션은 단지 그것의 도착시간, 데드라인, 보안분류 수준에 의해서만 식별할 수 있다는 것이다.

3. 복원 관리자의 개요

데이터베이스 시스템은 복원 가능한 자원 관리자이다. 실패(failure)로부터 복원하는 임무란 실패 전에 commit된 모든 트랜잭션의 결과를 포함하는 상태로 복귀하고, 실패 전에 발생한 abort된 트랜잭션의 결과나 실패 시점에서 활동하던 트랜잭션의 결과는 하나도 포함하여서는 안 된다. 세 종류의 실패가 있는데, 트랜잭션 실패, 시스템 실패, 미디어 실패가 그것이다. 트랜잭션 실패는 트랜잭션이 abort될 때 일어난다. 시스템 실패는 주기억 장치의 내용이 상실되거나 오염된 것을 의미한다. 이것은 전원이 나갔던지 또는 운영체제가 실패했을 때 일어난다. 이러한 시스템 실패 때문에 데이터베이스는 디스크와 같은 안정된 저장 매체에 저장되어야 한다. 미디어 실패는 안정된 저장 매체의 어느 부분이 파괴된 것을 의미한다. 예를 들면, 디스크의 섹터가 손상을 입는 경우가 해당된다. 주기억이나 안정된 저장 장치에서 데이터의 손실을 막기 위하여, 보통 데이터의 또 다른 복사본을 가능한 한 다른 표현법으로 유지한다. 이런 잉여의 복사본은 안정된 저장 장치나 또는 제 2의 안정된 저장 장치에 유지된다. 실패로부터 복원하는 주요한 전략은 다음과 같다. 트랜잭션 실패인 경우, 만약 트랜잭션이 취소된다면, 데이터 관리자는 트랜잭션이 쓴 모든 데이터의 이전 값을 복원할 것이다. 시스템 실패인 경우, 실패로부터 복원하기 위하여, 데이터 관리자는 실패 시에 활동 중인 모든 트랜잭션을 abort하고, 실패 전에 commit된 각 트랜잭션의 결과가 데이터베이스에 수정이 되도록 보장할 것이다. 미디어 실패인 경우는 시스템 실패와 비슷하다. 시스템과 미디어 실패로부터 복원하는 가장 일반적인 기법은 logging이다. log란 실패에 대응하기 위하여 사용되어지는 데이터의 제 2의 잉여적인 복사본이다.

트랜잭션의 관점에서 보면, 복원 시스템은 read, write, commit, abort 연산을 처리하는 저장 부시스템의 한 부분이다. 트랜잭션은 저장 시스템에 read 와 write 연산을 시작하기 전에 lock를 세팅할 책임이 있고, 트랜잭션이 commit 나 abort 한 후 까지 write lock를 갖고 있어야 한다. 복원 알고리즘에 페이지 단위 locking을 사용하는 것이 편리하다. 이것은 디스크 하드웨어가 한 페이지 writing을 all-or-nothing 연산으로 다루기 때문이다. 그래서 다음과 같은 가정을 하고 연구를 진행하였다. 데이터베이스는 page들의 집합으로 구성되어 있고, 한 트랜잭션에 의한 각 수정은 오직 한 page에 적용되고, 한 트랜잭션에 의한 각 수정은 하나의 page 전체를 출력하도록 하고, lock는 page에 걸리도록 한다.

