계층 최대 R-트리를 이용한 범위 상위-k 질의의 효율적 수행[1)]

홍석진°, 이상준, 이석호
서울대학교 공과대학 전기컴퓨터공학부
{jinny°, freude}@db.snu.ac.kr, shlee@cse.snu.ac.kr

Efficient Execution of Range Top-k Queries in a Hierarchical Max R-Tree

Seokjin Hong°, Sangjun Lee, Sukho Lee
School of Electrical Engineering and Computer Science

요 약

범위 상위-k 질의는 질의 범위 내의 다차원 데이타 중 값 애트리뷰트를 기준으로 상위 k개의 레코드를 반환하는 질의로 공간 데이타베이스와 데이타 웨어하우스에서 분석을 위해 많이 사용되는 유용한 질의 형태이다. 이 논문에서는 계층 최대 R-트리의 선택적인 탐색을 통해 범위 상위-k 질의를 효과적으로 수행하는 기법을 제시한다. 질의 결과를 포함하지 않는 단말 노드를 접근하지 않고 질의를 수행할 수 있기 때문에 적은 노드 접근으로도 질의 수행이 가능하며, 질의 범위의 크기에 관계없이 거의 일정한 성능을 보인다. 두 개의 우선순위 큐를 사용하며 질의 수행 공간을 최소화 하며, *In Sorted Node* 기법을 통해 기존 R-트리와 같은 팬아웃을 보장할 수 있다.

1. 서론

범위 질의란 다차원 데이타 중 특정 범위 내에 있는 데이타를 반환하는 것으로, 공간 데이타베이스[1]에서 많이 사용되는 질의의 형태이다. 범위 질의는 범위 내의 모든 데이타를 결과로 반환하게 되는데 데이타의 양이 많거나 질의 범위의 크기가 큰 경우, 질의 결과의 크기가 매우 커지고 질의 수행 시간도 오래 걸리게 된다. 일반적으로 사람들은 오랜 시간이 걸려 전체 결과를 얻는 것 보다는, 짧은 시간 내에 상위 일부분을 얻기 원하는 경우가 많다. 이렇게 범위 질의의 결과 중 데이타의 값을 기준으로 상위 k개를 반환하는 질의를 범위 상위-k 질의라고 한다.

이러한 범위 상위-k 질의는 다양한 형태로 적용될 수 있다. 가장 간단한 예로, 각 상점에 대한 정보로 상점의 위치와 상점의 매출량을 저장하고 있는 공간 데이타베이스에 대해 "특정 지역의 상점 중 매출액 기준 상위 100개의 상점을 구하라"와 같은 질의가 이에 속한다. 또한 각 지역별 온도와 습도 분포를 측정한 데이타나 시내 각 지점의 교통량을 측정한 데이타에 대해서도 비슷한 형태의 질의가 가능하다. 범위 상위-k 질의는 데이타 웨어하우스[2]에도 적용될 수 있다. YEAR, STATE, CUSTOMER_AGE를 차원 애트리뷰트, SALES_AMOUNT를 값 애트리뷰트로 하는 사실 테이블이 있다고 하면, "2000년에서 2002년 사이에 모든 주에서 발생한 20대의 구매 기록 중 가장 매출이 높은 50개의 (YEAR, STATE, CUSTOMER_AGE)와 해당 SALES_AMOUNT를 구하라"와 같은 질의가 그 예가 될 수 있다.

범위 상위-k 질의를 수행하는 일반적인 방법은 범위 질의 알고리즘을 통해 질의 범위 내의 모든 데이타를 검색하여 값에 의해 정렬한 후, 그 중 상위 k개를 반환하는 방법이다. 질의 범위가 상대적으로 작은 경우에는 범위 질의를 통해 범위 내의 모든 데이타를 접근하면 되지만, 범위가 큰 경우에는 질의 결과의 크기에 비해 질의 범위 내의 데이타의 개수가 너무 많기 때문에, 범위 내의 모든 데이타를 접근하는 것은 비효율적이다. 따라서 가능한 상위 k개 결과에 포함되지 않는 데이타에 대한 접근 없이 질의를 수행하는 방법이 필요하다.

