다차원 색인구조를 위한 효율적인 동시성 제어기법

(An Efficient Concurrency Control Algorithm for Multi-dimensional Index Structures)

김 영 호 † 송 석 일 †† 유 재 수 †††

(Young Ho Kim) (Seok Il Song) (Jae Soo Yoo)

요 약 이 논문에서는 질의의 지연을 최소화하는 효율적인 동시성제어 알고리즘을 제안한다. 다차원 색인구조에서 탐색연산을 지연시키고 전체적인 동시성을 떨어뜨리는 주 요인은 노드 분할과 MBR 변경연산이다. 제안하는 알고리즘에서는 분할 연산에 의한 질의의 지연을 최소화하기 위해 분할 노드에서의 배타 잠금 시간을 최소화한다. 분할 전체 기간동안 노드에 배타 래치를 획득하는 것이 아니고 분할 과정중 노드의 물리적인 분할 단계에서만 배타 래치를 획득한다. 또한, MBR 변경 시 발생하는 질의의 지연을 줄이기 위해 부분 잠금 결합(PLC:Partial Lock-Coupling)을 사용한다. PLC 기법은 MBR 증가 연산에 비해 상대적으로 발생 빈도가 적은 MBR 감소 연산에서만 잠금 결합을 수행하여 동시성을 향상시킨다. 성능평가를 위해 제안하는 알고리즘과 링크 기법을 기반으로하는 기존의 동시성 제어 기법을 바다-Ⅲ DBMS의 자료저장 시스템인 MIDAS-Ⅲ 상에서 구현한다. 다양한 환경에서의 성능평가를 통해 제안하는 알고리즘이 기존의 동시성 제어기법보다 처리율 및 응답시간에서 뛰어난 성능을 나타냄을 보인다.

키워드 : 동시성제어, 회복, 다차원 색인구조

Abstract In this paper, we propose an enhanced concurrency control algorithm that minimizes the query delay efficiently. The factors that delay search operations and deteriorate the concurrency of index structures are node splits and MBR updates in multi dimensional index structures. In our algorithm, to reduce the query delay by split operations, we optimize exclusive latching time on a split node. It holds exclusive latches not during whole split time but only during physical node split time that occupies small part of whole split time. Also to avoid the query delay by MBR updates we introduce partial lock coupling(PLC) technique. The PLC technique increases concurrency by using lock coupling only in case of MBR shrinking operations that are less frequent than MBR expansion operations. For performance evaluation, we implement the proposed algorithm and one of the existing link technique-based algorithms on MIDAS-Ⅲ that is a storage system of a BADA-Ⅲ DBMS. We show through various experiments that our proposed algorithm outperforms the existing algorithm in terms of throughput and response time.

Key words : concurrency control, recovery, multi-dimensional index structure

1. 서 론

지리 정보 시스템, 멀티미디어 데이타 베이스, 내용기반 이미지 데이타 베이스와 같은 응용들에 대한 요구가 높아지면서 그 핵심 기술 중 하나인 다차원 및 고차원 색인구조에 대한 연구가 매우 활발히 진행되어 왔다. 현재 그림, 사진, 지도 데이타, 캐드 데이타와 같은 이미지 정보는 폭발적으로 증가하고 있는 추세이며, 이에 따라 이들을 좀더 효율적으로 색인 할 수 있는 다차원 및 고차원 색인구조의 중요성은 더욱 가중될 것이다. 여러 다차원 색인 구조들(R-트리[1], R^*-트리[2], R -트리[3],

· 이 연구는 2001년도 학술진흥재단(KRF-2001-041-E00233)의 지원으로 수행되었음.

† 비 회 원 : 한국전자통신연구원
kyh05@etri.re.kr
†† 비 회 원 : 충북대학교 정보통신공학과
prince@netdb.chungbuk.ac.kr
††† 종신회원 : 충북대학교 전기전자 및 컴퓨터 공학부 교수
yjs@cbucc.chungbuk.ac.kr

논문접수 : 2001년 1월 15일
심사완료 : 2002년 10월 14일

TV-트리[4], X-트리[5], CIR-트리[6, 7] 등)이 제안되었고, 이들 중 몇몇은 실제 응용에서 사용되고 있다. 하지만 대부분의 응용에서 삽입할 때는 탐색이 발생하지 않도록 하는 동시성 제어 정책을 사용하고 있다. 응용들의 특성상 한번 색인이 구축되면 추가적인 삽입은 그렇게 빈번하게 발생하지 않기 때문이다. 하지만 온라인 서비스를 해주어야 하는 상황에서 이것은 적절하지 않으며 다차원 색인 구조의 응용분야를 축소시키게 되므로 반드시 적절한 동시성 제어 기법이 있어야 한다.

다차원 색인구조에서 가장 많이 발생하는 탐색 연산이 지연되는 경우는 노드의 넘침으로 인한 분할과 MBR의 축소 또는 확장으로 인한 MBR 변경을 상위 노드들에 반영할 때이다. 다차원 색인구조의 데이타에 대한 전체적인 순서화의 부족으로 노드 분할의 비용이 단일차원 색인구조 보다 매우 높다. 또한 MBR 변경 역시 매번 삽입이나 삭제 시에 발생가능하며, 변경이 발생하면 이를 상위 노드들에 전파해야 한다는 부담을 가지고 있다. R^{Link}-트리[8], GiST를 위한 동시성 제어 기법(CGiST)[9] 같은 기존의 동시성 제어 기법에서는 분할 및 MBR 변경 수행 시에 노드에 대해 탐색에 대한 배타 잠금을 획득하고 수행하는 기법을 사용하고 있다. 따라서 분할 및 MBR 변경 수행이 끝날 때까지 해당 노드에서 탐색 연산을 수행할 수가 없었다. 뿐만 아니라 분할과 MBR 변경을 수행할 때 잠금 결합을 사용하는데, 이는 삽입과 탐색의 동시성을 저하시킨다. TDIM(Top-Down Index region modifications)과 이를 확장한 CCU(Copy based Concurrent Update) 및 CCU_NQ(Copy based Concurrent Update with Non-blocking Queries) [10]에서는 비록 잠금 결합을 사용하지는 않지만 MBR 변경 연산 중 삭제에 대한 고려가 없다. 이는 응용 분야의 폭을 좁게 만들뿐만 아니라, 삭제를 고려했을 때 알고리즘이 복잡해지고 성능이 저하될 수 있다. 이 논문에서는 분할 및 MBR 변경 시에 발생하는 탐색 연산의 지연을 최소화하는 동시성 제어 알고리즘을 제안한다. 또한 잠금 결합의 사용으로 인해 발생하는 동시성 저하 문제를 해결하는 방법을 제안하고, 삽입과 삭제를 모두 고려한 동시성 제어 기법을 제안한다.

이 논문의 구성은 다음과 같다. 2장에서는 관련 연구로서 기존의 다차원 색인구조에서의 동시성 제어 기법에 대해 살펴본다. 3장에서는 제안하는 동시성 제어 기법의 특징 및 삽입, 삭제 그리고 탐색 연산에 대해서 설명한다. 4장에서는 제안된 알고리즘을 구현하고 GiST의 동시성 제어 기법과 성능을 비교 평가한다. 마지막으로 5장에서 결론을 맺는다.

2. 다차원 색인 구조의 동시성 제어 기법

기존의 다차원 색인 구조의 동시성 제어 기법들은 크게 잠금 결합 기법(lock-coupling technique)과 링크 기법(link-technique)으로 분류된다. 잠금 결합을 수행하는 동시성 제어 기법으로는 R^{Couple}-트리[11]가 대표적이다. 이 기법에서는 노드를 순회할 때 먼저 방문할 노드의 잠금을 획득한 후 현재 노드의 잠금을 해제하는 잠금 결합 기법을 사용한다. 잠금을 사용함으로써 트리의 일관성 유지와 정확한 탐색을 보장하고 있다. 분할을 수행하거나 MBR 변경을 상위 노드에 반영할 때는 작업이 끝날 때까지 참여하는 모든 노드에 잠금을 유지한다. 이 기법은 탐색이나 삽입 연산을 수행할 때 하나의 트랜잭션이 여러 노드에 잠금을 유지해야하고, 디스크의 I/O 기간 동안 잠금을 유지해야 하는 단점이 있다.

잠금 결합에 의해 발생하는 문제를 해결하기 위해 링크 기법을 이용하는 동시성 제어 기법이 제안되었다. 링크 기법 기반의 동시성 제어 알고리즘은 잠금 결합을 수행할 필요가 없다. 즉, 트리를 순회하는 동안에 최대 하나의 노드에만 잠금을 획득하면 된다. 하지만 여전히 MBR 변경이나 분할 연산을 수행할 때에는 동시에 여러 노드에 잠금을 획득해야 한다. 링크 기법을 기반으로 하는 다차원 색인 구조에 대한 동시성 제어 알고리즘의 대표적인 것으로 R^{Link}-트리[8]와 GiST를 위한 동시성 제어 기법(CGiST)[9]을 들 수 있다. R^{Link}-트리와 CGiST에서는 링크를 통해 상위 노드에 반영되지 않은 하위 노드의 분할을 감지하여 이를 보상할 수 있도록 하고 있다. 링크 기법의 기본 개념은 각 레벨의 모든 노드들을 오른쪽 링크를 통해 연결하여 탐색 연산 시에 잠금 결합을 수행하지 않아도 상위 노드에 반영되지 않은 하위 노드의 분할을 감지하여 이를 보상할 수 있도록 하는 것이다. R^{Link}-트리의 동시성 제어 알고리즘은 B^{Link}-트리[12]에서 제안된 링크 기법을 R-트리에 맞게 수정한 방법으로, 트리를 하향 순회할 때는 잠금 결합을 수행하지 않고도 동시성 제어가 가능하다.

