LBS 응용에서 이동 객체의 궤적 색인을 위한 직접 테이블 기반의 확장된 TB-트리의 구현

Implementation of Extended TB-Trees Based on Direct Table for Indexing Trajectories of Moving Objects in LBS Applications

심춘보*, 신용원**, 박병래***
순천대학교 정보통신공학부*, 부산가톨릭대학교 병원경영학과**, 부산가톨릭대학교 방사선학과***
Choon-Bo Shim(cbsim@sunchon.ac.kr)*, Yong-Won Shin(kevin@cup.ac.kr)**,
Byung-Rae Park(brpark@cup.ac.kr)***

요약

본 논문에서는 위치 기반 서비스에서 이동 객체의 궤적을 색인하기 위해 기존에 제안되었던 TB-트리의 성능을 개선시킬 수 있는 확장된 TB-트리(Extented TB-Tree:ETB-Tree)를 제안한다. ETB-트리는 선행 노드를 직접적으로 접근하기 위해 이동 객체의 처음 세그먼트와 마지막 세그먼트가 저장된 단말 노드를 가리키는 포인터 정보와 더불어 디스크에서의 페이지를 가리키는 페이지 번호를 별도의 테이블에 같이 유지함으로써 저장시 동일한 이동 객체의 선행노드를 빨리 검색할 수 있고, 궤적 질의시에도 바로 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. 마지막으로, 성능 평가를 분석한 결과, 제안하는 색인 기법이 기존의 색인 기법들에 비해 삽입과 궤적 질의의 검색 측면에서 더 우수함을 보임을 알 수 있다.

■ 중심어 : | 위치기반서비스 | 이동 객체 | 확장된 TB-트리 |

Abstract

In this paper, we propose an extended TB-tree, called ETB-tree, which can improve the performance of an existing TB-tree proposed for indexing the trajectories of moving objects in Location-Based Service(LBS). The proposed ETB-tree directly accesses the preceding node by maintaining a direct table, called D-Table which contains the page number in disk and memory pointers pointing the leaf node with the first and last lines segment of moving objects. It can improve the insertion performance by quick searching the preceding node of a moving object and retrieval performance owing to accessing directly the corresponding trajectories in disk for the trajectory-based query. In addition, the ETB-tree provides consistency of a tree by reflecting a newly inserted line segment to the tree both in memory and disk. The experimental results show that the proposed indexing technique gains better performance than other traditional ones with respect to the insertion and retrieval of a trajectory query.

■ Keyword : | Location-Based Service | Moving Objects | Extented TB-Tree |

접수번호 : #041011-002
접수일자 : 2004년 10월 11일
심사완료일 : 2005년 02월 28일
교신저자 : 심춘보 e-mail : cbsim@sunchon.ac.kr

I. 서 론

위치기반 서비스(Location-Based Services: LBS)란 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. 이동 객체의 색인 기술에 관련된 연구로는 현재와 미래 위치를 검색하기 위한 색인 기술[1-3], 과거 위치를 검색하기 위한 색인 기술[4-6], 그리고 이동 객체의 궤적을 효율적으로 검색하기 위한 궤적 색인 기술[9-9]로 나뉜다. 특히 이동 객체의 궤적을 위한 색인 기술로는 R*-트리[7], STR-트리[8], TB-트리[9] 등이 있으며 이 중에서 현재로는 TB-트리가 가장 성능이 우수한 것으로 알려져 있다.

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 단말 노드를 찾기 위해 루트 노드부터 단말 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 단말 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 질의 시에도 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. 또한 트리의 크기가 커질수록 시스템의 가용메모리의 제한으로 인해 성능에 영향을 미칠 수 있기 때문에 일정한 버퍼를 유지함으로써 해당 버퍼를 교체해 나가는 버퍼링 정책을 사용해 가용 메모리를 효율적으로 사용할 수 있다.

본 논문의 구성은 다음과 같다. 제2장에서는 이동 객체의 색인과 관련된 기존의 연구들을 소개하고 제3장에서는 제안하는 ETB-트리의 전체 구조와 알고리즘에 대해서 설명한다. 제4장에서는 타 연구와 제안하는 ETB-트리와의 성능평가를 기술하고 마지막으로 5장에서는 결론을 맺는다.

