과거 및 현재 정보의 통합 관리를 위한 이동체 색인 기법

이양구[0*], 이응재*, 남광우**, 류근호*
*충북대학교 데이터베이스 연구실, **한국전자통신연구원
*{leeyangkoo[0], eungjae, khryu}@dblab.chungbuk.ac.kr
**kwnam@etri.re.kr

Unified Index for Managing of Current Data and Past Trajectory of Moving Object Information

Yang-Koo Lee*, Eung-Jae Lee*, Kwang-Woo Nam**, Keun-Ho Ryu*
*Database Laboratory, Chungbuk National University, Korea
**Spatial Information Technology Center,
Electronics and Telecommunications Research Institute, Korea

요 약

최근 이동 통신 기술과 GPS 기술의 발달은 실시간으로 이동하는 차량이나 비행기 등과 같은 이동 객체의 위치 정보 서비스를 가능하게 하였고, 이로 인해 이동 객체의 정보를 빠르게 검색하고 저장하기 위한 기술에 대한 연구가 활발히 진행되고 있다. 지금까지의 연구는 이동 객체의 과거 궤적 정보, 현재 및 미래 정보를 다루기 위한 구조가 각각 분리되어 연구되어 왔다. 하지만 이동 객체는 시간의 흐름에 따라 연속적으로 변화하기 때문에 과거 정보와 현재 정보를 별도로 관리할 경우, 빈번한 위치 갱신이 발생하는 이동 객체의 환경에서는 심각한 처리 비용이 요구된다. 따라서 이 논문에서는 이동 객체의 위치 정보를 효과적으로 관리하기 위하여 이동 객체의 과거 궤적 정보뿐만 아니라 현재의 정보까지도 하나의 색인 내에 통합하여 관리하는 방법을 제안한다.

제안된 방법에서는 범위 질의와 같은 좌표 기반의 질의뿐만 아니라 이동 객체의 궤적과 관련된 질의까지도 처리 가능하도록 기존의 TB-tree를 확장한다. 또한 기존의 이동 객체 관리 기법들에서 처리하던 이동 객체의 궤적 정보를 저장하기 위한 방법을 개선하여 이동 객체 위치 정보의 중복 저장 문제를 해결하고, 전체 색인의 크기를 크게 줄일 수 있는 방법을 제안한다. 그리고 현재 정보의 관리를 통해 새로운 데이터 삽입 시 방문하는 node의 수를 줄이고, 삽입 시간을 단축하는 방법을 제시한다.

1. 서 론

최근 GPS 기술과 이동 통신 기술의 발달로 인해, 특수 목적으로만 사용되던 위치 추적 기술은 차량, 비행기, 선박, 휴대폰 사용자 등에도 그 응용 영역이 확대되고 있다.

차량이나 비행기, 선박, 휴대폰 사용자 등과 같은 실세계의 객체들은 시간이 경과함에 따라 자신의 위치를 연속적으로 변경하는 이동 객체이다[1,2]. 이러한 이동 객체는 시간의 흐름에 따라 매우 방대한 양의 정보를 포함하기 때문에 이들 정보를 효율적으로 관리하기 위해서는 객체를 빠르게 저장하고 검색할 수 있는 색인 기법이 필요하다.

이동 객체의 색인에 대한 연구는 크게 현재 위치 색인[3,4]과 과거 궤적 색인으로 나눌 수 있다. 현재 위치 색인은 객체의 현재 위치를 관리하고, 색인에 저장된 좌표와 속도 정보를 이용하여 가까운 미래 위치에 대한 검색을 수행하는데 초점을 맞추고 있다. 과거 색인은 객체의 과거 위치 정보와 궤적 정보를 관리하며, 이동 객체의 궤적은 시간 축을 포함하는 3차원 공간상에서 객체의 위치를 나타내는 두 점을 연결한 선분으로 표현된다.