그러나 R-트리의 특성상 트리를 거슬러 올라가는 연산에서는 잠금 결합을 수행해야한다. 그것은 동시에 삽입 연산을 수행하는 프로세스들이 삽입 연산 결과를 상위노드에 반영할 때 순서가 지켜지지 못함으로써 트리의 일관성을 유지하지 못하는 경우가 발생할 수 있기 때문이다. R^{Link}-트리에서는 R-트리와 B-트리의 구조적인 차이로 인해 B^{Link}-트리의 링크 기법을 수정 및 보완하였다. 이 과정에서 NSN(node sequence number)을

도입하여 이를 각 노드에 할당하게 되었고 엔트리의 구조도 <MBR, child_node>에서 <MBR, child_node, NSN>로 수정되었다. NSN 이라는 것은 노드가 새로 생성될 때 증가하는 값으로 각 노드는 유일한 NSN을 부여받는다. 이 방법에서는 어떤 트랜잭션이 노드 분할과 MBR 변경 등과 같은 트리 구조를 변경하는 연산을 수행할 때는 동시에 둘 이상 레벨의 노드들에 잠금을 유지해야 하며 이로 인해 탐색 연산이 다음 노드로 탐색을 진행할 때 I/O 시간동안 지연 될 수 있다. 또한, 엔트리를 표현하는 자료 구조가 <MBR, child_node, NSN>로 바뀌게 되면서 부가적인 정보 NSN으로 인해 저장공간 이용률이 저하되어 비 단말 노드의 팬 아웃(fan out)이 작아지게 된다는 단점을 갖는다.

CGiST에서 제안하고 있는 동시성 제어 알고리즘에서는 R^{Link}-트리의 문제점으로 지적된 엔트리의 자료구조에 NSN이 추가되어 저장공간을 낭비하는 문제를 해결하고 있다. CGIST에서도 역시 R^{Link}-트리에서와 같이 각각의 노드에 NSN을 할당하고 오른쪽 링크를 두어 분할을 검사하고 어디까지 오른쪽 링크를 따라 순회 할 것인지를 판별한다. 하지만 R^{Link}-트리와는 다르게 비 단말 노드의 엔트리에 하위 노드의 NSN을 기록해두지 않고도 하위 노드의 분할을 판별할 수 있는 방법을 제안하고있다. 이것은 전역계수기를 이용해 NSN을 할당함으로써 가능해진다. 이 기법 역시 문제점을 가지고 있다. 전역계수기를 사용하면 노드 분할을 수행할 때, 먼저 상위 노드에 잠금을 획득하고 분할을 진행해야 한다. 이외에도 회복 목적을 위해 분할이 끝날 때까지 분할에 참여하는 모든 노드들에 잠금을 유지한다. 이로 인해 [3]에서 보다 더 오랜 시간 동안 탐색연산이 지연될 수 있다.

인덱스 변경 연산과 노드 분할 연산에서의 잠금 결합으로 인한 동시성 저하를 줄이고, 노드 분할 시 탐색 연산의 지연을 줄이기 위해 TDIM을 기반으로 하는 CCU와 CCU_NQ가 제안되었다[10]. TDIM은 역시 링크 기법을 기반으로 하고 있는데, 삽입 연산 수행 시 엔트리 삽입 후 MBR 변경을 반영하기 위해 다시 상향 순회해야하는 부담을 줄인다. 즉, 하향 순회하면서 엔트리를 삽입할 노드를 찾고 동시에 MBR 변경을 수행한다. 삽입 수행을 위해 노드를 순회할 때는 탐색 및 MBR 변경 연산과 호환되는 잠금을 획득하고 트리를 순회하다가 MBR 변경이 발생하면 탐색과는 호환되고 다른 변경 연산과는 호환되지 않는 잠금으로 변경한다. 만약, 현재 방문하는 노드에서 미리 감지하지 못한 분할이 발생했다면 상위 노드로 가서 다시 이와 같은 작업을 반복해서 수행한다.

TDIM에서 분할을 수행할 때는 분할이 수행되는 노드에서 탐색 연산의 지연이 발생한다. CCU는 TDIM을 확장해서 TDIM에서 발생하는 탐색 연산의 지연을 줄인 방법이다. 노드 분할을 공유 영역의 노드에서 직접 수행하지 않고 로컬 영역으로 노드의 사본을 복사한 후, 로컬 영역의 복사된 노드에서 분할 작업을 수행한다. 로컬 영역에서 실제적인 분할이 수행되기 때문에, 이 기간동안 넘침이 발생한 공유 영역의 노드에서는 탐색 연산의 수행이 가능하다. 분할의 수행이 완료되면, 넘침이 발생한 공유 영역의 노드에 모든 다른 연산과 호환되지 않는 배타 잠금을 획득한 후, 로컬 영역에 존재하는 분할된 노드의 정보를 반영한다. CCU_NQ는 CCU에서 공유 영역에 다시 분할을 반영할 때 탐색이 지연되는 것을 없애기 위한 기법이다. LA(Logical Address)를 도입해서 이 문제를 해결한다. LA는 노드가 처음 할당될 때 그 노드의 물리적 주소를 갖고, 분할이나 MBR 변경으로 노드가 변경되면 변경된 노드의 주소로 바뀌게 된다.

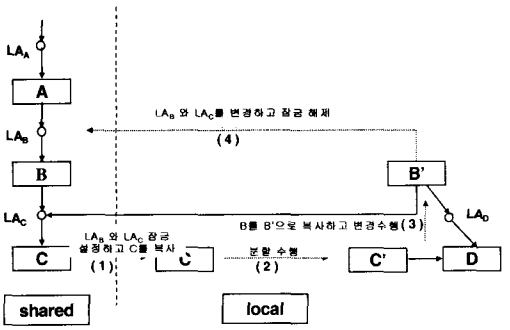

그림 1 CCU_NQ에서의 노드 분할

그림 1은 CCU_NQ에서 노드 분할을 수행하는 과정을 나타내는 그림이다. 노드 C에 새로운 엔트리의 삽입으로 넘침이 발생했을 때 분할을 수행하는 과정이다. 각 노드 A, B, C에 대한 LA인 LA_A, LAB_B, LA_C는 노드 생성 시에 각 노드의 물리적 주소를 할당받는다. 먼저 공유 영역의 노드 C를 로컬 영역으로 복사를 한다. 로컬 영역에서 분할을 수행하면 노드 C는 C′이 되고 새로운 노드 D가 생성된다. 노드 D가 생성될 때 D의 물리적 주소가 LA_D가 된다. 노드의 분할을 상위 노드 B에 반영하기 위해 공유 영역의 노드 B를 로컬 영역으로 복사한다. 노드 C′의 변경을 반영하고 새로운 노드

D에 대한 엔트리를 노드 B에 반영하면 노드 B가 B′이 된다. 이 때 노드 B′에서 D에 대한 엔트리의 자식 노드에 대한 포인터가 LA_D를 가리킨다. 더 이상 변경이 발생하지 않으면, 로컬 영역에 있는 분할된 노드 C′과 상위 노드 B′에 대한 정보를 반영하기 위해, LA_B는 노드 B′을 가리키도록 LA_C는 노드 C′을 가리키도록 포인터 정보를 변경하면 분할 수행이 완료된다.

TDIM은 [8, 9]의 삽입 연산에서 필요했던 잠금 결합의 필요성을 제거하였다. 그러나, 삭제 연산에 대해서는 전혀 고려하지 않고 있다. 삭제 연산에서는 삭제될 엔트리를 찾기 위해서는 완전 일치 질의처럼 트리 순회를 수행해야한다. 다차원 색인구조에서는 루트 노드부터 대상 엔트리 노드까지 여러 개의 경로를 가지므로, 현재 방문하는 노드가 삭제될 엔트리를 포함하는 노드의 조상 노드라는 것을 보장할 수 없다. 결과적으로, MBR 변경을 하향순회 형태로 처리하기 위해서는 대상 엔트리의 위치를 찾은 후에 MBR 변경을 수행해야한다. 그러나, 문제점이 이러한 작업들로 완전히 해결될 수 없다.

삭제 연산과 삽입 연산이 TDIM의 형태로 동시에 MBR 변경을 수행하게 되면 TDIM이 잠금 결합을 수행하지 않기 때문에 일관성이 깨지는 경우가 발생할 수 있다. 다음과 같은 상황을 생각할 수 있다. 삽입 연산이 새로운 엔트리(NE)를 삽입하기 위해 노드(N)를 방문하고, 엔트리들 중 자식 노드 포인터(CN)와 CN에 대한 MBR을 갖는 엔트리(E)를 선택한다. 삽입 연산은 MBR 변경을 수행하지 않고 트리의 순회를 계속한다. 그 다음에, 삭제 연산이 노드 N을 방문하고 CN에 대한 엔트리 E의 MBR을 변경한다. MBR 감소는 CN의 MBR이 삽입된 엔트리 NE를 포함하지 않도록 변경시켜, 트리의 일관성이 깨지는 경우가 발생할 수 있다. 그러므로, TDIM은 실제 응용에서 필요한 삭제 연산에 대한 처리가 없기 때문에 MBR 변경 연산에 대한 수정 없이 실제 응용에 사용되기 어렵다.

또한 CCU와 CCU_NQ는 질의의 지연을 최대한 줄였지만 효율적이지는 않다. 이 기법들은 분할 연산 수행시 로컬 영역의 복사된 노드에서 분할을 수행하므로 추가적인 공간이 필요하고 CCU_NQ에서는 쓰레기 노드를 주기적으로 처리해야한다. 이러한 특징들은 알고리즘의 구현을 어렵게한다. 이 논문에서는 대부분의 경우에 잠금 결합을 수행하지 않는 부분 잠금결합 (PLC:Partial Lock-Coupling)이라 불리는 MBR 변경 알고리즘을 제안하고 삭제 연산 역시 고려되어 있다. 또한, 추가적인 공간과 작업이 필요하지 않고 질의의 지연을 최소화하는 분할 알고리즘을 수행하는 동시성 제어 알고리즘을 제안한다.

