객체지향 데이타베이스 관리 시스템에서의 부분 철회

(Partial Rollback in Object-Oriented Database Management Systems)

김 원 영 † 이 영 구 †† 황 규 영 †††
(Won-Young Kim) (Young-Koo Lee) (Kyu-Young Whang)

요 약 데이타베이스 관리 시스템(DBMS)에서의 부분 철회는 수행 중인 트랜잭션 전체를 철회하지 않고 일부분만을 철회할 수 있는 유용한 기능으로 많은 관계형 DBMS(RDBMS)에서 지원되고 있다. 그러나, RDBMS와는 달리 객체지향 DBMS(OODBMS)에서는 객체 버퍼와 페이지 버퍼로 구성되는 이중의 버퍼구조를 유지하므로 페이지 버퍼만을 유지하는 RDBMS에서 사용하던 기존의 부분 철회 방식을 그대로 이용할 수 없다. 따라서, 이러한 이중 버퍼 내의 데이타를 효과적으로 부분 철회할 수 있는 새로운 회복 방법이 필요하다. 본 논문에서는 이중 버퍼구조를 사용하는 OODBMS를 위한 네가지 부분 철회 방식들을 제안한다. 제안된 부분 철회 방식들은 부분 철회의 주요 대상인 버퍼의 구조에 따라 단일 버퍼 기반 방식과 이중 버퍼 기반 방식으로 분류되며, 이들은 다시 페이지 버퍼 기반 부분 철회, 객체 버퍼 기반 부분 철회, 소프트 로그를 이용한 이중 버퍼 기반 부분 철회, 그리고 새도우를 이용한 이중 버퍼 기반 부분 철회 방식으로 세분화된다. 수학적 분석 및 실험을 통한 성능 평가에 의하면 새도우를 이용한 이중 버퍼 기반 부분 철회 방식의 성능이 가장 우수한 것으로 나타났다.

Abstract In database management systems(DBMSs), partial rollback is a useful facility that cancels part of the executed operations upon user's requests without a total rollback. Many relational DBMSs(RDBMSs) provide this facility. However, object-oriented DBMSs (OODBMSs) cannot utilize the previous recovery scheme of partial rollback used in RDBMSs since, unlike RDBMSs, they use a dual buffer consisting of an object buffer and a page buffer. Therefore, a new recovery scheme is required that rolls back the data efficiently in the dual buffer. We propose four partial rollback schemes in OODBMSs that use a dual buffer. We classify the proposed schemes into the single buffer based partial rollback scheme and the dual buffer based partial rollback scheme according to the number of buffers used for partial rollback processing. We further classify them into 1)the page buffer based partial rollback scheme, 2)the object buffer based partial rollback scheme, 3)the dual buffer based partial rollback scheme using soft log, and 4)the dual buffer based partial rollback scheme using shadows. We evaluate the performance by mathematical analysis and experiments. The results show that the dual buffer based partial rollback scheme using shadows provides the best performance.

1. 서 론

트랜잭션 철회(transaction rollback)는 데이타베이스 관리 시스템(DBMS)에서 오류가 발생한 경우 이미 수행된 연산들의 효과를 취소하는 기능이다[1][2][3]. 그러나, 이 기능을 이용하면 오류를 발생시킨 연산뿐만 아니라 그 이전에 성공적으로 수행되었던 연산들도 모두 취소하게 된다. 연산 결과에 대한 불만족이나 권한이 없는 데이타의 읽기 시도, 교착 상태 또는 존재하지 않는 데이타의 요구와 같은 심각하지 않은 오류가 발생한 경우 트랜잭션 전체를 철회하지 않고 오류가 발생한 트랜잭션의 일부만을 철회하기 위해 제안된 기능이 세이브

· 본 연구는 첨단정보기술연구센터를 통하여 과학재단의 지원과 웹기반 정보검색 OODBMS를 위한 Web-DBMS 통합기술의 개발과제를 통해 과학기술부의 지원을 받았음.
† 종신회원 : 한국전자통신연구원 실시간 DBMS팀 연구원
wykim@etri.re.kr
†† 종신회원 : 한국과학기술원 전자전산학과 전산학전공/첨단정보기술연구센터
yklee@mozart.kaist.ac.kr
††† 종신회원 : 한국과학기술원 전자전산학과 전산학전공/첨단정보기술연구센터 교수
kywhang@cs.kaist.ac.kr
논문접수 : 1999년 8월 31일
심사완료 : 2000년 9월 27일

포인트(savepoint) 개념을 이용한 부분 철회(partial rollback)이다[2][3]. 세이브포인트는 트랜잭션의 실행 도중에 시스템이나 사용자에 의해 설정된 임의의 시점이다. 이 기능을 이용하면 트랜잭션의 진행 중에 어떤 오류가 발생하더라도 오류가 발생한 이전의 세이브포인트 시점으로 트랜잭션의 상태를 되돌릴 수 있다.

이미 많은 DBMS에서 이러한 부분 철회 기능을 지원하고 있다. System R[4]에서는 부분 철회 기능을 RSS 인터페이스를 통하여 제공하였고, DB2[2]에서는 다중 튜플을 원자적으로 갱신하기 위해 내부적으로 부분 철회 기능을 사용하였다. 현재 널리 사용되고 있는 Oracle[5], Sybase[6]등의 상용 관계형 DBMS(relational DBMS: RDBMS)에서도 대부분 부분 철회 기능을 제공하고 있으며, 객체 관계 DBMS인 UniSQL[7]에서도 부분 철회 기능을 지원하고 있다. 이와 같이 부분 철회 기능을 지원하는 DBMS들이 이미 개발된 바 있으나, 부분 철회 지원을 위한 방법론에 관한 구체적인 연구의 발표는 그 수가 많지 않으며, 주로 RDBMS에 국한되어 왔다.

객체지향 데이타베이스 관리 시스템(object-oriented DBMS:OODBMS)은 CAD나 멀티미디어 저작 시스템과 같은 사용자와의 상호작용이 많은 응용들에서 그 하부구조로 사용되고 있다. 이러한 응용 환경에서는 사용자의 실수로 인해 수행된 연산이나 만족스럽지 못한 결과를 초래한 연산을 취소할 수 있는 기능이 꼭 필요하다. 따라서, OODBMS에서는 이러한 응용들을 위하여 부분 철회 기능의 지원이 매우 필수적이다. 대부분 OODBMS에서는 RDBMS와 달리 객체들에 대한 빠른 액세스를 지원하기 위해 페이지 버퍼와 객체 버퍼로 구성된 이중 버퍼(dual buffer)를 사용하고 있다[8][9]. 따라서, 이러한 구조에서는 페이지 버퍼뿐만 아니라 객체 버퍼에 걸쳐 존재하는 객체들을 어떻게 부분 철회할 것인가에 대한 체계적인 연구가 필요하다.

본 논문에서는 이중 버퍼를 가진 OODBMS를 위한 네가지 부분 철회 방식들을 제안하고 각 방식들의 성능을 비교하고 분석한다. 제안된 부분 철회 방식들은 부분 철회의 주요 대상인 버퍼의 구조에 따라 단일 버퍼 기반 방식과 이중 버퍼 기반 방식으로 분류된다. 단일 버퍼 기반 방식은 부분 철회시 하나의 버퍼만을 주요 처리 대상으로 사용하는 방식이며, 어떤 버퍼를 사용할 것인가에 따라 페이지 버퍼 기반 부분 철회 방식과 객체 버퍼 기반 부분 철회 방식으로 세분화된다. 이중 버퍼 기반 방식은 객체 버퍼와 페이지 버퍼 모두를 이용하여 부분 철회를 수행하는 방식이며, 객체 버퍼내에서의 회복용 데이타를 어떻게 관리할 것이가에 따라 소프트 로그를 이용한 이중 버퍼 기반 부분 철회 방식과 섀도우를 이용한 이중 버퍼 기반 부분 철회 방식으로 세분화된다. 제안된 각 방식의 성능을 규명하기 위하여 수학적인 분석과 실험에 의한 성능 평가를 수행한다.

논문의 구성은 다음과 같다. 제 2 절에서는 부분 철회에 대한 기존 연구를 살펴 보고 제 3절에서는 OODBMS에서의 이중 버퍼 구조와 이중 버퍼 구조가 부분 철회에 미치는 영향에 관하여 논의한다. 제 4 절에서는 OODBMS에서의 가능한 네가지 부분 철회 방식들을 제안하고 제 5 절에서는 각 부분 철회 방식의 성능을 비교하기 위한 시뮬레이션 모델과 성능 평가의 결과를 제시한다. 끝으로 제 6 절에서는 결론을 내린다.

2. RDBMS에서의 부분 철회

본 절에서는 관련 연구로서 RDBMS에서 부분 철회를 지원하는 방안을 소개한다. 먼저 RDBMS에서 부분 철회의 대상이 되는 자료구조와 자료 구조의 특성에 따른 회복 방식을 기술하고 부분 철회를 위한 두 가지 연산인 세이브포인트 설정 연산과 부분 철회 연산의 수행 알고리즘을 간략히 기술한다.

