ORB : 효율적인 질의 성능을 위한 R-tree 대량로딩 기법

이태원°, 이석호
서울대학교 전기컴퓨터공학부
warrior@db.snu.ac.kr°, shlee@cse.snu.ac.kr

ORB : R-tree Packing for better query performance

Taewon Lee°, Sukho Lee
School of Electric and Computer Engineering, Seoul National University

요 약

R-tree는 공간 데이터나 다차원 데이터의 효율적인 질의 처리를 위한 인덱스 구조이다. 다량의 데이터로부터 빠르게 인덱스를 생성하기 위해서 많은 다량로딩 기법들이 제안되었으나 이들은 공간이용률을 극대화하는 데에 초점을 맞춰 R-tree의 목적인 효율적인 질의 처리를 위한 개선의 여지가 남아 있다. 본 논문에서는 다량로딩 과정에서 인접한 노드들간의 겹치는 영역을 감소시켜 전체적으로 질의 처리 성능을 향상시킬 수 있는 기법을 제안한다. 실험 결과에서 보이듯이 지금까지 가장 효율적이라고 알려져 있는 STR 기법보다 질의 성능이 좋게 나오는 것을 확인할 수 있다.

1. 서론

R-tree는 공간 데이터나 다차원 데이터의 효율적인 질의 처리를 위해 데이터베이스에서 가장 널리 사용되는 인덱싱 기법이다. 대표적인 응용으로는 CAD, GIS등 공간 상의 개체를 관리하는 응용이나 여러 개의 키를 갖는 기존의 전통적인 데이터베이스 등을 들 수 있다. R-tree는 전체 구조를 재구성하지 않고도 데이터를 추가하거나 삭제할 수 있기 때문에 동적인 구조이다. R-tree에 데이터를 추가하거나 삭제하는 것은 기본적으로 한번에 하나씩 이루어진다[1]. 그러나 한 번에 하나씩 데이터를 추가하는 것은 다음과 같은 여러가지 문제점이 있다.

1) 전체 트리를 구성하는 데 걸리는 긴 시간,
2) 인덱스의 최적화되지 않은 공간 활용,
3) 비효율적인 인덱스 구조로 인해 질의 처리를 위해서는 불필요한 노드에 대한 많은 접근 발생

질의 처리의 효율성 측면에서 R-tree 구조를 효율적으로 만드는 것에 대한 여러 알고리즘들이 제안되었다[2, 3]. 이들은 R-tree에 비해서 노드 간의 겹침이 적은 효율적인 구조를 만들어주기 때문에 질의 성능을 많이 개선시켰지만, 이들 또한 데이터를 미리 가공해서 한꺼번에 인덱스를 생성하는 대량로딩(bulk loading) 기법들에 비해서는 비효율적이다.

기존의 대량로딩에 관련된 연구들은 중간 노드의 공간 점유율이 100%에 가깝게 하면서도 각 노드 사이의 겹치는 영역을 최소화하려고 시도해 왔다[4, 5, 6]. 이들 알고리즘은 공간상에서 가까이 위치한 데이터 개체들을 묶어서 리프 노드를 구성하고 다시 리프 노드들을 묶어서 상위 레벨의 노드들을 구성한다. 이렇게 해서 마지막 루트 노드가 생성될 때까지 트리의 밑에서부터 위로 반복적으로 묶어 나가는 bottom-up 방식을 따른다.

그런데 기존의 기법들은 일부 리프 노드를 제외하고는 공간 이용률을 최대로 하려는 특징 때문에 노드 간의 겹치는 영역이 증가하는 문제가 있다. 이는 결과적으로 질의 처리 성능에도 영향을 미치게 된다. R-tree는 궁극적으로 인덱스 내의 데이터에 대한 효율적인 질의 처리를 위해 이용되는 것이므로 공간 이용률보다는 질의 처리 성능이 중요시된다.

본 논문에서는 대량로딩의 질의 처리 성능을 더욱 개선해 빠르게 인덱스 구조를 생성하면서도 기존 어떤 방법보다도 개선된 질의 성능을 보이는 대량 로딩 기법을 제안한다. 이를 위해서 기존의 대량로딩 기법들이 목표로 하는 최대 공간이용률을 포기하고, 대신 전체적인 노드 간의 겹치는 영역을 줄이기 위한 방법을 제안한다. 기존의 Nearest-X 방법과 STR 기법과의 비교를 통해 질의 성능이 더 좋아짐을 보인다. 비교를 위해서 Tiger/Line 2000 데이터 집합과 인위적으로 생성한 데이터 집합을 이용해 실험한다. 실제 데이터베이스에서는 트리의 일부 노드가 버퍼에 저장되고, 이 버퍼로 인해서 성능에 엄청난 영향을 끼치게 된다[7]. 본 논문에서는 버퍼를 이용해 실험을 수행하였다.

