객체관계형 데이타베이스 시스템에서의 새로운 고립화 수준
(A New Isolation Level in Object-Relational DBMSs)

서 홍 석 † 장 지 웅 † 문 양 세 † 황 규 영 †† 홍 의 경 †††
(Hong-Suk Seo) (Ji-Woong Chang) (Yang-Sae Moon) (Kyu-Young Whang) (Eui-Kyung Hong)

요 약 데이타베이스의 성능향상을 위하여 관계형 DBMS에서는 엄정 이단계 로킹을 따르는 고립화 수준 3대신 고립화 수준 2의 변형인 커서 안정성이 유용하게 사용되어 왔다. 그러나, 객체 관계형 DBMS(object relational DBMS: ORDBMS)의 탐색항해 응용에 대해서 허상 포인터 문제, 갱신 분실 문제, 그리고 일관성을 잃은 복합객체를 읽는 문제와 같은 심각한 일관성 문제로 인하여 커서 안정성은 ORDBMS에서는 더 이상 유용한 수준이 되지 못한다. 본 논문에서는 ORDBMS에서 수준 3의 동시성 저하를 피하는 동시에 커서 안정성의 일관성 문제를 해결하는 새로운 고립화 수준인 탐색항해 안정성(*navigation stability*)을 제안한다. 먼저, 탐색항해 응용에 대한 커서 안정성의 일관성 문제를 분석한다. 다음으로, 커서 안정성을 확장하여 탐색항해 안정성을 정의하고, 탐색항해 안정성이 ORDBMS의 탐색항해 응용에 대한 커서 안정성의 일관성 문제를 일으키지 않음을 증명한다. 마지막으로, 성능 평가를 통해 수행시간이 긴 트랜잭션의 경우에 탐색항해 안정성은 수준 3에 비해 성능을 최대 200%까지 향상시키고, 평균 응답 시간을 최대 55% 줄이며, 트랜잭션의 철회율을 최대 77% 줄임을 보였다. 이러한 결과로부터, 탐색항해 안정성은 ORDBMS에서 일관성을 거의 희생하지 않고도 성능 향상을 위해 수준 3 대신 사용할 수 있는 유용한 고립화 수준임을 의미한다.

Abstract In order to enhance the performance, cursor stability, which is a variant of isolation level 2 in relational DBMSs, has been widely used in place of isolation level 3, which uses strict two phase locking. However, cursor stability is much less usable in object-relational DBMSs (ORDBMSs) because navigational applications in ORDBMSs can suffer from critical inconsistency problems such as dangling pointers, lost updates, and reading inconsistent complex objects. In this paper, we propose a new isolation level, navigation stability, that prevents the inconsistency problems of cursor stability for navigational applications, while avoiding significant degradation of the concurrency of level 3. First, we analyze the inconsistency problems of cursor stability for navigational applications. Second, we define navigation stability as an extension of cursor stability and show that it solves those inconsistency problems of cursor stability in ORDBMSs. For workloads consisting of transactions of long duration, compared with level 3, the throughput of navigation stability is enhanced by up to 200%; the average response time reduced by as much as 55%; and the abort ratio reduced by as much as 77%. From these results, we conclude that navigation stability is a useful isolation level in ORDBMSs that can be used in place of isolation level 3 to improve the performance and concurrency without significantly sacrificing consistency.

· 본 연구는 첨단정보기술연구센터를 통하여 한국과학재단의 지원을 받았음.
† 비 회 원 : 한국과학기술원 전산학과
hsseo@mozart.kaist.ac.kr
jwchang@mozart.kaist.ac.kr
ysmoon@mozart.kaist.ac.kr
†† 종신회원 : 한국과학기술원 전산학과 교수
kywhang@cs.kaist.ac.kr
††† 종신회원 : 서울시립대학교 전산통계학과 교수
ekhong@venus.uos.ac.kr
논문접수 : 2000년 7월 10일
심사완료 : 2001년 5월 18일

1. 서 론

데이타베이스 관리 시스템(database management system: DBMS)에서 동시성 제어(concurrency control)란 여러 트랜잭션의 동시 수행에 의하여 데이타의 일관성(consistency)이 파괴되지 않도록 트랜잭션간의 상호작용을 제어하는 작업이다[1]. 동시성 제어 방법 중에서 가장 널리 사용되고 있는 로킹(locking) 방법에서는 각

트랜잭션이 액세스하고자 하는 데이타에 대하여 미리 로크를 획득함으로써 일관성을 파괴할 수 있는 다른 트랜잭션의 충돌(conflict) 연산이 동시에 수행되는 것을 방지한다[2]. 로킹을 사용하는 대부분의 상용 DBMS는 트랜잭션이 획득한 모든 로크를 트랜잭션의 종료 시점에 반환하는 엄정 이단계 로킹(strict two-phase locking)을 제공한다[1].

모든 트랜잭션이 엄정 이단계 로킹을 따른다면 수행 결과로 발생한 스케쥴의 직렬성(serializability)이 보장되며, 따라서 데이타베이스 일관성이 유지된다[1]. 반면, 엄정 이단계 로킹에서는 트랜잭션이 로크를 유지하는 기간이 길기 때문에 수행시간이 긴 트랜잭션이 많은 경우에 동시성이 심각하게 저하되는 문제점을 가진다. 엄정 이단계 로킹의 동시성 저하 문제를 해결하기 위하여 상용 DBMS에서는 엄정 이단계 로킹 뿐만 아니라 이보다 완화된 여러 가지 다른 로킹 규약을 함께 제공하는데[2,3,4,5], 엄정 이단계 로킹과 이들 완화된 로킹 규약들을 고립화 수준(isolation levels)이라 한다[6].

기존의 고립화 수준으로는 0, 1, 2, 3[6], 그리고 커서 안정성(cursor stability)[7] 등이 있다. 수준 0, 1, 2, 3에서, 수준 3은 엄정 이단계 로킹을 의미하며, 낮은 수준일수록 로크를 획득하지 않거나 획득한 로크를 더 빨리 반환하는 방법으로 로킹 규약을 완화한다. 따라서, 낮은 고립화 수준일수록 동시성은 높아지는 반면, 일관성이 파괴될 가능성이 높아진다. 그리고, 커서 안정성은 수준 3보다 완화되고 수준 2보다는 덜 완화된 로킹 규약으로 수준 2와 3 사이의 고립화 수준이다.

커서 안정성은 커서를 사용하는 RDBMS의 응용에서 높은 일관성을 보장하면서 동시성을 높이는데 매우 유용하게 사용된다[2,8]. 커서 안정성은 커서가 레코드를 가리키는 동안 해당 레코드에 대한 로크를 유지하는 로킹 방법으로, 커서를 사용한 레코드 갱신에서 갱신 분실(lost update)[7]을 방지한다. 갱신 분실은 트랜잭션의 효과를 사라지게 하는 이상 현상(anomaly)으로 일관성을 파괴하는 주요 원인의 하나이다. RDBMS의 응용은 주로 커서를 사용하여 레코드들을 액세스하므로, 커서 안정성은 이러한 응용에 대하여 동시성을 높이면서도 갱신 분실을 방지하는 유용한 수준이다. 다시 말하면, 커서 안정성은 RDBMS에서 커서의 특성을 사용하여 수준 2보다는 일관성을 높이고 수준 3보다는 동시성을 향상시킨 방법이다.

최근 대부분의 RDBMS는 객체지향(object-oriented) 개념을 추가하여 확장한 객체 관계형 DBMS(object relational DBMS: ORDBMS)로 발전하고 있다[3,4,5,8]. ORDBMS는 RDBMS와 비교하여 데이타베이스의 구조와 데이타베이스를 액세스하는 방법에서 큰 차이점이 있다. 첫째, RDBMS에서는 데이타베이스가 레코드들의 집합으로 이루어지는 반면, ORDBMS에서는 참조관계(reference relationship)로 연결된 객체들의 집합인 복합객체(complex objects)[9]들의 집합으로 이루어진다. 둘째, RDBMS의 응용이 주로 커서를 사용하여 레코드를 액세스하는 반면, ORDBMS의 응용은 참조관계를 사용하여 복합객체를 탐색항해(navigation)한다. 복합객체를 탐색항해하는 ORDBMS의 응용을 탐색항해 응용(navigational application)이라 부른다.

ORDBMS에서 커서 안정성은 더 이상 유용한 고립화 수준이 되지 못한다. 왜냐하면, 탐색항해 응용을 커서 안정성에서 수행할 때 다음과 같은 세 가지 일관성 문제가 발생하기 때문이다. 첫째, 복합객체를 갱신할 때 갱신 분실이 발생할 수 있다. 둘째, 허상 포인터(dangling pointer)를 만나서 프로그램 오류가 발생할 수 있다. 탐색항해 응용은 주로 포인터 기반의 연산을 수행하므로, 허상 포인터는 심각한 문제가 된다. 셋째, 다른 트랜잭션의 갱신을 부분만 반영한 일관성을 잃은 복합객체의 상태를 읽을 수 있다. 일관성을 잃은 복합객체의 상태를 기반으로 복합객체를 갱신하면 데이타베이스의 일관성이 파괴될 수 있으므로 심각한 문제가 된다. 현재 탐색항해를 지원하는 상용 ORDBMS들[4,5]은 RDBMS에서와 동일한 방법으로 커서 안정성을 제공하며, 사용자가 직접 로킹을 제어하여 커서 안정성의 일관성 문제를 해결하도록 요구한다. 따라서, ORDBMS의 탐색항해 응용을 위해 커서 안정성을 대체할 수 있는 새로운 고립화 수준이 요구된다. 이러한 수준이 없는 상용 ORDBMS에서는 탐색항해 응용을 수준 3에서 수행해야 안전하나, 수준 3의 동시성 저하를 감수해야 한다.

본 논문에서는 새로운 고립화 수준인 **탐색항해 안정성**(*navigation stability*)을 제안한다. 첫째, 본 논문에서는 탐색항해 응용의 액세스 패턴을 분석하고 이를 바탕으로 탐색항해 모델을 정의한다. 그리고 이 모델에 근거하여 커서 안정성의 로킹 규약을 확장한 탐색항해 안정성을 정의한다. 둘째, 탐색항해 안정성이 탐색항해 모델에 속하는 응용에 대하여 커서 안정의 세 가지 문제를 해결함을 보인다.

논문의 구성은 다음과 같다. 제 2절에서는 관련연구로 기존의 고립화 수준들을 소개하고, 제 3절에서는 탐색항해 응용의 특징과 탐색항해 응용에 대한 커서 안정성의 일관성 문제를 소개한다. 제 4절에서는 새로운 고립화 수준인 탐색항해 안정성을 제안한다. 제 5절에서는 성능

평가 결과를 설명하고 이를 분석한다. 마지막으로 제 6 절에서 본 논문의 결론을 맺는다.