3. 제안하는 동시성 제어 알고리즘

먼저 제안하고자 하는 동시성 제어 알고리즘의 전체적인 특징에 대해서 설명하고 제안하는 알고리즘의 핵심이 되는 분할 알고리즘과 MBR 변경 알고리즘에 대해서 자세히 설명한다. 그리고, 제안하는 알고리즘의 실제 수행 과정을 삽입, 삭제, 탐색 연산으로 나누어 이들을 의사코드로 정리하여 설명한다. 마지막으로 제안하는 알고리즘의 잠금-래치간의 교착상태에 빠지지 않음을 보인다.

3.1 제안하는 알고리즘의 특징

제안하는 알고리즘의 특징을 요약하면 크게 다음과 같은 세 가지로 압축할 수 있다. 첫 번째, 색인구조의 일관성을 유지하기 위해서 잠금과 래치의 사용목적을 분리한다. 래치는 변경을 수행하는 연산과 탐색 및 트리 순회를 수행하는 연산의 동기화를 위해 사용한다. 잠금은 변경연산들간의 동기화를 위해 이용한다. 이 특징은 다음에 설명할 제안하는 MBR 변경기법과 노드분할 기법과 합쳐져서 탐색연산의 지연을 줄일 뿐 아니라 전체적인 동시성 성능을 향상시킨다. 두 번째, 탐색지연을 최소화하는 분할 기법을 제안한다. 다차원 색인구조에서 노드분할의 특징을 실험을 통해 분석하고 분할 중 노드의 내용이 물리적으로 변경되기 전까지 탐색연산이 분할중인 노드를 접근할 수 있도록 한다. 실험결과 분할 연산중 물리적인 변경시간 보다는 분할을 위한 계산시간이 대부분을 차지한다는 것을 알 수 있었으며 이 방법은 분할로 인한 탐색연산의 지연을 매우 효과적으로 줄인다.

마지막으로, MBR 변경연산을 위해 수행하는 잠금/래치 결합을 반으로 줄이는 PLC 기법을 제안한다. 다차원 색인구조에서 MBR 변경연산이 일관되게 수행되지 못할때는 탐색오류를 유발하게 된다. 다차원 색인구조에서는 일관된 MBR 변경연산을 위해서 잠금/래치 결합을 이용해서 트리를 거슬러 올라가면서 MBR을 변경하는 순서를 유지한다. 제안하는 PLC는 모든 MBR 변경연산이 잠금결합을 필요로 하지 않는 다는 것을 보이고 MBR이 축소되는 일부 변경연산에 대해서만 잠금결합을 사용하는 방법을 제안한다. 다음부터 이상의 특징에 대해서 기존방법과의 비교를 통해 또는 실험을 통한 분석을 통해 자세하게 설명한다.

3.1.1 링크 기법을 기반으로 한다.

링크 기법은 각 레벨의 모든 노드들을 오른쪽 링크를 통해 연결하여 탐색 연산시에 잠금 결합을 수행하지 않

아도 상위 노드에 반영되지 않은 하위 노드의 분할을 감지하여 이를 보상할 수 있도록 하는 기법이다. 제안하는 알고리즘에서는 [9]에서 제시하고 있는 링크 기법을 사용한다. 즉, 전역 계수기를 사용하여 NSN을 할당하는 방법을 취해서 비단말 노드의 엔트리에 NSN이 포함되어 저장공간의 효율을 떨어뜨리는 문제를 방지한다. 이 때 전역계수기는 LSN(log sequence number)을 이용한다. LSN역시 NSN처럼 단조증가하며 절대 감소하지 않는 특성을 갖는다.

3.1.2 래치와 잠금을 혼용한다.

래치는 한 노드에 여러 트랜잭션들이 접근할 때 이들간의 동기화를 이뤄주며, 노드의 물리적 일관성을 보장하기 위한 것이다. 분할 연산을 직렬로 수행시키는 방법으로 루트 노드에 잠금을 획득하는 트리 잠금을 사용한다. 즉 분할 연산은 트리 전체에서 동시에 한 번만 수행된다. 그러나 다른 탐색이나 삽입, 변경을 수행하는 트랜잭션은 동시에 수행이 가능하다. 단말 노드에 대한 잠금은 회복과 관련되어서 분할이나 MBR 변경이 완료될 때까지 유지한다. 여기서는 회복에 대한 언급은 하지 않는다.

탐색 연산은 잠금은 획득하지 않고 공유 래치 만을 획득한다. 다음 노드로 순회를 할 때는 현재 노드의 래치를 해제하고 다음 노드의 공유 래치를 획득한다. 즉, 탐색을 수행하는 트랜잭션은 항상 방문하는 노드에만 공유 래치를 유지한다. 탐색 연산은 잠금을 획득하지 않으므로 배타 잠금과 충돌이 발생하지 않는다. 그러므로 MBR 변경이나 분할이 수행되는 트랜잭션과도 동시에 수행될 수 있게되어 탐색 연산의 동시성을 최대한 보장할 수 있다

3.1.3 질의의 지연을 최소로 하는 분할을 수행한다.

다차원 색인구조는 노드내의 엔트리가 순서화 되어있지 않기 때문에, 노드 분할 수행 시 연산의 수행 비용이 매우 크고 탐색이나 삽입, 삭제 연산에 비해 수행 시간이 매우 길다. 대부분의 다차원 색인구조의 동시성 제어 알고리즘에서는 분할이 수행되는 노드에 질의의 수행을 막는 배타 잠금이나 배타 래치를 획득한 상태에서 분할을 수행한다. 또한 분할을 수행한 후 상위 노드에 반영하는 과정에서 넘침이 발생하면, 상위 노드에서도 분할을 수행해야 한다. 이러한 과정은 루트 노드의 분할까지 연속해서 발생할 수 있고 분할 수행 시간은 더욱 길어지게 된다. 따라서, 기존의 동시성 제어기법에서는 분할이 수행되는 모든 과정동안 질의가 지연되고 동시성의 저하가 발생한다.

분할 과정을 살펴보면 분할 차원 선택, 분할 위치 선택, 새로운 노드 할당, 선택된 차원과 위치에 의한 엔트리들의 나눔, 버퍼에 엔트리들을 복사, 그리고 실제 물리적 분할을 수행하는 과정으로 진행된다. 제안하는 알고리즘에서는 분할 수행 과정을 두 단계로 구분하였다. 첫 번째 단계는 분할위치 계산 단계이다. 이 단계는 주로 계산하는 과정인데 엔트리의 순서화의 부족으로 수행시간이 상대적으로 길고 분할 전체 과정의 대부분을 차지한다. 또한, 이 단계는 실제적인 분할 수행이 아니기 때문에 다른 연산의 동시 수행이 가능하다. 두 번째 단계는, 첫 번째 단계에서 구해진 분할 차원과 분할 위치를 이용해서 넘침이 발생한 노드의 엔트리를 두 개의 노드에 물리적으로 나누는 과정이다. 이 단계는 엔트리를 복사하고 링크를 연결하는 과정으로 DBMS의 공유 버퍼 상에서 발생하며 그 시간이 매우 짧다. 그러나, 다른 연산의 동시 수행은 가능하지 않다. 그림 3.1은 차원이 변화될 때 분할의 첫 번째 단계와 두 번째 단계의 수행 시간을 실험을 통해 비교한 그림이다. 그림을 보면 첫 번째 단계가 전체 분할 수행 시간의 약 90% 이상을 차지하는 것을 볼 수 있다.

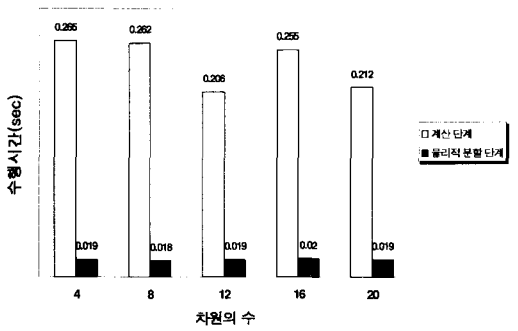

그림 2 분할 연산에서 각 단계의 수행시간 비교

제안하는 알고리즘에서는 노드 분할 수행 시 모든 분할 과정에 배타 래치를 획득하지 않고, 물리적인 분할을 수행하는 두 번째 단계에서만 배타 래치를 획득한다. 그림 3.1에서 볼 수 있듯이 전체 분할 과정의 90% 이상을 차지하는 첫 번째 단계는 현재 변경이 발생한 노드에 변경을 수행하지 않는 계산을 수행하는 과정으로 탐색 연산이 동시에 수행되어도 전혀 문제가 되지 않는다. 새로운 노드를 할당하고 초기화하는 과정이 있지만 아직 조상 노드에 기록되지 않고 링크의 연결도 되어있지 않기 때문에 탐색 연산에는 영향을 주지 않는다. 이러한 이유로 제안하는 분할 알고리즘에서는 첫 번째 단

게에서 탐색 연산의 동시 수행이 가능하도록 배타 래치가 아닌 공유 래치를 획득하고 수행한다. 탐색 연산은 잠금을 획득하지 않기 때문에 분할 연산이 획득한 배타 잠금에는 영향을 받지 않는다. 즉, 분할 과정의 대부분을 차지하는 첫 번째 단계에서 탐색 연산의 동시 수행이 가능하므로 분할 수행 중 탐색 연산의 지연이 최소화된다.

그림 3에서는 제안하는 분할 알고리즘의 잠금과 래치의 획득 및 해제과정을 보여주고 있다. 삽입연산은 먼저 삽입할 단말노드(B)를 선택하고 배타래치와 배타잠금을 획득한다. 단말노드에서 넘침이 발생하면 새로운 노드를 할당(B′)하고 B′에 배타잠금과 배타래치를 획득한다. 그리고 분할의 첫 번째 단계를 수행한다. 즉, 분할 차원과 분할 위치를 계산하고 B′으로 옮길 엔트리를 B′으로 복사한다. 이때까지 탐색연산들은 B의 모든 엔트리들을 볼수 있다. 다음에는 B의 공유래치를 해제하고 다시 배타래치를 획득한다. 배타래치를 성공적으로 획득하면 B의 엔트리들을 재구성한다. 재구성이 끝나면 B와 B′의 배타 래치를 해제하고 분할내용을 상위노드(A)에 반영하기 위해서 A에 배타잠금과 배타래치를 순서적으로 획득한다. 단말 노드의 배타잠금은 회복과 관련하여 분할이 완료될 때 까지 해제하지 않는다. B′에 대한 엔트리를 A에 삽입하고 A에 대한 배타래치를 해제하고 다시 공유래치를 획득한다. 마지막으로 B에 대한 MBR을 A에서 변경하고 분할을 마친다. 분할을 마치면서 단말 노드의 배타잠금을 해제한다.

