2008 한국정보처리학회 논문지 제7권 제7호(2000.7)

고차원 색인구조를 위한 회복기법의 설계 및 구현

송 석 일† · 이 석 희†† · 유 재 수†††

요 약

이 논문에서는 재삽입 연산을 사용하는 고차원 색인 구조를 위한 효과적인 회복기법을 제안한다. 제안하는 회복기법은 시스템 고장이나 트랜잭션 고장과 같은 다양한 고장이 발생할 때 재삽입 연산을 포함하는 삽입연산의 트랜잭션이 효과적으로 재수행과 복귀를 수행하도록 한다. 제안하는 회복기법은 WAL(Write Ahead Logging)프로토콜을 기반으로 한다. 또한, 제안하는 회복기법을 CIR-트리에 적용시켜서 순수 국내기술로 개발된 멀티미디어 DBMS BADA-Ⅲ의 하부저장구조인 MiDAS-Ⅲ에 구현한다. 이 논문에서 제안하는 회복 기법과 제안한 알고리즘을 적용하지 않은 방법과의 성능 비교를 통해서 제안하는 회복 기법이 재삽입 연산에 대해서 보다 효율적인 회복을 수행한다는 것을 보인다.

Design and Implementation of a Recovery Method for High Dimensional Index Structures

Seok-Il Song† · Seok-Hee Lee†† · Jae-Soo Yoo†††

ABSTRACT

In this paper, we propose a recovery method for high dimensional index structures. It recovers efficiently transactions including reinsert operations that needs undo or rollback due to system failures or transaction failures. It is based on WAL(Write Ahead Logging) protocol. We apply the method to the CIR-Tree and implement it based on MiDAS-Ⅲ which is the storage system of a multimedia DBMS, called BADA-Ⅲ. We also show through performance evaluation that the recovery method with our algorithm recovers reinsert operations efficiently over that without our algorithm.

1. 서 론

분자생물학, 의학 분야, CAD/CAM 시스템, VOD, 멀티미디어 데이터베이스 등 다양한 응용분야에서 발생하는 이미지 데이터의 양은 급속도로 증가되고 있다. 이러한 방대한 양의 이미지 데이터들을 대상으로 내용기반 이미지 검색이 요구되고 있다. 내용기반 이미지 검색 시스템에서 중요한 문제는 검색의 효율성이며, 이를 위해 적절한 색인 기법을 설계해야 한다. 내용기반 이미지 검색을 위한 전형적인 색인 구조는 이미지 데이터로부터 고차원의 특징 데이터를 추출한 후, 추출된 특징 데이터를 고차원 공간상의 하나의 점으로 간주하여 이를 색인하는 방법을 주로 사용한다.

지난 수년 동안 내용기반 이미지 검색을 위해 고차원 특징벡터를 기반으로 하는 유사성 검색이 데이터베이스 분야에서 매우 중요한 연구 과제로 부각되어 왔다. 이런 유사성 검색의 가장 핵심적인 문제 중 하나가 바로 수많은 다차원의 특징벡터들 중에서 어떻게 질의와 유사한 것들을 효율적으로 찾아내는가 하는 것이다. 이 문제를 해결하기 위해 고차원 색인구조에 대한 연구가 매우 활발히 진행되어 왔고 수많은 색인 구조들이 제안되었다.

기존의 고차원 색인 구조들이 실제 응용에 사용되기

※ 이 논문은 과학재단 특정기초과제(과제번호 : 1999-1-303-007-3) 연구비 지원에 의해 수행되었음.
† 준 회 원 : 충북대학교 대학원 정보통신공학과
†† 정 회 원 : 동아방송대학 인터넷방송과 교수
††† 종신회원 : 충북대학교 전기전자공학부 교수
논문접수 : 2000년 1월 25일, 심사완료 : 2000년 6월 10일

위해서는 색인 구조에 대한 적절한 회복 기법이 제공되어야 한다. 그러나 기존에 제안된 색인구조에 대한 회복 기법은 대부분 B-트리 계열을 위한 회복기법[3]이고, 고차원 색인구조에 대한 회복 기법으로는 참고문헌 [6, 7]정도이다. 일반적으로 고차원을 대상으로 하는 CIR-Tree[1], R*-Tree[8], SR-Tree[9], TV-Tree[5], SS-Tree[4]와 같은 색인구조에서는 재삽입 연산을 통해 색인 구조를 보다 효율적으로 구성하여 탐색 성능을 향상시키고 있다. 재삽입 연산이란 특정 노드에 넘침이 발생했을 때 이를 바로 분할하지 않고 그 노드의 중심점에서 멀리 떨어져 있는 엔트리들을 색인구조에서 일시적으로 삭제하고 다시 삽입하는 연산을 말한다. 재삽입 연산에 관해서는 참고문헌 [8]에서 여러 실험을 통해 재삽입 연산에 의한 색인의 성능향상을 입증하였다. 그러나 이에 대한 회복 기법에 관련된 연구는 전혀 없다.

재삽입 연산은 삽입 연산의 일부로써 동작하게 된다. 즉, 하나의 트랜잭션이 객체를 색인 구조에 삽입할 때 재삽입 연산은 그 트랜잭션에 포함되는 것이다. 따라서 어떤 트랜잭션이 재삽입 연산을 수반하는 삽입 연산을 수행할 때에 시스템 고장(System Failure) 이나 트랜잭션 고장(Transaction Failure)이 발생하면 수행되었던 재삽입 연산은 모두 복귀(Undo)되어야 한다. 재삽입 연산은 여러 개의 삽입 연산을 모아 놓은 것이므로 트랜잭션 철회를 수행할 때 다른 연산들을 지연시키고 재시동 회복 시에 회복시간을 길게 한다.

이 논문은 재삽입 연산을 포함하는 트랜잭션이 시스템 고장이나 트랜잭션 고장으로 인해 재수행 또는 복귀가 필요할 경우 이를 효율적으로 수행되도록 하는 회복기법을 제안한다. 이 논문에서 제안하는 회복기법은 기본적으로 WAL(Write Ahead Logging)프로토콜[2]을 기반으로 하고 있으며 재수행과 복귀를 수행한다. 제안하는 회복기법을 순수 국내기술로 개발된 멀티미디어 DBMS인 BADA-Ⅲ의 하부저장구조 MiDAS-Ⅲ에 CIR-Tree를 바탕으로 구현한다.

이 논문의 구성은 다음과 같다. 먼저 2장에서 기존 연구로 다차원 색인구조의 회복기법에 대해 분석하고 3장에서 제안하는 회복기법에 대해 자세히 기술한다. 4장에서는 제안하는 회복기법의 구현 내용을 기술하고, 5장에서는 제안하는 알고리즘에대한 성능 평가를 수행한다. 그리고, 5장에서 결론을 맺는다.

2. 기존의 다차원 색인구조의 회복기법

이 장에서는 기존에 제안되었던 다차원 색인구조를 위한 회복기법에 대해 기술한다. 기존의 다차원 색인구조를 위한 회복기법은 주로 WAL(Write Ahead Logging)을 기반으로 하여 페이지 지향 재수행(Page-Oriented Redo)과 논리적 복귀(Logical Undo)를 수행한다. 논리적 복귀라 함은 복귀를 수행할 때 색인 구조를 개방하고 루트부터 순회하여 해당하는 노드를 찾은 후 작업을 수행하는 것을 말한다. 반면에 페이지 지향 복귀/재수행은 색인 구조를 개방하지 않고 로그 레코드에 기록되어 있는 노드에서 작업을 수행하는 것을 말한다.