2. 관련 연구

우선 본 논문에서 사용하는 용어를 소개한다. 데이타베이스는 레코드 또는 객체와 같은 데이타 아이템들의 집합으로 이루어진다. 앞으로, 데이타 아이템을 RDBMS의 경우 레코드라고 말하고 x, y, z, ... 등으로 표기하며, ORDBMS의 경우 객체라고 말하고 o_1, o_2, ... 등으로 표기한다. 트랜잭션은 프로그램이 수행된 결과로 발생한 데이타 아이템을 액세스한 연산들의 시퀀스를 말하며, T_1, T_2, ... 등으로 표기한다. 트랜잭션에 포함된 연산들은 읽기 연산과 쓰기 연산으로 구성된다. 트랜잭션은 완료(commit) 연산 또는 철회(abort) 연산에 의해 종료한다. 스케줄은 여러 트랜잭션이 동시에 수행된 결과로 발생한 연산들의 시퀀스를 말하며, S_1, S_2, ... 등으로 표기한다.

트랜잭션 T_i의 객체 o_j에 대한 읽기 연산은 '$r_i[o_j]$'라 표기하고, o_j에 대한 쓰기 연산은 '$w_i[o_j]$'라 표기한다. 커서를 사용한 연산임을 구분하고자 할 때 커서를 사용한 읽기 연산을 'r' 대신 'rc'라고 표기하고, 커서를 사용한 쓰기 연산을 'w'대신 'wc'라고 표기한다. 트랜잭션 T_i의 완료 연산은 'c_i', 철회 연산은 'a_i'로 표기한다.

고립화 수준에서는 읽기 연산을 위해서는 공유 로크(shared lock)가 사용되고, 쓰기 연산을 위해서는 독점로크(exclusive lock)가 사용된다. 서로 다른 트랜잭션에 의해 요청된 동일한 데이타 항목에 대한 두 로크가 모두 공유 로크일 때 두 로크는 동시에 획득될 수 있으며, 두 로크 중 적어도 하나가 독점 로크이면 동시에 획득될 수 없다. 트랜잭션에 의해 획득된 로크의 로크 기간에는 트랜잭션의 완료 시점까지 로크가 유지되는 긴 기간(long duration)과 연산이 수행되는 동안만 유지되는 짧은 기간(short duration)이 사용된다. 긴 기간의 공유 로크를 간단히 긴 공유 로크라고 부른다. 같은 방법으로 정의된 짧은 공유 로크, 긴 독점로크, 그리고 짧은 독점 로크 등의 용어를 사용한다. 서로 다른 트랜잭션의 두 연산 중 적어도 하나가 쓰기 연산일 때 두 연산은 충돌한다(conflict)고 말한다.

고립화 수준 0, 1, 2, 3은 로크 모드와 로크 기간을 사용한 로킹 규약에 의해 표 1과 같이 정의된다[1]. 표 1의 수준들은, 높은 수준일수록 일관성이 파괴될 가능성이 낮으며, 수준이 낮을수록 동시성이 높아진다. 수준 3은 모든 연산에 대하여 긴 로크를 획득하는 엄정 이단계 로킹을 의미한다. 수준 2는 쓰기 연산에 대해서 수준 3과 같이 긴 독점 로크를 사용하나, 읽기 연산에 대해서는 짧은 공유 로크를 사용한다. 수준 2에서 트랜잭션은 데이타 항목을 읽을 때 짧은 공유 로크를 획득하기 때문에, 그 데이타 항목을 읽은 후에 다른 트랜잭션이 그 데이타 항목을 갱신할 수 있다. 따라서, 수준 2는 스케줄 $S_{lostupdate}$: $r_1[x]$ $r_2[x]$ ... $w_2[x]$ c_2 ... $w_1[x]$ c_1과 같은 갱신 분실의 발생을 허용한다.

수준 2보다 낮은 고립화 수준인 0과 1에서는 일관성을 잃은 데이타를 읽거나 생성할 수 있으므로, 매우 제한적인 응용에서만 사용된다. 이러한 이유로 인하여 실제로 대부분의 트랜잭션은 수준 2 이상에서 수행된다[2].

표 1 로킹 규약에 의한 고립화 수준 0, 1, 2, 3의 정의

수준	읽기 연산	쓰기 연산
3	긴 공유 로크	긴 독점 로크
2	짧은 공유 로크	긴 독점 로크
1	로크를 획득하지 않음	긴 독점 로크
0	로크를 획득하지 않음	짧은 독점 로크

커서 안정성[7]은 RDBMS에서 수준 2의 변형이다. 커서는 레코드의 집합을 순회하기 위해 사용되는 포인터의 일종이다[10]. 커서는 RDBMS에서 SQL 질의[11]를 만족하는 레코드의 집합을 순회하기 위해 주로 사용된다. 커서 안정성은 커서를 사용하지 않는 경우에는 수준 2와 동일하지만, 커서를 사용한 읽기 연산이 획득한 공유 로크의 로크 기간이 수준 2보다 더 긴 차이점이 있다. 커서를 사용하여 레코드를 읽으면 커서는 다음 레코드를 읽기 전까지 먼저 읽었던 레코드를 가리키는데, 커서 안정성은 커서가 레코드를 가리키는 동안 공유 로크를 유지한다[7]. 이러한 공유 로크의 로크 기간을 짧은 로크의 로크 기간과 구분하기 위하여 중간 기간(medium duration)이라고 한다[2]. 중간 공유 로크는 레코드가 갱신되지 않은 상태에서 커서가 다른 레코드로 이동할 때 반환된다. 이러한 커서 안정성은 수준 2에서 발생할 수 있는 갱신 분실을 막는데 효과적이다. 다음의 예는 수준 2에서 발생하지만, 커서 안정성에서는 방지되는 갱신 분실의 스케줄이다.

---

1) 최근 ANSI에서 수준 3을 SQL 질의를 사용할 때 발생할 수 있는 유령현상(phantom phenomenon)을 허용하는 수준인 'REPEATABLE READ'와 허용하지 않는 수준인 'SERIALIZABLE'로 나누어 정의하였다[7]. 그러나, 본 논문에서는 탐색항해를 기반으로 데이타를 액세스하는 응용을 다루기 때문에 유령현상을 다루지 않는다.

$S_{cs_lostupdate}$: $rc_1[x] r_2[x] \ldots w_2[x]\ c_2 \ldots wc_1[x]\ c_1$

트랜잭션 T_1과 T_2가 수준 2에서 수행되면 트랜잭션 T_2의 $w_2[x]$에 의한 쓰기 연산의 결과는 T_1의 $w_1[x]$에 의해 분실된다. 이는 수준 2의 읽기 연산이 짧은 공유 로크를 사용하므로 T_1이 $r_1[x]$ 이후에 즉시 T_2가 x를 갱신할 수 있기 때문이다. 반면에, 트랜잭션 T_1과 T_2가 커서 안정성에서 수행되면 $wc_1[x]$가 수행될 때까지 커서가 가리키는 x에 대한 공유 로크가 유지되기 때문에, $wc_1[x]$ 이전에 $w_2[x]$가 수행되는 것을 방지하므로 스케줄 $S_{cs_lostupdate}$은 발생하지 않는다.

커서 안정성은 RDBMS에서 매우 유용한 고립화 수준이다. RDBMS에서 응용 프로그램은 주로 커서를 사용하여 레코드를 액세스한다. 커서 안정성은 이들 응용에 대하여 갱신 분실을 방지하며, 커서는 짧은 기간이 지난 후 다른 레코드로 이동하므로 수준 3에 비해 높은 동시성을 제공하기 때문이다.

현재 탐색항해를 지원하는 상용 ORDBMS[4,5]는 기본적으로 RDBMS와 동일한 방법으로 커서 안정성을 제공하고 있다. 다만, RDBMS에서와는 달리 이들 ORDBMS에서는 객체에 대한 로킹을 제어하는 로킹 함수를 사용자에게 제공하여 사용자가 직접 로킹을 제어하도록 한다. 따라서, 상용 ORDBMS에서는 커서 안정성에서 탐색항해 응용이 수행될 때 발생할 수 있는 일관성 문제는 사용자가 직접 해결해야 한다.

3. 탐색항해 응용에서 커서 안정성의 문제점

본 절에서는 먼저 ORDBMS의 탐색항해 응용의 특징을 소개하고, 탐색항해 응용을 커서 안정성에서 수행할 경우 발생하는 일관성 문제를 설명한다.

3.1 복합객체와 탐색항해 인터페이스

ORDBMS의 여러 가지 특징들 중에서 복합객체와 탐색항해 인터페이스를 설명한다. 이들 두 가지 특징은 ORDBMS의 탐색항해 응용의 데이타 액세스 패턴을 결정하는 중요한 요소이다.

객체는 속성들의 값으로 구성되는데, 속성에는 객체 식별자(OID)(또는 참조)를 속성 값으로 가지는 참조타입 속성이 있다. 객체 o_i가 객체 o_j에 대한 참조를 속성 값으로 가질 때, o_i는 o_j를 **참조한다**고 말하며, 이러한 두 객체의 참조관계를 $o_i \rightarrow o_j$로 표기한다. o_i가 o_j와 **참조관계로 연결되어 있다**는 것은 $o_1, \ldots, o_k$의 객체들이 존재하여 $o_i \rightarrow o_1,\ o_1 \rightarrow o_2, \ldots, o_{k-1} \rightarrow o_k,\ o_k \rightarrow o_j$인 참조관계들을 가지는 것을 말한다. 복합객체는 **루트**(*root*)라고 부르는 한 객체로부터 참조관계로 연결된 객체들의 집합을 말한다[8]. 복합객체에 포함된 객체를 **컴포넌트**(*component*) **객체**라고 말한다.

최근의 여러 ORDBMS는 복합객체를 효율적으로 액세스하기 위해서 탐색항해 액세스를 지원한다[4,5]. 탐색항해 액세스는 하나의 컴포넌트 객체를 액세스하고 참조관계를 따라가면서 다음 객체를 액세스하는 복합객체의 액세스 방법을 말한다.

탐색항해 액세스를 지원하기 위해 ORDBMS들이 제공하는 프로그램 인터페이스를 탐색항해 인터페이스라고 한다[4,5]. 탐색항해 인터페이스는 객체를 조작하는 함수들을 포함한다. 예를 들면, 객체를 읽는 함수와 갱신하는 함수가 있다. 이들 연산은 액세스하고자 하는 객체에 대한 참조를 포함하는 입력 변수를 가진다.

