# R-Tree에서 지연 없는 검색을 위한 버전 기반의 동시성 제어 기법

°김민성*, 김명근*, 배해영*
*인하대학교 컴퓨터공학과
e-mail: {°jinminsheng, kimmkeun, hybae}@dblab.inha.ac.kr

# An Concurrency Control Technique Based on Version Techniques for Non Blocking Queries in R-Tree

°Min-Sheng Jin*, Myoung-Keun Kim*, Hae-young Bae*
*Dept of Computer Science, Inha University

요 약

R-Tree 색인은 기존의 공간 데이터베이스관리시스템에서 공간 데이터 검색을 위하여 많이 사용되고 있는 공간 색인기법이다. 하지만 R-Tree 색인에서 기존의 잠금(Lock) 기반 동시성 제어는 갱신연산의 잠금으로 인해 검색연산의 블록킹 오버헤드(blocking overhead)가 발생한다.

본 논문[1]에서는 R-Tree 색인에서 검색연산의 블록킹 오버헤드의 주요 원인이 되는 노드 분할 연산과 MBR(Minimum Bounding Rectangle) 갱신연산에 대해 각각 노드단위와 노드 엔트리 단위의 버전(Version)을 생성하고 유지하여 동시에 발생하는 검색연산이 갱신연산으로 인한 지연이 없이 자신에 알맞은 버전을 읽음으로써, 검색성능을 높일 수 있는 버전 기반의 동시성 제어 기법을 제안한다.

## 1. 서론

인터넷 및 이동 통신기기에서 공간정보를 이용하는 다양한 온라인 서비스들은 공간 객체에 대한 검색이 빈번히 발생하며, 검색결과에 대한 빠른 응답시간을 요구한다. 이러한 응용 서비스들의 수요를 충족시키기 위하여 공간 데이터베이스 관리 시스템에서 공간색인에서의 지연 없는 검색연산을 위한 동시성 제어 기법에 대한 연구가 필요하다.

R-Tree 색인기법[1]은 기존의 공간 데이터베이스 관리시스템에서, 공간 데이터 접근을 위하여 많이 사용하고 있는 공간색인 기법이다. R-Tree 색인을 위한 동시성 제어 기법에 대한 기존의 연구는 잠금 기법을 기반으로 활발히 진행되었다[3, 4, 5, 6]. 하지만 R-Tree 색인에서의 잠금 기반의 동시성 제어 기법은 R-Tree의 갱신연산의 고비용성 때문에 갱신연산으로 인한 검색연산의 블록킹 오버헤드를 갖는다. 그 원인은 다음과 같다. 첫 번째, R-Tree에서 단말 노드에 대한 엔트리의 삽입, 삭제 연산은 상위 노드 엔트리의 MBR 변경을 일으킬 수 있으며, 또 이 변경은 루트 노드까지 전달될 수 있다. 두 번째, 다차원 공간에서 공간 객체에 대한 순차적인 정렬순서가 없으므로 노드 분할 시, 1차원 색인에 비해 노드 분할을 위한 비용이 크다. 따라서 검색연산과의 충돌을 피하기 위하여 갱신연산은 종료될 때까지 변경에 참여되는 모든 노드에 배타적 잠금(Exclusive Lock)을 유지하고 있어야 하므로 검색연산은 갱신연산으로 인해 오래 동안 지연된다. 이러한 문제점을 해결하기 위하여 여러 가지 동시성 제어 기법[3, 4, 6]들이 제안 되었지만 이들 대부분이 잠금 기반의 기법으로써 갱신연산으로 인한 검색연산의 지연을 줄일 수는 있지만 완전히 제거하지는 못한다.

갱신연산으로 인한 검색연산의 지연을 제거하기 위한 효율적인 방법 중의 하나가 버전기법을 이용하

1) 본 연구는 대학 IT연구센터 육성・지원사업의 연구결과로 수행되었음

는 것이다. 본 논문에서는 R-Tree 색인구조에서 버전을 이용하여 갱신연산으로 인한 검색연산의 지연 없는 버전 기반의 동시성 제어기법을 제안한다.