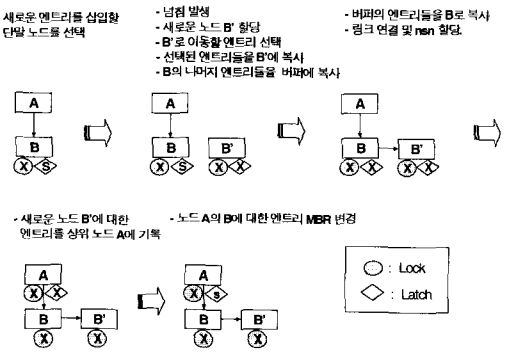

그림 3 분할 수행 과정에서 래치 및 잠금의 사용

3.1.4 PLC 기법을 수행한다

기존에 제안되었던 동시성 제어 기법들은 변경된 MBR을 반영하기 위해 여러 노드에 동시에 잠금을 유지하였다. 제안하는 동시성 제어 기법은 [2, 3]처럼 상향 순회하면서 *MBR* 변경을 수행한다. 그러나, 기존의 동시성 제어 기법과는 달리 PLC를 도입하여 잠금 결합을 부분적으로 수행한다. PLC 기법은 MBR이 감소되는 삭제나 분할 연산을 수행하는 경우만 잠금 결합을 수행한다. 삽입으로 인한 MBR 증가 연산은 잠금 결합을 수행하지 않는다. 일반적으로 다차원 색인구조에서는 MBR이 증가되는 연산이 감소되는 연산보다 많이 발생하므로 PLC 기법은 높은 동시성을 보장한다. 그러나, PLC 기법을 올바르게 사용하기 위해서는 트리의 일관성이 깨지는 경우를 피해야한다. 부분적인 잠금 결합을 수행하기 때문에 몇 가지 경우에는 적절한 보상을 해주지 않으면 트리의 일관성이 깨지는 경우가 생길 수 있다. 다음에서 이 보상 방법에 대해서 자세히 기술한다.

엔트리가 단말노드에 삽입되면 그 엔트리를 포함하도록 조상노드들의 MBR을 변경한다. 이때 이를 수행하는 트랜잭션이 철회되거나 또는 트랜잭션이 완료된 후 다른 트랜잭션에 의해서 그 엔트리가 삭제되지 않는 한, 한번 그 엔트리를 포함하도록 조정된 MBR은 항상 삽입된 엔트리를 포함하고 있다. 이 성질을 이용하여 삽입 연산의 경우 조상 노드에 MBR을 반영할 때 자신이 단말 노드에 삽입한 엔트리를 현재 MBR이 포함하고 있는지를 판별해서, 포함하고 있으면 MBR 변경을 수행하지 않고 포함하고 있지 않으면 포함하도록 MBR을 조정한다. 이렇게 하면 잠금 결합을 수행하지 않아서 나중에 수행된 삽입이나 삭제 연산이 먼저 조상 노드에 MBR을 변경하거나 먼저 발생한 다른 삽입 연산보다 먼저 조상 노드에 MBR 변경을 수행하는 경우가 생기더라도 전혀 문제가 발생하지 않는다.

삭제 연산의 경우 잠금 결합을 수행하므로 다른 삽입이나 삭제 연산이 먼저 조상 노드에 반영을 수행할 수 없다. 또한, 삭제 연산이 삽입 연산보다 먼저 상위 노드에 반영되는 경우에도 나중에 수행되는 삽입 연산이 MBR 변경을 정상적으로 수행하므로, 삭제 연산은 포함 여부를 비교할 필요 없이 무조건 MBR 변경을 수행하고 조상 노드로 올라가서 수행을 계속한다. 대부분의 다차원 색인 구조의 응용에서는 삭제연산의 빈도수가 삽입에 비해 훨씬 적으므로 삭제연산에 대해 잠금 결합을 수행해도 전체적인 동시성을 저하시키지 않는다.

그림 4에서는 삽입과 삭제연산이 동시에 수행될 때 PLC가 어떤 과정으로 수행되는 지를 보여준다. 그림과 같은 경우에 삽입은 삭제를 앞지를 수 없기 때문에 세 가지 경우가 가능하다. 그 중 직렬로 수행되는 경우를 제외한 나머지에 대해서 살펴보면 다음과 같다.

첫 번째는 노드 5에서 엔트리 E의 MBR 변경 수행이

T2, T1, T3의 순서로 수행되는 경우이다. 먼저 삭제 연산 T2가 변경을 수행하는데, PLC에서 MBR 감소 연산은 별도의 검사 없이 그대로 수행되므로 E의 MBR을 E2로 변경시킨다. 그 다음은 삽입 연산 T1이 수행된다. T1은 MBR을 증가시키는 연산이므로 PLC에 의해서 새로 삽입된 엔트리가 MBR에 포함되는지의 여부를 검사한다. 현재 MBR E2는 T1이 삽입한 엔트리를 포함하므로 별도의 MBR 변경을 수행하지 않는다. 따라서 T1은 변경을 수행하지 않고 조상 노드로 올라간다. 마지막으로 삽입 연산인 T3가 수행된다. 삽입 연산이므로 엔트리의 포함여부를 비교해서 MBR의 변경여부를 결정한다. 현재상태는 MBR이 E2이므로 T3가 삽입한 엔트리를 포함하지 않는다. 따라서 T3는 MBR E2를 E3로 변경한다.

두 번째는 E의 MBR 변경 수행이 T2, T3, T1의 순서로 수행되는 경우이다. 첫 번째와 마찬가지로 먼저 T2에 의해 MBR이 E2로 변경된다. 다음 T3는 현재 MBR이 E2로 자신이 삽입한 엔트리를 포함하지 않으므로 MBR E2를 E3로 변경한다. 마지막으로 T1이 수행되는데 T1이 삽입한 엔트리가 현재 MBR E3에 포함되므로 변경없이 그대로 조상 노드로 올라간다. 두 경우 모두 MBR 변경이 정상적으로 수행되는 것을 볼 수 있다. 즉 잠금 결합을 수행하지 않는 트랜잭션보다 다른 트랜잭션이 먼저 조상 노드에서 변경을 수행해도 MBR 변경은 문제가 발생하지 않는다.

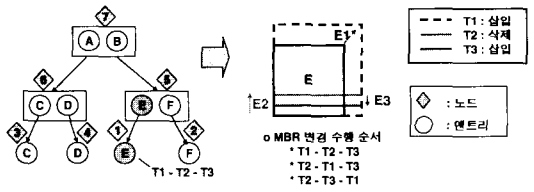

그림 4 PLC를 수행하는 MBR 변경 과정의 예

3.2 삽입 연산

삽입 연산은 새로운 엔트리를 적절한 단말 노드를 찾아서 삽입하고, 새로운 엔트리의 삽입으로 MBR 변경이나 노드 분할이 발생하면 이를 처리해주는 모든 과정을 말한다. 그림 5의 의사코드가 삽입연산을 이루는 하나의 부 모듈이다. ***Insert***는 모든 삽입 연산을 수행하는 주 함수이다. ***FindNode***는 새로운 엔트리를 삽입할 단말 노드와 루트에서 단말 노드까지의 경로를 찾는 함수이다. ***Split***은 노드에서 발생하는 모든 넘침을 처리하는 함수이며 ***FixMBR***은 새로운 엔트리의 삽입이나 분할로 노드의 MBR이 변경되었을 때 조상 노드의 MBR 변경을 수행하는 함수이다.

이 각각의 의사코드에 대해서 자세히 설명을 한다. 삽입을 하려면 먼저 새로운 엔트리를 삽입할 적절한 단말 노드를 찾아야 한다. 이 단말 노드를 찾는 기능을 하는 것이 ***FindNode***이다. ***FindNode***를 수행하면 루트부터 새로운 엔트리가 삽입될 단말 노드까지의 경로가 저장된 경로 스택과 찾은 단말 노드에 대한 배타 잠금과 공유 래치가 결과로 반환된다. ***FindNode***를 수행할 때는 순회하는 노드에 공유 래치를 획득하고 수행을 한다. 새로운 엔트리가 삽입될 적절한 엔트리를 선택하고 그 엔트리가 가리키는 노드로 하향 순회를 계속한다. 다음 노드를 방문하기 전에 현재 노드의 래치를 해제한다. 노드가 비 단말 노드이면 공유 래치를 획득하고 방문한다. 단말 노드이면 배타 잠금과 공유 래치를 획득하고 수행을 종료한다.

FindNode를 통해서 엔트리가 삽입될 단말 노드를 찾았으면 그 노드에 엔트리가 삽입될 여유 공간이 있는가를 체크한다. 여유 공간이 있으면 엔트리를 삽입하고 엔트리의 삽입으로 MBR이 변경되는지를 검사한다. 변경이 발생하면 경로 스택에 저장된 조상 노드들에 반영하고 수행을 마치고, 변경이 발생하지 않으면 수행을 종료한다.