3.2 탐색항해 응용에서의 커서 안정성의 일관성 문제

탐색항해를 지원하는 상용 ORDBMS들[4,5]은 커서 안정성을 제공한다. 이들 ORDBMS들이 제공하는 커서 안정성은 RDBMS의 로킹 방법을 그대로 탐색항해 인터페이스에도 적용한다. 즉, ORDBMS에서 제공하는 커서 안정성에서는 탐색항해 인터페이스를 사용한 읽기 연산에 대하여 짧은 공유 로크를 획득한다.

탐색항해 응용에서는 커서를 이용하여 원하는 복합객체의 루트 객체를 액세스할 수는 있지만, 해당 복합객체의 루트 객체가 아닌 컴포넌트 객체를 액세스하기 위해서는 탐색항해 인터페이스를 사용해야 한다. 이로 인하여, 탐색항해 응용을 커서 안정성에서 수행하면 허상 포인터 문제, 갱신 분실 문제, 그리고 일관성을 잃은 복합객체를 읽는 문제의 세 가지 문제점이 발생한다. 본 절에서는 이들 세 가지 문제를 자세히 설명한다. 설명의 편의를 위해서 다음을 가정한다. 데이타베이스는 컴포넌트 객체들 o_1, o_2, o_3로 구성된 복합객체를 포함한다. 루트 객체는 객체 o_1이고, 두 참조관계 $o_1 \rightarrow o_2$와 $o_2 \rightarrow o_3$가 포함되어 있다. 또한, ORDBMS는 컴포넌트 객체를 단위로 갱신을 데이타베이스에 반영한다고 가정한다. 즉, 객체의 일부 속성 값이 갱신되더라도 객체의 모든 속성 값을 데이타베이스에 새로 저장한다. 실제로, ODMG 모델을 따르는 ORDBMS는 이러한 방법으로 객체의 갱신을 데이타베이스에 반영한다[12].

허상 포인터 문제

ORDBMS는 데이타베이스의 참조 무결성(referential integrity)다른 객체에 의해 참조되고 있는 객체는 데이타베이스에 반드시 존재해야 하는 성질[13]을 유지해야 한다. 삭제된 객체에 대한 참조를 허상 포인터라고 말하며[14], 이러한 허상 포인터의 발생은 참조 무결성이 파괴되는 것을 의미한다. 탐색항해 응용은 허상 포인터를 따라서 복합객체를 항해하면 비정상적으로 종료하므로,

허상 포인터는 심각한 문제가 된다.

일반적으로, 모든 트랜잭션은 옳은(correct) 트랜잭션이라고 가정한다. 옳은 트랜잭션은 혼자 수행할 때 참조 무결성을 보장한다. 이를 위해서, 트랜잭션이 객체를 생성, 삭제, 또는 수정하면, 그 객체와 그 객체를 참조하는 객체들이 허상 포인터를 포함하지 않아야 한다. 따라서, 객체를 삭제하는 트랜잭션은 더 이상 삭제된 객체를 참조하지 않도록 삭제된 객체를 참조하고 있는 객체들을 갱신하는 연산을 포함한다. 그리고, 객체를 생성 또는 수정하는 트랜잭션은 그 객체가 참조하는 객체들이 데이타베이스에 존재하는 것을 확인하기 위해서 이들 객체들을 읽는 연산을 포함한다. 본 논문에서는 생성 또는 수정하는 연산과 읽기 연산은 제 4절에서 정의할 동일한 단위 탐색항해에 속한다고 가정한다.

비록 모든 트랜잭션은 혼자 수행할 때 참조 무결성을 유지하지만, 커서 안정성에서 다수의 트랜잭션이 동시에 수행하면 더 이상 참조 무결성은 보장되지 않는다. 커서 안정성에서는 객체에 대한 참조가 허상 포인터가 되거나, 허상 포인터가 데이타베이스에 완료될 수 있다. 트랜잭션이 허상 포인터를 통해 삭제된 객체를 읽으려고 시도하는 것을 트랜잭션이 **허상 포인터를 만난다**고 정의한다.

다음은 커서 안정성에서 트랜잭션이 허상 포인터를 만나는 스케줄의 예이다.

$S_{dp_encounter}$: $rc_1[o_1]r_1[o_2]$ ... $w_2[o_3]w_2[o_2]c_2r_1[o_3]a_1$

T_1은 참조관계 $o_1 \rightarrow o_2$와 $o_2 \rightarrow o_3$를 따라가면서 o_1으로부터 o_3까지 항해하는 트랜잭션이며, 다음에 따라갈 참조관계가 'NULL'이면 항해를 종료한다. T_2는 o_3를 삭제하고 o_2가 포함하고 있는 o_3에 대한 참조가 'NULL'이 되도록 o_2를 갱신한다. 두 트랜잭션 T_1과 T_2이 각각 혼자 수행하면 허상 포인터를 만나지 않는다. 커서 안정성에서 T_1은 o_2를 읽고 o_2에 대한 공유 로크를 즉시 반환한다. 그런데, T_2가 완료하면 T_1이 읽은 o_2에 포함된 o_3에 대한 참조는 허상 포인터가 된다. 그러므로, T_1은 o_3를 읽으려고 할 때 허상 포인터를 만나며 프로그램 오류로 인하여 비정상적으로 종료한다.

다음은 허상 포인터가 데이타베이스에 완료되는 스케줄의 예이다.

S_{dp_create}:$rc_1[o_1]r_1[o_2]r_1[o_3]$ ... $w_2[o_3]w_2[o_2]c_2$ ... $w_1[o_2]c_1$

트랜잭션 T_1은 o_1으로부터 항해하여 o_2를 읽고 그 값을 기반으로 o_2를 갱신한다. 참조 무결성을 위한 가정에 의해 T_1은 o_2를 갱신하기 전에 o_3를 먼저 읽는다. T_2는 앞의 스케줄 $S_{dp_encounter}$의 T_2와 동일한 트랜잭션이다. 이때 T_2가 o_3를 이미 삭제했음에도 불구하고, T_1은 T_2가 o_2를 갱신하기 전에 읽은 o_2를 기반으로 o_2를 갱신한다. T_1이 읽은 o_2는 o_3에 대한 참조를 포함하고 있고, T_1의 갱신에 의해 다시 데이타베이스에 저장된다. 따라서, 허상 포인터인 o_2에 포함된 o_3에 대한 참조가 데이타베이스에 완료된다.

갱신 분실 문제

커서 안정성에서는 트랜잭션이 탐색항해 인터페이스를 사용하여 컴포넌트 객체를 읽을 때 짧은 공유 로크를 사용하기 때문에, 컴포넌트 객체를 갱신할 때 갱신 분실이 발생할 수 있다. 커서 안정성에서 갱신 분실이 발생하는 스케줄의 예는 다음과 같다.

$S_{cs_lostupdate}$: $rc_1[o_1]r_1[o_2]$ ... $w_2[o_2]c_2$... $w_1[o_2]c_1$

트랜잭션 T_1은 o_1으로부터 항해하여 o_2를 읽고 갱신한다. 트랜잭션 T_2는 o_2를 갱신한다. T_1은 o_2를 읽고 즉시 o_2에 대한 공유 로크를 반환하기 때문에 T_2는 T_1이 o_2를 읽고 갱신하는 사이에 o_2를 갱신하고 완료한다. 이러한 순서로 수행하면, T_2의 갱신은 나중에 수행된 T_1의 갱신에 의해 분실된다. 그러므로, 커서 안정성에서 탐색항해 인터페이스를 사용하여 객체를 갱신하면 갱신 분실이 발생한다.

일관성을 잃은 복합객체를 읽는 문제

복합객체는 참조관계로 연결된 하나 이상의 객체로 이루어진다. 참조관계에는 흔히 무결성 제약 조건(integrity constraint)이 부여된다.

수학적으로, 트랜잭션 T_1이 일관성을 잃은 복합객체를 읽는 것은 복합객체에 속한 임의의 두 객체 o_1과 o_2를 읽을 때 다른 트랜잭션 T_2가 다음과 같은 순서로 수행 것을 말한다.

$S_{readskew}$: $r_1[o_1]$ ... $w_2[o_1]$ ... $w_2[o_2]$ ... $c_2r_1[o_2]$ ...

트랜잭션 T_1은 o_1과 o_2를 읽고, 트랜잭션 T_2는 o_2와 o_3를 갱신한다. 스케줄 $S_{readskew}$에서 T_1은 T_2가 갱신하기 전의 o_1을 읽은 반면, T_2가 갱신한 후의 o_2를 읽는다. 즉, T_1은 T_2의 갱신 결과를 일부분은 읽고 일부분은 읽지 못한다. 이러한 이상 현상을 읽기 스큐(read skew)라고 부른다[7].

커서 안정성에서 탐색항해 응용은 다음의 스케줄에서와 같이 일관성을 잃은 복합객체를 읽는다.

$S_{cs_readskew}$: $rc_1[o_1]r_1[o_2]$ ... $w_2[o_2]w_2[o_3]c_2r_1[o_3]$...

T_1은 참조관계 $o_1 \rightarrow o_2$와 $o_2 \rightarrow o_3$ 를 따라가면서 o_1으로부터 o_3까지 항해하는 트랜잭션이다. 그리고 T_2는 o_2와 o_3를 갱신한다. 커서 안정성에서 T_1은 o_2에 대한 공유 로크를 즉시 반환하므로, T_1이 o_2을 읽고 o_3를 읽기 전에 T_2가 o_2와 o_3를 갱신하는 것이 가능하다. 따라서, 스케줄 $S_{cs_readskew}$은 읽기 스큐를 포함하여 T_1은 일

관성을 잃은 복합객체를 읽는다.

3.3 기존 ORDBMS의 해결 방법과 문제점

탐색항해를 제공하는 상용 ORDBMS[4,5]에서는 탐색항해 응용에 대한 커서 안정성의 일관성 문제를 프로그래머에 의존하여 해결한다. 즉, 프로그래머는 커서 안정성의 문제를 피하기 위해서 ORDBMS에서 제공된 로킹 함수를 사용하여 직접 로킹을 제어해야 한다[4,5]. 이러한 방법은 프로그램을 복잡하게 하고 신뢰성을 떨어뜨린다. 특히, 동시성 제어상의 오류는 프로그램이 혼자 수행될 때는 정상적으로 동작하기 때문에 그 원인을 찾기가 힘들다. 따라서, 커서 안정성에서 발생하는 탐색항해 응용에 대한 일관성 문제를 피하기 위해 ORDBMS가 필요한 로킹을 제공하는 새로운 고립화 수준이 요구된다.

