위치 기반 서비스에서 이동 객체의 궤적을 위한 TB-트리의 확장

심춘보*, 강홍민⁰, 엄정호, 장재우
부산가톨릭대학교 컴퓨터정보공학부*, 전북대학교 컴퓨터공학과,
cbsim@cup.ac.kr*, {hmkang⁰, jhum, jwchang}@dblab.chonbuk.ac.kr

Extension of TB-Tree for Trajectory of Moving Objects in Location-Based Services

Choon-Bo Shim*, Hong-Min Kang⁰, Jung-Ho Um, Jae-Woo Chang
School of Computer Information Engineering, Catholic University of Pusan*
Dept. of Computer Engineering, Chonbuk National University

요 약

시간의 흐름에 따라 그 위치가 빈번히 변화하는 이동 객체의 특성상 실시간으로 증가하는 이동 객체의 궤적 정보를 효과적으로 관리할 수 있는 효율적인 색인 기법이 요구된다. 따라서 본 논문에서는 이동 객체의 궤적을 색인하기 위해 기존에 제안되었던 TB 트리의 성능을 개선시킬 수 있는 확장된 TB-트리(Extented TB-Tree:ETB-Tree)를 제안한다. 기존의 TB 트리는 이동 객체의 궤적 세그먼트를 삽입할 때마다 선행 세그먼트를 가지고 있는 리프 노드를 찾기 위해 루트 노드부터 리프 노드까지 순회해야만 하기 때문에 불필요한 노드 접근으로 인한 오버헤드가 발생한다. 이를 위해 ETB 트리는 선행 노드를 직접적으로 접근하기 위해 이동 객체의 처음 세그먼트와 마지막 세그먼트가 저장된 리프 노드를 가리키는 포인터 정보와 더불어 디스크에서의 페이지를 가리키는 페이지 번호를 별도의 테이블에 같이 유지한다. 따라서, 저장시 동일한 이동 객체의 선행노드를 빨리 검색할 수 있고, 궤적 질의시 직접적으로 디스크에 접근해 해당 객체의 궤적들을 검색함으로써 검색 성능을 향상시킬 수 있다. 아울러 ETB 트리는 새로운 이동 객체의 궤적 정보가 삽입될 때마다 메모리 상의 트리뿐만 아니라 디스크에 반영함으로써 트리의 일관성을 유지한다.

1. 서론

최근 GPS 및 무선 데이터 전송능력이 있는 휴대용 단말기의 등장과 이동 무선 컴퓨팅 기술의 발달로 인해 위치기반 서비스(Location-Based Services: LBS)가 무선 인터넷 시장에서 중요한 이슈로 대두되고 있다. LBS란 이동통신망을 기반으로 사람이나 사물의 위치를 정확하게 파악하고 이를 활용하는 응용시스템 및 서비스를 말한다. LBS가 보편화 되어감에 따라 다양한 서비스들이 일정한 장소에서 뿐만 아니라, 휴대용 전화기, PDA, 노트북과 같은 이동성을 지닌 단말기를 이용하여 이동 중에도 지속된다. 따라서 시간의 흐름에 따라 객체가 이동하면서 그 위치를 연속적으로 변경하는 특징을 지닌 이동 객체를 효율적으로 저장 및 관리할 수 있는 색인 기술이 요구된다.

