주기억 데이타베이스에서 공간 데이타에 대한 효율적인 인덱스 구조

강은호° 김경창
홍익대학교 컴퓨터공학과
{ehkang°, kckim}@cs.hongik.ac.kr

An Efficient Index Structure for Spatial Data in Main Memory Database

Eunho Kang° Kyungchang Kim
Dept. of Computer Engineering, Hongik University

요 약

주기억 데이타베이스 시스템은 기존의 디스크 기반 데이타베이스 시스템과 달리 빠른 처리속도와 주기억 장치의 효율적인 사용이 주된 관심 사항이다. 본 논문에서는 주기억 데이타베이스에서 공간 데이타를 위한 효율적인 인덱스구조를 제시한다. 기존에 제시된 주기억 데이타베이스를 위한 인덱스 기법으로는 T-트리, Hash 계열 기법등이 제시되었으나, 이러한 모든 인덱스 기법은 1차원 데이타를 위한 인덱스 기법으로 공간 데이타에는 적용이 불가능하다. 이러한 제약을 극복하기 위해서 본 논문에서는 T-트리에 R-트리 개념을 추가 하였다.

1. 서 론

최근 들어 컴퓨터 하드웨어 기술 혁신으로 대용량의 메모리 칩이 계속 개발되고 있으며, 이와 함께 실시간 응용시스템의 요구가 증대되고 있다. 이러한 실시간 응용시스템을 위한 처리 방법으로 주기억 데이타베이스(Main Memory Database ,MMDB)가 제안되었다[1][2].

주기억 데이타베이스란 전체 데이타베이스의 원본(primary store)을 주기억장치에 상주시켜 응용프로그램들에 대한 실시간 처리를 보장해주는 시스템이다.

주기억 데이타베이스 시스템을 위한 인덱스 기법으로는 AVL-트리, B-트리, T-트리, T*-트리등이 제안되었다. 그러나, 이러한 인덱스 기법은 정수, 실수, 문자 등과 같은 1차원 데이타에만 가능하고 2차원 이상의 데이타를 포함하는 공간 데이타에는 적용이 불가능하다.

본 논문에서는 주기억장치 내에 전체 데이타베이스를 상주시키는 것으로 가정하고, 이 상황에 적합한 새로운 공간 데이타에 대한 인덱스 구조를 제안하고자 한다.

본 논문의 구성은 2장에서 관련연구를 살펴보고 3장에서는 본 논문에서 제시하는 새로운 주기억 데이타베이스에서 공간 데이타의 인덱스 구조에 대해 설명하며, 4, 5장에서는 성능 분석과 결론 및 향후 연구과제를 제시한다.

2. 관련연구

디스크 기반 공간 데이타에 대한 인덱싱 기법은 해쉬 기반(hash-based)과 트리 기반(tree-based) 기법으로 분류될 수 있다. 해쉬 기반(hash-based) 기법은 그리드 파일을 기반으로 하며 대표적으로 그리드(grid) 파일, 그리드 파일의 변형인 EXCELL 기법, 두레벨 그리드(two-level grid) 파일, 트윈 그리드(twin grid)등이 있다. 트리 기반 방법은 계층화된 검색 트리를 기반으로 하며 대표적으로 R-트리[3], 이에 대한 변형인 R*-트리[4], R+-트리, 그리고 Quad 트리, k-d 트리 등이 있다. 이 중 해쉬 기반 기법은 균등하지 않은 분포를 지니는 공간 데이타에 대해서는 좋지 않은 결과를 가져온다는 점과 오버플로우 발생 및 이에 따라 효율이 저하되는 문제점을 지닌다. 따라서 많은 디스크기반 공간 데이타베이스에서는 트리 기반 인덱싱 기법을 선호하고 있으며 이중 R-트리와 그 변형 모델이 가장 많이 사용되고있다.

T-트리와 T*-트리는 AVL-트리와 B-트리의 특성을 결합해서 새롭게 변형되어 나온 트리 구조로서 빠른 처리속도와 주기억 장치 사용의 최적화라는 주기억 데이타베이스의 특성을 위해 Lehman이 제안했다[5][6]. T-트리, T*-트리는 한 노드안의 여러 개의 데이타들을 가지는 B-트리의 특성과 이진검색 및 높이 균형의 AVL-트리 특성을 결합하여 만든 인덱스 구조이다.