기존에 제안된 과거 위치와 궤적 정보를 지원하는 색인으로는 3DR-tree, STR-tree, TB-tree 등이 있다. 3DR-tree[5]는 R-tree에 단순히 시간차원을 추가하여 3차원으로 표현한 구조로써 기존의 R-tree[6]에서 나타나는 Dead Space, Overlap등의 단점을 가지고 있고, 궤적 질의를 효율적으로 처리하지 못한다. STR-tree와 TB-tree[7]는 이동 객체의 궤적 정보를 효율적으로 관리하기 위한 구조로써 이동 객체의 공간적인 위치에 대

한 질의 처리 능력보다는 궤적과 관련된 질의의 빠른 처리를 위하여 궤적을 보호하며 정보를 저장하는 특성이 있다.

실세계의 이동 객체들은 시간 경과에 따라 연속적으로 자신의 위치를 변경시키기 때문에 객체의 새로운 위치에 대한 삽입이 매우 빈번하게 발생한다. 따라서 질의 성능뿐만 아니라 새로운 위치 정보에 의한 삽입 비용을 줄이는 것이 중요하다. 또한, 이동 객체를 효율적으로 관리하기 위해서는 객체의 현재 위치에 대한 검색이 가능해야 하고, 시간에 따라 증가하는 색인의 크기를 감소시킬 수 있는 방법이 필요하다.

이 논문에서는 이동 객체의 현재 위치 정보를 별도로 관리하여 새로운 객체의 삽입 시 발생하는 비용을 감소시키는 방안을 제시하고, 이동 객체의 궤적을 표현하기 위해 중복된 정보를 제거하여 색인의 크기를 감소시키는 방법을 제안한다. 또한, 이동 객체의 현재 위치 정보와 과거 궤적 정보를 하나의 색인으로 통합 관리하는 방법을 제안한다. 현재 데이터에 대한 질의는 객체의 가장 최근의 정보를 포함하는 엔트리의 timestamp를 "now(*)"로 표현하여 처리한다.

이 논문의 구성은 다음과 같다. 먼저 2장에서는 관련 연구를 통해 기존에 연구된 이동 객체의 과거 색인 구조를 소개하고, 3장에서는 기존에 연구된 색인 구조의 분석을 통해 이 논문에서 다루고자 하는 문제를 정의한다. 4장에서는 이 논문에서 제안하는 색인의 구조와 알고리즘을 기술하고, 마지막으로 5장에서 결론 및 향후 연구를 기술한다.

2. 관련 연구

이동 객체를 색인하기 위한 색인 기법들은 대부분 2차원 공간 R-Tree를 3차원으로 단순 확장한 구조 또는 그 변형 구조로 나타난다. 이 장에서는 기존의 이동 객체 색인 기법인 3DR-tree, STR-tree, TB-tree 등에 대해 소개하고 각각의 장단점을 기술한다.

2.1 3D R-tree

3D R-tree는 2차원 공간에 시간 차원을 추가한 구조로써 이차원 영역(region)에 시간 축을 고려한 3차원 MBB의 형태로 표현한다.

그림 1은 3D R-tree의 구조를 나타내고 있다. 왼쪽의 그림은 객체의 시간에 따른 변화를 3차원의 구조로 나타낸 것이다. 여기서 MBB A, B, C, D는 각각의 이동 객체를 나타내고, MBB R1, R2는 MBB A,B와 C,D를 포함하는 상위 node를 나타낸다. 오른쪽의 그림은 왼쪽의 그림을 tree로 나타낸 것이다.

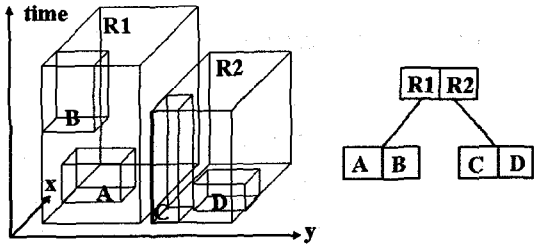

그림 1. 3D R-tree의 색인 구조

3D R-tree는 불필요한 정보가 없기 때문에 저장 공간의 사용이 경제적이고, time interval 질의에 효율적이라는 장점이 있다. 그러나 객체의 위치가 정적일 경우 Dead-space가 많아짐에 따라 색인 자체에 성능 저하가 발생할 수 있으며, 시간 영역의 범위가 증가함에 따라 timestamp 질의의 성능은 떨어진다.