II. 연구방법

본 장에서는 이동 객체의 궤적과 관련이 있는 기존의 색인 기법들 즉, R*-트리, STR-트리, TB-트리, LUR-트리를 간략히 소개한다.

첫째, R*-트리[7]는 기존에 제안된 R-트리[11]의 변형된 트리로서 기본적인 구조와 연산은 기존의 R-트리와 거의 동일하며, 삽입이나 검색 성능을 개선하여 이동 객체를 위한 다양한 색인 기법들이 제안되기 전에 많이 사용되었던 색인 기법이다. R*-트리는 기존의 R-트리가 데이타를 삽입할 때 최소 영역 증가 정책(least area enlargement policy)을 사용하여 영역 정보만을 고려했던 부분을 개선하여 중복(over-lap)과 가장자리(margin) 정보를 추가적으로 고려함으로써 삽입 성능을 향상시켰을 뿐만 아니라, 재 삽입 정책을 이용하여 R-트리의 공간활용도 부분을 향상시킨 색인구조이다.

둘째, STR-트리[8]는 시공간 범위 질의 등과 같은 좌표 범위 질의를 위해 R-트리를 3차원으로 확장한 3D R-트리에 이동 객체의 궤적을 처리하기 위하여 알고리즘을 수정한 것이다. 단순히 R-트리의 단말 노드 구조만 바뀔 경우 공간 낭비가 증가하고 궤적이 보호되지 않는 문제가 발생하기 때문에, R-트리의 삽입과 분할 알고리즘을 수정한 방법이다. STR-트리는 보존 파라미터(preservation parameter)를 사용하여 같은 객체의 궤적을 근접한 페이지에 저장하도록 하였다. STR-트리는 공간적인 특성을 고려할 뿐 아니라 부분적인 궤

적을 보전하려 한다. 즉, 동일한 이동체의 궤적을 최대한 근접하게 만든다. STR-트리는 저장되는 객체의 수가 많아지면, 성능이 저하되고, 궤적에 관련된 질의를 제외하면, R*-트리보다 성능이 떨어진다.

셋째, TB-트리[9]는 극단적으로 이동 객체의 궤적에만 초점을 맞추어 제안된 색인 기법이다. 즉, 시간에 따라 연속적으로 변화하는 위치 정보를 표현하기 위해 이동 객체의 궤적을 라인 세그먼트(line segment)로 저장하는 방법이다. 동일한 이동 객체에 속하는 라인 세그먼트들을 같은 단말 노드에 삽입하여 유지하는 궤적 보존 전략(trajectory preservation strategy)을 채택하고 있다. 따라서 공간적으로 근접하더라도, 같은 궤적의 선분이 아니면, 다른 노드에 저장된다. 효율적인 궤적 검색을 위해, 동일한 이동 체의 궤적을 저장하고 있는 단말노드끼리는 링크로 연결되어 있어 임의의 노드로부터 해당하는 전체 궤적을 검색할 수 있다. TB-트리는 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 불필요한 많은 노드를 접근해야 하는 문제점을 가지고 있다.

마지막으로, LUR-트리[2]는 공간 색인을 위해 제안되었던 기존의 R-트리를 기반으로 갱신 비용(update cost)을 줄이고 현재 위치 정보를 색인하기 위해 제안되었던 기법이다. 기존 방법에서는 삭제 및 삽입 연산으로 효율적 검색이 가능한 색인을 구성할 수 있으나, 갱신 연산은 노드의 분할 또는 합병을 유발하기 때문에 갱신 비용이 높다. Lazy 갱신은 이러한 문제점을 해결하기 위해 이동 객체의 위치 변경으로 발생하는 갱신 연산을 노드 내에서의 갱신으로 제한함으로써 노드의 구조 변경에 따른 비용을 감소시켰다. LUR-트리는 EMBR(Extended MBR)을 이용해 갱신 비용을 줄여 성능 향상을 꾀하고 있으며, 기존의 R-트리가 지원했던 영역질의, k-NN 질의, 공간 조인 질의를 모두 지원한다. 또한 LUR-트리는 구조적으로 DirectLink라고 하는 별도의 인덱스 구조를 둠으로써 해당 oid를 가지고 있는 단말 노드를 빠르게 찾을 수 있는 형태를 띠고 있다.