본 논문의 구성은 다음과 같다. 2장에서는 기존의 대량로딩 기법들에 대해 간략히 살펴본다. 3장에서는 본 논문에서 제안하는 대량로딩 기법에 대해 자세히 설명한다. 4장에서는 실험결과를 제시하고 마지막으로 5장에서는 결론을 맺는다.

2. 관 련 연 구

이 절에서는 3가지의 대량로딩 기법에 대해 설명한다. 이들은 모두 비슷한 과정을 거쳐 수행된다. 설명을 위해 총 N개의 데이터 개체가 있고, 각 노드에는 최대 M개의 엔트리를 포함할 수 있다고 하자.

공통적으로 대량로딩 기법은 먼저 N개의 데이터 개체를 정렬하여 M개의 데이터를 포함하는 ⌈N/M⌉개의 그룹으로 분할한 후, 각각을 리프 노드로 구성하여 트리의 제일 밑에서부터 인덱스를 생성해 나간다. 리프 노드를 모두 구성하면 상위 레벨에서 다시 리프 노드를 ⌈N/M⌉개의 그룹으로 분할하고 중간 노드를 구성하는 작업을 루트 노드가 생성될 때까지 반복한다.

각 대량로딩 기법들은 데이터를 어떻게 정렬하고 어떻게 분할하는지에 있어서만 다른 접근 방법을 사용한다.

Nearest-X(NX) : 각 사각형을 x좌표에 의해 정렬한다. 그런 후에 x좌표가 작은 것부터 M개씩 묶어서 하나의 노드를 구성한다.

Hilbert Sort(HS) : 각 사각형을 Hilbert 커브를 이용해 정렬한다. 그런 후에 역시 M개씩 묶어서 하나의 노드를 구성한다.

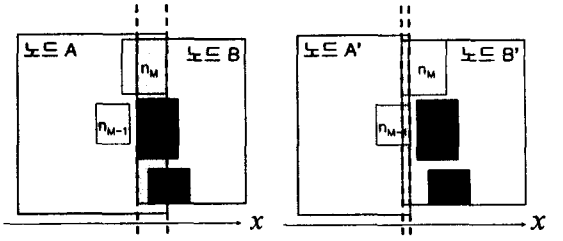
Sort-Tile-Recursive(STR) : x좌표를 이용해 데이터를 정렬한 후 전체 데이터 개체를 연속된 $\sqrt{N/M}$ 개의 구간으로 나눠 각 구간에서 $\sqrt{N/M}$ 개의 리프 노드를 구성할 수 있도록 한다. 각 그룹 안의 데이터 개체를 y좌표를 이용해 정렬한 후, M개씩 묶어서 리프 노드를 구성한다. 이 방식의 특징은 전 영역을 타일처럼 조각을 내어 겹치는 영역을 최소화하려고 시도한다는 점이다. 실제로 노드 수는 같지만 겹치는 영역이 많이 줄어들어 질의 처리 성능이 기존의 NX나 HS 기법에 비해 우수하다.

그러나 기존의 대량로딩 기법들은 모두 노드 내의 엔트리 비율을 최대화하는 것을 기본으로 하고 이 상태에서 겹치는 영역을 최소화하려고 시도해 왔다. 리프 노드가 항상 M개씩의 엔트리를 포함하도록 하기 위해서 정렬된 사각형을 M개씩 묶어 나가기 때문에 리프 노드의 MBR을 조절해 겹치는 영역을 줄일 수가 없다.

본 논문에서는 이런 문제점에 착안, R-tree의 공간 이용률을 약간 낮추면서 겹치는 영역을 좀 더 줄여나가는 방식을 통해 질의 성능을 향상시키는 방법을 제안한다. 공간 이용률이 낮아짐으로 해서 리프 노드 수가 조금 증가하지만 전체적으로 노드 간의 겹치는 영역을 줄여서 성능 향상을 가져올 수 있다.

3. ORB (Overlap-Reduced Bulkloading) 기법

이미 어떤 기준에 의해 정렬되어 있는 사각형들을 M개씩 묶어서 리프 노드를 구성하게 될 경우, 해당 리프 노드의 MBR은 조정이 불가능하다. 이는 M개의 사각형을 포함하는 최소의 사각형이 해당 노드의 MBR이기 때문이다. 이 절에서는 대량로딩 방식으로 한 번에 인덱스를 구성하되 그 과정에서 큰 비용을 들이지 않고 겹치는 영역의 넓이를 감소시켜 질의 성능을 높이는 방법을 제안한다.



