이동체 데이터베이스를 위한 R-tree 기반의 메인 메모리 색인

이 창우[0], 안 경환, 홍 봉희
부산대학교 컴퓨터공학과
{cwlee, khan, bhhong}@pusan.ac.kr

Main Memory Index Based on R-tree for Moving Object Databases

Changwoo Lee[0], Kyounghwan An, Bonghee Hong
Dept. of Computer Engineering, Pusan National University

요 약

최근에는 물류 및 수송 관리, 교통 정보 서비스 등과 같은 위치 기반 서비스의 요구가 증대되고 있다. 위치 기반 서비스에서 클라이언트들의 빈번한 보고 데이터를 처리하기 위해서는 서버에서 메인 메모리 DBMS를 유지하는 것이 필요하다. 그러나 기존 연구에서는 이러한 이동체 데이터베이스를 위한 메인 메모리 색인이 없으므로 이에 적합한 색인이 필요하다.

다차원 색인으로 영역 질의에 뛰어난 성능을 보이는 색인으로 R-tree가 있는데, 이는 디스크 환경을 고려하여 설계되었기 때문에 메인 메모리에서는 효율을 보장하지 못한다. 이 논문에서는 R-tree를 변형하여 이동체 데이터베이스를 위한 메인 메모리 색인을 제시한다. 이 논문에서 새로 제시한 성장 노드 구조와 동적 재구성 및 큰 영역을 가진 노드 분할 정책은 영역 질의의 성능을 향상시킨다. 실험은 제안한 색인이 이동체 데이터베이스를 위한 적합한 메인 메모리 구조임을 보여준다.

1. 서론

위치 기반 서비스를 위해서는 이동체의 위치 정보를 관리하고 빠른 검색을 지원할 수 있는 이동체 데이터베이스가 필요하다. 기존의 이동체 데이터베이스를 위한 시공간 색인으로는 디스크 기반의 색인이 대부분이다. 그러나, 이동체의 수가 많아지거나 데이터의 보고되는 회수가 많아질 경우, 서버의 병목 현상(bottle neck)으로 인하여 이를 실시간으로 처리할 수 없다. 따라서, 클라이언트의 요구에 빠르게 응답할 수 있다는 점으로 인해 메인 메모리 색인이 새롭게 제기되고 있다.

메인 메모리 색인으로는 T-tree[1]가 가장 널리 알려져 있는데, 이는 1차원 데이터를 위한 색인이다. 이를 다차원인 이동체 데이터베이스 환경에 적용할 경우, 보고되는 시간 데이터가 계속 커지게 된다. 따라서, 정렬 시에 큰 키(key) 값으로 인한 색인의 회전(rotation) 비용의 급격한 증가로 인하여 색인의 성능이 아주 나빠지게 된다. 기존의 디스크 기반의 다차원 색인들은 디스크 환경을 고려하여 만들어졌기 때문에, 가장 큰 목표는 디스크 I/O회수를 최소화하는 것이다. 그러므로 디스크 기반 색인을 그대로 메인 메모리에 적용할 경우 효율적인 사용을 보장하지 못한다. 따라서 이동체 데이터베이스를 위한 메인 메모리 색인이 필요하다.

이 논문에서는 메인 메모리의 특성을 이용하여 분할 연기로 인한 삽입 성능의 향상을 위하여 성장 노드를 제안하고, 노드간의 중첩을 줄이기 위한 방법으로 동적 재구성 정책과 사장 공간을 줄이기 위한 방법으로 큰 영역을 가진 노드 분할 정책을 제안한다. 제안 방법은 이동체 데이터베이스 환경에서 빠른 영역 질의를 제공한다.

이 논문의 구성은 다음과 같다. 먼저 2장에서는 관련 연구를 소개하고, 3장에서는 대상 환경과 문제 정의를 설명한다. 4장에서는 제안 색인을 기술하고, 5장에서는 실험을 통하여 논문에서 제시한 방안을 기존의 연구와 비교한다. 마지막으로 6장에서는 결론 및 향후 연구를 기술한다.

