이산적으로 변화하는 시공간 객체의 이력 관리를 위한 다중 버전 색인의 설계

이양구[0], 이응재, 류근호
충북대학교 데이터베이스 연구실
{leeyangkoo[0], eungjae, khryu}@dblab.chungbuk.ac.kr

Design of Multiversion Index for History Management of Discretely Changing Spatio-Temporal Objects

Yang-Koo Lee[0], Yeung-Jae Lee, Keun-Ho Ryu
Dept. of Computer Science, Chungbuk National University

요 약

실세계의 공간 객체는 시간이 흐름에 따라 위치 또는 모양과 같은 속성이 변경되고, 이로 인해 대용량의 이력 데이터가 발생된다. 이러한 시공간 객체의 이력을 효율적으로 저장하고 빠르게 검색하기 위하여 색인 기법들에 대한 연구가 최근까지 활발히 진행되어 왔다. 그러나 기존의 공간 데이터베이스의 색인 기법들은 시공간 객체의 이력을 지원하기에 적절치 못하거나 실세계의 객체에 직접 적용하기 어려운 문제점을 갖는다. 따라서 이 논문에서는 이산적으로 변화하는 시공간 객체의 이력을 효율적으로 관리하기 위해 다중 버전 구조의 색인을 설계한다. 그리고 실세계의 시공간 객체가 갖는 일반적인 특징을 분석하여 그 결과를 토대로 확장된 알고리즘을 제안한다.

1. 서 론

토지 구역이나 통신 선로 등과 같은 실세계 객체들은 시간이 흐름에 따라 자신의 위치나 형태 등과 같은 공간 속성이 변화하는 시공간 객체이다[1,2]. 예를 들어, 토지 구역의 경우 객체는 시간이 경과함에 따라 토지의 합병, 분할, 재구성 등에 의해 객체의 형태나 위치에 대한 정보가 변할 수 있으며, 객체의 변화는 이산적이고 불규칙적이라는 특징을 갖는다. 이러한 시공간 객체를 관리하기 위해서는 객체의 현재 정보와 모든 이력 정보를 저장하여야 하기 때문에 데이터의 양이 방대하고 구조가 복잡해진다. 따라서 시공간 객체의 이력을 효율적으로 저장하고 조작 및 검색을 하기 위한 색인 방법이 필요하다.

이 논문에서는 토지 구역과 같이 이산적으로 변화하는 시공간 객체의 이력을 효율적으로 관리하기 위하여 다중 버전 구조의 색인을 설계한다. 그리고 시공간 객체가 갖는 특징을 분석하여, 그 결과를 토대로 기존의 시공간 색인 방법인 HR+-tree의 알고리즘을 확장한다. 다중 버전 구조는 현재 또는 특정 시점에 대한 질의에 빠르게 접근할 수 있고, 비교적 짧은 연속적인 시간 구간에 대한 질의에도 좋은 성능을 유지한다.

이 논문의 구성은 다음과 같다. 먼저 2장에서는 시공간 객체가 갖는 일반적인 특징을 분석하고, 3장은 관련 연구로써 기존에 제안된 색인 방법들의 장단점을 분석한다. 그리고 4장은 다중 버전 공간 색인의 기본 구조에 대해 기술하고 5장에서는 확장된 알고리즘을 기술한다. 마지막으로 6장에서 결론 및 향후 연구 방향을 제시한다.

† 이 연구는 한국 과학재단 지정 청주대학교 RRC(정보통신연구센터)의 지원으로 수행되었음

2. 시공간 객체의 일반적인 특징

시공간 객체는 시간 변화에 따라 공간상에서 다양한 유형으로 자신의 이력을 변화시킨다. 예를 들어 토지의 경우 지형의 분할에 의한 축소, 지형의 합병, 지형의 재구성 등에 의한 공간 정보의 변화뿐만 아니라 지명, 용도, 소유주 등의 비공간적인 정보도 변화하는 특징을 갖는다.

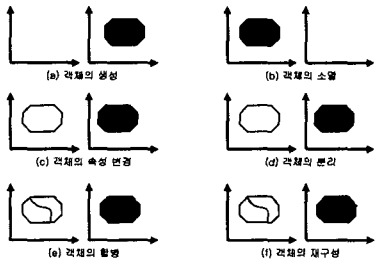

그림 1. 시공간 객체의 변화 유형

그림 1은 시간에 따른 공간 객체의 생성, 객체의 소멸, 비공간적인 정보의 변경과 객체의 분리, 합병, 구역의 재구성 등의 이력의 변화 유형을 나타낸다. 이러한 유형은 객체의 삽입과 갱신에 영향을 미친다. 예를 들어 객체가 생성될 경우는 삽입만 고려하면 되지만, 객체의 속성이 변경되거나 재구성될 경우는 삽입과 함께 변경되는 객체의 이력에 대한 갱신도 고려하여야 한다.

