무결성 검사 트랜잭션을 위한 회복 기법
(Recovery Mechanism for Integrity Checking Transactions)

민 경 섭 † 이 강 우 †† 김 형 주 †††
(Kyungsub Min) (Kangwoo Lee) (Hyoung-Joo Kim)

요 약 최근 많이 부각되고 있는 여러 데이타베이스 응용들은 복잡하고 많은 무결성 규칙을 요구한다. 이러한 응용에서 갱신 트랜잭션은 많은 양의 무결성 규칙을 동반하며, 시스템 고장시 많은 수의 트랜잭션이 검사 단계에 있을 경우가 증가하여 철회될 가능성이 높아진다. 그러나 일반적으로 대부분의 트랜잭션 수행은 무결성 제약 규칙을 만족하므로, 무결성 검사를 통과하여 완료될 트랜잭션을 시스템 고장 회복시에 철회시키는 것은 시스템의 자원을 낭비하는 결과를 초래하게 된다. 이에 본 논문에서는 이러한 트랜잭션이 가지는 성질을 이용하여, 시스템 고장 시에도 철회되지 않고 계속해서 남은 작업을 수행하고 종료될 수 있도록 하는 방법을 제시하고, 실험을 통해 제안한 방법의 효용성을 보였다.

Abstract The new database applications need complicated integrity checking rules. Update transactions in new database applications require a large number of complicated integrity checkings. In case of system failure many transactions may be in state of integrity constraint checking phase and be highly likely rollbacked. Since, however, many transactions satisfy the integrity constraints generally, it causes the system resources to be wasted to rollback the transaction at system recovery phase. This paper proposes the recovery mechanism for integrity checking transactions. It does not rollback the integrity checking transaction at system recovery phase. After recovery phase, the system may continue to execute the remaining integrity constraints of the transaction. By simulation, we examine the validity of the proposed recovery mechanism.

1. 서 론

저장된 데이타를 보호하는 업무는 DBMS 에서 매우 중요한 기능이다. 일반적으로 DBMS 는 데이타 보안(security), 데이타 무결성(integrity), 데이타 일관성(consistency)의 세 가지 차원에서 데이타를 보호하게 된다[3].

이 중 데이타 무결성은 데이타베이스가 만족해야 하는 무결성 규칙(integrity rule)[1]을 설정하고, 데이타베이스에 갱신이 발생하면 변경된 데이타베이스 상태가 이미 설정된 무결성 제약 규칙들을 만족시키는지 여부를 조사하는 방법을 통해 얻어진다. 그러나 무결성 검사는 일반적으로 사용자의 실수, 어플리케이션 및 입력 자료의 오류 등으로부터 데이타의 무결성을 보호하는 것으로[13], 대부분의 경우에는 무결성 검사를 통과한다.

무결성 제약 조건은 트랜잭션 수행 중 매 갱신 수행 후마다 검사하는 방법과, 갱신 트랜잭션이 완료될 때 검사하는 방법[2] 두 가지가 있다. 전자의 경우 트랜잭션의 수행 과정에서 잠시동안 일관되지 않은 상태를 보이는 경우에도 이를 오류로 판단하여 트랜잭션이 철회되는 문제점을 갖는다[1]. 이러한 문제는 후자처럼 트랜잭션이 완료될 때 검사하면 자연스럽게 해결되고, 갱신한 최

· 본 연구는 정보통신부에서 지원하는 '미래로/DB통합시스템을 위한 SRP확장' 개발과제의 일부인 'SRP Server 확장 과정' 중에 수행되었음

† 학생회원 : 서울대학교 컴퓨터공학과
ksmin@gaston.snu.ac.kr
†† 비 회 원 : 서울대학교 전산과학과
kwlee@oopsla.snu.ac.kr
††† 종신회원 : 서울대학교 컴퓨터공학과 교수
hjkim@oopsla.snu.ac.kr

논문접수 : 1997년 8월 13일
심사완료 : 1999년 3월 4일

1) 이는 무결성 제약 규칙(integrity constraint rule), 무결성 제약(integrity constraint), 제약 규칙(constraint rule)이라고도 한다.
2) 트랜잭션 철회시에는 데이타베이스가 이전의 무결한 상태로 전이되므로, 이 작업이 불필요하다.

종 효과(net effect)를 검사하기 때문에, 불필요한 무결성 제약 조건의 검사를 생략할 수 있는 장점도 있다[6]. SQL 표준 SQL92 는 무결성 검사 작업을 완료시에 수행하는 것을 표준으로 정하고 있으며[2], 최근에 발표된 ORACLE 8 에서도 이를 따르고 있다. 이에 본 논문에서는 무결성 검사가 트랜잭션 완료시 수행되는 시스템을 가정한다.

최근에 새롭게 등장하는 의사결정(decision support)이나 데이타 웨어하우징(data warehousing) 등의 주요 데이타베이스 응용분야에서는 다양한 정보 관리 유지와 다양한 사용자의 요구를 충족시켜주기 위해서 데이타베이스가 유지해야 하는 의미적인 성질이 급속하게 증가하게 되어, 많은 수의 무결성 제약 조건을 필요로 하고, 매 갱신으로 인해 검사할 제약 조건의 수도 증가하게 된다. 또한 복잡한 의미적 성질로 인해, 개개의 무결성 규칙 검사에 필요한 시간이 길어지게 된다. 우리는 편의상 상대적으로 장시간의 무결성 검사가 동반되는 갱신 트랜잭션을 무결성 검사 트랜잭션(Integrity Checking Transaction: ICT)라 부르기로 한다.

