# 데이타베이스 화일의 삭제를 위한 미처리 연산의 효율적 수행 기법

## (An Efficient Scheme of Performing Pending Actions for the Removal of Database Files)

박 준 현 † 박 영 철 ††

(Jun Hyun Park) (Young Chul Park)

**요 약** 본 논문은 데이타베이스 관리시스템이 데이타베이스를 저장할 디스크 공간을 직접 관리하는 환경에서 데이타베이스 화일들의 삭제를 위해 미처리 연산을 정확하고 효율적으로 수행하는 기법을 제시한다. 미처리 연산의 수행과 관련하여 회복시에 회복 프로세스는 완료가 결정되었으나 아직 종료하지 않은 미종료 트랜잭션들의 아직 수행되지 않은 미처리 연산들을 식별하고 이들을 완전히 수행할 수 있어야 한다. 본 논문의 기본 아이디어는 로그 화일에 기록된 로그레코드들을 분석함으로써 회복 프로세스가 그 연산들을 식별할 수 있게 하는 것이다. 이 기법은 ARIES의 트랜잭션, 퍼지 검사점, 그리고 회복의 수행을 확장한 것으로서 다음의 방법을 사용한다. 첫째, 회복시에 미종료 트랜잭션들을 식별하기 위하여, 각 트랜잭션은 트랜잭션의 완료와 미처리 연산 수행의 시작을 함께 나타내는 미처리연산수행시작('pa_start') 로그레코드를 기록한 후에 미처리 연산을 수행하며, 그리고 나서 트랜잭션종료 로그레코드를 기록한다. 둘째, 회복시에 미종료 트랜잭션들의 미처리 연산 리스트를 복구하기 위하여, 각 트랜잭션은 미처리연산수행시작 로그레코드에 그 트랜잭션의 미처리 연산 리스트를 수록하며 퍼지 검사점은 검사점종료 로그레코드에 완료가 결정된 트랜잭션들의 미처리 연산 리스트를 수록한다. 셋째, 회복시에 다음에 수행할 미처리 연산을 식별하기 위하여, 각 트랜잭션은 미처리 연산을 수행하면서 변경하는 각 페이지에 대하여 그 페이지에 대한 재수행 정보를 기록하는 로그레코드에 다음에 수행할 미처리 연산의 식별 정보를 함께 수록한다.

***Abstract*** In the environment that database management systems manage disk spaces for storing databases directly, this paper proposes a correct and efficient scheme of performing pending actions for the removal of database files. As for performing pending actions, upon performing recovery, the recovery process must identify unperformed pending actions of not-yet-terminated transactions and then perform those actions completely. Making the recovery process identify those actions through the analysis of log records in the log file is the basic idea of this paper. This scheme, as an extension of the execution of transactions, fuzzy checkpoint, and recovery of ARIES, uses the following methods: First, to identify not-yet-terminated transactions during recovery, transactions perform pending actions after writing 'pa_start' log records that signify both the commit of transactions and the start of executing pending actions, and then write 'end' log records. Second, to restore pending-actions-lists of not-yet-terminated transactions during recovery, each transaction records its pending-actions-list in 'pa_start' log record and the checkpoint process records pending-actions-lists of transactions that are decided to be committed in 'end_chkpt' log record. Third, to identify the next pending action to perform during recovery, whenever a page is updated during the execution of pending actions, transactions record the information that identifies the next pending action to perform in the log record that has the redo information against the page.

· 본 논문은 한국과학재단 목적기초연구(과제번호: 2000-2-51200-002-3) 지원으로 수행되었음.

† 비 회 원 : 한국컴퓨터통신(주) 연구소
jhpark@unisql.com

†† 종신회원 : 경북대학교 컴퓨터과학과 교수
ycpark@knu.ac.kr

논문접수 : 2000년 11월 2일
심사완료 : 2001년 6월 18일

## 1. 서 론

데이타베이스 관리시스템은 테이블 또는 색인의 생성이 요구되면 이들 각각에 대하여 하나의 데이타베이스 화일을 생성하며 테이블에 대한 화일을 테이타 화일이라 하고 색인에 대한 화일을 색인 화일이라 한다. 색인의 삭제 요구는 해당 색인 화일을 삭제하게 되고 테이블의 삭제 요구는 해당 데이타 화일과 그 테이블을 기반으로 생성된 모든 색인 화일들을 삭제하게 된다. 따라서, 테이블 또는 색인의 삭제 요구가 발생한 시점에 해당 화일(들)을 삭제하기 위한 모든 작업들을 수행한다면 많은 양의 데이타를 로깅(logging)해야 하며, 화일을 삭제한 트랜잭션이 복귀(rollback)할 경우에는 삭제한 화일을 회복(recovery)하여야 하므로 시스템에 많은 부담을 초래한다.

화일 삭제에 따른 시스템의 부담을 최소화하기 위하여 데이타베이스 관리시스템은 화일 삭제를 수행하기 위한 작업들 중 일부 작업들을 트랜잭션이 화일 삭제를 요구한 시점에 수행하는 연산인 즉시 연산(immediate action)으로 처리하며, 나머지 작업들을 그 트랜잭션의 완료가 결정된 후에 그리고 그 트랜잭션의 종료 이전에 수행하는 연산인 미처리 연산(pending action)[5, 9]으로 처리한다. 따라서, 완료가 결정되었으나 아직 종료하지 않은 트랜잭션 즉, 미종료(not-yet-terminated) 트랜잭션의 수행 중에 고장이 발생하더라도 그 트랜잭션의 아직 수행되지 않은 미처리 연산들은 회복시에 회복 프로세스에 의해 완전히 수행되어야 하며 그 트랜잭션은 정상 종료되어야 한다.

ARIES[9]는 데이타베이스 화일들을 운영체제 화일들로 관리하는 환경을 가정하였다. 이 환경에서는 운영체제의 화일 시스템이 데이타베이스 화일에 할당된 디스크 공간을 관리하므로 데이타베이스 화일에 할당된 디스크 공간의 반환은 그 데이타베이스 화일에 대응되는 운영체제 화일을 삭제하기 위한 운영체제 루틴(routine)을 호출하는 하나의 연산(single action)으로 수행된다. ARIES는 화일 삭제를 해당 트랜잭션의 완료가 결정되는 시점까지 연기한 후, 두단계 완료규약(two-phase commit protocol)에 따라 완료준비 단계에서 완료준비('prepare') 로그레코드에 미처리 연산들을 수록하여 로깅하며, 완료 단계에서 트랜잭션종료('end') 로그레코드를 기록하고 트랜잭션이 획득한 모든 로크(lock)들을 해제한 후 미처리 연산으로 삭제할 각 화일에 대하여 'OSfile_return' 로그레코드를 기록하고 그 화일이 차지한 공간을 운영체제에게 반환한다. 이러한 화일 삭제 방법에 의해 회복 프로세스는 완료준비 로그레코드에 수록된 삭제할 운영체제 화일들 중 삭제되지 않은 운영체제 화일들을 식별하여 삭제할 수 있다.

바다-II[12, 13, 14, 15, 18]를 비롯하여 바다-III[1, 19], UniSQL[21], Orion[8], Informix[7], MS SQL Server[10] 등 대부분 데이타베이스 관리시스템의 저장 시스템(storage system)과 Shore[3], EXODUS[2], WiSS[4] 등의 저장 시스템은 성능 향상을 위하여 물리적으로 연속한 디스크 공간인 원시 디바이스(raw device) 또는 논리적으로 연속한 디스크 공간인 운영체제 화일의 모임으로 데이타베이스 공간(database space)을 구성하며 그 데이타베이스 공간 내에서 자체적으로 데이타베이스 화일들을 생성, 관리, 삭제한다. 이러한 환경에서 데이타베이스 화일의 삭제는 다수의 페이지들에 변경을 유발하는 다수의 연산들로 수행되므로 ARIES의 화일 삭제 기법을 그대로 적용할 수는 없으며 화일 삭제의 정확성을 보장하고 그 연산의 수행 효율을 높이는 것은 간단하지 않다[9].

본 논문은 트랜잭션, 검사점 그리고 회복 프로세스 간에 로깅을 이용하여 미처리 연산을 인수인계하는 방법을 제안한다. 이 방법은 화일 삭제에 관한한 ARIES의 트랜잭션, 검사점 그리고 회복의 수행 기법을 데이타베이스 관리시스템이 디스크 공간을 직접 관리하는 환경으로 확장한 기법으로서 다음의 방법을 사용한다.

첫째, 각 트랜잭션은 완료가 결정되면 완료의 준비 작업으로 미처리 연산으로 삭제할 화일들의 화일 식별자들로써 미처리 연산 리스트(pending actions list: PAL)라 불리우는 하나의 리스트를 형성한다.

둘째, 트랜잭션의 완료 작업은 미처리 연산 리스트를 가진 경우에는 그 리스트를 미처리 연산수행시작('pa_start') 로그레코드에 수록하여 기록한 후에 미처리 연산을 수행하며, 그리고 나서 트랜잭션종료 로그레코드를 기록한다. 미처리연산수행시작 로그레코드는 트랜잭션의 완료와 미처리 연산 수행의 시작을 함께 나타낸다.

셋째, 각 트랜잭션은 미처리 연산을 수행하면서 변경하는 각 페이지에 대하여 그 페이지에 대한 재수행(redo) 정보를 기록하는 로그레코드에 다음에 수행할 미처리 연산의 식별 정보를 함께 수록한다.

넷째, 퍼지 검사점은 미종료 트랜잭션들의 미처리 연산 리스트를 검사점종료('end_chkpt') 로그레코드에 수록한다.

다섯째, 회복시에 회복 프로세스는 미종료 트랜잭션들을 식별하며 그 트랜잭션들의 아직 수행되지 않은 미처리 연산들을 파악하고 그 연산들의 수행을 완료하고 그

트랜잭션들을 정상 종료시킨다.

본 논문이 제안하는 화일 삭제 기법은 다음의 두가지 중요 특징을 가진다.

첫째, 화일 삭제시에 해당 데이타 화일에 대한 배타적 로크를 제외한 어떠한 로크도 요구하지 않으며 반환된 디스크 공간은 그 즉시 다른 화일에 할당될 수 있다. 따라서, 화일 삭제 시에 반환하는 디스크 공간에 대하여 별도의 로크를 요구하지 않으며 디스크 공간을 요구하는 트랜잭션의 경우에도 할당받는 디스크 공간에 대해 어떠한 로크도 요구하지 않는다.

둘째, 회복시에 회복 프로세스는 기록된 로그레코드들의 분석을 통해 미종료 트랜잭션들에 의해 삭제 중이거나 삭제되어야 하는 화일들의 미처리 연산 수행 상태를 식별하므로 이들 식별된 화일에 대하여 화일 삭제의 미완료된 부분만을 수행한다.

본 논문의 나머지 구성은 다음과 같다. 제 2절에서는 공개된 시스템인 Shore 1.1[3]의 분석을 통해 파악한 사항들을 바탕으로 Shore에서 채택하고 있는 화일 삭제를 위한 미처리 연산의 수행 기법을 제시하며 본 논문의 화일 삭제 기법이 기반으로 하는 바다-II의 데이타베이스 공간과 화일의 관리 구조를 기술한다. 제 3절에서는 화일 삭제를 위해 수행해야 하는 연산들을 분석하고 이들을 즉시 연산과 미처리 연산으로 분류하며, 화일 삭제와 관련하여 화일 정리 리스트와 미처리 연산 리스트를 소개한다. 제 4절에서는 화일 삭제와 관련하여 트랜잭션의 시작, 완료, 그리고 복귀에 관하여 기술하며, 트랜잭션에 의한 즉시 연산들과 미처리 연산들의 수행 알고리즘을 제시한다. 제 5절에서는 ARIES의 퍼지 검사점을 미처리 연산을 고려하여 확장한 사항에 대하여 기술한다. 제 6절에서는 화일 삭제를 위한 미처리 연산을 고려하여 ARIES의 재시작 회복과 매체 회복의 확장한 사항들을 기술한다. 제 7절에서는 본 논문에서 제시하는 미처리 연산의 수행 기법과 Shore의 기법을 비교한다. 마지막으로, 제 8절에서 결론을 맺는다.