2.2 STR-tree

STR-tree는 R-tree를 3차원으로 확장한 3DR-tree에 이동 객체의 궤적을 다루기 위하여 알고리즘을 수정한 tree이다. STR-tree의 엔트리 구조는 <ID, trajectory#, MBR, orientation>으로 구성된다. 여기서 orientation은 {1,2,3,4}로 표현되는 객체의 이동 방향을 나타낸다.

그림 2는 시간 차원으로 확장된 R-tree의 MBB와 이동객체의 이동을 line segments로 나타낸 것이다. 여기서 이동 객체의 이동 방향은 {1,2,3,4}의 숫자 도메인으로 나타낸다. 그림에서와 같이 이동 객체의 궤적은 R-tree의 MBB를 시간에 따라 연속적으로 연결하여 표현한다.

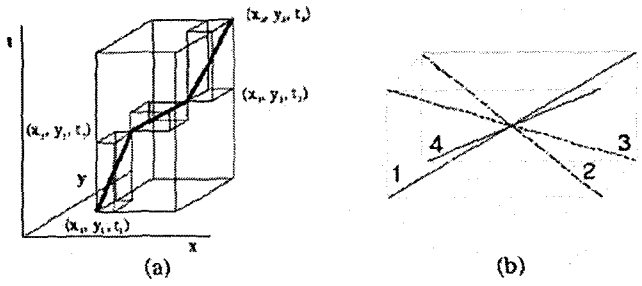

그림 2. (a) line segments mapping
(b) Trajectory의 표현

STR-tree는 객체의 삽입/분할 방법이 R-tree와 다르다. STR-tree의 삽입은 공간 근접뿐만 아니라 일부 궤적 보존도 고려한다. 예를 들어 새로운 line segment의 삽입이 발생하면 STR-tree는 가능한 이전에 삽입되었던 궤적에 가까운 곳에 삽입한다. 이러한 방법은 이전의 궤적을 포함하는 node를 넓혀주는 새로운

FindNode 알고리즘이 필요하다. 만약, 이전의 궤적이 포함된 node에 새로운 엔트리가 저장될 빈 공간이 있으면 새로운 segment를 삽입하고, 그렇지 않으면 node는 분할한다.

분할 알고리즘은 p라는 보존 파라메타를 포함한다. 이 파라메타는 궤적 보존에 대한 저장 level의 수를 의미한다. 먼저 분할이 발생하면, p-1 부모 node가 full인지 확인하고 full이 아니면 leaf node를 분할하고, 그렇지 않으면, 현재 삽입 경로의 오른쪽의 모든 node를 포함하는 subtree로부터 leaf node를 선택한다.

STR-tree는 저장되는 객체의 수가 많아지면, 성능이 떨어지고, 궤적 질의를 제외하면, 단순히 R-tree를 3차원으로 확장한 3DR-tree보다 성능이 떨어진다.

2.3 TB-tree

TB-tree는 이동 객체의 궤적 보존을 목적으로 제안된 색인 방법으로 leaf node는 같은 궤적에 속하는 segment만을 포함한다. TB-tree는 궤적을 정확하게 보존하는 것이 목적이기 때문에, 궤적 질의에 매우 효율적이다. 그러나 일반적인 시공간 범위 질의에서는 STR-tree와 3DR-tree보다 성능이 떨어진다.

그림 3은 TB-tree의 구조를 나타낸 것이다. 동일한 객체를 저장한 node는 연결 리스트를 통해 서로를 연결하여 궤적에 대한 질의 처리의 효율을 높였다.