본 논문의 가정과는 달리, RDBMS로부터 확장된 상용 ORDBMS[4,5]에서는 갱신을 속성 단위로 데이타베이스에 반영한다. 이 경우에 커서 안정성에서 허상 포인터 문제는 부분적으로 방지된다. 예를 들어, 갱신을 속성 단위로 반영하면 제 3.2절에서 제시한 스케줄 S_{dp_create}는 발생하지 않는다. 왜냐하면, 트랜잭션 T_1과 T_2가 객체 o_2의 서로 다른 속성을 갱신하므로 T_1이 객체 o_2의 갱신을 완료하더라도 T_2가 o_2를 갱신한 결과는 분실되지 않기 때문이다. 그러나, 갱신을 속성 단위로 반영하더라도 스케줄 $S_{dp_encounter}$는 발생할 수 있다. 왜냐하면, 갱신을 반영하는 단위와 무관하게 T_1은 T_2가 o_2를 갱신하기 전에 읽은 o_2에 포함된 o_3의 참조를 통해 o_3를 읽으려고 하기 때문이다.

4. 탐색항해 모델과 탐색항해 안정성

본 절에서는 커서 안정성의 세 가지 일관성 문제를 해결하는 새로운 고립화 수준인 탐색항해 안정성을 제안한다. 제 4.1절에서는 탐색항해 응용의 복합객체에 대한 액세스 패턴인 탐색항해 모델을 정의한다. 제 4.2절에서는 탐색항해 안정성을 제안하고, 이 고립화 수준이 탐색항해 모델하에서 커서 안정성의 세 가지 일관성 문제를 해결함을 보인다. 제 4.3절에서는 탐색항해 안정성을 ORDBMS에 구현하기 위해 요구되는 이슈와 이의 해결책을 제시한다.

4.1 탐색항해 모델

탐색항해 응용은 일반적으로 복합객체의 루트 객체를 액세스한 후, 탐색항해 인터페이스를 사용하여 그 루트 객체로부터 참조관계들을 따라 가면서 컴포넌트 객체들을 액세스한다[11]. 본 논문에서는 커서를 사용하여 루트 객체를 액세스하는 방법을 다룬다. 실제로, UniSQL의 OML(object manipulation language) 인터페이스[5], Oracle의 SQL 인터페이스[4], 그리고 ODMG의 이름이 부여된 객체(named object)[11]를 사용하여 루트 객체를 액세스하는 방법은 커서를 사용하는 방법으로 간주할 수 있다.

대부분의 탐색항해 응용은 커서가 루트 객체로부터 다른 루트 객체로 이동하면, 커서가 이전 루트 객체를 가리키면서 항해하여 읽은 컴포넌트 객체들을 다음 항해에서 액세스하지 않는다. 이러한 액세스 패턴에 기반하여 본 논문에서는 복합객체 액세스 순서와 복합객체를 갱신하는 방법을 다음과 같이 가정한다

가정 1. (복합객체 액세스 순서) 탐색항해 응용은 먼저 커서를 사용하여 액세스 하고자 하는 복합객체의 루트 객체를 액세스한다. 그리고, 커서가 루트 객체를 가리키는 동안 탐색항해 인터페이스를 사용하여 루트 객체로부터 참조관계로 연결된 컴포넌트 객체들을 액세스한다.

가정 2. (복합객체 갱신 방법) 탐색항해 응용은 커서가 루트 객체를 가리키는 동안만 그 복합객체를 갱신하며, 갱신하는 새로운 값은 현재 커서가 가리키고 있는 루트 객체로부터의 항해에서 읽은 컴포넌트 객체들을 기반으로 결정한다.

본 논문에서는 탐색항해 응용의 수행 결과로 발생한 트랜잭션을 **탐색항해 트랜잭션**(*navigational transaction*)이라 정의한다. 탐색항해 트랜잭션의 연산은 커서를 사용한 읽기 연산, 탐색항해 인터페이스를 사용한 읽기 연산과 쓰기 연산으로 이루어진다. 탐색항해 트랜잭션의 각 연산은 그 연산을 호출한 항해가 시작된 루트 객체와 그 루트 객체를 가리키는 커서의 쌍을 가진다. 가정 1에 의해 각 연산에 대하여 이러한 쌍은 유일하게 존재한다. 연산에 대한 이러한 커서와 루트 객체의 쌍을 **탐색항해 시작점**(*navigational start point*)이라고 정의한다. 탐색항해 트랜잭션의 연산들은 연산의 탐색항해 시작점이 같은 동치관계(equivalence relation)에 의해 분할(partition)된다. 이때 분할의 각 원소를 **단위 탐색항해**(*unit navigation*)라고 정의한다. 단위 탐색항해에 속한 임의의 연산에 대한 탐색항해 시작점의 커서를 **탐색항해 커서**(*navigational cursor*)라고 정의한다. 그리고, 탐색항해 시작점의 루트 객체를 **탐색항해 루트**(*navigational root*)라고 정의한다.

4.2 탐색항해 안정성

정의 1. 탐색항해 트랜잭션에 대하여 다음과 같은 두 가지 규약을 따르는 로킹 규약을 **탐색항해 안정성**(*navigational stability*)이라 정의한다.

[규약 1:] 트랜잭션은 탐색항해 인터페이스를 사용하는 읽기 연산을 제외한 모든 연산에 대해 커서 안정성과 동일한 로킹 규약을 준수한다.

[규약 2:] 트랜잭션의 각 단위 탐색항해에서 획득한 공유 로크는 탐색항해 커서가 탐색항해 루트를 가리키는 동안 반환하지 않는다.

규약 1은 탐색항해 루트에 대해 커서 안정성의 일관성을 제공하며, 규약 2는 단위 탐색항해에서 액세스 된 컴포넌트 객체들에 대하여 커서 안정성의 일관성을 제공한다. 즉, 단위 탐색항해에서 액세스 된 객체는 그 단위 탐색항해가 수행되는 동안 다른 트랜잭션에 의해 갱신되지 않는다. 결국, 단위 탐색항해는 이단계 로킹 기법에 따라 로크를 획득하며 탐색항해 커서가 다른 객체로 이동하는 시점에 단위 탐색항해에서 획득된 공유 로크들을 반환한다.

이제, 탐색항해 안정성이 커서 안정성의 세 가지 일관성 문제를 해결함을 보인다. 첫째, 정리 1에서 탐색항해 안정성은 갱신 분실을 방지함을 보인다. 둘째, 정리 2에서 탐색항해 안정성은 각 단위 탐색항해가 일관성을 잃은 복합객체를 읽는 문제를 방지함을 보인다. 셋째, 정리 3에서 탐색항해 안정성은 허상 포인터 문제를 해결함을 보인다.

정리 1. 모든 탐색항해 트랜잭션이 탐색항해 안정성에서 수행하면 갱신 분실이 발생하지 않는다.

증명: 탐색항해 모델의 가정 2에 의해 트랜잭션은 현재 수행되고 있는 단위 탐색항해 내에서 읽은 객체를 기반으로 객체를 갱신한다. 탐색항해 안정성의 규약 2에 의해 현재 수행중인 단위 탐색항해가 읽은 객체에 대한 공유 로크는 반환하지 않는다. 그러므로, 객체를 읽고 갱신하는 사이에 다른 트랜잭션이 그 객체를 갱신할 수 없다. 따라서, 갱신 분실은 발생하지 않는다. □

예를 들어, 제 3절의 커서 안정성에서 갱신 분실이 발생한 스케줄 $S_{cs_lostupdate}$의 경우, 만약 두 트랜잭션 T_1과 T_2가 탐색항해 안정성에서 수행되었다면 $S_{cs_lostupdate}$은 발생할 수 없다. 그 이유는 다음과 같다. T_1이 객체 o_2를 읽고 갱신하는 연산은 모두 동일한 단위 탐색항해에 포함되므로, o_2에 대한 공유 로크가 o_2를 갱신하는 시점까지 유지된다. 따라서, T_2는 T_1이 완료할 때까지 o_2를 갱신할 수 없다.

정리 2. 모든 탐색항해 트랜잭션이 탐색항해 안정성에서 수행되면 트랜잭션의 각 단위 탐색항해는 일관성을 잃은 복합객체를 읽지 않는다.

증명: 각 단위 탐색항해가 복합객체를 읽을 때 읽기 스큐가 발생할 수 없음을 증명하면 충분하다. 단위 탐색항해가 한 복합객체에 포함된 객체 o_1를 읽은 후 o_2를 읽는다고 가정하자. 두 읽기 연산은 동일한 단위 탐색항해에 속하므로 o_1에 대한 공유 로크는 o_2를 읽을 때까지 유지된다. 따라서, 트랜잭션의 o_1과 o_2에 대한 읽기 연산들 사이에서 다른 트랜잭션이 o_1과 o_2를 모두 갱신하는 것은 불가능하다. 그러므로 단위 탐색항해에서 읽기 스큐는 발생하지 않는다.

예를 들어, 제 3절의 커서 안정성에서 일관성을 잃은 복합객체를 읽는 스케줄 $S_{cs_readskew}$의 경우, 만약 두 트랜잭션 T_1과 T_2가 탐색항해 안정성에서 수행되었다면 $S_{cs_readskew}$은 발생할 수 없다. 그 이유는 다음과 같다. T_1이 객체 o_2를 읽고 o_3를 읽는 연산은 동일한 단위 탐색항해에 포함되므로, o_2에 대한 공유 로크가 o_3를 읽는 시점까지 유지된다. 따라서, T_2는 T_1이 o_3를 읽을 때까지 o_2를 갱신할 수 없다.

정리 3. 데이타베이스는 허상 포인터를 포함하지 않는다고 가정한다. 모든 탐색항해 트랜잭션이 탐색항해 안정성에서 수행되면, 어느 트랜잭션도 허상 포인터를 만나지 않으며 또한 허상 포인터가 데이타베이스에 완료되지도 않는다.

증명: 부록 참조. □

예를 들어, 제 3절의 커서 안정성에서 허상 포인터를 만나는 두 스케줄 S_{dp_create}와 $S_{dp_encounter}$의 경우, 두 트랜잭션 T_1과 T_2가 탐색항해 안정성에서 수행되었다면 S_{dp_create}와 $S_{dp_encounter}$은 발생할 수 없다. 이유는 다음과 같다. 첫째, S_{dp_create}에서는 T_1이 o_2를 읽은 후 o_2를 갱신하기 전에 T_2가 o_2를 갱신하기 때문에 허상 포인터가 발생한다. 그러나, T_1과 T_2가 탐색항해 안정성에서 수행되었다면, T_1가 o_2에 대한 공유 로크를 o_2를 갱신할 때까지 유지하기 때문에 T_2는 T_1가 완료할 때까지 o_2를 갱신할 수 없다. 그러므로 허상 포인터 $o_2 \rightarrow o_3$는 데이타베이스에 완료될 수 없다. 둘째, $S_{dp_encounter}$에서 T_1과 T_2가 탐색항해 안정성에서 수행되었다면, T_1의 o_1을 가리키는 커서가 다른 루트 객체로 이동하기 전까지 T_2는 o_2를 갱신할 수 없기 때문에 참조관계 o_2 o_3는 허상 포인터가 될 수 없다. 그러므로 T_1은 허상 포인터를 만나지 않는다.