2. 관련 연구

기존의 색인에는 크게 메인 메모리 기반의 색인과 디스크 기반의 색인으로 나눌 수 있다. 가장 널리 알려진 메인 메모리 기반의 색인으로는 T-tree[1]가 있고, 디스크 기반의 색인으로는 R-tree[2], R*-tree[3]가 있다.

T-tree[1]는 기존의 B-tree와 AVL-tree로부터 진화되어 나온 색인이다. T-tree는 이진 검색과 높이 균형을 가지는 AVL-tree의 성질을 가지고 있고, 한 노드안에 여러 개의 데이터를 가지는 B-tree의 성질을 가지고 있다. 이러한 성질로 인하여 빠른 처리 속도와 메모리 사용의 최적화라는 메인 메모리의 특성에 적합한 구조로 알려져 있다. 그러나, T-tree는 1차원 데이터에만 적용 가능한 색인으로서 다차원인 이동체 데이터베이스 환경에서는 서론에서 언급했듯이 회전 비용의 급격한 증가로 인하여 색인의 성능이 아주 나빠지게 된다.

R-tree[2]와 R*-tree[3]는 저장되는 데이터 객체를 최소 경계 박스(MBB: Minimum Bounding Box)로 표현하며 B-tree에 대해서 k차원으로 확장한 모델이다. 다차원 데이터에 대하여 높은 성능을 나타내며, 디스크에 적합한 구조로 알려

져 있다. 그러나 R-tree 계열은 디스크의 특성을 고려한 디스크 기반의 색인으로서 디스크 I/O회수를 줄이는 데에 초점이 맞추어져 있으므로, 메인 메모리 DBMS에서는 최적의 성능을 보장할 수 없다.

메인 메모리 기반 색인 구조의 목표는 CPU cycle에 다른 전체 수행 시간을 줄이고, 가능한 한 적은 메모리 공간을 사용하는 것이다[1]. 따라서 R-tree계열에서 이러한 메인 메모리 기반 색인 구조의 목표에 상대적으로 적합한 것은 Linear Split을 사용하는 R-tree이다.

3. 대상 환경 및 문제 정의

3.1 대상 환경

이 논문에서는 이동체가 주기적으로 자신의 위치 정보와 필요 시에 질의 정보를 서버로 전송한다. 서버에서는 클라이언트의 위치 정보를 저장하고, 클라이언트에서 질의 요청이 오면, 저장하고 있는 위치 정보를 이용하여 응답하는 환경을 대상으로 하고 있다. 이 논문에서는 질의 중에서 가장 일반적인 영역 질의(Range Query)의 성능 향상을 목적으로 하고 있다.

3.2 문제 정의

첫째, R-tree를 메인 메모리 기반 색인 구조로 사용할 때, 노드를 분할하는 기준이 메인 메모리의 특성을 고려하지 않고 있다. 디스크 기반 색인의 경우에는 디스크 I/O회수의 최소화를 위해 노드의 크기를 맞추기 위해 분할하지만, 메인 메모리 기반에서는 노드의 크기에 아무런 제약이 없다.

둘째, R-tree는 삽입 순서(insertion order)에 의해 영향을 받기 때문에, 색인의 성능이 나빠지는 결과를 초래할 수 있다. 기존의 R*-tree에서는 재 삽입 전략을 사용하지만, CPU Cycle에 따른 전체 수행 시간을 줄이자는 메인 메모리 구조의 목표에는 R*-tree는 적합하지 않다.

셋째, 노드간의 심한 중첩(overlap)은 색인의 검색 성능에 영향을 준다. 즉 노드 간의 심한 중첩은 색인의 검색 성능을 저하시키는 데 결정적인 요인으로 작용한다.

넷째, 영역의 크기가 같은 레벨에 있는 다른 노드들에 비해서 아주 큰 노드는 검색 성능을 저하시키는 요인이 된다. 아주 큰 크기를 가진 노드들은 많은 사장 공간(dead space)을 유발할 가능성이 크기 때문에 영역 질의 시에 많이 접근됨으로써 성능 저하의 요인이 된다.