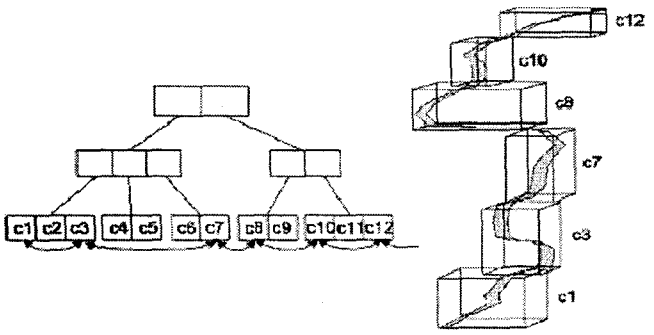

그림 3. TB-tree의 구조

새로운 객체를 삽입하기 위해 먼저 최근의 궤적을 포함하는 leaf node를 찾아 삽입한다. 만약, leaf node가 full이면, node는 분할되지 않고 새로운 node를 추가하여 삽입함으로써 객체의 궤적을 보존한다. 여기서 새롭게 추가되는 leaf node는 non-full인 부모 node로부터 rightmost 경로로 삽입된다. 따라서 TB-tree는 시간의 흐름에 따라 왼쪽으로부터 오른쪽으로 성장한다.

3. 문제 정의

이동 객체는 시간이 경과함에 따라 자신의 위치를 연속적으로 변경하기 때문에, 새로운 객체의 정보에 의한 삽입 연산이 매우 빈번하게 발생한다. 또한 시간의 흐름에 따라 색인의 크기 역시 매우 증가하게 된다. 이러한 문제들은 질의 성능과 함께 이동 객체를 관리하기 위한 중요한 요소이다. 그러나 기존의 색인 기법들은 질의 성능의 개선에 많은 비중을 두고 있기 때문에, 이러한 문제에 대해서는 크게 고려하지 않고 있다.

이 장에서는 기존에 제안된 색인 방법의 고찰을 통해 이 논문에서 해결하고자 하는 문제점을 분석한다.

3.1 위치 정보의 중복 저장

기존에 연구된 대부분의 색인 방법은 이동 객체의 궤적을 표현하기 위해 객체의 이동을 line segment를 포함하는 MBB로 나타낸다. leaf node의 엔트리를 MBB로 나타낼 경우 각각의 엔트리는 궤적 생성에 필요한 line segment를 표현하기 위해 많은 정보를 중복 저장하게 된다. 그림 4는 이동 객체의 궤적을 MBB로 나타낸 그림이다. 그림에서 보는바와 같이 Trj_1, Trj_2, Trj_3은 궤적 생성을 위한 MBB를 구성하기 위해 각각 (x1,y1,t1),(x2,y2,t2),(x3,y3,t3)의 정보를 중복 저장하고 있다. 이러한 segment의 중복 저장은 leaf node의 크기를 증가시키는 원인이 되며, 이는 곧 색인의 크기가 증가하는데 결정적인 역할을 한다.

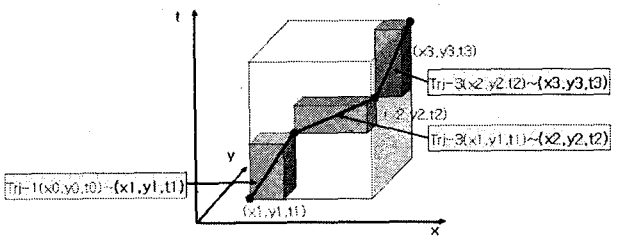

그림 4. line segment의 표현으로 인한 데이터의 중복

그림 5는 기존의 R-tree에서 각각의 레벨에 대한 node의 개수를 평가한 도표이다.