참고문헌 [7]에서 제시하는 회복기법은 WAL(Write Ahead Logging)을 기반으로 하며 페이지 지향 재수행(Page-Oriented Redo)과 논리적 복귀(Logical Undo)를 기본으로 한다. 참고문헌 [7]에서 제안한 회복기법에서는 연산들을 단말 노드에 엔트리를 삽입하고 삭제하는 내용 변경 연산과 노드 분할 및 노드 삭제, 비단말 노드의 MBR 변경을 포함하는 트리 구조 변경 연산으로 나누어 다루고 있다. 변경 연산들을 둘로 분리하여 트리 구조 변경 연산은 그것을 발생시킨 트랜잭션과 독립적으로 다루어진다. 즉, 트랜잭션 수행 중에 트리 구조 변경 연산이 발생하여 이에 대한 수행을 마치면 트리 구조 변경 연산에 참여했던 노드들의 잠금을 해제해서 다른 트랜잭션들이 이 변경 내용을 볼 수 있게 한다. 이것은 동시성 성능을 높이는 효과가 있다.

엔트리 삽입이나 삭제로 인한 단말 노드의 변경은 트랜잭션이 철회(Rollback)되면 항상 복귀되고 트랜잭션이 완료되면 재수행 된다. 이런 반면, 트리 구조 변경 연산은 별개의 회복 가능한 단위(Atomic Actions)로 취급한다. 트리 구조 변경 연산을 별개의 단위로 취급하여 트랜잭션들 사이의 복귀 의존 관계를 만들지 않고 이를 포함한 트랜잭션이 완료(Commit)되지 않더라도 다른 트랜잭션들이 변경 내용을 볼 수 있게 한다. 즉, 트리 구조 변경 연산(노드 분할 등)에 참여한 페이지들을 그 트랜잭션이 완료 될 때까지 잠금 상태로 둘 필요가 없다는 것이다. 이를 지원하기 위해 참고문헌 [7]에서는 논리적인 복귀를 수행한다. 예를 들면, 한 트랜잭션이 엔트리를 단말 노드에 삽입하고, 이어서 그 단말 노드가 분할되어 그 트랜잭션이 삽입된 엔트리가 새로 생성된 노드로 옮겨진 경우를 생각해 보자. 후에 삽입을 일으킨 트랜잭션이 취소가 되어 페이지 복귀를 시도할 때 삽입된 엔트리는 원래 단말 노드에서 다른 노드로 옮겨졌기 때문에 트랜잭션은 그

엔트리를 찾을 수 없게 된다. 따라서 트리를 다시 순회하면서 엔트리 삭제연산을 수행해야 한다.

참고문헌 [6]에서 제시하는 회복 기법 역시 WAL 규약에 기반하며 페이지 지향 재수행과 논리적 복귀를 기본으로 한다. 또한 변경연산을 노드 삭제 및 MBR 변경을 포함하는 트리 구조 변경 연산과 단순 변경 연산으로 나누어 처리한다. 하지만, 참고문헌 [6]에서 제안한 방법은 참고문헌 [7]에서 제안한 방법과는 다르게 트리 구조 변경연산을 NTA(Nested Top Action)형태로 처리한다. 이들간의 가장 큰 차이점은 참고문헌 [7]에서는 트리 구조 변경연산의 범위가 단말/비단말 노드의 분할에서 끝나지만 참고문헌 [6]에서는 단말 노드의 분할과 분할된 내용을 상위노드에 반영하면서 발생할 수 있는 다른 노드 분할을 포함한다. 즉, 참고문헌 [6]에서는 트리 구조 변경 연산이 진행되다가 시스템 고장(System Failure)이 발생하면 변경 내용이 복귀되어야 한다. 트리 구조 변경 연산은 그 연산이 완료되는 데로 다른 트랜잭션들이 이것을 볼 수 있게 된다. 즉, 트리 구조 변경 연산이 완료되는 데로 여기에 참여했던 잠금들을 해제하여 동시성 성능을 높인다.

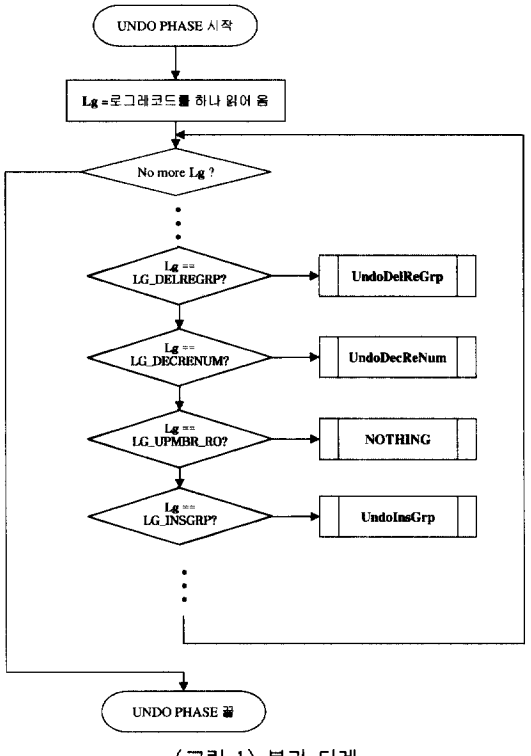
이상의 연구에서 제시하고 있는 회복 기법은 WAL 규약에 기반하며 페이지 지향 재수행과 논리적 복귀를 수행한다. 또한 색인구조에서 발생하는 변경연산들을 트리 구조 변경연산과 일반 내용 변경연산들로 나누고 트리 구조 변경연산의 경우 NTA나 Atomic Action으로 처리하고 이 트리 구조 변경연산의 수행이 완료되면 다른 트랜잭션들이 이 변경 내용을 볼 수 있게 하여 동시성 성능을 높이고 있다. 하지만 이들 회복 기법은 재삽입 연산을 효과적으로 처리할 수 있는 어떠한 방법도 제시하고 있지 않다. 재삽입 연산을 구성하는 하나 하나의 삽입연산에 모두 이와 같은 회복 기법을 적용하면 재삽입 연산을 수행하던 연산이 철회되거나 시스템 고장으로 인한 복귀시에 매우 많은 시간이 걸릴 것이다.

3. 재삽입 연산을 고려한 회복기법 설계

이 장에서는 회복을 위해 트리의 변경연산들을 분류하고 각각의 연산에 대해 로그 레코드의 내용과 그에 대한 복귀 및 재수행 동작을 설명한다.

3.1 개 요

이 논문에서 제시하는 회복 기법은 기본적으로 ARIES (Algorithm for Recovery and Isolation Exploiting Semantics)에 기반하고 있으며, 페이지 지향 재수행(Page Oriented Redo)과 페이지 지향 복귀(Page Oriented Undo)를 그 기본으로 한다. 상황에 따라 노드 지향 복귀가 수행될 수 없을 때는 논리적 복귀(Logical Undo)를 수행한다. 전체적인 회복 과정은 ARIES에서 제안하는 것처럼 로그레코드 분석단계, 재수행 단계, 복귀 단계로 이루어진다는 것을 가정한다. 로그 레코드 분석단계에서는 회복을 수행할 가장 오래된 로그 레코드를 결정하고 재수행 단계에서는 그 로그 레코드부터 가장 최근의 로그 레코드까지 재수행을 수행한다. 그리고 마지막으로 완료되지 않은 트랜잭션들을 복귀시키게 된다. (그림 1)에서 복귀 단계를 보여 주고 있다. 로그 레코드를 가장 최근의 것부터 후 방향으로 하나씩 읽어 가면서 그 로그 레코드의 종류에 따라 그에 대응되는 적절한 복귀 동작을 수행한다.