RDBMS에서는 부분 철회시 트랜잭션에 의해 갱신된 데이타베이스의 상태 뿐만 아니라 진행 중인 트랜잭션의 실행 상태를 복원해야 한다. 먼저, 트랜잭션에 의해 갱신된 데이타베이스는 페이지 버퍼와 디스크에 걸쳐 존재하는 갱신된 데이타들로 이루어진다. RDBMS에서는 이러한 데이타베이스의 상태를 세이브포인트 시점으로 되돌리기 위해서 트랜잭션의 철회 및 파손 회복을 위해 데이타의 갱신을 기록한 로그를 이용한다. 진행 중인 트랜잭션의 실행 상태는 트랜잭션이 수행되는 동안에 메모리상에서 점유한 자원인 로크, 커서, 그리고 볼륨이나 화일에 대한 액세스 정보 등으로 이루어 진다[2]. 이러한 자료구조들은 트랜잭션의 진행과 함께 로크의 획득 및 반환, 커서의 생성 및 소멸, 그리고 화일의 열기 및 닫기 등으로 인해 갱신이 자주 발생하고 데이타베이스에 비해 자료구조의 크기가 상당히 작다는 특징을 가진다. 이러한 특징으로 인해 RDBMS에서는 세이브포인트 설정시 해당 트랜잭션에 대한 실행 상태의 이미지인 스냅샷(snapshot)을 저장해두었다가 해당 세이브포인트로의 철회시 저장해 두었던 스냅샷을 복구하는 방식을 이용한다[2].

RDBMS에서의 세이브포인트 설정 연산과 부분 철회 연산은 다음과 같이 수행된다[3][4]. 세이브포인트 설정 연산 시에는 진행 중인 트랜잭션의 실행 상태와 현재 로그에 기록된 마지막 로그 레코드의 LSN인 SaveLSN

을 메모리에 기록하고 세이브포인트 식별자를 사용자에게 반환한다. 사용자는 세이브포인트 식별자를 통해 원하는 세이브포인트로의 철회를 요청할 수 있다. 부분 철회 연산 시에는 주어진 세이브포인트 식별자에 대응되는 SaveLSN 이후에 해당 트랜잭션에 의해 기록된 로그 레코드들을 발생한 시간의 역순으로 UNDO하고 기록되었던 트랜잭션의 실행 상태를 복원한다.

3. OODBMS에서의 이중 버퍼

RDBMS와 비교하여 OODBMS가 가지는 가장 큰 구조적 특징의 하나는 이중 버퍼를 갖는다는 것이다. 본 절에서는 OODBMS에서의 이중 버퍼의 구조적 특징과 이중 버퍼에서 객체의 액세스 과정을 살펴보고 이중 버퍼가 본 논문의 주요 논점인 부분 철회에 미치는 영향을 기술한다.

OODBMS에서는 페이지 버퍼와 객체 버퍼로 구성되는 이중 버퍼를 유지한다[10]. 페이지 버퍼는 디스크 입출력을 최소화하기 위한 것으로, 디스크에서와 동일한 형태로 객체들을 페이지 단위로 관리한다. 객체 버퍼는 포인터 스위즐링(pointer swizzling)[9]을 통하여 객체의 빠른 액세스를 지원하기 위한 것으로, 페이지 버퍼로부터 읽어들인 객체들을 메모리내에서 포인터를 사용하여 직접 액세스할 수 있도록 변환된 형태로 관리한다. 또한, 페이지 버퍼는 여러 사용자에 의해 공유되는 반면 객체 버퍼는 한 사용자에 의해 독점된다[10]. 특히, 객체를 주고 받는(object-shipping) 클라이언트-서버 환경[11]에서는 이중 버퍼가 클라이언트와 서버에 분산되어 관리된다. 페이지 버퍼는 서버에 유지되고 객체 버퍼는 클라이언트에 유지된다.

이중 버퍼를 가지는 OODBMS에서 객체를 액세스하는 과정은 다음과 같다. 어떤 객체를 액세스하기 위해서는 먼저 디스크에 있는 데이타베이스로부터 객체가 속한 데이타 페이지를 페이지 버퍼로 스왑-인(swap-in)하고 페이지 버퍼에서 해당 객체만을 읽어서 객체 버퍼로 스왑-인한다. 스왑-인의 수행시, 객체 버퍼나 페이지 버퍼가 가득찬 경우에는 버퍼 교체 알고리즘에 의해 희생자가 선정된다. 이 때, 희생자로 선정된 객체가 갱신되어 있는 경우 스왑-아웃(swap-out)되면서 페이지 버퍼에 반영된다. 해당 객체가 속한 데이타 페이지가 페이지 버퍼에 없는 경우에는 해당 페이지가 디스크로부터 다시 스왑-인되어야 한다. 페이지 버퍼에서 희생자로 선정되어 교체되는 페이지가 갱신되어 있는 경우 스왑-아웃되면서 디스크에 반영된다.

이와 같이, OODBMS에서는 데이타베이스가 디스크, 페이지 버퍼, 그리고 객체 버퍼에 산재되어 있으므로 부분 철회시 이들 모두의 상태를 회복해야 한다. 특히, 객체 버퍼의 부분 철회는 파손 회복이나 트랜잭션 철회와 달리 추가적인 조치가 필요하다. 파손 회복이나 트랜잭션 철회 이후에는 해당 트랜잭션이 더 이상 진행하지 않으므로 그 트랜잭션이 독점적으로 사용하던 객체 버퍼내의 상태를 단순히 무시하면 된다. 그러나, 부분 철회 이후에는 해당 트랜잭션이 계속 진행하면서 객체 버퍼를 사용하므로 부분 철회 연산은 객체 버퍼의 상태도 세이브포인트 시점으로 되돌려야 한다.

OODBMS의 객체 버퍼에서 수행된 갱신은 대응되는 로그 레코드가 기록되지 않을 뿐만 아니라 디스크에도 반영되지 않은 상태이다. RDBMS에서의 프로그램의 실행 상태와 같이 세이브포인트 시점의 스냅샷을 메모리에 저장해두었다가 부분 철회시 복구하는 방식을 객체 버퍼에 적용하는 것을 고려할 수 있다. 그러나, 이 방식은 세이브포인트마다 객체 버퍼의 스냅샷을 유지하기 위한 메모리 오버헤드가 너무 크기 때문에 바람직하지 않다. 이러한 요인을 고려하여 본 논문에서는 객체 버퍼를 포함한 데이타베이스의 상태를 회복할 수 있는 네가지 부분 철회 방식을 제시하고 이 방식들의 성능 평가를 수행한다.

4. OODBMS에서의 부분 철회 방식들

본 절에서는 OODBMS에서 부분 철회를 지원하기 위한 네가지 방식들을 제안한다. 제안된 방식들은 부분 철회가 수행되는 버퍼의 수에 따라 단일 버퍼 기반 부분 철회 방식과 이중 버퍼 기반 부분 철회 방식으로 분류되며, 부분 철회 전략에 따라 다시 네가지 방식으로 세분화된다. 이들 부분 철회 방식의 기본 개념, 동작 방식 및 장단점을 제시한다.

4.1 단일 버퍼 기반 부분 철회 방식

단일 버퍼 기반 부분 철회 방식은 부분 철회를 위한 회복 데이타의 관리 및 회복 과정이 페이지 버퍼나 객체 버퍼 중 한쪽에서 수행된다. 단일 버퍼 기반 부분 철회 방식은 이러한 회복 과정이 수행되는 버퍼의 종류에 따라서 페이지 버퍼 기반 부분 철회 방식(page buffer based partial rollback method: PB)과 객체 버퍼 기반 부분 철회 방식(object buffer based partial rollback method: OB)으로 구분한다.

4.1.1 페이지 버퍼 기반 부분 철회 방식(PB)

PB는 부분철회를 위하여 세이브포인트 시점에 객체

버퍼의 갱신된 객체들을 페이지 버퍼로 강제적으로 반영시키고 부분 철회시 페이지 버퍼의 상태만을 회복하는 방식이다. 부분 철회시 객체 버퍼에 있는 객체들은 지정된 세이브포인트 이후에 변경되었을 가능성이 있으므로 모두 삭제된다. 페이지 버퍼의 상태를 회복하기 위한 회복 데이타는 로그를 이용한다. 앞으로, 다른 부분 철회 방식에서 사용되는 소프트 로그(soft log)와 구별하기 위해 로그를 하드 로그(hard log)라고 호칭한다.

세이브포인트 설정 연산은 객체 버퍼 내에서 갱신된 객체들[1]을 페이지 버퍼에 강제적으로 반영하고, 해당 트랜잭션의 현재 실행 상태와 그 트랜잭션에 의해 기록된 마지막 로그 레코드의 LSN인 SaveLSN을 메모리에 기록한 후 해당 세이브포인트 식별자를 사용자에게 반환한다. 부분 철회 연산은 주어진 세이브포인트 식별자에 대응되는 SaveLSN 이후부터 해당 트랜잭션에 의해 기록된 로그 레코드들을 수행된 역순으로 UNDO하고 나서 트랜잭션의 실행 상태를 해당 SaveLSN과 함께 저장되었던 트랜잭션 실행 상태로 복구하고, 객체 버퍼에 있는 객체들을 삭제한다. 삭제된 객체들은 해당 객체에 대한 액세스 요청이 있을 때 객체 버퍼로 다시 읽어오게 된다.

