이동 객체 위치 색인을 위한 R-트리 갱신 기법

권동섭° 이상준 이석호
전기.컴퓨터공학부,
서울대학교
{subby, freude}@db.snu.ac.kr°, shlee@db.snu.ac.kr

R-tree Update Technique for Indexing the Positions of Moving Objects

Dongseop Kwon° Sangjun Lee Sukho Lee
School of Electrical Engineering and Computer Science,
Seoul National University

요 약

최근에 이동 객체의 위치를 추적하는 기술은 여러 응용 분야에서 중요성이 증대되고 있다. 그러나 지속적으로 움직이는 이동 객체의 위치를 추적하기 위해서는 매우 많은 수의 인덱스 변경 연산을 수행하여야 하므로 R-트리와 같은 전통적인 공간 인덱스 구조로는 처리하기 어렵다. 이러한 문제를 해결하기 위하여 객체의 움직임을 간단한 선형 함수로 가정하여 색인하는 연구들이 있어왔지만, 실제 응용에서는 객체의 움직임이 매우 복잡하므로 이러한 방법을 이용하기 적합하지 않다. 본 논문에서는 복잡한 움직임을 가지는 객체를 효율적으로 색인하기 위한 R-트리의 지연 갱신 기법을 제안한다. 이 기법은 객체가 이동할 때마다 트리의 구조를 변경하지 않고, 객체가 이전에 속해 있던 R-트리의 MBR(Minimum Bounding Rectangle)을 벗어날 때만 트리의 구조를 변경하므로 R-트리의 갱신 연산 비용을 크게 줄일 수 있다. 뿐만 아니라, 기본적인 R-트리의 구조와 연산을 그대로 이용하므로 다양한 R-트리 변종 트리에서도 쉽게 적용이 가능하고, R-트리를 이용하여 이미 구축되어 있는 다양한 응용 환경에 쉽게 이용할 수 있다.

1. 서 론

이동 컴퓨팅 응용은 이제 중요한 컴퓨터 응용 분야의 하나로 발달하였다. 이동 능력은 새로운 형태의 응용 시스템의 구현을 가능하게 해주었고, 새로운 시장으로서 자리매김 하고 있다. 특히 GPS(Global Positioning System)와 같은 위치 추적 장비의 발달은 일상생활에서도 쉽게 위치 추적 기능을 이용한 응용 환경을 이용할 수 있도록 해주고 있다. 이러한 위치 추적 기술은 현재는 단순한 차량 네비게이션 시스템이나 휴대폰 서비스 등에 이용되고 있지만, 앞으로는 전자 상거래나 물류 전송 시장 등에서 없어서는 안 될 중요한 요소가 될 것이다. 이러한 응용 환경을 위해서는 대량의 위치 데이타를 효율적으로 저장 관리할 수 있는 방법이 필요하다.

하지만, 전통적인 데이타베이스 시스템은 데이타에 명시적으로 변경 연산이 수행될 때까지는 불변하는 것을 가정하고 있으므로 이러한 위치 정보를 데이타베이스에 저장하기 위해서는 객체의 위치가 이동할 때마다 변경 연산을 수행해주어야 하는 문제점이 있다. 특히 이동 객체의 위치는 쉬지 않고 연속적으로 움직이기 때문에 데이타베이스가 처리해 주어야하는 데이타베이스 갱신 연산은 이동 객체의 수가 증가함에 따라 기하급수적으로 증가하게 된다. 더욱이, 일반적으로 공간상의 위치 정보는 다차원 데이타이므로 R-트리[1]와 같은 공간 인덱스를 이용하여 색인되는데, 이러한 인덱스에서 위치를 이동시키기 위해서는 단지 데이타만 변경해주는 것이 아니라 데이타 변경에 의하여 인덱스 노드들을 분할, 합병해 주어야하는 트리 구조의 구조적인 변경이 필요하다. 이미지 데이타베이스나 GIS(Geographic Information System)과 같은 기존의 R-트리 응용에서는 데이타는 한번 입력받으면 크게 변하지 않으므로 주로 삽입이나 삭제 연산만을 변경 연산의 방법으로 고려하여 왔으나, 연속적으로 움직이는 이동 객체의 위치 변경을 R-트리의 삭제, 삽입 연산만으로 처리하는 것은 큰 부담이 되므로 적합하지 않다.