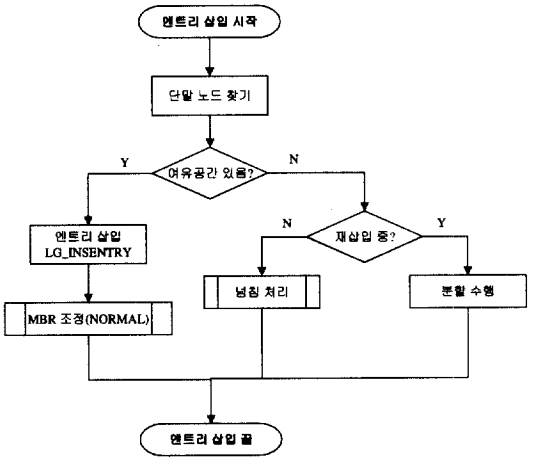
(그림 1) 복귀 단계

이 논문에서 제시하는 회복 기법에서는 변경 연산들을 다음과 같이 분류한다. 먼저 하나의 노드에 엔트리를 삽입하는 일과 삭제하는 일이 있다. 이것들은 ***일반***

변경연산으로 구분한다. 엔트리 삽입이나 삭제로 인해 노드의 MBR(Minimum Bounding Region)이 변경되어 이를 상위노드에 반영하는 일을 ***MBR 변경 연산***으로 구분한다. 마지막으로 한 노드에 넘침이 발생하여 이를 재삽입과 분할로 처리하는 것을 ***넘침 처리 연산***이라한다. 이 넘침 처리 연산에는 노드를 분할하는 **노드 *분할 연산***이 있고 노드의 엔트리들중 일부를 삭제하고 이로 인해 변경된 MBR을 상위에 반영하고 삭제한 엔트리들을 하나씩 다시 삽입하는 ***재삽입 연산***이 있다.

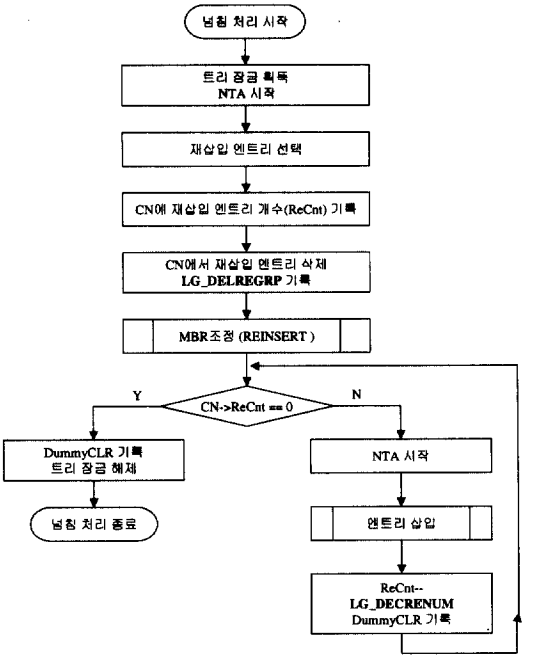
3.2 삽입 연산의 수행 절차

재삽입을 수행하는 고차원 색인 구조에서의 엔트리 삽입은 (그림 2)와 같이 수행된다. (그림 2)를 보면 먼저 트리를 순회하면서 삽입할 새로운 엔트리가 삽입될 가장 적절한 위치를 찾는다. 일단 새 엔트리가 삽입될 단말 노드를 찾으면 그 단말 노드가 새 엔트리를 수용할 수 있는지 없는 지를 판단한다. 새로운 엔트리를 수용할 공간이 없으면 넘침이 발생하게 되고 (그림 3)의 넘침 처리 모듈을 통해 넘침 처리를 시작하게 된다. 넘침 처리를 수행 할 때는 바로 현재 노드를 분할하지 않고 먼저 재삽입을 수행한다. 재삽입 엔트리를 선택하여 이를 삭제하고 삭제로 인한 MBR 변경을 (그림 4)의 MBR 조정 모듈을 통해 조상 노드들에 반영한다. 그리고 삭제한 재삽입 엔트리들을 하나씩 색인 구조에 삽입한다.

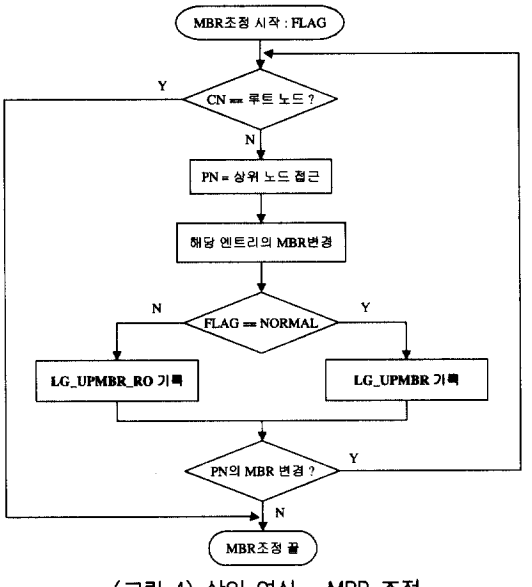


(그림 2) 삽입 연산

재삽입 도중 발생하는 넘침은 모두 분할로 처리한다. 재삽입을 수행해도 현재 노드에 여전히 넘침이 발생하면 분할을 수행한다. 분할 수행으로 생성된 노드의 정보를 부모 노드에 삽입할 때 다시 넘침이 발생하면 역시 앞에서와 같이 재삽입을 수행하고 수행한 결과 다시 넘침이 발생하면 위의 일을 다시 반복하고 삽



(그림 3) 삽입연산 - 넘침 처리



(그림 4) 삽입 연산 - MBR 조정

입을 종료한다. 새로운 엔트리를 삽입할 단말노드에서 새 엔트리를 수용할 수 있으면 이를 단말 노드에 삽입하고 이로 인해 노드의 MBR이 변경되면 (그림 4)의 MBR 조정 모듈을 통해 조상노드들에 변경된 MBR을 반영하고 삽입을 종료한다.

(그림 3)에서 재삽입이 수행되는 부분을 보면 삭제한 엔트리들이 모두 삽입 모듈을 통해 삽입되는 것을 볼 수 있다. 재삽입 엔트리 하나의 삽입이 재삽입을 수행하지 않는 삽입 연산과 같기 때문에 재삽입 연산이 수행되는 도중에 트랜잭션 고장이나 시스템 고장이 발생하면 각각의 삽입으로 인한 모든 변경이 복귀되어야 한다. 이것은 시스템 재시작의 속도를 느리게 하며 트랜잭션 철회시 다른 트랜잭션들에 영향을 미치게 되어 탐색연산이나 다른 삽입 연산들을 지연시킨다. 이 논문에서는 재삽입 연산으로 인한 변경을 모두 복귀하지 않고도 색인구조의 일관성을 유지할 수 있는 방법을 제시한다.