III. 직접 테이블 기반의 확장된 TB-트리

1. 문제 정의

기존의 궤적-기반 질의처리를 위해 제안되었던 TB-트리의 문제점을 정리하면 다음과 같다.

- TB-트리는 새로운 라인 세그먼트를 삽입할 때 궤적 보전 전략의 특성상 선행 라인 세그먼트가 삽입된 단말 노드를 검색해야 하는 오버헤드가 있다.
- TB-트리는 시간이 지남에 따라 계속해서 증가하는 이동 객체의 특성으로 인해 색인의 크기가 증가할수록 삽입될 객체의 선행 노드를 검색하는 데 드는 비용 부담이 크다.
- TB-트리는 궤적-기반 질의처리 시에 해당 궤적을 검색하기 위해 적어도 한 번은 트리의 루트에서부터 단말 노드까지 트리를 순회해야하는 문제점이 있다.
- TB-트리는 디스크 기반의 색인구조가 아니기 때문에 색인의 크기가 증가해 색인 전체가 메모리에 상주되지 못하는 경우에 문제가 발생한다.

본 논문에서는 기존의 TB-트리의 문제점을 보완하고 이동객체의 궤적을 구성하는 라인 세그먼트를 삽입할 때의 삽입 성능을 개선하며 저장 공간 측면과 궤적-기반 질의처리의 비용을 줄이기 위해 기존의 TB-트리를 확장한 형태의 ETB-트리를 제안한다.

2. 전체 구조

제안하는 ETB-트리의 전체 구조는 [그림 1]과 같다. ETB-트리의 구조는 기존의 TB-트리와 거의 유사하며, 단지 선행 노드를 직접적으로 접근하기 위해 이동 객체의 처음 세그먼트와 마지막 세그먼트가 저장된 단말 노드를 가리키는 포인터 정보와 더불어 디스크에서의 페이지를 가리키는 페이지 번호를 유지하는 직접 테이블(Direct Table:D-Table)이 존재한다. D-Table 구조는 헤더 정보와 단말 노드에 대한 포인터 정보와 디스크의 페이지를 가리키는 레코드 정보로 구성된다. D-Table 헤더는 total_rec과 id_len으로 구성된다. 먼저 total_rec은 저장된 전체 이동 객체의 수를 의미하며, id_len은 이동 객체의 부가적인 정보를 가리키는 식별

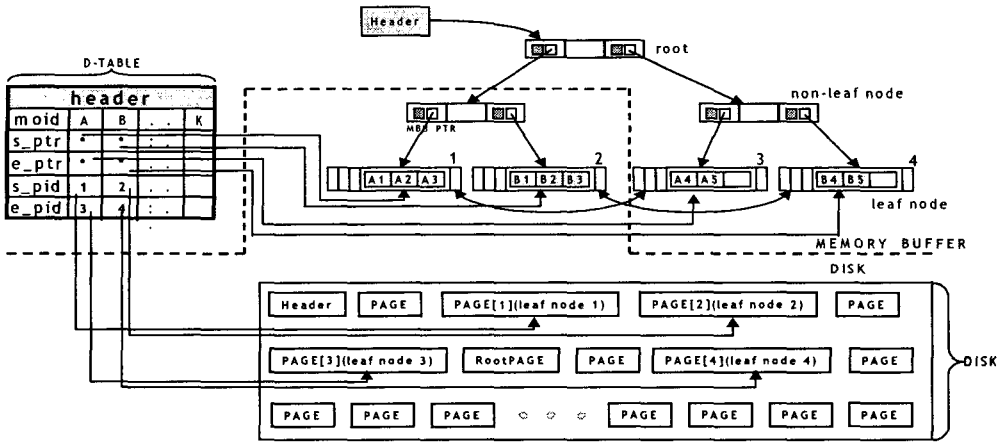


그림 1. 제안하는 ETB-트리의 전체 구조

자를 위한 식별자 길이 정보를 나타낸다. D-Table의 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 단말 노드의 메모리 포인터와 디스크 상에서의 페이지 번호, 이동 객체의 마지막 라인 세그먼트를 가지고 있는 단말 노드의 메모리 포인터와 디스크 상에서의 페이지 번호를 나타낸다.