본 논문의 나머지 구성은 다음과 같다. 2장에서는 R-Tree 동시성 제어를 위한 여러 관련연구와 문제점을 살펴보고 3장에서는 본 논문에서 제안하는 R-Tree 색인을 위한 버전 기법을 설명하고 4장에서 버전 기반 R-Tree색인을 위한 동시성 제어에 대해 설명한 다음 5장에서 결론을 제시한다.

## 2. 관련 연구

제안된 기존의 R-Tree 색인을 위한 동시성 제어 기법은 잠금 기반의 동시성 제어 기법과 링크(Link) 기반의 동시성 제어 기법으로 분류된다.

잠금 기반의 동시성 제어 기법은 잠금 결합(Lock Coupling)기법[1]이 대표적이다. 이 기법은 검색연산을 위해 노드를 순회할 때 검색할 노드에 먼저 공유 잠금(shared lock)을 선점하고 다음 노드를 검색하기 위해서는 해당 노드에 공유 잠금을 선점한 후 현재 노드의 잠금을 해제하는 잠금 결합을 사용한다. 또한 노드 분할을 진행하거나 갱신연산으로 인한 MBR 변경을 상위 노드에 반영할 때는 갱신연산이 끝날 때까지 변경에 참여되는 모든 노드들에 배타적 잠금을 유지 한다. 이 기법은 검색연산이나 갱신연산을 위해 하나의 트랜잭션이 여러 개 노드에 잠금을 유지하고 있어야 되므로 검색효율이 낮다. 이런 문제를 해결하기 위하여 링크 기법을 이용한 동시성 제어 기법이 제안 되였다. M.Kornacker등이 제안한 $R^{Link}$-Tree[2]와 GiST를 위한 동시성 제어 기법(CGiST)[3]은 형제노드(sibling node)를 링크로 연결하여 링크를 통해 상위 노드에 아직 반영되지 않은 노드 분할을 감지하고 이를 보상한다. 하지만 $R^{Link}$-Tree에서는 노드 분할 감지를 위하여 비단말 노드의 엔트리에 해당 자식 노드의 생성 순서인 LSN(Logical Sequence Number)을 추가적으로 유지하며 이로 인해 비 단말 노드의 팬아웃(fanout)이 감소되고 전체적인 트리의 높이가 높아져 검색 질의 성능이 저하되는 문제점이 있다. CGiST는 $R^{Link}$-Tree의 팬아웃 감소문제를 해결하기 위하여 전역계수기를 유지하고 각각의 노드에 노드순서번호(Node Sequence Number)를 부여하여 노드 분할을 감지하였지만 노드 분할을 위하여 먼저 상위 노드에 잠금을 선점하여야 하고 또 분할이 끝날 때까지 분할에 참여하는 모든 노드들에 잠금을 유지하여야 하는 문제점이 있다.

결론적으로 R-Tree색인에서 잠금 기반의 동시성 제어 기법은 갱신연산으로 인한 검색연산의 지연을 완전히 제거할 수 없다. 본 논문에서 제안하는 동시성 제어 기법은 버전 기법을 사용하여 갱신연산으로 인한 검색연산의 지연을 제거한다.

## 3. R-Tree 색인을 위한 버전 기법

### 3.1 노드 구조

제안 기법에서 단말 노드와 비단말 노드는 같은 구조를 갖는다. [표 1]은 제안 기법의 노드 자료구조를 설명하고 있다. 각각의 노드는 노드 헤더(Headder)와 엔트리들로 구성된다. 노드 헤더에는 단말 노드와 비 단말 노드를 구분하기 위한 플래그(Flag), 노드에 포함된 엔트리 개수, 노드 분할을 감지하기 위한 플래그, 갱신연산 시 다른 갱신연산과의 충돌을 방지하기 위한 잠금 등의 정보가 유지된다. 엔트리에는 자신의 버전 엔트리를 가리키는 포인터(Pointer) 값이 저장된다.