(a) (b)

[그림 1] 겹치는 영역을 줄이는 노드 구성

3.1 개요

[그림 1]에서 각 엔트리를 x좌표를 기준으로 정렬했을 때 차례로 n_1, n_2, $\cdots n_M$, n_{M+1}, $\cdots$ 이라 하자. 첫번째 리프 노드를 노드 A라 하면 [그림 1] (a)에서는 노드 A에 M개의 사각형을 포함해야 하는 조건 때문에 노드 A는 $n_1 \sim n_M$까지를 포함하는 MBR을 가지게 된다. 노드 B는 n_{M+1}부터 n_{2M}까지를 포함하게 될 것이다. 이 때 n_M의 MBR의 $Upper_x$값(사각형의 MBR을 표현하는 두 점의 x좌표 중 큰 값)과 n_{M+1}의 MBR의 $Lower_x$값(사각형의 MBR을 표현하는 두 점의 x좌표 중 작은 값)에 의해 겹쳐지는 영역이 결정된다. R-tree의 노드는 최대 M개의 엔트리를 가질 수 있기 때문에 기존 방식과 달리 공간이용률을 100%로 제한하지 않고, 겹치는 영역이 최소가 되는 경계점을 찾아 노드를 구성한다. 즉, [그림 1] (b)와 같이 노드 A'는 M보다 적은 개수의 엔트리를 포함하도록 허용함으로써 노드 A' 와 노드 B' 의 겹치는 영역을 크게 줄일 수 있다. 이렇게 리프 노드를 구성하는데 드는 비용은 기존 대량로딩 기법과 비교했을 때 CPU 상에서의 O(N) (N은 전체 데이터의 수)의 추가 비교 연산 정도가 되므로 인덱스 구성에 드는 비용은 크게 증가하지 않는다.

3.2 ORB 기법에 대한 상세 알고리즘

이 기법은 고차원의 R-tree에 대해서도 확장이 가능하지만, 설명을 위해 여기서는 2차원 R-tree를 기준으로 한다.

먼저 사각형들을 x좌표를 따라 정렬한 뒤, x축을 따라 구간을 나누는 과정이다. 이 과정에서 각 구간 사이의 겹치는 영역의 넓이가 최소가 되도록 작업을 한다. 이렇게 나눠진 구간 내의 사각형들을 다시 y좌표를 기준으로 정렬한 후, 리프 노드를 구성해 나간다. 이 때도 인접한 리프 노드 간의 겹치는 영역의 넓이가 최소가 되도록 하는 분할을 찾는다. 단, 요구되는 공간이용률이 있는 경우, 해당 공간이용률 범위 내에서 겹치는 넓이가 최소가 되도록 한다.

전체 사각형의 수를 N, 한 노드가 최대로 포함할 수 있는 사각형의 수를 M이라 하고 공간이용률이 100%가 되도록

대량로딩을 하는 경우, 전체 리프 노드의 수는 대략 ⌈N/M⌉개가 된다. 이를 균등하게 $\lceil\sqrt{\lceil N/M\rceil}\rceil$ 개의 구간으로 나누면 각 구간에는 약 $\lceil\sqrt{\lceil N/M\rceil}\rceil$개의 리프 노드가 포함되게 된다. 그러면 각 구간에 포함된 사각형의 개수는 마지막 구간을 제외하고 대략 S=$\lceil\sqrt{\lceil N/M\rceil}\rceil$ * M개가 된다. S는 100% 공간이용률을 가정했을 때의 구간 내 사각형의 개수가 된다. 효율적으로 겹치는 영역을 줄이기 위해 구간 내에 포함될 사각형의 수를 일정한 범위로 한정지어 해당 범위 내에서의 최적의 분할을 찾는다. 범위는 p라는 파라미터로 정의할 수 있는데, (1-p)*S~(1+p)*S로 구간 내의 사각형의 수의 범위를 한정한다. 여기서 p를 작게 할수록 최적의 분할을 찾기는 어렵지만 대량로딩 시간이 단축되고, 그 값을 크게 하면 좀 더 최적의 분할을 찾을 수 있지만, 구간이 많아져 전체적으로 노드 수가 증가해 질의 처리 성능을 상대적으로 저하시킬 수 있다. 구체적으로는 다음과 같이 수행한다.

먼저 사각형을 x좌표를 기준으로 정렬한다. 이 때 $Lower_x$값을 기준으로 정렬한다. 구간 내에는 최소한 (1-p)*S개의 사각형이 존재해야 하므로(마지막 구간은 예외) 처음 (1-p)*S개의 사각형을 첫번째 구간에 포함시킨다. 그리고 이들 사각형의 MBR중 가장 큰 Max_x= $Upper_x$값을 기록한다. 이제 Max_x를 좀 더 정확히 정의하자. 두번째 구간에 포함되는 제일 작은 $Lower_x$값을 갖는 사각형을 n_l ((1-p)*S+1≤ l≤ (1+p)*S)라 하면, Max_x는 $Max(Upper_x(n_u))$ ((1-p)*S≤ u≤ l-1)라 할 수 있다. 결국 Max_x- $Lower_x(n_l)$를 최소로 하는 l을 찾음으로써 두 구간의 겹치는 영역을 최소화할 수 있다. 이렇게 해서 모든 후보 사각형에 대해 겹치는 영역을 가장 적게 만드는 l을 찾았으면 (1-p)*S +1~ l 까지를 첫번째 구간에 포함시켜 첫번째 구간을 결정할 수 있다.