탐색항해 모델의 가정 1과 2를 따르지 않는 탐색항해 응용에 대하여 정리 1 ~ 3은 더 이상 만족하지 않는다. 가정 1과 2를 만족하지 않는 응용은 탐색항해 커서가 다른 루트 객체로 이동한 후에도 이미 얻은 객체 참조를 사용하여 계속 탐색항해를 하는 것을 의미한다. 탐색항해 안정성에서 이러한 탐색항해에 의한 읽기 연산에 대해서는 짧은 공유로크를 사용한다. 따라서, 탐색항해

안정성에서는 이러한 응용에 대하여 제 3.2절에서 제시한 커서 안정성의 일관성 문제가 모두 발생한다. 그리고, 탐색항해 안정성에서는 각 단위 탐색항해가 일관성을 잃은 복합객체를 읽는 문제를 방지하지만, 트랜잭션의 서로 다른 두 단위 탐색항해가 동일한 복합객체를 액세스하면 트랜잭션은 일관성을 잃은 복합객체를 읽을 수 있다. 반면, 수준 3에서는 모든 응용에 대하여 이러한 일관성 문제가 발생하지 않는다. 그러나, 대다수의 탐색항해 응용은 탐색항해 모델의 가정을 만족하므로 이러한 탐색항해 안정성의 일관성 문제는 크게 문제가 되지 않는다.

탐색항해 안정성에서는 단위 탐색항해에서 액세스한 복합객체는 탐색항해 커서가 다른 루트 객체로 이동할 때까지 안정된(stable) 상태로 유지된다. 즉, 다른 트랜잭션에 의해 갱신될 수 없다. 반면, 커서 안정성에서는 단지 루트 객체만 안정된 상태로 유지한다. 그러므로, 정리 1 ~ 3에 의해 탐색항해 안정성은 탐색항해 응용에 대한 커서 안정성의 일관성 문제를 해결하는 커서 안정성의 확장이다.

수준 3과 커서 안정성과의 비교를 통해 탐색항해 안정성의 동시성을 분석한다. 두 연산의 충돌의 타입에는 세 가지가 있다. 같은 스케줄에 속하는 충돌인 두 연산 op_1과 op_2가 있을 때 '쓰기→읽기' 충돌 타입은 op_1이 op_2보다 먼저 수행되고 op_1은 쓰기 연산, op_2는 읽기 연산인 충돌의 타입을 말한다. 이와 같은 방법으로, 다른 두 충돌 타입 '쓰기→쓰기'와 '읽기→쓰기'가 정의된다. 스케줄에서 충돌인 두 연산이 나타나면 나중에 객체를 액세스하려는 연산은 먼저 액세스한 연산이 로크를 반환할 때까지 대기한다. 이때 나중에 수행되는 연산의 대기 모드는 먼저 객체를 액세스한 연산이 획득한 로크의 로크 기간에 의해 결정된다. 만약 긴 로크를 획득했다면 **긴 대기 모드**(*long delay mode*), 짧은 로크를 획득했다면 **짧은 대기 모드**(*short delay mode*), 그리고 중간 로크를 획득했다면 **중간 대기 모드**(*medium delay mode*)라고 정의한다.

표 2는 수준 3과 탐색항해 안정성, 그리고 커서 안정성의 충돌 타입에 대한 대기 모드를 비교한다. 수준 3에서는 읽기 연산과 쓰기 연산 모두 긴 로크를 사용하기 때문에 모든 충돌 타입에 대해 나중에 수행되는 연산은 모두 긴 대기 모드로 대기한다. 반면, 탐색항해 안정성은 쓰기 연산만 긴 로크를 사용하고 읽기 연산은 커서가 루트 객체를 가리키는 동안만 유지되는 중간 로크를 사용한다. 따라서 '읽기→쓰기' 충돌 시 쓰기 연산은 중간 대기 모드로 대기한다. 그리고, 커서 안정성에서는 커서를 사용한 읽기 연산의 경우 '읽기→쓰기' 충돌 시 쓰기 연산은 중간 대기 모드로 대기하며, 커서를 사용하지 않은 읽기 연산의 경우 '읽기→쓰기' 충돌 시 짧은 대기로 대기한다. 긴 대기 시간은 중간 대기 시간보다 훨씬 길며, 중간 대기 시간은 짧은 대기 시간보다 약간 길다. 그러므로, 탐색항해 안정성은 수준 3보다 동시성을 높인다.

표 2 수준 3, 탐색항해 안정성, 그리고 커서 안정성에서 충돌에 대한 대기 모드의 비교

충돌의 타입	수준 3	탐색항해 안정성	커서 안정성
읽기→쓰기	긴 대기	중간 대기	중간/짧은 대기[2]
쓰기→읽기	긴 대기	긴 대기	긴 대기
쓰기→쓰기	긴 대기	긴 대기	긴 대기

2 : 단, 커서를 사용한 읽기 연산의 경우, 탐색항해 안정성과 같은 '중간 대기'이고 그렇지 않은 경우는 '짧은 대기'이다

4.3 구현상의 이슈와 해결 방법

본 절에서는 ORDBMS에 탐색항해 안정성을 구현할 때 발생하는 이슈와 해결 방법을 제시한다. 첫째, 탐색항해 커서가 다른 루트객체로 이동할 때 단위 탐색항해에 의해 액세스 된 객체들에 대한 로크들 중 공유 로크들을 반환하기 위해 액세스 된 객체들의 목록을 유지해야 한다. 이 목록을 유지하기 위해 연산이 수행되는 시점에 그 연산의 탐색항해 시작점을 알아야 하는데, 이를 위해 다음과 같은 방법을 제시한다. ORDBMS는 탐색항해를 지원하기 위해 탐색항해 인터페이스에서 참조 타입의 프로그램 변수를 제공한다[4,5]. 제안하는 방법은 탐색항해 시작점을 함께 저장할 수 있도록 참조 타입의 데이타 구조를 확장하는 것이다. 즉, 커서를 사용하여 얻은 탐색항해 루트의 참조를 참조 타입의 변수에 지정할 때 탐색항해 시작점을 변수에 함께 저장할 수 있도록 한다. 이 참조 타입의 변수는 각 연산의 인자로 전달된다. 이러한 방법에 의해, 각 연산이 수행되는 시점에 그 연산의 탐색항해 시작점을 알 수 있다. 이 목록을 유지함에 의해 객체에 대한 로크 요청은 단위 탐색항해에서 처음 액세스 될 때와 공유 로크가 갱신에 의해 독점 로크로 변환되는 경우에만 이루어진다.

둘째, 객체가 두 개 이상의 단위 탐색항해에 의해 동시에 액세스되는 경우에 이 객체에 대한 공유 로크는 이들 단위 탐색항해들이 모두 종료할 때까지 유지되어야 한다. 이를 위해서, DBMS의 로킹 방법에서 사용하고 있는 로크 횟수(lock count)[2]를 이용한다. 로크 횟수는 객체에 대하여 트랜잭션이 로크를 획득한 횟수를 말한다. 단위 탐색항해에서 객체가 처음 읽혀질 때 로크

횟수를 1씩 증가하며 단위 탐색항해가 종료하면 로크 횟수를 1씩 감소한다. 로크 횟수가 0이면 실제로 로크를 반환한다.

5. 성능 평가

본 절에서는 시뮬레이션을 통하여 제안한 탐색항해 안정성과 수준 3의 성능을 비교하고 탐색항해 안정성이 크게 성능을 향상시킴을 보인다. 먼저 제 5.1절에서 시뮬레이션 모델을 설명한다. 그리고 제 5.2절에서 실험 방법을 설명한 후, 제 5.3절에서 실험 결과를 소개하고 분석한다.

5.1 시뮬레이션 모델

본 논문에서는 성능 평가를 위해 Wang 및 Rowe의 모델[15](간단히 **WR 모델**이라 함)을 확장한다. WR 모델은 클라이언트/서버 구조의 DBMS에 대한 동시성 제어 방법의 성능을 평가하기 위한 모델이다. 본 실험에서는 탐색항해 안정성을 지원하기 위해 복합객체와 커서 연산을 추가하여 WR 모델을 확장한다. 시뮬레이터는 시뮬레이션 언어인 CSIM[16]을 사용하여 구현하였다.

시뮬레이터의 시스템 구조는 그림 1과 같이 크게 클라이언트, 서버, 그리고 네트워크의 세 가지 모듈로 구성된다. 클라이언트는 트랜잭션 생성자, 트랜잭션 관리자, 그리고 자원 관리자로 이루어진다. 트랜잭션 생성자는 트랜잭션이 완료하면 또 하나의 트랜잭션을 생성하는 방법으로 트랜잭션을 생성한다. 트랜잭션 관리자는 캐쉬의 객체들을 사용하여 트랜잭션을 수행한다. 서버는 트랜잭션 관리자, 로크 관리자, 그리고 자원 관리자로 구성된다. 트랜잭션 관리자는 로크 관리자와 버퍼 관리자, 그리고 자원 관리자와 상호 협력하여 서버에 대한 트랜잭션의 요청을 수행한다. 자원관리자는 CPU와 디스크를 선도착 선처리(first come, first served) 방식으로 서비스한다.

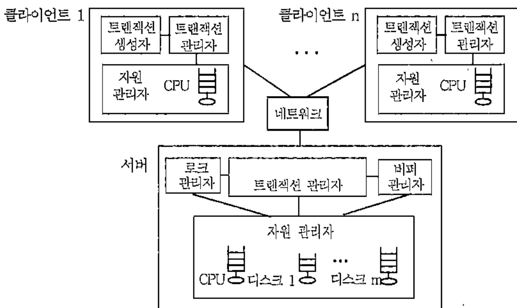


그림 1 ORDBMS의 시뮬레이션 시스템 구조

표 3은 시스템 변수와 설명, 그리고 그 값을 요약한 것이다. 시스템 변수는 기본적으로 WR 모델과 동일하며, 여기서는 WR 모델과 다른 가지를 변수만을 설명한다. 평균 메시지 지연 시간 *NetDelay*는 최근에 많이 사용되는 고속 LAN의 대역폭인 100Mbps로 가정하여 0.08msecs로 설정하였다. CPU 속도는 WR 모델과 같이 서버가 클라이언트보다 두 배가 빠르게 하였으나, 하드웨어 기술 발전을 고려하여 양쪽 모두 50배가 증가한 값으로 설정한다. 데이타 디스크의 개수 *NDataDisks*는 디스크 액세스에 의한 병목현상에 의해 실험 결과가 왜곡되지 않도록 하기 위해 충분히 많은 5개로 설정한다.