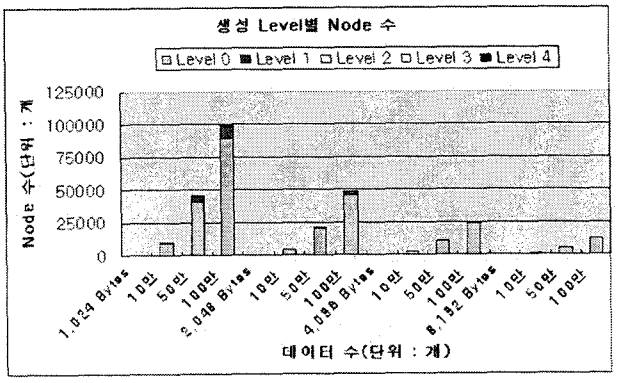

그림 5. 3DR-tree : Level별 node의 개수

그림에서 보는바와 같이 색인의 크기를 결정짓는 것은 leaf node임을 알 수 있다. 이와 같은 결과로 볼 때, 그림 4 엔트리에서 중복된 정보를 제거할 경우, 전체 색인의 크기는 거의 절반으로 감소하고, 이에 따라 디스크 공간은 매우 효율적으로 활용될 수 있을 것이다.

3.2 분할 관리되는 색인의 비효율성

이동객체를 관리하기 위한 시스템들은 대부분 객체의 현재 정보에 대한 색인과 과거 궤적에 대한 색인을 별도로 관리한다. 이러한 시스템 구조에서 이동 객체의 궤적을 처리할 경우의 예를 들면, 새로운 객체의 데이터가 삽입되면, 시스템은 다음과 같은 작업을 수행할 것이다.

1. 현재 색인 구조에서 삽입된 객체의 이전 데이터를 검색
2. 이전의 데이터와 새로운 데이터의 범위를 갖는 MBR을 구성하여 line segment를 생성
3. 이전의 데이터를 삭제하고 새로운 데이터를 삽입
4. 과거 색인 구조에 MBR로 구성된 line segment를 삽입

그러나 이러한 과정의 궤적 데이터의 처리는 새로운 데이터가 발생할 때마다 현재와 과거 색인 구조에 갱신을 반영해야 하기 때문에 많은 오버헤드가 발생한다. 따라서 시간의 흐름에 따라 연속적으로 위치를 변경하는 이동 객체를 관리하기 위해서는 단일 색인 내에서 객체의 현재 정보와 과거 정보를 지원할 수 있는 구조가 필요하다.

3.3 이동 객체의 삽입 비용

TB-tree에서 새로운 데이터의 삽입 연산이 발생할 때, 색인은 이동 객체의 궤적을 정확하게 보존하기 위해 같은 궤적을 포함하는 leaf node를 찾아야 한다. 일단, leaf node를 찾으면, 객체의 가장 최근의 정보를 포함하는 엔트리를 찾아 line segment를 구성한다. 이런 경우에, 궤적을 생성하기 위해 leaf node를 찾는데 대부분의 시간을 소비하게 된다. 새로운 객체 정보의 삽입이 매우 빈번하게 발생하는 이동 객체의 환경에서 이것은 큰 단점으로 작용한다. 따라서 객체의 삽입 연산에 소비되는 시간을 감소시키는 것은 색인의 성능에 영향을 미치는 중요한 요인이 된다.

4. 제안된 색인 기법

이 논문에서 제안하는 색인 방법은 기존의 궤적 색인 방법인 TB-tree의 구조를 이용한다. 제안된 색인 방법에서는 이동 객체의 궤적 정보를 저장하기 위한 leaf node의 구조를 변경함으로써 현재 색인을 지원하고, 전체 색인의 크기를 절반 수준으로 감소시킨다. 또한 B-tree를 사용하여 객체의 현재 정보를 포함하는 leaf node를 공유함으로써 객체의 검색 시 TB-tree의 root node를 경유하지 않고 leaf node에 빠르게 접근할 수 있다.