## 2. 관련 연구

본 절은 관련 연구로써 Shore의 화일 삭제 기법을 분석하고 본 논문이 기반으로 하는 바다-II의 데이타베이스 공간과 화일의 관리 방법을 기술한다. 본 절은 Shore의 화일 삭제에 관련한 세부 사항들을 간략화하기 위하여 색인을 가지지 않는 데이타 화일의 삭제로 한정하며, 화일 삭제에서 회복에 관련된 연산들에 대해서만 다룬다.

### 2.1 Shore의 화일 관리 구조와 화일 삭제 기법

Shore에서 데이타베이스 공간은 하나 이상의 볼륨(volume)으로 구성되며 각 볼륨은 하나의 세그먼트(segment)만을 가진다. 하나의 세그먼트는 물리적 또는 논리적으로 연속한 디스크 공간으로서 익스텐트(extent)들로 나뉘어진다. 하나의 익스텐트는 세그먼트 내에서 연속한 페이지들의 모임이며 화일에 디스크 공간을 할당하거나 화일로부터 디스크 공간을 반환받는 단위이다. Shore에서 하나의 화일이 차지하는 공간은 하나의 볼륨에 국한된다.[1)]

각 세그먼트는 그 세그먼트에서 생성된 화일들에 할당된 디스크 공간과 자유 디스크 공간을 관리하기 위하여 화일 맵(File Map: FM)과 익스텐트 맵(Extent Map: EM)을 가진다. 각 세그먼트의 익스텐트 맵은 그 세그먼트에 존재하는 익스텐트들의 수 만큼의 엔트리들을 가지며 익스텐트 맵의 i 번째 엔트리인 EM[i]는 그 세그먼트의 i 번째 익스텐트의 상태를 나타낸다. 각 익스텐트 맵 엔트리는 그 익스텐트에 속한 페이지들의 사용 여부를 나타내는 비트맵인 페이지 맵을 가지며 해당 세그먼트에서 각 화일에 할당된 익스텐트들의 익스텐트 맵 엔트리들을 익스텐트 맵 엔트리 리스트라고 하는 이중 연결 리스트로 유지하기 위한 필드들을 가진다. 각 세그먼트의 화일 맵은 화일 맵 엔트리들을 가진다. 화일 맵의 i 번째 엔트리인 FM[i]는 화일 번호가 i인 화일에 대한 화일 맵 엔트리로서 그 화일의 익스텐트 맵 엔트리 리스트의 선두 엔트리의 번호를 값으로 가지는 필드 뿐만 아니라 그 화일의 삭제 상황을 표시하기 위한 하나의 필드를 가진다. 익스텐트 맵 엔트리들을 수용하는 페이지들을 익스텐트 맵 페이지라 하며 화일 맵 엔트리들을 수용하는 페이지들을 화일 맵 페이지라 한다.

Shore에서 각 트랜잭션은 시스템의 정상 운영시에 화일 삭제의 진행 상황을 화일 맵과 익스텐트 맵에 표시하고 이들 표시에 대한 로그레코드들을 기록하며 회복시에 회복 프로세스는 화일 맵과 익스텐트 맵에 표시된 정보를 바탕으로 수행중이거나 아직 수행되지 않은 미처리 연산들을 식별하고 이들의 수행을 완료한다. 이 방법의 상세 사항은 다음과 같다.

Shore는 트랜잭션의 수행 중 색인을 가지지 않는 데이타 화일의 삭제가 요구되면, 그 화일에 대해 배타적(exclusive: X) 모드의 로크를 획득하고, 그 화일의 화일 맵 엔트리에 "화일 삭제(t_deleting_store)" 표시를

1) Shore는 화일이란 용어 대신에 스토아(store)라는 용어를 사용한다. 본 논문은 화일이란 용어로 통일하여 사용한다.

하고, 그 화일의 화일 식별자를 그 트랜잭션의 미처리 연산 리스트에 수록한 후 그 화일의 화일 설명자(file descriptor)를 삭제한다.

전역 트랜잭션은 완료준비 요구에 대해 그 트랜잭션이 미처리 연산 리스트를 가지고 있다면 검사점 뮤텍스(checkpoint mutex)[2]를 잡은 상태에서 트랜잭션의 완료상태를 xct_prepared로 변경하고 미처리 연산 리스트를 수록한 완료준비('xct_prepare') 로그레코드를 강제쓰기 유형으로 기록한 후 검사점 뮤텍스를 해제한다.[3] 트랜잭션의 완료 요구에 대해, 미처리 연산 리스트를 가진 지역 트랜잭션 또는 전역 트랜잭션은 다음의 작업들을 차례로 수행한다.

1. 트랜잭션의 완료상태를 xct_committing으로 변경한 후 검사점 뮤텍스를 잡은 상태에서 트랜잭션의 완료상태를 xct_freeing_space로 설정하고 그 트랜잭션의 완료와 미처리 연산의 시작을 함께 나타내는 'xct_freeing_space' 로그레코드를 강제쓰기 유형으로 기록한다.

2. 미처리 연산 리스트에 등록된 각 화일에 대하여 그 화일의 화일 맵 엔트리에 "익스텐트 반환(t_store_freeing_exts)" 표시를 한 후, 그 화일의 익스텐트 맵 엔트리 리스트에 속한 모든 익스텐트 맵 엔트리들을 초기화한 후 그 화일의 화일 맵 엔트리를 초기화한다. 익스텐트 맵 엔트리의 초기화는 그 엔트리에 있는 필드들 중 익스텐트 맵 엔트리 리스트의 다음 익스텐트 맵 엔트리를 가리키는 필드만을 제외한 나머지 필드들을 초기화하며 반환한 익스텐트들에 대해 익스텐트 로크(extent lock)를 획득하여 그 익스텐트들이 다른 화일에 할당되지 않도록 한다. 익스텐트 맵 엔트리 리스트의 다음 익스텐트 맵 엔트리가 현재 익스텐트 맵 페이지가 아닌 다른 익스텐트 맵 페이지에 존재하거나 그 리스트의 끝인 경우, 그 익스텐트 맵 페이지에서의 변경분에 대한 재수행 정보만을 수록한 로그레코드를 기록한 후 현재 익스텐트 맵 페이지를 버퍼에서 탈착한다.

3. 익스텐트 로크를 포함하여 그 트랜잭션이 획득하고 있는 모든 로크들을 해제한다.

4. 검사점 뮤텍스를 잡은 상태에서 트랜잭션의 완료상태를 xct_ended로 설정하고 트랜잭션종료('xct_end') 로그레코드를 비강제쓰기 유형으로 기록한다.

Shore의 퍼지 검사점은 트랜잭션의 완료상태가 xct_prepared인 트랜잭션에 한하여 그 트랜잭션의 미처리 연산 리스트를 검사점종료 로그레코드에 수록한다. Shore의 재시작 회복은 ARIES의 재시작 회복에 아래의 두 사항들을 추가하였다.

1. 재시작 재수행 단계를 마친 후에 그리고 재시작 취소 단계를 수행하기 전에 모든 볼륨의 화일 맵에서 할당 상태에 있으며 "익스텐트 반환"으로 표시된 각 화일 맵 엔트리에 대하여 그 화일 맵 엔트리의 익스텐트 맵 엔트리 리스트에 속하면서 초기화되지 않은 익스텐트 맵 엔트리들을 초기화한 후 그 화일 맵 엔트리를 초기화한다.

2. 재시작 취소 단계를 마친 후에 모든 볼륨의 화일 맵에서 할당 상태에 있으며 "화일 삭제"로 표시된 각 화일 맵 엔트리에 대하여 그 화일 맵 엔트리의 익스텐트 맵 엔트리 리스트에 속한 모든 익스텐트 맵 엔트리들을 초기화한 후 그 화일 맵 엔트리를 초기화한다.

Shore는 이 과정에서 발생하는 모든 페이지 변경 연산에 대하여 redo-only 유형의 로그를 기록한다.

### 2.2 바다-II에서 데이타베이스 공간과 화일의 관리

바다-II에서 데이타베이스 공간은 하나이상의 볼륨으로 구성되며 각 볼륨은 Shore와는 달리 하나이상의 세그먼트로 구성된다. 바다-II에서 하나의 화일은 Shore와 같이 하나의 볼륨에 국한된다. 각 화일은 시스템 내에서 그 화일을 유일하게 식별하기 위하여 <그 화일이 생성된 볼륨의 볼륨 번호(volume number), 그 볼륨에서 그 화일을 유일하게 식별하는 화일 번호(file number)>의 쌍으로 구성된 화일 식별자를 가지며, 그 화일에 대한 제반 정보들의 모임인 화일 설명자를 가지며, 그 화일의 데이타를 저장하는 디스크 공간으로서 그 화일에 할당된 익스텐트들을 가진다. 색인 화일의 경우, 그 색인이 기반으로 하는 데이타 화일에서 그 색인을 유일하게 식별하는 색인 번호(index number)를 별도로 부여받는다. 색인 화일의 화일 설명자를 특별히 색인 설명자(index descriptor)라 한다.

각 볼륨은 그 볼륨의 특정 페이지인 볼륨 상태 페이지(volume status page)에 그 볼륨에서 최대로 생성 가능한 화일들의 수 만큼의 비트들로 구성된 화일 할당 테이블(file allocation table: FATable)을 가지며, 화일 할당 테이블의 각 비트는 그 비트의 인덱스를 화일 번호로 가지는 화일의 존재 여부를 나타낸다. 화일 할당

---

2) Shore는 트랜잭션 완료상태의 변경과 완료규약 관련 로그레코드의 기록이 하나의 작업으로서 퍼지 검사점의 수행과 배타적으로 수행되도록 하기 위하여 검사점 뮤텍스를 잡은 상태에서 트랜잭션의 완료상태를 변경하고 완료상태에 대한 로그레코드를 기록한다.

3) Shore에서 완료준비 로그레코드는 다섯개의 로그레코드들('xct_prepare_st', 'xct_prepare_lk', 'xct_prepare_alk', 'xct_prepare_stores', 'xct_prepare_fi')로 구성된다. 이들 중 트랜잭션의 미처리 연산 리스트를 수록하는 로그레코드는 'xct_prepare_stores' 로그레코드이다. 본 논문에서는 이들 로그레코드들을 통칭하여 'xct_prepare' 로그레코드라 한다.

테이블의 i 번째 비트인 FATable[i]의 값이 1이면 화일 번호가 i인 화일이 존재함을 나타내고, 0이면 존재하지 않음을 나타낸다.

바다-II는 화일에 할당된 디스크 공간을 관리하기 위하여 각 세그먼트에 화일 맵과 익스텐트 맵을 두며 이들의 구조는 2.1절에 제시한 Shore의 구조와 유사하다. 다만, 각 화일 맵 엔트리가 그 세그먼트에서 그 화일의 익스텐트 맵 엔트리 리스트의 선두 엔트리의 번호를 값으로 가지는 필드만을 가지며 그 화일의 삭제 상황을 표시하는 필드를 가지지 않는 것이 다르다.

화일 설명자들은 각 볼륨에서 하나의 디렉토리 루트 페이지와 하나 이상의 디렉토리 리프 페이지들로 구성된 두단계 디렉토리 구조에 저장된다. 디렉토리 리프 페이지는 소속된 볼륨의 디렉토리 카탈로그 화일(directory catalog file)의 페이지이며 하나의 화일 설명자를 하나의 엔트리로 저장한다. 디렉토리 루트 페이지는 소속된 볼륨의 시스템 카탈로그 화일(system catalog file)의 페이지이며 각 디렉토리 리프 페이지의 페이지 식별자를 하나의 엔트리로 저장함으로써 특정 디렉토리 리프 페이지로의 신속한 접근을 제공한다. 색인 설명자들은 각 볼륨에서 그 볼륨에 생성된 색인 화일들의 색인 설명자들을 일반 데이타 화일의 데이타 레코드와 동일한 방법으로 저장하는 색인 카탈로그 화일(index catalog file)에 저장된다.

그림 1은 Shore와 바다-II에서 각 볼륨의 화일 관리 구조를 나타낸 것이다.

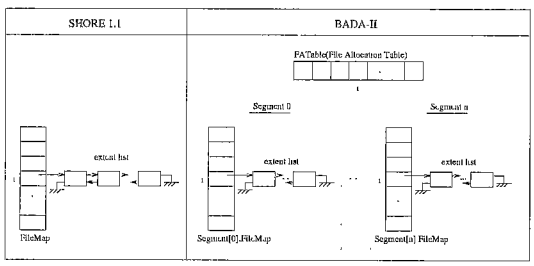


그림 1 Shore와 바다-II에서 각 볼륨의 화일 관리 구조

본 논문은 위의 두가지 구조를 비교하지 않는다. 다만, 화일 삭제와 관련된 작업만을 비교한다. 본 논문이 제시하는 화일 삭제 방법은 Shore에도 적용될 수 있다.