캐쉬(cache)는 휘발성 기억 장치로서, 최근에 접근되었던지 또는 최근에 수정된 데이터베이스 페이지들의 복사본을 가지고 있다. 캐쉬는 안정된 데이터베이스에 반영이 안된 수정사항을 가질 수 있다. 그런 페이지를 dirty page라 한다. dirty page를 정확히 관리하는 일이 복원 시스템의 중요한 임무의 하나이다. log는 디스크에 저장되는 순차화일로서 데이터베이스에 행해진 수정사항들을 서술하는 레코드들의 모임이다. log 레코드는 수정된 페이지의 주소, 수정을 행한 트랜잭션의 ID, write 된 페이지의 after-image, 그리고 write 된 페이지의 before-image를 갖고 있다. 그리고 log는 트랜잭션이 commit 되었는지, abort 되었는지에 대한 정보도 갖고 있다. log 는 상충(conflict)하는 연산들의 순서를 정확히 반영하여야 한다. 그래서 수정이 그 순서대로 수행되어야 한다. 그 이유는 실패 후에 일어났던 순서대로 복원 시스템이 작업을 다시 수행하여야 하기 때문이다.

복원 관리자는 commit과 abort 연산의 처리에 책임이 있는 요소 시스템이다. 그리고 그것은 시스템 실패 후 복원을 기동시키는 restart 연산 처리에 책임이 있고, 아울러 데이터

베이스를 일관성 있는 상태로 가져오며, 거기서 트랜잭션을 다시 처리할 수 있다. 복원 관리자의 기능은 그림 2. 에 도시되어 있고, 그 연산들의 기능은 다음과 같다.

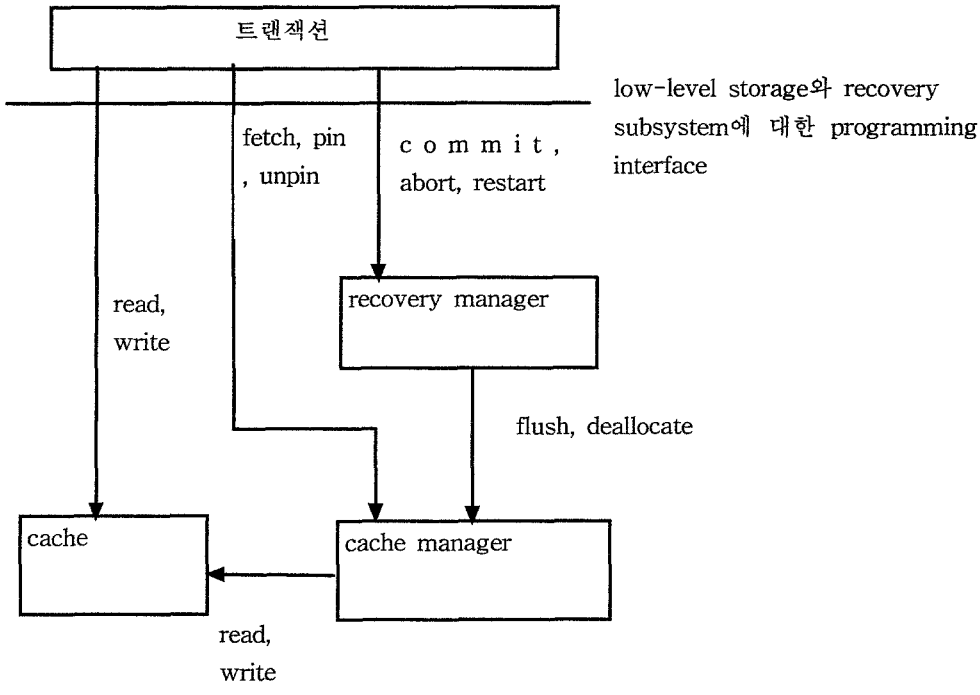

그림 2. 복원 관리자 모델. 복원 관리자는 commit, abort, restart를 구현하기 위해서 cache manager를 부른다.

Commit(Ti) : 트랜잭션 Ti의 수정된 페이지들을 안정된 데이터베이스에 영구히 저장한다. 그것의 효과는 원자적이어야 하고, 시스템 실패가 있더라도 all-or-nothing 적이어야 한다. 트랜잭션이 한번 commit 되면 abort 시킬 수 없다.