무결성 검사 트랜잭션이 많은 시스템에서 시스템 고장이 발생하는 경우에는 많은 수의 ICT 들은 완료 단계에 수행하는 무결성 검사 단계에 있을 경우가 높게 된다. 기존의 트랜잭션 처리 방법에 의하면, 이러한 트랜잭션들은 고장 회복 중에 모두 철회되며, 이를 해결하기 위한 방법을 제안하고 있지 않다 그러나 앞서 언급한 바와 같이 일반적으로 대부분의 트랜잭션은 완료시 무결성 검사 조건을 만족하기 때문에, 이러한 트랜잭션에 대해 무조건 철회시키는 방법보다는 시스템 고장으로 중단된 무결성 검사를 계속하게 하여, 가능하면 완료시키는 것이 보다 합리적일 것이다.

이에 본 논문에서는 시스템 고장으로 무결성 검사 단계에서 중단된 ICT 들을 무조건 철회시키지 않고, 중단된 무결성 검사를 계속하게 하여, 가능한 ICT 를 완료시켜, 불필요한 트랜잭션 철회를 막는 방법을 제시하고, 제시된 방법의 타당성을 시뮬레이션을 통해 보이도록 한다. 제안된 방법은 현재 고장 회복 기법으로 많은 사용되고 있는 ARIES[14] 를 기반으로 설계되었다. 그러나 제안된 방법은 로그를 바탕으로 한 고장 회복 기법을 사용하는 기타의 시스템[5,10]에서도 사용될 수 있다.

본 논문의 구성은 다음과 같다. 2장은 ICT 와 장시간 수행 트랜잭션 간의 공통점과 대비적인 특징들을 살펴본다. 3장에서는 2장에서 기술한 ICT의 특징을 바탕으로 무결성 검사 단계에 있는 ICT 를 처리하는 방법을 제시한다. 4장에서는 3장에서 제시한 기법을 ARIES 로 접합하는 과정을 설명한다. 5장에서는 제시한 방법의 타당성을 실험을 통해 보인다. 마지막으로 6장에서 결론을 기술한다.

2. ICT의 특성

ICT는 실제 수행되는 트랜잭션과 관련된 많은 무결성 검사 작업을 수행함으로써 장시간 수행 트랜잭션의 특성을 띠게 된다

일반적인 장기간 수행 트랜잭션의 전형적인 문제점은 첫째로, 트랜잭션 수행 시간의 장기화에 따른 다른 트랜잭션들과 충돌 증가 현상과 이에 따른 장시간의 대기 문제와, 교착상태의 발생 횟수의 증가와 이에 따른 트랜잭션의 철회율의 증가를 들 수 있고, 둘째로는 수행시간의 장기화로 시스템 고장이 발생하는 경우, 수행이 완료되지 않은 상태로 남아있게 될 가능성이 매우 높아져 해당 트랜잭션의 철회로 인해 장기간 동안 수행해온 모든 결과의 무효화 작업을 해야 하는 문제를 들 수 있다[9]. 많은 연구자들은 장기간 수행 트랜잭션이 철회될 때 발생하는 여러 문제점을 해결하기 위해 많은 노력을 기울여왔으며, 많은 해결법을 제시하였으나[8, 7, 11], 아직까지는 일반적인 장기간 수행 트랜잭션의 문제점을 해결할 수 있는 해법을 제시하지 못했으며, 실제로 사용하기에도 많은 제약점을 갖고 있다[11].

하지만, ICT는 다음과 같은 점에서 다른 장시간 수행 트랜잭션과 다른 특성을 가진다.

첫째, 무결성 검사에 필요한 모든 규칙들은 트랜잭션이 수행되는 과정과 완료 시점에서 결정된다. 즉, 트랜잭션이 성공적으로 완료되기 위해 필요한 모든 제약 규칙들은 실제 무결성 검사 시작 이전에 모두 결정된다.

둘째, 무결성 검사 단계에서는 데이타의 갱신 작업이 발생되지 않으므로, 연쇄적인 무결성 규칙 검사 작업은 발생되지 않으며, 반복적인 무결성 검사는 부수-효과(side-effect)를 발생시키지 않는다.

셋째, 무결성 검사는 트랜잭션 완료 단계에서 수행되기 때문에, 무결성 검사를 마친 트랜잭션은 바로 완료 또는 철회하는 작업만을 남기게 된다. 즉, 무결성 검사는 트랜잭션을 구현한 프로그램의 마지막 단계에서 수행되기 때문에, 트랜잭션의 추후 수행에 영향을 주는 프로그램 수준의 문맥 정보는 없다.

넷째, 검사할 무결성 규칙들은 데이타베이스 갱신에 의해 얻어지므로, 트랜잭션의 수행 중에 저장한 로그 레코드들만으로 트랜잭션이 완료하기 위해 필요한 제약 규칙들을 결정할 수 있다. 그러므로, 무결성 검사 단계

에서 고장이 발생하여, 재 수행되는 트랜잭션은 수행 중에 발생시킨 로그 레코드외의 추가의 문맥 정보는 필요 없다.

다섯째, 무결성 검사에 필요한 록은 대부분 cursor stability 수준을 사용할 수 있어, 록의 충돌은 일반적인 장기간 수행 트랜잭션의 그것보다는 완화될 수 있다.

이와 같은 특성들은 위에서 기술한 장시간 수행 트랜잭션의 문제점들을 완화하며, ICT를 위한 회복 기법을 제안하기 위한 토대를 제공한다.

3. ICT 에 대한 철회 기법

본 장에서는 시스템 고장시 무결성 검사 단계에 있는 ICT 를 단순히 철회시키지 않고, 남은 무결성 검사 단계를 마치도록 하여, 불필요한 트랜잭션 철회를 막는 방법을 제안한다.

3.1 ICT 를 위한 철회 기법: 기본 구상

제안되는 방법은 전 장에서 기술한 ICT 가 갖는 특징을 바탕으로 설계되었다.