## 3. 화일 삭제 연산의 분석

본 절은 화일 삭제의 연산들을 분석하고, 화일의 생성과 삭제 요구의 식별 정보를 보관하는 화일 정리 리스트(file disposition list)와 미처리 연산으로 삭제할 화일들의 화일 식별자들을 리스트로 지닌 미처리 연산 리스트를 제시한다.

### 3.1 화일 삭제를 위한 연산들의 분석

바다-II는 현재 참조되고 있는 카탈로그 화일들과 데이타 화일들의 화일 설명자들을 메모리 상에 유지하기 위하여 개방 화일 테이블(OpenFileTable)을 두며 현재 참조되고 있는 색인 화일들의 색인 설명자들과 색인의 구조 변경 연산[11, 18]에 관련한 정보를 메모리 상에 유지하기 위하여 개방 색인 테이블(OpenIndexTable)을 둔다. 따라서, 화일 설명자는 디스크에는 디렉토리 카탈로그 화일 또는 색인 카탈로그 화일에 존재하며 메모리 상에는 개방 화일 테이블 또는 개방 색인 테이블에 존재할 수 있다.

바다-II에서 하나의 화일을 삭제하는 과정은 그 화일에의 배타적 접근을 보장받기 위하여 그 화일에 대한 (삭제할 화일이 색인 화일인 경우에는 그 색인 화일이 소속된 데이타 화일에 대해) 배타적 모드의 로크를 획득한 다음, 아래의 다섯 가지 화일 삭제 연산(File Removal Action: FRA)들을 차례로 수행한다.

[FRA1] 디스크 공간에 저장된 화일 설명자를 삭제한다.

[FRA2] 메모리 공간에 보관된 화일 설명자를 무효화한다.

[FRA3] 화일의 일부 내용을 담고 있는 버퍼 페이지들을 버퍼에서 무효화한다.

[FRA4] 화일에 할당된 디스크 공간을 반환한다.

[FRA5] 화일 식별자를 반환한다.

화일 삭제의 다섯 가지 연산들 중 화일 설명자에 대한 FRA1과 FRA2는 그 연산이 수행된 후 그 트랜잭션이 복귀하더라도 시스템의 정확성과 효율성에 문제가 없으므로 즉시 연산으로 수행한다. 그러나, FRA3, FRA4, 그리고 FRA5는 화일 삭제의 정확성과 효율성을 위하여 미처리 연산으로 수행한다.

삭제할 화일의 화일 설명자를 지닌 개방 화일 테이블 또는 개방 색인 테이블의 엔트리를 무효화하는 이유는 만약 그 엔트리를 무효화하지 않는다면 그 엔트리는 삭제된 화일의 화일 식별자를 새로 부여받은 다른 화일을 개방할 경우에 그 화일의 화일 설명자를 가진 엔트리로 사용되는 오류가 발생할 수 있기 때문이다.

삭제할 화일의 버퍼 페이지들을 버퍼에서 무효화하는 이유는 시스템의 성능 향상에 도움이 되기 때문이다. 삭제할 화일의 버퍼 페이지들을 무효화함으로써 다른 유효 페이지들이 버퍼 페이지 교체(buffer page replace-

ment)의 희생자로 선택될 가능성을 줄이고, 삭제할 화일의 버퍼 페이지들이 버퍼 페이지 교체의 희생자로 선정되어 불필요하게 디스크에 쓰여지는 것을 방지하며, 버퍼 변경페이지 테이블(buffer dirty pages table)에서 그 페이지들에 대한 엔트리들을 제거하게 되어 검사점종료 로그레코드의 크기를 줄일 수 있을 뿐만 아니라 그 로그레코드에 바탕을 둔 회복 시의 재수행 시작점(RedoLSN)의 결정에서 이들 페이지들을 배제할 수 있기 때문이다. 그 무효화 작업을 미처리 연산으로 수행하는 이유는 정확성 때문이다. 임시 화일은 그 화일을 생성한 트랜잭션에 의해서만 접근되므로 임시 화일을 삭제하는 경우, 버퍼 페이지들의 무효화 작업을 즉시 연산으로 수행하더라도 다른 트랜잭션에 영향을 미치지 않는다. 그러나, 바다-II와 같이 비강제 반영(no-force) 정책을 따르는 버퍼 관리자의 환경에서 다수 트랜잭션들에 의해 접근되는 정규 화일을 삭제하는 경우, 그 화일의 버퍼 페이지들을 무효화하는 작업은 이미 완료한 다른 트랜잭션들에 의해 변경된 버퍼 페이지들을 무효화할 수 있다. 버퍼 페이지들의 무효화 작업을 즉시 연산으로 수행한다면 그 작업을 수행한 후에 퍼지 검사점이 수행될 경우, 그 페이지들에 대한 변경페이지 테이블의 엔트리들은 검사점 수행시 검사되지 않는다. 이 상태에서 정전이 발생한다면 회복 과정에서 재구성하는 변경페이지 테이블에는 그 페이지들의 엔트리들이 등록되지 않으므로 재수행 단계에서 재수행하는 로그레코드들의 범위에 그 페이지들에 대한 로그레코드들이 포함되지 않을 수 있다. 이로 인해 그 페이지들을 변경하였으며 이미 완료된 다른 트랜잭션들의 원자성과 지속성을 위배할 수 있다. 바다-II는 그 무효화 작업을 그 화일의 화일 유형에 상관없이 일관되게 미처리 연산으로 수행한다.

삭제할 화일에 할당된 디스크 공간의 반환을 미처리 연산으로 수행하는 이유는 만약 화일 삭제의 즉시 연산으로 수행한다면 그 연산을 수행한 트랜잭션이 복귀할 경우, 그 화일의 내용을 다시 복구하기 위한 부담이 매우 크기 때문이다.

화일 식별자는 화일에 관련된 제반 사항들에 접근하기 위한 근원으로 사용되므로 화일 식별자의 반환은 화일 삭제의 가장 마지막에 수행되어야 한다. 데이타 전용 로킹(data-only locking) 기법[11]을 사용하는 바다-II의 환경에서 화일을 삭제하는 트랜잭션이 그 화일에 대하여 획득한 로크를 해제하기 전에 그 화일의 화일 식별자를 반환한다면 그 화일 식별자는 다른 트랜잭션에 의해 새로 생성되는 화일의 화일 식별자로 부여될 수 있으며 화일을 삭제한 트랜잭션이 복귀하여도 그 화일 식별자를 되돌려 받을 수 없는 문제가 발생할 수 있다.

### 3.2 화일 정리 리스트와 미처리 연산 리스트

바다-II는 트랜잭션의 화일 생성 또는 화일 삭제의 요구에 대하여 그 요구의 식별정보로 구성된 하나의 화일 정리 엔트리(file disposition entry)를 형성하며 이러한 화일 정리 엔트리들을 각 요구가 발생한 순으로 정렬한 하나의 리스트로 유지한다. 바다-II는 그 리스트를 화일 정리 리스트라 하며 트랜잭션 테이블의 각 엔트리 즉, 트랜잭션 엔트리(transaction entry)는 해당 트랜잭션의 화일 정리 리스트를 가리키는 FileDispositionList 필드를 가진다[17].

하나의 화일 정리 엔트리는 <FileType, OpType, DataFileID, IndexNo, IndexFileID, IndexRecID, prev>의 필드들을 가진다. FileType은 화일의 유형을 나타낸다. OpType은 화일에 취해진 연산의 유형을 나타내며 화일 생성의 경우에는 FILE_CREATE를, 화일 삭제의 경우에는 FILE_DESTROY를 값으로 가진다. DataFileID, IndexNo, IndexFileID는 화일의 식별자 정보를 나타낸다. prev는 화일 정리 리스트를 단일 연결 리스트로 유지하기 위하여 사용된다.

각 트랜잭션은 미처리 연산들의 정보에 대한 로깅과 그 연산들의 수행 과정의 식별을 위하여 그 트랜잭션의 완료가 결정된 시점에 그 트랜잭션의 미처리 연산 식별 정보들을 취합하여 이들을 미처리 연산 리스트라 불리우는 하나의 리스트로 형성한다. 화일 삭제에 관한한, 바다-II는 그 트랜잭션의 삭제할 화일들의 화일 식별자들을 화일 정리 리스트에서 추출하여 그 정보만으로 미처리 연산 리스트를 형성한다. 이를 위하여 각 트랜잭션 엔트리는 FilePAL과 FilePALSize 필드들을 가진다. FilePAL 필드는 그 트랜잭션의 미처리 연산 리스트를 가리키며 FilePALSize 필드는 그 트랜잭션의 미처리 연산 리스트에 수록된 화일 식별자들의 수를 나타낸다.

## 4. 트랜잭션에 의한 화일 삭제

본 절은 회복 과정에서 미종료 트랜잭션의 식별이 용이하도록 확장한 완료규약 관련 로그레코드와 트랜잭션의 종료 과정을 기술하고, 화일 삭제를 위한 즉시 연산과 미처리 연산의 수행 알고리즘을 제시한다.

### 4.1 완료규약 관련 로그레코드와 트랜잭션의 종료

바다-II의 완료규약 관련 로그레코드들과 트랜잭션의 완료상태는 표 1과 같다.

바다-II는 트랜잭션의 시작시 그 트랜잭션의 시작을 나타내는 트랜잭션시작('begin') 로그레코드를 기록하지 않으며, 어떠한 로그레코드도 기록하지 않은 트랜잭션이

표 1 바다-II에서 완료규약 관련 로그레코드와 트랜잭션의 완료상태

| 로그레코드 유형 | 로그레코드의 의미 | 트랜잭션의 완료상태 |
|---|---|---|
| (no 'begin') | 트랜잭션의 시작 | Unprepared |
| 'pa_start' | 트랜잭션의 완료와 미처리연산수행의 시작 | PAStarted |
| 'end' | 트랜잭션의 종료 | Terminated |
| 'prepare' | 전역 트랜잭션의 완료준비 | Prepared |
| 'rollback' | 전역 트랜잭션의 전체복귀시작 | Unprepared |

완료 또는 전체 복귀(total rollback)할 경우에는 트랜잭션종료 로그레코드를 기록하지 않는다[16].

표 1에서 미처리연산수행시작 로그레코드는 미처리 연산의 정확성을 위해 ARIES의 완료규약 관련 로그레코드들의 집합에 새로이 추가한 로그레코드로서 미처리 연산 리스트를 가진 트랜잭션의 완료와 미처리 연산 수행의 시작을 함께 나타낸다.

트랜잭션의 완료가 요구되면, 각 트랜잭션은 화일 삭제에 관련한 완료 준비작업을 아래와 같이 수행한다.

• 생성한 임시 화일들 중 삭제 요구가 없는 임시 화일들에 대하여 화일 삭제의 즉시 연산을 수행한다. 삭제 요구가 없는 임시 화일이란 화일 정리 리스트에 그 임시 화일의 생성을 나타내는 엔트리는 존재하지만 삭제를 나타내는 엔트리가 존재하지 않는 화일을 의미한다. 이러한 화일에 대하여 즉시 연산을 수행함으로써 화일 정리 리스트에는 그 트랜잭션이 생성한 각 임시 화일에 관하여 FILE_CREATE를 나타내는 엔트리와 FILE_DESTROY를 나타내는 엔트리의 쌍이 존재하게 된다.

• 화일 정리 리스트에서 연산의 유형이 FILE_DESTROY인 엔트리들 즉, 삭제할 화일들의 화일 식별자들을 추출하여 미처리 연산 리스트를 형성한다.

지역 트랜잭션의 완료 작업은 그 트랜잭션의 수행 중 로그레코드의 기록 여부와 미처리 연산 리스트의 소유 여부에 따라 아래의 세 가지 종료형태 중 하나로 종료한다.

1. 어떤 로그레코드도 기록하지 않은 경우. 그 트랜잭션은 완료 시에도 로그레코드를 기록하지 않는다. 따라서, 그 트랜잭션의 완료상태를 Terminated로 변경한 후, 그 트랜잭션의 수행 중에 개방한 스캔(scan)들을 모두 닫고 스캔 정보를 유지하는 메모리 공간을 해제하는 스캔 마무리 작업을 수행하고, 획득하고 있는 모든 로크들을 해제한 후, 자신의 트랜잭션 엔트리를 초기화함으로써 종료한다.[4] 그림 2-a는 이 경우를 나타낸다.