Abort(Ti) : 모든 데이터를 트랜잭션 Ti 가 수행되기 전의 상태로 복원시킨다. abort 효과는 역전시킬 수 없다.

Restart : 시스템 실패 시 수행 중이었던 모든 트랜잭션은 abort시킨다. 실패 전에commit된 트랜잭션의 수정사항 중 안정된 데이터베이스에 저장이 안된 것은 바로 안정된 데이터베이스에 저장시킨다. restart의 결과로서 데이터베이스는 commit된 수정은 포함하고, abort된 수정은 포함하지 않는다.

이런 연산을 구현하기 위해서 복원 관리자는 어떤 규칙을 따르는데, 그 규칙의 핵심은 수정된 데이터를 언제 디스크에 저장하는지를 제어하는데 있다.

4. 복원 관리자의 설계

logging에 기초한 복원 관리자 설계에 대해서 서술한다. log은 각 write, commit, abort 연산에 대해서 log record를 가지고 있다. 어떤 시스템이 covert channel secure가 되려면 다음과 같은 세 가지 요구사항을 충족시켜야 한다.

값 보안 : 만약 어떤 주체에 의해서 접근된 객체의 값이, 더 높은 보안 허가 수준에 속하는 주체들의 행동에 의해서 변경되지 않는다면 그 시스템은 value secure 하다고 한다.
지연 보안 : 어떤 주체에 의해서 경험된 지연이 더 높은 보안 허가 수준을 갖는 주체들의 행동에 의해서 영향을 받지 않는다면 delay secure 하다고 한다.
복원 보안 : 트랜잭션 restart의 발생이 시스템에서 더 높은 보안 허가 수준에 있는 주체들의 있고 없음에 상관없이, 어떤 주체에 대하여 똑같이 나타난다면 recovery secure 하다고 한다.
본 논문의 프로토콜은 어떤 높은 보안 수준의 트랜잭션이 낮은 보안 수준의 트랜잭션으로부터 버퍼 슬롯을 선점하도록 허락하지 않는다. 낮은 보안 수준의 트랜잭션이 높은 보안 수준의 트랜잭션과 갈등을 빚을 때, 항상 높은 보안 수준의 트랜잭션이 restart 된다. 그러므로 우리의 프로토콜은 복원 보안을 만족시킨다.

4.1 abort 설계

연산 Abort(Ti)를 생각하고, 트랜잭션 Ti가 페이지 P를 write 했다고 가정한다. 만약 P가 안정된 저장장치로 전송되지 않았다면, 복원 관리자는 단순히 P를 deallocate 시킨다. P가 안정된 저장장치로 전송되었다면, 복원 관리자는 P의 before-image를 안정된 저장장치에 write 해야 한다. 보통 트랜잭션 Ti가 P에 대한 수정을 log 해두기 때문에 이 문제는 간단히 해결할 수 있다. 만약 트랜잭션 Ti 가 시스템 실패시점에서 수행 중이었고, restart 프로시저가 문제를 해결하기 위해서 Abort(Ti)를 수행시키고 있었다고 가정하자. 실패 전에 Ti 의 P에 대한 수정 내용이 안정된 데이터베이스에 전송되었고, 그러나 실패 전에 그 수정사항이 log에는 전송되지 않았다고 생각하자. 이 경우에 P 에 대한 수정의 취소는 불가능하다. 왜냐하면, P의 before-image가 손실되었기 때문이다. 이것을 방지하기 위해서 다음과 같은 write-ahead log protocol 규칙을 이행하여야 한다. "uncommitted된 수정을 그것의 before-image가 포함된 log record가 log에 전송될 때까지 안정된 데이터베이스에 전송해서는 안된다." 이 규칙을 이행하는 간단한 방법은 uncommitted된 수정은 결코 안정된 데이터베이스에 flush 하지 않는다. 단지 그것을 log에 flush 한다. 트랜잭션이 commit 된 후에 그 수정을 안정된 데이터베이스에 flush 한다. 따라서 이것은 undo를 방지한다.