첫째, 완료 준비(prepare to commit) 단계에 들어간 ICT 는 무결성 검사 단계에서 검사해야 할 모든 무결성 규칙을 알 수 있으며[6], 무결성 검사 단계 중에는 데이타 갱신이 없어 연쇄적으로 검사해야 할 무결성 검사가 생기지 않으므로, 준비 단계에 들어간 ICT 에 대해 앞으로 검사해야 할 무결성 규칙들을 준비(prepare) 로그 레코드에 저장한다면, 시스템 고장으로 중단된 무결성 검사 규칙들은 해당 ICT 의 준비 로그 레코드를 보고 모두 알 수 있다.

둘째, 무결성 검사는 데이타 갱신 작업이 없으므로, 이 단계는 idempotent 한 성질을 갖는다. 즉, 시스템 고장으로 인해 무결성 검사가 중단된 ICT 에 대해서는 준비 로그 레코드에 저장된 검사할 무결성 규칙들을 이용하여, 무결성 검사를 무조건 처음부터 재 수행하여도 시스템의 일관성을 저해하지 않는다.

셋째, 무결성 검사는 트랜잭션의 완료 단계에서 수행되므로, 시스템 고장후 회복 시에 무결성 검사를 재개한 트랜잭션은 추가로 수행할 작업이 고정되어 있기 때문에, 추가의 전방(rollforward) 작업을 위한 트랜잭션의 지속적 문맥 관리가 필요 없다.

위의 특징을 바탕으로 한 ICT 고장 회복 방법의 기본 개요는 다음과 같다. 정상 수행시, 무결성 검사를 시작하는 트랜잭션은 먼저 검사해야 할 무결성 규칙을 준비 로그 레코드에 저장하고 무결성 검사를 수행한다. 무결성 검사를 통과하고, 기타 준비 단계를 마친 트랜잭션은 완료하게 되고, 무결성 검사를 실패한 트랜잭션은 철회하게 된다.

시스템 고장이 발생하여 고장 회복이 수행되면, 시스템은 먼저 분석 단계에서 준비 로그 레코드는 있으나, 아직 완료나 철회 로그 레코드가 없는 ICT 를 찾는다. 이 트랜잭션에 대해, 재 수행 단계에서는 저장된 준비 로그 레코드에 기록된 무결성 규칙들을 이용하여 무결성 검사를 재 수행시켜, 모든 검사를 통과하면 완료시키고, 검사에 통과하지 못하면 다음에 수행되는 무효화 단계에서 철회시킨다(그림 1 참조).

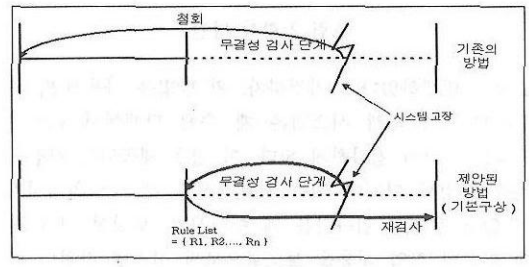


그림 1 기본 구상을 사용한 재시작 고장 회복시 ICT 복구

무결성 검사 규칙들은 데이타베이스의 갱신을 통해서만 발생하고, 데이타베이스에 대한 갱신은 순차적으로 로그에 기록된다. 그러므로 고장 회복의 분석 단계에서 참조하는 로그 레코드를 통해, 필요한 무결성 검사 규칙들을 얻을 수 있어, 준비 로그 레코드에 검사할 무결성 규칙을 저장하지 않아도 된다.

또한, 로그에 저장하는 무결성 검사 규칙들은 실제로는 이미 메타 파일의 형태로 시스템에 저장되어 있으므로, 로그 레코드에 저장되는 정보는 트랜잭션의 수행 중에 얻어진 검사할 무결성 규칙을 유일하게 식별할 수 있는 식별자 정보가 되어, 로그 레코드의 크기는 그리 증가하지 않는다.

3.2 ICT를 위한 철회 기법: 향상 방안

앞선 기본 구상은 추가의 로그 레코드나, 지속적 데이타의 변경을 필요로 하지 않고, 분석 단계에서 약간의 알고리즘 수정을 바탕으로 하고 있다. 위 기본 구상은 로그를 기반으로 한 대부분의 시스템에 적용 가능하다는 장점을 갖는다.

그러나, 위 방법은 시스템 고장으로 인해 불필요하게 철회되는 ICT를 막을 수는 있지만, 이미 검사가 끝난 무결성 규칙이 있는 경우에도 무결성 검사를 처음부터 다시 검사해야 하는 단점을 가지고 있다.

예를 들어, 트랜잭션 T가 완료 단계에서 검사할 총 10개의 규칙에서 9개의 규칙을 통과한 상태에서 시스템

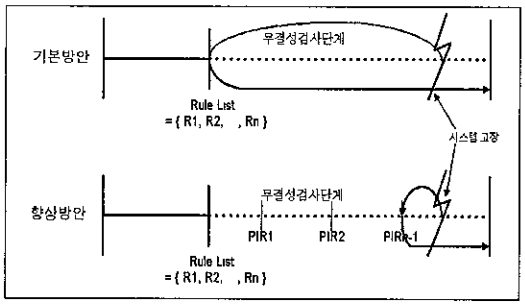


그림 2 향상 방안

고장이 발생하였다고 가정하자. 위 방법을 사용하면, 트랜잭션 T에 대해 시스템은 재 수행 단계에서 10개의 규칙들을 다시 검사하게 된다. 이 경우 제공되는 장점은 무결성 검사 단계에 있는 트랜잭션을 무조건 철회시키지 않고 무결성 검사만을 재 수행시켜, 무결성 검사 단계 이전의 작업 내용을 보호함으로써 시스템 자원의 낭비를 막는다는 점이다. 만일, 우리가 예상하는 응용분야와 같이 복잡한 무결성 검사가 잦은 경우에는 무결성 검사에 필요한 비용이 트랜잭션 전체 수행 비용의 상당 부분을 차지하기 때문에, 위 방법으로 얻을 수 있는 이익은 상대적으로 작다. 제안된 방법이 기존의 방법보다 많은 비용을 절약하기 위해서는 정상 수행 중 이미 통과한 무결성 검사 규칙을 고장 회복 단계에서 다시 수행하는 일을 막도록 하는 것이 중요하다.