2. 로그레코드를 기록하였지만 미처리 연산 리스트를 가지지 않은 경우. 'end' 로그레코드를 지연된 강제쓰기(delayed-force) 유형으로 기록한 후 트랜잭션의 완료상태를 Terminated로 변경한다. 그리고 나서, 스캔 마무리 작업을 수행하고, 획득한 모든 로크들을 해제하며, 'end' 로그레코드가 로그 화일에 반영되지 않은 경우에는 그 로그레코드를 로그 화일에 강제쓰기한 다음 자신의 트랜잭션 엔트리를 초기화함으로써 종료한다. 그림 2-b는 이 경우를 나타낸다.

3. 미처리 연산 리스트를 가진 경우. 'pa_start' 로그레코드에 미처리 연산 리스트를 수록하여 지연된 강제쓰기 유형으로 기록하고 트랜잭션의 완료상태를 PAStarted로 변경한 다음, 스캔 마무리 작업을 수행하고, 미처리 연산 리스트에 수록된 화일들에 대한 화일 삭제의 미처리 연산들을 수행하고, 획득한 모든 로크들을 해제한다. 그리고 나서, 'end' 로그레코드를 비강제쓰기(no-force) 유형으로 기록하고 트랜잭션의 완료상태를 Terminated로 변경한 다음, 'pa_start' 로그레코드가 로그 화일에 반영되지 않은 경우에는 그 로그레코드를 로그 화일에 강제쓰기하고, 자신의 트랜잭션 엔트리를 초기화함으로써 종료한다. 그림 2-c는 이 경우를 나타낸다.

전역 트랜잭션의 완료 작업은 지역 트랜잭션의 완료 작업과 비교해서 완료규약 관련 로그레코드의 기록과 트랜잭션 완료상태의 변경만 다르므로 이들 사항에 대해서만 아래에 기술한다.

1. 어떤 로그레코드도 기록하지 않은 경우. 완료준비(prepare) 메세지에 대해 로그레코드를 기록하지 않고 트랜잭션의 완료상태만 Prepared로 설정하며, 완료(commit) 메세지에 대해 로그레코드를 기록하지 않고 트랜잭션의 완료상태만 Terminated로 설정한다.

2. 로그레코드를 기록하였지만 미처리 연산 리스트를 가지지 않은 경우. 완료준비 메세지에 대해 그 트랜잭션이 획득한 로크 정보를 수록한 'prepare' 로그레코드를 강제쓰기 유형으로 기록하고 트랜잭션의 완료상태를 Prepared로 설정하며, 완료 메세지에 대해 'end' 로그레코드를 강제쓰기 유형으로 기록하고, 트랜잭션의 완료상태를 Terminated로 설정한다.

3. 미처리 연산 리스트를 가진 경우. 완료준비 메세지에 대해 그 트랜잭션이 획득한 로크 정보와 그 트랜잭션의 미처리 연산 리스트를 수록한 'prepare' 로그레코

4) 트랜잭션 종료의 마지막 작업인 트랜잭션 엔트리의 초기화는 그 트랜잭션의 수행 중에 메모리 공간을 할당받아 보관한 저장점(savepoint) 정보, 화일 관련 정보, 미처리 연산 리스트가 있으면 그 메모리 공간을 모두 해제한다.

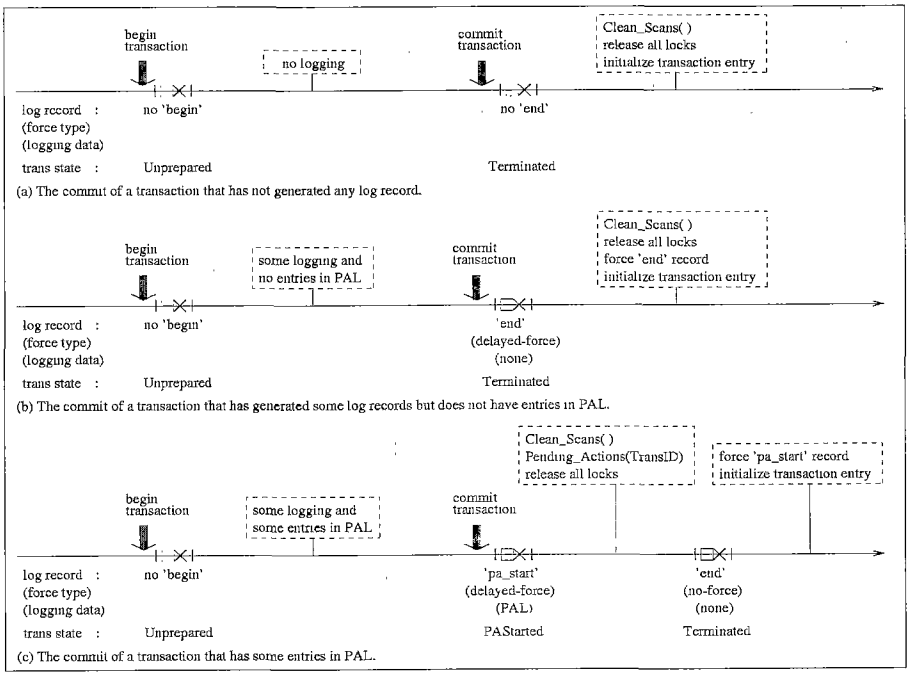


그림 2 지역 트랜잭션의 완료

드를 강제쓰기 유형으로 기록하고 트랜잭션의 완료상태를 Prepared로 설정하며, 완료 메세지에 대해 미처리 연산 리스트를 지니지 않은 'pa_start' 로그레코드를 강제쓰기 유형으로 기록하고 트랜잭션의 완료상태를 PAStarted로 설정하며 미처리 연산 리스트에 수록된 화일들에 대한 화일 삭제의 미처리 연산을 수행한 후 'end' 로그레코드를 비강제쓰기 유형으로 기록하고 트랜잭션의 완료상태를 Terminated로 설정한다.

트랜잭션의 전체 복귀는 [17]에 기술한 복귀 작업을 수행한 후 로그레코드의 기록 여부에 따라 다음의 두가지 종료 형태 중 하나로 종료한다.

1. 어떤 로그레코드도 기록하지 않은 경우. 'end' 로그레코드를 기록하지 않으며, 트랜잭션의 완료상태만 Terminated로 변경한 후 트랜잭션 엔트리를 초기화한다.

2. 로그레코드를 기록한 경우. 'end' 로그레코드를 비강제쓰기 유형으로 기록하고, 트랜잭션의 완료상태를 Terminated로 변경한 후 트랜잭션 엔트리를 초기화한다.

전역 트랜잭션이 Prepared 완료상태에서 전체 복귀하는 경우에는 그 트랜잭션의 복귀시작을 나타내는 'rollback' 로그레코드를 기록하고 그 트랜잭션의 완료상태를 Unprepared로 변경한 후 해당 복귀 작업을 수행하며, 복귀 작업을 마친 후에는 앞서 기술한 두 번째 종료 방법에 따라 그 트랜잭션을 종료한다.

### 4.2 화일 삭제를 위한 즉시 연산의 수행 알고리즘

색인 화일의 삭제를 위한 즉시 연산의 수행 절차는 다음과 같다. (1) 그 색인 화일이 정규 색인 화일이면 그 색인 화일이 소속된 데이타 화일에 대해 배타적 모드의 로크를 획득한다. 임시 색인 화일인 경우에는 로크를 획득하지 않는다. (2) 그 색인이 소속된 데이타 화일과 그 색인 화일을 차례로 개방하여 그 색인 화일의 화일 식별자를 구한다. (3) 색인 카탈로그 화일에서 그 색인의 색인 설명자를 삭제하고 그 변경 사항을 redo/undo 유형의 로그레코드로 기록한다. (4) 그 색인이 소속된 데이타 화일의 화일 설명자에서 그 색인의 존재를 삭제한다. 이 작업은 개방 화일 테이블의 엔트리에 등록된 그 데이타 화일의 화일 설명자에서 그 색인의 존재를 삭제하고 디렉토리 카탈로그 화일에 저장된 그 데이타 화일의 화일 설명자에서 그 색인의 존재를 삭제하며 이에 대해 redo/undo 유형의 로그레코드를 기록함으로써 수행된다. (5) 그 색인 화일의 화일 유형, FILE_DESTROY 연산 유형, 그 색인의 식별 정보 그리고 그 색인의 색인 설명자가 저장된 색인 카탈로그 화일에서의 해당 레코드의 식별자로 하나의 화일 정리 엔트리를 형성하고 그 엔트리를 화일 정리 리스트의 후미에 연결

한다. (6) 개방 색인 테이블에서 그 색인의 엔트리를 무효화한다. (7) 해당 데이타 화일을 닫는다.

데이타 화일의 삭제를 위한 즉시 연산의 수행 절차는 다음과 같다. (1) 그 데이타 화일이 정규 화일이면 그 데이타 화일에 대해 배타적 모드의 로크를 획득한다. 임시 데이타 화일이면 로크를 획득하지 않는다. (2) 그 데이타 화일을 개방한다. (3) 디렉토리 카탈로그 화일에서 그 데이타 화일의 화일 설명자를 삭제하고 그 변경 사항에 대해 redo/undo 유형의 로그레코드를 기록한다. (4) 그 데이타 화일에 소속된 모든 색인들을 삭제한다. (5) 그 데이타 화일의 화일 유형, FILE_DESTROY 연산 유형 그리고 그 데이타 화일의 식별 정보로 구성된 하나의 화일 정리 엔트리를 형성하고 그 엔트리를 화일 정리 리스트의 후미에 연결한다. (6) 개방 화일 테이블에서 그 데이타 화일의 엔트리를 무효화한다.

데이타 화일에 소속된 모든 색인들의 삭제는 개방 화일 테이블에 등록된 그 데이타 화일의 화일 설명자에서 그 데이타 화일을 바탕으로 생성된 각 색인을 식별하여 아래의 절차대로 삭제한다. (1) 색인 화일을 개방하여 그 색인 화일의 화일 식별자를 구한다. (2) 색인 카탈로그 화일에서 그 색인의 색인 설명자를 삭제하고 그 변경 사항에 대해 redo/undo 유형의 로그레코드를 기록한다. (3) 그 색인 화일의 화일 유형, FILE_DESTROY 연산 유형, 그 색인 화일의 식별 정보 그리고 Index RecID 필드는 NIL_RID로 하여 하나의 화일 정리 엔트리를 형성하고 그 엔트리를 화일 정리 리스트의 후미에 연결한다.[5] (4) 개방 색인 테이블에서 그 색인의 엔트리를 무효화한다.

### 4.3 화일 삭제를 위한 미처리 연산의 수행 알고리즘

트랜잭션의 완료 과정에서 화일 삭제를 위한 미처리 연산의 수행은 그 트랜잭션의 미처리 연산 리스트의 선두 엔트리부터 후미 엔트리까지 각 엔트리가 나타내는 화일 식별자에 대하여 FRA3, FRA4, FRA5를 차례로 수행한다. 그림 3은 그 과정을 나타낸다.

그림 3의 Pending_Actions() 함수는 트랜잭션의 완료 시와 본 논문의 6.3 절에서 제시할 재시작 회복의 아직 수행되지 않은 미처리 연산의 수행 단계에서 호출된다. 전자의 경우에는 StartPALIndex를 0으로 호출하며 후자의 경우에는 StartPALIndex를 다음에 삭제할 화일에 대한 미처리 연산 리스트에서의 엔트리 번호로 호출한다.

```
Pending_Actions(TransID,Trans_Table,StartPALIndex)
{
    for (PALIndex = StartPALIndex;
            PALIndex<Trans_Table[TransID].FilePALSize;
            PALIndex++) {
        FileID=Trans_Table[TransID].FilePAL[PALIndex];
        BUF_InvalidateFile(FileID);                    /* FRA3 */
        Deallocate_Disk_Space(TransID,FileID,PALIndex,0);
                                                       /* FRA4 */
        Delete_FileID(TransID,FileID,PALIndex); /* FRA5 */
    }
}
```

그림 3 화일 삭제를 위한 미처리 연산의 수행

미처리 연산의 수행 중에 변경한 페이지에 대하여 생성하는 로그레코드들은 회복 과정에서 재수행의 대상이 될 수는 있지만 취소의 대상은 되지 않는다. 따라서, 이들은 취소 정보를 가지지 않으며 재수행 정보와 미처리 연산 식별 정보를 가지는 redo-only 유형이다.