[표 1] 노드구조

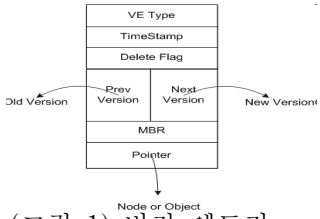
| | | |
|---|---|---|
| 노드 헤더 | NodeType | 노드 타입을 구분하기 위한 플래그 |
| | Latch | 갱신연산에 대한 노드의 잠금 |
| | SplitFlag | 노드 분할을 나타내는 플래그 |
| | RecCnt | 노드에 포함된 엔트리 개수 |
| 엔트리 | VerEntry | 버전을 가리키는 포인터 |

### 3.2 논리적 버전

제안 기법에서는 노드 엔트리의 MBR 변경 기록을 저장하기 위하여 버전 엔트리(VE:Version Entry) 개념을 도입한다. 버전 엔트리는 노드 엔트리의 MBR이 변경 될 때마다 생성되는 논리적 단위이다. 새로 생성된 버전 엔트리는 노드 엔트리의 기존의 버전 리스트에 연결 되여 노드 엔트리의 새로 갱신된 정보를 반영한다. 제안 기법에서는 이렇게 노드 엔트리 단위로 변경을 반영하는 버전을 논리적 버전으로 정의한다.

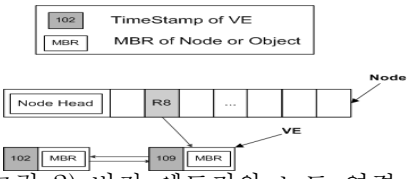
버전 엔트리에는 기존의 공간 색인의 노드 엔트리가 갖고 있던 정보 외에 버전 유지를 위한 일련의 유용한 정보를 저장한다. (그림 1)에서 버전 엔트리에 대해 설명하고 있다. 버전 엔트리에는 버전 엔트리 유형 식별을 위한 플래그가 저장되어 있다. 즉 비 단말 노드 엔트리를 위한 버전 엔트리인지 아니면 단말 노드 엔트리를 위한 버전 엔트리인지를 구분하기 위해서이다. 그 외에 버전 식별을 위한 타임스탬프(TimeStamp)와 이전 버전과 연결하기 위한 포인터 및 새로운 버전을 가리키기 위한 포인터, 엔트리의 삭제여부를 알려주는 삭제 플래그, 자식 노

드의 모든 엔트리들을 포함하는 MBR 혹은 공간 객체의 MBR값이 저장되고 또한 자식 노드를 가리키는 포인터 혹은 공간 객체를 가리키는 RID 등 정보가 유지된다.




(그림 1) 버전 엔트리

버전 엔트리들은 이중 연결 리스트(double linked list)로 연결된다. 노드의 엔트리는 항상 노드엔트리에 대한 변경이 발생할 때마다 버전 엔트리 리스트 중에서 가장 최신 버전을 가리키도록 변경된다. (그림 2)에서 버전 엔트리들로 구성된 버전 연결 리스트와 노드 엔트리 사이의 연결을 보여주고 있다. 그림에서 버전 엔트리들은 이중 연결 리스트로 연결되어 있고 노드엔트리는 타임스탬프 값이 가장 큰 버전 엔트리를 가리키고 있음을 볼 수 있다.



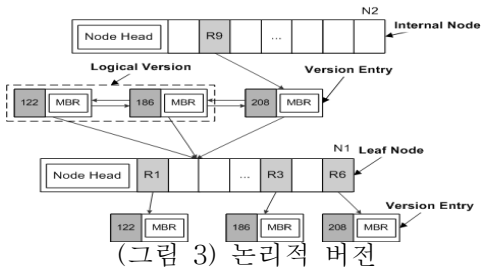

(그림 2) 버전 엔트리와 노드 연결

(그림 3)는 노드 엔트리의 MBR 변경에 의한 논리적 버전 생성 개념을 설명하고 있다. 그림에서 노드 N1에서의 엔트리 삽입으로 부모노드 N2의 엔트리 R9가 변경되게 되고 이로 인해 타임스탬프 값이 122, 186, 208인 R9의 버전 엔트리가 차례로 생성된다. 그림에서 비 단말 노드 엔트리 R9의 버전 엔트리 중, 타임스탬프 값 122, 186을 가지는 버전 엔트리들은 비 단말 노드 엔트리 R9의 갱신 전 정보를 반영하는 논리적 버전이다.