이를 위해, 우리는 트랜잭션 완료 단계에서 트랜잭션이 각 무결성 제약 규칙 검사를 통과할 때마다, 통과된 제약 규칙의 식별자를 로그 레코드 - 우리는 이를 무결성 검사 통과 로그 레코드(Passed Integrity Rule: PIR 로그 레코드)라 부른다. - 에 저장하는 방식을 제안한다. PIR 로그 레코드는 지속 저장점의 역할을 수행하는 로그 레코드로서, 고장 회복시 재 수행 단계에서, 전체 무결성 규칙을 다시 검사하지 않고, 저장된 PIR 로그 레코드를 검사하여, 검사할 규칙들 중 PIR 로그 레코드에 기록된 규칙들은 제외시킬 수 있도록 해준다(그림 2 참조). 검사할 무결성 조건이 과도한 경우에는, 심한 PIR 로그 레코드 생성을 막기 위해, 제약 규칙 그룹 단위로 하나의 PIR 로그 레코드를 생성할 수도 있다. 이 방법을 사용하면, 시스템 고장시 이미 통과한 규칙들을 제외한 나머지 규칙들만을 고장 회복 과정에서 검사하게 되므로, 앞서의 방법보다 자원 낭비를 줄일 수 있는 장점을 갖게 된다.

한편, 이 방법은 단점으로 새로운 로그 레코드의 추가와, 무결성 검사 단계에서 로그 레코드의 생성으로 인한 약간의 검사 시간 증가를 가져올 수 있다. 그러나, 제시된 방법은 무결성 검사 단계의 idempotency 성질을 이용하므로, PIR 로그 레코드가 반드시 생성 즉시 로그 장치에 강제로 쓰여질 필요가 없이, 로그 버퍼에서 여러 PIR 로그 레코드가 한번에 저장되어도 무방하기 때문에, 시스템의 큰 부담은 되지 않는다. 물론, PIR 로그 레코드를 로그 장치에 동기화된 쓰기를 사용하지 않는 경우에는, 시스템 고장시 PIR 로그 레코드가 로그 버퍼에 남아 있을 경우가 있어, 일부 규칙은 이미 통과되었어도 고장 회복 과정에서 다시 검사될 수 있다.

앞선 방식의 또 다른 단점으로는 고장 회복 과정에서 무결성 검사 과정을 완료하지 못한 트랜잭션의 남은 무결성 검사 작업을 재 수행 단계에서 다시 수행하도록 하기 때문에, 고장 회복 과정에 소요되는 시간이 길어지는 문제점을 들 수 있다. 시스템의 가용성(availablity)이 중요한 시스템의 경우에는 빠른 시간 내에 시스템 고장 회복을 완료해야 하기 때문에, 제안된 방법은 이러한 시스템의 경우에는 가용성을 저하시키는 요인을 제공한다.

우리는 이러한 문제점을 해결하기 위해, 남은 무결성 검사를 고장 회복 과정의 재 수행 단계에서 수행하지 않고, 고장 회복을 완료한 후 다른 일반 트랜잭션들과 함께 수행시키는 방법을 채택/제안한다. 남은 무결성 규칙들에 대해 검사를 통과하지 못한 트랜잭션은 일반적인 트랜잭션 철회 방법에 의해 철회된다.

이 방법을 사용하기 위해서는 무결성 검사 작업이 남은 트랜잭션에 대해서, 해당 트랜잭션이 시스템 고장 당시 갖고 있던 모든 락(lock)을 유지시켜 주어야 한다.

이를 위해, 트랜잭션 완료 단계에서 저장되는 준비 로그 레코드에, 해당 트랜잭션이 유지하고 있던 모든 락 정보를 기록하는 방법을 사용한다. 이 방법은 이미 일부 DBMS 의 2단계 완료 규약에서 사용하는 기법을 구현하기 위해 사용하는 기법[15]이기 때문에, 이를 지원하는 시스템에 대해서는 시스템 수정으로 인한 추가의 부담이 필요없다.

4. 회복 기법의 ARIES 로의 응용

4.1 ARIES

이 절에서는 ICT 회복 기법을 적용하기 위해, 기존에 제안된 회복 기법중 기본적이며 강력한 기능을 가지고 있는 ARIES에 대해 먼저 간략히 살펴보도록 한다.

ARIES 는 로그(log)를 이용하여 데이타베이스의 변화를 추적하며, 데이타베이스에 수행된 작업에 의해 변경된 내용보다 변경에 대한 정보를 기록한 로그 레코드가

먼저 로그에 기록되도록 보장하는 WAL(Write-Ahead Logging) 프로토콜을 사용한다.

트랜잭션들은 부분 철회(partial rollback)나 전체 철회(total rollback)와 같은 무효화 작업(undo work) 과정을 효과적으로 수행하기 위해서 정상처리 과정이나 시스템 고장 시, 고장 회복(restart recovery) 과정에서 보상 로그 레코드(CLR: Compensation Log Record)라는 특별한 종류의 로그 레코드를 로그에 저장한다. CLR은 무효화 단계(undo phase)에서 사용되는 로그 레코드로서 다음에 무효화할 작업을 가리키는 포인터(Undo-NxtLSN)를 적절히 사용하여 이미 무효화된 작업들은 다시 무효화하지 않도록 해주어 빠른 회복을 보장해 준다.