4.3.1 버퍼 페이지들의 무효화 (FRA3)

바다-II에서 페이지 식별자는 그 페이지가 소속된 볼륨의 볼륨 번호, 세그먼트의 세그먼트 번호, 그리고 그 세그먼트에서 자신을 유일하게 식별하기 위한 페이지 번호(page number)로 구성되며 그 페이지가 소속된 화일의 화일 식별자를 포함하지 않는다. 그대신, 각 페이지는 그 페이지의 헤더(header)에 그 페이지가 소속된 화일의 화일 식별자를 지닌다. 따라서, 삭제할 화일의 버퍼 페이지들의 무효화 작업은 버퍼에 수록된 각 페이지의 화일 식별자 값을 참조하여 그 화일의 버퍼 페이지들을 지닌 버퍼 제어 블록(buffer control block)들을 식별하고 그 버퍼 제어 블록들을 LRU 리스트와 버퍼 해쉬 체인(buffer hash chain)에서 제거한 후 무효 버퍼 제어 블록 리스트에 연결하고, 버퍼 변경페이지 테이블에서 그 페이지들에 대한 엔트리들을 초기화함으로써 수행된다.

이 작업을 수행하면서 바다-II는 무효화한 페이지들의 페이지 식별자들로 Invalid_Pages_List를 형성하며 이를 수록한 'OSfile_return' 로그레코드를 기록한다. 이는 ARIES의 재시작 분석 단계가 가지고 있던 기존 'OSfile_return' 로그레코드의 처리 부분을 수용하기 위한 것으로 재시작 회복의 분석 단계에서 버퍼 변경페이지 테이블에 등록된 엔트리들 중 삭제된 화일의 페이지

5) 데이타 화일의 삭제에 의해 그 데이타 화일에 생성된 모든 색인들을 삭제하는 경우이므로 이 연산의 수행 중에 저장점이 설정될 수 없으며 그 저장점으로의 부분 복귀를 고려할 필요가 없다. 따라서, 개방 화일 테이블에 존재하는 그 데이타 화일의 화일 설명자에서 각 색인의 존재를 삭제할 필요가 없으며 그 색인에 대한 화일 정리 엔트리에서 IndexRecID 필드는 그 색인의 색인 설명자의 레코드 식별자를 가질 필요가 없다.

들에 대한 엔트리들을 제거함으로써 최적의 RedoLSN을 결정하기 위함이다.[6]

4.3.2 디스크 공간의 반환 (FRA4)

디스크 공간의 반환 작업은 삭제할 화일이 소속된 볼륨의 첫번째 세그먼트부터 마지막 세그먼트까지 차례로 탐색하면서 각 세그먼트에 대하여 그 화일의 익스텐트 맵 엔트리 리스트의 모든 익스텐트 맵 엔트리들을 초기화하고 화일 맵에서 그 화일의 화일 맵 엔트리를 초기화함으로써 수행된다. 그림 4는 디스크 공간의 반환 작업을 나타낸다.

```
Deallocate_Disk_Space(TransID, FileID, PALIndex, StartSegment
Num)
{
    get VolumeNum and FileNum from FileID;
    get NumSegs from Volume_Table[VolumeNum];
    for(SegmentNum=StartSegmentNum; SegmentNum<Num
    Segs; SegmentNum++) {
        Get_FirstExtent(VolumeNum,SegmentNum,FileNum,
            FirstExtent);
        if (FirstExtent != NIL_EXT) {
            Free_EM_Entries(TransID,FileID,SegmentNum,
                FirstExtent,PALIndex);
            Clear_FM_Entry(TransID,FileID,SegmentNum,PAL
                Index);
        }
    }
}
```

그림 4 화일에 할당된 디스크 공간의 반환

그림 4의 Deallocate_Disk_Space() 함수는 트랜잭션의 완료 시와 본 논문의 6.3 절에서 제시할 재시작 회복의 아직 수행되지 않은 미처리 연산의 수행 단계에서 호출된다. 전자의 경우에는 StartSegmentNum을 0으로 호출하며 후자의 경우에는 StartSegmentNum을 다음에 검사할 세그먼트의 세그먼트 번호로 호출한다.

그림 4에서 Get_FirstExtent() 함수는 해당 볼륨의 해당 세그먼트에서 삭제할 화일에 할당된 익스텐트들이 존재하면 그 익스텐트 맵 엔트리 리스트의 선두 익스텐트 맵 엔트리의 번호를, 존재하지 않으면 NIL_EXT를 FirstExtent를 통해 반환한다.

그림 4에서 Free_EM_Entries() 함수는 지정된 세그먼트에서 삭제할 화일의 익스텐트 맵 엔트리 리스트의 모든 익스텐트 맵 엔트리들을 초기화하는 작업을 수행하는 것으로서 그 리스트의 선두 엔트리부터 후미 엔트리까지 차례로 탐색하면서 그 익스텐트 맵 엔트리들을 모두 초기화할 때까지 다음의 과정을 반복한다. 아래의 과정에서 익스텐트 맵 엔트리들의 초기화 작업은 페이지 래취(latch)에 의한 교착상태(deadlock)를 방지하기 위하여 버퍼에는 한번에 하나의 익스텐트 맵 페이지만 장착한다. (1) 다음에 초기화할 익스텐트 맵 엔트리를 가진 익스텐트 맵 페이지를 WRITE 모드로 버퍼에 장착한다. (2) 그 익스텐트 맵 엔트리에서 시작하는 리스트에서 그 익스텐트 맵 페이지에 존재하는 모든 익스텐트 맵 엔트리들을 초기화하며 그 익스텐트 맵 페이지에서 그 리스트의 마지막 익스텐트 맵 엔트리의 다음 익스텐트 맵 엔트리의 엔트리 번호를 StartExtent On NextPage에 기억한다. (3) 'pa_freeEME' 로그레코드를 기록하고 그 로그레코드의 LSN을 익스텐트 맵 페이지의 PageLSN에 반영한다. (4) 버퍼에서 그 익스텐트 맵 페이지를 탈착한다.

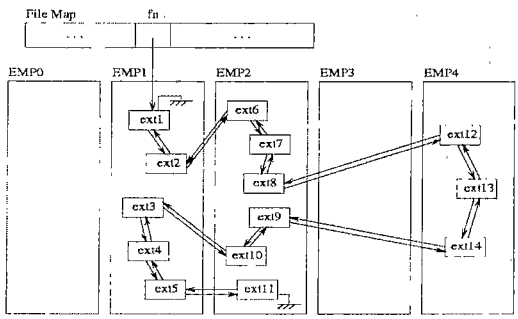


그림 5 특정 세그먼트에서 삭제할 화일의 익스텐트 맵 엔트리 리스트의 예

예를 들어, 어떤 세그먼트에서 화일 번호가 fn인 화일에 할당된 익스텐트들의 익스텐트 맵 엔트리 리스트가 그림 5와 같이 존재한다고 하자. 이 경우, EMP1의 <ext1, ext2>, EMP2의 <ext6, ext7, ext8>, EMP4의 <ext12, ext13, ext14>, EMP2의 <ext9, ext10>, EMP1의 <ext3, ext4, ext5>, 그리고 EMP2의 <ext11>의 순서로 각 익스텐트 맵 페이지를 차례로 접근하여 그 익스텐트 맵 엔트리들을 초기화한다.

6) ARIES는 페이지 식별자에 화일 식별자를 포함하므로 'OSfile_return' 로그레코드에 화일 식별자만을 수록하더라도 회복 과정에서 버퍼 변경페이지 테이블에 수록된 <PageID, RecLSN>의 쌍에서 PageID만을 참조하여 그 화일에 소속된 페이지들을 판별할 수 있다. 그러나, 바다-II는 그렇지 못하기 때문에 'OSfile_return' 로그레코드에 무효화한 페이지들의 페이지 식별자들을 모두 수록한다. 만약, 버퍼 변경페이지 테이블의 각 엔트리에 해당 페이지가 소속된 화일의 화일 식별자 정보를 보관하는 필드를 추가한다면 화일 식별자를 이용하여 버퍼 변경페이지 테이블에서 그 화일에 소속된 페이지들의 엔트리들을 삭제할 수 있다.

```
Free_EM_Entries(TransID, FileID, SegmentNum, StartExtent,
PALIndex)
{
get PageID of the first extent map page, say FirstEMPageID,
        in the segment SegmentNum;
    StartExtentOnThisPage=StartExtent;
    while (StartExtentOnThisPage !=NIL_EXT) {
        EMPageID=FirstEMPageID+(StartExtentOnThisPage/
        EXTS_PER_PAGE);
        Page=fix&latch(EMPageID, 'WRITE');
        get StartExtentOnNextPage while initializing EM
            entries that exist in the page EMPageID and also
            in the EM entry list that starts from StartExtent
            OnThisPage;
        Log_Write(TransID,'pa_freeEME',EMPageID, ...,
            StartExtentOnThisPage,PALIndex,StartExtent
            OnNextPage,LgLSN);
        Page.PageLSN=LgLSN;
        unfix&unlatch(Page);
        StartExtentOnThisPage=StartExtentOnNextPage;
    }
}
```

그림 6 익스텐트 맵 엔트리들의 초기화

그림 6은 익스텐트 맵 엔트리들의 초기화 과정을 나타낸다. 그림 6에서 EXTS_PER_PAGE는 하나의 익스텐트 맵 페이지에 수록되는 익스텐트 맵 엔트리들의 최대 수효를 나타낸다. 익스텐트 맵 엔트리들을 초기화하면서 기록하는 'pa_freeEME' 로그레코드는 PALIndex를 통하여 현재 삭제하는 화일을 식별할 수 있으며, EMPage ID를 통하여 SegmentNum을 식별할 수 있으며, Start ExtentOnThisPage를 통하여 EMPageID 페이지에서 초기화한 익스텐트 맵 엔트리들을 식별할 수 있으며, StartExtentOnNextPage를 통하여 다음에 반환할 익스텐트의 익스텐트 맵 엔트리 번호를 식별할 수 있게 한다.

화일 맵 엔트리의 초기화 작업은 그림 7과 같이 그 화일 맵 엔트리가 위치하는 화일 맵 페이지를 WRITE 모드로 버퍼에 장착하고, 그 화일 맵 엔트리에 NIL_EXT를 기록함으로써 그 화일 맵 엔트리를 초기화한 후, 'pa_clearFME' 로그레코드를 기록하고 그 로그레코드의 LSN을 그 화일 맵 페이지의 PageLSN에 반영한 다음, 버퍼에서 그 화일 맵 페이지를 탈착함으로써 수행된다.

그림 7에서 FMENTS_PER_PAGE는 하나의 화일 맵 페이지에 수록되는 화일 맵 엔트리들의 최대 수효를 나타낸다. 화일 맵 엔트리들을 초기화하면서 기록하는 'pa_clearFME' 로그레코드는 PALIndex를 통하여 현재

```
Clear_FM_Entry(TransID, FileID, SegmentNum, PAL
Index)
{
    get VolumeNum and FileNum from FileID;
    get PageID of the first file map page, say First
        FMPageID, in the segment SegmentNum;
    FMPageID=FirstFMPageID+(FileNum/FMENTS_
        PER_PAGE);
    FMEnteryIndex=FileNum% FMENTS_PER_PAGE;
    Page=fix&latch(FMPageID, 'WRITE');
    Page.FM[FMEnteryIndex]=NIL_EXT;
    Log_Write(TransID,'pa_clearFME',FMPageID, ...,
        FMEnteryIndex,PALIndex,LgLSN);
    Page.PageLSN = LgLSN;
    unfix&unlatch(Page);
}
```

그림 7 화일 맵 엔트리의 초기화

삭제하는 화일을 가리키며, FMPageID를 통하여 SegmentNum을 확인할 수 있으며, FMEnteryIndex를 통하여 FMPageID 페이지에서 그 화일의 엔트리를 식별할 수 있게 한다.