따라서 본 논문에서는 이동 객체의 궤적을 색인하기 위해 기존에 제안되었던 TB 트리(Trajectory-Bundle Tree)의 성능을 개선시킬 수 있는 확장된 TB-트리(Extented TB-Tree:ETB-Tree)를 제안한다. 기존의 TB 트리는 이동 객체의 궤적 세그먼트를 삽입할 때마다 선행 세그먼트를 가지고 있는 리프 노드를 찾기 위해 루트 노드부터 리프 노드까지 순회해야만 하기 때문에 불필요한 노드 접근으로 인한 오버헤드가 발생한다. 이를 위해 ETB 트리는 선행 노드를 직접적으로 접근하기 위해 이동 객체의 처음 세그먼트와 마지막 세그먼트가 저장된 리프 노드를 가리키는 포인터 정보와 더불어 디스크에서의 페이지를 가리키는 페이지 번호를 별도의 테이블에 같이 유지한다. 따라서, 저장시 동일한 이동 객체의 선행노드를 빨리 검색할 수 있고, 궤적 질의시에도 직접 디스크에 접근해 해당 객체의 궤적들을 검색함으로써 검색 성능을 향상시킬 수 있다. 아울러 ETB 트리는 새로운 이동 객체의 궤적 정보가 삽입될 때마다 메모리 상의 트리 구조 뿐만 아니라 디스크에 반영함으로써 트리의 일관성을 유지한다.

본 논문의 구성은 다음과 같다. 2장에서는 이동 객체를 위한 색인 기법과 관련된 기존의 연구들을 살펴보고 3장에서는 기존 TB 트리를 확장한 ETB 트리에 대해서 설명한다. 4장에서는 기존의 R* 트리와 TB 트리 그리고 제안하는 ETB 트리와의 성능평가를 기술하고 마지막으로 5장에서는 결론 및 향후 연구를 제시한다.

2. 관련 연구

이동 객체의 위치 정보를 색인하기 위한 기존의 연구는 크게 세 가지로 분류된다. 먼저 이동 객체의 현재 및 과거 정보를 효율적으로 처리하기 위한 시공간 접근 기법[1]과 예측된 미래 시간을 처리하기 위한 시공간 접근 기법[2] 마지막으로 이동 객체의 궤적 기반 질의를 효과적으로 처리하기 위한 시공간 접근 기법이다. 그 중에서도 궤적 기반 질의를 위한 시공간 접근 기법은 본 논문과 매우 밀접한 관련이 있으며 주로 STR 트리[3]와 TB 트리[4]가 여기에 속한다. 이들 연구는 시간에 따라 연속적으로 변화하는 위치 정보를 표현하기 위해, 이동 객체의 궤적을 라인 세그먼트(line segment)로 저장하는 방법이다. 여기서 TB 트리는 동일한 이동 객체에 속하는 라인 세그먼트들을 같은 리프 노드에 삽입하여 유지하는 궤적 보존 전략(Trajectory Preservation Strategy)을 채택함으로써 저장 공간의 오버헤드가 없으며, 아울러 이동 객체의 궤적 질의에 매우 탁월한 성능을 보인다. 반면에 TB 트리는 이동 객체의 새로운 라인 세그먼트를 삽입할 때마다 궤적 보존 전략의 특성상 동일한 객체의 선행 라인 세그먼트가 삽입된 리프 노드를 찾아야 한다. 따라서 루트 노드에서부터 리프 노드까지 트리를 순회하여야 한다. 이는 삽입시 불필요한 많은 노드를 접근해야 하는 문제점을 가지고 있다. 특히 시간이 지남에 따라 계속해서 증가하는 이동 객체의 특성상 트리의 크기가 커질수록 삽입될 객체의 선

본 연구는 한국과학재단 2003년 전반기 신진연구자연수지원사업(M02-2003-000-10317-0)의 지원을 받았음.

행 노드를 검색하는데 많은 비용이 든다.