갱신에 관련된 로그 레코드는 실제 데이타의 물리적 이미지를 저장할 뿐아니라 수행된 행위를 저장할 수도 있다. 이는 CLR 이 이전에 수행된 작업에 대해 물리적 역 연산으로 무효화 작업을 수행하지 않을 수 있음을 나타낸다(logical undo).

ARIES는 다음과 같은 세 가지 중요한 자료구조를 유지한다. 먼저 디스크의 각 데이타 페이지의 헤더에 page_LSN 을 두어, 가장 최근의 페이지 갱신으로 생성된 로그 레코드의 LSN 을 유지한다. 트랜잭션 테이블에는 DBMS 가 수행하고 있는 트랜잭션들에 대한 정보를 유지한다. 마지막으로 갱신 페이지 테이블(dirty pages table)에는 버퍼에 적재된 페이지들 중 디스크로 쓰여지지 않은 갱신된 페이지에 대한 정보를 유지한다.

위의 자료 구조를 이용하여 시스템은 트랜잭션들을 처리한다. 갱신이 발생하면, 갱신 작업이 수행되는 과정에서 트랜잭션 테이블, 갱신 페이지 테이블, 수정된 페이지의 page_LSN 이 수정된다. 만약 수행 중이던 트랜잭션에 교착상태, 에러 발견과 같은 상황이 발생하면 수행 중이던 트랜잭션은 철회된다. 철회시 저장점 설치 여부에 따라 부분 철회 또는 전체 철회를 한다.

시스템은 트랜잭션을 처리하는 과정에서 시스템 고장 발생시 회복 작업을 줄이기 위해 주기적으로 검사점(checkpoint)을 설치한다. 검사점 설치 시에는 현재의 시스템 수행 상태, 즉 트랜잭션 테이블, 갱신 페이지 테이블 및 필요한 정보들이 로그에 저장된다.

시스템 고장이 발생하면 시스템은 고장 회복 과정을 수행한다. 고장 회복 과정은 먼저 가장 최근에 설정한 검사점부터 로그를 읽음으로써 복구할 대상 트랜잭션들과 데이타 페이지를 분석한다(분석 단계). 다음, 시스템 상태 정보를 바탕으로 데이타베이스를 시스템 고장이 발생하기 전 상태를 만드는 작업을 수행한다. 이는 로그에서 재 수행 가능한 로그 레코드를 순차적으로 수행시킴으로서 얻을 수 있다(재수행 단계). 마지막으로, 위의 과정에서 얻은 데이타베이스에서 아직 완료되지 못한 트랜잭션이 수행한 작업을 모두 철회시킴으로써 데이타베이스를 일관된 상태로 복구한다(무효화 단계).

ARIES는 이외에도 중첩 최상위 연산(nested top action), 선택적 재 시작(selective restart), 지연 재 시작(deferred restart), 퍼지 이미지 복사(puzzy image copy), 매체 회복(media recovery), 세밀한 동시성 제어 등을 지원할 수 있다. 그리고 사용하는 버퍼 관리 방식(예, steal, no-force, 등[12])에도 유연성 있게 결합될 수 있다.

4.2 ARIES에의 적용

이 절에서는 앞서 제안한 방법을 ARIES 를 바탕으로 구현할 때 정상 작동 및 고장 회복 과정에서 수정이 필요한 수행 단계를 설명한다.

무결성 검사 단계 트랜잭션 처리를 위해 시스템은 각 규칙이 통과할 때마다 생성할 로그 레코드, 즉 전 장의 PIR 로그 레코드가 필요하다. 이 로그 레코드는 redo-only 로그 레코드로, 무결성 검사 단계에서 통과한 무결성 규칙의 식별자가 기록된다. 이 로그 레코드를 이용하여 고장 회복중 분석 단계에서 중단된 무결성 검사중 이미 통과한 규칙을 제외시킬 수 있다.

시스템의 정상 수행시 또한 고장 회복 과정에서 트랜잭션의 상태를 기록하는 트랜잭션 테이블의 State 필드에 무결성 검사 단계(IC_Check)를 추가하고, 무결성 검사 단계시 검사할 무결성 검사 규칙을 저장하는 필드 RuleList 를 추가로 갖는다.

4.2.1 정상 처리 과정(Normal Processing)

시스템은 갱신 트랜잭션이 준비 단계로 들어갈 때, 트랜잭션 테이블의 해당 트랜잭션의 RuleList 를 준비 로그 레코드에 기록하고, 준비 로그 레코드를 로그에 강제로 저장하므로써 무결성 검사 단계를 시작한다.

무결성 검사 단계에 들어선 트랜잭션은 검사에 필요한 규칙들을 트랜잭션 테이블의 RuleList를 참조하여 수행한다. 검사할 각 규칙이 통과할 때마다, PIR 로그 레코드를 생성한다. 만일 여러 규칙에 대해 하나의 PIR 로그 레코드를 생성하는 경우에는 통과한 모든 규칙의 식별자를 PIR 로그 레코드에 기록한다. 검사할 모든 규칙을 통과하면, 기타 남은 준비 작업을 수행한 후, 트랜잭션은 완료 로그 레코드를 로그에 저장하고, 완료된다. 검사에 실패하게 되면, 트랜잭션은 철회되고, 철회 로그 레코드가 로그에 저장되고 트랜잭션은 종료된다.

검사점 설치시 ARIES 에서는 트랜잭션 테이블의 내

용을 검사점 로그 레코드에 저장하게 되는데, 이때 수행 중인 각 트랜잭션의 RuleList 필드도 함께 검사점 로그 레코드에 저장한다. 검사점 로그 레코드에 기록된 RuleList 들은 고장 회복 시에 사용된다.