단말 노드에 엔트리를 삽입할 여유 공간이 존재하지 않으면 ***Split***을 호출하여 분할을 수행한다. 분할은 항상 단말 노드에서 조상 노드로 전파된다. 분할이 발생하는 단말 노드에는 ***FindNode***에 의해 배타 잠금과 공유 래치가 획득되어 있다. 탐색을 수행하는 트랜잭션은 노드에 공유 래치만 획득하기 때문에 분할이 수행되는 노드에서 동시 수행이 가능하다. 먼저 분할은 공유 래치를 획득한 상태에서 계산 단계를 수행한다. 계산 단계가 끝나면 공유 래치를 해제하고 배타 래치를 획득한 다음 물리적 분할 단계를 수행한다. 탐색 연산은 배타 래치를 획득하고 수행하는 물리적 분할 단계에서만 노드에 접근할 수 없는데, 이 기간은 전체 분할 과정에서 10% 미만의 작은 부분을 차지한다. 분할 과정의 대부분을 차지하는 계산 단계에서는 탐색 연산의 동시 수행이 가능하다. 따라서 탐색 연산의 지연이 최소화되고 동시성이 향상된다.

노드의 분할이 완료되면, 분할된 노드의 MBR 변경을 조상 노드에 반영하고 새로 생성된 노드의 MBR을 조상 노드에 기록해야 한다. 조상 노드에 배타 잠금을 획득하기 전에 분할된 노드의 잠금은 유지한 상태에서 배타 래치를 해제한다. 즉, 잠금 결합을 수행한다. 이것은

노드가 분할되면 노드의 MBR은 줄어들게 되므로 제안된 동시성 제어 기법에 의해 잠금 결합을 수행한다. 먼저 분할된 노드의 MBR을 반영하고, 새로 생성된 노드의 엔트리를 기록한다. 여유 공간이 있다면 엔트리를 기록하고 조상 노드에 엔트리 삽입으로 변경된 MBR을 반영한다. 여유 공간이 없으면 단말 노드에서와 같은 방법으로 분할을 수행한다. 그림 7은 분할 알고리즘의 의사코드를 보여준다.

단말 노드에 새로운 엔트리를 삽입할 여유 공간이 존재하면 엔트리를 삽입하고, 엔트리의 삽입으로 노드의 MBR이 변경되는지 체크한다. MBR의 변경이 발생하면 FixMBR을 호출하여 경로 스택 상에서 더 이상 변경이 발생하지 않는 노드까지 상향 순회하면서 반영을 수행한다. 그림 8은 PLC를 수행하는 제안하는 MBR 변경 알고리즘의 의사코드를 보여준다.

삽입으로 인한 MBR 변경은 항상 MBR을 증가시킨다. 따라서, FixMBR은 제안된 PLC 기법에 의해 잠금 결합을 수행하지 않는다. 현재 노드의 잠금과 래치를 해제하고 조상 노드에 배타 잠금과 공유 래치를 획득한다. 이 때 잠금 결합을 수행하지 않기 때문에 현재 노드의 잠금을 해제하고 조상 노드의 잠금을 획득하기 전에 다른 트랜잭션에 의해 나중에 수행된 삽입이나 삭제가 먼저 조상 노드의 잠금을 획득하고 수행을 할 수가 있다. 이런 경우 때문에 기존의 동시성 제어 기법들은 MBR 변경을 모두 잠금 결합으로 수행했다. 왜냐하면 나중에 수행된 연산이 먼저 조상 노드의 MBR을 변경하면, 먼저 수행되고 나중에 조상 노드에 도착된 연산에 의해 나중에 수행된 변경이 무시되는 경우가 발생할 수 있기 때문이다. 즉, 트리의 일관성이 깨지는 경우가 발생할 수 있다.

기존의 동시성 제어 기법들은 일관성이 깨지는 문제 때문에 잠금 결합을 수행하였다. 이 결과 MBR 변경은 올바르게 수행되지만 동시성은 떨어지게 되었다. 그러나, 이 논문에서 제안된 PLC 기법은 삽입 연산의 경우 잠금 결합의 형태로 수행되지 않아도 일관성을 유지할 수 있다. 삽입 연산에서는 새로 삽입된 엔트리가 노드의 MBR에 포함되는지만 비교한다. 비교해서 포함되지 않는 경우는 나중에 수행된 다른 연산이 먼저 반영을 수행하지 않은 경우이다. 이 때는 새로운 엔트리를 포함하도록 MBR을 변경시켜주면 된다.

만약 새로운 엔트리가 MBR에 포함된다면 이미 다른 연산이 먼저 반영을 수행한 경우이이다. 이 경우에는 현재의 삽입은 반영을 하지 않고 조상 노드로의 수행을 계속한다. 이렇게 해서 잠금 결합을 수행하지 않아도 트리의 일관성이 유지되고, 잠금 결합으로 인한 동시성 저하가 발생하지 않는다.

```
Insert( Entry leafentry, Node rootnode )

Function Begin
  leafnode = FindNode(leafentry, rootnode, path);
  If ( leafentry의 삽입으로 leafnode 에 넘침이 발생 )
    tree lock 획득;
    Split(leafentry, leafnode, path);
    tree lock 해제;
    모든 노드의 lock을 해제;
    Function End
  Else
    leafnode의 s-latch 해제;
    leafnode의 x-latch 획득;
    leafnode에 leafentry 삽입;
    If ( leafnode의 MBR이 변경 )
      leafnode의 x-latch 해제;
      FixMBR( leafnode, path );
      모든 노드의 lock을 해제;
      Function End
    Else
      leafnode의 x latch 해제;
      모든 노드의 lock을 해제;
    End If
  EndIf

Function End
```

그림 5 삽입 알고리즘의 의사코드(Insert)

```
FindNode(Entry leafentry, Node rootnode, PathStack path)

Function Begin
  node = rootnode;
  node에 s-latch 획득;
  currentlevel = node.level ;
  Push [node, node.nsn] into path;
  Loop
    childentry[node, mbr] =node에서 적절한 자식엔트리를 선택;
    currentlevel을 1 감소;
    node의 s-latch 해제 ;
    node = childentry.node;
    node에 s-latch 획득;
    If (currentlevel == leaf level)
        node에 x-lock 획득;
    End If
    If ( global_nsn > node.nsn )
        // node에는 s latch와 단말노드일 경우 x-lock 획득
        node = 이웃 노드들 중 가장 적절한 node 선택;
    End If
    If ( currentlevel == leaflevel )
        Exit Loop
    End If
    Push [node, node.nsn] into path;
  End Loop
Function End
```

그림 6 삽입 알고리즘의 의사코드(FindNode)

```
Split(Entry leafentry, Node node, PathStack path )

Function Begin
  node의 분할차원 및 분할위치 계산;
  newnode = 새로운 노드 할당;
  newnode에 x-lock과 x-latch 획득;
  entries = 분할차원과 분할위치를 계산하고 이에 따라
            newnode로 이동할 엔트리 선택;
  entries를 newnode로 복사;
  If ( node의 latch가 x-latch가 아님 )
    node의 s-latch를 해제하고 x-latch 획득;
  End If
  node에서 entries를 삭제하고 node 재구성;
  node의 링크를 newnode에 복사 node 의 링크를 newnode에 연결;
  node의 node.nsn을 newnode에 복사;
  global_nsn을 증가시키고 node.nsn에 할당;
  비 단말 노드 엔트리 internalentry[newnode, mbr] 생성;
  node 와 newnode의 x-latch 해제;
  parentnode = POP( path );
  parentnode에 x-latch과 x-lock 획득;
  parentnode에서 node에 대한 엔트리를 찾고 엔트리의 MBR 변경;
  If ( node가 단말 노드가 아님 )
    node와 newnode의 x-lock 해제;
  End If
  If ( internalentry의 삽입으로 parentnode 에 넘침이 발생 )
    SplitNode( internalentry, parentnode, path );
  End If
  parentnode에 internalentry 삽입;
  parentnode의 x-latch 해제;
  If ( parentnode의 MBR이 변경 )
    parentnode의 x-latch 해제;
    FixMBR(parentnode, path );
  End If
Function End
```

그림 7 분할 알고리즘의 의사코드(Split)

```
FixMBR ( Node node, PathStack path )

Function Begin
  parentnode = POP( path ); // node의 부모 노드 결정
  If ( MBR이 감소드는 연산 ) // 삭제/분할 연산
    parentnode에 x-lock과 s-latch 획득;
    node의 x-lock 해제; // 삭제/분할 연산은 잠금 결합 수행
  Else
    parentnode에 x-lock과 s-latch 획득;
  End If
  parentnode에서 node에 대한 엔트리 MBR 변경;
  parentnode의 s-latch 해제;
  If ( parentnode의 MBR이 변경 )
    If ( MBR 증가 연산 ) // 삽입 연산
      parentnode의 x-lock 해제;
    End If
    FixMBR( parentnode, path );
  End If
  parentnode의 latch 및 lock 해제;
Function End
```

그림 8 MBR 변경 알고리즘의 의사코드(FixMBR)

3.3 삭제 연산

그림 9는 삭제 알고리즘의 의사코드를 보여준다. 삭제를 수행할 때는 먼저 삭제할 노드가 있는 단말 노드를 찾아야 한다. 삭제할 단말 노드를 찾을 때는 삽입 연산에서 사용되는 ***FindNode***를 사용해서는 찾을 수 없다. 왜냐하면, 다차원 색인구조에서는 MBR의 겹침과 다른 삽입이나 삭제로 인한 MBR 변경으로 동일한 엔트리를 다시 삽입했을 때, 처음 삽입되었던 그 노드에 다시 삽입된다는 보장을 할 수가 없다. 그러므로 삭제 시에는 삭제할 엔트리가 있는 단말 노드를 찾기 위해 ***GetEntryLocation***을 호출한다.

GetEntryLocation은 루트 노드부터 삭제할 엔트리를 포함하는 모든 노드를 순회하면서 단말 노드까지 내려가 보고 그 곳에 해당 엔트리가 없으면 다시 엔트리를 포함하는 비 단말 노드를 계속 순회하게 된다. 삭제할 노드가 존재하는 단말노드를 찾게 되면 루트 노드로부터 해당 단말 노드까지의 경로가 저장된 스택과 노드에서 삭제할 엔트리의 위치가 결과 값으로 반환된다. 삭제할 엔트리를 찾게되면 ***RemoveEntry***를 호출하여 해당 단말 노드에서 엔트리를 삭제한다. 엔트리 삭제로 MBR 변경이 발생하면 ***FixMBR***을 호출하여 조상 노드들에 MBR 변경을 반영한다. 삭제 연산은 MBR 감소 연산이므로 ***FixMBR*** 수행 시 잠금 결합을 수행한다.