3. 제안하는 ETB 트리

3.1 전체 구조

본 논문에서는 이동 객체의 궤적을 색인하기 위해 기존에 제안되었던 TB 트리의 문제점 즉, 이동 객체의 궤적을 구성하는 라인 세그먼트를 삽입할 때의 삽입 오버헤드를 해결하고 저장 공간 측면과 궤적 질의 처리의 효율성을 높이기 위한 확장된 TB-트리인 ETB 트리를 제안한다. 제안하는 ETB 트리의 전체 구조는 그림 1과 같다. ETB 트리의 구조는 기존의 TB 트리와 유사하며, 선행 노드를 직접적으로 접근하기 위해 이동 객체의 처음 세그먼트와 마지막 세그먼트가 저장된 리프 노드를 가리키는 포인터 정보와 더불어 디스크에서의 페이지를 가리키는 페이지 번호를 유지하는 테이블이 존재한다. 테이블 구조는 헤더 정보와 리프 노드에 대한 포인터 정보와 디스크의 페이지를 가리키는 레코드 정보로 구성된다. 테이블 헤더는 total_rec과 id_len으로 구성된다. 먼저 total_rec은 저장된 전체 이동 객체의 수를 의미하며, id_len은 이동 객체의 부가적인 정보를 가리키는 식별자를 위한 식별자 길이 정보를 나타낸다. 또한 테이블의 각 레코드에는 moid, s_ptr, s_pid, e_ptr, 그리고 e_pid를 저장하며, 각각은 이동 객체의 식별자, 이동 객체의 첫 번째 라인 세그먼트를 가지고 있는 리프 노드의 메모리 포인터와 디스크 상에서의 페이지 번호, 이동 객체의 마지막 라인 세그먼트를 가지고 있는 리프 노드의 메모리 포인터와 디스크 상에서의 페이지 번호를 나타낸다.

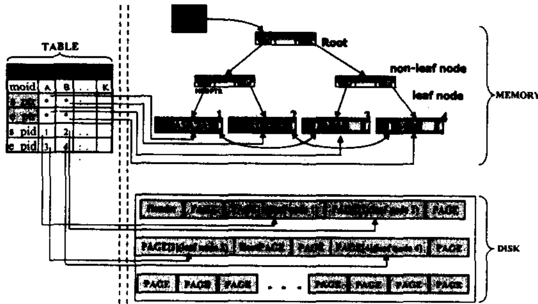

그림 1. ETB 트리의 전체 구조

따라서 ETB 트리는 이동 객체의 궤적에 대한 라인 세그먼트를 삽입할 때 기존의 TB 트리처럼 트리의 루트 노드에서부터 리프 노드까지 순회하지 않고 테이블에서 해당 객체의 마지막 라인 세그먼트를 포함하고 있는 리프 노드를 가리키는 포인터(e_ptr)와 디스크 상에서의 페이지 번호(e_pid)를 이용하여 한 번에 직접적으로 리프 노드에 접근함으로써 삽입 시간을 현저히 줄일 수 있다. 아울러 불필요한 노드 접근을 제거해 삽입 성능을 향상시킬 수 있다. 또한 이동 객체의 궤적 질의 처리시 해당 객체의 선행 궤적 정보를 페이지 번호를 통해 직접 디스크로부터 읽기 때문에 페이지 접근 횟수를 줄일 수 있다. 디스크 기반의 구조를 가지고 있어 새로운 이동 객체의 라인 세그먼트를 삽입시 해당 객체의 궤적을 바로 디스크에 반영해 메모리 상의 트리 구조와 디스크 상의 트리 구조 사이의 일관성을 유지한다. 이를 통해 메모리 상에 존재하는 ETB 트리의 문제가 발생했을 때 디스크로부터 트리 정보를 다시 로딩하여 트리를 재구성할 수 있다.