페이지가 버퍼에서 액세스 될 확률 *BufferHit*는 0.2로 설정한다. 디스크의 탐색시간(seek time)은 *SeekLow*과 *SeekHigh*사이의 균일 분포를 가정한다. 디스크 속도에 대한 변수들 *SeekLow*, *SeekHigh*, *DiskRot*, *DiskTran*의 값은 Seagate's Cheetah 15X[17]을 기준으로 정한 값이다.

표 3 시스템 변수와 그의 설명, 그리고 실험에서 사용한 변수값

변수명	설명	변수값
NetDelay	네트워크의 평균 메시지 지연 시간	0.08 msecs
PacketSize	한 메시지 패킷의 최대 크기	4,096 bytes
MsgCost	한 메시지 패킷을 송/수신하는데 드는 CPU 비용	5,000 instructions
NClients	클라이언트의 수	1, 20, 40, $\cdots$, 100개
NClientCPUs	한 클라이언트가 가진 CPU 개수	1개
ClientMips	클라이언트 CPU의 속도	50 MIPS
NserverCPUs	서버가 가진 CPU 개수	1개
ServerMips	서버 CPU의 속도	100 MIPS
NDataDisks	데이타 디스크의 수	5개
NLogDisks	로그 디스크의 수	1개
BufferHit	페이지를 버퍼에서 액세스할 확률	0.2
SeekLow	디스크의 최소 탐색시간	0 msec
SeekHigh	디스크 최대 탐색시간	8.4 msecs
DiskRot	디스크 평균 회전시간	2.0 msecs
DiskTran	한 디스크 블럭의 전송시간	0.1 msecs
PageSize	디스크 블럭의 크기	4,096 bytes
InitDiskCost	디스크 액세스를 초기화하는 CPU 비용	5,000 instructions
ServerProc Page	서버에서 한 페이지를 처리하는데 드는 CPU 비용	10,000 instructions
ClientProc Obj	클라이언트에서 한 객체를 처리하는데 드는 CPU 비용	20,000 instructions

데이타베이스는 2,000개의 복합객체들로 이루어진다. 각각의 복합객체는 모두 10개의 컴포넌트 객체들을 포함하며, 각 컴포넌트 객체의 크기는 모두 100바이트로 가정한다.

트랜잭션은 WR 모델에서 지원한 5개의 연산 (1)~(5)과 탐색항해를 모델링하기 위해 추가한 3개의 커서 연산 (6)~(8)로 이루어진다.

(1) *BeginXact*: 트랜잭션을 시작한다.
(2) *ReadObject*: 객체를 클라이언트의 캐쉬로 가져와서 읽는다.
(3) *UpdateObject*: 클라이언트 캐쉬에서 객체를 갱신한다.
(4) *CommitXact*: 트랜잭션을 완료한다.
(5) *AbortXact*: 트랜잭션을 철회한다.
(6) *OpenCursor*: 커서를 연다.
(7) *FetchCursor*: 커서를 사용하여 루트 객체를 캐쉬로 가져와서 읽는다.
(8) *CloseCursor*: 커서를 닫는다.

그림 2는 WR 모델을 확장한 탐색항해 트랜잭션 모델을 나타낸다. 탐색항해 트랜잭션은 단위 탐색항해들의 시퀀스이다. 각각의 단위 탐색항해는 하나의 *FetchCursor* 연산과 여러 번의 *ReadObject*, *UpdateObject* 연산으로 구성된다. 트랜잭션 크기(그림 2의 *transaction_size*)는 트랜잭션이 액세스하는 복합객체의 개수로 정의한다. 그림에서 'dotimes(n)'과 'end dotimes'의 의미는 이들 사이의 연산들을 n번 반복하는 것을 의미한다.

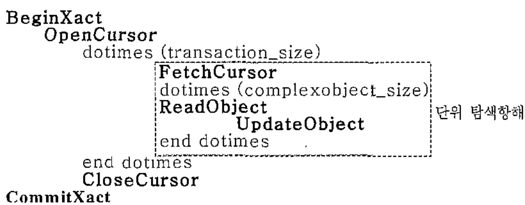


그림 2 탐색항해 트랜잭션 모델

탐색항해 트랜잭션의 계산 집약적인(computation intensive) 특징을 반영하기 위해 디스크 I/O 시간과 클라이언트의 CPU에서 소모되는 시간의 비율을 1:1로 설정한다. 트랜잭션은 트랜잭션 크기에 의해 정의된 세 가지 유형을 사용한다. 즉, 모든 트랜잭션 크기가 10인 경우('SHORT'이라 함), 모든 트랜잭션 크기가 50인 경우('LONG'이라 함), 그리고 트랜잭션 크기가 [10, 50]사이에서 균일분포를 가지는 경우('VLENGTH'라 함)이다. 트랜잭션이 복합객체를 갱신할 확률 *ProbWrite*('갱신 확률'이라 함)에 따른 성능 변화를 관찰하기 위해 갱신 확률 0.1, 0.5, 0.9 세 값에 대해 실험을 수행한다.

5.2 실험 방법

실험은 트랜잭션 유형과 읽기 전용 트랜잭션과 읽기 및 쓰기 트랜잭션의 비율(간단히 '읽기:쓰기 비율'이라고 함) 두 항목을 변화시켜 가면서 네 가지 실험을 수행한다. 실험 1은 읽기:쓰기 비율이 8:2이고 트랜잭션의 유형이 LONG인 실험으로 큰 크기의 트랜잭션에 대하여 실험한다. 실험 2에서는 읽기:쓰기 비율이 8:2이고 트랜잭션의 유형이 SHORT인 실험으로 작은 크기의 트랜잭션에 대하여 실험한다. 실험 3에서는 읽기:쓰기 비율이 8:2이고 트랜잭션의 유형이 VLENGTH인 실험으로 다양한 크기의 트랜잭션에 대하여 실험한다. 실험 4에서는 읽기:쓰기 비율이 2:8이고 트랜잭션의 유형이 VLENGTH인 실험으로 읽기:쓰기 비율의 효과를 분석하기 위해 실험한다. 실험은 본 논문에서 제안한 고립화 수준인 탐색항해 안정성('NS'로 표기)과 수준 3('2PL'로 표기)의 성능을 비교 평가한다. 각 실험은 세 가지 갱신 확률 0.1, 0.5, 0.9에 대하여 수행한다.

각 실험은 총 5,000개의 트랜잭션이 완료될 때까지의 성능을 평가한다. 만약 트랜잭션이 교착상태에 의해 철회되면, 철회된 트랜잭션은 동일한 클라이언트에서 1초 후에 다시 시작한다. 실험은 동시에 수행되는 트랜잭션의 수를 변화하면서 성능을 측정한다. 성능을 평가하기 위한 척도로는 단위시간당 트랜잭션 처리율(간단히 트랜잭션 처리율이라고 함), 트랜잭션의 평균 응답시간, 그리고 트랜잭션당 평균 철회율(=총철회횟수/완료된트랜잭션의수) (간단히 철회율이라고 함)을 사용한다.

5.3 실험 결과

실험 1 : 큰 크기의 트랜잭션에 대한 효과를 분석하는 실험(LONG)

이 실험은 큰 크기의 트랜잭션에 대한 효과를 분석하기 위한 실험이다. 그림 3는 실험1의 트랜잭션 처리율을 나타낸다. 그림 3(a)는 갱신 확률이 0.1인 경우이고, (b)는 갱신 확률이 0.5인 경우이며, (c)는 갱신 확률이 0.9인 경우이다. 그림 3을 보면, NS는 클라이언트의 수가 20이상이면 PL에 비해 트랜잭션 처리율을 크게 향상시킨다. 그림 3(a)에서 2PL의 경우 트랜잭션 처리율이 20 이상에서 빠르게 감소하는 반면, NS의 경우는 60 이상에서 매우 느리게 감소한다. 트랜잭션은 매우 많은 객체를 액세스하므로 트랜잭션들간에 동일한 객체를 액세스할 빈도가 높아진다. 읽기 트랜잭션의 비율이 높고 트랜잭션의 수행시간이 길기 때문에, 클라이언트의 수가 증

가할수록 '읽기→쓰기' 충돌이 많이 발생하며 그 충돌에 대하여 긴 대기 모드로 대기한다. 그러므로 2PL의 트랜잭션 처리율은 급격히 감소한다. 반면, NS에서 트랜잭션은 '읽기→쓰기'에 대하여 중간 모드로 대기하기 때문에 클라이언트의 수가 60일 때까지는 계속 증가하며 60보다 클 때도 느리게 감소한다. 결국, NS는 2PL에 비해 성능이 크게 우수하다.

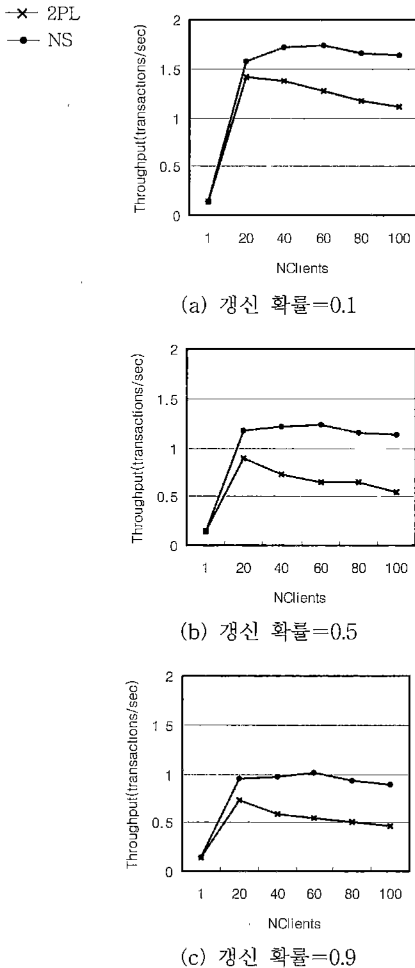


그림 3 트랜잭션 유형이 LONG이고 읽기:쓰기 비율이 8:2인 경우의 트랜잭션 처리율

그림 3(b)와 (c)는 그림 3(a)와 유사한 모양을 가진다. 그러나, 두 그래프의 높이는 갱신 확률이 높아질수록 낮아진다. 이는 갱신 확률이 높아질수록 트랜잭션은 더 많은 쓰기 연산을 가지며 충돌 발생 확률도 더 높아지기 때문이다. 그래프를 보면 클라이언트의 수가 커질수록 두 수준의 격차는 커진다. 특히, 그림 3(b)에서 NS는 2PL에 비하여 트랜잭션 처리율을 200%까지 높인다.

그림 4는 실험 1의 평균 응답 시간을 보여준다. 그림 4에서 클라이언트의 수가 증가할수록 NS의 평균 응답 시간은 2PL에 비해 천천히 증가한다. 많은 트랜잭션이 동시에 수행되면 충돌에 의한 대기 시간의 증가와 교착상태에 의한 철회 횟수의 증가로 인해 응답시간은 증가한다. NS는 충돌 발생 시 대기 시간을 줄이기 때문에 2PL에 비해 응답 시간을 줄인다. 특히, 그림 4(b)에서 NS은 2PL에 비하여 최대 55%까지 평균 응답 시간을 줄인다.