3.3 일반 변경 연산

일반 변경 연산은 루트 노드를 포함한 비단말 노드나 단말 노드에 엔트리를 하나 삽입하거나 삭제하는 연산을 말한다. 모든 경우에 노드 지향 복귀와 재수행을 수행한다. 이것은 한 트랜잭션이 노드에 엔트리를 삽입하거나 삭제하면 그 트랜잭션이 종료되기 전에는 절대로 다른 트랜잭션이 그 노드에 엔트리를 삽입하거나 삭제하지 못하게 하기 때문이다. 다음은 일반 변경 연산을 위해 정의된 로그 레코드에 포함되는 내용과 복귀 및 재수행 동작 과정을 설명한다.

3.3.1 삽입 로그 레코드(LG_INSENTRY)

- 로그 레코드의 용도
  단말 또는 비단말 노드에 엔트리를 삽입할 때 기록하는 로그 레코드
- 로그 레코드의 포함 내용
  엔트리를 삽입한 노드 식별자, 노드내에서 삽입한 엔트리의 식별자
- 복귀/재수행 방법
  노드 지향 복귀 및 재수행

3.3.2 삭제 로그 레코드(LG_DELENTRY)

- 로그 레코드의 용도
  단말 또는 비단말 노드에 엔트리를 삭제할 때 기록하는 로그 레코드
- 로그 레코드의 포함 내용
  엔트리를 삽입한 노드 식별자, 삭제한 엔트리
- 복귀/재수행 방법
  노드 지향 복귀 및 재수행

3.4 MBR 변경 연산

MBR 변경연산은 한 노드에 엔트리를 삽입하여 이 노드의 MBR이 변경되었을 때 상위 노드에 이를 반영하고 상위 노드의 MBR이 변경되었으면 다시 그 상위 노드에 반영하는 작업을 말한다. 이런 일련의 작업들은 하나의 NTA(Nested Top Action)로써 처리한다. 트랜잭션 고장이나 시스템 고장이 발생했을 때 완료되지 않은 MBR 변경 연산은 노드 지향으로 복귀되며, 완료된 MBR 변경 연산은 고장이 발생해도 복귀되지 않는다. 이 작업에 대한 복귀와 재수행은 모두 노드 지향 형태로 처리된다. 다음은 MBR 변경 연산을 위해 정의된 로그 레코드에 포함되는 내용과 복귀 및 재수행 동작 과정을 설명한다.

3.4.1 MBR 변경 로그 레코드(LG_UPMBR)

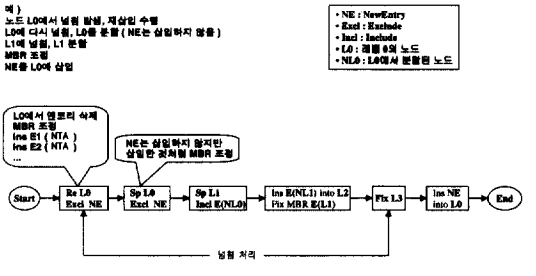
- 로그 레코드의 용도
  노드에 엔트리를 삽입하거나 삭제함으로 인해 MBR의 변경이 발생하여 이 내용을 상위 노드에 반영할 때 기록하는 로그레코드
- 로그 레코드의 포함 내용
  MBR이 변경된 노드의 식별자, 변경되기 전의 MBR, 변경된 후의 MBR
- 복귀/재수행 방법
  노드 지향 복귀 및 재수행, 복귀 시에는 MBR을 변경되기 전의 MBR로 변경시키고, 재수행 시에는 변경된 후의 MBR로 변경

3.5 넘침 처리 연산

넘침 처리 연산은 노드에 엔트리를 삽입할 때 여유 공간이 없어 재삽입과 분할로 이를 처리하는 작업을 말한다. 넘침 처리 연산은 하나의 NTA로 처리된다. 즉, 넘침처리를 위해 수행되는 모든 노드 분할 및 이로 인한 MBR 변경과 재삽입 수행은 하나의 NTA안에서 처리된다. 이 NTA안에서 수행되는 재삽입 연산의

각각의 삽입 역시 NTA로 처리된다. 완료된 넘침처리 연산은 트랜잭션이 완료(Commit)되기 전에 시스템 고장이나 트랜잭션 고장이 발생하여도 복귀되지 않는다. 완료되지 않은 넘침 처리 연산은 다음과 같이 복귀된다. 먼저, 노드의 분할과 그로 인한 MBR 변경은 노드 지향 복귀된다. 하지만 재삽입 연산은 모두 복귀되지 않고 일부만 복귀된다. 이미 삽입된 재삽입 엔트리는 복귀되지 않는다.

(그림 5)에서 넘침 처리 연산이 수행되는 것을 보여준다. (그림 5)에서 보는 바와 같이 노드에 넘침이 발생하면 이를 재삽입으로 처리하며 이것이 넘침 처리의 시작이며 NTA의 시작이다. 재삽입을 수행해도 여전히 현재 노드에 넘침이 발생하면 노드를 분할한다. 분할된 노드를 상위 노드에 반영하고 분할로 인한 MBR 변경을 역시 상위노드에 반영하고 더 이상 넘침이 발생하지 않으면 종료한다. 이때 NTA 역시 종료하게 된다. (그림 5)에서 처럼 최초에 단말 노드에 넘침을 발생시킨 엔트리의 삽입은 NTA안에 포함시키지 않고 NTA가 완료된 후에 삽입한다. 이것이 NTA안에 포함되면 트랜잭션이 완료되지 못했을 때 복귀가 되지 않기 때문이다.



(그림 5) 넘침 처리 연산의 예

3.5.1 재삽입 연산

앞서 설명한 대로 재삽입 연산은 넘침 처리 연산이라는 NTA의 일부이다. 재삽입 연산은 다음과 같은 일련의 작업들로 이루어진다. 먼저 넘침이 발생하면 그 노드에서 일부 엔트리들(재삽입 엔트리)을 삭제하고 그 재삽입 엔트리들의 수를 노드의 헤더 영역에 기록한다. 그리고 삭제로 인한 MBR 변경을 그 노드의 조상 노드들에 반영한다. 마지막으로 재삽입 엔트리들을 하나씩 색인 구조에 삽입한다. 이때 각각의 재삽입 엔트리의 삽입은 NTA로 처리된다. 즉 완료된 재삽입 엔트리 하나의 삽입은 그것을 포함하는 트랜잭션이 완료되기 전에 고장이 발생하여도 복귀되지 않는다. 하나의 재삽입 엔트리 삽입이 완료되면 노드의 헤더영역에 저장해 놓은 재삽입 엔트리의 수를 하나 감소시킨다. 이것은 하나의 재삽입 엔트리 삽입이 완료되면 각종 고장이 발생해도 복귀되지 않게 하여 회복의 양을 줄이기 위함이다.

