다차원 색인구조를 위한 효율적인 동시성 제어기법

김영호[U], 송석일, 이석희, 유재수
충 북 대 학 교 정 보 통 신 공 학 과
{yhkim, prince, seoklee, yjs}@pretty.chungbuk.ac.kr

An Efficient Concurrency Control Scheme for Multi-dimensional Index Structures

Young-Ho Kim[U], Seok-Il Song, Seok Hee Lee, Jae Soo Yoo
Department of Computer and Communication Engineering,
Chungbuk National University

요 약

이 논문에서는 다차원 색인 구조에서 질의를 지연시키는 주된 요인인 노드 분할연산과 MBR(Minimum Bounding Regions) 변경 연산에 대해 효율적으로 대처하는 동시성 제어 기법을 제안한다. 분할 시 탐색이 지연되는 시간을 최소화 하기 위해 대부분의 과정에서 질의와 호환되는 공유 래치를 획득한 후 수행하고 분할이 발생된 노드에 엔트리들이 복사되는 동안만 배타 래치를 획득는 방법을 사용한다. MBR 변경 연산의 동시성을 높이기 위해 부분적인 잠금 결합을 사용한다. 즉, MBR 변경 연산 중 주로 발생되는 MBR이 증가되는 삽입연산은 잠금 결합을 수행하지 않고, 감소되는 삭제 연산만 잠금 결합을 수행한다. 또한 성능 평가를 통하여 제안된 동시성 제어 기법이 GiST의 동시성 제어 알고리즘에 비해 처리율 관점에서 성능이 우수함을 보인다.

1. 서론

지리 정보 시스템, 멀티미디어 데이터 베이스 와 같은 응용들에 대한 요구가 높아지면서 그 핵심기술중 하나인 다차원 및 고차원 색인구조에 대한 연구가 매우 활발히 진행되어 왔다. 현재 그림, 사진, 지도데이터, 캐드 데이터와 같은 이미지 정보는 폭발적으로 증가하고 있는 추세이며 이에 따라 이들을 좀 더 효율적으로 색인 할 수 있는 다차원 및 고차원 색인구조의 중요성은 더욱 더 가중될 것이다. 여러 다차원 색인 구조들이 제안되었고 이들 중 몇몇은 실제 응용에서 사용되고 있다. 하지만 대부분의 응용에서 삽입할 때는 탐색이 발생하지 않도록 하는 동시성제어 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 색인 구조에서 탐색 연산이 지연되는 경우는 노드의 넘침으로 인한 분할과 MBR의 축소 또는 확장으로 인한 MBR 변경을 상위 노드들에 반영할 때이다. 다차원 색인구조의 특성상 노드분할의 비용은 다른 1차원 색인구조보다 매우 높다. 또한, MBR 변경 역시 매번 삽입이나 삭제 시에 발생가능하며 이를 상위 노드들에 전파해야 한다는 부담을 가지고 있다. 기존의 동시성 제어 기법은 이 두 가지 탐색연산을 지연시키는 문제에 대해서 효율적으로 대처하지 못하고 있다. 뿐만 아니라 MBR 변경을 수행하기 위해 잠금 결합을 수행하는데 이는 삽입의 동시성을 저하시킨다. 이 논문에서는 위의 두 가지 경우에 발생할 수 있는 탐색 연산의 지연을 최소화하는 동시성 제어 알고리즘을 제안한다. 또한 MBR 변경 시 수행하는 잠금 결합을 해결하는 방법을 제안한다.

본 논문은 99년도 정보통신부의 대학기초 지원 사업 연구비 지원에 의하여 수행 되었음

이 논문의 구성은 다음과 같다. 2장에서는 관련 연구로서 기존의 다차원 색인구조에서의 동시성 제어 기법에 대해 살펴보고, 3장에서는 제안하는 동시성 제어 기법에 대해서 설명하고, 4장에서는 제안된 알고리즘을 구현하고 성능평가 한다. 마지막으로 5장에서 결론을 맺는다.