PB는 제 2 절에서 기술한 객체 버퍼를 갖지 않는 RDBMS의 부분 철회 방식 위에 세이브포인트 설정 연산시 객체 버퍼의 플러시(flush)를 추가한 방식이다. 이 방식은 객체 버퍼에 대한 별도의 회복 데이타 없이 하드 로그에 의존하여 OODBMS에서의 부분 철회를 지원할 수 있으므로 그 구현이 단순하다는 장점을 갖는다. 그러나, 세이브포인트 설정 연산마다 객체 버퍼에 갱신된 상태로 있는 객체들을 모두 페이지 버퍼에 강제 반영시켜야 하는 오버헤드가 있다. 이 오버헤드는 객체 교체의 횟수를 증가시킬 뿐만 아니라 갱신된 객체들을 페이지 버퍼에 반영시키면서 페이지 버퍼의 교체를 야기하게 되어 디스크 입출력 횟수를 증가시킨다. 앞으로 이러한 오버헤드를 강제 플러시 오버헤드(forced flush overhead)라고 부른다. 특히, 저작과 같이 사용자와의 상호작용이 많은 응용에서 세이브포인트는 한 트랜잭션 안에서 사용자의 필요에 따라 빈번하게 설정될 수 있으므로 이러한 강제 플러시 오버헤드는 심각한 성능의 저하를 야기시킬 수 있다. 강제 플러시 오버헤드 뿐만 아니라 부분 철회시 객체 버퍼에 있었던 객체들은 모두 삭제되어 객체 버퍼의 버퍼링의 효과가 없어지게 되므로 부분 철회 이후에는 객체의 스왑-인이 더 많이 요구된다. 클라이언트-서버 구조인 경우 이러한 객체 교체의 비용은 더욱 증가하게 되므로 성능 저하의 문제는 더 심각해진다.

4.1.2 객체 버퍼 기반 부분 철회 방식(OB)

OB는 PB의 심각한 강제 플러시 오버헤드 및 객체 버퍼의 버퍼링 효과 감소 문제를 해결하기 위한 것으로서 객체 버퍼의 상태를 직접 회복하는 부분 철회 방식이다. OB에서는 이를 위해 객체 버퍼내에서 발생한 모든 갱신을 기록한 소프트 로그를 유지한다. 물론, 페이지 버퍼내에서 발생한 갱신을 기록한 하드 로그가 이와는 별도로 유지되지만 부분 철회를 위해서는 전혀 사용되지 않는다.

하드 로그는 파손 회복을 지원하기 위해서 반드시 안전한 디스크에 기록되어야 하지만 소프트 로그는 부분 철회를 위해서만 사용되어 트랜잭션의 진행 동안에만 유지되므로 디스크에 관리될 필요가 없고 부분 철회나 트랜잭션의 종료시 관련된 소프트 로그의 로그 레코드들이 손쉽게 삭제될 수 있도록 메모리에 유지한다. 또한, 부분 철회를 위해서만 사용되므로 소프트 로그 레코드들은 UNDO 정보만을 유지한다.

세이브포인트 설정 연산은 해당 트랜잭션의 현재 실행 상태와 소프트 로그의 마지막 로그 레코드의 LSN인 SaveLSN을 메모리에 기록하고 해당 세이브포인트 식별자를 반환한다. 부분 철회 연산은 주어진 세이브포인트 식별자에 대응되는 SaveLSN 이후부터 해당 트랜잭션에 의해 기록된 소프트 로그의 로그 레코드들을 수행된 역순으로 UNDO하고 나서 트랜잭션의 실행 상태를 해당 SaveLSN과 함께 저장되었던 트랜잭션 실행 상태로 복구한다. 이 때, 객체 버퍼에 있는 객체들은 바로 UNDO될 수 있지만 스왑-아웃되어 있는 객체들을 UNDO하기 위해서는 먼저 페이지 버퍼로부터 해당 객체를 스왑-인해야 한다. 또한, UNDO가 완료된 로그 레코드들은 소프트 로그에서 삭제된다. 부분 철회시 페이지 버퍼에서는 하드 로그의 UNDO가 수행되지 않으며 객체 버퍼에서의 UNDO로 인해 스왑-아웃된 객체로 인한 갱신은 하드 로그에 일반 갱신 연산으로 기록된다.

OB는 PB의 오버헤드를 모두 해결하며, 객체 버퍼의 크기가 충분히 커서 부분 철회시 UNDO되는 객체들이 대부분 객체 버퍼에 있는 경우에는 좋은 성능을 기대할 수 있다. 그러나, 부분 철회시 UNDO된 로그 레코드를 삭제함에도 불구하고 갱신 연산의 수가 많은 경우

소프트 로그를 유지하기 위한 메모리 오버헤드가 매우 심각하다는 단점이 있다. 뿐만 아니라, 부분 철회시

1) 객체 버퍼에서 객체에 대한 수정 플래그(dirty flag)가 True인 객체들을 의미한다.

현재 객체 버퍼에 존재하지 않는 객체를 UNDO하기 위해서는 해당 객체를 스왑-인해야 하고 이로 인해 다른 객체가 스왑-아웃되어야 한다. 따라서, OB에서 객체 버퍼가 충분히 크지 않은 경우에는 부분 철회시 객체 버퍼로부터 교체되는 객체의 수가 증가하게 되고, 이 결과 페이지 버퍼에서의 교체 및 디스크 입출력의 횟수가 증가하게 될 뿐만 아니라 하드 로그도 증가하게 되어 성능이 저하된다.

4.2 이중 버퍼 기반 부분 철회 방식

단일 버퍼 기반 부분 철회 방식이 가지는 성능 저하의 근본적인 원인은 이중 버퍼에 걸쳐서 존재하는 갱신된 객체들을 세이브포인트 연산 시점이나 부분 철회 연산 시점에 한쪽의 버퍼로 옮겨 주어야 하는 데서 비롯된다. 이중 버퍼 기반 부분 철회 방식(dual buffer based partial rollback method:DB)은 이러한 오버헤드를 피하기 위해 이중 버퍼에 걸쳐서 존재하는 갱신된 객체들을 가능한 한 현재 있는 각각의 버퍼에서 관리하고 회복한다. 객체 버퍼에서의 회복을 위해서는 객체 버퍼내의 갱신을 기록한 회복 데이타를 유지하고, 페이지 버퍼에서의 회복을 위해서는 하드 로그를 이용한다.

DB에서 객체는 세이브포인트 시점과 부분 철회 시점에 걸친 갱신 여부에 따라서 그림 1과 같이 세가지 유형으로 나누고 회복 방식은 다음과 같다. 먼저, 유형 1의 객체는 세이브포인트 시점 이전에만 갱신된 경우로서, 부분 철회시 회복 과정은 필요없다. 단, 이 방식에서는 PB와 같이 세이브포인트 설정시 강제적으로 객체를 플러시하지 않기 때문에 세이브포인트 이전에 객체 버퍼에서 갱신된 객체가 버퍼 교체 알고리즘에 의해 세이브포인트 이후에 페이지 버퍼에 반영될 수 있다. 따라서, 이러한 객체의 갱신은 세이브포인트 이전에 수행된 것이므로 부분 철회시 UNDO되지 않도록 한다. 이러한 객체를 지연객체라 부른다. 이를 위하여 유형 1의 객체가 스왑-아웃되고 이에 대한 갱신이 하드 로그 상에 기록될 때 관련된 세이브포인트의 saveLSN을 로그 레코드에 함께 기록하고, 부분 철회시 목적하는 세이브포인트의 saveLSN과 동일한 saveLSN을 기록한 로그 레코드는 UNDO하지 않도록 한다. 이로 인해 DB 방식은 로그 레코드에 saveLSN값을 유지하는 추가적인 로깅 오버헤드과 두개의 LSN에 대한 비교 연산 정도의 미미한 성능상의 오버헤드를 가진다.

유형 2의 객체는 세이브포인트 시점 이후에만 갱신된 경우로서, 부분 철회 시점에 객체 버퍼에 그대로 존재하는 경우와 페이지 버퍼로 스왑아웃된 경우가 다르게 처리된다. 전자의 경우에는 갱신이 객체 버퍼에만 반영되어 있으므로 객체 버퍼에서 해당 객체를 단순히 삭제하는 반면, 후자의 경우에는 스왑-아웃되어 페이지 버퍼에 반영될 때 그 갱신을 기록한 하드 로그의 로그 레코드를 UNDO함으로써 이전의 값을 복원한다. 유형 3의 객체는 세이브포인트 전후에 각각 갱신이 있었던 경우로서, 이 경우에는 객체 버퍼의 회복 데이타를 이용하여 회복을 수행한다.

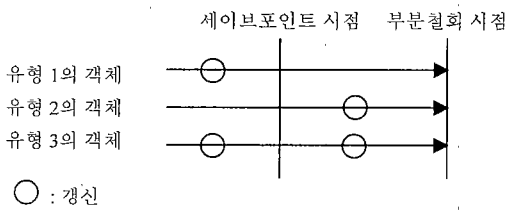


그림 1 객체의 유형

DB의 회복 방식은 객체의 유형 별로 PB와 OB의 방식을 결합한 방식이다. 유형 2의 객체는 PB에서와 동일한 방식으로 회복이 수행되고 유형 3의 객체는 OB에서와 같이 회복 데이타를 유지하여 회복이 수행됨을 알 수 있다.