시공간 객체에 대해 고려해야할 또 다른 특징은 다음과 같이 나뉠 수 있다.

(i) 시간에 따라 객체는 이산적으로 변화한다.

(ii) 객체가 변경되는 시간이 매우 불규칙적이다.
(iii) 객체의 상태가 매우 지속적으로 유지된다.

여기서 객체의 변화가 이산적이고, 불규칙적이라는 것은 객체의 미래 상태에 대한 예측이 어렵고, 갱신 주기가 일정하지 않다는 것을 의미한다. 그리고 객체의 상태는 지속적으로 유지되므로 객체가 마지막으로 갱신된 시간 이후의 상태 정보를 반영하기 어렵다.

3. 관련 연구

일반적으로 공간 객체를 관리하기 위한 색인으로 R-tree[3]가 있다. 그러나 R-tree는 객체의 현재 상태만을 나타낼 수 있기 때문에 시공간 객체의 이력 정보를 표현하지 못한다. 최근에는 R-tree를 다차원으로 확장하여 시간과 공간을 표현할 수 있는 색인 방법들이 연구되고 있다. 대표적인 예로써 3D R-tree[4]는 R-tree를 이용하여 시나리오가 정해져 있는 멀티미디어 데이터 색인을 위하여 제안되었다. 그러나 3D R-tree는 "*now*" 또는 "*until changed*" 데이터에 대한 처리가 어렵기 때문에 객체의 모든 움직임은 시간에 대해 closed 되어 있다고 가정한다. 그러므로 3D R-tree는 모든 객체들의 *lifespan*이 미리 알려져 있고 정적인 객체들만 입력될 수 있다. 이러한 가정은 실제 실세계에서 필요로 하는 이력에 대한 개념을 지원하기 어렵다는 문제점을 갖는다.

다중 버전 구조의 색인으로 HR-tree[5]는 시공간 객체의 연속적인 상태를 표현하기 위하여 R-tree에 오버래핑의 개념을 적용하여 확장한 색인 방법이다. HR-tree에서는 객체의 이력 정보를 유지하기 위하여 각각의 timestamp에서 변경된 node들만 복제하여 새로운 R-tree를 구성하여 이전의 R-tree에 *branches*를 통해 연결한다. 그러나 HR-tree는 단 하나의 entry가 변경되었다 하더라도 전체 경로를 복제해야 하기 때문에 시간에 따라 객체가 변경되는 빈도에 따라 저장 공간이 현저히 증가하게 되는 단점이 있다.

HR-tree의 단점을 보완할 목적으로 같은 node에 다른 버전의 entry를 허용하지 못한다는 제약을 없애 node의 활용을 높인 HR+-tree[6]가 제안되었다. HR+-tree에서 새로운 node는 오직 overflow가 발생할 때만 생성된다. 따라서 저장 공간은 HR-tree보다 더 효율적으로 활용될 수 있다.

4. 다중 버전 공간 색인 구조

다중 버전 색인의 전체 구조는 하나의 R-tree와 다수의 B-tree의 조합으로 구성되고, 각각의 버전을 관리하는 배열로 구성된다. 그리고 하나의 버전에 다른 버전의 entry들을 저장할 수 있고, node는 overflow가 발생할 경우만 분할된다. 또한 각각의 버전은 root node에서 overflow가 발생할 경우에 새롭게 생성된다.

객체의 entry는 <*S*, T_s, T_e, *Pointer*>로 구성된다. S는 R-tree에서 정의된 MBR이고 T_s(T_e)는 객체가 삽입(삭제)된 timestamp를 나타낸다. 마지막으로 *Pointer*는 하위 node 또는 실제 레코드를 가리키는 pointer이다. entry의 *lifespan*은 *semi-closed interval*로 표현한다. 만약, entry가 현재 시간까지 삭제되지 않았다면 entry의 lifespan은 (T_s, *)로 표현하고, entry가 삭제되었다면 (T_s, T_e)로 표현한다. 그리고 T_e가 "*" 인 상태를 *alive*라고 하고 그렇지 않으면 entry는 *dead*라고 한다.