2. 관련 연구

기존의 다차원 색인 구조의 동시성 제어 기법들은 크게 잠금 결합 기법과 링크기법으로 분류된다. 잠금 결합을 수행하는 동시성 제어기법으로는 R^{couple}-트리[6] 가 대표적이다. 다음 노드를 순회할 때 먼저 잠금을 획득한 후 현재 노드의 잠금을 해제하는 잠금 결합기법을 사용한다. 또한 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.

이런 문제를 해결하기 위해 링크 기법을 이용하는 동시성 제어 기법이 제안되었다. R^{Link}-트리[3] 와 GiST를 위한 동시성 제어기법(CCGiST)[2]에서는 링크를 통해 상위 노드에 반영되지 않은 하위 노드의 분할을 감지하여 이를 보상할 수 있도록 하고 있다. R^{Link}-트리는 비 단말 노드의 엔트리에 추가적으로 엔트리가 가리키는 노드의 일련 번호인 NSN(Node Sequence Number)을 유지하며 이로 인해 비 단말 노드의 팬 아웃이 작아지는 단점이 있다. CGiST에서는 R^{Link}-트리의 팬아웃 감소문제를 해결하기 위해 전역 계수기를 사용하고 있다. 전역계수기를 사용하면 노드 분할을 수행할 때 현재 노드를 분할하기 전에 먼저 상위 노드에 잠금을 획득하고 분할을 진행해야 하며 분할이 끝날 때까지 분할에 참여하는 모든 노드들에 잠금을 유지해야 한다.

인덱스 변경 연산과 노드 분할에서의 잠금 결합으로 인한 동시성 저하를 줄이고, 노드 분할 시 탐색 연산의 지연을 막기 위해 TDIM (Top-Down Index region modifications), CCU(Copy based Concurrent Update)와 CCU_NQ(Copy based Concurrent Update with Non-blocking Queries)

가 제안되었다[7]. TDIM은 삽입 수행 시 엔트리를 삽입할 노드를 찾기 위해 하향 순회하고, 엔트리 삽입 후 MBR 변경을 반영하기 위해 다시 상향 순회해야하는 오버헤드를 줄인다. 즉, 엔트리를 삽입할 노드를 찾기 위해 하향 순회하면서 MBR 변경을 동시에 수행한다. 탐색 및 MBR변경연산과 호환되는 잠금을 획득하며 트리를 순회하다가 MBR 변경을 수행할때는 탐색과는 호환되고 다른 모든 변경 연산과는 호환되지 않는 잠금으로 변경한다. 만약 현재 방문하는 노드에서 미리 감지하지 못한 분할이 발생했다면 상위 노드로 가서 다시 이와 같은 작업을 반복해서 수행한다. TDIM의 분할은 CCGiST와 거의 유사하여 분할 중에는 탐색 연산이 지연된다.

CCU는 TDIM의 탐색 연산의 지연을 줄이기 위해 제안된 방법이다. 이 기법은 공유 영역의 노드에서 분할을 수행하지 않고 이를 로컬영역으로 복사하여 수행한다. 분할이 다 수행되면, 공유 영역에 모든 다른 연산과 호환되지 않는 배타 잠금을 획득한 후 분할된 노드의 정보를 반영한다. CCU_NQ는 CCU에서 공유 영역에 분할을 반영할 때 탐색이 지연되는 것조차 피하기 위해 제안된 기법이다. 여기서는 LA(Logical Address)를 사용해서 이러한 문제를 해결한다. LA는 노드가 처음 할당될 때 그 노드의 물리적 주소를 갖고, 분할이나 MBR 변경으로 노드가 변경되면 변경된 노드의 주소로 바뀌게 된다.

이 방법들의 기본이 되는 TDIM의 경우 삭제에 대한 고려가 없다. 삭제를 수행하기 위해서는 삽입할 단말 노드 찾는 방법으로는 삭제할 엔트리가 존재하는 단말 노드를 찾을 수 없다. 이를 찾기 위해서는 완전 일치 질의(Exact Match Query)를 수행하여야 하는데 이를 위해서는 가능한 모든 경로를 탐색해야하고, 이 경로를 가지고 다시 하향식 MBR 변경을 수행해야 하는 수고를 감소해야 한다. 또한 CCU_NQ의 경우 분할 수행 후 상위 n레벨까지 분할이 발생되면 n+1 개의 쓰레기 노드가 생기게 된다.