3.2 알고리즘

ETB 트리의 삽입 알고리즘은 기존의 TB 트리의 삽입 알고리즘과 유사하다. 단지 삽입시 테이블 정보를 이용해서 이동 객체의 선행 라인 세그먼트를 찾고, 아울러 새로운 리프 노드가 생성되면 테이블 정보를 갱신한다. 이동 객체의 라인 세그먼트를 삽입하기 위해, 그림 2의 ETB 트리 삽입 알고리즘에서와 같이 ETB_Insert() 함수를 호출한다. 여기서는 새로 삽입될 라인 세그먼트의 선행 라인 세그먼트가 저장되어 있는 리프 노드가 존재하고 아울러 여유 공간이 있다면, 그 리프 노드에 삽입하고 종료한다. 만약에 여유 공간이 없다면 새로운 리프 노드를 생성하고 생성된 리프 노드에 삽입하고 종료한다. 만약 선행 라인 세그먼트의 리프 노드가 존재하지 않으면, 새로운 이동 객체의 첫 번째 라인 세그먼트로 간주하고 새로운 리프 노드를 생성해서 저장하고 종료한다.

```
Algorithm ETB_Insert(L, MOID)
Input
  L : 새로 삽입될 라인 세그먼트(Line segment)
  MOID : 라인 세그먼트 L이 속해 있는 이동 객체의 식별자
Output : 성공 혹은 실패

1 삽입될 라인 세그먼트 L의 선행 세그먼트가 저장된 리프 노드 N'를
찾기 위해 ETB_FindLeafNode(MOID, Link_Type) 함수 호출
2 만약 리프 노드 N'가 발견되면
  2.1 리프 노드 N'에 여유 공간이 있다면
    2.1.1 라인 세그먼트 L을 리프 노드 N'에 삽입
  2.2 여유 공간이 없다면 즉, 가득 차 있으면
    2.2.1 새로운 리프 노드를 생성하기 위해
          ETB_MakeNewNode(L, MOID, N') 함수를 호출
3 리프 노드 N'가 발견되지 않으면
  3.1 기존의 삽입된 선행 라인 세그먼트가 없기 때문에 즉, 처음
      삽입된 이동 객체의 라인 세그먼트이므로 새로운 리프 노드를
      생성하기 위해 ETB_MakeNewNode(L, MOID, N') 함수를 호
      출
```

그림 2. ETB 트리의 ETB_Insert() 함수

ETB_FindLeafNode()는 그림 3에서와 같이 해당 MOID와 첫 번째 라인 세그먼트의 리프 노드인지 또는 마지막 라인 세그먼트의 리프 노드인지를 가리키는 LinkType을 가진다. 이를 통해 테이블 정보를 참조해서 MOID의 s_ptr 즉, 첫 번째 라인 세그먼트의 리프 노드 또는 마지막 라인 세그먼트의 리프 노드를 반환한다.

```
Algorithm ETB_FindLeafNode(MOID, LinkType)
Input
  MOID : 라인 세그먼트 L이 속해 있는 이동 객체의 식별자
  LinkType : First 혹은 Last값을 가지며 해당 MOID의 첫 번째
             리프 노드를 찾을 것인지 아니면 마지막 리프 노드를
             찾을 것인지를 나타내는 flag
Output : 검색한 리프 노드 혹은 Null 값을 반환

1 MOID를 이용해서 테이블 정보를 검색
2 해당 MOID가 테이블 존재하면
  2.1 LinkType이 First이면
    2.1.1 첫 번째 라인 세그먼트를 가리키는 s_ptr를 반환
  2.2 LinkType이 Last이면
    2.2.1 마지막 라인 세그먼트를 가리키는 e_ptr를 반환
3 테이블에 존재하지 않으면
  3.1 Null 반환
```

그림 3. ETB 트리의 ETB_FindLeafNode() 함수

ETB_MakeNewNode()에서는 그림 4에서와 같이 삽입하려고 하는 이동 객체의 라인 세그먼트가 이전에 존재하는 경우, 새로운 리프 노드를 생성하고 선행 노드와 새로운 리프 노드를 연결 리스트로 링크시키고 테이블 정보를 갱신한다. 반면에 처음 삽입되는 이동 객체의 라인 세그먼트일 경우는 새로운 리프 노드를 생성하고 리프 노드를 테이블의 s_ptr로 설정한다. 아울러 디스크에도 생성된 리프 노드의 페이지를 생성하고 정보를 반영시킨다. 그리고 마지막으로 생성된 리프 노드의 부모 노드에서부터 시작하여 필요하면 루트 노드까지 올라가면서 해당 MBR 정보를 갱신한다.

```
Algorithm ETB_MakeNewNode(L, MOID, N)
Input
  L : 새로 삽입될 라인 세그먼트(Line segment)
  MOID : 라인 세그먼트 L이 속해 있는 이동 객체의 식별자
  N : 라인 세그먼트 L의 선행 라인 세그먼트가 저장된 리프 노드
```

```
Output : 성공 혹은 실패

1 라인 세그먼트 L이 저장될 새로운 리프 노드 N' 생성
2 선행 노드 N이 Null아니면
  2.1 N'를 N의 다음 노드로 연결(연결 리스트)
  2.2 마지막 노드를 N에서 N'로 테이블 정보 갱신
  2.3 N'를 위한 디스크의 페이지 생성과 갱신
3 N이 Null이면
  3.1 N'를 테이블의 s_ptr로 갱신
4 상위에 있는 부모에서부터 시작하여 필요하면 루트까지 MBR 정보
  갱신
```