임의의 세이브포인트에 대한 객체의 유형을 구분하기 위해 세이브포인트가 설정되면 이 시점에 객체 버퍼에 갱신된 상태로 있었던 객체들의 리스트인 세이브포인트 변경 객체 리스트(savepoint updated object list: SUOL)를 유지한다. 어떤 세이브포인트가 설정된 이후 해당 SUOL에 속한 임의의 객체가 객체 버퍼에서 더 이상 갱신되지 않았다면 이 객체는 이 세이브포인트에 대해 유형 1이 된다. 이 SUOL에 속한 객체가 스왑-아웃되지 않고 갱신된다면 유형 3이 되고 이 SUOL에 속하지 않는 객체가 갱신된다면 유형 2의 객체가 된다.

객체 버퍼에서 관리하는 회복 데이타에 따라서 소프트 로그를 이용한 이중 버퍼 기반 부분 철회 방식(dual buffer based partial rollback method using soft log:DB-SL)과 섀도우를 이용한 이중 버퍼 기반 부분 철회 방식(dual buffer based partial rollback method using shadows:DB-SO)이라고 부른다. 앞으로 이 두가지 방식을 자세히 살펴 본다.

4.2.1 소프트 로그를 이용한 이중 버퍼 기반 부분 철회 방식(DB-SL)

DB-SL은 주어진 세이브포인트에 대해 기록된 SUOL에 속한 객체에 대해서 소프트 로그를 이용하여 부분 철회를 수행하고 그 이외의 객체에 대해서는 객체 버퍼에 있는 객체를 삭제하거나 하드 로그를 이용하여

부분 철회를 수행하는 방식이다. 세이브포인트 설정 연산은 SUOL과 트랜잭션의 현재 실행 상태와 해당 트랜잭션에 대한 하드 로그상의 마지막 로그 레코드의 LSN인 HardSaveLSN, 그리고 소프트 로그의 마지막 로그 레코드의 LSN인 SoftSaveLSN을 메모리에 기록하고 해당 세이브포인트 식별자를 반환한다. 임의의 SUOL에 속한 객체가 갱신되면 해당 갱신을 기록한 로그 레코드를 소프트 로그에 추가한다. 부분 철회 연산은 객체 버퍼에서 주어진 세이브포인트 식별자에 대한 SUOL에 속하지 않는 객체들을 삭제하고, 소프트 로그에서 대응되는 SoftSaveLSN까지의 로그 레코드들 중 해당 SUOL에 속한 객체들에 대한 로그 레코드들을 UNDO하고, 대응되는 HardSaveLSN까지의 하드 로그 레코드를 UNDO한[2] 후, 저장되었던 트랜잭션의 실행 상태를 복구한다. 소프트 로그 레코드의 UNDO시 객체 버퍼에 있는 객체들은 그대로 UNDO될 수 있지만 스왑-아웃되어 있는 객체들을 UNDO하기 위해서는 먼저 페이지 버퍼로부터 해당 객체를 스왑-인해야 한다.

DB-SL은 OB와 같이 소프트 로그로 인한 오버헤드를 가지지만 하드 로그를 이용한 회복을 병행하기 때문에 다음과 같은 큰 차이가 있다. 첫째, DB-SL에서는 SUOL에 속한 객체들에 대해서만 소프트 로그를 유지하므로 이 방식의 메모리 오버헤드가 OB에 비해서 매우 작다.

둘째, DB-SL에서는 소프트 로그에 의해 UNDO되는 객체의 수가 작으므로 이를 위해 객체 버퍼에 없는 객체들을 페이지 버퍼로부터 스왑-인하는 오버헤드도 OB에 비해서 상당히 작다. 그러나, DB-SL은 여전히 소프트 로그를 유지하기 위한 메모리 오버헤드를 가지며, 부분 철회시 UNDO되는 객체를 위하여 페이지 버퍼로부터 객체 버퍼로의 객체 교체가 요구된다.

4.2.2 섀도우를 이용한 이중 버퍼 기반 부분 철회 방식(DB-SO)

DB-SO에서는 DB-SL의 부분 철회시 객체의 교체를 피하기 위해서 소프트 로그 대신에 섀도우들을 이용하여 부분 철회를 수행한다. 섀도우는 어떤 시점에 한 객체의 사본(copy)이다. 세이브포인트 설정 연산은 SUOL과 트랜잭션의 현재 실행 상태와 HardSaveLSN를 메모리에 기록하고 해당 세이브포인트 식별자를 반환한다. 임의의 SUOL에 속한 객체가 최초로 갱신될 때 이 객체에 대한 섀도우를 메모리에 생성한다. 부분 철회 연산은 객체 버퍼에서 세이브포인트 이후에 기록된 SUOL들에 속하지 않는 객체들을 삭제하고 세이브포인트 이후에 만들어진 섀도우로 객체 버퍼내의 현재의 객체를 대체하고 HardSaveLSN까지의 하드 로그 레코드를 UNDO한[3] 후, 저장되었던 트랜잭션의 실행 상태를 복구한다.

DB-SO에서는 DB-SL과 비교하면 다음과 같은 세가지 장점을 가진다. 첫째, DB-SO에서는 부분 철회시 SUOL에 속한 객체를 복원하고자 할 때 비록 해당 객체가 객체 버퍼에 존재하지 않더라도 메모리에 있는 섀도우를 그대로 객체 버퍼에 복사하기 때문에 페이지 버퍼로부터 다시 해당 객체를 스왑-인하는 과정이 필요하지 않다. 둘째, DB-SL에서는 SUOL에 속한 객체에 대한 다수의 갱신을 모두 소프트 로그에 기록하는 반면, DB-SO에서는 첫 갱신에 대해서만 섀도우를 유지하므로 DB-SL에 비해 적은 메모리 오버헤드를 가진다. 세째, DB-SL의 메모리 오버헤드는 SUOL에 속한 객체들의 갱신의 수에 따라 매우 가변적이고 예측할 수 없는 반면, DB-SO의 메모리 오버헤드는 최악의 경우 각 SUOL에 속한 객체들의 합계만큼의 섀도우들을 가짐으로 항상 예측이 가능하다.

4.3 네가지 부분철회 방식의 적용 예제

본 절에서는 예제를 통하여 네가지 부분철회 방식의 회복 방식을 기술한다. DBMS에서 그림 2와 같은 객체들이 존재하는 경우에 각 방식에서 세이브포인트 시점 이후에 갱신된 객체를 어떻게 회복하는 지를 차례대로 기술한다.

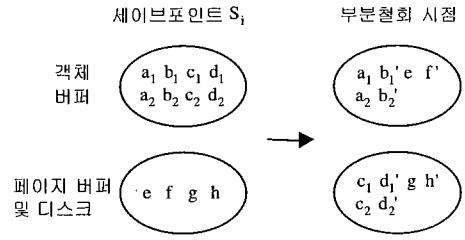


그림 2 예제 임의의 데이타 상태

그림 2는 세이브포인트 시점과 부분철회 시점에 객체 버퍼와 페이지 버퍼 및 디스크에 존재하는 객체들을 보여 준다. 물론, 객체 버퍼에 존재하는 객체인 경우 페이지 버퍼나 디스크에도 같은 객체가 존재하지만 객체 버퍼에 존재하는 객체의 상태가 그 객체의 상태를 대표하므로 객체 버퍼에만 표시하도록 한다. 그림 2에서 이름에 첨자 2를 포함한 객체는 세이브포인트 시점에 이

2) 단, 지연 객체들은 UNDO되지 않는다.

3) 단, 지연 객체들은 UNDO되지 않는다.

미 갱신된 상태이나 페이지 버퍼에는 반영되지 않은 객체를 의미하고 '(quote)를 포함한 객체는 세이브포인트 이후의 갱신된 객체를 의미한다. 즉, b_2'는 세이브포인트 이전에 갱신된 상태로 객체 버퍼에 있다가 세이브포인트 이후에 다시 갱신된 객체를 의미한다.

PB의 경우 세이브포인트 시점에 $a_2,...,d_2$를 페이지 버퍼로 플러시하기 때문에 이 객체들은 $a_1,...,d_1$과 동일한 방식으로 처리된다. 부분철회 시점에 객체 버퍼에 있는 객체들은 모두 삭제되고 페이지 버퍼에서 d_1, d_2, h가 하드 로그에 의해 UNDO되어 이전의 상태를 회복한다.

OB의 경우 객체버퍼에서 세이브포인트 이후에 있었던 갱신이 모두 소프트 로그로 기록되고 부분철회 시 UNDO된다. b_1, b_2, f는 객체 버퍼에 있으므로 바로 UNDO될 수 있지만 d_1, d_2, h는 그렇지 않으므로 객체 버퍼로 다시 스왑-인된 후 UNDO된다.

DB의 경우 객체 버퍼에서 SUOL에 속한 객체들 중에서도 세이브포인트 이후에 갱신이 있는 b_2, d_2에 대해서만 회복 정보를 기록한다. 부분철회 시 페이지 버퍼에서는 하드 로그에 의해 d_1, d_2, h가 UNDO되지만 c_2는 지연 객체이므로 UNDO되지 않는다. 객체 버퍼에서는 b_1, f가 삭제될 뿐만 아니라 SUOL에 속한 b_2, d_2에 대해서는 UNDO가 수행된다. DB-SL에서는 객체 버퍼에 없는 d_2를 UNDO하기 위해 먼저 객체 버퍼로 읽어 들여야 하지만 DB-SO에서는 이러한 과정 없이 d_2를 갱신하기 전에 유지했던 새도우를 객체 버퍼로 올리는 것으로 회복을 완료한다.