4.1 기본 구조

제안된 색인의 구조는 이동 객체의 과거 궤적을 보존하기 위해 기존의 TB-tree 구조와, 각각의 궤적에 대하여 가장 최근에 궤적 정보가 저장된 node를 관리하기 위한 B-tree 구조를 합친 형태를 갖는다. 그림 6은 제안된 색인의 기본 구조를 나타낸다. TB-tree는 객체의 궤적과 현재 정보를 저장하기 위해 사용되고, B-tree는 leaf node를 공유하기 위해 사용된다. 즉, 객체의 모든 위치 정보는 TB-tree에 저장되고, B-tree는 각각의 궤적별로 가장 최신의 정보가 저장된 node를 관리하며, node는 <trajectory ID, TB-tree leaf node ID> 값이 저장된다.

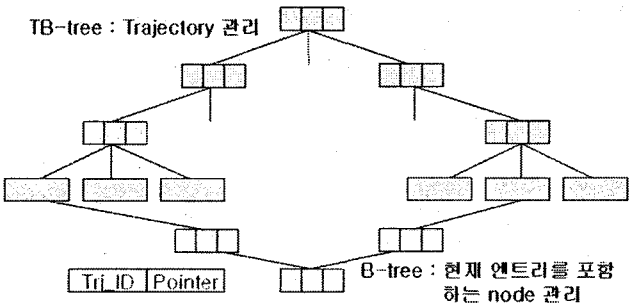

그림 6. 이동 객체 색인 구조

두 종류의 색인을 관리할 때, B-tree의 경우 trajectory ID가 dey값으로 사용되기 때문에 새로운 trajectory가 삽입될 경우에만 색인 구조가 변경되기 때문에 오버헤드가 크게 발생하지 않는다. 또한, B-tree의 사용으로 인해 발생하는 이점들과 비교할 때, 갱신으로 인한 오버헤드는 크지 않다. B-tree의 사용으로 인한 이점은 다음 절에서 구체적으로 기술한다.

4.2 Node 구조

각각의 node는 같은 궤적의 line segment를 포함하는 leaf node의 set으로 구성된다. 이때, leaf node에 저장되는 이동 객체의 정보는 MBR이 아니다. leaf node에는 각각의 timestamp에 대한 공간상의 point 데이터

만 저장을 하며 궤적에 대한 질의 처리 시 각각의 엔트리에 포함된 point 데이터를 연결하여 궤적 정보를 복원한다. 이러한 방법은 MBR로 구성되는 leaf node의 크기를 크게 감소시킬 뿐만 아니라 이동 객체의 모든 궤적에 대해 최소 비용으로 접근할 수 있다. 또한, 특정 영역에 이동 객체가 포함되었는가 또는 벗어났는가 등과 같은 위상 관련 궤적 질의를 쉽게 처리할 수 있다.

non-leaf node와 leaf node는 그림 6과 같은 구조와 같다. leaf node는 엔트리를 MBB로 표현하지 않고 공간상의 point 데이터로 저장한 후, 저장된 엔트리들을 서로 연결하여 line segment를 구성한다.

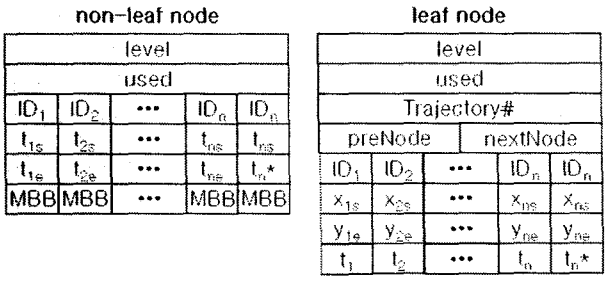

그림 7. node 구조

각각의 node의 구성에서 Trajectory#은 같은 궤적을 갖는 궤적 식별자를 의미하고, preNode와 nextNode는 같은 궤적을 포함하는 leaf node를 연결하는 pointer이다. 그리고 "*"는 객체의 현재 상태를 저장하기 위해 추가된 "now" 엔트리를 의미한다.

그림 8은 제안된 node의 구조가 TB-tree와 마찬가지로 정확하게 궤적을 계산하는 과정을 나타낸다.