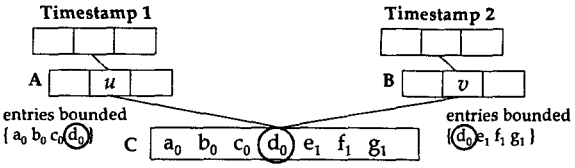

그림 2. Node 구조

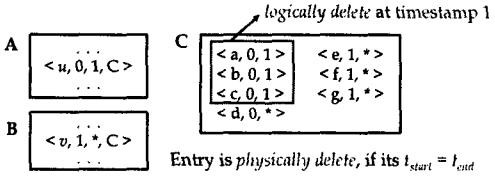

그림 3. node의 entry 정보

그림 2와 그림 3은 timestamp1과 timestamp2에서 leaf node C를 공유하고 있는 node의 구조와 실제 저장된 entry들을 나타내는 예이다. 여기서 entry d, e, f, g는 현재 *alive*이고, a, b, c는 *dead*이다. 즉 그들은 timestamp 1에서 논리적으로 삭제되었다. 만약, entry의 T_s = T_e 일 경우 객체는 물리적으로 삭제된다. 여기서 논리적 삭제는 *alive* 상태였던 entry의 T_e를 현재 timestamp로 변경하고 현재 timestamp를 T_s로 갖는 새로운 *alive* entry를 생성하는 것을 의미하고, 물리적 삭제는 entry를 영구적으로 삭제하는 것을 의미한다.

5. 확장된 알고리즘

다중 버전 공간 색인 구조는 각각의 timestamp에 대하여 객체의 entry는 *semi-closed interval*을 가지기 때문에 객체의 변경이 발생할 때까지 이력을 유지할 수 있는 장점이 있다. 이로 인해 질의 처리 시 복잡한 연산 없이 시간에 대해 정적인 상태에 있는 객체에 빠른 접근을 유도할 수 있다. 그러나 객체의 변화가 발생할 경우에는 객체의 가장 최근의 entry를 찾아 삭제하고 새로운 *lifespan*을 갖는 entry를 삽입하여야 한다. 이렇게 변경전의 객체와 변경후의 객체를 명확히 분리함으로써 보다 정확한 질의 결과를 얻을 수 있다.

그러나 이러한 과정은 분명하게 색인의 성능에 영향을 미친다. 특히, 객체의 갱신이 빈번하게 발생하는 객체의 경우는 심각한 성능 저하를 초래할 수 있다. 따라서 다중 버전 공간 색인 구조는 객체의 변화가 이산적이고 불규칙적으로 발생하는 객체, 그리고 객체의 *lifespan*이 장기간 지속되는 실세계 객체를 색인 하는데 적합하다.

이 논문에서는 이러한 실세계 객체가 갖는 특징을 색인에 반영하여 시공간 객체의 이력을 보다 명확하게 관리할 수 있도록 기존의 시공간 색인 방법인 HR+-tree의 알고리즘을 확장하여 이 문제를 적용하였다.

5.1 검색 알고리즘

다중 버전 공간 색인에서 node는 *branches*에 의해 공유되기 때문에 범위 질의인 경우 공유되는 node에 의해 중복이 발생한

다. 이러한 중복 방문을 피하기 위하여 이 논문에서는 HR-tree의 negative pointer과 유사한 방법으로 질의 시점의 timestamp를 저장하고 있는 check time pointer를 사용한다.

먼저 node를 방문할 때 node의 check time pointer가 질의 시점과 같은가를 비교한다. 만약 같다면 그 node는 이미 방문되어진 node이므로 더 이상 방문하지 않게 되고, 만약 다르다면 방문된 node의 check time pointer를 질의 시점의 timestamp로 변경하여 나중에 다시 방문되지 않도록 한다. 표 1은 자세한 검색 알고리즘을 기술하고 있다.

표 1 검색 알고리즘

Algorithm Search(MBR S, T_s, T_e, Search Time CT)
1. Ts와 Te를 포함하는 root를 검색
2. S와 overlap되는 node를 검색
   만약 node의 check time pointer가 CT와 같으면
      2단계를 수행
   그렇지 않으면
      S와 overlap되는 entry를 검색
3. 2단계를 반복 수행

5.2 삽입 및 갱신 알고리즘

시공간 객체의 현재 상태는 데이터베이스의 갱신이 발생한 가장 최근의 timestamp로부터 지속적으로 유지된다. 만약, 새로운 객체가 삽입될 경우에는 삽입될 위치를 찾아 객체를 삽입하면 된다. 그러나 기존의 객체가 변경되는 경우는 먼저 변경된 객체의 가장 최근의 entry를 찾아 그 객체의 T_e를 현재 timestamp로 변경한 후에 삽입하게 된다.