[7]에서는 다차원 배열 형태의 데이타 큐브[3]를 기반으로 범위 상위-k 질의를 수행하는 기법을 제안하였다. 하지만 이 기법은 배열을 기반으로 하기 때문에 밀도가 낮은 데이타를 저장하는데 많은 비용이 필요하며, 각 차원의 도메인이 비연속적이어야 하는 제약조건이 있다.

이 논문에서는 계층 최대 R-트리(Hierarchical Max R-Tree, HMR-트리)를 통해 다차원 데이타를 저장하고 범위 상위-k 질의를 효율적으로 수행하는 기법을 제시한다. HMR-트리는 R-트리[4]의 각 중간 노드마다 하위 노드의 최대값을 저장하는 구조이다. 질의 수행시 HMR-트리의 각 중간 노드에 저장된 최대값을 통해 질의 범위 내의 단말 노드 중 가장 큰 최대값을 포함한 단말 노드부터 우선적으로 접근하게 되므로 데이타의 양이 많아지고 질의 범위가 커지더라도 질의 범위 내의 모든 단말 노드를 접근하지 않고 질의를 수행할 수 있다, 또한 점증적으로 질의 결과를 반환할 수 있으므로 파이프라인 형태의 질의 수행이 가능하다. 질의 수행의 공간 오버헤드를 줄이기 위해 두 개의 우선순위 큐를 사용하여 질의를 수행하며, *In Sorted Node*기법을 통해 기존 R-트리와 같은 팬아웃을 보장한다.

이 논문의 구성은 다음과 같다. 2절에서 범위 상위-k 질의를 수행하기 위한 자료구조인 계층 최대 R-트리와 알고리즘을 소개한 후, 3절에서 질의 수행 공간과 노드의 효율적인 구성 기법에 대해 살펴본다. 4절에서 실험 결과를 분석하고, 5절에서 결론을 맺는다.

2. 계층 최대 R-트리를 이용한 범위 상위-k 질의의 처리

2.1. 자료 구조

범위 상위-k 질의의 대상이 되는 데이타 D는 (*location, value, pointer*)의 형태의 다차원 점 데이타로 구성된다. *location*은 다차원 데이타의 위치를 나타내고, *value*는 순위의 기준이 되는 다차원 데이타의 값을 나타낸다. *pointer*는 다차원 레코드의 부가 정보를 카리키는 포인터 값으로 다차원 데이타가 *value*만을 유지하는 경우에는 생략이 가능하다. 모든 점 데이타는 $location \in R_{space}$, $value \in V$를 만족한다. R_{space}는 점 데이타를 이루는 도메인으로, $R_{space} \subseteq R^d$를 만족하는 d차원 공간의 일부이다. V는 *value*를 구성하는 도메인이다. 사용자의 질의는 (R_Q, k)로 구성된다. R_Q는 질의의 대상이 되는 영역을 나타내며, $R_Q \subseteq R_{space}$, $k \in \{i \mid i=1, 2, 3, ...\}$의 조건을 만족한다. R_Q 내에 포함된 데이타 중 *value*가 큰 순서대로 상위 k

1) 이 논문은 2003년도 두뇌한국21사업에 의하여 지원되었음

개가 결과로 반환된다.

이 논문에서는 범위 상위-k 질의를 효율적으로 수행하기 위한 자료구조로 계층 최대 R-트리(Hierarchical Max R-Tree, HMR-트리)를 사용한다. HMR-트리는 R-트리의 중간 노드마다 하위 노드들에 대한 최대값을 저장한 자료구조이다. HMR-트리는 최대값을 어디에 저장하는가에 따라 여러 기법으로 구성할 수 있다. *In Entry* 기법은 노드의 각 엔트리마다 최대값을 저장하며, *In Node* 기법은 노드마다 최대값을 저장한다. *In Sorted Node* 기법은 *In Node* 기법으로 구성된 HMR-트리의 각 자식 노드들을 최대값 순으로 정렬하는 기법이다. 2절에서는 *In Entry* 기법을 중심으로 범위 상위-k 질의 처리 알고리즘을 설명하고, *In Node, In Sorted Node* 기법에 대해서는 3절에서 설명하기로 하겠다.

그림 1은 *In Entry*기법으로 구성된 2차원 HMR-트리의 예이다. 엔트리 E_{11}은 자식 노드 N_{11}의 최대값 79를 E_{12}는 자식 노드 N_{12}의 최대값 85를 저장하며, E_1은 자식 노드 N_1의 엔트리 E_{11}, E_{12} 에 대한 최대값인 85를 저장한다.