3.4 탐색 연산

탐색 연산의 수행은 [2, 3]에서 제안하는 방법과 같다. 단지 제안하는 방법에서는 탐색 연산 수행 시 래치만 사용하고, 래치 결합은 수행하지 않는다. 이 논문에서는 탐색 연산의 동시성을 최대한 향상시키는데 초점을 두고 있다. 즉, 탐색 연산을 지연시키는 연산들을 최대한 줄여서 탐색의 동시성을 향상시키는 것이다. 탐색을 수행할 때는 오직 공유 래치만을 획득한다. 배타 잠금을 획득하고 수행하는 분할이나 MBR 변경 연산이 공유 래치를 가지고 수행되고 있으면 동시에 수행이 가능하게 되어 탐색 연산의 지연이 줄어들게 된다. 트리를 순회할 때는 항상 하나의 노드에만 래치를 유지하고, 하위 노드에서 발생하는 분할은 링크를 통해서 감지하고 처리할 수 있다.

3.5 알고리즘의 정확성 증명

제안하는 동시성 제어 알고리즘에서는 래치와 잠금을 혼용한다. 동시에 여러 트랜잭션이 수행되면서 래치와 잠금을 획득하는 과정에서 래치-래치, 잠금-잠금 또는 래치-잠금에 의한 교착상태가 발생할 수 있다. 이 절에서는 제안하는 동시성 제어 알고리즘에서는 위와 같이 수행되는 경우에 교착상태가 발생하지 않음을 보인다.

표 1에서는 제안하는 알고리즘의 정확성 증명에서 사용될 기호와 그에 대한 설명을 하고 있다. 이 표는 제안하는 동시성제어 알고리즘에서 발생할 수 있는 모든 연산들을 제시한다. 다차원 색인구조에서 동시에 수행 가

```
Delete( Entry leafentry, Node rootnode )

Function Begin
  leafnode = GetEntryLocation(leafentry, rootnode, path);
  leafnode에서 leafentry 삭제;
  If ( leafnode의 엔트리가 전체의 20%이상 또는 Node Empty )
    If ( leafnode가 Empty )
      parentnode = POP( path )
      parentfnode의 엔트리 MBR을 음수로 변환;
      leafnode 및 parentnode의 latch와 lock을 해제;
      Function End
    Else if ( leafnode의 MBR이 변경 )
      FixMBR( leafnode, path );
    Else
      leafnode의 latch와 lock을 해제;
    End If
  End If
Function End
```

```
GetEntryLocation(leafentry, rootnode, path)
Function Begin
  rootnode에 S-latch 획득;
  push(path,rootnode);
  node = rootnode;
  If(node가 비단말 노드)
    childentries = leafentry를 포함하는 node내의 엔트리들;
    Loop // childentries 의 모든 엔트리에 대해
      GetEntryLocation(leafentry, childentries++ >node, path);
    End Loop
    pop(path);
  Else If(node가 단말 노드)
    If(node에 leafentry와 일치하는 것이 있으면)
      수행을 끝내고 path와 node를 반환;
      Function End
    Else
      Function End
      pop(path);
Function End
```

그림 10 삭제 알고리즘의 의사코드(Delete, GetEntryLocation)

능한 연산과 그렇지 않은 연산이 있을 수 있는데 표 2에서 이 관계를 표시한다. ○는 서로 동시에 수행 가능함을 나타내고 ×는 그렇지 않음을 나타낸다.

표 1 연산의 표기 및 설명

표 기	설 명
OP_{SE}	탐색 연산
OP_{FN}	엔트리를 삽입할 단말 노드를 찾는 연산
OP_{SS}	넘침이 발생한 노드에서 계산 과정을 수행하는 연산
OP_{SX}	넘침이 발생한 노드에서 물리적인 분할을 수행하는 연산
OP_{UM}	노드의 MBR 변경을 조상 노드로 전파하는 연산

표 2 동시에 수행 가능한 연산들

	OP_{SE}	OP_{FN}	OP_{SS}	OP_{SX}	OP_{UM}
OP_{SE}	○	○	○	○	○
OP_{FN}	○	○	○	○	○
OP_{SS}	○	○	×	×	○
OP_{SX}	○	○	×	×	○
OP_{UM}	○	○	○	○	○

표 2로부터 동시에 수행할 수 있는 연산들은 OP_{SE}-OP_{SE}, OP_{SE}-OP_{FN}, OP_{SE}-OP_{SS}, OP_{SE}-OP_{SX}, OP_{SE}-OP_{UM}, OP_{FN}-OP_{FN}, OP_{FN}-OP_{SS}, OP_{FN}-OP_{SX}, OP_{FN}-OP_{UM}, OP_{SS}-OP_{UM}, OP_{SX}-OP_{UM} 그리고 OP_{UM}-OP_{UM} 이다. 노드 분할 연산은 트리잠금에 의해서 직렬화 되므로 OP_{SS}와 OP_{SX}는 동시에 수행될 수 없다. 발생 가능한 연산의 모든 쌍들이 교착상태에 빠지지 않음을 보이면 제안하는 알고리즘은 교착상태를 발생시키지 않는다는 것을 증명할 수 있다.

3.5.1 OP_{SE}-OP_{SE}, OP_{SE}-OP_{FN}, OP_{SE}-OP_{SX}, OP_{SE}-OP_{SS}, OP_{SE}

OP_{SE}는 공유래치만을 이용해서 탐색을 수행한다. 따라서 다른 연산과 래치-잠금간 교착상태가 발생하지 않는다.

3.5.2 OP_{FN}-OP_{FN}

OP_{FN}은 비 단말 노드를 순회할 때 공유래치만을 획득한다. 현재 노드에 래치를 획득하고 하위노드로 내려가기 전에 그 래치를 해제한다. 단말노드에서는 배타잠금을 획득하고 배타래치를 획득한다. 이 잠금과 래치는 엔트리 삽입후 MBR변경(OP_{UM}) 또는 노드분할(OP_{SX}, OP_{SS})을 수행하게 되면 분할과 MBR 확장의 경우에는 상위노드에 잠금과 래치를 획득한 후 해제하고 그렇지 않은 경우에는 바로 해제한다. OP_{FN}과 OP_{FN}의 경우에 잠금과 래치를 획득/해제하는 순서가 같으므로 비 단말 노드에서도 교착상태에 빠지지 않는다.

3.5.3 OP_{FN}-OP_{SS}

OP_{SS}는 상향 순회하면서 배타잠금-공유래치의 순서로 노드에 래치와 잠금을 획득한다. OP_{SS}는 잠금결합을 수행하면서 조상노드들로 연산을 진행시킨다. OP_{FN}은 비 단말 노드에서 잠금을 획득하지 않으므로 교착상태에 빠질 수 없다. 단말노드에서는 두 연산 모두 잠금-래치의 순서로 획득하므로 교착상태에 빠질 수 없다.

3.5.4 OP_{FN}-OP_{SX}

OPS는 먼저 OP_{SS}가 노드에 배타잠금과 공유래치를

획득한 상태에서 공유래치를 해제하고 다시 배타래치를 획득한다. 이 경우은 노드의 래치가 배타래치라는 것을 제외하고는 OP_{FN}-OP_{SS}와 같은 경우이다. OP_{FN}은 OP_{SX}의 배타래치와 충돌이 발생하지만 그 자신이 잠금/래치 결합을 수행하지 않으므로 래치-잠금간 교착상태에는 빠지지 않는다.

3.5.5 OP_{FN}-OP_{UM}

OP_{UM}은 비 단말 노드에서만 수행되며 배타잠금-배타래치의 순서로 획득을 한다. MBR 확장의 경우에는 부모노드의 잠금을 획득하기 전에 현재 노드의 래치와 잠금을 해제하고 MBR 축소의 경우에는 부모노드의 잠금을 획득한 후 현재 노드의 래치와 잠금을 해제한다. OP_{FN}은 비단말 노드에서는 공유래치만 획득한다. 비 단말 노드에서는 서로 공유 래치를 획득하므로 래치의 충돌이 발생하지 않는다. 단말노드에서는 OP_{UM}과 충돌이 발생하는 일이 없다.

3.5.6 OP_{SS}-OP_{UM}

OP_{SS}와 OP_{UM}은 모두 상향 순회한다. 두 연산 모두 배타잠금-공유래치의 순서로 획득을 한다. OP_{SS}가 잠금 결합을 수행하지만 두 연산 모두 잠금-래치를 획득하는 순서가 같고, 순회하는 방향이 같기 때문에 잠금 충돌은 발생하지 않는다. 두 연산 모두 비 단말 노드에서는 서로 공유 래치를 획득하고 잠금을 획득한 상태에서 래치를 획득하므로 래치의 충돌이 발생하지 않는다. 따라서, OP_{FN}과 OP_{UM}은 서로 교착상태에 빠지지 않는다.

3.5.7 OP_{SX}-OP_{UM}

OP_{SX}와 OP_{UM}은 모두 상향 순회하면서 수행을 한다. 두 연산 모두 배타 잠금을 획득한 후 각각 배타 래치와 공유 래치를 획득한다. OP_{SS}가 공유래치를 해제해도 여전히 배타 잠금을 유지하고 있기 때문에 배타잠금을 먼저 획득하는 OP_{UM}은 대기하다가 OP_{SX}가 수행을 마치면 배타 잠금을 획득하고 수행된다. 두 연산은 모두 잠금을 획득한 상태에 래치를 획득하므로 교착상태에 빠질 수 없다. OP_{UM}이 MBR 축소로 인해 잠금결합을 수행한다 하더라도 잠금결합의 방향이 같으므로 교착상태에 빠지지 않는다.

3.5.8 OP_{UM}-OP_{UM}

OP_{UM}은 비단말 노드에서 상향으로 수행이 되며 잠금과 래치의 획득순서가 일정하난. 따라서 두 OP_{UM}은 잠금-래치간 교착상태에 빠지지 않는다.