4.3.3 화일 식별자의 반환(FRA5)

화일 식별자의 반환 작업은 (1) 삭제할 화일이 소속된 볼륨의 볼륨 상태 페이지를 WRITE 모드로 버퍼에 장착하고, (2) 화일 할당 테이블에서 그 화일의 화일 번호에 해당하는 비트를 0으로 설정하고, (3) 그 화일의 화일 유형에 따라 볼륨 상태 페이지에 있는 다른 필드들이 지닌 정보들을 조정한 후, (4) 볼륨 상태 페이지에 발생한 변경 사항에 대하여 'pa_deleteFID' 로그레코드를 기록하고 그 로그레코드의 LSN을 볼륨 상태 페이지의 PageLSN에 반영한 다음, (5) 버퍼에서 볼륨 상태 페이지를 탈착함으로써 수행된다. 그림 8은 화일 식별자

```
Delete_FileID(TransID, FileID, PALIndex)
{
    get VolumeNum and FileNum from FileID;
    get PageID of the volume status page of the
        volume VolumeNum;
    Page = fix&latch(PageID, 'WRITE');
    Page.FATable[FileNum] = 0;
    Update other fields of the volume status page;
    Log_Write(TransID,'pa_deleteFID',PageID, ...,
        FileID,PALIndex,LgLSN);
    Page.PageLSN = LgLSN;
    unfix&unlatch(Page);
}
```

그림 8 화일 식별자의 반환

의 반환 작업을 나타낸다.

볼륨 상태 페이지에서 화일 식별자를 반환하면서 기록하는 'pa_deleteFID' 로그레코드는 PALIndex를 통해 현재 삭제하고 있는 화일을 유일하게 식별할 수 있게 한다.

4.3.4 미처리 연산 중의 오류 처리

미처리 연산의 수행 중에 발생 가능한 트랜잭션 고장에 대하여 고려하여야 한다. 트랜잭션 고장 중에는 해당 데이타베이스 관리시스템 자체의 오동작에 의한 고장과 그 데이타베이스 관리시스템이 기반으로 하는 하드웨어 또는 운영체제의 오동작에 의한 고장이 있을 수 있다.

첫번째 유형의 고장 예로 데이타 디스크 공간의 부족으로 인한 화일 확장의 오류, 로그 공간의 부족으로 인한 로그 기록의 오류, 버퍼 공간의 부족으로 인한 버퍼 장착의 오류 등이 있다. 이 유형의 고장은 데이타베이스 관리시스템의 각 관리자들이 모든 연산의 수행 상황에 대하여 정확하게 동작할 수 있도록 정확히 구현함으로써 해결할 수 있다.

두번째 유형의 고장은 하드웨어 또는 운영체제의 오동작에 의해 발생하기 때문에 데이타베이스 관리시스템과 관계없이 발생하는 고장이다. 이러한 고장의 예로 디스크에 저장된 데이타를 읽기 위한 읽기 연산의 오류, 데이타를 디스크에 쓰기 위한 쓰기 연산의 오류, 메모리 공간을 할당받으려는 요구의 오류, 프로세스 또는 쓰레드(thread)가 수면(sleep)하기 위한 요구의 오류 등이 있다. 이 유형의 고장에 대하여 시스템을 비정상 종료시킴으로써 즉, 트랜잭션 고장을 시스템 고장으로 강제 변경시킴으로써 그 고장에 대해 재시작 회복으로 아직 수행되지 않은 미처리 연산의 수행을 완료하게 하여 트랜잭션의 원자성과 지속성을 보장한다.

## 5. 퍼지 검사점의 확장

재시작 회복시 재수행해야 하는 작업량을 줄이기 위하여 주기적으로 수행하는 검사점들 중 퍼지 검사점은 정상적인 트랜잭션의 수행과 병행하여 수행되는 검사점이다. ARIES의 퍼지 검사점은 검사점시작('begin_chkpt') 로그레코드를 기록한 후, 트랜잭션 테이블을 검사하여 현재 수행 중인 트랜잭션들의 수행상태인 <TransID, State, LastLSN, UndoNxtLSN>을 수록한 트랜잭션 리스트(transaction list)를 형성하고 버퍼의 변경페이지 테이블을 검사하여 현재 변경되었거나 앞으로 변경될 예정인 페이지들의 상태를 수록한 변경페이지 리스트(dirty pages list)를 형성한 다음 그 리스트들을 검사점종료 로그레코드로서 로그 화일에 기록하고 마지막으로 마스터 레코드를 변경한다.

퍼지 검사점이 완료 과정에 있는 트랜잭션의 미처리 연산 리스트를 검사점종료 로그레코드에 반영하지 않는다면 그 트랜잭션이 미처리 연산을 수행하는 도중에 고장이 발생할 경우에는 회복 과정에서 그 트랜잭션의 미처리 연산 리스트는 트랜잭션 테이블에 복구되지 않을 수 있다. 예를 들어, 그림 9와 같이 지역 트랜잭션 $T_i$가 'pa_start' 로그레코드를 로깅한 다음, 미처리 연산 리스트에 등록된 화일들을 삭제하는 도중에 퍼지 검사점이 시작되어 완료되었으며, $T_i$가 미처리 연산을 완료하기 전에 고장이 발생하였다고 가정하자.

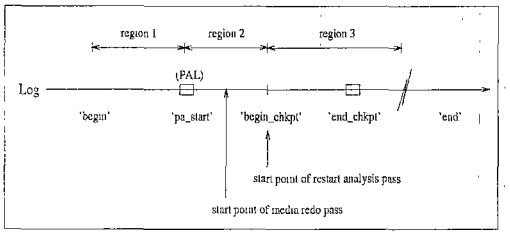


그림 9 미처리 연산 수행 중의 검사점 수행과 시스템 고장

이 경우, 퍼지 검사점이 $T_i$의 미처리 연산 리스트를 로깅하지 않았다면 재시작 회복과 매체 회복 모두에서 $T_i$의 미처리 연산 리스트는 아래와 같이 트랜잭션 테이블에 복구되지 않을 수 있다.

1. 재시작 회복의 분석 단계는 마지막 완성 검사점의 검사점시작 로그레코드부터 검색을 시작하므로 $T_i$의 'pa_start' 로그레코드는 재시작 분석단계에 의해 검색되지 않는다.

2. 매체 회복의 재수행 단계는 매체 복사 검사점(media copy checkpoint)의 검사점종료 로그레코드에 수록된 변경페이지 리스트에서 최소의 RecLSN이 가리키는 로그레코드부터 재수행을 시작한다. 만약, 로그레코드들의 재수행이 그림 9의 범위 2에 있는 로그레코드부터 시작한다면 $T_i$의 'pa_start' 로그레코드는 매체 재수행 단계에 의해 검색되지 않는다.

완료 과정에 있는 트랜잭션의 미처리 연산 리스트를 검사점종료 로그레코드에 반영하기 위해 ARIES의 퍼지 검사점을 확장하여 트랜잭션 테이블의 검사 작업시에 (1) 트랜잭션의 완료상태가 PAStarted이거나 (2) 트랜잭션의 완료상태가 Prepared이면서 미처리 연산 리스트를 가지고 있는 경우에 한하여 그 트랜잭션의 수행상태

와 함께 그 트랜잭션의 미처리 연산 리스트를 트랜잭션 리스트에 수록하며, 그 외의 경우이면 그 트랜잭션의 수행상태만 트랜잭션 리스트에 수록한다.

## 6. 회복 알고리즘의 확장

ARIES의 재시작 회복은 분석 단계, 재수행 단계, 그리고 취소 단계의 세 단계로 수행되며 매체 회복은 재수행 단계와 취소 단계의 두 단계로 수행된다. 본 장은 미처리 연산을 고려하여 ARIES의 회복 알고리즘을 확장한 사항들을 제시한다.

확장한 재시작 회복은 (1) 재시작 분석 단계에서 미종료 트랜잭션들의 미처리 연산 리스트를 트랜잭션 테이블에 복구하고 RedoLSN의 설정을 조정하며, (2) 재시작 재수행 단계에서 미종료 트랜잭션들이 마지막으로 수행한 미처리 연산을 식별하고, (3) 재시작 취소 단계 후에 아직 수행되지 않은 미처리 연산의 수행 단계를 추가하여 미종료 트랜잭션들의 아직 수행되지 않은 미처리 연산들의 수행을 완료한 후 그 트랜잭션들을 종료시킨다.

확장한 매체 회복은 재수행 단계, 취소 단계, 그리고 아직 수행되지 않은 미처리 연산의 수행 단계의 세 단계로 수행되며, 매체 재수행 단계는 재시작 회복의 분석 단계와 재수행 단계에 추가된 사항을 그대로 가지며, 아직 수행되지 않은 미처리 연산의 수행 단계는 재시작 회복의 미처리 연산 수행 단계와 동일하다.

재시작 회복과 매체 회복의 취소 단계는 미처리 연산에 무관한 단계이므로 본 논문에서 논의하지 않는다.

### 6.1 재시작 분석 단계의 확장

ARIES의 재시작 분석 단계는 [14]에서 허상 트랜잭션(dangling transaction)의 검출과 해결을 위하여 수정되었다. 본 논문은 수정된 재시작 분석단계에 대하여 미처리 연산을 고려하여 아래의 세가지 기능을 추가한다.

첫째, 미종료 트랜잭션들의 식별과 그 트랜잭션들의 미처리 연산 리스트를 트랜잭션 테이블에 복구하기 위하여 아래의 작업들을 추가한다.

1. 'prepare' 로그레코드를 검색하는 경우, 그 로그레코드에 미처리 연산 리스트가 수록되어 있지만 그 로그레코드를 생성한 트랜잭션의 트랜잭션 엔트리에 미처리 연산 리스트가 등록되어 있지 않다면 그 미처리 연산 리스트를 그 트랜잭션 엔트리에 복구한다.

2. 'pa_start' 로그레코드를 검색하는 경우, 그 로그레코드를 생성한 트랜잭션의 트랜잭션 엔트리에 대하여 완료상태를 'PAStarted'로 설정하며 그 트랜잭션 엔트리에 미처리 연산 리스트가 등록되어 있지 않다면 그 미처리 연산 리스트를 그 트랜잭션 엔트리에 복구하며 그 트랜잭션 엔트리의 LastLSN을 그 로그레코드의 LSN으로 설정한다.

3. 마지막 완성 검사점의 검사점종료 로그레코드에 수록된 트랜잭션 리스트의 각 엔트리에 대하여 그 엔트리에 해당하는 트랜잭션의 트랜잭션 식별자가 종료 트랜잭션 리스트(complete_trans_list)에 등록되어 있지 않으며 그 엔트리가 미처리 연산 리스트를 가지고 있으며 그 엔트리에 해당하는 트랜잭션의 트랜잭션 엔트리의 미처리 연산 리스트가 등록되어 있지 않은 경우에 한하여 그 미처리 연산 리스트를 그 트랜잭션 엔트리에 복구한다.[7]

둘째, 'OSfile_return' 로그레코드를 검색하는 경우, 본 논문의 5.3.1절에 제시한 개념에 의해 그 로그레코드에 대한 ARIES의 원래 기능인 화일 삭제 기능을 대체하여 그 로그레코드의 Invalid_Pages_List에 등록된 모든 페이지 식별자들을 변경 페이지 테이블에서 삭제한다.

셋째, 미처리 연산 리스트를 지닌 미종료 트랜잭션의 마지막 로그레코드가 마지막 완성 검사점의 검사점시작 로그레코드 이전에 기록되더라도 그 로그레코드가 재시작 재수행 단계에서 검색되게 함으로써 그 트랜잭션의 미처리 연산 식별 정보를 구할 수 있게 하기 위하여 분석 단계의 마지막에 결정하는 RedoLSN을 버퍼 변경페이지 테이블에 등록된 엔트리들의 RecLSN들과 트랜잭션 테이블에서 완료상태가 'PAStarted'인 미종료 트랜잭션들의 LastLSN들 중 최소 LSN으로 설정한다.

### 6.2 재시작 재수행 단계의 확장

로그레코드의 유형이 'pa_freeEME', 'pa_clearFME', 그리고 'pa_deleteFID' 중의 하나인 로그레코드를 미처리 연산 관련 로그레코드라 한다. 각 미종료 트랜잭션이 마지막으로 수행한 미처리 연산을 식별하기 위하여 ARIES의 재시작 재수행 단계를 다음과 같이 확장한다.