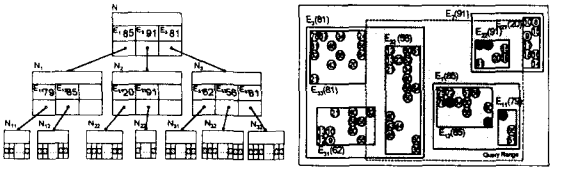


그림 1. 2차원 HMR-트리의 예

2.2 범위 상위-k 질의 알고리즘

이 절에서는 HMR-트리를 기반으로 효율적으로 범위 상위-k 질의를 수행하는 알고리즘을 제시한다. 알고리즘은 우선순위 큐를 통해서 수행된다. 3.1절에서 질의 수행 공간의 효율적인 사용을 위해 두 개의 큐를 사용하는 기법을 설명하기로 하고, 이 절에서는 하나의 큐를 사용하여 질의를 수행하는 기법을 설명하도록 하겠다.

노드의 각 엔트리가 우선순위 큐의 아이템으로 사용된다. 아이템은 (*entry, value*)로 구성되며, 아이템의 *value*가 클수록 큐에서 높은 우선순위를 갖게된다. 레벨이 1 이상인 중간 노드의 경우에는 엔트리에 저장된 최대값이 아이템의 *value*로 사용되며, 레벨이 0인 단말 노드에서는 실제 데이타의 *value*를 아이템의 *value*로 사용하게 된다. 알고리즘은 크게 *initialize*과 *drill down*의 두 과정으로 구성된다.

initialize 과정에서는 우선순위 큐를 초기화한다. HMR-트리의 가장 상위 레벨 노드를 접근하여 질의 범위에 포함되거나 질의 범위에 겹치는 엔트리를 우선순위 큐에 삽입한다.

두 번째 *drill down* 과정에서는 우선순위 큐에서 가장 *value*가 큰 엔트리를 하나 뽑아 해당되는 자식 노드의 엔트리들로 확장해 나가는 식으로 질의를 수행하게 된다. *drill down* 과정은 다음과 같이 구성된다.

- 우선순위 큐에서 아이템 하나를 뽑는다.
- 해당 아이템의 레벨이 0이면 결과로 출력한다.
- 레벨이 1이상이면, 해당하는 하위 노드의 모든 엔트리를 읽어서 질의 범위에 포함되거나 겹치는 엔트리들을 우선순위 큐에 삽입한다.
- 위의 과정을 상위 k개의 결과가 나올 때까지 반복한다.

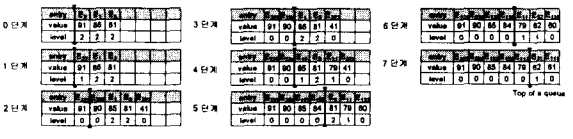


그림 2. 범위 상위-k 질의 알고리즘의 수행과정

그림 2는 그림 1의 HMR-트리에서 주어진 질의 범위 내의 데이타 중 상위 5개의 데이타를 찾는 범위 상위-k 질의의 수행 예를 보여주는 그림이다. 각 단계는 질의 수행 과정을 나타내며, 단계별 우선순위 큐의 내부를 보여주고 있다. 0 단계는 *initialize* 과정을 마친 후의 큐의 그림이며, 1~7단계는 *drill down* 과정을 나타낸다.

3. 질의 수행 공간과 노드의 효율적인 구성

3.1 우선순위 큐 관리

질의가 수행됨에 따라 우선순위 큐에 삽입되는 엔트리의 수는 계속 증가하게 되지만, 그 중 상위 일부분만이 상위 k개의 결과를 얻기 위해 실제로 사용된다. 따라서 큐의 상위 일부분만을 유지하고도 질의 수행이 가능하다면 많은 공간을 사용하지 않고 질의를 수행할 수 있을 것이다.