abort 연산을 처리하기 위해서 복원 관리자는 취소된 트랜잭션에 의해서 수정된 안정된 데이터베이스 페이지들의 수정을 무효로 해서 원래의 상태로 복원 시켜야 한다. 이것은 그 트랜잭션의 log 레코드의 끝에서부터 거꾸로 검사를 시작해서 그 트랜잭션에 의해서 수정된 각 페이지들의 before-image를 안정된 데이터베이스에 write 한다. 효율성을 높이기 위해서 복원 관리자는 log에 있는 그 트랜잭션의 수정된 모든 레코드들의 링크드 리스트를 유지한다. 그 리스트의 앞부분은 트랜잭션 서술자로서 각 트랜잭션에 대해서 서술하는 자

료구조이다. 그 서술자는 각 트랜잭션에 의해서 write 된 마지막 log 레코드의 포인터를 가지고 있다. log에 있는 각 수정사항 레코드들은 동일한 트랜잭션에 의해서 write 된 직전 수정사항 레코드에 대한 포인터를 가지고 있다. 리스트를 관리하기 위해서 트랜잭션이 수정사항 레코드를 log에 write 할 때, 그 수정사항 레코드는 그 트랜잭션에 대한 직전 log 레코드에 대한 포인터를 갖도록 하고, 트랜잭션 서술자는 이 새로운 수정사항 레코드를 가르키도록 한다.

write-ahead log protocol의 문제점에 대해서 생각해보자. uncommit된 트랜잭션에 의한 그 페이지에 대한 수정사항을 서술하는 모든 수정사항 레코드들이 log에 이미 flush 되지 않는 한, 캐쉬로부터 수정된 페이지들을 안정된 데이터베이스에 결코 flush하지 않음을 시스템이 보장할 필요가 있다. 이것을 위하여 각 캐쉬 슬롯의 캐쉬 서술자에 하나의 필드를 추가한다. 이 필드는 write-ahead log protocol을 이행하기 위해 필요한 log 페이지를 가르키고 있다. 즉 이 필드는 그 캐쉬 슬롯에 대한 최근의 수정을 서술하는 수정사항 레코드를 갖고 있는 log 페이지의 주소를 갖고 있다. 이것을 dependent log page address라 부른다. 데이터베이스 페이지 P가 수정될 때마다, P의 캐쉬 슬롯의 dependent log page address도 따라서 그 수정의 log 레코드를 포함하는 페이지를 가르키도록 수정된다. 하나의 캐쉬 슬롯을 flush하기 전에 캐쉬 관리자는 dependent log page가 캐쉬에 없고 수정도 안되었음을 체크하여야 한다. 만약 dependent log page 가 캐쉬에 있고 수정되었다면 그 dependent log page를 먼저 flush 하여야 한다.

4.2 commit 설계

commit 은 트랜잭션의 모든 수정사항이 지속됨을 의미하기 때문에, Commit(Ti)에서 Ti의 모든 수정사항은 commit 전에 안정된 저장장치(log이나 안정된 데이터베이스)에 있어야 한다. 이것을 위하여 복원 관리자는 다음의 규칙을 이행하여야 한다. force-at-commit 규칙 : "트랜잭션의 모든 수정된 페이지들의 after-images가 안정된 저장장치(log이나 안정된 데이터베이스)에 저장될 때까지 그 트랜잭션을 commit하여서는 안된다." force-at-commit 규칙을 이행하는 간단한 방법은 트랜잭션의 commit 전에 트랜잭션의 수정사항을 안정된 데이터베이스에 flush하는 것이다. write-ahead log protocol을 이행하는 간단한 방법과 force-at-commit을 이행하는 간단한 방법은 상호 모순적이다. 따라서 어느 방법을 취하던지 undo 또는 redo 는 필요하게 될 것이다.