이 논문에서는 삭제와 삽입이 모두 고려된 MBR변경 방법과 분할 수행 시 추가적인 쓰레기 처리를 수행하지 않는 동시성 제어 기법을 제안한다.

3. 제안하는 동시성 제어 알고리즘

3.1. 제안하는 알고리즘의 특징

이 논문에서 제안하는 동시성 제어 알고리즘의 특징은 다음과 같다. 첫 번째로, 제안하는 기법은 링크 기법을 기반으로 한다. 탐색을 수행하는 트랜잭션이 잠금 결합을 하지 않고 순회를 하여도 상위 페이지에 반영되지 않은 하위 노드의 분할을 감지하여 처리할 수 있다. 전역 계수기를 사용하여 NSN을 할당하는 방법을 취하는데, 전역계수기는 LSN (Log Sequence Number)을 이용한다.

두 번째, 래치와 잠금을 혼용한다. 래치는 한 노드에 여러 트랜잭션들이 접근할 때 이 들간의 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 완료 될 때까지 유지한다. 여기서는 회복에 대해서는 언급하지 않는다.

세 번째, 탐색 연산은 공유 래치 만을 획득한다. 다음 노드로 순회를 할 때는 현재 노드의 래치를 해제하고 다음 노드의 공유 래치를 획득한다. 따라서 탐색을 수행하는 트랜잭션은 방문하는 노드에만 공유 래치를 유지한다. 탐색 연산은 잠금을 획득하지 않으므로 배타 잠금과 충돌이 발생하지 않는다. 그러므로 MBR 변경이나 분할이 수행되는 트랜잭션과도 동시에 수행될 수 있으므로 다차원 색인 구조에서 가장 많이 발생하는 탐색 연산의 동시성을 최대한 보장할 수 있다

네 번째는, 노드의 분할 수행 시 모든 분할 과정에 배타 래치를 획득하지 않고, 분할이 발생된 노드에 엔트리를 복사하는 동안만 배타 래치를 획득한다. 즉, 분할 과정에서 탐색 연산의 지연을 최소화한다. 분할 연산의 대부분은 분할 차원의 선택, 분할 위치의 선택, 새로운 노드 할당과 선택된 분할 위치에 의해 엔트리를 나눠서 새로 할당된 노드에 복사하는 과정이 차지하는데, 이러한 연산의 수행은 노드를 변경시키는 것이 아니므로 공유 래치를 획득하고 수행할 수 있다.

다섯 번째는 부분적인 잠금 결합의 수행이다. 기존에 제안되었던 동시성 제어 기법들은 변경된 MBR을 반영하기 위해 여러 노드에 동시에 잠금을 유지하였다. 제안하는 부분적인 잠금 결합(Partial Lock Coupling)기법은 삽입 연산의 동시성을 향상시킨다. 삽입으로 인한 MBR 증가 연산은 잠금 결합 을 수행하지 않고, 삭제로 인한 MBR 감소 연산과 분할의 경우에는 잠금 결합을 수행하는 기법이다. 엔트리가 노드에 삽입되면 그 엔트리를 포함하도록 MBR이 조정되어 상위 노드에 반영된다. 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 상위 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 상위 노드에 MBR을 변경하거나 먼저 발생한 다른 삽입 연산보다 먼저 상위 노드에 MBR 변경을 수행하는 경우가 생기더라도 전혀 문제가 발생하지 않는다. 삭제 연산의 경우 잠금 결합을 수행하므로 특별한 비교 없이 MBR을 반영하고 상위 노드로 수행을 계속한다. 대부분의 다차원 색인 구조의 응용에서는 삭제연산의 빈도수가 삽입에 비해 훨씬 적으므로 삭제연산에 대해 잠금 결합을 수행해도 전체적인 동시성을 저하시키지 않는다.