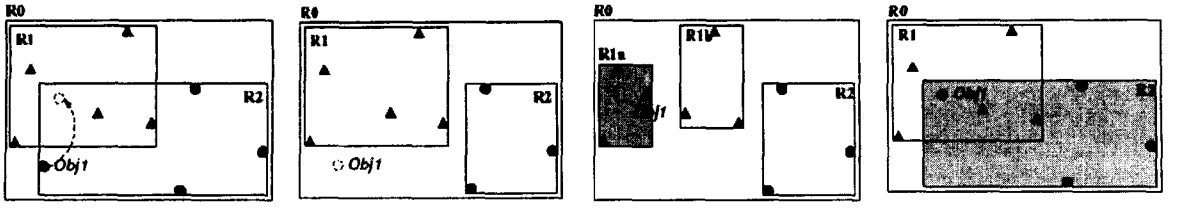
본 논문은 이러한 문제점을 해결하기 위하여 이동객체의 위치 색인을 위한 R-트리의 갱신 연산기법을 제안한다. 제안 기법은 이동 객체의 위치 변경을 객체의 이전 위치가 속한 R-트리의 MBR(Minimum Bounding Rectangle)과 새로운 위치와의 관계를 고려하여 변경연산을 수행하므로 기존의 R-트리의 삭제, 삽입 연산을 수행했을 때보다 연산 비용을 크게 줄일 수 있다. 본 논문에서는 제안기법을 R*-트리[2] 상에 구현하여 기존의 기법과의 성능을 비교 분석하였고, 이를 통하여 제안 기법이 다양한 환경에서 매우 효율적인 성능을 보여줌을 제시한다.

본 논문의 구성은 다음과 같다. 2장에서는 관련 연구를 간단히 살펴보고, 3장에서는 본 논문에서 제안하는 변경기법을 소개한다. 그리고 4장에서는 제안 기법의 성능을 실험을 통하여 분석하고, 5장에서 결론을 맺는다.

2. 관련 연구

이동객체의 위치와 같은 시공간 데이타를 색인하는 가장 간단한 방법은 R-트리와 같은 기존의 다차원 공간 인덱스 구조를 그냥 이용하는 것이다. 지금까지 다차원 공간 데이타를 위한 다양한 인덱스 구조들이 발표되어 왔다. 하지만, 앞에서 언급했듯이 기존의 다차원 인덱스 구조는 이동 객체의 위치를 저장하기에는 적합하지 않다.

이와는 별도로 최근에는 이동 객체의 위치를 저장하기 위한 인덱스 구조에 대한 연구가 활발히 진행 중이다. 이러한 연구들은 저장할 이동 객체의 위치의 종류에 따라 2가지로 나눌 수 있다. 하나는 이동 객체의 이동한 모든 경로를 저장하는 방법[3][4]이고, 다른 하나는 이동 객체의 현재 위치를 저장하는 기법이다. 본 논문은 후자의 문제를 다룬다. 이동 객체의 현재



(a) 변경전 : Obj1을 새 위치로 이동시키려함 (b) Obj1 삭제 : R2의 MBR이 변경됨 (c) Obj1을 새 위치에 삽입 : R1노드가 분할됨 (d) 지연 갱신 기법을 이용한 경우

그림 1. 기존의 R-트리 갱신 기법과 지연 갱신 기법의 수행 예

위치를 저장기법에 대한 대부분의 연구[5][6][7][8]는 객체의 위치를 단순한 선형 함수로 가정하고 객체의 속도와 방향만을 저장하는 기법들이다. 그리고 객체의 속도가 변경되었을 때만 데이타베이스를 변경하게 된다. 이러한 방법은 객체가 항상 같은 속력을 가지고 같은 방향으로 이동하고 있을 때는 데이타를 변경할 필요가 없으므로 효과적이지만, 실제 환경에서는 객체가 등속도 운동을 하는 경우가 거의 없으므로 실용적이지 못하다. 따라서 일반적으로 약간의 오차를 허용하는 방법으로 이용되지만 이 역시 객체의 움직임이 복잡하면 이용할 수 없고, 이러한 오차를 허용하기 힘든 응용 분야에서는 적용이 불가능하다.