commit 연산을 처리하기 위해서 복원 관리자는 commit 레코드를 log의 끝에 추가하고 log를 안정된 저장장치에 flush 한다. flush 연산은 주기억 장치에 있는 모든 log 페이지들과 flush된 페이지들이 디스크에 성공적으로 write 된 후에야 완료된다. 이 시점에서 트랜잭션은 commit되고 복원 관리자는 이 사실을 caller에게 통지할 수 있다. 트랜잭션을 commit 하기 위해서 log을 flushing 하는 일은 bottleneck이라 할 수 있다. 만약 log을 갖고 있는 디스크가 1초에 K disk-writes을 한다면, K 는 전체 시스템에 대해서 초당 트랜잭션의 최대수가 된다. 이런 bottleneck을 완화하기 위해서 시스템은 group commit이라 부르는 최적화 기법을 사용한다. commit 레코드를 log에 추가한 후에, 복원 관리자는 log 페이지를 flushing 하기 전에 작은 인공적인 지연을 도입한다. 이 지연은 1/K 정도이다. 이

지연 기간 동안에 만약 다른 트랜잭션이 실행되고 있다면, 그들이 수정 레코드나 commit 레코드나 abort 레코드를 log의 끝에 추가할 수 있다. 만약 시스템이 분주하다면, 잇점은 log 페이지들이 이 기간 동안에 채워지고, 복원 관리자는 이 지연 기간의 끝에 도달하면, full page의 flushing을 끝마치게 된다. 따라서 log에 대한 각 flush 연산은 많은 트랜잭션을 commit 시킬 수 있고 복원 관리자는 disk channel의 모든 역량을 얻을 수 있다.

4.3 restart 설계

restart를 수행하기 위해서는 기록하는 레코드가 필요하게 된다. 즉 실패 시점에서 어떤 트랜잭션이 수행 중이었던가를 알아두어서 그것들을 abort 시킬 수 있다. 그리고 commit 된 트랜잭션의 어떤 수정사항이 안정된 데이터베이스에 write 되지 않았는지를 알아두어서 그것들을 redo 할 수 있다. 더구나 restart는 fault-tolerant 이어야 한다. 그것은 restart 수행중에 시스템 실패가 일어나면 restart를 재수행 시킬 수 있어야 한다. 이것은 항상 시스템은 restart를 정확히 수행할 수 있는 상태를 유지하여야 함을 의미한다.

restart를 구현하기 위해서 복원 관리자는 log을 스캔하여서 어느 트랜잭션이 abort 되어야 하고, 어느 트랜잭션이 redo 되어야 하는지를 찾아내어야 한다. 모든 restart 알고리즘은 주기적으로 하는 checkpoint 연산에 의존하고 있고, checkpoint 연산은 log의 상태와 안정된 데이터베이스의 상태를 동기화시키는 것이다. 복원 관리자는 이 checkpoint 연산을 수행한다. checkpoint 연산은 다음과 같은 절차로 수행된다. 우선 어떤 새로운 update, commit, abort 연산을 받는 것을 중지한다. 모든 활동중인 트랜잭션의 리스트를 각 트랜잭션의 최근의 log 레코드에 대한 포인터와 함께 만든다. 캐쉬에 있는 수정된 페이지들을 flush 한다. 활동중인 트랜잭션들의 리스트와 log 포인터를 포함하는 checkpoint 레코드를 log에 write한다. 그 후에 새로운 update, commit, abort 연산을 받기 시작한다.

restart 알고리즘은 log을 앞에서부터 차례대로 스캔해서 각 log 레코드를 전적으로 처리한 다음에, 다음의 log레코드로 나아간다. 그것의 목적은 첫째 최근의 checkpoint 후에 수행된 모든 수정에 대해서 redo를 하는 것이고, 그 후에 commit 되지 않은 수정은 undo를 하는 것이다. 그것은 최근의 checkpoint 레코드에서 시작한다. restart 알고리즘은 commit 된 트랜잭션 리스트, abort된 트랜잭션 리스트, 그리고 최근의 checkpoint 레코드에 저장된 값으로 초기화되는 active 트랜잭션 리스트를 유지한다.