지금까지 제안하는 동시성 제어 기법의 특징에 대해서 살펴 보았다. 네 번째와 다섯 번째 특징으로 인해 탐색 연산의 지연이 최소화되고, 삽입 연산의 동시성을 최대한으로 높일 수 있다. 이 논문에서는 고차원 색인 구조의 특징들을 고려해 가장 많이 수행되는 탐색 연산과 그 다음으로 많이 수행되는 삽입 연산의 동시성을 최대한 높이는 기법을 설계하고 구현한다. 다음에서는 동시성 제어 기법을 사용한 삽입 연산에 대해 기술한다.

3.2. 삽입 연산

삽입 연산은 새로운 엔트리를 적절한 단말 노드를 찾아서 삽입하고, 새로운 엔트리의 삽입으로 MBR 변경이나 노드 넘침으로 인한 분할이 발생하면 이를 처리해주는 모든 과정을 말한다. 삽입을 하려면 먼저 새로운 엔트리를 삽입할 적절한 단말 노드를 찾아야 한다. 이 단말 노드를 찾는 기능을 하는 것이 FindNode이다. FindNode를 수행하면 루트부터 새로운 엔트리가 삽입될 단말 노드까지의 경로가 저장된 경로 스택과 찾은 단말 노드에 대한 배타 잠금과 공유 래치가 결과로 반환된다. FindNode를 수행할 때는 순회하는 노드에 공유 래치를 획득하고 수행을 한다. 새로운 엔트리가 삽입될 적절한 엔트리를 선택하고 그 엔트리가 가리키는 노드로 하향 순회를 계속한다. 다음 노드를 순회할 때는 현재 노드의 공유 래치를 해제한다. 다음 노드가 비 단말 노드이면 공유 래치를 획득하고 순회를 계속하고 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 MBR의 변경 여부를 검사하여, 변경이 발생하면 경로 스택에 저장된 상위 노드에 반영하고 수행을 마친다.

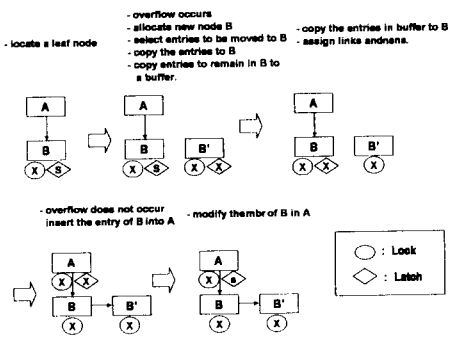

그림 1 분할 수행 과정

삽입할 여유 공간이 존재하지 않으면 SplitAll을 호출하여 분할을 수행한다. 분할은 항상 단말 노드에서 상위로 전파된다. 분할이 발생하는 단말 노드에는 FindNode에 의해 배타 잠금과 공유 래치가 획득되어 있다. 탐색을 수행하는 트랜잭션은 노드에 공유 래치만 획득하기 때문에 배타 잠금과의 충돌이 발생하지 않으므로 분할되는 노드에서 수행이 가능하다. 분할은 항상 새로운 엔트리의 삽입으로 단말노드에서 발생한다. 분할에서 수행되는 것으로는 분할 차원의 선택, 분할 위치의 선택, 새로운 노드의 생성, 선택된 차원과 위치에 의한 엔트리들의 나눔, 나눠진 두 그룹의 엔트리 중 새로운 노드에 복사, 링크와 NSN의 할당, 버퍼에 있는 엔트리의 원 노드에의 복사 등이 있다. 이 중 대부분의 수행은 공유 래치를 획득한 상태에

서 수행되고, 링크와 NSN의 할당, 버퍼에 있는 엔트리의 노드로의 복사만 배타 래치를 획득한 상태에서 수행된다. 탐색 연산은 배타 래치를 획득한 노드에 접근할 수 없는데, 이 기간은 전체 분할 과정에서 아주 작은 부분을 차지하므로 대부분의 기간은 탐색 연산의 수행이 가능하다. 따라서 탐색 연산의 동시성은 매우 향상된다. 그림 1에서는 분할 수행 과정에서의 잠금과 래치의 사용을 나타내고 있다.