5. 부분 철회 방식들의 성능 평가

본 절에서는 제안된 네가지 부분 철회 방식들의 성능 평가를 위한 실험 환경들과 그 결과를 기술한다. 제 5.1 절에서는 평가하고자하는 성능 지수를 기술하고 제 5.2 절에서는 성능 평가를 위한 실험 모델을 기술하고 제 5.3 절에서는 각 부분 철회 방식들의 성능을 분석과 실험에 의해 평가한다.

5.1 성능 지수

성능 평가에서 다음의 세가지 성능 지수를 비교 평가한다.

◎ *nSwaps*는 객체 버퍼와 페이지 버퍼 사이에 전달된 객체의 수로서 각 방식에서의 객체 교체를 위한 비용을 분석하기 위한 것이다.

◎ *nDiskIOs*는 페이지 버퍼의 페이지 교체로 인한 디스크 입출력 횟수와 하드 로그를 액세스하기 위한 로그 디스크 입출력 횟수를 더한 값으로서 각 방식에서의 총 디스크 입출력 비용을 분석하기 위한 것이다.

◎ *MemoryOverhead*는 소프트 로그 및 새도우 등의 자료구조의 유지를 위한 저장 공간의 크기로서 각 방식에서 객체 버퍼의 회복을 위해 메모리에 별도로 관리하는 자료구조로 인한 공간 오버헤드를 비교하기 위한 것이다.

5.2 실험 모델

본 절에서는 부분 철회 방식들의 성능 평가를 위하여 본 논문에서 사용한 시스템 모델과 부하(work load) 모델에 대해서 기술한다. 사용된 시스템 모델은 Franklin 등[12]의 실험 환경을 참조하여 본 실험 목적에 맞도록 조정한 것이다.

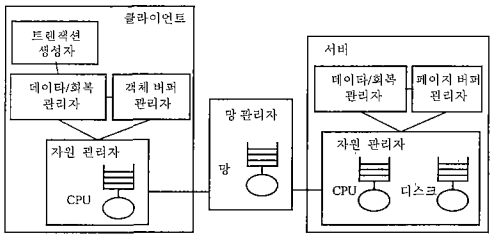


그림 3 시스템 모델

시스템 모델은 그림 3과 같이 디스크가 없는 한 클라이언트와 디스크를 가진 한 서버가 하나의 망을 통해 연결된 구조를 갖는다. 클라이언트는 객체 버퍼 관리자, 데이타/회복 관리자, 트랜잭션 생성자, 및 자원 관리자로 구성된다. 객체 버퍼 관리자는 객체 교체를 위해 LRU 정책을 사용하고 데이타/회복 관리자는 부분 철회 방식에 따라서 소프트 로그나 새도우 등을 관리하고 객체 버퍼에서의 트랜잭션의 부분 철회 및 완료 연산을 수행한다. 트랜잭션 생성자는 뒤에서 설명되는 부하 모델에 따라서 트랜잭션을 생성한다. 자원 관리자는 CPU 서비스와 망 액세스를 제공한다. 서버는 페이지 버퍼 관리자, 데이타/회복 관리자 및 자원 관리자로 구성된다. 페이지 버퍼 관리자는 페이지 교체를 위해 LRU 정책을 사용하고, 데이타/회복 관리자는 사용하는 부분 철회 방식에 따라 하드 로그를 관리하고 트랜잭션 부분 철회 및 완료를 수행한다. 자원 관리자는 CPU 서비스와 디스크/망 액세스를 제공한다.

시스템을 구성하는 매개변수들은 표 1과 같다. 한 페이지의 크기는 4K byte, 한 객체의 크기는 200 byte로 설정하였고 한 페이지 안에 기록되는 객체의 수는 20개로 가정하였다. 소프트 로그의 한 로그 레코드는 UNDO 정보만을 기록하므로 그 크기가 객체의 크기와 같다고 설정하였다. 또한, 하드 로그의 한 로그 레코드

는 UNDO 정보와 REDO 정보를 모두 기록하므로 객체의 두배 크기로 설정하였고 이러한 로그 레코드를 읽고 쓰기 위한 로그 버퍼의 크기는 한 페이지로 설정하였다. 데이타베이스의 크기는 약 3,000,000개의 객체를 포함하는 600 MB로 설정하였다.

표 1 시스템 매개변수

한 페이지의 크기	4096 bytes
한 객체의 크기	200 bytes
하드 로그의 한 로그 레코드의 크기	객체의 크기 * 2
소프트 로그의 한 로그 레코드의 크기	객체의 크기
로그 버퍼의 크기	1 페이지
데이타베이스의 크기	600 MB(3,000,000여개의 객체)
페이지 버퍼의 크기(N_{PB})	128 페이지(512 KB)
객체 버퍼의 크기(N_{OB})	320, 640, ..., 5440

트랜잭션의 액세스 구조는 그림 4와 같이 표현된다. m_1은 트랜잭션의 시작부터 첫번째 세이브포인트 시점까지의 객체 액세스 횟수이고 m_2는 첫번째 세이브포인트부터 부분 철회 시점까지의 객체 액세스 횟수이고 m_3는 부분 철회 시점부터 완료 연산의 수행 시점까지 객체 액세스 횟수이다. 연속되는 두 세이브포인트사이의 객체 액세스 횟수는 m_2/N_S로 동일하게 설정하였다. 항상 첫번째 세이브포인트로 철회하도록 설정한 것은 세이브포인트 설정 연산의 수행 횟수가 달라지더라도 항상 같은 양을 부분 철회하도록 함으로써 순수한 세이브포인트 설정 연산의 오버헤드를 분석하기 위한 것이다.

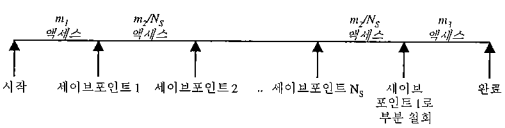


그림 4 한 트랜잭션에서의 액세스 구조

본 실험에서 사용한 시스템 부하를 조절하는 매개변수들은 표 2와 같다. 단일 클라이언트-서버 구조에서 수행되는 트랜잭션의 수는 성능 지수에 영향을 미치지 못하므로 트랜잭션의 수를 1로 고정하였다. 세이브포인트 설정 연산은 필요에 따라 여러번 수행될 수 있으나 부분 철회 연산은 상대적으로 자주 수행되지 않는다. 따라서, 부분 철회 연산의 실행 횟수는 1회로 고정하였다. 또한, 일반적으로 트랜잭션에서는 참조 연산이 갱신 연산보다 많이 실행되므로 본 실험에서는 갱신 연산의 비율을 전체의 0.3으로 설정하였다.

표 2 부하 모델의 매개변수들

트랜잭션의 수	1
세이브포인트의 수(N_S)	1, 3, 5, 7, 9
부분 철회의 수	1
갱신 연산의 비율(RW)	0.3
트랜잭션 당 객체 액세스 횟수(M)	10000 ($M = m_1+m_2+m_3$)
트랜잭션 당 액세스하는 객체의 수	3000, 10000
반복 액세스되는 객체의 액세스 순서	지수 분포(참조 평균:685.0, 갱신 평균:456.7)
트랜잭션의 액세스 구조	m_1 = 2000, m_2 = 5000, m_3 = 3000

다음의 세가지 각도에서 실험이 수행되었다. 첫째, 이중 버퍼 구조에서 객체 버퍼의 크기가 성능에 미치는 영향을 파악하기 위해 객체 버퍼의 크기를 조절하는 실험을 수행하였다. 객체 버퍼의 크기는 객체 버퍼에 유지되는 객체의 수로 정의되고 320[4]부터 5440[5]까지 320씩 증가된다. 둘째, 세이브포인트 설정 연산의 실행 횟수가 성능에 미치는 영향을 파악하기 위해 세이브포인트의 수를 조절하는 실험을 수행하였다. 세이브포인트의 수는 1부터 9까지 둘씩 증가된다. 세째, 객체 버퍼에서의 적중율(hit ratio)이 성능에 미치는 영향을 파악하기 위하여 적중율이 0인 반복 액세스되는 객체가 전혀 없는 경우와 적중율이 0이상인 반복 액세스되는 객체가 있는 경우로 구분하여 실험하였다. 먼저, 반복 액세스되는 객체가 전혀 없는 경우로서 트랜잭션 당 액세스 횟수가 10000이고 액세스되는 객체의 수가 10000개인 경우를 실험한다. 또한, 반복 액세스되는 객체가 있는 경우로서 트랜잭션 당 액세스 횟수는 10000이고 액세스되는 객체의 수가 3000인 경우를 실험한다. 일반적으로 DBMS에서의 객체 액세스는 일부의 객체가 빈번하게 액세스되고 나머지 객체는 상대적으로 적게 액세스되는 경향을 가지는 것으로 가정된다[12]. 본 실험에서는 이러한 경향을 지수분포로 모델링하였다. 3000개 이상의 서로 다른 객체를 가진 배열(array)을 생성하고, 주어진 참조 및 갱신 평균에 대한 지수 분포 함수를 통해 1에서 3000 사이의 값을 10000개 구하고, 이 값들을 배열의 인덱스로 하여 객체 액세스 순서를 생성하였다.