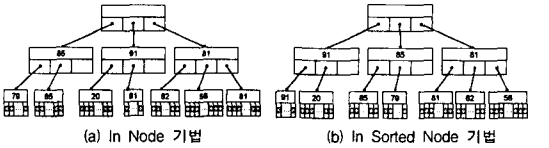
만일 모든 엔트리가 질의 범위에 완전히 포함된다면 우선순위 큐의 상위 k개의 엔트리에는 적어도 k개 이상의 결과가 포함되어 있다는 것을 보장할 수 있다. 큐에서 하나의 엔트리를 뽑는 경우를 생각해 보자. 만일 해당 엔트리가 단말 노드의 엔트리일 경우, 자식 노드가 없으므로 엔트리의 수는 하나 줄어들지만 해당 엔트리가 결과로 반환되므로 구해야 할 결과의 개수도 하나 줄어들어 결과의 손실은 일어나지 않는다. 만일 뽑힌 엔트리가 중간 노드의 엔트리라면, 뽑힌 엔트리의 최대값을 결정한 엔트리를 비롯한 자식 노드의 모든 엔트리는 질의 범위에 완전히 포함되므로 모두 큐에 다시 삽입되며, 이 경우 결과의 손실은 일어나지 않는다. 하지만 질의 범위에 일부만 겹치는 엔트리의 경우, 해당 엔트리의 최대값을 결정한 자식 노드의 엔트리가 질의 범위 바깥에 있다면, 뽑아낸 최대값이 다시 큐에 삽입되지 않으며, 최악의 경우 하나의 엔트리도 다시 큐에 삽입되지 않는 상황이 발생할 수 있다. 이 경우 상위 k개를 다 찾지 못하거나, 중간에 값을 빠뜨릴 수도 있다.

따라서 질의 범위에 완전히 포함되는 엔트리와 질의 범위에 일부만 겹치는 엔트리를 위한 우선순위 큐를 각각 따로 유지한다면, 질의 범위에 완전히 포함되는 엔트리를 위한 큐는 그 크기를 k로 제한할 수 있을 것이다. 질의 범위에 겹치는 엔트리 중, 질의 범위에 일부만 겹치는 엔트리들은 질의 범위의 가장자리에 있는 노드뿐이며, 대부분의 엔트리가 질의 범위 내에 완전히 포함되므로 질의 수행 공간을 크게 줄일 수 있다.

3.2. 노드의 효율적인 구성

In Entry 기법에서는 MRA-tree[5], aggregate cubetree[6]처럼 자식 노드의 최대값을 부모 노드의 엔트리에 저장한다. 이 방법은 자식 노드를 접근하지 않고도 자식 노드의 최대값을 알 수 있는 장점이 있는 반면 엔트리의 크기가 커져 노드의 팬아웃이 줄어드는 단점이 있다. 노드의 팬아웃 감소는 트리를 구성하는데 필요한 노드의 개수를 증가시킬 뿐 아니라, 노드의 증가로 인해 트리의 높이가 높아질 수 있는 문제를 야기시킨다.

만일 집계값을 부모 노드의 엔트리가 아닌 노드 자신에 저장한다면 일반 R-트리와 같은 팬아웃을 보장할 수 있을 것이다. 이를 *In Node* 기법이라 한다. 그림 3(a)는 그림 1의 HMR-트리를 *In Node* 기법으로 구성한 HMR-트리이다.



(a) In Node 기법 (b) In Sorted Node 기법

그림 3. In Node 기법과 In Sorted Node 기법으로 구성한 HMR-트리

In Entry 기법에서 각 엔트리마다 저장하던 자식 노드의 최대값을 *In Node* 기법에서는 자식 노드 자신에 저장하고 있다. *In Node* 기법은 최대값을 엔트리마다 저장하지 않고 노드당 하나만 저장함으로써 엔트리의 크기를 감소시켜 기존 R-트리와 거의 비슷한 팬아웃을 보장할 수 있다.

In Node 기법으로 구성된 HMR-트리에서도 *In Entry* 기법과 비슷한 방법으로 범위 상위-k 질의 알고리즘이 수행된다. 하지만 자식 노드의 최대값이 부모 노드의 엔트리에 없기 때문에 각 엔트리마다 자식 노드의 최대값을 읽기 위해서는 직접 자식 노드를 방

문해서 최대값을 읽어내야 한다. 따라서 이로 인해 노드 접근 횟수가 증가한다는 단점이 있다.