노드의 분할을 수행하면 분할된 노드의 MBR 변경을 상위 노드에 반영하고, 새로 생성된 노드의 MBR을 상위 노드의 기록해야 한다. 상위 노드는 FindNode에서 만들어진 경로 스택에 저장되어 있다. 상위 노드에 배타 잠금과 배타 래치를 획득하기 전에 분할된 노드의 배타 래치를 해제하고 잠금은 유지한다. 즉, 래치 결합은 하지 않고 잠금 결합을 수행한다. 이것은 노드가 분할되면 노드의 MBR은 줄어들게 되므로 제안된 동시성 제어 기법에 의해 MBR 감소 연산은 잠금 결합을 수행한다. 먼저 분할된 노드의 MBR 을 반영하고, 새로 생성된 노드의 엔트리를 기록한다. 만약, 여유 공간이 있다면 엔트리를 기록하고 경로 스택상의 상위 노드에 엔트리 삽입으로 변경된 MBR을 반영한다. 여유 공간이 없으면 단말 노드에서와 같은 방법으로 분할을 수행하고 상위에 반영한다.

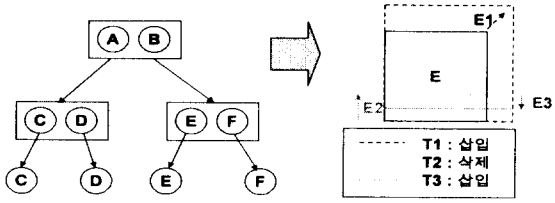

그림 2 MBR 변경 수행 과정

더 이상 MBR 변경이 없거나 루트 노드까지 경로 스택에서 하나씩 노드를 꺼내서 수행하고 루트 노드까지 여유 공간이 없으면 루트 노드의 분할을 수행한다. 루트 노드의 분할은 단말 노드나 비 단말 노드에서의 분할과 다르게 수행된다. 색인 구조에서 루트 노드는 항상 같은 값으로 유지한다. 루트 노드가 분할될 때는 새로운 두 개의 노드를 생성해서 선택된 차원과 위치에 의해 나뉘어진 두 그룹의 엔트리를 새로운 두 개의 노드에 복사하고 루트 노드에는 두 노드의 MBR을 기록한다.

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 반영을 수행한다. 그림 2에서는 동일 단말 노드에서 삽입과 삭제 연산들의 수행 후 MBR 변경에 대해 나타내고 있다. 삽입으로 인한 변경은 항상 MBR을 증가시킨다. 따라서 FixMBR은 제안된 동시성 제어 기법에 의해 잠금 결합을 하지 않고 수행한다. 현재 노드의 배타 잠금과 배타 래치를 해제하고 상위 노드에 배타 잠금과 배타 래치를 획득한다. 이 때 잠금 결합을 수행하지 않기 때문에 현재 노드의 잠금을 해제하고 상위 노드의 잠금을 획득하기 전에 다른 트랜잭션에 의해 나중에 수행된 삽입이나 삭제가 먼저 잠금을 획득하고 수행을 마친 다음, 현재의 삽입 연산이 잠금을 획득할 수 있다. 그래서 기존의 동시성 제어 기법들은 MBR 변경을 모두 잠금 결합으로 수행했다. 왜냐하면, 나중에 수행된 연산이 먼저 상위 노드의 MBR을 변경하면, 먼저 수행되고 나중에 상위에 도착된 연산에 의해 나중에 수행된 변경이 무시되는 경우가 발생할 수 있기 때문이다. 그런데 이 기법들은 MBR 변경은 올바르게 수행되지만 동시성의 저하를 가져왔다. 그러나, 이 논문에서 제안된 동시성 제어 알고리즘은 삽입 연산의 경우 잠금 결합없이 새로 삽입된 엔트리가 MBR에 포함되는지 비교하여 포함되지 않으면 나중에 수행된 연산이 먼저 반영되지 않은 것이므로 새로운 엔트리를 포함하도록 MBR을 변경하고, 만약 새로운 엔트리가 MBR에 포함된다면 다른 연산이 먼저 반영을 수행한 것으로 현재의 삽입은 반영을 하지 않고 상위 노드로의 수행을 계속한다.