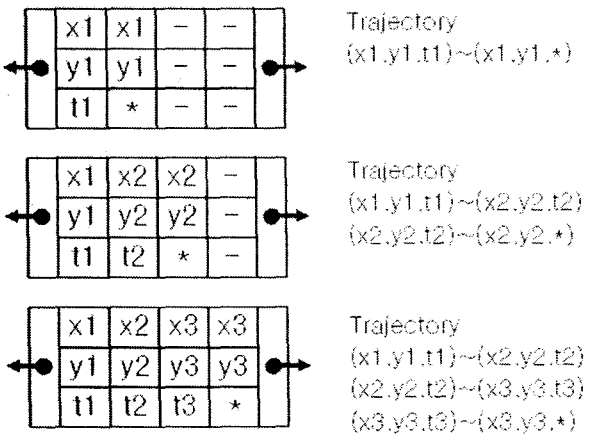

그림 8. 객체 삽입에 따른 궤적 정보

그림을 보면 node에 객체의 공간상의 point는 시간 경과에 따라 순차적으로 저장된다. 객체의 현재 정보를 나타내기 위해 "*" 상태가 추가되는 과정을 보여준다. 이 때 leaf node의 "*"는 상위 node로 전파된다. 만약, node가 가득 차면 leaf node는 분할되어야 한다. 이럴 경우, TB-tree와 마찬가지로 새로운 node를 생성하여 분할하게 되며, 여기서 분할되는 node의 가장 마지막 엔트리는 새롭게 추가되는 node의 첫 번째 엔트리로 복사된다. 이럴 경우, 궤적 정보의 중복 문제가 발생한다. 그러나 이것은 추가적인 디스크 I/O 없이 정확한 궤적을 생성할 수 있다는 장점을 갖는다.

그림 9는 leaf node에 분할이 발생한 경우에 새로운 node를 추가하는 과정을 나타낸다.

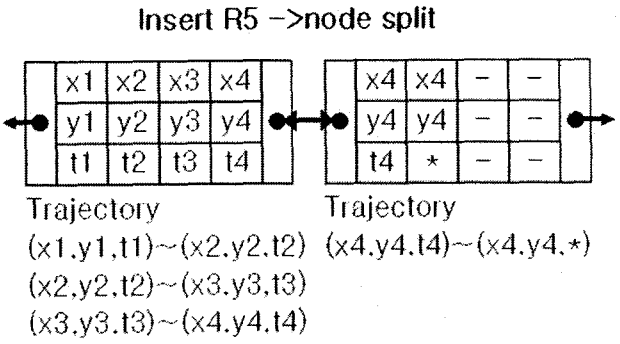

그림 9. leaf node 분할

4.3 삽입 정책

새로운 데이터의 삽입할 경우에는 현재 정보를 저장하고 있는 B-tree를 사용한다. TB-tree를 사용하여 새로운 데이터를 삽입할 때는 이전의 궤적 정보가 저장된 leaf node를 검색하기 위하여 overlap 질의를 수행한다. 이러한 과정은 많은 node를 검색해야 하기 때문에 새로운 객체의 삽입 시간이 증가하는 원인이 된다. B-tree의 node는 객체 식별자와 객체의 가장 최근의 정보를 포함하는 leaf node pointer로 구성된다.