이상으로 동시에 수행될 수 있는 연산의 모든 경우에 대해 교착상태가 발생하지 않음을 보였다. 따라서, 제안하는 알고리즘은 잠금-래치간 교착상태가 발생하지 않는다.

4. 실 험

4.1 실험 환경 및 방법

이 논문에서는 제안하는 알고리즘을 CIR-트리에 적용시켜 구현하였다. 구현을 한 기반은 바다-DBMS의 하부 저장 시스템인 MiDAS-Ⅲ 이다. 실험에 사용된 컴퓨터는 Sun UltrSparc-Ⅱ 기종에 2개의 CPU, 그리고 512 MBytes의 주기억 공간을 가지고 있다. 제안하는 알고리즘의 비교대상은 [9]에서 제안한 CGIST이며 실험의 공정성을 위해서 MiDAS-Ⅲ를 기반으로 CIR-트리에 적용하여 구현하였다.

표 3 성능 평가 파라미터

파라미터	설 명	값
DS	데이타 개수	50000, 100000
NS	페이지(노드) 크기	4K, 16K
NB	버퍼풀의 버퍼 개수	40, 80, 120
ND	데이타 차원	5, 9, 10, 20
K	K-NN 질의의 K	1, 2, 4
NP	동시에 수행되는 프로세스의 수	50, 100

실험에 사용된 파라미터는 표 3과 같다. 페이지(노드)의 크기는 4K, 16K로 하였으며 실험에 사용된 데이타는 10, 20 차원의 균등분포 데이타와 동영상으로부터 추출한 9차원의 실제 데이타이다. MiDAS-Ⅲ의 버퍼 개수는 40-120개로 변화시켜 가면서 실험하였다. 실험 방법은 먼저 일정 개수의 데이타로 색인을 구축한 후 50-100개의 프로세스를 통해 각각 탐색연산과 삽입연산을 수십에서 수백회 수행시킨다. 이때 총 동시수행 프로세스중 삽입연산을 수행하는 프로세스와 탐색연산을 수행하는 프로세스의 비율을 변화 시켜가면서 측정한 결과를 가지고 탐색연산과 삽입연산의 처리율과 응답시간을 계산한다. 처리율은 단위 시간에 수행되는 트랜잭션의 수이고 응답시간은 하나의 트랜잭션이 수행되는 시간을 의미한다.

실제로 이 논문에는 지면을 줄이는 차원에서 10차원의 균등분포 데이타와 9차원의 실 데이타에 대한 성능 평가 결과만을 제시하고 있다. 차원을 변화 시켜 가면서 측정 했을때는 차원이 증가하게 되면 전체적인 탐색연산의 성능과 삽입연산의 성능이 더 저하되었고 차원이 감소하게 되면 전체적인 성능이 향상되었다. 하지만 동시성 측면에서 두 방법의 성능 차이가 눈에 띄게 나지는 않았다. 동시수행 프로세스의 수도 50 과 100에 대해서 모두 실험을 해보았다. 이때는 동시수행 프로세스

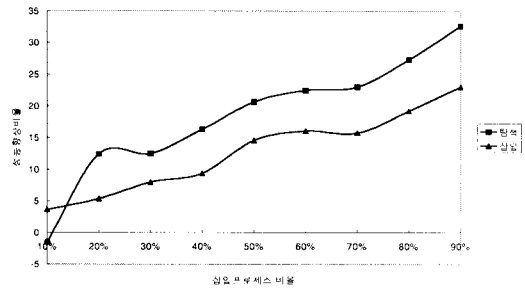

그림 11 탐색과 삽입연산의 성능향상 비율

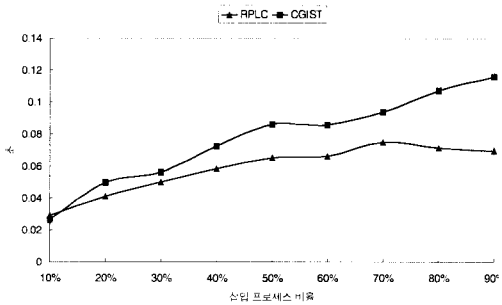

그림 12 탐색연산의 응답시간

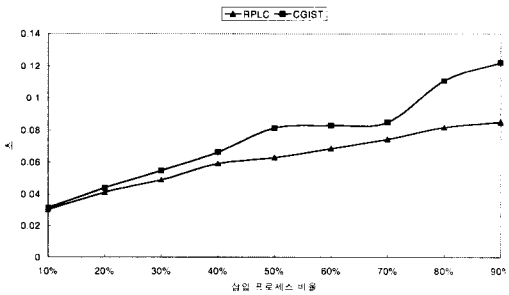

그림 13 삽입연산의 응답시간

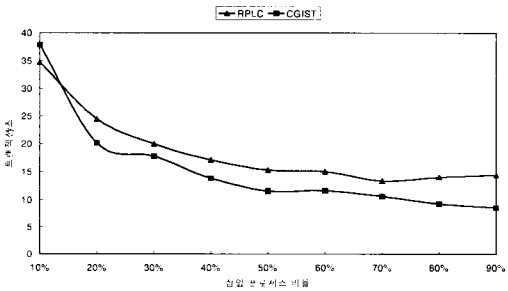

그림 14 탐색연산의 처리율

의 수가 100일 때 더 성능차이가 뚜렷하게 나타났다.

4.2 실험 결과

4.2.1 10차원 균등 분포 데이타

그림 12~그림 15는 전체 50개의 동시 프로세스들 중 삽입프로세스의 비율을 10-90% 로 변화시켜 가면서 CGIST와 RPLC의 삽입 및 탐색 트랜잭션의 처리율과 응답시간을 보여준다. 이 그림들이 나타내는 성능 차이를 그림 11과 같이 요약할 수 있다. 이 그림은 삽입 프로세스의 비율이 증가할 때 성능향상의 정도가 평균 얼마인지를 보여준다. 그림에서 보면 탐색 연산의 경우 삽입 프로세스가 10%일 때 는 약 2% 정도의 성능저하를 보이지만 20% 이상 증가하게 되면 약 13% 이상 향상됨을 볼 수 있고 10%-40% 까지는 평균 18 % 정도의 성능향상이 있음을 볼 수 있다. 삽입프로세스 비율이 10% 일 때 2% 정도의 성능저하가 나타나는 것은 삽입 프로세스가 20% 이상일때의 성능향상정도를 보면 거의 무시할 수 있을 것이다.

삽입에 대한 성능향상 정도는 10%~40%의 삽입 프로세스의 비율일 때 약 10% 정도의 성능향상을 가져왔다. 모든 성능평가 결과를 전체적으로 보아서 RPLC가 CGIST 보다 항상된 성능을 보인다. 이유는 MBR 변경을 위해서 잠금 결합을 수행하지 않으므로 탐색연산에 대한 지연이 CGIST보다 줄게 되고 분할을 수행할 때도 분할을 위한 계산 동안에는 탐색연산이 접근 가능하기 때문이라고 분석할 수 있다. 동시수행 프로세스의 수가 100개 일때가 50개인 경우에 비해서 약 3배가량 더 성능향상을 보였다. 이런 결과를 놓고 볼 때 제안하는 방법이 부하가 더 큰 상황에서 더 좋은 성능을 발휘할 수 있다고 말할 수 있다.

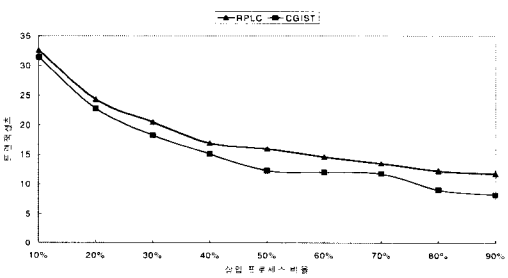

그림 15 삽입연산의 처리율

4.2.2 9차원 실 데이타

그림 17~그림 20은 9차원 실 데이타에 대해서 삽입 프로세스의 비율을 10~90% 로 변화시켜 가면서 삽입 및 탐색 트랜잭션의 처리율과 응답시간을 보여주고 있다. 역시 이 실험에서도 동시수행 프로세스의 개수는 50이었다. 4.2.1에서와 마찬가지로 이 그림들이 나타내는 성능 차이를 그림 12로 요약해 볼 수 있다. 그림에서 보면 탐색 연산의 경우 삽입 프로세스가 10%-40% 일 때 약 36% 정도의 성능향상을 가져오는 것을 볼 수 있다. 이는 균등데이타를 이용한 실험에 비해서 약 2배 이상의 성능 향상 정도이다. 전체적으로 보았을 때도 실 데이타를 이용했을 때가 약 3배정도 더 성능이 향상됨을 볼 수 있다. 삽입에 대해서도 같은 양상을 보인다. 실 데이타를 통한 실험이 약 2배 이상의 성능향상 정도를 보였다.

원인을 분석해 본다면 균등분포의 경우 데이타에 핫-스팟(Hot-Spot)이 존재하지 않는 반면 실 데이타는 전체적으로 데이타의 분포가 한곳으로 집중되는(skewed) 경향을 보인다. 즉, 핫-스팟이 존재한다. 이런 상황에서 삽입과 탐색을 수행하게 되면 특히 빈번히 접근하는 데이타 영역(노드)에서는 보다 많은 연산사이의 충돌이 발생한다. 제안하는 방법이 이런 충돌이 빈번할수록 더 성능향상을 보이는 경향이 있음을 4.2.1에서 밝힌바 있다.