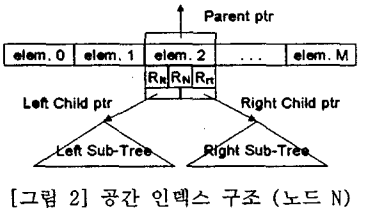
T-트리와 T*-트리는 B-트리와 같은 디스크 기반 인덱스 방법들에 비해 빠른 검색 속도와 효율적인 메모리 사용을 보장해주는 인덱스 기법이지만, 정수, 실수, 문자와 같은 1차원 데이타에 대해서만 적용이 가능하다는 문제점을 지닌다.

3. 주기억 데이타베이스에서 공간 데이타 인덱스 구조

본 논문에서 제시하는 새로운 인덱스 구조는 T-트리를 기반으로 한다. 그러나, 기존의 T-트리는 공간 데이타를 표현할 수 없기 때문에 T-트리 구조에 디스크 기반 공간 데이타베이스에서 가장 효율적이며 많이 사용되는 R-트리의 개념을 도입한다.

3.1 인덱스 구조

공간 데이타를 둘러싸는 최소 직사각형으로 공간 데이타를 표현 할 때 본 논문에서 제시하는 새로운 인덱스 구조는 높이 균형 트리인 T-트리 구조를 기본으로 사용하며 각 트리 노드의 구조는 다음 [그림 1]와 같다.



[그림 2] 공간 인덱스 구조 (노드 N)

[그림 1]와 같이 제시되는 인덱스 노드의 구조는 T-트리 노

본 연구는 한국과학재단 특정기초연구(과제번호:R01-2001-000-00540-0(2002)) 지원으로 수행되었음.

드와 같이 부모노드에 대한 포인터, 왼쪽 자식 노드에 대한 포인터, 오른쪽 자식 노드에 대한 포인터, 여러개의 엔트리들, 그리고 새롭게 정의되는 왼쪽 서브트리의 최소 직사각형(R_{lt}:Rectangle of left subtree), 자신 노드 N의 엔트리들에 대한 최소 직사각형(R_N : Rectangle of node N), 오른쪽 서브트리의 최소 직사각형(R_{rt}:Rectangle of right subtree)으로 구성된다. 각 최소 직사각형란 R-트리에서와 같이 I 값으로 정의된다.

$I = (I_0, I_1, \ldots, I_{n-1})$, 여기서 n은 차원 수를 의미하며 I_i는 최소 직사각형의 i-1차원 값으로 최소값과 최대값으로 이루어진다.

왼쪽 서브트리의 최소 직사각형이란 왼쪽 서브 트리내의 모든 엔트리들을 포함하는 최소 직사각형을 의미하고, 오른쪽 서브트리의 최소 직사각형 역시 오른쪽 서브트리내의 모든 엔트리들에 대한 최소 직사각형을 의미한다. 자신 노드에 대한 최소 직사각형이란 자신 노드의 모든 엔트리들을 포함하는 최소 직사각형을 의미한다.

노드의 각 엔트리들은 R-트리의 리프 노드 엔트리와 마찬가지로 (tuple-id, rectangle)의 형태로 이루어진다. 이때 tuple-id는 주기억 데이타베이스이기 때문에 주기억 장치내에 위치한 공간 데이타에 대한 포인터로서 주기억장치내의 데이타에 대한 완전 주소값 또는 데이타를 포함하는 페이지 번호와 페이지내에서의 오프셋으로 표현될 수 있다. *rectangle*은 해당 데이타를 포함하는 최소 직사각형(I)에 해당된다. T-트리 노드 안의 엔트리들은 최소값에서 최대값순으로 정렬되어 저장되는 반면, 새롭게 제시하는 트리구조에서는 노드안 엔트리들간의 정렬이 이루어지지 않는다. 이는 공간 데이타를 문자, 정수, 실수와 같이 1차원적으로 정렬시킬 수 없음에 기인한다.

그 이외에 본 논문에서 제시하는 트리 노드의 구조는 T-트리 노드구조와 일치한다. 노드의 종류는 자식 포인터의 존재유무에 따라 내부노드(internal node), 반-리프노드(half-leaf node), 리프노드(leaf-node)로 구별되며, 트리의 구조는 T-트리와 같은 높이 균형의 2진 트리 형태를 지닌다. 그리므로, 높이의 불균형이 발생되면 로테이션을 수행해서 트리의 높이가 균형이 되도록 유지한다.