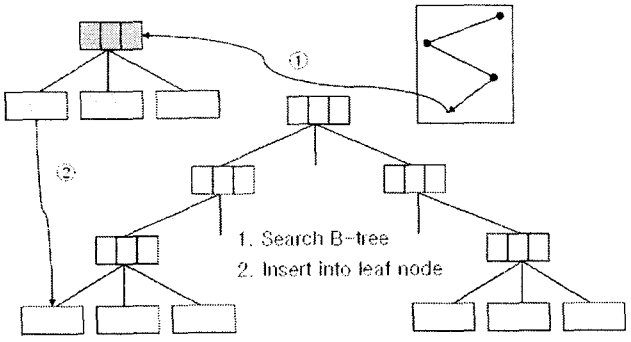

그림 10. B-tree를 사용한 객체의 삽입

B-tree는 시간이 흐름에 따라 이동 객체의 정보가 증가하더라도 B-tree는 이동 객체의 개수만큼의 범위를

가지기 때문에 항상 일정한 크기를 유지할 수 있다. 또한 B-tree를 사용하면 객체 식별자를 포함하는 질의를 처리할 수 있다. 그리고 B-tree는 새로운 객체가 삽입될 때, 객체의 가장 최근의 정보가 포함된 leaf node에 직접 접근할 수 있기 때문에 삽입 비용을 감소시킬 수 있다. 그림 10은 새로운 객체의 데이터를 삽입하는 과정을 나타낸다. 삽입이 발생하면 먼저, B-tree를 통해 삽입할 데이터의 가장 최근의 정보를 포함하는 node를 검색하여 해당하는 위치에 엔트리를 삽입한다.

표 1은 자세한 삽입 알고리즘을 나타낸다.

표 1. 삽입 알고리즘

```
Algorithm Insert(newEntry)
INS1
  Find leaf node in B-tree
INS1
  If leaf node N is found,
     If leaf node already full
     Find non-full parent node
     Find right-most path N' from found non-full parent
     node until(Leaf level+1)
     Create new leaf node and insert new line segment
     Insert new line segment into node N'
     Update B-tree using new leaf node
     If necessary, split or Adjust tree
     Update linked list
  Else
     Insert new line segment into found leaf node N
     If necessary, split or Adjust tree
  Else
     Find right-most path N' from root node until
     (Leaf level+1)
     Create new leaf node for new line segment
     Insert new line segment into node N'
     Insert new trajectory indentifier and leaf node into
     B-tree
```

5. 결론

최근에 이동 통신 기술과 GPS 기술의 발달과 함께 실시간으로 이동하는 차량이나 비행기, 선박 등과 같은 이동 객체의 위치 정보 서비스에 대한 요구가 증대되고 있다.

이 논문에서는 실시간으로 변화하는 이동 객체를 효율적으로 저장하고 검색하기 위한 색인 구조를 제안하였다. 제안된 색인 구조 이동 객체의 현재 위치 정보뿐만 아니라 과거 궤적 정보까지도 효율적으로 접근할 수 있다. 특히, leaf node의 크기를 줄임으로써 전체 색인의 크기를 현저히 감소시키는 방법을 제안하였다. 그리고 단일 색인 내에서 현재 색인과 과거 색인을 통합 관리하는 구조를 제안하였다. 이러한 구조를 통해 이동 객체의 현재 위치 정보에 빠르게 접근할 수 있고, 새로운 객체의 삽입 비용을 감소시킬 수 있다.

향후 연구로는 제안된 색인 구조의 구현과 다른 색인 방법들과의 비교를 통한 성능 평가가 필요하다.

참고 문헌

[1] R. H. Guting, M. F. Bhlen, 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, pp.1-42, 2000.

[2] 신기수, 안윤애, 배종철, 정영진, 류근호, "GIS를 이용한 시공간 이동 객체 관리 시스템", 한국정보처리학회 논문지, 제8-D권, 2호, pp.105-116, 2001년 4월.

[3] S. Saltenis, C. S. Jensen, S. T. Leutenegger, M. A. Lopez, "Indexing the Positions of Continuously Moving Objects," proc. of the 2000 ACM SIGMOD Conference on Management of Data, Dallas, Texas, USA, 2000

[4] G. Kollios, D. Gunopulos, V. J. Tsotras, "On Indexing Mobile Objects," Proc. Of the 18th ACM Symposium on Principles of Database Systems, 1999

[5] Y. Theodoridis, M. Vazirgiannis, T. K. Sellis, "Spatio-Temporal Composition and Indexing for Large Multimedia Applications", Multimedia Systems 6, 4, pp.284-298, 1998.

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

[7] D. Pfoser, C. S. Jensen, Y. Theodoridis, "Novel Approaches to the Indexing of Moving Object Trajectories," Proc. of the 26th Conference on VLDB, Cairo, Egypt, 2000