최근에 이러한 문제를 해싱을 이용하여 극복하려는 새로운 기법[9]이 제안되었다. 이 방법은 전체 공간을 일정한 개수의 격자로 자르고 객체가 속한 격자의 번호만을 데이타베이스에 저장한 후 객체가 이 격자에 속한 동안은 위치의 변경을 수행하지 않고 다른 객체로 이동한 경우만 데이타베이스에 변경 연산을 수행한다. 이 방법은 위의 연구들과는 달리 복잡한 객체의 움직임에도 적용이 가능하지만, 객체의 위치가 편중된 경우 오버플로에 대한 처리에 대한 고려가 되어있지 않으므로 객체들의 위치 분포에 따라 성능이 급격히 나빠질 수 있는 단점이 있다.

3. 제안 기법

3.1 기본 아이디어

1장에서 설명했듯이 R-트리를 이용하여 이동 객체의 위치를 색인하는 방법은 문제가 있다. 예를 들어, 그림 1(a)와 같은 상황에서 Obj1의 위치가 그림과 같이 변경되었다고 가정하면 기존의 R-트리에서는 이전 위치를 인덱스에서 삭제하고 새로운 위치를 인덱스에 삽입하여야만 한다. 이때 리프노드의 최대 저장 가능 객체의 수를 5라고 가정하자. 위와 같은 경우 Obj1을 우선 삭제하면 객체를 삭제한 노드 R2는 그림 1(b)와 같이 MBR이 변경되어야 한다. 그리고 이는 R1 노드에도 반영되어야 한다. 실제 이러한 MBR의 변경은 트리의 루트까지 파급될 수 있다. 뿐만 아니라 경우에 따라서는 노드에 언더플로가 발생할 수 있고, 이러한 경우 트리 노드의 병합이 필요하다. 그리고 새로운 위치를 트리에 삽입하면 새로운 위치는 R1 노드에 삽입되어야 하는데 이 경우 R1 노드에 오버플로가 발생하므로 노드가 그림 1(c)와 같이 분할되어야 한다. 이러한 분할 역시 트리의 루트까지 파급될 수도 있고, 새로운 루트가 만들어질 수도 있다. 위의 예제의 경우 위치 변경을 위하여 4번의 디스크 쓰기 접근(R2, R1a, R1b, R0)이 필요하다. 그리고 트리 노드의 분할이나 병합은 트리의 루트까지 파급될 수 있으므로 최악의 경우는 훨씬 많은 수의 디스크 접근이 필요하다.

하지만, 위의 경우는 Obj1의 변경된 위치가 여전히 R2 노드 내부에 존재하므로 이 경우 R2노드 내부의 Obj1의 위치만 변경하더라도 그림 (1)d와 같이 R-트리의 구조에는 아무런 이상이 없다. 물론 MBR이 최소 경계 사각형이 되지 않지만, 실제 트리를 검색하거나 조작하는 데는 아무런 문제가 없다. 이와 같이 새로운 위치가 기존의 리프노드의 MBR의 범위를 벗어나지 않는 경우 노드 내부의 위치만을 변경하고, 트리의 구조 변경을 하지 않고 지연하는 방법이 본 논문에서 제안하는 지연 갱신 기법이다. 지연 갱신 방법으로 위의 예를 처리하는 경우 R2노드에 대한 디스크 쓰기 접근 1회면 처리가 가능하다. 따라서 이동 객체의 위치 변경 비용을 크게 줄 일 수 있다. 특히 이동 객체의 경우 새로운 위치는 항상 이전 위치의 인근에 있을 수밖에 없으므로 효용성은 더욱 커지게 된다.

3.2 지연 갱신 알고리즘

```
   Procedure LazyUpdate(oid, newpos)
   Input: oid 객체 ID, newpos 새로운 위치
   begin
1:   oldpos ← GetPosition(oid);
2:   N ← FindLeafNode(oldpos);
3:   if newpos ∈ N.MBR then
4:      N.Entry[oid].pos ← newpos;
5:      WriteNode(N);
     else
6:      Delete(oid,oldpos);
7:      Insert(oid,newpos);
     end
   end
```

알고리즘 1. 지연 갱신 알고리즘

지연 갱신 기법의 구체적인 알고리즘은 알고리즘 1과 같다. 새로운 위치가 기존의 노드의 MBR내부이면 지연 갱신을 수행하고, 그렇지 않은 경우는 삭제, 삽입 방법을 이용하였다.