### 3.3 물리적 버전

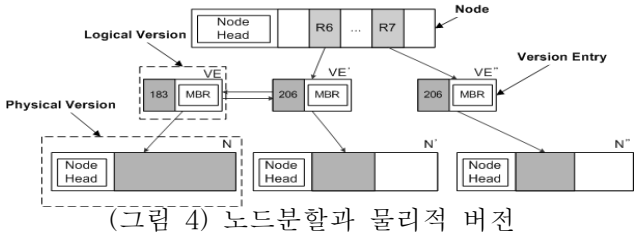
노드 분할이 발생할 시에 제안 기법은 노드 단위의 버전을 유지한다. 노드 분할은 기존 노드는 그대로 유지하고 노드의 복사 본을 생성하여 분할을 수행한다. 기존 노드는 노드 분할 전의 노드 상태를 반영하는 노드 단위의 버전으로 유지된다. 이렇게 노드 단위로 변경을 반영하는 버전을 제안 기법은 물리적 버전으로 정의 한다.

노드 분할을 수행하면 상응한 부모 노드 엔트리의 MBR 변경이 발생한다. 이를 위하여 상응한 부모 노드 엔트리의 버전 리스트에 새로운 버전 엔트리를




(그림 3) 논리적 버전

생성하여 추가한다. 이 버전 엔트리에는 분할 되여 새로 생성된 노드에 대한 MBR 변경을 반영한다. 새로 생성된 버전 엔트리는 분할로 인해 새로 생성된 노드를 가리킨다. 새로 생성된 나머지 노드를 위하여 상응한 부모 노드에 새로운 엔트리가 삽입 된다. 부모 노드에 대한 엔트리 삽입은 새로운 버전 엔트리를 생성하고 삽입된 노드 엔트리에 새로 생성된 버전 엔트리를 가리키는 포인터를 저장하게 된다. (그림 4)는 노드의 분할 과정과 노드 분할로 인한 물리적 버전 생성 및 논리적 버전과 물리적 버전 연결을 설명하고 있다.




(그림 4) 노드분할과 물리적 버전

(그림 4)에서 노드 N은 엔트리 삽입 공간이 없다. 이때 노드 N에 대한 새로운 엔트리의 삽입은 노드 N의 분할을 일으키게 된다. 노드 분할을 위하여 노드 N의 복사 본 N′를 생성한다. 다음 노드 N의 복사 본 N′에서 분할을 수행한다. 결과로 노드 N′, N″가 생성된다. 기존 노드 N은 노드 N′에 대한 물리적 버전으로 된다. 기존 노드 N에 대응한 버전 엔트리 VE는 노드 분할로 인한 노드 엔트리 R6의 논리적 버전이 되고 노드 N과의 연결 상태를 그대로 유지한다. 다음, 부모 노드의 엔트리 R6에 상응한 새로운 버전 엔트리를 추가한다. 새로 생성된 버전 엔트리 VE′는 노드 분할로 인해 새로 생성된 노드 N′를 위하여 추가된 것이다. 버전 엔트리 VE″는 노드 분할로 인해 새로 생성된 노드 N″를 위한 부모 노드 엔트리의 삽입을 위하여 생성된 것이다.

### 3.4 버전의 타임스탬프

제안 기법에서 버전들은 갱신연산 종료 시에 갱신연산의 일련의 순차적인 순서인 갱신연산계수(W-OSN: Write Operation Sequence Number)를 버전의 타임스탬프 값으로 저장한다. W-OSN은 색

인의 전체 영역에서 참조되는 계수기로써, 색인에 대한 갱신연산이 종료될 때마다 단조롭게 하나씩 증가한다. 검색연산은 검색을 시작할 때 W-OSN 값을 기억하고 이것을 버전 선택의 기준으로 삼는다. 하지만 검색연산은 W-OSN값을 증가 시키지 않는다.