3.2 검색 기법

앞에서 설명한 바와 같은 인덱스 구조를 사용한 포인트, 지역에 대한 검색을 위해서는 자신노드의 엔트리들에 대한 최소 직사각형(R_N), 왼쪽 서브트리의 직사각형(R_{lt}), 오른쪽 서브트리의 직사각형(R_{rt})을 이용한다. 어떤 지역에 대한 검색 요청이 들어올 경우 우선, 자신 노드의 최소 직사각형과 교차되는 지점이 있는지 검사후, 존재시엔 자신 노드안 엔트리들에 대한 구체적인 검색이 시작된다. 그런다음, 왼쪽 서브트리 최소 직사각형과의 교차점이 있는지 검색후, 존재하면 왼쪽 자식 노드를 방문하고, 같은 방식으로 오른쪽 서브트리 최소 직사각형과의 검색작업을 수행한다. 2차원 데이터에 대한 제시하는 트리 예제는 [그림 2]과 같다.

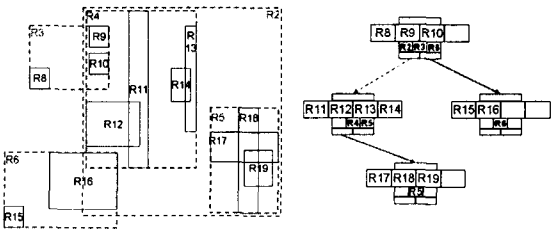
[그림 2]에서 만약 R17 지역 안의 한 지점을 검색할 경우, 우선 루트 노드에서 자신(R3)과의 교차 여부를 확인후, 왼쪽 서브트리(R2)와의 교차 여부, 오른쪽 서브트리(R6)와의 교차 여부를 검색한다. 그러면, 해당 지점은 왼쪽 서브트리와 교차점이 있는 것을 알 수 있고, 왼쪽 자식 노드(R4)를 루트로 하여 다시 검색 루틴을 수행해 오른쪽 서브트리(R5)를 방문한다. 이렇게 해서 방문한 오른쪽 자식 노드에서 원하는 엔트리 R17을 검색해 낼 수 있다. 검색 알고리즘은 다음과 같다.

· 검색(Search) 알고리즘

step1. 루트노드의 최소 직사각형(R_N)을 이용해 찾고자하는 사각형 S 또는, 포인터 P의 값과 교차 지점이 있는지 판단하여 교차시, 루트 노드의 각 엔트리들에 대한 교차여부를 조사해 교차하는 엔트리들을 반환한다.

step2. 루트노드의 왼쪽 서브트리의 최소 직사각형(R_{lt})과 S 또는 P가 교차 지점이 있으면 왼쪽 자식 노드를 루트로 하여 step1을 호출한다.

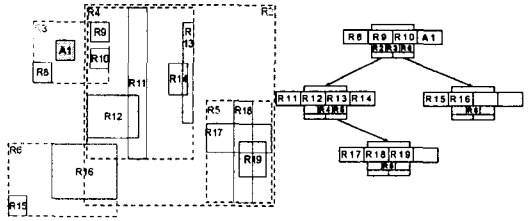
step3. 루트노드의 오른쪽 서브트리의 최소 직사각형(R_{rt})과 S 또는 P가 교차 지점이 있으면 오른쪽 자식 노드를 루트로 하여 step1을 호출한다.



[그림 2] 새 트리 구조

3.3 삽입 기법

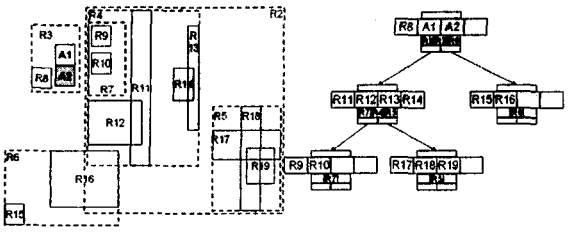
삽입 기법은 R-트리의 삽입 기법과 비슷하다. 차이점은 R-트리에서는 삽입시 후보 노드는 무조건 리프 노드가 되어야 하지만 여기서 제시된 기법에서는 리프가 아닌 노드에서도 삽입이 가능하다. 만약 삽입으로 인해 노드의 오버플로우가 발생할 경우, R-트리 혹은 R*-트리의 분할 알고리즘을 이용해 엔트리들을 기존의 노드와 새롭게 생성한 노드에 분산 배치한다. 새롭게 생성된 노드는 분할이 발생된 노드를 중심으로 최소 직사각형의 증가크기가 최소가 되는 왼쪽 서브트리 또는 오른쪽 서브트리로 찾아 들어간후, 리프 혹은 반-리프 노드의 자식 노드가 되도록 배치한다. 이는 검색시 최소 직사각형의 크기를 줄여주고, 트리의 불균형 발생시 T-트리의 로테이션 알고리즘 이용을 원할히 하기 위한 방안으로 고안되었다. [그림 2]에서 제시된 인덱스 구조에 새롭게 A1, A2를 삽입할 경우 인덱스 구조는 다음과 같은 변환 과정을 거친다.