In Sorted Node 기법에서는 노드 내의 엔트리를 해당 자식 노드의 최대값에 따라 정렬하여 저장한다. *In Node* 기법에서는 노드 내의 모든 엔트리에 대한 자식 노드를 전부 방문해야 하는데 비해, *In Sorted Node* 기법에서는 각 엔트리가 최대값 순으로 정렬되어 있으므로, 각 엔트리가 가리키는 자식 노드를 방문하다가 어느 값 이하의 최대값을 갖고 있는 자식 노드는 방문하지 않아도 되므로 *In Entry*와 같은 노드 접근 감소 효과를 가져올 수 있다. 그림 3(b)는 *In Sorted Node*로 구성한 HMR-트리이다.

4. 실험 및 분석

HMR-트리의 성능을 측정하기 위해 HMR-트리의 여러 기법과 기존 R-트리에 대해 범위 상위-k 질의를 수행하고 노드 접근 횟수를 측정하였다. 실험은 펜티엄III 1GHz, 512MB의 주기억장치를 갖는 Linux 기계에서 2~5 차원의 데이타를 임의로 생성하여 수행하였으며, 두 개의 우선순위 큐를 사용하였다.

그림 4는 질의 범위의 크기를 변화시켜가면서 노드 접근 횟수를 측정한 결과이다. 2차원 데이타 300만개를 HMR-트리의 여러 기법과 R-트리로 구성하여 상위 100개를 검색하는 질의를 수행하였다. 기존 R-트리의 경우 질의 범위내의 모든 노드를 접근하여 질의 범위 내에 속하는 모든 데이타를 읽은 후 그 중 상위 k개를 선택해야 하는 반면, HMR-트리를 사용하면 값이 큰 노드부터 차례로 읽어낼 수 있으므로 질의 범위의 크기와 상관없이 일정한 노드 접근을 보인다.

HMR-트리는 각 기법에 따라 노드 접근 횟수의 차이가 있다. 여러 기법 중에서 *In Entry* 기법과 *In Node Sorted* 기법이 거의 비슷하게 좋은 성능을 보이며, *In Node* 기법은 두 기법보다는 전반적으로 많은 노드 접근 횟수를 보인다. 이는 *In Node* 기법의 경우 자식 노드의 최대값을 얻기 위해 자식 노드를 방문하여 야 하기 때문이다. *In Node Sorted* 기법의 경우에는 노드 내의 엔트리가 max 값에 의해 정렬되어 있기 때문에 노드 내의 엔트리가 가리키는 자식 노드를 모두 방문하지 않아도 되므로 *In Entry* 기법과 비슷한 성능을 보이게 된다.

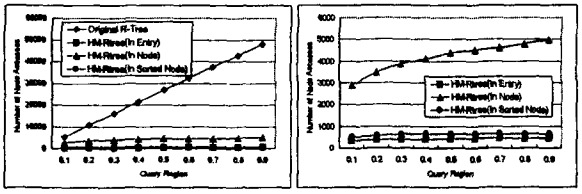


그림 4. 질의 범위의 크기 변화에 따른 노드 접근회수

그림 5는 각 HMR-트리 기법과 기존 R-트리의 팬아웃과 높이를 비교한 결과이다. 2차원 트리에 대해 노드 크기를 1024 byte로 고정하였을 경우 R-트리와 HMR-트리의 *In Node* 기법, *In Node Sorted* 기법은 노드 내 각 엔트리의 크기가 20 byte인 반면, *In Entry* 기법은 엔트리마다 최대값을 저장해야 하므로 24 byte 만큼의 공간이 필요하다. 따라서 기존 R-트리와 HMR-트리의 다른 기법들이 노드 내 엔트리의 수가 50개인 반면 *In Entry* 기법은 42개로 줄어들며, 이로 인해 예를 들어 300만개의 데이타를 저장할 경우 기존 R-트리와 HMR-트리의 다른 기법의 높이가 4인 반면, *In Entry*기법은 높이가 5로 늘어나게 된다.