ETB-트리는 이동 객체의 궤적에 대한 라인 세그먼트를 삽입할 때 기존의 TB-트리처럼 트리의 루트 노드에서부터 단말 노드까지 순회하지 않고 D-Table에서 해당 객체의 마지막 라인 세그먼트를 포함하고 있는 단말 노드를 가리키는 포인터(e_ptr)와 디스크 상에서의 페이지 번호(e_pid)를 이용하여 한 번에 직접적으로 단말 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. 제안하는 색인 구조는 디스크 기반 색인 구조이며 이는 색인 전체가 가용메모리 공간상에 상주되지 못할 경우를 위해 색인의 일부분을 메모리 상주시킨 후에 LRU(Least Recently Used) 정책을 이용해 메모리 상의 노드와 디스크의 페이지를 교환하도록 되어 있다. 특히 트리에 새로운 데이타가 삽입되거나 수정이 일어나는 경우에는 그 즉시 노드의 변경된 내용을 디스크에 반영하게 된다.

3. 알고리즘

ETB-트리의 삽입 알고리즘은 기존의 TB-트리의 삽입 알고리즘과 거의 유사하다. 단지 삽입 시 D-Table 정보를 이용해서 이동 객체의 선행 라인 세그먼트를 찾으며, 아울러 새로운 단말 노드가 생성되게 되면 D-Table 정보를 갱신한다. 이동 객체의 라인 세그먼트를 삽입하기 위해서는 [그림 2]의 ETB-트리 삽입 알고리즘에서와 같이 ETB_Insert() 함수를 호출한다. ETB_Insert() 함수는 삽입하려고 하는 객체가 기존의 트리에 존재하는 객체인지 확인하기 위해 ETB_FindLeafNode() 함수를 호출한다. 이 함수는 트리에 삽입하려고 하는 객체와 같은 객체가 이미 존재한다면 그 객체의 마지막 라인 세그먼트가 저장되어 있는 단말노드를 반환한다. 만일 삽입하려는 객체와 같은 객체

가 존재하지 않는다면 즉, 처음 삽입하는 객체라면 널(Null)값을 반환을 한다. ETB_Insert() 함수는 ETB_FindLeafNode() 함수에서 반환되는 값을 확인하고 새로운 단말 노드를 만들 것인지 아니면 트리에 있는 단말 노드에 삽입할 것인지를 결정한다. 만약에 단말 노드에 여유 공간이 없다면 새로운 단말 노드를 생성하고 생성된 단말 노드에 삽입한 후 종료한다. 그렇지 않고 선행 라인 세그먼트의 단말 노드가 존재하지 않으면 새로운 이동 객체의 첫 번째 라인 세그먼트로 간주하고 새로운 단말 노드를 생성해서 저장하고 종료한다.

```
Algorithm 1: ETB_Insert Algorithm
Procedure 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: 1 삽입될 라인 세그먼트 L의 선행 세그먼트가 저장된 단말 노드 N'를 찾기 위해
   ETB_FindLeafNode(MOID, Link_Type) 함수 호출
2: 2 만약 단말 노드 N'가 발견되면
3:   2.1 단말 노드 N'에 여유 공간이 있다면
4:     2.1.1 라인 세그먼트 L을 단말 노드 N'에 삽입
5:   2.2 여유 공간이 없다면 즉, 가득 차 있으면
6:   2.2.1 새로운 단말 노드를 생성하기 위해 ETB_MakeNewNode(L, MOID,
       N') 함수를 호출
7: 3 단말 노드 N'가 발견되지 않으면(N'가 NULL이면)
8:   3.1 기존의 삽입된 선행 라인 세그먼트가 없기 때문에 즉, 처음 삽입된 이동
       객체의 라인 세그먼트이므로 새로운 단말 노드를 생성하기 위해
       ETB_MakeNewNode(L, MOID, N') 함수를 호출
```