4. 이동체 데이터베이스를 위한 메인 메모리 색인

3장에서 언급한 문제를 해결하기 위해 4장에서는 성장 노드 정책, 동적 재구성 정책, 큰 영역을 가진 노드 분할 정책 등을 제안한다.

제안 방법의 설명을 위해서 다음과 같은 용어를 정의한다.

정의 1. EOR(Excessive Overlap Ratio): 노드간의 심한 중첩 비율(Excessive Overlap Ratio)을 나타낸 말로서, 두 개의 노드의 중첩 비율이 EOR 이상일 경우에는 심한 중첩을 나타낸다.

노드 간의 중첩 비율을 30%, 40%, 50%, 60%로 바꾸어 가면서 실험한 결과 EOR이 40%, 즉 0.4일 때 가장 좋은 성능이 나타났다.

정의 2. EAR(Excessive Area Ratio): 노드간의 심한 영역 크기 비율(Excessive Area Ratio)을 나타낸 말로서, 같은 레벨 노드들의 평균 영역 크기와 해당 노드의 영역 크기의 비율이 EAR이상이면 심한 사장 공간을 나타낸다.

같은 레벨 노드들의 평균 크기의 2배, 3배, 4배, 5배, 10배로 바꾸어 가면서 실험한 결과 3배, 즉 EAR이 3일 때, 가장 좋은 성능이 나타났다.

4.1 성장 노드 (Growing Node)

문제 정의에서 언급했듯이, 메인 메모리 기반에서는 노드의 크기에 제약이 없기 때문에, 이 논문에서는 단말(Leaf) 노드 레벨에서 성장 노드(Growing Node)를 제안한다. 성장 노드란 노드의 용량이 가득 찼을 때, 해당 노드가 분할할 이유가 없다면, 분할하지 않고 계속 성장하는 노드를 말한다. 기존 R-tree에서는 노드의 용량이 가득 차면, 분할(split)을 실시하지만, 여기서는 노드간의 중첩이 아주 심한 경우와 노드의 영역의 크기가 아주 큰 경우에만 분할을 한다. 성장을 하는 방법은 그림 1과 같이 새로운 노드를 만들어 기존 노드에 연결을 시키는 방법을 사용하여 성장 노드로 전환하는 것이다. 즉 검색 성능을 저하시키는 노드 간의 중첩 비율이 EOR을 넘지 않고, 영역의 크기가 EAR을 넘지 않으면 노드는 계속 성장(growing)하는 것이다.

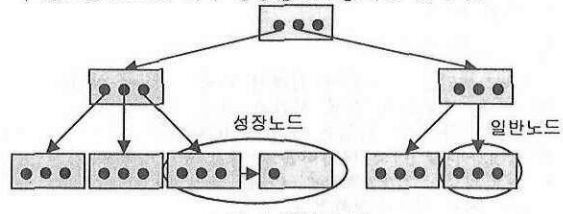

그림 1 성장 노드

비단말 노드에서 성장 노드를 사용하는 경우에는 노드 당 엔트리들의 수가 많아진다. 그러면 삽입 시에, 삽입할 단말 노드를 찾기 위해서 비단말 노드에 있는 엔트리 수를 비교하는 알고리즘인 ChooseSubtree를 수행할 때, 비교하는 엔트리의 수가 많아지기 때문에 삽입 성능이 현저히 저하된다. 따라서 성장 노드는 단말 노드에서만 사용한다. 성장 노드를 사용함으로써, 분할 연기와 비단말(Non-Leaf) 노드에서의 엔트리 수 감소로 인한 삽입 성능의 향상을 가져온다.

성장 노드의 분할은 일반 노드의 분할 알고리즘인 Linear Split과 같은 형태로 행해진다. 일반 노드의 분할이 노드 용량+1의 크기를 가지고 분할 하는 반면에, 성장 노드의 분할은 성장 노드의 크기, 즉 일반 노드보다 많은 수의 엔트리를 가지고 분할한다는 데 그 차이가 있다. 하나의 성장 노드를 분할하고 난 후의 두 개의 그룹의 크기에 따라 일반 노드의 용량 보다 크면 성장 노드, 일반 노드의 용량 보다 작으면 일반 노드로 정해진다.