첫째, 각 미종료 트랜잭션이 마지막으로 수행한 미처리 연산의 식별 정보를 보관하기 위하여 <PALIndex, Log Type, SegmentNum, StartExtOnNextPage>로 구성된 PENDINFO 구조체를 정의하고 PENDINFO 구조체의 메모리 주소를 값으로 가지는 Pending_Info 필드를 트랜잭션 테이블의 각 엔트리에 둔다.

둘째, 재시작 재수행 단계의 기존 작업에 추가하여 회복 프로세스가 검색한 로그레코드가 미종료 트랜잭션이

7) 종료 트랜잭션 리스트는 [14]의 재시작 분석단계와 매체 재수행 단계에서 종료한 트랜잭션들의 트랜잭션 식별자들을 기록해 두는 리스트이다.

기록한 마지막 로그레코드이면서 미처리 연산 관련 로그레코드인 경우, 하나의 PENDINFO 구조체를 확보하고 그 로그레코드에 수록된 미처리 연산 식별 정보를 추출하여 그 구조체에 수록한 후 이를 그 트랜잭션의 트랜잭션 엔트리의 Pending_Info 필드에 보관한다.

그림 10은 재시작 재수행 단계의 확장된 알고리즘이다. 그림 10의 의사코드에서 새로 추가한 부분과 변경한 부분을 표현하기 위해 첫째 칼럼에 *를 삽입하였다.

그림 10에서 GetSegmentNum() 함수는 페이지 식별자에서 그 식별자에 해당하는 페이지가 소속된 세그먼트의 세그먼트 번호를 추출하여 반환하는 기능을 수행한다.

```
* RESTART_REDO(RedoLSN,Trans_Table,Dirty_Pages)
  {
     Open_Log_Scan(RedoLSN);
     LogRec = Next_Log();
     WHILE NOT(End of Log) {
        IF (LogRec.Type == ('update'|'compensation') &&
            LogRec is redoable &&
            LogRec.PageID IN Dirty_ Pages &&
            LogRec.LSN >= Dirty_Pages[LogRec.PageID].RecLSN) {
            Page = fix&latch(LogRec.PageID, 'X');
            IF (Page.PageLSN < LogRec.LSN) {
               Redo_Update(Page, LogRec);
               Page.PageLSN = LogRec.LSN;
            } ELSE {
               Dirty_Pages[LogRec.PageID].RecLSN=Page.PageLSN+1;
            }
            unfix&unlatch(Page);
*           IF (LogRec.TransID IS IN Trans_Table &&
*              LogRec.LSN == Trans_Table[LogRec.TransID].LastLSN
*              && LogRec.Type is a pending action related type) {
*              Trans_Table[LogRec.TransID].Pending_Info=new
*                    PENDINFO;
*              PAInfo->PALIndex = LogRec.PALIndex;
*              PAInfo->LogType = LogRec.Type;
*              if (LogRec.Type == 'pa_freeEME') {
*                 PAInfo->SegmentNum=
*                          GetSegmentNum(LogRec.PageID);
*                 PAInfo->StartExtOnNextPage =
*                          LogRec.StartExtOnNextPage;
*              } else if (LogRec.Type == 'pa_clearFME') {
*                 PAInfo->SegmentNum=
*                          GetSegmentNum(LogRec.PageID);
*              } else ; /* 'pa_deleteFID' : do nothing */
*           }
        }
        LogRec = Next_Log();
     } /* WHILE */
  }
```

그림 10 확장된 재시작 재수행 단계의 의사코드

### 6.3 재시작 회복의 아직 수행되지 않은 미처리 연산의 수행 단계

재시작 취소 단계 후에 수행하는 아직 수행되지 않은 미처리 연산의 수행 단계는 각 미종료 트랜잭션에 대하여 재시작 분석 단계에서 복구한 그 트랜잭션의 미처리 연산 리스트와 재시작 재수행 단계에서 확보한 그 트랜잭션이 마지막으로 수행한 미처리 연산의 식별 정보를 바탕으로 그 트랜잭션의 아직 수행되지 않은 미처리 연산들의 수행을 완료하고 그 트랜잭션의 트랜잭션종료 로그레코드를 기록한 후 그 트랜잭션의 트랜잭션 엔트리를 초기화함으로써 그 트랜잭션을 종료시킨다.

아직 수행되지 않은 미처리 연산의 수행은 각 미종료 트랜잭션의 마지막 로그레코드에 따라 다르다.

첫째, 트랜잭션 엔트리의 Pending_Info 필드가 NULL인 경우. 이는 그 트랜잭션의 마지막 로그레코드가 'pa_start' 로그레코드임을 의미한다. 이 경우, 그 트랜잭션의 미처리 연산 리스트에 수록된 모든 화일들에 대해 화일 삭제의 미처리 연산을 수행한다.

둘째, 트랜잭션 엔트리의 Pending_Info 필드가 NULL이 아닌 경우. 이 경우에는 마지막으로 수행한 미처리 연산의 LogType에 따라 아래와 같이 아직 수행되지 않은 미처리 연산의 수행을 완료한다.

1. 'pa_freeEME'인 경우. 그 트랜잭션의 FilePAL[PAL Index]에 해당하는 화일에 대하여 그 화일이 소속된 볼륨의 SegmentNum이 가리키는 세그먼트에서 StartExtOnNextPage가 가리키는 익스텐트의 익스텐트 맵 엔트리를 선두로 하는 익스텐트 맵 엔트리 리스트에 연결된 익스텐트 맵 엔트리들을 초기화하는 과정부터 미처리 연산의 수행을 시작하여 그 트랜잭션의 나머지 아직 수행되지 않은 미처리 연산의 수행을 완료한다. 만약, StartExtOnNextPage가 NIL_EXT이면 그 세그먼트에서 그 화일의 화일 맵 엔트리를 초기화하는 작업부터 수행을 시작한다.

2. 'pa_clearFME'인 경우. 그 트랜잭션의 FilePAL[PALIndex]에 해당하는 화일에 대하여 그 화일이 소속된 볼륨에서 (SegmentNum+1)이 가리키는 세그먼트부터 마지막 세그먼트까지 차례로 탐색하면서 각 세그먼트에서 그 화일에 할당된 익스텐트들을 반환하는 연산부터 미처리 연산의 수행을 시작하여 그 트랜잭션의 나머지 아직 수행되지 않은 미처리 연산의 수행을 완료한다.

3. 'pa_deleteFID'인 경우. 그 트랜잭션의 FilePAL[PALIndex+1]에 해당하는 화일부터 FilePAL에 등록된 마지막 엔트리에 수록된 화일 식별자의 화일까지 그 화일들을 삭제하기 위한 미처리 연산들을 차례로 수행한다.

재시작 회복의 아직 수행되지 않은 미처리 연산 수행 단계의 알고리즘은 그림 11과 같다.[8)]

```
RESTART_MEDIA_PENDING(Trans_Table)
{
    for (each Trans_Table entry, TransID, whose State is 'PAStarted') {
        if (Trans_Table[TransID].Pending_Info == NULL) {
            Pending_Actions(TransID,Trans_Table,0);
        } else {
            PALIndex = SegmentNum = FileID = NIL;
            PAInfo = Trans_Table[TransID].Pending_Info;
            switch (PAInfo->LogType) {
              case 'pa_freeEME'   :
                    PALIndex = PAInfo->PALIndex;
                    SegmentNum = PALInfo->SegmentNum;
                    FileID = Trans_Table[TransID].FilePAL[PALIndex];
                    if (PAInfo->StartExtOnNextPage != NIL_EXT) {
                        Free_EM_Entries(TransID,FileID,SegmentNum,
                                PAInfo-> StartExtOnNextPage,PALIndex);
                    }
                  Clear_FM_Entry(TransID,FileID,SegmentNum,PALIndex);
              case 'pa_clearFME'  :
                    if (PALIndex != NIL) PALIndex = PAInfo->PALIndex;
                    if (SegmentNum != NIL)
                                SegmentNum = PAInfo-> SegmentNum;
                    if (FileID!= NIL)
                                FileID=Trans_Table[TransID].FilePAL [PALIndex];
                    Deallocate_Disk_Space(TransID,FileID,PALIndex,
                                SegmentNum+1);
                    Delete_FileID(TransID,FileID,PALIndex);
              case 'pa_deleteFID' :
                    if (PALIndex != NIL) PALIndex = PAInfo->PALIndex;
                    Pending_Actions(TransID,Trans_Table,PALIndex+1);
            } /* end of switch */
        } /* end of else */
        write 'end' log record and remove the entry from Trans_Table;
    } /* end of for */
}
```

그림 11 아직 수행되지 않은 미처리 연산 수행 단계의 의사코드

아직 수행되지 않은 미처리 연산 수행 단계에서 미종료 트랜잭션의 아직 수행되지 않은 미처리 연산들을 수행하면서 페이지들에 발생하는 변경 사항들에 대해 기록하는 로그레코드는 정상적인 트랜잭션 수행시의 로깅 방법과 동일하게 재수행 정보와 함께 미처리 연산 식별 정보를 수록한 redo-only 유형의 로그레코드를 기록한다. 이는 시스템 고장이 반복하여 발생하더라도 회복 과정에서 미처리 연산의 수행을 정확하고 신속하게 종료할 수 있게 하기 위함이다.

### 6.4 매체 재수행 단계의 확장

확장된 매체 재수행 단계는 ARIES의 기존 작업에 다음의 두 가지 사항을 추가한다. 첫째, 미종료 트랜잭션들의 수행상태를 복구하면서 'pa_start', 'prepare', 그리고 매체 복사 검사점의 검사점종료 로그레코드에 수록된 트랜잭션들의 미처리 연산 리스트를 해당 트랜잭션 엔트리들에 복구하며, 'pa_start' 로그레코드에 대해서는 트랜잭션의 완료상태를 PAStarted로 변경한다. 둘째, 재수행 단계의 마지막에 트랜잭션의 완료상태가 PAStarted인 각 미종료 트랜잭션이 생성한 마지막 로그레코드에서 미처리 연산 식별 정보를 추출하여 그 트랜잭션의 트랜잭션 엔트리의 Pending_Info 필드에 보관한다.

8) 이 알고리즘에서 사용된 switch 문은 C 언어의 switch 문과 동일한 기능을 가진다.

## 7. 성능 평가

본 절은 화일 삭제에 대한 본 논문의 미처리 연산 수행 기법과 Shore의 미처리 연산 수행 기법의 비교를 통해 본 논문이 제시하는 기법의 우수성을 보인다. 이들의 비교를 공정히 수행하기 위하여 Shore의 화일 관리 구조를 바탕으로 비교한다. 또한, 미처리 연산의 수행 기법에 중점을 두고 화일 삭제에 관련한 다른 세부 사항들을 간략히 하기 위하여 색인을 가지지 않는 데이타 화일의 삭제로 한정하며, 화일 삭제에 관련된 연산들 중에서 디스크 공간의 반환과 화일 식별자의 반환에 대해서만 다룬다. 이들의 비교를 로그레코드의 기록과 그 크기, 로크의 획득과 해제, 그리고 재시작 회복의 수행 기법의 세가지로 구분하여 수행한다.

### 7.1 로그레코드의 기록과 그 크기 비교

본 논문이 제시하는 화일 삭제 기법을 Shore의 저장 시스템 구조에 반영할 경우, 하나의 볼륨이 하나의 세그먼트만을 가지므로 별도의 화일 할당 테이블을 가질 필요가 없으며 화일 맵 엔트리의 번호가 그 엔트리에 해당하는 화일의 화일 번호로 동작한다. 이러한 환경에서, Shore와 바다-II가 화일 삭제에서 저장 시스템의 관리와 관련된 로그레코드의 비교는 아래의 표 2와 같다.