재삽입 연산 수행 중에도 MBR 변경연산이 수행되는데 이때의 MBR 변경연산은 재삽입 연산을 포함한 NTA가 완료되지 않은 상태에서 시스템 또는 트랜잭션 고장이 발생하였을 때 노드 지향 형태로 복귀되지 않는다. 이유는 재삽입 연산을 수행하게 되면 재삽입 엔트리들이 색인 구조에 삽입되고 이로 인해 재삽입 엔트리 삭제로 인해 축소되었던 MBR은 다시 확장 될 수 있다. 따라서 이를 노드 지향 형태로 복귀하게 되면 MBR이 일관성을 갖지 못한다. 제안하는 회복 기법에서는 재삽입 연산 중에 발생한 MBR 변경은 재삽입 엔트리의 삭제를 복귀하면서 재조정하게 된다. 다음에서 재삽입 연산을 수행하면서 기록하는 로그 레코드들과 그 각각의 복귀/재수행 동작에 대해서 설명한다.

① 재삽입 엔트리 삭제 레코드(LG_DELREGRP)

- 로그 레코드의 용도
  단말 또는 비단말 노드에서 선택된 재삽입 엔트리들을 삭제할 때 기록하는 로그레코드

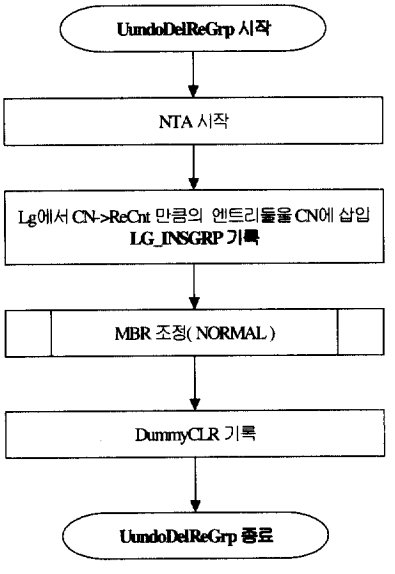
- 로그 레코드의 포함 내용
  재삽입이 발생한 노드식별자, 삭제한 엔트리들, 엔트리들의 개수, 경로 스택

- 복귀/재수행 방법
  노드 지향 복귀 및 재수행

앞에서 설명한 바와 같이 이 로그 레코드를 복귀할 때에 항상 삭제한 모든 엔트리들을 다시 삽입하지는 않는다. 복귀 시에는 먼저 그 노드의 헤더 영역의 재삽입될 엔트리의 개수를 보고 노드에 다시 삽입할 엔트리의 개수를 결정한다. 이렇게 엔트리를 삽입하고 나면 이로 인한 MBR 변경을 조상 노드들에 반영한다. 이 로그 레코드의 복귀 과정은 NTA로 처리되며 NTA의 끝을 알리는 Dummy CLR은 이 로그 레코드의 Prev-LSN을 가리키게 된다.

재삽입 엔트리 삭제 레코드(LG_DELREGRP)에 대한 복귀 동작을 (그림 6)에서 보여 주고 있다. 이 로그 레코드의 복귀는 NTA로 처리되므로 먼저 NTA의 시작

을 알린다. 그리고 복귀가 수행될 노드(CN)에서 현재 재삽입 되지 않고 남아 있는 엔트리들의 개수(ReCnt)를 읽어 온다. 로그 레코드에 기록된 *삭제한 엔트리들* 중에서 ReCnt만큼만 다시 CN에 삽입한다. 삽입한 후 변경된 MBR을 (그림 4)의 MBR 변경 모듈을 통해서 조상노드들에 반영한다.



(그림 6) LG_DELREGRP의 복귀

② 재삽입 엔트리 수의 변경 레코드(LG_DECRENUM)

- 로그 레코드의 용도
  재삽입이 수행되고 있는 단말 또는 비단말 노드의 첫번째 레코드에 유지되는 삽입되지 않고 남아있는 재삽입 엔트리의 수를 하나 감소시킬때 기록하는 로그레코드

- 로그 레코드의 포함 내용
  노드의 식별자

- 복귀/재수행 방법
  노드 지향 복귀 및 재수행
  이 로그 레코드는 재삽입 엔트리 삽입 NTA에 포함되어 완료된 재삽입 엔트리 삽입에 대해서는 복귀하지 않고 그 개수를 노드에 유지하여 LG_DELREGRP 로그 레코드를 복귀 할 때 사용한다.

③ 재삽입 엔트리 삭제로 인한 MBR 변경 레코드(LG_UPMBR_RO)

- 로그 레코드의 용도
  재삽입 엔트리들을 삭제하여 변경한 MBR을 상위에 반영할 때 기록하는 로그 레코드

- 로그 레코드의 포함 내용
  노드의 식별자, 변경된 후의 MBR

- 복귀/재수행 방법
  재수행만 수행, 변경된 후의 MBR로 MBR을 변경

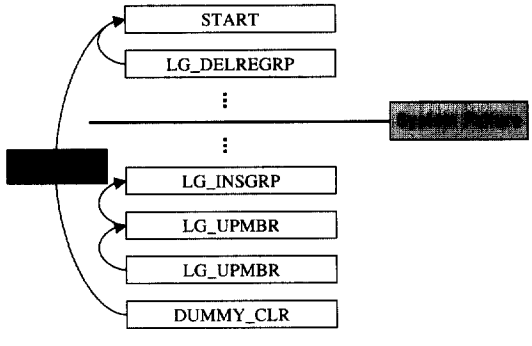
LG_DELREGRP 로그 레코드를 복귀하여 엔트리들을 다시 재삽입이 수행되던 노드에 삽입할 때 이에 대한 MBR을 조상 노드들에 반영한다. 재삽입으로 인해 MBR이 변경되었을 수 있으므로 복귀를 수행하지 않는다.

④ 재삽입 복귀 로그 레코드(LG_INSGRP)

- 로그 레코드의 용도
  LG_DELREGRP를 복귀할 때 재삽입 엔트리들의 일부 또는 전부를 삽입하는 경우에 기록하는 로그 레코드

- 로그 레코드의 포함 내용
  노드 식별자, 엔트리들

- 복귀/재수행 방법
  노드 지향 복귀 및 재수행



(그림 7) 재삽입 연산의 회복

(그림 7)에서 재삽입 로그 레코드에 대한 간단한 복귀의 예를 보여준다. 이 상황은 재삽입 엔트리들을 삭제하고 이로 인해 변경된 MBR을 상위 조상노드에 반

영한 상태에서 시스템 고장이 발생한 경우이다. 재시작 시에 이를 모두 재수행하고 다시 복귀할 때 재삽입 엔트리 삭제 레코드(LG_DELREGRP)를 만난 경우이다. 이 때에는 먼저 해당 노드의 헤더에서 현재 재삽입 된 엔트리들의 개수를 알 수 있고 이를 통해 노드에 다시 삽입할 재삽입 엔트리들을 결정한다. 이들을 노드에 재삽입 한 후 변경된 MBR을 경로스택을 이용해 조상 노드들에 반영한다. 이런 과정이 끝나면 DummyCLR을 기록하여 NTA의 끝을 알린다.

(그림 3)에서 재삽입이 수행되는 부분을 보면 삭제한 엔트리들이 모두 삽입 모듈을 통해 삽입되는 것을 볼 수 있다. 재삽입 엔트리 하나의 삽입이 재삽입을 수행하지 않는 삽입 연산과 같기 때문에 재삽입 연산이 수행되는 도중에 트랜잭션 고장이나 시스템 고장이 발생하면 각각의 삽입으로 인한 모든 변경이 복귀되어야 한다. 이것은 시스템 재시작의 속도를 느리게 하며 트랜잭션 철회시 다른 트랜잭션들에 영향을 미치게 되어 탐색연산이나 다른 삽입 연산들을 지연시킨다. 이 논문에서는 재삽입 연산으로 인한 변경을 모두 복귀하지 않고도 색인구조의 일관성을 유지할 수 있는 방법을 제시한다.