5.3 분석 및 평가

본 절에서는 실험 결과를 통해 네가지 부분 철회 방식의 성능을 비교 및 평가한다. 참고로, Appendix에는

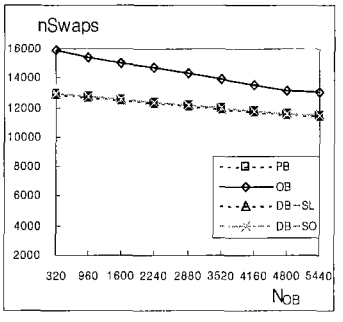
4) 본 실험에서의 어떠한 m_2/N_S보다 작은 값
5) 본 실험에서의 m_2보다 큰 값

반복 액세스되는 객체가 없는 경우 네가지 방식의 성능 지수를 수식으로 분석하여 실험 결과와 일치함을 보였다.

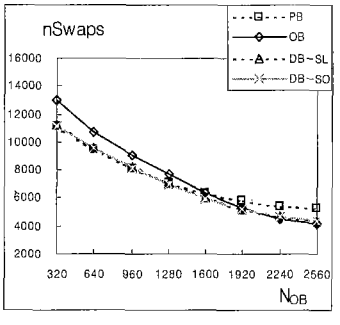
5.3.1 실험 및 결과

실험 1에서는 객체 버퍼의 크기에 따른 네가지 방식의 성능을 비교하고 실험 2에서는 세이브포인트의 수에 따른 성능을 비교한다. 실험 3에서는 객체 버퍼의 크기와 세이브포인트의 수에 따른 메모리 오버헤드를 비교한다. 각 실험을 반복 액세스되는 객체가 없는 경우와 있는 경우에 대해서 각각 수행한다. 각 방식의 성능 지수를 간단히 각 방식의 약칭 밑에 아래 첨자로 표현한다. 예를 들어, PB의 *nSwaps*은 $nSwaps_{PB}$으로 표현한다.

실험 1 세이브포인트의 수(N_S)가 1인 경우 객체 버퍼의 크기에 따른 네가지 부분 철회 방식들의 성능을 살펴본다. 그림 5는 객체 버퍼의 크기에 따른 네 방식의 *nSwaps*를 보인 것이다. 그림 5(a)에 의하면 반복 액세스되는 객체가 없는 경우 객체 버퍼의 크기가 증가함에 따라 각 방식의 *nSwaps*값은 다소 감소하지만, PB와 DB는 동일한 값을 가지는 반면, OB는 다른 방식에 비해 더 큰 값을 가지는 것으로 나타났다. 이는 다른 방식에서는 부분 철회시 객체 버퍼에 남아 있는 객체를 삭제하는 데 반하여 OB에서는 부분 철회시 객체 버퍼에 없는 객체를 UNDO하기 위해 다시 스왑-인하고 이로 인해 다른 객체가 스왑-아웃될 뿐만 아니라 부분철회 이후에 UNDO된 객체가 스왑-아웃되기 때문이다.



(a) 반복 액세스되는 객체가 없는 경우



(b) 반복 액세스되는 객체가 있는 경우

그림 5 N_S = 1일때의 *nSwaps*

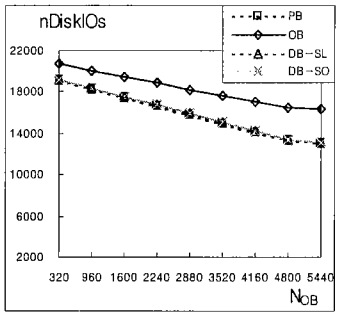
그림 5(b)에 의하면 반복 액세스되는 객체가 있는 경우에는 객체 버퍼의 크기가 증가함에 따라 각 방식의 성능상의 우열이 달라짐을 알 수 있다[6]. 그림 5(b)에서 N_{OB}가 960인 지점을 경계로 $nSwaps_{PB}$가 $nSwaps_{DB}$보다 높은 값을 가질 뿐만 아니라 1600인 지점을 경계로 $nSwaps_{OB}$보다 더 높은 값을 가지게 된다. 이는 반복 액세스되는 객체가 있는 경우 객체 버퍼가 커지면 OB와 DB는 객체 버퍼의 적중율이 증가하여 객체의 교체 수가 줄어드는 반면, PB는 강제 플러시로 인해 한번 이상 페이지 버퍼에 반영되는 객체의 수가 늘어나기 때문이다. 그림 5(b)에는 잘 나타나지 않지만 DB-SL은 DB-SO보다 조금 큰 *nSwaps*을 갖는다. 이는 DB-SL이 부분 철회시 UNDO를 위해 현재 객체 버퍼에 없는 객체를 다시 스왑-인 하는 경우가 발생하기 때문이다[7].

그림 6는 객체 버퍼의 크기가 증가함에 따라 네가지 방식의 *nDiskIOs*가 *nSwaps*와 유사한 경향임을 보여준다. OB 이외의 방식들은 객체 버퍼의 크기가 커짐에 따라서 *nSwaps*보다 더 급격히 *nDiskIOs*가 감소하는데 이는 교체되는 객체의 수의 감소로 인해 디스크 입출력의 수가 감소될 뿐만 아니라 부분 철회시 페이지 버퍼에서 UNDO되는 객체의 수가 감소하기 때문이다. 반면, OB는 부분철회시 하드 로그를 UNDO하지 않기 때문에 이러한 *nDiskIOs*의 감소폭은 상대적으로 떨어지지만 이로 인해 객체 버퍼의 크기가 작은 경우 *nSwaps*에 비해 더 작은 *nDiskIOs* 값을 가진다. 특히, 반복 액세스되는 객체가 있는 경우 $nDiskIOs_{OB}$는 DB와 유사한 성능을 보인다 .

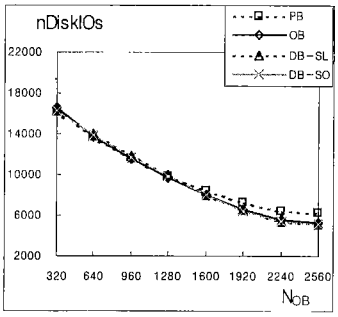
이 실험에 의하면 세이브포인트의 수가 1인 경우 객

6) 객체 버퍼의 크기가 2560까지만 보이면 것은 반복 액세스되는 객체가 있는 경우 객체 버퍼의 크기가 2560이 되면 부분 철회 시점까지 액세스되는 객체가 모두 객체 버퍼에 있게 되므로 객체 버퍼가 증가하더라도 성능은 증가하지 않기 때문이다.

7) 반복 액세스되는 객체가 없는 경우에는 부분 철회시 객체 버퍼에서 UNDO되는 객체가 없기 때문에 DB-SL과 DB-SO가 동일한 성능을 보인다. 따라서, 그림 5(a)에서 $nSwaps_{DB-SL}$과 $nSwaps_{DB-SO}$가 동일하게 나타난다.

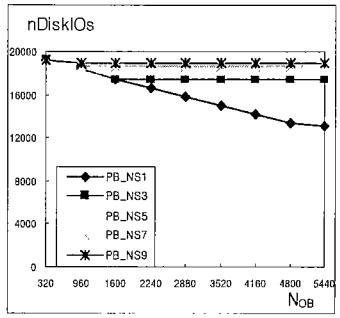


(a) 반복 액세스되는 객체가 없는 경우

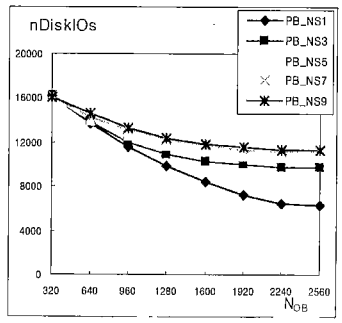


(b) 반복 액세스되는 객체가 있는 경우

그림 6 N_S = 1일때의 *nDiskIOs*



(a) 반복 액세스되는 객체가 없는 경우



(b) 반복 액세스되는 객체가 있는 경우

그림 7 N_S = 1,3,5,7,9 일때의 $nDiskIOs_{PB}$

체 버퍼의 크기가 증가함에 따라 네가지 방식의 성능은 모두 향상되지만 다음과 같이 네가지 방식의 상대적인 성능을 고찰할 수 있다. PB는 반복 액세스되는 객체가 있는 경우 객체 버퍼의 크기가 커지면 강제 플러시 오버헤드로 인하여 다른 방식에 비하면 성능이 떨어지고, OB는 객체 버퍼의 크기가 상당히 크고 반복 액세스되는 객체가 있는 경우 객체 버퍼에서의 적중률이 증가하여 DB 방식과 유사한 성능을 보이는 반면, DB는 객체 버퍼의 크기뿐만 아니라 반복 액세스되는 객체의 존재와 무관하게 상대적으로 좋은 성능을 보임을 알 수 있다.