표 2 로그레코드의 기록

| 연산 | Shore | 바다-II |
|---|---|---|
| 화일 맵 엔트리에 화일삭제 표시 | 수행함 | 없음 |
| 전역 트랜잭션의 완료준비 로그레코드 | 'xct_prepare' (미처리 연산 리스트를 포함함) | 'prepare' (미처리 연산 리스트를 포함함) |
| 완료와 미처리 연산 수행의 시작 로그레코드 | 'xct_freeing_space' (미처리 연산 리스트를 포함 안함) | 'pa_start' (지역 트랜잭션에 한하여 미처리 연산 리스트를 포함함) |
| 화일 맵 엔트리에 익스텐트 반환 표시 | 수행함 | 없음 |
| 익스텐트 맵 엔트리의 초기화 | 수행함 | 수행함 |
| 화일 맵 엔트리의 초기화 | 수행함 | 수행함 |
| 검사점 종료 로그레코드 | 미처리 연산 리스트를 포함함 | 미처리 연산 리스트를 포함함 |

바다-II는 화일 맵 엔트리에 화일삭제 표시와 익스텐트 반환 표시를 하지 않으며 따라서 해당 로그레코드들을 기록하지 않는다. 바다-II는 'pa_start' 로그레코드에 지역 트랜잭션에 한하여 미처리 연산 리스트를 포함하며 익스텐트 맵 엔트리들을 초기화하면서 생성하는 로그레코드들에 미처리 연산 식별 정보를 추가로 로깅하지만 그 정보의 양이 많지 않기 때문에 시스템에 부담이 되지 않는다. 따라서, 시스템의 정상 수행시의 페이지 접근 횟수와 로깅 부담 면에서는 두 기법의 성능이 거의 같다.

### 7.2 로크의 획득과 해제 비교

Shore와 바다-II는 화일 삭제의 즉시 연산으로 그 데이타 화일에 대하여 배타적 모드로 트랜잭션 기간의 로크를 획득한다. Shore는 삭제할 화일에 대한 미처리 연산이 완료될 때까지 그 화일의 익스텐트 맵 엔트리 리스트를 유지하기 위하여 그 화일의 모든 익스텐트들에 대해 익스텐트 로크를 획득해야 하고 그 로크들을 최소한 그 화일의 화일 맵 엔트리를 초기화한 후에라야 해제할 수 있다. 예를들어, 하나의 데이타 화일이 1,000개의 페이지들로 구성되고 하나의 익스텐트가 8개의 페이지를 차지하는 경우, 그 화일을 삭제하기 위해서는 125개의 익스텐트들에 대한 익스텐트 로크를 획득해야 한다.

Shore의 이러한 익스텐트 로킹 기법은 (1) 익스텐트 로킹을 위한 별도의 로킹 테이블이 필요하며, (2) 익스텐트 로크의 설정과 해제에 따른 시간적 부담을 지니며, (3) 익스텐트의 할당시에도 할당된 익스텐트에 대한 순간 기간(instant duration)의 배타적 모드의 익스텐트 로크를 획득해야 하며, 그리고 (4) 삭제 중인 화일의 익스텐트를 할당받는 트랜잭션은 그 화일을 삭제하는 트랜잭션이 그 로크를 해제할 때 까지 블록되거나 다른 익스텐트를 재할당 받아야 하는 부담을 가진다.

바다-II의 화일 삭제 기법은 익스텐트 로킹을 하지 않으며 반환된 익스텐트들은 그 즉시 다른 화일에 할당될 수 있으며 익스텐트의 할당 시에도 할당받는 익스텐트에 대하여 익스텐트 로크를 획득할 필요가 없다. 그 이유는 삭제할 화일의 미처리 연산이 완료되지 않은 상태에서 그 화일의 익스텐트 맵 엔트리 리스트의 반환한 어떤 익스텐트가 다른 화일에 할당되더라도 회복시에 로그레코드들의 분석을 통해 아직 반환하지 않은 잔여 익스텐트 맵 엔트리 리스트를 확인할 수 있기 때문이다.

### 7.3 재시작 회복 기법의 비교

화일 삭제에 관련된 재시작 회복 기법의 비교는 아래의 표 3과 같다.

표 3 재시작 회복의 비교

| ARIES의 단계 | Shore | 바다-II |
|---|---|---|
| 분석 단계 | ARIES의 분석 단계 | ARIES의 분석 단계 + 미종료 트랜잭션들의 미처리 연산 리스트 복구 + RedoLSN의 조정 |
| 재수행 단계 | ARIES의 재수행 단계 | ARIES의 재수행 단계 + 미종료 트랜잭션들이 마지막으로 수행한 미처리 연산을 식별 |
| | 모든 볼륨의 모든 화일 맵 엔트리에 대하여 익스텐트 반환 표시에 대한 미처리 연산의 수행 | |
| 취소 단계 | ARIES의 취소 단계 | ARIES의 취소 단계 |
| | 모든 볼륨의 모든 화일 맵 엔트리에 대하여 화일 삭제 표시에 대한 미처리 연산의 수행 | 미종료 트랜잭션들의 아직 수행하지 않은 미처리 연산들을 수행 |

Shore는 미종료 트랜잭션의 존재 여부에 관계없이 재시작 재수행 단계 후에 미종료 트랜잭션들이 삭제 중이던 화일들을 식별하기 위하여 그리고 재시작 취소 단계 후에 미종료 트랜잭션들이 삭제하려 했던 화일들을 식별하기 위하여 각각 한 차례씩 모든 볼륨의 모든 화일 맵 엔트리들을 검사해야 하며, 식별된 각 화일 맵 엔트리에 대해 그 화일의 익스텐트 맵 엔트리 리스트를 모두 탐색해야 한다.

바다-II의 화일 삭제 기법은 회복시에 미종료 트랜잭션들에 의해 삭제 중이거나 삭제되어야 하는 화일들을 식별하여 화일 삭제의 미완료된 부분만을 수행함으로서 모든 볼륨의 모든 화일 맵 엔트리들을 검사하지 않는다.

## 8. 결 론

트랜잭션은 화일 삭제에 관련된 작업들을 미처리 연산으로 수행하여야 하며, 회복 프로세스는 미종료 트랜잭션들의 화일 삭제에 관련된 아직 수행되지 않은 미처리 연산들의 수행을 완료한 후 그 트랜잭션들을 정상 종료시켜야 한다. 본 논문은 화일 삭제의 이러한 요구조건을 만족시키기 위하여 트랜잭션, 검사점 그리고 회복 프로세스 간에 로깅을 이용한 미처리 연산의 인수인계 방법을 제안하였다.

본 논문의 화일 삭제 방법은 Shore의 화일 삭제 방법과 비교하여 다음의 두가지 중요 특징을 가진다. 첫째, 삭제할 화일에 할당된 익스텐트들을 반환하면서 이들에 대하여 익스텐트 로크를 획득하지 않으며 반환된 익스텐트들은 그 즉시 다른 화일에 할당될 수 있으며 화일

에 익스텐트를 할당할 때에도 할당받는 익스텐트에 대하여 익스텐트 로크를 획득할 필요가 없다. 둘째, 회복 프로세스는 미종료 트랜잭션들에 의해 삭제 중이거나 삭제되어야 하는 화일들을 재시작 회복의 분석 단계와 재수행 단계에서 식별하여 재시작 회복의 취소 단계 후에 화일 삭제의 미완료된 부분만을 수행한다.

본 논문에서 제시한 미처리 연산의 수행 기법은 화일의 삭제 외에도 트랜잭션의 수행 중 많은 양의 데이타를 로깅해야 하고 트랜잭션의 복귀 시에 취소 부담이 큰 연산들에 대하여 그 연산들을 트랜잭션의 미처리 연산으로 수행하는 데 유사하게 적용될 수 있다.

## 참 고 문 헌


[1] M. O. Chae, K. H. Hong, M. Y. Lee, J. Kim, O. J. Cho, S. T. Jun and Y. K. Kim, "Design of the Object Kernel of BADA-III: An Object-Oriented Database Management System for Multimedia Data Services," *The 1995 Workshop on Network and Systems Management,* August 1995.

[2] M. Carey et. al., "Object and File Management in the EXODUS Extensible Database System," *In Proceedings 12th International Conference on Very Large Data Bases,* pp. 91-100, 1986.

[3] M. Carey et. al., "Shoring Up Persistent Applications," *In Proceedings ACM SIGMOD International Conference on Management of Data,* Vol. 23, No. 2, pp. 383-394, 1994.

[4] M. Chou et. al., "Design and Implementation of the Wisconsin Storage System," *Software Practice and Experience,* Vol. 15, No. 10, pp. 943-962, 1985.

[5] R. Dey, M. Shan, and I. Traiger, "Method for dropping data sets," *IBM Tech. Disclosure Bull. 25,* 11A, pp. 5453-5455, April 1983.

[6] J. Gray and A. Reuter, *Transaction Processing: Concepts and Techniques,* Morgan Kaufmann Publishers, Inc., pp. 851-886, 1993.

[7] *INFOMIX-Universer Server Administrator's Guide,* INFORMIX Press, 1997.

[8] W. Kim, *Introduction to Object-Oriented Databases,* The MIT Press, 1990.

[9] C. Mohan, D. Haderle, B. Lindsay, H. Pirahesh, and P. Schwarz, "ARIES: A Transaction Recovery Method Supporting Fine-Granularity Locking and Partial Rollbacks Using Write-Ahead Logging," *ACM Transactions on Database Systems,* Vol. 17, No. 1, 1992.

[10] *Microsoft SQL SERVER Administrator's Companion 6.0,* Microsoft Corporation.

[11] C. Mohan and F. Levine, "ARIES/IM: An Efficient and High Concurrency Index Management Method Using Write-Ahead Logging," *In Proceedings ACM SIGMOD International Conference on Management of Data,* pp. 371-380, June 1992.

[12] 박성철, 박영철, 김완석, "바다-II의 트랜잭션 처리", *데이타베이스 연구회지,* 제 12권 4호, pp. 76-96, 1996년 12월.

[13] 박영철, 김강호, "바다-II의 배타적 래취와 공유 래취의 설계 및 구현", *데이타베이스 연구회지,* 제 11권 2호, pp. 111-133, 1995년 7월.

[14] 박영철, 박준현, 김준, 이진수, "허상 트랜잭션: 퍼지 검사점에 따른 회복의 문제점", *정보과학회논문지(B),* 제 25권 3호, pp. 426-443, 1998년 3월.

[15] Y. C. Park, J. H. Park and D. Y. Huh, "Mini-Savepoints: Firewall for Atomic Updates," *In Proceedings 5th International Conference on Database Systems For Advanced Applications,* Melbourne, Australia, pp. 293-302, April 1997.

[16] 박준현, 박영철, "상태미반영 로그레코드: 퍼지 검사점과 트랜잭션의 비등기 수행에 따른 회복의 문제점", *정보과학회논문지(B),* 제26권 5호, pp. 621-638, 1999년 5월.

[17] 박준현, 박영철, "바다-II에서 저장점과 부분 복귀의 설계와 구현", *정보과학회논문지: 데이타베이스,* 제 27권 3호, pp.578-600, 2000년 9월.

[18] 박준현, 박영철, 이진수, "$B^+$-트리에서 키와 구분자의 저장과 탐색", *정보과학회논문지(C),* 제 3권 6호, pp. 568-580, 1997년 12월.

[19] 박준호, 박성철, 심광훈, 성준화, 박영철, "GIS 응용을 위한 바다-III의 다단계 사전인출과 지연쓰기의 설계 및 구현", *한국지리정보학회지,* 제 1권 2호, pp. 67-79, 1998년 12월.

[20] P. J. Peterson and J. P. Strickland, "Log Write-Ahead Protocols and IMS/VS Logging," *In Proceedings 2nd ACM SIGACT-SIGMOD Symposium on Principles of Database Systems,* Atlanta, March 1983.

[21] *UniSQL/X User's Manual,* UniSQL, Inc., 1996.



박 준 현

1995년 대구대학교 전자계산학과 졸업(학사). 1997년 경북대학교 컴퓨터과학과 졸업(석사). 2000년 8월 경북대학교 컴퓨터과학과 졸업(박사). 2000년 8월 ~ 현재 한국컴퓨터통신(주) 연구소. 관심분야는 저장 시스템, 멀티미디어 데이타베이스 시스템

박 영 철

1977년 서울대학교 전기공학과 졸업(학사). 1981년 서울대학교 전기공학과 졸업(석사). 1986년 Northwestern University(전산학, 석사). 1989년 Northwestern University(전산학, 박사). 1977년 ~ 1979년 육군 병장 전역. 1981년 ~ 1993년 울산대학교 전자계산학과 부교수. 1990년 ~ 1992년 울산대학교 중앙전자계산소 소장. 1991년 ~ 1992년 한국전자통신연구소 초빙연구원. 1992년 ~ 1994년 한국정보과학회 데이타베이스연구회 운영위원. 1993년 ~ 현재 경북대학교 컴퓨터과학과 교수. 관심분야는 데이타베이스 시스템, 멀티미디어 데이타베이스, 객체지향 데이타베이스, 트랜잭션 처리 등임