4.2 동적 재구성 정책

문제 정의에서 언급했듯이, 기존의 R-tree는 노드간의 중첩이 심하기 때문에 이는 검색 성능을 저하시킨다. 따라서 이 논문에서는 노드간의 중첩을 감소시키기 위해서 동적 재구성 정책을 제안한다.

동적 재구성 정책은 단말 노드에서 해당 노드와의 중첩 비율이 EOR을 넘는 형제(sibling) 노드를 합병(merge)한 후에 재분할하는 것이다. 동적 재구성 정책은 일반 노드가 가득 찼을 때와 성장 노드에서 데이터가 삽입될 때 수행된다. 데이터가 삽입될 때마다 노드간의 중첩 비율을 계산하면 삽입 성능이 많이 저하되므로, 데이터 삽입으로 인해 해당 노드의 크기가 커질 경우에만 동적 재구성 정책을 수행한다. 만약 삽입으로 인해 확장된 영역(Enlargement Area)이 0일 때는 중첩 비율을 계산할 필요가 없으므로 동적 재구성이 일어나지 않는다.

동적 재구성 정책은 삽입할 노드와 EOR이상의 중첩 비율을 가진 형제 노드와 수행하게 되는데, 두 노드의 상태에 따라 성장 노드와 성장 노드, 성장 노드와 일반 노드, 일반 노드와 성장 노드, 일반 노드와 일반 노드 사이에서 각각 이루어질 수 있고, 그 결과도 여러 가지로 나올 수 있다.

동적 재구성 정책의 사용은 검색에서 노드간의 중첩을 감소시킴으로써 성능 향상을 가져온다.

4.3 큰 영역을 가진 노드 분할 정책

문제 정의에서 언급했듯이 EAR이상의 값을 가진 노드는 많은 사장 공간을 유발할 가능성이 크다. 이는 검색 성능을 저하시키는 요인이 된다.

이 논문에서는 이러한 큰 영역을 가진 노드를 분할함으로써 사장 공간을 줄이고자 한다. 큰 영역을 가진 노드 분할 정책은 중첩이 EOR이하 일 때, 그리고 영역의 크기가 EAR이상 일 때 수행된다. 분할 후에 노드간의 중첩이 EOR이상 일 경우에는 분할 전으로 돌아간다. 이러한 큰 영역을 가진 노드 분할 정책을 사용함으로써 사장 공간이 줄어 들기 때문에 검색 성능의 향상을 가져 온다.

5. 성능 평가

5.1 실험 요소

제안 색인을 평가하기 위해서 다음과 같은 삽입과 검색 데이터를 사용하여 검색 시간과 검색 시 비교하는 엔트리 수를 측정하였다. 삽입은 50만개(1000*500: 1000개의 이동체가 500번 보고)의 데이터를 가지고 수행하였고 검색은 전체 영역의 5%, 10%, 20%의 크기를 가진 1000개의 영역 질의(Range Query)를 수행하였다.

5.2 실험 파라미터

미리 수행해 본 실험에서의 결과를 토대로 EOR이 0.4이상일 때를 노드간의 중첩이 심하다고 판단하는 기준으로 삼았고, 노드간의 중첩이 심하지 않을 때, EAR이 3이상인 노드를, 큰 영역을 가진 노드라고 판단하는 기준으로 삼았다.

5.3 실험 세부 사항

각 알고리즘 및 색인은 C로 구현하였고, 윈도우 XP pro에서 1GB의 메인 메모리, CPU는 Pentium IV 3.06Ghz를 가지고 실험하였다. 데이터 집합은 GSTD 생성기[4]를 가지고 생성하였다. GSTD 생성기는 시간 간격 마다 이동체 위치인 점 좌표를 생성하기 때문에, 이전 위치를 참조하여 선분을 생성하여 색인에 저장하였다. 이동체 위치 데이터를 생성하기 위한 매개 변수로, 이동체의 초기 분포는 가우시안 분포이며 이동체의 이동은 랜덤이다.