restart 알고리즘이 새로운 log 레코드를 만나면 다음과 같은 일을 한다. 만약 log 레코드가 update 레코드라면, 그 update의 after-image를 캐쉬에 write 한다. 그 트랜잭션의 identifier를 active 리스트에 추가한다. 만약 log 레코드가 commit 레코드라면 그 트랜잭션을 commit 리스트에 추가한다. 그리고 active 리스트에서 그것을 제거한다. 만약 log 레코드가 abort 레코드라면 그 트랜잭션의 모든 update를 undo 한다. 그 트랜잭션을 abort 리스트에 추가하고 active 리스트로부터 그것을 제거한다. log의 끝에 도달한 시점에서 active 리스트는 실패 전에 수행을 시작했으나 실패 전에 commit 나 abort가 되지 않은 트랜잭션들만 포함하게 된다. restart 알고리즘은 이런 active 리스트에 포한된 트랜잭션들을 abort 시킨다. restart 알고리즘이 수행되고 있는 한 사용자는 트랜잭션을 수행시킬 수 없다.

5. 결론

본 논문에서 안전한 실시간 데이터베이스 시스템, 즉 covert-channel-free security를 유지하는 시스템에서의 복원 관리자를 설계하였다. 그런 시스템은 안전이 중요한 장소나 보안을 유지하는 것이 기업의 성패에 중요한 요소로 작용하는 곳에서 응용 분야를 찾을 수 있다. firm deadline을 갖는 응용시스템을 전제로 하고 안전한 동시성 제어나 버퍼 관리 프로토콜은 다른 사람이 개발한 것을 원용하고, 본 논문에서는 안전한 복원 관리자를 설계하는데 중점을 두었다. 미래의 연구 과제로는 본 논문은 중앙 집중식 안전한 실시간 데이터베이스 시스템을 전제로 하였는데 분산된 안전한 실시간 데이터베이스 시스템에 대한 연구가 필요하다. 본 논문은 사전 지식이 없는 트랜잭션 처리를 전제로 했는데, 사전 지식이 있는 경우의 트랜잭션 처리에 대한 연구가 필요하다.

참고문헌

1. V. Atluri, E. Bertino and S. Jajodia, "Degrees of Isolation, Concurrency Control Protocols and Commit Protocols", Database Security, VIII: Status and Prospects, 1994.
2. P. A. Bernstein, N. Goodman and V. Hadzilacos, Concurrency Control and Recovery in Database Systems, Addison Wesley, 1987.
3. S. Davidson, I. Lee and V. Wolfe, "A Protocol for Timed Atomic Commitment", Proc. of 9th Intl. Conf. on Distributed Computing Systems, 1989.
4. J. Goguen and J. Meseguer, "Security Policy and Security Models", Proc. of IEEE Symp. on Security and Privacy, 1982.
5. J. Haritsa, M. Carey and M. Livny, "Data Access Scheduling in Firm Real-Time Database Systems", Real-Time Systems Journal, 4(3), 1992.
6. I. E. Kang and T. F. Keefe, "On transaction processing for multilevel replicated databases", Proc. of the European Symp. on Research in Computer Security, 1992.
7. W. Lampson, "A Note on the Confinement Problem", Comm. of ACM, October 1973.
8. L. LaPadula and D. Bell, "Secure computer systems: Unified Exposition and Multics Interpretation", The Mitre Corp., March 1976.
9. N. Soparkar, E. Levy, H. Korth and A. Silberchatz, "Adaptive Commitment for Real-Time Distributed Transactions", TR-92-15, Computer Science Department, University of Texas-Austin, 1992.