실험 2 세이브포인트의 수를 변화시켰을 때 객체 버퍼의 크기에 따른 네 방식의 성능을 살펴본다. PB는 세이브포인트 연산시 강제 플러시를 수행하기 때문에 세이브포인트의 수에 따라 성능이 달라지는 반면, 다른 방식들은 세이브포인트 수에 무관한 성능을 보이기 때문에 PB의 경우만을 기술한다. 그림 7은 세이브포인트의 수와 객체 버퍼의 크기에 따른 $nDiskIOs_{PB}$를 나타내고 있다. 그림에서 세이브포인트의 수에 따른 성능의 변화를 나타내기 위해 세이브포인트의 수를 PB의 이름 뒤에 붙여서 표기한다. 예를 들어, *PB_NS1*은 세이브포인트의 수가 1인 경우 PB의 성능이다.

PB는 그림 7(a)에서 보이는 바와 같이 반복 액세스되는 객체가 없는 경우 세이브포인트의 수가 1을 초과하게 되면 객체 버퍼의 크기가 커지게 되더라도 성능이 향상되지 않는다. 그림에서 보면 N_S = 5일 때, N_{OB}가 960을 넘게 되면 *nDiskIOs*의 값이 18000정도에서 고정되고 있다. 이러한 성능의 고정은 다음과 같은 이유 때문이다. PB에서는 부분 철회시 객체 버퍼에 있는 객체를 삭제하는데 이러한 삭제로 인해 스왑-아웃되는 객체의 수가 줄어들게 되고 이로 인해 페이지 버퍼에서의 입출력과 기록되는 하드로그의 양도 줄어들게 된다. 이렇게 삭제되는 객체의 수는 마지막 세이브포인트 이후부터 부분 철회이전까지 액세스된 객체의 수와 객체 버퍼의 크기 중 작은 값에 갱신연산의 비율을 곱한 값 $R_W*min(m_2/N_S, N_{OB})$이다. 즉, 세이브포인트가 빈번하게 설정되면 객체 버퍼상의 갱신된 객체가 빈번하게 플러시되어 객체 버퍼의 크기가 커지더라도 삭제되는 객체의 수가 m_2/N_S로 고정되므로 성능이 향상되지 않는

다. 또한, 그림 7(b)에서 보이는 바와 같이 반복 액세스되는 객체가 있는 경우에도 세이브포인트의 수가 1을 초과하게 되면 객체 버퍼의 크기가 커지더라도 플러시되는 객체의 수가 늘어나고 이로 인해 디스크 입출력이 증가하게 되므로 $nDiskIOs_{PB}$의 감소폭이 상대적으로 둔화됨을 알 수 있다. 따라서, PB에서는 세이브포인트의 수가 증가하면 객체 버퍼의 크기가 증가하더라도 성능이 그리 향상되지 못한다.

실험 3 네가지 방식의 메모리 오버헤드를 살펴본다. 반복 액세스되는 객체가 없는 경우 OB는 소프트 로그를 유지하기 위한 메모리 오버헤드는 매우 큰 반면 다른 방식들의 메모리 오버헤드는 모두 0이다. 이는 PB에서는 본래 객체 버퍼에 대한 회복 정보를 별도로 유지하지 않고, DB에서는 반복 액세스되는 객체가 없는 경우 SUOL에 속한 객체들에 대한 더 이상의 갱신이 없으므로 그에 따라 회복 데이타도 유지되지 않기 때문이다.

반복 액세스되는 객체가 있는 경우 객체 버퍼의 크기와 세이브포인트 수에 따른 네가지 방식의 메모리 오버헤드는 그림 8과 같다. 이 그림에서 그래프의 가로축은 객체 버퍼의 크기이고 세로축은 새도우의 수나 소프트 로그의 로그 레코드의 수로 나타낸 메모리 오버헤드이다. DB-SL과 DB-SO는 세이브포인트의 수에 따라 다른 메모리 오버헤드를 갖기 때문에 이를 표현하기 위해 두 방식의 이름 뒤에 세이브포인트 수를 붙여서 기술하였다.

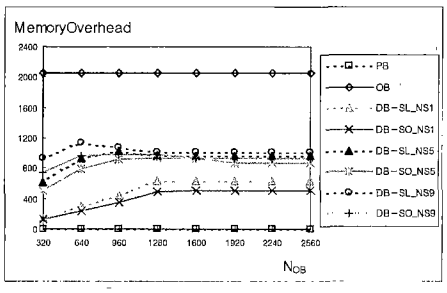


그림 8 반복 액세스되는 객체가 있고 N_S = 1,5,9인 경우 메모리 오버헤드

예상대로 PB의 메모리 오버헤드는 항상 0이고, OB의 메모리 오버헤드는 부분 철회 시점까지 갱신된 객체의 수로서 세이브포인트의 수나 버퍼의 크기에 영향을 받지 않고 항상 일정하다. 반면, DB-SL과 DB-SO에서는 그림 8에서와 같이 세이브포인트 수가 증가함에 따라서 그 메모리 오버헤드가 증가하나, 그 값이 OB의 메모리 오버헤드에 비하면 상당히 작다. 이는 세이브포인트의 수가 증가하면 세이브포인트 마다 유지되는 SUOL로 인해 관리되는 객체의 수가 증가하게 되나, 이러한 객체의 수는 부분 철회 시점까지 갱신되는 모든 객체의 수에 비하면 매우 작기 때문이다. 예상대로 DB-SL이 DB-SO에 비해 더 높은 메모리 오버헤드를 보였다. 이는 DB-SL은 SUOL에 속한 객체가 갱신될 때마다 회복 데이타를 기록하는 반면 DB-SO에서는 세이브포인트가 설정되고 나서 최초의 갱신에 대해서만 회복 데이타를 기록하기 때문이다.

실험의 결과를 다음과 같이 정리할 수 있다. PB는 세이브포인트의 수가 증가하면 플러시되는 객체의 수가 많아지고 이에 따라 기록되는 로그 레코드의 수 및 디스크 입출력 횟수가 증가하여 성능이 저하된다. 이러한 성능의 저하는 세이브포인트 설정 연산시의 강제 플러시 오버헤드로 인하여 발생한다. 또한, 이 방식은 이러한 강제 플러시 오버헤드로 인하여 객체 버퍼의 크기의 증가로 인한 성능 향상 효과는 미미하다.

OB는 세이브포인트의 수와 무관한 성능을 보이나 객체 버퍼의 크기와 반복 액세스되는 객체의 유무에 좌우되는 성능을 보인다. 객체 버퍼의 크기가 작은 경우에는 부분 철회시 현재 객체 버퍼에 없는 객체를 UNDO하기 위해 교체되는 객체의 수가 많아지고 이로 인해 디스크 입출력이 증가하여 성능이 떨어진다. 반면, 객체 버퍼의 크기가 커지면 이러한 오버헤드가 상당히 감소하고 또한 반복 액세스되는 객체가 많아지게 되면 객체 버퍼에서의 적중율이 높아지게 되므로 객체의 교체가 줄어들게 되어 좋은 성능을 보인다. 물론 OB는 소프트 로그를 유지하기 위해 심각한 메모리 오버헤드를 감수해야 한다.

DB는 OB와 마찬가지로 세이브포인트의 수와 무관한 성능을 보일 뿐만 아니라 객체 버퍼의 크기와 반복 액세스되는 객체의 유무에도 무관한 좋은 성능을 보인다. 또한, 이 방식은 OB에 비해서는 작은 메모리 오버헤드를 가진다. 비록 PB와 비교하면 DB가 어느 정도의 메모리 오버헤드를 가지지만 세이브포인트 설정 연산이 빈번하게 실행될 수 있음을 고려한다면 DB의 성능이 더 뛰어나다고 말할 수 있다. 두 방식중 DB-SO가 DB-SL에 비해 좋은 성능을 보인다. DB-SL은 SUOL에 속한 객체의 갱신마다 소프트 로그를 유지하고 부분 철회시 현재 객체 버퍼에 없는 객체를 UNDO하기 위해 스왑-인을 수행하는 반면 DB-SO는 해당 객체의 첫 갱신에 대해서만 새도우를 유지하고 부분 철회시 이러한

스왑-인이 필요하지 않기 때문이다.

6. 결 론

CAD나 멀티미디어 저작과 같은 OODBMS 응용은 사용자와의 상호 작용이 많이 있으나 이를 위해 필수적으로 지원해야 할 OODBMS에서의 부분 철회 기능에 대한 연구는 현재 거의 전무한 상태이다. OODBMS에서는 RDBMS와 달리 이중 버퍼 구조를 사용하기 때문에 트랜잭션에 의해 갱신된 데이타베이스가 페이지 버퍼 뿐만 아니라 객체 버퍼에 산재되어 있으므로 OODBMS에서의 부분 철회는 이중 버퍼에 산재되어 있는 데이타베이스의 상태를 사용자가 원하는 세이브포인트 시점으로 회복시켜주어야 한다.

본 논문에서는 OODBMS에서의 부분 철회를 지원하는 네가지 방식을 제안하였다. 제안된 부분 철회 방식은 부분 철회가 수행되는 버퍼의 수에 따라 단일 버퍼 기반 부분 철회 방식과 이중 버퍼 기반 부분 철회 방식으로 구분된다. 단일 버퍼 기반 부분 철회 방식에는 페이지 버퍼 기반 부분 철회 방식과 객체 버퍼 기반 부분 철회 방식이 있고 이중 버퍼 기반 부분 철회 방식에는 소프트 로그를 이용한 방식과 섀도우를 이용한 방식이 있다.