5.4 실험 결과

5.4.1 삽입 성능

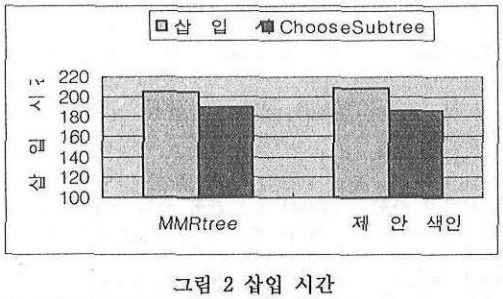

그림 2 삽입 시간

그림 2와 같이 삽입 시간은 기존의 메인 메모리 R-tree와 비슷한 것을 알 수 있다. 전체 삽입 시간에서 단말 노드의 성장 노드 정책과 동적 재구성 정책에 따른 분할 연기로 인하여 삽입 성능의 향상을 가져오지만, 성장 노드 분할과 합병 후 재분할 등으로 인하여 증가된 계산 시간 때문에 전체 삽입 시간은 유지된다.

5.4.2 검색 성능

그림 3과 그림 4는 앞의 실험에서 생성된 색인에 대하여 각각 5%, 10%, 20%의 영역 질의를 수행함으로써 검색 성능을 측정한 것이다.

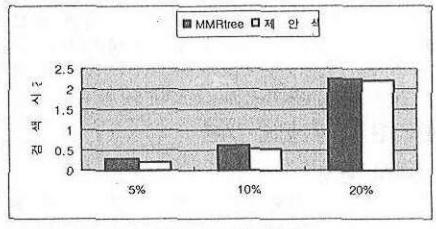

그림 3 검색 시간

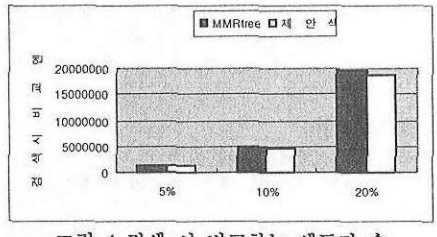

그림 4 검색 시 비교하는 엔트리 수

그림 3, 4에서 보는 바와 같이, 검색 시간과 검색 시 비교하는 엔트리 수가 전체적으로 5%~20% 정도 감소하는 것을 알 수 있다. 실험 결과에서 보듯이, 제안한 동적 재구성 정책으로 인한 노드 간의 중첩 감소와 큰 영역을 가진 노드 분할 정책으로 인한 사장 공간의 감소로 인하여 검색 성능의 향상을 가져 온다.

6. 결론

이 논문에서는 이동체 데이터베이스를 위한 메인 메모리 색인을 제안하였다. 제안된 색인은 메인 메모리에 적재한 기존의 R-tree와 실험을 통해서 비교하였다. 실험에서, 제안된 색인은 기존 색인에 비해 삽입 성능을 유지하는 반면에 영역 질의 시에 검색 성능의 향상을 가져왔다.

실험에서 본 바와 같이 검색 성능을 향상시키기 위해서는 노드간의 중첩을 줄이고, 많은 사장 공간을 유발할 가능성이 높은, 아주 큰 영역을 가진 노드의 크기를 줄이는 방법이 필요하다.

7. 참고 문헌

[1] Tobin J. Lehman, Michael J. Carey "A Study of Index Structures for Main Memory Database Systems", in Proceedings 12th Int'l. conf. on Very Large Databases, Kyoto, Aug. 1986, pp.204~230

[2] A. Guttman, " R-trees: A dynamic index structure for spatial searching" , ACM SIGMOD Conference, pp.47-54, 1984.

[3] N. Beckmann and H. P. Kriegel, "The R*-tree: An Efficient and Robust Access Method for Points and Rectangles", In Proc. ACM SIGMOD, pp.332-331, 1990.

[4] Y. Theodoridis, J. R. O Silva and M.A Nascimento, " On the Generation of Spatiotemporal Datasets" , SSD, Hong Kong, LNCS 1651, Springer, p147-164, 1999.

[5] 전봉기, 홍봉희 " 이동체의 색인을 위한 시간 기반 R-tree의 설계 및 구현" , 한국 정보 과학회 정보과학회논문지: 데이터베이스 제30권 제3호 pp320~335, 2003