3.5.2 노드 분할 연산

노드 넘침을 처리하는 과정에서 노드를 분할하는 작업을 말한다. 이 연산은 넘침 처리 연산 NTA의 일부이다. 이 작업을 수행 할 때에는 다음과 같은 로그 레코드를 기록한다.

① 분할 노드 삭제 레코드(LG_DELGRP)

- 로그 레코드의 용도
  단말 또는 비단말 노드에서 원래 노드에서 분할된 노드로 이동할 엔트리들을 삭제할 때 기록하는 로그 레코드

- 로그 레코드의 포함 내용
  노드 식별자, 삭제한 엔트리들, 노드 내에서 삭제한 엔트리들의 식별자, 엔트리들의 개수

- 복귀/재수행 방법
  노드 지향 복귀 및 재수행, 재수행 시에는 해당 노드에서 엔트리들의 식별자를 이용해 엔트리들을 삭제한다. 복귀시에는 삭제한 엔트리들을 다시 노드에 삽입한다.

② 분할 노드 이동 레코드(LG_INSGRP_RO)

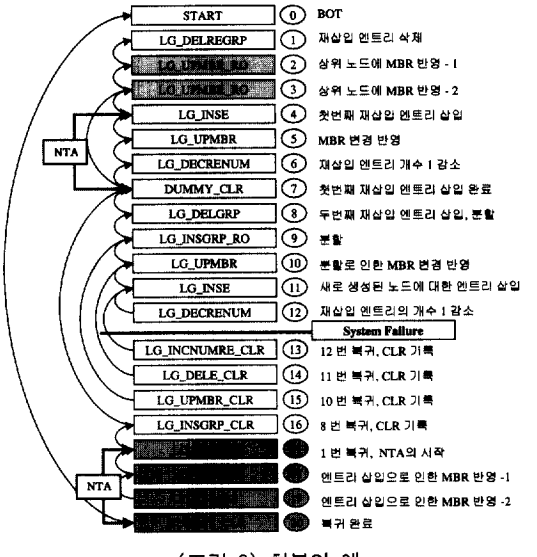
- 로그 레코드의 용도
  단말 또는 비단말 노드에서 분할을 수행하는 동안 새로 할당한 노드에 엔트리들을 이동할 때 기록하는 로그 레코드

- 로그 레코드의 포함 내용
  노드 식별자, 삽입한 엔트리들, 엔트리들의 개수

- 복귀/재수행 방법
  재수행만 수행, 이 로그 레코드는 새로 생성된 노드에 엔트리를 삽입하는 것이므로 굳이 복귀를 수행할 필요가 없다.

3.6 회복 과정의 예

앞에서 정의한 로그 레코드를 가지고 로그 레코드와 그에 대한 회복을 예를 들어서 설명한다. (그림 8)은 색인 구조에 엔트리를 삽입하는 동안 엔트리 삽입으로 인해 넘침이 발생하고 이 넘침을 재삽입으로 처리하면서 기록한 로그 레코드를 시스템 고장이 발생하여 재시작시 재수행 및 복귀를 하는 내용이다. (그림 8)에서 로그 레코드를 표시하는 기호 중에 LG_xxxx_CLR은 복귀를 할 때 기록하는 보상 로그레코드를 의미한다.



(그림 8) 회복의 예

1번 로그 레코드는 재삽입을 수행하기 위해 노드에서 재삽입 엔트리를 삭제하면서 기록한 로그 레코드이

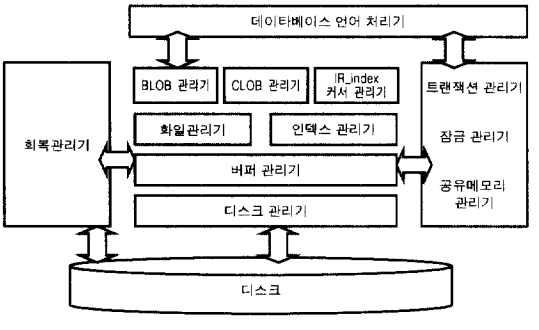
다. 2번은 노드에서 삭제한 재삽입 엔트리들의 개수를 첫 번째 레코드에 기록하면서 기록한 로그 레코드이다. 3, 4번은 재삽입 엔트리들을 삭제한 후에 이로 인해 변경된 MBR을 조상 노드들에 반영하면서 기록한 로그 레코드들이다. 5, 6, 7번은 첫 번째 재삽입 엔트리들을 트리에 삽입하고 이로 인한 MBR 변경을 상위 노드에 반영하고 재삽입이 진행되고 있는 노드의 첫 번째 레코드를 하나 감소시키는 과정에서 기록한 로그 레코드들이다. 8번은 하나의 NTA(한 개의 재삽입 엔트리 삽입)를 끝냈다는 의미에서 기록하는 Dummy CLR이다. 9, 10, 11, 12, 13번은 두 번째 재삽입 엔트리를 삽입하는 과정에서 해당 노드에서 넘침이 발생하여 분할을 수행하고 분할된 노드에 대한 엔트리를 상위에 삽입하고 재삽입 엔트리들의 개수를 하나 감소시키는 과정에서 기록한 로그레코드들이다. 이 이후에 시스템 고장이 발생하였고 그 이후의 로그 레코드들은 1~13번까지의 로그 레코드들을 복귀하는 과정에서 기록한 CLR들이다.

14번은 13번 로그 레코드를 복귀하고 그에 대한 CLR을 기록한 것이다. 15번은 12번 로그 레코드를 복귀하면서 기록한 CLR이다. 복귀 중에 8번 로그 레코드를 만나게 되면 그 안의 로그 레코드들은 복귀하지 않고 바로 4번 로그 레코드를 복귀한다. 5, 6, 7번은 하나의 NTA이기 때문이다. 3번 로그 레코드는 복귀하지 않는다. 이 로그 레코드는 재수행 Only 로그 레코드이며 이 로그 레코드를 복귀하게 되면 1번 로그 레코드를 제대로 복귀 할 수 없게 된다. 1번 로그 레코드에 대한 복귀 절차는 다음과 같다. 먼저 재삽입이 발생했던 노드의 첫 번째 레코드에 기록된 재삽입되지 않고 남아 있는 엔트리들의 개수를 읽어온다. 그리고 로그 레코드에 저장된 엔트리들 중 그 개수만큼의 엔트리들을 노드에 삽입하게 된다. 노드에 엔트리들을 삽입하면서 변경된 MBR을 조상노드들에 반영한다. 이 과정은 NTA(21, 22, 23, 24)로 처리하고 이 NTA의 Dummy CLR은 1번의 PrevLSN인 0번 로그 레코드를 가리키게 된다.

4. 설계한 회복 기법의 구현

이 논문에서는 설계한 회복기법을 CIR-Tree에 적용하여 BADA-Ⅲ의 하부 저장 시스템인 MiDAS-Ⅲ에서 구현하였다. 구현은 2개의 Ultra Sparc CPU, 128MBytes의 주기억장치의 하드웨어와 Solaris 2.5 운영체제를 탑재한 플랫폼에서 이루어졌다. 구현에 사용된 프로그램 언어는 GNU C 2.7.1이다.