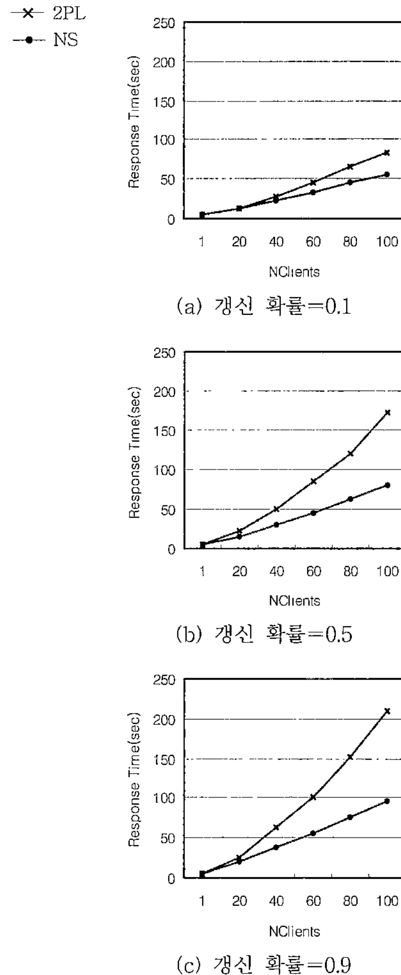
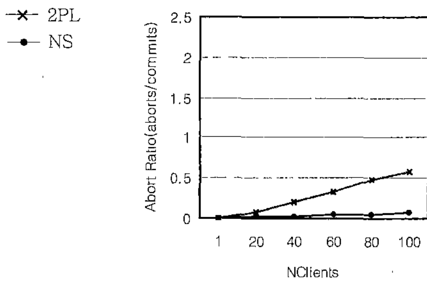
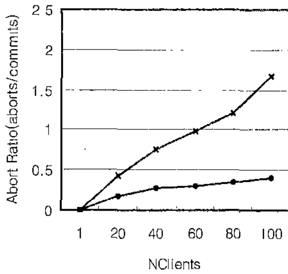


그림 4 트랜잭션 유형이 LONG이고 읽기:쓰기 비율이 8:2인 경우의 평균 응답 시간

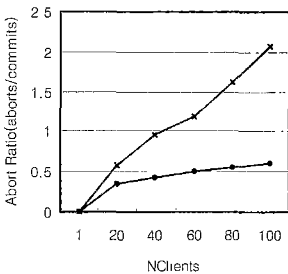
그림 5는 실험 1의 철회율을 보여준다. 그림 5에서 클라이언트의 수가 증가할수록, NS의 철회율은 2PL에 비해 천천히 증가한다. 트랜잭션은 2PL에서 획득한 로크를 더 오랫동안 유지하므로 더 많은 교착상태가 발생하고 따라서 더 많은 트랜잭션이 철회된다. 특히, 그림 5(b)에서 NS은 2PL에 비하여 최대 77%까지 철회율을 줄인다.



(a) 갱신 확률=0.1



(b) 갱신 확률=0.5



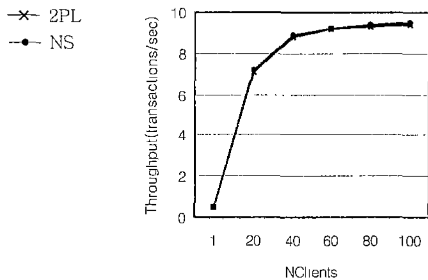
(c) 갱신 확률=0.9

그림 5 트랜잭션 유형이 LONG이고 읽기:쓰기 비율이 8:2인 경우의 철회율

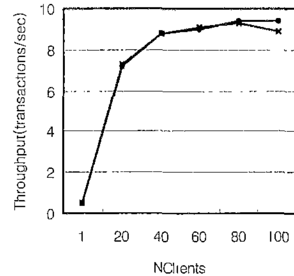
실험 2 : 작은 크기의 트랜잭션에 대한 효과를 분석하는 실험(SHORT)

이 실험은 작은 크기의 트랜잭션에 대한 효과를 분석하기 위해 수행한다. 그림 6는 실험 2에 대한 트랜잭션 처리율을 나타낸다. 그림 6을 보면, NS의 트랜잭션 처리율은 2PL과 거의 동일하다.

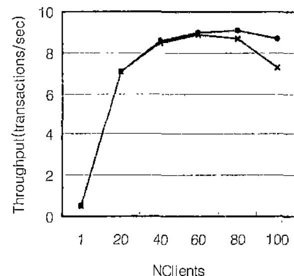
이는 트랜잭션 크기가 작을 때는 짧은 수행 시간에 의해 충돌이 적게 발생하기 때문이다. 그러나, 그림 6(b)에서 보면 2PL의 트랜잭션 처리율은 클라이언트의 개수 100에서 감소했으며, 그림 6(c)에서 보면 클라이언트의 개수가 80일 때부터 감소하였다. 이는 트랜잭션이 더 많은 쓰기 연산을 수행하여 충돌 발생 확률이 높아지기 때문이다.



(a) 갱신 확률=0.1



(b) 갱신 확률=0.5



(c) 갱신 확률=0.9

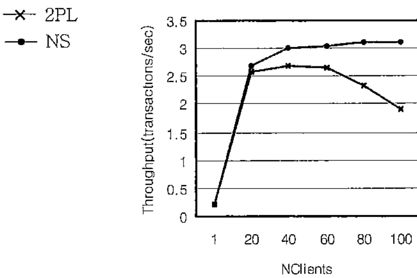
그림 6 트랜잭션 유형이 LONG이고 읽기:쓰기 비율이 8:2인 경우의 트랜잭션 처리율

실험 3: 다양한 크기의 트랜잭션에 대한 효과를 분석하는 실험(VLENGTH)

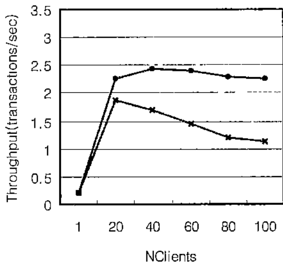
이 실험은 실제 환경에서와 같이 다양한 크기의 트랜잭션에 대한 효과를 분석하기 위한 실험이다. 그림 7는 실험 3에 대한 트랜잭션 처리율을 나타낸다. 트랜잭션 크기가 [10, 50]사이에서 균일하게 생성되기 때문에 그들 중에서 상당 부분의 크기는 50에 가깝다. 이러한 이유로 NS와 2PL의 트랜잭션 처리율은 실험 1의 결과와 유사하다. 즉, 클라이언트의 수가 20 이상이면 NS는 2PL에 비해 크게 우수하다. 실험 결과로부터 다양한 크기의 트랜잭션이 함께 수행되는 실제 환경에서 NS가 2PL에 비해 성능을 크게 향상시킴을 알 수 있다.

실험 4: 읽기:쓰기 비율의 효과를 분석하는 실험

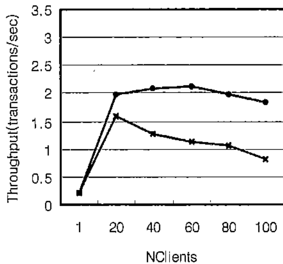
이 실험은 읽기 및 갱신 트랜잭션이 읽기 전용 트랜잭션에 비해서 매우 많은 경우의 효과를 분석하기 위한 실험이다. 그림 8은 실험 4에 대한 트랜잭션 처리율을 나타낸다. 이 경우, NS와 2PL의 트랜잭션 처리율의 차이는 읽기:쓰기 비율이 8:2인 경우에 비해 크게 줄어들었다. 그리고 높은 갱신 확률에 대하여 클라이언트의 수가 커질 때, 2PL뿐만 아니라 NS의 트랜잭션 처리율도 급격히 저하함을 볼 수 있다. 그 이유는 읽기:쓰기 비율이 2:8이면 쓰기 연산이 긴 로크를 요청하기 때문에 충돌이 더 자주 발생하기 때문이다. 실험 결과는 NS의 성능이 여전히 2PL보다 우수함을 보여준다.



(a) 갱신 확률=0.1

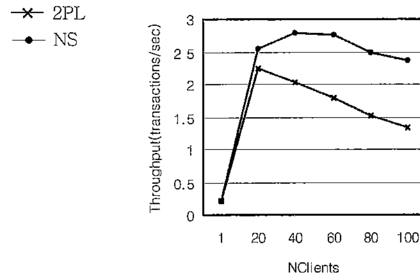


(b) 갱신 확률=0.5

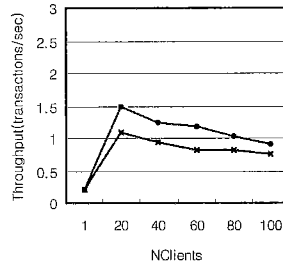


(c) 갱신 확률=0.9

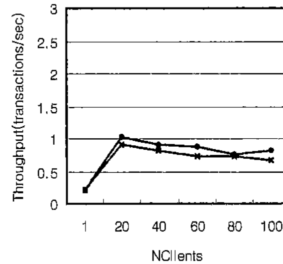
그림 7 트랜잭션 유형이 VLENGTH이고 읽기:쓰기 비율이 8:2인 경우의 트랜잭션 처리율



(a) 갱신 확률=0.1



(b) 갱신 확률=0.5



(c) 갱신 확률=0.9

그림 8 트랜잭션 유형이 VLENGTH이고 읽기:쓰기 비율이 2:8인 경우의 트랜잭션 처리율

6. 결 론

고립화 수준 2의 변형인 커서 안정성은 엄정 이단계 로킹(고립화 수준 3)의 동시성 저하를 피하기 위한 방법으로 RDBMS에서 유용하게 사용되어 왔다. 그러나 ORDBMS의 탐색항해 응용에 대해서는 일관성 문제로 인하여 커서 안정성은 더 이상 유용한 고립화 수준이 되지 못하는 문제가 발생하였다.