4.2.2 고장 회복 과정(Restart Recovery)

고장 회복 과정은 기존의 ARIES 와 같이 가장 최근에 성공적으로 저장된 검사점 로그 레코드부터 시작하여 분석 단계, 재 수행 단계, 무효화 단계를 거쳐 실시한다. 이 절에서는 ICT 고장 회복을 위해 추가된 부분만을 설명한다.

분석 단계에서는 매 로그 레코드를 생성 시간대로 읽어 가면서, 읽은 로그 레코드에 따라 분석 작업을 수행한다. 읽은 로그 레코드가 검사점 로그 레코드인 경우에는 검사점 로그 레코드에 저장된 트랜잭션 테이블을 읽어 들여 트랜잭션 테이블에 반영시킨다. 이때, 트랜잭션 테이블 내의 RuleList 도 반영된다. 읽은 로그 레코드가 준비 로그 레코드인 경우에는 준비 로그 레코드에 기록된 검사할 무결성 검사 규칙들을 읽어 해당 트랜잭션의 RuleList를 갱신하고, 트랜잭션의 상태(State)를 IC_Check로 바꾼다. 읽은 로그 레코드가 PIR 로그 레코드인 경우에는 PIR 로그 레코드에서 통과된 검사 규칙을 읽어, 해당 트랜잭션의 RuleList 에서 제거한다. 이러한 방법으로 RuleList에는 트랜잭션이 완료시 검사해야 할 무결성 규칙 중 아직 검사하지 못한 무결성 규칙만이 남게 된다.

재 수행 단계에서는 기존 ARIES의 재 수행 단계를 모두 수행한 후, 트랜잭션 테이블을 조사하여, State가 IC_Check인 각 트랜잭션에 대해 트랜잭션 테이블의 RuleList 필드를 참조하여, 남은 무결성 규칙을 계속 수행한다. 남은 무결성 검사를 모두 통과한 트랜잭션은 완료 로그 레코드를 저장한 뒤, 완료시킨다. 만일 무결성 검사에서 실패한 트랜잭션은 해당 트랜잭션의 상태를 단순히 진행 중인 상태로 두어, 다음으로 이어질 무효화 단계에서 자동적으로 철회될 수 있도록 한다.

무효화 단계에서는 기존의 ARIES에서의 무효화 단계에 수정이 필요 없이, 단순히 트랜잭션 테이블에 기록된 트랜잭션 중 아직 종료되지 않은 모든 트랜잭션을 철회시키면 된다.

중단된 트랜잭션의 무결성 검사를 고장 회복 과정이 완료된 후, 정상 작동 시에 수행시키려면, 분석 단계에서 조사된 무결성 검사 단계의 트랜잭션에 대해 해당 트랜잭션이 갖고 있던 모든 락을 재 설정해 주고, 무효화 단계에서 철회되지 않도록 하기 위해 상태를 IC_Check로 둔다.

이 절에서 볼 수 있듯이, 제안된 방법은 기존의 ARIES 고장 회복 방법에 매우 적은 수정을 통해 구현될 수 있음을 알 수 있다.

5. 성능평가

본 장에서는 ICT에 대한 회복 과정의 성능을 알아보기 위한 성능 평가 실험 모델을 제시하고, 그 결과를 보이도록 하겠다. 제안된 방법의 성능 평가를 위해서 본 논문에서는 프로세스 중심 시뮬레이터인 C++SIM[4]을 사용했다.

5.1 성능 평가 모델

그림 3 은 본 성능 평가에 사용되는 실험 모델이다. 시뮬레이션의 작업(workload)는 Client에 의해 생성된다. 하나의 Client는 한 순간에 하나의 트랜잭션을 생성시키며, 하나의 트랜잭션의 수행이 끝나면, 계속하여 새로운 트랜잭션을 생성시키는 작업을 반복한다. Client의 수는 곧 동시에 수행되고 있는 트랜잭션의 수를 의미하게 된다. Client 에 의해 생성되는 각 트랜잭션은 0번에서 5번까지의 갱신 작업을 수행한다고 설정하였다. 이것은 본 시뮬레이션이 ICT를 위한 고장 회복과 기존의 고장 회복과의 성능 비교를 위한 것이므로, 갱신 작업이 자주 발생하는 환경을 고려한 것이다. 하지만, 트랜잭션의 크기는 계좌이체와 같은 간단한 작업을 기준으로 하여 약간 크거나 작은 것을 고려하였다. 갱신 트랜잭션이 완료될 때 검사해보아야 할 규칙의 수는 1개에서 7개 중 임의로 선택하도록 하였다. 이것은 기본키, 널, 유일성, 값에 대한 제약 등의 존재 여부와 간단한 트리거의 존재 여부를 고려한 것이다. Failure Generator는 시스템 고장을 유발하기 위한 개체이다. 시스템 고장 빈도수는 수행되는 각 트랜잭션에 대해 약 1% 확률로 시스템 고장으로 정지되게 하였다. 이 값은 [11]에 근거한값

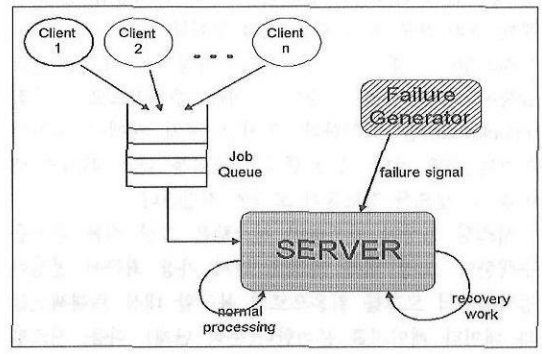


그림 3 실험 모델

이다. Server는 데이타베이스 시스템을 시뮬레이션하는 개체로, 고장 회복 알고리즘을 수행한다. Server는 Queue를 통해 Client 로 들어오는 트랜잭션 연산에 대한 로그를 생성하고, Failure Generator로부터 시스템 고장을 요구받는 경우에는 고장 회복을 수행하여, 소요되는 고장 회복 시간을 계산한다. 검사점 설치는 Server에 의해 생성되는 로그레코드의 양에 의해 결정된다. 기타 위 실험에서 사용하는 주요 실험 인자들과 그것들의 설정 값은 표 1에 설명되어 있다.