4. 구현 및 성능 평가

이 논문에서 제안한 동시성 제어 기법의 우수성을 보이기 위해 기존의 동시성 제어기법중 하나인 CCGiST와 비교하였다. 제안하는 알고리즘 과 CCGiST를 CIR-Tree에 적용시켜 실제 DBMS의 하부 저장 시스템인 MiDAS-III에서 구현하고 실험하였다. 실험에 사용된 플랫폼은 Sun UltrSparc II, 2개의 CPU, 512 MBytes의 주기억 공간에 Solaris가 탑재된 워크스테이션이었다. 실험에 사용된 데이터는 20 차원의 실수형 가상데이터로 균등분포를 갖는다. 노드의 크기는 4K, 버퍼의 개수는 100개로 하였다.

실험 방법은 먼저 20,000개의 데이터로 DB와 인덱스를 구축한다. 그리고 500개를 삽입하는 프로세스 4개와 50개씩 탐색하는 프로세스를 4개에서 12개까지 4개씩 변화시키면서 동시에 수행시킨다. 질의는 KNN질의를 사용했으며 K는 1로 고정하였다. 평가에 사용된 척도는 처리율이다.

그림 3에서 제안하는 알고리즘과 CCGiST에서 탐색연산의 처리율을 보여주고 있다. 그림에서 볼 수 있듯이 제안하는 방법이 기존의 방법보다 처리율 면에서 20% 정도 높음을 알 수 있다. 하지만 이는 그림상의 수치만을 가지고 평균을 낸 것이고 곡선의 형태로 보아 그 차이는 더욱더 커질 것이 확실하다.

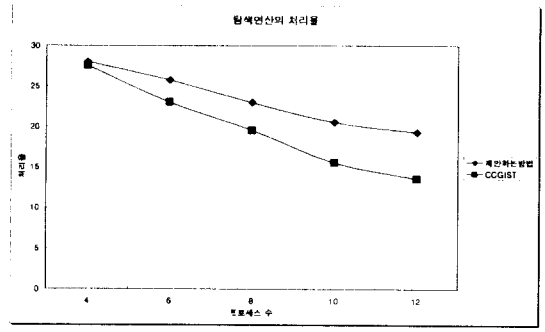

그림 3 탐색연산의 처리율

5. 결 론

이 논문에서는 다차원 색인 구조에서 질의를 지연시키는 두 가지 요인인 분할연산과 MBR 변경연산에 대해 효율적으로 대처하는 동시성 제어 기법을 제안하였다. 또한 제안하는 방법의 우수성을 보이기 위해 자료저장 시스템인 MiDAS-III의 다차원 색인 구조인 CIR-Tree[1]에 적용하여 구현하고 기존의 동시성 제어 기법과 성능을 비교 평가하였다. 성능 평가 결과 기존의 방법보다 탐색연산의 처리율 면에서 우수함을 볼 수 있었다. 향후 연구에서는 보다 다양한 형태의 실험을 통해 제안하는 동시성 제어 기법의 특성을 파악한다.

참고 문헌

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

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

[3] M. Kornacker and D. Banks, "High-Concurrency Locking in R-Trees," In Proc. 21st Int'l Conference on VLDB, pages 134-145, September 1995.

[4] 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, 17(1), pages 94-162, March 1992.

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

[6] V. Ng and T. Kamada, "Concurrent Accesses to R-Trees," In Proc. of Symposium on Large Spatial Databases, pages 142-161, 1993.

[7] K.V.Ravi Kanth, David Serena, Ambuj K.Singh, "Improved Concurrency Control Techniques for Multi-dimensional Index Structures," Parallel Processing Symposium, IPPS/SPDP pages 580-586, 1998.