본 논문에서는 네가지 부분 철회 방식의 평가를 위해 수식 및 실험을 통한 분석을 수행하였다. Appendix에서는 반복 액세스되는 객체가 없는 경우에 대해서 수식에 의한 분석을 수행하였고 반복 액세스되는 객체가 없는 경우와 있는 경우에 대해서 실험을 통한 분석을 수행하였으며, 이러한 수식에 의한 분석 결과는 반복 액세스되는 객체가 없는 경우의 실험 결과와 99% 일치하였다. 이러한 분석 결과를 기반으로 네가지 부분 철회 방식의 성능을 요약하면 다음과 같다. 페이지 버퍼 기반 부분 철회 방식은 RDBMS에서 사용되었던 기존의 부분 철회 방식을 단순히 확장하여 구현할 수 있으므로 구현이 단순하다는 장점을 가지나 세이브포인트 연산이 빈번하게 수행되는 경우 강제 플러시 오버헤드로 인하여 성능이 떨어진다는 단점이 있다. 객체 버퍼 기반 부분 철회 방식은 객체 버퍼가 충분히 큰 경우 우수한 성능을 보이지만 회복용 데이타인 소프트 로그를 유지하기 위한 메모리 오버헤드가 심각하다는 단점이 있다. 이중 버퍼 기반 부분 철회 방식은 그 구현 방법이 복잡하지만 세이브포인트의 수와 객체 버퍼의 크기에 큰 영향을 받지 않고 항상 좋은 성능을 보인다. 특히, 섀도우를 이용한 방식은 소프트 로그를 이용한 방식과 비교하여 더 나은 성능을 보인다. 또한, 이중 버퍼 기반 부분 철회 방식의 메모리 오버헤드는 객체 버퍼 기반 부분 철회 방식에 비해 상당히 작다. 따라서, 이 논문에서는 섀도우를 이용한 이중 버퍼 기반 부분 철회 방식이 가장 좋은 방식이라고 결론짓는다.

부분 철회 기능은 사용자와 상호 작용이 많은 시스템에 있어서는 매우 필수적인 기능이다.따라서, 제안된 부분 철회 방식들과 각 방식들에 대한 분석 및 실험 결과는 OODBMS 뿐만 아니라 이중 버퍼를 사용하는 시스템의 설계 및 구현에 많은 도움이 될 수 있을 것이라 생각된다.

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] Mohan, C. et al., "ARIES: A Transaction Recovery Method Supporting Fine-Granularity Locking and Partial Rollback Using Write Ahead Logging," *ACM Trans. on Database Systems*, Vol. 17, No. 1, pp. 94--162, Mar. 1992.

[4] Gray, J. et al., "The Recovery Manager of the System R Database Manager," *ACM Computing Surveys*, Vol. 13, No. 2, pp. 223--242, June 1981.

[5] Oracle, Inc., *Oracle8 Server Concepts Relases 8.0*, Vol. 2, Part No. D009-350-9602-2, 1993.

[6] Sybase, Inc., *Sybase SQL Server Reference Manual*, Doc. 32401-01-1000-02, 1993.

[7] UniSQL, Inc., *UniSQL/X User's Manual*, Doc. D009-350-9602-2, 1996.

[8] Kemper, A. and Kossman, D., "Adaptable-Pointer Swizzing Strategies in Object Bases," In *Proc. Tenth Int'l Conf. on Data Engineering*, IEEE, pp. 155--162, Apr. 1993.

[9] Kim, W., *Introduction to Object-Oriented Databases, Computer System Series*, The MIT Press, 1st ed., 1990.

[10] Loomis, M.E.S., *Object Databases: The Essentials, Addison-Wesley*, 1995.

[11] Ozsu, M.T., Dayal, U., and Valduriez, P., *Distributed Object Managment*, Morgan Kaufmann, 1994.

[12] Franklin, M.J., Carey, M.J., and Livny, M., "Transactional Client-Server Cache Consistency: Alternatives and Performance," *ACM Trans. on Database Systems*, Vol. 22, No. 3, pp. 315--363, Sept. 1997.

APPENDIX 수식으로 유도된 성능 지수

주어진 환경에서 반복 액세스되는 객체가 없는 경우에 대하여 부분 철회 방식들의 성능 지수를 기술한다. 수식에서 사용되는 명칭 및 가정은 표 1, 2 와 그림 4를 따른다.

nSwaps 객체 버퍼의 객체 교체의 횟수 *nSwaps*는 스왑-인된 객체의 수 *nSwapIns*와 스왑-아웃되거나 플러시된 객체의 수 *nWriteOuts*를 더한 값이다.

$nSwaps_{PB} = M + R_W * (M - min(m_2/N_S, N_{OB}))$

$nSwaps_{OB} = M + R_W * (M + 2max(m_2-N_{OB}, 0))$

$nSwaps_{DB} = M + R_W * (M - min(m_2, N_{OB}))$

PB에서 $min(m_2/N_S, N_{OB})$은 NS번째 세이브포인트 이후에 액세스되어 객체 버퍼에 남아있다가 부분 철회시 삭제되는 객체의 수이다. OB에서 $2max(m_2-N_{OB}, 0)$는 부분 철회시 UNDO될 m_2개의 객체 중 현재 객체 버퍼에 있지 않은 객체들이 스왑-인되고 이로 인해 다른 객체들이 스왑-아웃되는 수이다. DB에서 $min(m_2, N_{OB})$은 첫번째 세이브포인트 이후 액세스되어 객체 버퍼에 남아있다가 부분 철회시 삭제되는 객체의 수이다.

nDiskIOs 디스크 입출력 횟수 *nDiskIOs*는 페이지 버퍼에 스왑-인되거나 페이지 버퍼로부터 스왑-아웃된 페이지를 위한 입출력의 수 *nPageIOs*, 부분 철회 시 페이지 버퍼에서 갱신된 객체를 UNDO를 위한 페이지 입출력의 수 *nUndoPageIOs*, 그리고 하드 로그를 위한 입출력의 수 *nLogPageIOs*로 구분된다.

*nPageIOs*는 스왑-인된 객체가 속한 페이지에 대한 디스크 입출력 횟수 nSwapIns[8]와 스왑-아웃되거나 플러시된 객체들이 속한 페이지들을 디스크로부터 읽고 쓴 횟수 *2*nWriteOuts*의 합에서 완료 후에도 페이지 버퍼에 남아있는 페이지들의 수 N_{PB}를 뺀 값이 된다. *nUndoPageIOs*는 주어진 세이브포인트부터 부분 철회 시점 동안에 스왑-아웃된 객체의 갱신을 UNDO하기 위해 디스크로부터 관련 페이지를 읽어 들이고 다시 기록하기 위한 디스크 입출력 횟수이다. *nLogPageIOs*는 로그 레코드들을 읽고 쓰기 위한 페이지 단위의 입출력 횟수로서, 한 페이지에 들어가는 로그 레코드의 수는 10이고 한 트랜잭션의 수행을 가정하므로 액세스되는 로그 레코드의 수를 10으로 나눈 값이다.

$nDiskIOs_{PB} = M + R_W *(2.1*(M+m_2 - 2min(m_2/N_S, N_{OB})))-N_{PB}$

$nDiskIOs_{OB} = M + R_W *(max(m_2-N_{OB}, 0)+2.1*(M+max(m_2-N_{OB}, 0)))-N_{PB}$

$nDiskIOs_{DB} = M + R_W *(2.1*(M+m_2- 2min(m_2, N_{OB}))+0.1min(m1, N_{OB}))-N_{PB}$

MemoryOverhead OB를 제외한 부분 철회 방식들의 메모리 오버헤드는 모두 0이다. 이는 PB에서는 본래 객체 버퍼에 대한 회복 정보를 별도로 유지하지 않고, DB에서는 반복 액세스되는 객체가 없는 경우 SUOL에 속한 객체들에 대한 더 이상의 갱신이 없으므로 그에 따라 회복 정보도 유지되지 않기 때문이다. 소프트 로그를 유지하기 위한 OB의 메모리 오버헤드는 부분 철회 이전까지 기록된 로그 레코드의 수와 부분 철회 후 트랜잭션 완료 이전까지 기록된 로그 레코드의 수중에서 최대값이 된다.

8) 여기서는 객체의 클러스터링 효과를 무시하여 액세스하는 객체들이 모두 서로 다른 페이지 내에 존재한다고 가정한다.

김 원 영

1989년 2월 이화여자대학교 전산학과 졸업(학사). 1991년 2월 한국과학기술원 전산학과 졸업(석사). 1998년 8월 한국과학기술원 전산학과 졸업(박사). 1998년 9월 ~ 1999년 2월 한국과학기술원 인공지능연구센터 Post Doc. 1999년 3월 ~ 현재 한국전자통신연구원 실시간 DBMS팀 선임연구원. 관심분야는 데이타베이스 관리 시스템, 분산 데이타베이스, 실시간 데이타베이스

이 영 구

1992년 과학기술대학 전산학과 졸업(B.S.). 1994년 한국과학기술원 전산학과 졸업(M.S.). 1994년 ~ 현재 한국과학기술원 전산학과 박사과정. 관심분야는 OLAP, 다중 키 액세스, 공간 데이타베이스

황 규 영

정보과학회논문지 : 데이타베이스
제 27 권 제 1 호 참조