3.3 Extended MBR

지연 갱신 기법의 경우 객체가 어떠한 MBR의 가장자리에서 있는 경우 쉽게 MBR 밖으로 이동할 수 있다. 이러한 경우 MBR을 인위적으로 일정한 크기만큼 확장하여 사용하면 지연 갱신 기법의 효율을 높일 수 있다. 이렇게 리프노드의 MBR을 일정한 크기 ε만큼 확장한 것이 Extended MBR(EMBR)이다. EMBR의 ε를 크게 주면 갱신 연산 비용은 줄지만 리프 노드간의 오버랩이 커지므로 검색 성능에는 악영향을 끼칠 수 있다.

4. 성능 평가

본 연구에서는 이러한 지연 갱신 기법을 R*-트리에 구현하여

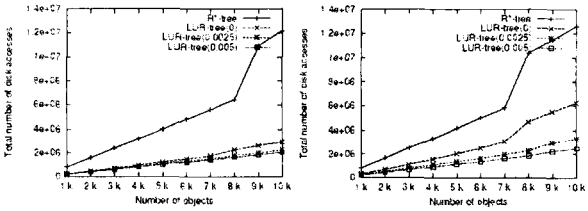
기존의 기법과 비교 분석해보았다. 실험은 펜티엄 1GHz CPU, 512MB의 메모리, 40GB E-IDE HDD를 가진 Linux 기계

표 1. 실험 데이타

	초기 분포	이동방향
U-R dataset	Uniform 분포	무작위
U-D dataset	Uniform 분포	방향성 지님
G-R dataset	Gaussian분포	무작위
G-D dataset	Gaussian분포	방향성 지님

에서 수행되었다. 이동 객체를 위한 실제 데이타는 공개된 것이 없는 관계로 실험을 위해서는 [10]을 이용하여 데이타를 생성하여 사용하였다. 데이타 분포에 따른 성능을 비교하기 위하여 표1과 같이 4가지의 데이타를 생성, 실험하였다.

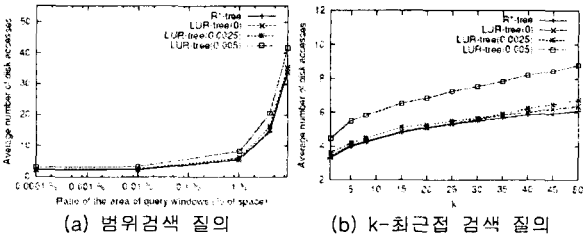
실험 결과에서 R*-tree는 전통적인 R*-트리이고, LUR-tree는 지연 기법을 적용한 R-트리를 의미한다. 그리고 뒤의 숫자는 EMBR의 확장값 ε를 의미한다. EMBR의 효과를 비교 분석하기 위하여 ε의 크기에 따라 각 0, 0.0025, 0.005의 세 가지의 지연 갱신 기법 트리를 이용하였다. 여기서 ε가 0인 것은 EMBR을 이용하지 않고 기존의 MBR을 이용한 트리이다.



(a) U-R dataset (b) G-D dataset

그림 2. 이동객체 수에 따른 갱신 연산의 총 디스크 접근 회수

그림 2는 이동객체의 수를 1,000개에서 10,000개까지 변화시켜가며 측정한 각 객체 당 100번의 갱신 연산을 수행했을 때의 총 디스크 접근 회수이다. 4가지 데이타 종류에 모두 유사한 결과를 보였으므로 여기서는 U-R data와 G-D data만을 보겠다. 자세한 실험 결과는 [11]을 참고하기 바란다. 실험 결과 지연 갱신 기법을 이용하면 디스크 접근 회수를 평균 2배 이상 줄일 수 있었고, ε를 증가시킴에 따라 그 효과는 더 커졌다.