본 논문에서는 ORDBMS에서 수준 3의 동시성 저하를 피하고 커서 안정성의 일관성 문제를 해결하는 새로운 고립화 수준인 탐색항해 안정성을 제안하였다. 본 논문에서는 우선 커서 안정성을 탐색항해 응용에 적용했을 때 나타나는 허상 포인터 문제, 갱신 분실 문제, 그리고 일관성을 잃은 복합객체를 읽는 문제가 발생함을 밝혔다. ORDBMS에서 커서 안정성의 일관성 문제는 RDBMS에서 발생하지 않던 새로운 문제들이다. 다음으로, 커서와 탐색항해를 사용하여 복합객체를 액세스하는 패턴을 분석하고 이를 기반으로 새로운 고립화 수준인 탐색항해 안정성을 정의하였다. 그리고, 제안한 탐색항해 안정성이 커서 안정성의 세 가지 일관성 문제를 일으키지 않음을 증명하였다. 마지막으로, 다양한 실험을 통해 탐색항해 안정성이 동시성과 성능이 수준 3에 비해 크게 향상됨을 보였다. 특히, 수행 시간이 긴 트랜잭션이 많은 경우, 탐색항해 안정성은 수준 3에 비해서 트랜잭션 처리율을 최대 200%까지 향상시키고, 평균 응답 시간을 최대 55% 줄이며, 철회율을 최대 77% 줄였다.

이러한 결과로부터 다음과 같은 결론을 내릴 수 있다. 첫째, 탐색항해 안정성은 커서 안정성의 일관성 문제를 해결하는 커서 안정성의 확장이다. 둘째, 탐색항해 안정성은 ORDBMS의 탐색항해 응용이 성능 향상을 위해 수준 3 대신 선택할 수 있는 유용한 고립화 수준이다.

참 고 문 헌

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

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

[3] Chamberlin, D.D., *A Complete Guide to DB2 Universal Database*, Morgan Kaufmann, 1998.

[4] Oracle, Inc., Oracle Call Interface Programmer's Guide, Volumes 1 & 2, Release 8.0 Part No. A58234-01, 1997.

[5] UniSQL, C Application Programming Interface Reference Manual, 1996.

[6] Gray, J., et al., Granularity of Locks and Degrees of Consistency in a Shared Database, In *Readings in Database Systems*, 2nd ed., Michael Stonebraker, ed., Morgan Kaufmann, 1994(originally published as IBM Research Report RJ1654 in 1975).

[7] Berenson, H., et al, "A Critique of ANSI SQL Isolation Levels," In *Proc. Int'l Conf. on Management of Data*, ACM SIGMOD, pp.1-10, San Jose, California, May 1995.

[8] Keller, T., Graefe, G., and Maier, D., "Efficient Assembly of Complex Objects," In *Proc. Int'l Conf. on Management of Data*, ACM SIGMOD, pp.148-157, Denver, Colorado, May 1991.

[9] Stonbraker, M. and Moore, D., *Object-Relational DBMSs: The Next Great Wave*, Morgan Kaufman Publishers, 1996.

[10] Melton, J. and Simon, A. R., *Understanding the New SQL: A Complete Guide*, Morgan Kaufmann, 1993.

[11] Whang, K. Y. and Krishnamurthy, R., "Query Optimization in a Memory-Resident Domain Relational Calculus Database System," *ACM Trans. on Database Systems*, Vol. 15, No. 1, pp.67-95, Mar. 1990.

[12] Cattell, R.G.G. and Barry, D.K., *The Object Database Standard: ODMG2.0*, Morgan Kaufmann, 1997.

[13] Ullman, J. D. and Widom, J., *A First Course in Database Systems*, Prentice Hall, 1997.

[14] Silberschatz, A., Korth, H. F., and Sudarshan, A., *Database System Concepts*, McGraw-Hill, 3rd ed., 1997.

[15] Wang, Y. and Rowe, A. L., "Cache Consistency and Concurrency Control in a Client/Server DBMS Architecture," In *Proc. Int'l Conf. on Management of Data*, ACM SIGMOD, pp.367-376, Denver, Colorado, June 1991.

[16] Mesquite, Inc., CSIM User Guide, http://www.mesquite.com, 1998.

[17] Seagate, Inc., Seagate Barracuda 50 Family Product Specification, http://www.seagate.com, 1999.

부록: 정리 3의 증명

다수의 탐색항해 트랜잭션이 탐색항해 안정성에서 수행된 결과로 발생한 스케줄 S가 연산들간에 순서 '<'를 가진다고 하자. 스케줄 S에 대해 다음의 두 명제를 증명하면 충분하다.

(1) S는 허상 포인터를 데이타베이스에 완료하지 않는다.

(2) S에 포함된 어느 트랜잭션도 허상 포인터를 만나

지 않는다.

명제 (1)을 증명하는 대신 다음의 명제 (1)′을 증명하면 충분하다.

(1)′ S에 포함된 읽기 연산은 허상 포인터를 포함한 객체를 읽지 않는다.

그 이유는 다음과 같다. 만약 명제 (1)′이 만족하고 명제 (1)이 만족하지 않는다면, S는 어떤 허상 포인터를 데이타베이스에 완료한다. 그러면, S이후에 혼자 수행되는 트랜잭션 T가 그 허상 포인터를 포함한 객체를 읽는다고 할 때 S와 T로 구성된 스케줄 S′은 명제 (1)′을 만족하지 않는다. 이것은 명제 (1)′이 만족한다는 가정에 대한 모순이다. 그러므로, 명제 (1)′은 명제 (1)의 충분 조건이다.

명제 (1)′의 증명: 증명은 명제 (1)′이 만족하지 않는다고 가정하고 모순을 유도한다. S에 포함된 o1에 대한 읽기 연산들 중 $r_1[o_1]$이 허상 포인터인 o_2에 대한 참조를 포함한 o_1을 읽는 첫 읽기 연산이라고 하자. 그러면, o_2를 삭제하는 연산 $w_2[o_2]$를 포함하는 트랜잭션 T_2가 존재하여 $w_2[o_2] < c_2 < r_1[o_1]$을 만족해야 한다. T_2는 참조 무결성을 유지하기 위해 $w_2[o_2]$시점에 o_2를 참조하는 모든 객체를 더 이상 o_2를 참조하지 않도록 갱신한다. 그런데, $r_1[o_1]$은 o_2에 대한 참조를 포함하는 o_2를 읽었으므로 참조관계 $o_1 \rightarrow o_2$를 생성하는 연산 $w_3[o_1]$을 포함하는 트랜잭션 T_3가 존재한다. 그리고 $w_3[o_1] < c_3 < r_1[o_1]$의 순서를 만족해야 한다.

트랜잭션 T3는 o2의 존재를 확인하기 위해 읽기 연산 $r_3[o_2]$를 포함해야 한다. o_2가 삭제된 후에는 o_2를 읽을 수 없기 때문에 $r_3[o_2] < w_2[o_2]$이어야 한다. 이때, $w_3[o_1]$이 수행되는 순서로 (i) $w_3[o_1] < r_3[o_2] < w_2[o_2]$, (ii) $r_3[o_2] < w_3[o_1] < w_2[o_2]$, 또는 (iii) $r_3[o_2] < w_2[o_2] < w_3[o_1]$의 세 경우만이 가능하다. 그런데, (i)과 (ii)의 경우는 T_2가 참조 무결성을 유지하기 위해서 o_1이 o_2를 참조하지 않도록 o_1을 갱신하기 때문에, T_2가 완료된 후에 $r_1[o_1]$이 o_2에 대한 참조를 포함한 o_1을 읽지 못한다. 따라서 가정에 대해 모순이다. 또한, (iii)의 경우는 옳은 트랜잭션의 가정에 의해 $r_3[o_2]$는 $w_3[o_1]$과 동일한 단위 탐색항해에 속하므로 T_3가 획득한 o_1에 대한 공유 로크 때문에 $r_3[o_2] < w_2[o_2] < w_3[o_1]$의 순서로 수행될 수 없다. 따라서 이 경우 역시 가정에 대해 모순이다. 그러므로, 허상 포인터를 포함한 객체를 읽는 스케줄 S가 존재한다는 가정은 모순이다.

명제 (2)의 증명: 증명은 명제 (1)′의 증명처럼 모순을 유도한다. 트랜잭션 T_1은 $r_1[o_1]$에 의해 읽은 o_1은 o_2에 대한 참조를 포함하며, 허상 포인터인 o_2에 참조를 사용하여 o_2를 읽는(즉, $r_1[o_2]$) 연산을 포함한다고 가정하자. 그러면, $r_1[o_1]$과 $r_1[o_2]$은 동일한 단위 탐색항해에 포함되어야 한다. 명제(1)′의 증명에 의해 $r_1[o_1]$이 읽은 o_1은 허상 포인터를 포함하지 않으므로 o_2를 삭제하는 트랜잭션 T_2가 존재하여 T_2의 완료 연산 c_2는 $r_1[o_1] < c_2 < r_1[o_2]$를 만족해야 한다. 그런데, T_2는 참조 무결성을 유지하기 위해 참조관계 $o_1 \rightarrow o_2$를 허상 포인터가 되지 않도록 o_1을 갱신하는 연산 $w_2[o_1]$을 포함해야 한다. 그리고, 탐색항해 안정성에서 T_1은 o_1에 대한 공유 로크를 $r_1[o_1]$과 같은 단위 탐색항해에 속하는 연산 $r_1[o_2]$가 수행되는 시점까지 유지한다. 따라서, $r_1[o_1] < w_2[o_1] < r_1[o_2]$의 순서로 수행될 수 없으며, 독점 로크는 완료 시점까지 유지되므로 $w_2[o_1] < r_1[o_1] < c_2$일 수도 없다. 그러므로, 허상 포인트를 만나는 스케줄은 발생할 수 없으므로 S의 트랜잭션 T_1이 허상 포인터를 만났다는 가정은 모순이다.

결과적으로, 명제 (1)′과 (2)의 증명에 의해 정리 3의 증명이 완료된다.

서 홍 석

1988년 2월 서울대학교 수학과 학사. 1990년 2월 한국과학기술원 응용수학과 석사. 1996년 3월 ~ 현재 한국과학기술원 전산학과 박사과정. 1990년 2월 ~ 현재 삼성 SDS 재직중. 관심분야는 객체관계형 DBMS, 동시성 제어, 분산 시스템

장 지 웅

1993년 2월 연세대학교 컴퓨터공학과 학사. 1995년 2월 한국과학기술원 전산학과 석사. 1995년 3월 ~ 현재 한국과학기술원 전산학과 박사과정. 관심분야는 동시성 제어, 실시간 DBMS

문 양 세

정보과학회논문지 : 데이타베이스 제 28 권 제 1 호 참조

황 규 영

정보과학회논문지 : 데이타베이스 제 28 권 제 1 호 참조

홍 의 경
1981년 서울대학교 사범대학 수학과(학사). 1983년 한국과학기술원 전산학과(석사). 1991년 한국과학기술원 전산학과(박사). 1984년 ~ 현재 서울시립대학교 전산통계학과 교수. 1997년 ~ 1998년 한국정보과학회 논문지 편집위원. 2000년 ~ 현재 한국정보과학회 데이타베이스연구회 편집부장. 관심분야는 Database Systems, XML, Data Mining, Data Warehouse, GIS, Distributed Database