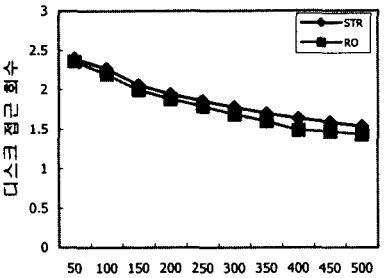
그리고 나서 다시 l +1번째 사각형부터 시작해 위 과정을 반복하면서 구간을 나눠가면 된다. 이 때 남아있는 사각형의 개수가 (1+p)*S+m(한 노드에 들어가는 최소의 엔트리 수; 보통 M/2)보다 많으면 반복을 하고, 그렇지 않으면 나머지를 하나의 구간으로 만들면 된다.

이렇게 구간이 결정되면 각 구간 내의 사각형에 대해 y좌표를 기준으로 하여 정렬을 수행한다. 이 때는 $Lower_y$ 값을 기준으로 정렬을 한다. 구간 내의 사각형들을 최대 M개씩 묶어서 리프 노드를 구성하게 된다. 한 노드에는 루트 노드가 아닌 이상 최소한 m개의 엔트리를 포함해야 하므로 구간을 나눌 때 (1-p)*S~(1+p)*S개의 사각형을 포함하도록 했던 것에서 m~M개의 사각형을 포함하는 것만 달라지고 구간을 나누는 과정과 동일하게 리프 노드를 구성해 나가게 된다. 리프 노드를 생성해 나갈 때 남은 사각형의 개수가 M보다 적어지면 더 이상 나누지 않고 하나의 노드를 구성한다.

고차원으로의 확장은 지면 관계상 생략하나 간단하게 확장이 가능하다.

4. 실험

실험에 사용한 데이터 집합은 Tiger/Line과 가상으로 생성된 zipf 분포를 따르는 데이터 집합을 사용했다. 실험에서 비교 대상으로 삼은 대량로딩 기법은 Nearest-X와 Hilbert 그리고 STR 기법이다. 그러나 STR을 제외한 다른 기법들은 결과의 차이가 너무 커 STR과의 정확한 비교가 되지 않아 [그림 2]에 나타내지 않았다.



[그림 2] 버퍼 크기에 따른 질의 처리 성능 비교

그림에서 볼 수 있듯이 전반적으로 꾸준히 질의 처리시 비용이 적게 드는 것을 확인할 수 있다. 이 결과는 2차원 데이터인 경우이고 고차원으로 갈수록 겹치는 영역이 전체 영역에 비해 차지하는 비율이 커져서 본 논문에서 제시한 방법에 의하면 더 효율적인 질의 처리가 가능해진다.

5. 결론

본 논문에서는 기존 대량로딩 기법들이 공간이용률을 극대화하는데 치중해 질의 처리 성능을 더 개선할 수 있는 부분을 간과한 점에 착안, 전체적으로 겹치는 영역을 더욱 줄일 수 있는 대량로딩 기법을 제안하였다.

참 고 문 헌

[1] Antonin Gutmann, "*R-trees: A dynamic index structure for spatial searching*," ACM SIGMOD, pp.47~57, 1984.

[2] Ibrahim Kamel and Christos Faloutsos, "*Hilbert R-tree: An Improved R-tree using Fractals*," VLDB, pp.500~509, 1994.

[3] N. Beckmann, H.-P. Kriegel, R. Schneider and B. Seeger, "*The R*-tree: an efficient and robust access method for points and rectangles*," ACM SIGMOD, pp.322-331, 1990.

[4] N. Roussopoulos and D. Leifker, "*Direct Spatial Search on Pictorial Databases Using Packed R-Trees*," ACM SIGMOD, pp.17~31, 1985.

[5] I. Kamel and C. Faloutsos, "*On Packing R-trees*," CIKM, pp.490-499, 1993.

[6] S. T. Leutenegger, J. M. Edgington and M. A. Lopez, "*STR: A Simple and Efficient Algorithm for R-tree Packing*," ICDE, pp.497~506, 1997.

[7] S. T. Leutenegger and M. A. Lopez, "*The Effect of Buffering on the Performance of R-Trees*," ICDE, pp.164~171, 1998.