## 4. 버전 기반 R-Tree 색인을 위한 동시성 제어

제안 기법은 버전 기반 R-Tree 색인에서 공간 객체에 대한 검색, 삽입, 삭제 세 가지 연산을 지원한다. 동시에, 제안 기법은 오래된 버전으로 인한 공간 오버헤드를 최소로 줄이기 위하여 사용되지 않는 오래된 버전에 대해 연산자(Operation)단위로 주기적으로 확인하고 삭제한다. 하지만 본 논문에서는 버전 쓰레기 삭제에 대해서 제시하지 않는다.

이 장의 나머지 부분에서는 삽입과 검색 알고리즘에 대해서 설명한다. 삭제과정은 삽입과 유사함으로 본 논문에서 생략한다.

### 4.1 삽입 알고리즘

삽입 연산은 색인의 노드에 새로운 키 값을 위한 엔트리들을 삽입하는 과정이다. 제안하는 기법의 삽입 연산은 크게 두개 단계로 나뉜다. 즉 삽입연산 수행단계와 삽입연산 종료단계이다. 삽입연산 수행단계는 키 삽입에 적합한 단말 노드 검색과정과 노드에 대한 엔트리 삽입과정 및 이로 인한 부모 노드들의 엔트리 조정 과정을 포함한다. 삽입연산 종료단계는 삽입연산으로 인해 새로 생성된 버전 엔트리들에 실제 타임스탬프 값을 설정하고, 새로운 버전 생성으로 인한 이전 버전 엔트리들을 쓰레기 버전 수집기와 연결시키며, 삽입연산을 위해 노드들에 선점하였던 잠금들을 해제하는 과정을 거친다. 삽입연산은 삽입과정에 다른 갱신연산과의 충돌을 피하기 위하여 갱신할 노드에 잠금을 선점한다. 그러나 이 잠금은 검색연산에서 무시된다. 삽입연산 과정에 부모 노드의 분할을 감지하면 현재 삽입연산을 취소하고 삽입연산을 재 시작한다.

### 4.2 검색 알고리즘

검색 연산은 단말 노드에서 검색 질의 범위에 속하는 모든 엔트리들을 찾는 과정이다. 공간 객체에 대한 검색은 색인의 루트 노드로부터 시작한다. 검색연산은 시작할 때 현재의 W-OSN값을 저장한다. 검색연산에서 비 단말노드에 대한 검색은 노드의 엔트리들에 대해 질의 영역에 겹치는지를 하나씩 조사하여 하위 노드로의 탐색 경로를 결정하기 위한 것이다. 노드 엔트리에 대한 검색은 노드 엔트리들이 가리키는 버전 리스트를 따라가 현재 저장하고 있는 W-OSN 값보다 타임스탬프 값이 작은 버전들 중에서 타임스탬프 값이 가장 큰 버전을 선택한다. 즉 W-OSN 값보다 타임스탬프 값이 작은 버전 중에서 가장 최신 버전을 선택한다. 다음 선택된 버전의 MBR과 질의 영역의 사각형이 겹치는지 판단하고 만약 겹친다면 하위 노드로 이동하여 탐색을 계속한다. 이 과정을 반복하며 전체 트리를 탐색하여 질의에 부합되는 공간 객체들을 검색하여 반환한다. 제안 기법은 검색 과정에 검색할 노드에 대한 잠금을 선점하지 않는다. 따라서 검색연산은 갱신연산으로 인한 블록킹이 발생하지 않는다. 이는 갱신에 대한 버전 유지를 통하여 가능하다.

## 5. 결론

본 논문에서는 R-Tree 색인에서 갱신연산으로 인한 검색연산의 지연이 없는 버전 기반의 동시성 기법을 제안 하였다. 본 논문에서 제안한 동시성 제어 기법은 갱신연산으로 인한 노드 엔트리의 MBR의 변경에 대하여 엔트리 단위의 논리적 버전을 생성한다. 노드 분할을 위해서는 노드 단위의 물리적 버전을 생성하여 갱신연산으로 인한 검색연산 지연의 주요 원인이 되는 문제점을 해결 하였다.

## 참고문헌

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

[2] C.Mohan " ARIES/KVL: A Key-value locking method for concurrency control of multi-action transactions operating on B-tree indexs." Proceedings of the Int. Conf. on VLDB, August 1990

[3] 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.

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

[5] 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.

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