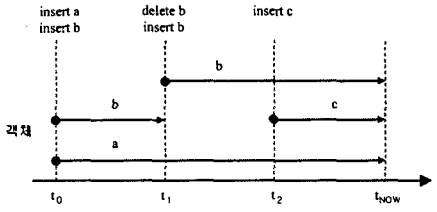

그림 4. 시간 변화에 따른 객체의 life-time

그림4는 시간에 따른 객체의 속성 변화를 그림으로 나타낸 것이다. 만약, 객체 b가 T_0에서 T_{now}까지 객체의 위치가 변경되었다고 가정할 때, timestamp T_1에서 객체는 새로운 T_s를 갖고 T_0에서의 T_e는 T_1의 T_s로 변경되어야 한다. 즉, T_0는 T_1에서 논리적으로 삭제되어야 한다. T_2에서도 마찬가지로 T_1에서의 T_e가 변경되어야 하고 새로운 T_s와 $*(now)$의 범위를 갖는다. 따라서 객체가 변경되면, 기존 객체의 삭제 후 삽입 과정을 수행한다. 표 2는 자세한 삽입 및 갱신 알고리즘을 기술한다.

5.3 삭제 알고리즘

삭제 알고리즘은 HR+tree와 유사하게 진행된다. 먼저 삭제할 객체를 찾은 후, T_s와 T_e가 같은지를 확인하고 만약, 같으면 entry를 물리적으로 삭제하고 그렇지 않으면 엔트리를 논리적으로 삭제한다. 그리고 HR+tree의 알고리즘에 따라 객체의 삭제로 인한 node의 변화를 조정한다.

표 2. 삽입 및 갱신 알고리즘

Algorithm Insert(Entry On)
1. Find Leaf(On)를 호출하여 entry를 검색
   만약, 찾았다면
      entry를 논리적으로 삭제
   Choose subtree(On)를 호출하여 entry를 삽입
2. 만약, 객체의 삽입 후 node가 overflow이면
   treat overflow를 호출하여 node를 분할
   상위 node 쪽으로 성장하며 node의 변화를 조정
3. 만약, root에서 version split이 발생하면
   현재 timestamp의 root table에 새로운 entry들을 생성
4. 만약, root에서 key split이 발생하면
   root table에서 가장 최근의 entry들을 갱신

6. 결론

이 논문에서는 시공간 객체의 현재 정보와 과거의 이력 정보를 관리하기 위하여 색인 설계 시 고려해야할 객체의 특징을 분석하였다. 그리고 다중 버전 구조의 색인을 설계하였고, 시공간 객체가 갖는 특징을 색인에 적용하여 알고리즘을 확장하였다. 이 논문에서 사용한 다중 버전 색인 구조는 변경되지 않은 객체에 대하여 *lifespan*을 유지할 수 있는 장점이 있고, 이로 인해 질의 처리 시 복잡한 연산 없이 시간에 대해 정적인 상태에 있는 객체를 효율적으로 관리할 수 있다.

향후 연구로는 다양한 성능 평가를 통해 다른 색인 방법과의 비교 분석이 필요하다. 그리고 갱신으로 인한 오버헤드를 감소시킬 수 있는 방법에 대한 연구가 필요하다.

참고 문헌

[1] R. H. Guting, M. H. Bohlen, M. Erwig, C. S. Jensen, N. A. Lorentzos, M. Schneider, M. Vazirgiannis, "A Foundation for Representing and Querying Moving Objects", ACM Transactions on Database Systems, 25(1), 2000.

[2] 신기수, 안윤애, 배종철, 정영진, 류근호, "GIS를 이용한 시공간 이동 객체 관리 시스템", 한국정보처리학회 논문지, 제8-D권, 2호, 2001년 4월.

[3] A. Guttman, "R-trees: A Dynamic Index Structure for Spatial Searching", Proc. ACM SIGMOD Conf, 1984.

[4] M. Vazirgiannis, Y. Theodoridis, T. Sellis, "Spatio-Temporal Composition and Indexing for Large Multimedia Applications", Multimedia Systems, 6(4), 1998.

[5] M. A. Nascimento, J. R. O. Silva, "Towards Historical R-trees", ACM SAC, 1998.

[6] Y. Tao, D. Papadias, "Efficient Historical R-Trees", Proc. of IEEE Conf. on Scientific and Statistical Database Management, 2001.