그림 4. ETB 트리의 ETB_MakeNewNode() 함수

4. 실험 환경 및 성능 평가

본 논문에서는 기존의 TB 트리와 제안하는 ETB 트리를 구현하여 삽입 성능과 궤적 질의에 대해서 성능 평가를 수행하였으며 수행 환경은 *Pentium-Ⅳ 1.7GHz CPU*와 *메인 메모리 512MB*, 윈도우 2000, 그리고 C++ 언어를 이용하였다. 성능 평가에 사용한 실험 데이터는 GSTD 알고리즘[5]을 이용하여 표 1과 같이 세 가지 종류의 데이터를 생성하였다. 성능 평가는 기존의 R^* 트리[6], TB 트리, 그리고 ETB 트리에 대해서 삽입 성능과 검색 성능을 시간과 페이지 접근 횟수 측면을 고려하여 수행하였다.

표 1. 실험 데이터의 종류

데이터 종류 / 파라미터	데이터 1 (D1)	데이터 2 (D2)	데이터 3 (D3)
전체 라인 세그먼트의 수	200,000	200,000	200,000
이동 객체의 수	100	2,000	20,000
객체당 라인 세그먼트의 수	2,000	100	10

그림 5는 삽입 성능에 대한 성능 평가 결과이다. 그래프에서와 같이 R^* 트리가 가장 많은 시간이 소요되며 이는 R^* 트리의 특성상 가까운 영역에 있는 라인 세그먼트들을 같은 페이지를 삽입하기 위해 많은 페이지 분할이 발생하기 때문이다. TB 트리와 ETB 트리에 대해서는 실험 데이터 1이나 2일 때는 거의 성능이 비슷하며 실험 데이터 3(D3)일 때 ETB 트리가 기존의 TB 트리보다 더 나은 성능을 보인다. 이유는 기존의 TB 트리는 전체 트리를 순회하는 횟수가 이동 객체의 수에 비례하기 때문에 ETB 트리보다 성능이 떨어진다. 따라서 ETB 트리가 TB 트리에 비해 최대 600% 정도의 성능 향상을 보인다.