MiDAS-Ⅲ는 데이터의 저장, 접근, 동시성 및 회복을 지원하는 저장 관리 시스템이다. (그림 9)에서 MiDAS-Ⅲ의 전체적인 구성을 보여주고 있다. 데이터를 처리하는 부분은 디스크에 대한 데이터의 입출력과 메모리 관리 그리고 인덱스 관리, 커서관리, IR-Index 관리 등이다. 트랜잭션을 관리하는 부분은 잠금 관리, 작업관리, 회복관리 부분이다. MiDAS-Ⅲ의 회복 관리는 ARIES [1]를 구현하여 수행하고 있다. CIR-Tree를 위한 회복 기법은 MiDAS-Ⅲ에서 사용하는 ARIES를 기반으로 구현되었다. 즉, 로그 레코드의 쓰기, 읽기와 같은 회복을 위한 기본적인 연산은 MiDAS-Ⅲ에서 제공하는 함수들을 사용하였다. 또한 MiDAS-Ⅲ에서 제공하는 로그 분석단계, 재수행 단계, 복귀 단계의 큰 틀 안에서 CIR-Tree를 위한 적절한 로그 레코드 형태와 각 로그 레코드를 위한 자료 구조를 정의하였다. 정의한 로그 레코드들이 재수행 단계와 복귀단계에서 어떤 동작을 수행해야 하는지를 정의하고 이를 재수행/복귀함수로 구현하였다.



(그림 9) MiDAS-Ⅲ 시스템 내부 구성도

<표 1>은 실제 구현을 하면서 정의한 로그 레코드 형태와 각 로그 레코드를 위한 자료 구조 그리고 각 로그 레코드들을 위한 복귀 및 재수행 동작을 정의한 Redo/Undo 함수들이다. 표의 "재수행·복귀 로그"는 복귀와 재수행이 가능한 로그 레코드들을 의미하고 "복귀 로그"란 복귀만 수행하는 로그레코드, "보상 로그"란 로그 레코드들을 복귀할 때 기록하는 보상 로그 레코드를 각각 의미한다. 여기에 제시되는 로그, 복귀 및 재수행 함수는 본 논문에서 구현한 함수들이다. 로그 함수들의 경우에는 CIR-Tree에 맞게 로그 레코드를 구성하는 부분은 본논문에서 구현한 것이지만 이를 실제로 로그

〈표 1〉 로그레코드의 종류와 재수행/복귀 함수

Log Record Type		Data Structure	Redo Function	Undo Function
재수행·복귀 로그	LG_INSENTRY	T_LG_ENTRY	cir_RedoInsEntry	cir_UndoInsEntry
	LG_DELGRP	T_LG_GRP	cir_RedoDelGrp	cir_UndoDelGrp
	LG_UPDATEMBR	T_LG_MBR	cir_RedoUpdateMBR	cir_UndoUpdateMBR
	LG_UPDATENSN	T_LG_NSN	cir_RedoUpdateNSN	cir_UndoUpdateNSN
	LG_UPDATELINK	T_LG_LINK	cir_RedoUpdateLink	cir_UndoUpdateLink
	LG_DELENTRY	T_LG_ENTRY	cir_RedoDelEntry	cir_UndoDelEntry
	LG_INSGRP	T_LG_GRP	cir_RedoInsGrp	cir_UndoInsGrp
	LG_DELREGRP	T_LG_REGRP	cir_RedoDelReGrp	cir_UndoDelReGrp
	LG_DECRENUM	T_LG_RENUM	cir_RedoDecReNum	cir_UndoDecReNum
복귀 로그	LG_UPDATENSN_RO	T_LG_NSN_RO	cir_RedoUpdateNSN_RO	-
	LG_UPDATEMBR_RO	T_LG_MBR_RO	cir_RedoUpdateMBR_RO	-
	LG_INITPAGE_RO	T_LG_INITPAGE_RO	cir_RedoInitPage	-
	LG_INSGRP_RO	T_LG_GRP	cir_RedoInsGrp	-
	LG_INSENTRY_RO	T_LG_ENTRY	cir_RedoInsEntry	-
보상 로그	LG_INSENTRY_CLR	T_LG_ENTRY	cir_RedoDelEntry	-
	LG_DELGRP_CLR	T_LG_GRP	cir_RedoInsGrp	-
	LG_UPDATEMBR_CLR	T_LG_MBR_RO	cir_RedoUpdateMBR_RO	-
	LG_UPDATENSN_CLR	T_LG_NSN_RO	cir_RedoUpdateNSN_RO	-
	LG_UPDATELINK_CLR	T_LG_LINK_RO	cir_RedoUpdateLink_RO	-
	LG_DELENTRY_CLR	T_LG_ENTRY	cir_RedoInsEntry	-
	LG_DECRENUM_CLR	T_LG_RENUM	cir_RedoIncReNum	-
	LG_INSGRP_CLR	T_LG_GRP	cir_RedoDelGrp	-

에 기록하는 함수는 MiDAS의 LOG_LogAppend() 함수를 이용한다. 복귀 및 재수행 함수들은 CIR-Tree의 로그 레코드를 적절히 복귀 및 재수행하는 형태로 작성되었으며 이들의 호출은 MiDAS의 로그 관리자에 의해 수행된다.

5. 성능 평가

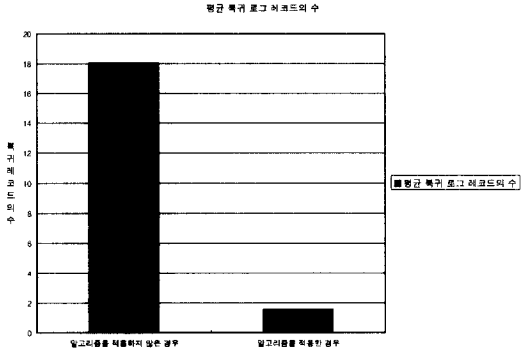
성능 평가의 수행은 4장에서 언급한 플랫폼에서 이루어 졌다. 사용된 데이터 집합은 10차원 100000 개의 균등 분포의 특성을 갖는다. CIR-Tree 인덱스의 페이지 크기는 16 Kbytes로 하였고 이때 단말 노드에 들어갈 수 있는 최대 엔트리의 개수는 300개였다. 단말 노드에 넘침이 발생해서 재삽입을 수행하는 경우에 선택되는 엔트리의 개수는 전체 개수의 30%로 하였다. 이 경우에는 90개가 재삽입 연산에 의해 노드에서 삭제되고 다시 삽입되게 된다.

비교 대상은 본 논문에서 제안한 알고리즘과 이를 적용하지 않고 재삽입 연산에 대한 회복을 처리하는 방법을 비교하였다. 즉, 알고리즘을 적용하지 않았다는 것은 ***[트랜잭션 시작 - 삽입 - 재삽입 발생 - 재삽입 엔트리 1, 2, 3, ... n 삽입 - 트랜잭션 종료]***와 같은 시나리오의 경우에 트랜잭션이 종료하기 전에 시스템 고장이 발생한 경우에 ***[재삽입 엔트리 n 복귀 - n-1 복귀 ... - ... - 1 복귀 - 삽입 복귀]***와 같이 회복을 수행하는 방법을 말한다.