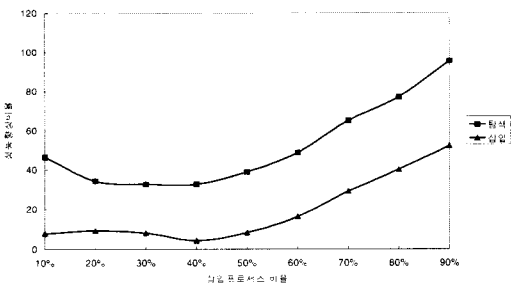

그림 16 탐색과 삽입연산의 성능향상 비율

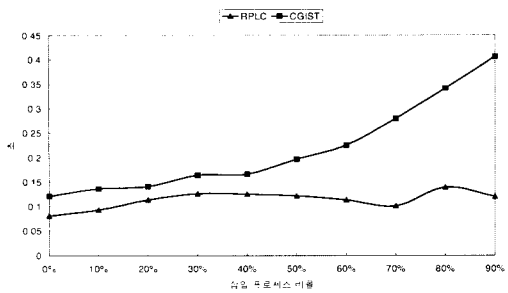

그림 17 탐색연산의 응답시간

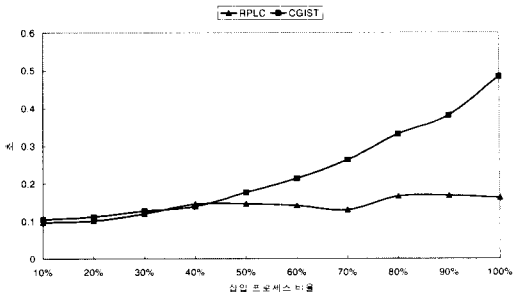

그림 18 삽입연산의 응답시간

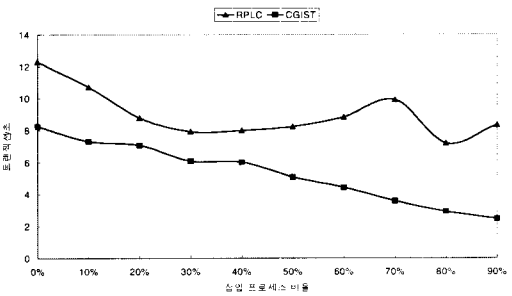

그림 19 탐색연산의 처리율

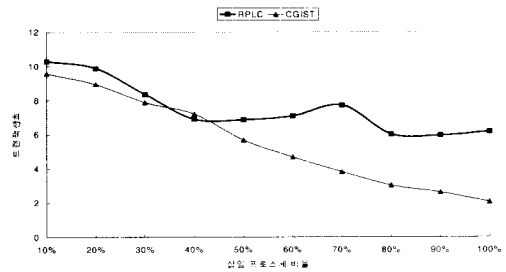

그림 20 삽입연산의 처리율

4.2.3 삭제연산의 성능 평가

그림 21은 삭제연산에 대한 응답시간을 나타낸 그림이다. 총 50개의 동시 수행 프로세스가 각각 50회의 삽입연산을 수행했을 때의 응답시간을 측정한 것이다. 그림의 응답시간과 삽입연산의 응답시간을 비교해보면 약 1.5 배정도 더 느린 것을 알수 있다. 삭제의 경우 Get EntryLocation을 수행하여 삭제할 엔트리가 저장된 단말 노드를 찾게 되는데 이때 GetEntryLocation은 정확 탐색(exact match)방법과 유사한 방법으로 수행이 된다. 이는 FindNode와 비교할 때 훨씬더 복잡하게 처리된다. 또한, 삭제는 래치결합을 이용해 MBR 변경을 수

행하므로 응답시간이 삽입 보다 더 느리다. 실험 결과를 보면 CGIST 보다 RPLC가 삭제연산의 응답시간이 근소하게 앞서는 것을 볼수 있다. CGIST의 경우 MBR 변경은 삽입과 같은 절차로 처리가 된다. 즉, MBR변경을 해야 하는 조상노드들에 다중으로 래치를 획득한다. 응답시간의 차이는 이 원인으로부터 온다고 볼 수 있다.

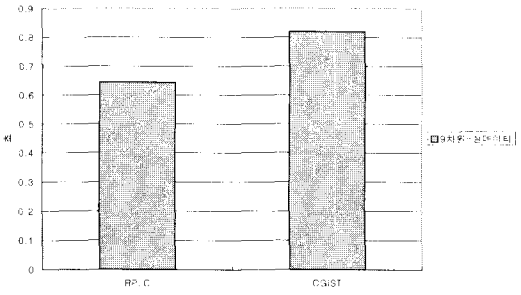

그림 21 삭제연산의 응답시간

5. 결 론

이 논문에서는 다차원 색인 구조에서 질의를 지연시키고 동시성을 저하시키는 두 가지 요인인 분할 연산과 MBR 변경 연산에 대해 효율적으로 대처하는 PLC를 기반으로 하는 동시성 제어 알고리즘을 제안하였다. 제안하는 알고리즘은 탐색 연산이 지연되는 것을 줄이고 변경 연산의 동시성을 향상시켰다. 제안하는 알고리즘에서는 노드 분할 시 질의를 지연시키는 배타 래치의 획득 시간을 최소화하는 방법을 사용하였다. 즉, 노드 분할 과정 중 물리적인 분할이 수행되는 시간만 배타 래치를 획득하고, 나머지 과정에서는 공유 래치를 획득하여 질의가 동시에 수행될 수 있도록 하는 방법이다. 또한 MBR 변경 시에 발생하는 질의의 지연을 줄이기 위해 PLC 기법을 도입하였다. 이 기법은 다차원 색인구조에서 MBR 증가 연산에 비해 발생이 적은 MBR이 감소되는 연산을 수행하는 경우에만 잠금 결합을 수행하여 MBR 변경 연산의 동시성을 향상시킨 방법이다.

제안하는 알고리즘의 성능 평가를 위해 바다 DBMS의 하부 자료저장 시스템인 MiDAS-Ⅲ의 다차원 색인 구조인 CIR-트리에 적용하여 구현하고 GiST의 동시성 제어 기법(CGiST)과 성능을 비교 평가하였다. 성능 평가 결과 CGIST보다 탐색 및 삽입 트랜잭션 모두 처리율과 응답시간에서 제안하는 방법이 우수한 성능을 나타내는 것을 볼 수 있었다. 특히, 균등분포의 데이타보다는 실 데이타를 가지고 실험한 경우와 동시수행 프로세스의 수가 많을수록 성능의 차이가 더욱 뚜렷하게 나타나는 현상을 볼수 있었다. 이런 실험 결과는 제안하는 방법이 보다 작업부하가 큰 환경에서 더 다른 방법에 비해 효과적으로 동작한다는 것을 뒷받침한다.

실제 상용의 DBMS에서는 항상 동시성 제어 기법과 회복 기법이 같이 고려가 된 설계가 이루어져야 한다. 향후 연구에서는 제안한 동시성 제어 알고리즘에 맞는 회복 기법을 고려하여 실제적인 DBMS에서 사용할 수 있는 동시성 제어 기법을 설계하려고 한다. 또한, 이 논문에서 제안한 방법은 유령현상의 고립수준(No-Phantom Isolation Level)을 지원하지 않고 있다. 이 역시 향후 연구에서 고려하도록 하겠다.

참 고 문 헌

[1] A. Guttman, "R-Trees: A dynamic index structure for spatial searching," In Proc. ACM SIGMOD Conf., pp. 47-57, 1984.

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

[3] T.Sellis, N. Roussopoulos and C. Faloutsos. "The R+-Tree: a dynamic index for multi-dimensional objects," In Proc. VLDB Conf., pp. 507-518, 1987.

[4] K. Lin, H. V. Jagadish and C. Faloutsos. "The TV-Tree an index structure for high dimensional data," VLDB Journal, pp. 517-542, 1994.

[5] S. Berchtold, D. A. Keim and H. P. Kriegel. "The X-Tree: An index structure for high-dimensional data," In Proc. VLDB Conf., pp. 28-39, 1996.

[6] 이석희, 유재수, 조기형, 허대영, "CIR-Tree : 효율적인 고차원 색인기법," 한국정보과학회 논문지(B), 한국정보과학회, 제26권 제6호, pp. 724-734, 1999.

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

[7] Jae Soo Yoo, Myung Geun Shin, Seok Hee Lee, Kil Seong Choi, Ki Hyung Cho and Dae Young Hur, "An Efficient Index Structure for High Dimensional Image Data," In Proc. AMCP Conf., pp. 134-147, 1998.

[8] M. Kornacker and D. Banks, "High-Concurrency Locking in R-Trees," In Proc. VLDB Conf., pp. 134-145, 1995.

[11] V. Ng and T. Kamada, "Concurrent Accesses to R-Trees," In Proc. SLSD Conf., pp. 142-161, 1993.

[10] K.V.Ravi Kanth, David Serena and Ambuj K.Singh, "Improved Concurrency Control Techni-

ques for Multi-dimensional Index Structures," In Proc. IPPS/SPDP Conf., pp. 580-586, 1998.
[12] P.L. Lehmann and S.B. Yao, "Efficient locking for concurrent operations on B-Trees," ACM TODS, 6(4), pp. 650-670, 1981.

김 영 호

1999년 충북대학교 정보통신공학과 공학사. 2001년 충북대학교 정보통신공학과 공학석사. 현재 한국전자통신연구원 근무. 관심분야는 SAN(Storage Area Network), 동시성 제어, 고차원색인 구조, 데이타베이스 시스템

송 석 일

1988년 충북대학교 공과대학 정보통신공학과(공학사). 2000년 충북대학교 정보통신공학과(공학석사). 2002년 충북대학교 정보통신공학과(박사과정 수료). 관심분야는 고차원 데이타 색인, 정보검색프로토콜(Z39.50), SGML, OODBMS, 저장 시스템, 회복기법, 동시성제어

유 재 수

1989년 전북대학교 공과대학 컴퓨터공학과(학사). 1991년 한국과학기술원 전산학과(공학석사). 1995년 한국과학기술원 전산학과(공학박사). 1995년 ~ 1996년 목포대학교 전산통계학과 전임강사. 1996년 ~ 현재 충북대학교 전기전자 및 컴퓨터공학부 부교수. 관심분야는 데이타베이스 시스템, 정보검색, 멀티미디어 데이타베이스, 분산 객체 컴퓨팅