표 1 실험 인자와 설정 값

실험 인자	설정값
검사점 설치 주기	300개의 로그레코드가 저장된 시점
디스크 접근 시간	10ms 에서 30ms
트랜잭션당 갱신 작업	0에서 5번
갱신 작업당 무결성 규칙 수	1개에서 7개
시스템 고장 빈도율	수행되는 전체 트랜잭션의 1%

5.2 성능 평가 결과

그림 4, 5, 6, 7은 각각 동시 수행 트랜잭션의 수가 대략 10, 20, 30, 40 일 때, 시스템 고장 발생시 회복 시간을 측정한 것이다. 그림에서 IC_RECOVERY 로 표시된 그래프는 본 논문에서 제시한 과정을 사용했을 때의 회복 시간을 나타낸 것이며, NO IC_RECOVERY는 기존의 과정을 사용했을 때의 회복 시간을 나타낸다. 종축의 트랜잭션 수는 시스템 고장이 발생하는 간격을 나타내는 것으로, 두 점간의 간격은 시스템 고장이 발생하는 간격이라 할 수 있다. 이는 대략 동시에 수행되는 트랜잭션 수와 수행된 트랜잭션 수간의 1% 정도되는 시점을 나타낸다. 횡축은 각 시스템 고장 시점에서의 회복 시간을 나타낸다.

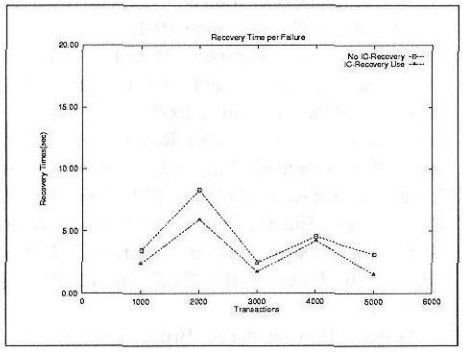


그림 4 트랜잭션 10개를 동시에 수행할 때

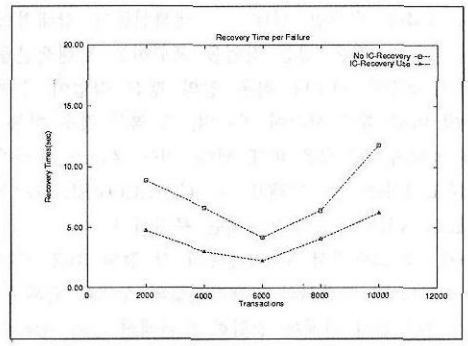


그림 5 트랜잭션 20개를 동시에 수행할 때

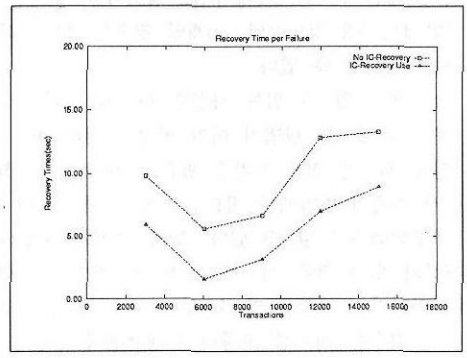


그림 6 트랜잭션 30개를 동시에 수행할 때

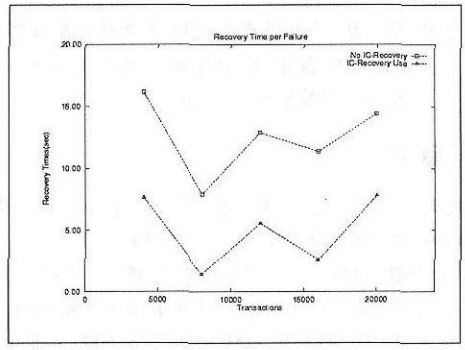


그림 7 트랜잭션 40개를 동시에 수행할 때

각 그림에서 같은 그래프 내에서의 각 시스템 고장 후 회복 시간의 격차는 시스템 고장 시에 수행되던 트랜잭션들의 갱신 작업 포함 수, 즉 트랜잭션의 길이, 와 그 때까지 수행 완료된 각 트랜잭션 내의 갱신 작업의 수에 따라 나타나는 현상이다. 갱신 작업의 수가 많고,

많은 작업을 수행한 상태의 트랜잭션들이 상대적으로 많은 경우는, 적은 갱신 작업을 포함하는 트랜잭션들이 많이 수행중인 경우나 적은 양의 갱신 작업이 수행된 경우에 비해 회복 시간이 커지며, 그 역의 경우 회복 시간이 적어지는 특성을 띠게 된다. 이는 우리가 중점적으로 보고자하는 두 그래프 - IC_RECOVERY와 NO IC_RECOVERY - 의 비교와는 무관하다.

한편, 각 그림에서 두 그래프의 각 점의 회복 시간의 격차는 시스템 고장시의 트랜잭션들의 상태를 말해준다. 즉, 무결성 검사 단계에 진입한 트랜잭션 수가 많을수록 그 차이가 커지게 되며,대략 10개의 트랜잭션이 동시에 수행되는 경우는 두 시스템간의 회복시간의 비가 4:3 정도이며, 20개인 경우는 2:1에서 3:2 정도이며, 30개인 경우는 2:1에서 5.3 정도이며, 40개인 경우는 2:1 이하로 나타나는 것을 볼 수 있다.