실험에서는 엔트리 하나의 삽입을 하나의 트랜잭션으로 하였다. 따라서 100000개의 삽입은 100000개의 트랜잭션이 순서적으로 발생한다는 것을 의미한다. 100000개의 삽입 트랜잭션이 순서적으로 발생할 때 임의의 시점에서 시스템 고장 또는 트랜잭션 고장을 발생시키고 이를 회복할 때 복귀하는 로그 레코드의 개수를 측정하였다. 이때 총 고장 발생 회수는 500회였다. 각각의 고장 발생시 복귀한 로그 레코드의 개수를 측정하고 나중에 이들의 평균을 내었다. 평균 복귀해야할 로그 레코드가 많다는 것은 그만큼 회복에 걸리는 시간이 많다는 것을 의미한다. 이외에도 100000개의 삽입을 수행하면서 기록한 로그 레코드의 개수도 각각 측정하였다.

(그림 10)은 평균 복귀로그 레코드의 개수를 나타낸 그래프이다. 그림에서 보듯이 알고리즘을 적용한 경우에는 평균 2개의 로그 레코드를 복귀한 반면 그렇지 않은 경우에는 평균 18개의 로그 레코드를 복귀하고 있다. 약 9배의 로그 레코드를 더 복귀해야 한다. 10차원 100000개의 데이터를 페이지 크기 16K에서 구축하

였을 때 트리의 높이는 2였다.



(그림 10) 평균 복귀 레코드의 수

<표 2>는 10000개를 삽입하면서 기록한 총 로그 레코드의 개수를 각각 표시한 것이다. 알고리즘을 적용했을 때 약 2000개 정도(1.5%)의 로그 레코드를 더 기록했는데 이것은 전체 로그레코드의 개수에 비교했을 때 거의 무시 할 수 있는 양이다. 실험 결과 제안하는 알고리즘을 이용하면 그렇지 않은 경우에 비해 매우 효율적으로 재삽입 연산에 대한 회복을 수행한다는 것을 알 수 있었다.

〈표 2〉 총 기록한 로그레코드의 개수

구 분	알고리즘을 적용한 경우	알고리즘을 적용하지 않은경우
총 기록한 로그 레코드의 개수	144350	141915

6. 결 론

이 논문은 고차원 색인 구조에 대한 효과적인 회복 기법에 대한 것으로, 재삽입 연산을 포함하는 트랜잭션이 시스템 고장이나 트랜잭션 고장으로 인해 복귀 또는 재수행 될 때 보다 효율적으로 수행되도록 하는 회복기법을 제안하였다. 또한 기존에 제시된바 없는 재삽입 연산에 적절한 회복기법을 최초로 제안하였다. 이 논문에서 제안한 회복기법은 재삽입 연산을 수행할 때 발생하는 변경을 시스템 고장 또는 트랜잭션 고장 뒤에 효율적으로 복귀할 할 수 있는 방법을 제공하여 재시작의 속도를 빠르게 하였고, 재삽입 연산을 포함하는 트랜잭션의 철회가 다른 트랜잭션에 영향을 주는 것을 최소화하였다. 또한 제안된 회복기법을 멀티미디어 DBMS인 BADA의 하부저장구조 MiDAS에 CIR-Tree를 바탕으로 구현하였다. 또한, 본 논문에서 구현한 회복 기법과 제안한 알고리즘을 적용하지 않은 방법과의 비료를 통해서 제안하는 회복 기법이 재삽입 연산에 대해서 보다 효율적인 회복을 수행한다는 것을 보였다.

참 고 문 헌

[1] 이석희, 유재수, 조기형, 허대영, "CIR-Tree : 효율적인 고차원 색인기법", 한국정보과학회 논문지, Vol.26, No.6, pp.724-734, June 1999.

[2] C. Mohan, D. Harderle, B. Lindsay, H. Pirahesh and P. Schwarz, "ARIES : A Transaction Recovery Method Supporting Fine-Granularity Locking and Partial Rollbacks Using Write Ahead Logging," ACM TODS, Vol.7, No.1, pp.94-162, 1992.

[3] C. Mohan and F. Levine, "ARIES/IM : An Efficient and High Concurrency Index Management Method Using Write-Ahead Logging," ACM SIGMOD Proc., pp.371-380, 1992.

[4] D. A. White and R. Jain, "Similarity Indexing with the SS-tree," ICDE96 Proc., pp.515-523, 1996.

[5] K.-I. Lin, H. Jagadish and C.Faloutsos, "The TV-tree : An Index Structure for High-Dimensional Data," VLDB Journal, Vol.3, No.4, pp.517-549, 1994.

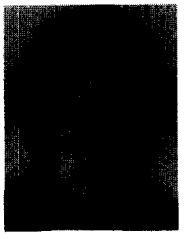
[6] M. Kornacker, C. Mohan and J. Hellerstein, "Concurrency and Recovery in Generalized Search Trees," ACM SIGMOD, pp.62-72, 1997.

[7] M. Kornacker and D. Banks, "High-Concurrency Locking in R-trees," 21'nd VLDB Proc., pp.134-145, 1995

[8] N. Beckmann, H. P. Kriegel, R. Schneider and B. Seeger, "The R*-tree : An Efficient and Robust Access Method for Points and Rectangles," ACM SIGMOD Proc., pp.322-331, 1990.

[9] N. Katayama and S. Satoh, "The SR-tree : An Index Structure for High-Dimensional Nearest Neighbor Queries," ACM SIGMOD Proc., pp.369-380, 1997.

[10] S. Wang, J. M. Hellerstein and I. Lipkind. "Near-Neighbor Query Performance in Search Trees," UC Berkeley Technical Report CSD-98-1012, 1998.



송 석 일

e-mail : prince@pretty.chungbuk.ac.kr

1998년 충북대학교 정보통신공학과 (공학사)

2000년 충북대학교 정보통신공학과 (공학석사)

2000년~현재 충북대학교 정보통신공학과 박사 과정

관심분야 : 데이터베이스 시스템, 트랜잭션, 저장 시스템, 멀티미디어 정보검색, XML, 정보검색 프로토콜 등

이 석 희

e-mail : seoklee@dab-c.ac.kr

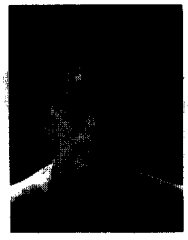
1994년 충북대학교 정보통신공학과 (공학사)

1998년 충북대학교 정보통신공학과 (공학석사)

2000년 충북대학교 정보통신공학과 박사과정 수료

2000년~현재 동아방송대학 인터넷방송과 전임강사

관심분야 : 데이터베이스 시스템, 정보검색, 인터넷 방송, 분산 객체 컴퓨팅 등



유 재 수

e-mail : yjs@cbucc.chungbuk.ac.kr

1989년 전북대학교 컴퓨터공학과 (학사)

1991년 한국과학기술원 전산학과 (공학석사)

1995년 한국과학기술원 전산학과 (공학박사)

1995년~1996년 목포대학교 전산통계학과 전임강사

1996년~현재 충북대학교 공과대학 전기전자공학부 조교수

관심분야 : 데이터베이스 시스템, 정보검색, 멀티미디어 데이터베이스, 분산 객체 컴퓨팅 등