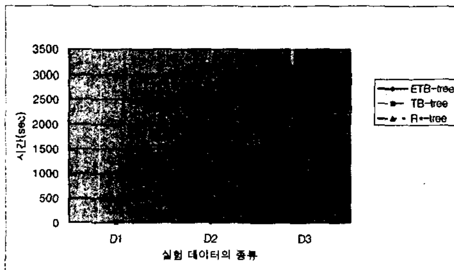

그림 5. 삽입 성능 평가 결과

그림 6은 궤적 질의에 대한 검색 성능을 디스크 페이지 접근 횟수로 나타낸 것이다. 삽입과 마찬가지로 R^* 트리가 가장 많은 시간이 소요되며 이는 R^* 트리의 특성상 동일한 이동 객체의 궤적을 검색하기 위해서는 각 라인 세그먼트마다 전체 트리를 모두 순회해야 하기 때문이다. TB 트리와 ETB 트리에 대해서는 실험 데이터 1(D1)일 때 거의 비슷하며 실험 데이터 2와 3일 때는 ETB 트리가 기존의 TB 트리보다 더 나은 성능을 보인다. 이유는 이동 객체의 수가 많을수록 즉, 객체당 라인 세그먼트의 수가 적을수록 ETB 트리는 단지 테이블의 크기가 커질 뿐 테이블에서 해당 MOID를 검색한 후 적은 수의 페이지를 접근해서 궤적 정보를 검색할 수 있기 때문이다. 반면에 객체당 라인 세그먼트의 수가 커질수록 연결 리스트로 링크되어 있는 리프 노드들을 모두 접근해야만 한다. 따라서 성능 평가 결과 최대 약 1000% 정도의 성능 향상을 보인다.

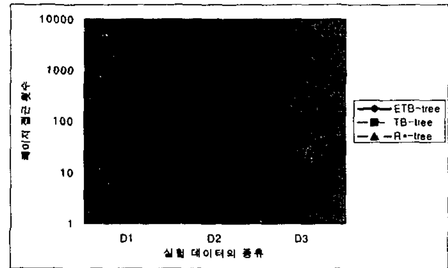

그림 6. 궤적 질의에 대한 검색 성능 평가 결과

5. 결론

본 논문에서는 이동 객체의 궤적을 색인하기 위해 기존에 제안되었던 TB 트리의 성능을 개선시킬 수 있는 확장된 TB-트리인 ETB 트리를 제안하였다. 기존의 TB 트리는 이동 객체의 궤적 세그먼트를 삽입할 때마다 선행 세그먼트를 가지고 있는 리프 노드를 찾기 위해 루트 노드부터 리프 노드까지 순회해야만 하기 때문에 불필요한 노드 접근으로 인한 오버헤드가 발생한다. 따라서 선행 노드를 직접적으로 접근하기 위해 이동 객체의 처음 세그먼트와 마지막 세그먼트가 저장된 리프 노드를 가리키는 포인터 정보 및 디스크에서의 페이지를 가리키는 페이지 번호를 별도의 테이블에 유지함으로써, 저장시 동일한 이동 객체의 선행노드를 빨리 검색할 수 있고, 궤적 질의시 직접 디스크에 접근해 해당 객체의 궤적들을 검색함으로써 검색 성능을 향상시킨다. 성능 평가 결과 삽입 성능은 거의 5-6배 정도의 성능 향상을 보이며, 궤적 검색 질의에 대한 성능 평가는 약 10배 정도의 성능 향상을 나타낸다. 앞으로의 향후 연구로는 실제 리얼 데이터를 이용하여 다양한 검색 질의 측면에서의 성능 평가를 수행하는 것이다.

[참고문헌]

[1] Y. Theodoridis, M. Vazirgiannis, T. Sellis, "Spatio-Temporal indexing for Large Multimedia Applications", Proc. of the 3rd IEEE Conf. On Multimedia Computing and Systems(ICMCS), Hiroshima, Japan, June 1996.

[2] S. Saltenis, C. S. Jensen, S.T. Leutenegger, and M. A. Lopez, "Indexing the Positions of Continuously Moving Objects", Proc. of ACM SIGMOD on Management of data, pp 331-342, 2000.

[3] D. Pfoser, Y. Theodoridis, and C.S. Jensen, "Indexing Trajectories of Moving Point Object ", Chorochronos Technical Report, Ch-99-3, October, 1999.

[4] D. Pfoser,C.S.Jensen, Y. Theodoridis, "Novel Approaches in Query Processing for Moving Objects", Proc. of the 26th VLDB Conf., Cairo, Egypt, pp 395-406 2000.

[5] Yannis Theodoridis, JeffersonR.O Silva,Mario A.Naschimento, "On the Generation of Spatiotemporal Datasets", *Chorochronos Technical Report*, Ch-99-01, January 1999.

[6] N. Beckmann and H. P. Kriegel, "The R*-tree:An Efficient and Robust Access Method for Points and Rectangles", In Proc. ACM SIGMOD, pp 322-331, 1990.