(a) 범위검색 질의 (b) k-최근접 검색 질의

그림 3. 검색 질의 수행 시 평균 디스크 접근 회수

그림 3(a)는 측정한 범위 검색질의를 수행했을 때의 평균 디스크 접근 회수이다. 여기서는 질의 범위를 전체 공간 크기의 0.00001%에서 1%까지 변경시켜가며 각 100회의 질의를 수행했을 때의 평균 디스크 접근 회수이다. 그리고 그림 3(b)는 k-최근접 검색 질의의 결과이다. k를 1부터 50까지 변경해가며 각 100회의 질의를 수행했을 때의 평균 디스크 접근 회수이다. 지면 관계상 G-D dataset의 결과만 수록하였다. 자세한 실험 결과는 [11]을 참고하기 바란다. 실험 결과 지연 갱신 기법을 이용하는 경우 MBR간의 오버랩이 증가하므로 약간의 오버헤드가 있었지만, 질의 수행 성능은 크게 차이가 나지 않았다. 그리고 예상했던 바와 같이 EMBR의 ε가 커짐에 따라 접근 횟수가 증가하였다.

실험 결과, 지연 검색 기법은 갱신 연산의 비용은 크게 감소시켜주는 장점이 있었다. 물론, 검색 질의 시에는 약간의 오버헤드를 가지고 있지만 이동 객체의 위치 추적과 같은 응용은 데이타의 갱신 연산이 매우 많고 갱신 연산의 효율적인 처리가 매우 중요하므로 이러한 경우, 지연 검색 기법을 이용하면 성능을 크게 개선할 수 있다.

4. 결론

이동 객체의 위치는 변경이 끊임없이 발생하는 특성 때문에 이러한 갱신을 고려하지 않은 기존의 공간 인덱스 기법으로는 너무 많은 갱신 연산 때문에 색인하기 힘든 문제점이 존재한다. 기존의 연구들 역시 이동 객체의 움직임을 단순한 선형 함수로 가정하고 있기 때문에 실제 응용환경에서는 이동 객체의 복잡한 움직임을 처리할 수 없는 문제점이 있다. 이러한 문제를 해결하기 위해서 본 논문에서는 R-트리의 갱신 연산을 이동 객체의 새로운 위치와 기존에 속해있던 MBR의 관계를 이용하여 개선하여 R-트리에서의 갱신 연산 비용을 크게 줄일 수 있는 지연 갱신 기법을 제안하였다. 이 방법을 이용하면 R-트리의 갱신 비용을 크게 줄일 수 있을 뿐 아니라 R-트리의 기존 알고리즘을 그대로 이용할 수 있어서 기존 응용환경에 쉽게 적용할 수 있는 장점이 있다. 그리고 실험을 통하여 제안기법의 우수성을 실증하였다.

참고문헌

[1] Antonin Guttman, R-Trees: A Dynamic Index Structure for Spatial Searching, Proc. SIGMOD, 1984

[2] Norbert Beckmann, Hans-Peter Kriegel, Ralf Schneider, Bernhard Seeger, The R*-Tree: An Efficient and Robust Access Method for Points and Rectangles, Proc. SIGMOD, 1990

[3] Dieter Pfoser, Christian S. Jensen, Yannis Theodoridis, Novel Approaches in Query Processing for Moving Object Trajectories, Proc. VLDB, 2000

[4] Yufei Tao, Dimitris Papadias, MV3R-Tree: a spatio-temporal access method for timestamp and interval queries, Proc. VLDB, 2001

[5] George Kollios and Dimitrios Gunopulos and Vassilis J. Tsotras, On Indexing Mobile Objects, Proc. PODS, 1999

[6] Pankaj K. Agarwal and Lars Arge and Jeff Erickson, Indexing Moving Points, Proc. of PODS, 2000

[7] Simonas Saltenis, Christian S. Jensen, Scott T. Leutenegger, Mario A. Lopez, Indexing the Positions of Continuously Moving Objects, Proc. SIGMOD, 2000

[8] Simonas Saltenis, Christian S. Jensen, Indexing of Moving Objects for Location-Based Services, Proc. ICDE, 2002

[9] Zhexuan Song and Nick Roussopoulos, Hashing Moving Objects, Proc. of Mobile Data Management, 2001

[10] Yannis Theodoridis, Jefferson R. O. Silva, Mario A. Nascimento, On the Generation of Spatiotemporal Datasets, Proc. of Symp. on Spatial Databases, 1999

[11] Dongseop Kwon, Sangjun Lee, Sukho Lee, Indexing the Current Positions of Moving Objects Using the Lazy Update R-tree, Proc. of Mobile Data Management, 2002