[그림 3] A1 삽입시 구조

A1이 삽입될 경우 우선 루트노드를 시작으로 자신(R3)에 삽입할 경우와 왼쪽 서브트리(R2)에 삽입할 경우, 오른쪽 서브트리(R6)에 삽입할 경우를 비교해 최소직사각형이 제일 적게 증가하는 위치를 선택한다. 그러므로 새로 삽입된 A1은 루트에 삽입되고 [그림 3]과 같은 인덱스 구조를 생성한다.

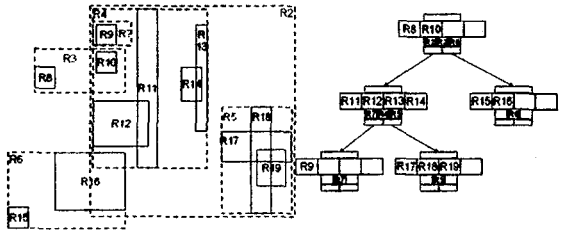
[그림 3]에서 다시 A2를 삽입할 경우에는 위와 같은 방법으로 최적의 인덱스 노드(R3)를 선택한다. 그러나 A2를 R3에 삽입하기 위한 빈 엔트리가 존재하지 않기 때문에 노드 분할이 발생한다. 그러므로 R-트리나 R*-트리의 분할 알고리즘을 이용해 새로운 노드(R7)를 생성한 다음, 새로 생성된 노드(R7)의 연결 위치를 결정한다. 그 과정은 분할이 발생된 노드(R3)를 루트로 새 노드(R7)을 포함할 경우 최소 직사각형이 적게 증가되는 왼쪽(R2) 오른쪽(R6) 서브트리를 검색해 들어간다. 선택한 자식 노드(R4)가 리프 또는 반-리프 일 경우, 트리의 높이를 고려해 높이균형이 이루어지도록 새 노드(R7)를 선택노드(R4)의 자식노드가 되도록 연결한다[그림 4].



[그림 4] A2 삽입시 구조

3.4 삭제 기법

인덱스 노드 안의 엔트리를 삭제를 할 경우에는 해당 엔트리를 포함하는 노드를 검색해 삭제한다. 본 논문에서 제시하는 트리구조는 리프뿐만 아니라 모든 위치의 노드에 데이타를 저장하기 때문에 삭제는 모든 노드에서 이루어질 수 있다. 만약, 내부노드에서 삭제로 인해 언더플로우가 발생할 경우에는 삭제 발생 노드의 서브트리 엔트리들 중 가장 가까운 위치의 노드 엔트리를 검색하여 삭제 발생 노드로 이동시킨다.



[그림 5] A1, A2 삭제시 구조

노드 안의 최소 엔트리 수를 2로 정의할 때, [그림 4]에서 A1, A2를 삭제할 경우의 트리 구조는 다음 [그림 5]과 같다.

A1이 삭제될 경우에는 단순히 노드에서 해당 엔트리를 삭제한 후 알고리즘이 종료되지만, A2를 삭제할 경우에는 루트 노드에서 언더플로우가 발생한다. 이런 경우에는 루트안의 엔트리 R8과 최소의 직사각형을 유지하는 노드 R7를 검색한 후, R7에서 최소의 직사각형을 형성하는 엔트리 R10을 선택해 루트로 이동시킨다.

4. 성능 분석

기존의 주기억장치 데이타베이스 환경하에서 효율적인 인덱스 기법은 많이 제시되었으나, 이는 모두 1차원 데이터를 위한 인덱스 구조이므로, 본 논문에서 제시하는 인덱스 구조와는 비교가 불가능하다. 그러므로, 본 논문에서는 기존에 공간 데이타에 가장 많이 사용되는 R-트리 계열 인덱스 기법간의 논리적인 성능을 비교한다. 성능 비교를 위해 우선 R-트리 계열의 인덱스 구조를 디스크가 아닌 주기억 데이타베이스 환경으로 변경시켰다는 가정 하에 성능을 비교한다. 성능비교는 트리에 대한 검색, 삽입, 삭제 시간 비교를 한다.