이 결과에서 알 수 있는 사실은 다음과 같다. 첫째, 본 논문에서 제시한 방법이 이전 방법에 비해 항상 더 적은 양의 시스템 회복 시간을 필요로 한다. 이는 회복 과정에서 기존의 방법에서 필요로 했던 철회 작업의 감소로 비롯되었다고 할 수 있다. 둘째, 동시에 수행되는 트랜잭션의 수가 많은 시스템일수록 두 방법간의 회복 시간의 차이가 증가함을 알 수 있다. 이는 동시 작업의 수가 많아질수록 시스템 고장시 트랜잭션이 무결성 검사 단계에 있게될 확률이 증가함을 나타내며, 이로써 무결성 검사 단계에 있는 트랜잭션의 철회가 불합리함을 알 수 있다.

이로써, 무결성 검사 단계에 있는 트랜잭션은 회복 단계에서 철회시키지 않고, 본 논문에서 제시한 방법을 적용하는 것이 바람직함을 알 수 있다.

6. 결 론

복잡하고, 많은 수의 무결성 제약 조건을 갖고 있는 데이타베이스 응용에서의 갱신 트랜잭션은 트랜잭션 완료시 검사할 많은 수의 규칙으로 트랜잭션 완료 준비 단계시 수행되는 무결성 검사 단계가 길어지게 된다. 장기간의 트랜잭션 무결성 검사 단계는 시스템 고장시 보다 많은 수의 트랜잭션이 검사 단계에서 있을 경우가 증가된다. 일반적으로 완료하려는 트랜잭션은 대부분 무결성 검사 단계를 통과는 경우가 높다는 점에 있어서, 고장 회복시 이러한 트랜잭션들을 무조건 철회시키는 것은 불필요한 트랜잭션 철회에 따른 시스템 자원의 낭비를 초래한다.

본 논문에서는 무결성 검사 트랜잭션이 갖는 성질을 이용하여, 고장 회복 시 이러한 트랜잭션들에 대해 시스템 고장으로 중단된 무결성 검사를 계속하게 하여, 완료할 수 있도록 하는 방법을 제안하였다. 제안된 방법은 ARIES에 최소한의 확장으로 설계되었으나, 로그를 기반으로 한 대부분의 고장회복 방식에서 사용될 수 있는 장점을 갖는다.

참 고 문 헌

[1] M. M. Astrahan et al. "System R: A Relational Approach to Database Management". ACM Transactions on Database Systems, 1(2):97-137, June 1976.

[2] C.J. Date with Hugh Darwen. "A Guide to The SQL Standard : Third Edition", Addison Wesley, 1993

[3] Don Chamberlin, editor. Using the new DB2 : IBM's object-relational database system. Data Management Systems. Morgan Kaufmann Publishers, Inc., San Mateo, CA, 1996.

[4] Computing Lab., University of Newcastle. C++SIM, 1990.

[5] R. A. Crus. "Data recovery in IBM Database 2", IBM Systems Journal, 23(2):178-188, 1984.

[6] Jennifer Widom and Sheldom J Finkelstein. "Set-Oriented Production Rules in Relational Database Systems". In Proc. of the ACM SIGMOD Conf. on Management of Data, pages 259-270, 1990

[7] Umeshwar Dayal, Meichun Hsu, and Rivka Ladin "A Transaction Model for Long-Running Activities". In Proc. of the Conf. on VLDB, pages 113-122, 1991.

[8] Hector Garcia-Molina. "Sagas". In Proc. of the ACM SIGMOD Conf. on Management of Data, pages 249-259, 1987.

[9] Jim Gray. "The Transaction Concepts: Virtues and Limitations". In Proc. of the Conf. on VLDB, pages 144-154, September 1981.

[10] Jim Gray et al. "The Recovery Manager of the System R Database Manager". ACM Computing Surveys, 13(2):223-242, June 1981.

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

[12] Theo Haerder and Andreas Reuter. "Principles of Transaction-Oriented Database Recovery". ACM Computing Surveys, 15(4):287-317, December 1983.

[13] Michael M. Hammer and Dennis J. Mcleod. "Semantic Integrity in a Relational Database System". In Proc. of the Conf. on VLDB, pages 25-47, 1975.

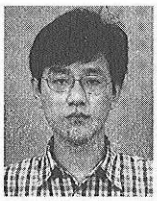
[14] C. Mohan, Don Haderle, Bruce Lindsay, Hamid Pirahesh, and Peter Schwarz. "ARIES: A Transaction Recovery Method Supporting Fine-

Granularity Locking and Partial Rollbacks Using Write-Ahead Logging". ACM Transactions on Database Systems, 17(1):94-162, March 1992.

[15] C. Mohan, B. Lindsay, and R. Obermarck. "Transaction Management in the R* Distributed Database Management System". ACM Transactions on Database Systems, 11(4):378-396, December 1986.

민 경 섭

1995년 한국항공대 전자계산학과 학사 졸업. 1997년 서울대 컴퓨터공학과 석사 졸업. 현재 서울대 인지과학 박사과정. 관심분야는 Transaction Processing, Workflow Modeling/processing



이 강 우

1991년 2월 서울대학교 계산통계학과(이학사). 1993년 2월 서울대학교 계산통계학과 전산과학전공(이학석사). 1993년 3월부터 현재까지 서울대학교 전산과학과 박사과정. 관심분야는 트랜잭션 처리 시스템, 질의어 처리 시스템, 객체관계형 데이타베이스 시스템

김 형 주

제 26 권 제 1 호(B) 참조