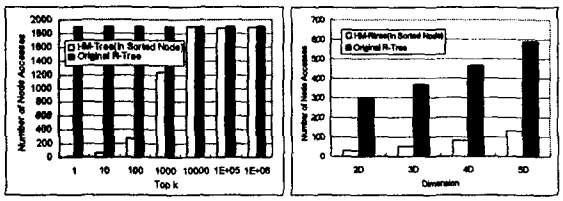
Original R-Tree	1024	20	50	4
HM-Rtree(In Entry), MRA-tree	1024	24	42	5
HM-Rtree(In Node, In Sorted Node)	1024	20	50	4

그림 5. R-트리와 HMR-트리의 팬아웃 및 높이 비교

따라서 *In Sorted Node* 기법으로 구성한 HMR-트리는 기존 R-트리와 비슷한 팬아웃을 보이는 *In Node* 기법의 장점과, 더 적은 수의 노드 접근으로 질의 수행이 가능한 *In Entry* 기법의 장점을 절충한 기법이라 할 수 있다.

그림 6(a)는 HMR-트리에서 질의 결과 크기인 k를 변화해가며 질의를 수행한 결과이다. *In Sorted Node* 기법의 HMR-트리를 사용했으며, 100만개의 2차원 데이타에 대해 질의 범위를 10%로 하여 질의를 수행하였다. 역시 기존 R-트리가 k값에 상관없이 항상 큰 노드 접근 횟수를 보이는 반면, HMR-트리는 k값이 작은 경우는 적은 수의 노드 접근만으로 질의 수행이 가능하며 k값이 커짐에 따라 노드 접근 횟수가 증가하게 된다.

그림 6(b)는 HMR-트리에서 차원을 변화시켜가면서 질의를 수행한 결과이다. 10만개의 데이타에 대해 질의 범위를 30%로 하여 상위 100개를 구하는 질의를 수행하였다. 차원을 증가시켜도 기존 R-트리에 비해 항상 적은 수의 노드 접근만으로 질의 수행이 가능한 것을 알 수 있다.



(a) k의 변화 (b) 차원의 변화

그림 6. k와 차원의 변화에 따른 노드접근 회수

5. 결론

이 논문에서는 다차원 데이타를 계층 최대 R-트리로 구성하여 범위 상위-k 질의를 효율적으로 처리하는 기법을 제시하였다. HMR-트리는 기존 R-트리의 각 중간 노드마다 하위 노드에 대한 최대값을 저장하는 구조이며, 범위 상위-k 질의 알고리즘에서는 질의 범위 내의 모든 단말 노드를 접근하지 않고, 우선순위 큐를 통해 큰 값을 갖는 데이타를 포함한 단말 노드부터 순서대로 접근하여 질의를 처리한다. 질의 수행 공간을 줄이기 위해 두 개의 우선순위 큐를 사용하였으며, *In Sorted Node* 기법을 통해 기존 R-트리와 같은 팬아웃을 보장할 수 있다.

실험에 따르면 제안한 기법이 기존 R-트리를 사용하여 범위 상위-k 질의를 수행하는 경우에 비해 항상 좋은 성능을 보였으며, 질의 범위의 크기에 관계없이 노드 접근 횟수가 거의 일정하였다. k 값이 작은 경우에는 기존 R-트리에 비해 15%정도의 노드 접근만으로 질의를 처리할 수 있었으며, k 값이 큰 경우에도 R-트리와 비슷한 성능을 유지하였다.

참고문헌

[1] Ralf Hartmut Guting, "An Introduction to Spatial Database Systems.", VLDB Journal 3(4), p.357-399, 1994.

[2] Surajit Chaudhuri, Umeshwar Dayal, "An Overview of Data Warehousing and OLAP Technology.", SIGMOD Record 26(1), p.65-74, 1997.

[3] Jim Gray, Adam Bosworth, Andrew Layman, Hamid Pirahesh, "Data Cube: A Relational Aggregation Operator Generalizing Group-By, Cross-Tab, and Sub-Total.", ICDE 1996, p.152-159

[4] Antonin Guttman, "R-Trees: A Dynamic Index Structure for Spatial Searching.", SIGMOD Conference 1984, p.47-57

[5] Iosif Lazaridis, Sharad Mehrotra, "Progressive Approximate Aggregate Queries with a Multi-Resolution Tree Structure.", SIGMOD Conference 2001

[6] Seokjin Hong, Byoungho Song, Sukho Lee, "Efficient Execution of Range-Aggregate Queries in Data Warehouse Environments.", ER 2001, p.299-310

[7] Zheng Xuan Loh, Tok Wang Ling, Chuan-Heng Ang, Sin Yeung Lee, "Adaptive Method for Range Top-k Queries in OLAP Data Cubes.", DEXA 2002, p.648-657