트리의 전체 노드수를 n으로 정의하고, 한 노드의 엔트리수를 m으로 정의 할 경우 검색, 삽입, 삭제 시간 비교는 아래 <표 1>와 같다.

검색의 경우 R-트리 계열은 무조건 리프노드까지 방문을 해야 하지만, 새로운 구조에서는 항상 리프 노드까지 방문해야 될 필요가 없기 때문에 위의 표와 같은 결과를 갖는다. 삽입시에도 R-트리 계열은 항상 리프노드를 방문해야 되며, 최악의 경우 분할 작업이 리프에서 루트까지 진행 될 수 있기 때문에 위와 같은 결과를 갖는다. 그러나, 새로운 인덱스 기법에는 모든 노드에 삽입이 이루어질 수 있으며, 분할 발생시에도 한번의 리프노드까지 방문으로 모든 작업이 종료된다. 삭제의 경우, 언더플로우가 발생하면 R-트리 계열의 인덱스 기법에서는 발생 노드의 엔트리들은 모두 재삽입 과정을 거치게 되므로 위와 같은 결과를 갖는다.

		새로운 인덱스	R-트리 계열
검색시	최소	$O(1)$	$O(m \log_m n)$
	최대	$O(\log_2 n + m)$	$O(m \log_m n)$
삽입시	최소	$O(1)$	$O(m \log_m n)$
	최대	$O(\log_2 n + m)$	$O(2m \log_m n)$
삭제시	최소	$O(1)$	$O(m \log_m n)$
	최대	$O((m+2)\log_2 n)$	$O(m^2 \log_m n)$

<표 1> 인덱스 기법 시간 비교

5. 결론 및 향후 연구 과제

본 논문에서는 주기억 데이타베이스에서 공간 데이타를 위한 효율적인 인덱스 구조를 제안하였다. 지금껏 제안되어 온 공간 데이타에 대한 인덱스 구조는 디스크 기반의 데이타베이스를 위한 구조였기 때문에 디스크 접근 횟수의 최소화와 디스크 저장 공간의 효율적 사용에 그 초점이 맞춰져 있었다. 그러나, 주기억 데이타베이스에서는 디스크 기반 데이타베이스와는 달리 빠른 처리 속도와 메모리 공간 사용의 최적화라는 목적을 두고 있다.

본 논문에서는 이러한 문제점에 기인해서, T-트리에 R-트리 개념을 도입한 새로운 인덱스 구조를 제안하였다. 새로운 인덱스 구조는 높이 균형의 이진 검색 트리 형태로, 하나의 노드에는 여러개의 데이타 엔트리들을 저장하며, 추가적으로 각 자신 노드 최소 직사각형 (R_N)과 왼쪽 서브트리 최소 직사각형(R_{lt}), 오른쪽 서브트리 최소 직사각형(R_{rt})을 저장한다.

본 논문에서 제시하는 인덱스 구조는 R-트리와 달리 이진 검색에 의해 내부 노드에서 완료될 수 있기 때문에 빠른 검색이 가능하다.

향후 연구과제로는 제안된 인덱스 구조에 대한 구체적인 구현 알고리즘과 시뮬레이션에 의한 성능 분석이 필요하며 새로운 인덱스 구조를 사용하는 조인과 같은 질의처리 방법에 대한 연구가 필요하다.

6. 참고 문헌

[1]. Tobin J. Lehman, Eugene J. Shekita, Luis-Felipe Carera "An Evaluation of Starburst's Memory Resident Storage Component", IEEE Trans. on knowledge and Data Eng., vol. 4, December 1992.

[2]. Margarte H. Eich "Main Memory Database Research Directions" Technical Report 880-CAE-35

[3]. Guttman, A. "R-tree: A dynamic index structure for spatial searching", In Proceedings of the ACM SIGMOD International conference on Management of data, 1984

[4]. N. Beckmann, H.-P. Kriegel, R. Schneider, and B. Seeger, "The R*-tree: An Efficient and Robust Access method for Points and Rectangles", In Proc. ACM SIGMOD International Conference on management of Data, pp. 332-331, 1990

[5] Tobin J. Lehman, Michael J. Carey "A Study of Index Structures for Main Memory Database Management Systems", in Proceddings 12th Int'l. conf. on Very Large Databases, kyoto, Aug. 1986, pp294-303

[6] 최공림, 김경창, 김기룡, " T*-트리 : 주기억 데이타베이스에서의 효율적인 색인기법" 한국통신학회 논문지, 제 21권 제 10호, 1996