그림 2. ETB_Insert() 함수

ETB_MakeNewNode() 함수는 트리에 노드가 가득 차 있을 경우 또는 새로운 객체의 첫 번째 라인 세그먼트가 삽입되는 경우에 호출되는 함수로서 다음과 같은 순서로 수행된다. ETB-트리는 단말 노드를 포함해서 자식 노드에는 부모 노드를 가리키는 포인터를 가지고 있기 때문에 해당 노드의 부모 노드에 대한 정보를 얻을 수 있으며 해당 노드의 변경된 정보를 부모 노드에게 반영시킬 수 있다. 먼저 새로운 단말 노드를 생성하고 이 단말 노드와 이전의 단말 노드 간에 연결 리스트의 링크를 설정한다. 이전의 단말 노드는 새로운 라인 세그먼트가 삽입될 때 D-Table을 통하여 찾은 단말 노드이다. 두 번째로 트리의 루트 노드를 검색하여 루트 노드가 여유 공간이 있는지 없는지를 확인한다. 루트 노드에 여유 공간이 없다면 새로운 루트 노드를 생성하고 기존의 루트 노드를 자식으로 가지는 트리를 구성한다. 그리고 삽입될 단말 노드의 트리 레벨(depth)까지 새로운 비단말 노드들을 생성한다. 이 비단말 노드와 단말 노드를 연결하고 함수를 끝마친다. 만약에 루트 노드에 여유 공간이 있다면 이것은 루트 노드 아닌 부모 노드가 여유 공간이 없어서 이 함수를 호출한 것이므로 미리 저장되어 있는 트리의 가장 오른쪽의 단말 노드를 검색하여 이 단말 노드의 부모 노드를 따라 트리를 탐색한다. 부모 노드를 따라 트리를 탐색하는 도중에 여유 공간이 없는 노드를 만났을 경우, 여유 공간이 없는 노드로부터 자식 노드(비단말 노드)를 단말 노드의 트리 레벨까지 생성한다. 그리고 부모 노드의 MBR을 갱신하고 함수를 끝마친다.

[그림 3]은 이동 객체의 궤적 세그먼트들을 제안하는 ETB-트리에 삽입하는 과정을 보이기 위한 이동 객체 A와 B의 예제 궤적 세그먼트들이다.

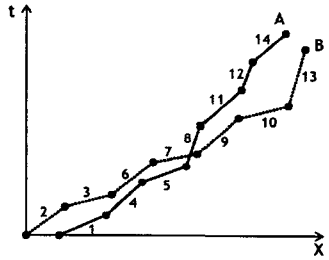


그림 3. 예제 궤적 세그먼트

[그림 4]에서 [그림 7]까지는 데이터를 이용하여 ETB-트리에 삽입하는 과정을 도식화한 것이다.

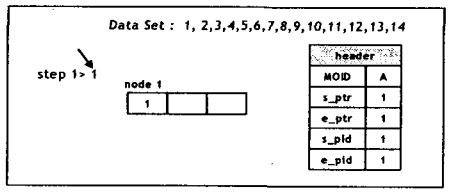


그림 4. 데이터 삽입 과정-1

[그림 4]와 같이 첫 번째의 궤적 세그먼트가 들어오면 첫 번째 단말 노드에 삽입을 하고 D-Table에는 궤적의 처음 노드와 마지막 노드가 Node 1로 같은 값을 갖는다.

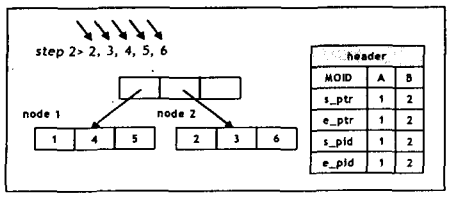


그림 5. 데이터 삽입 과정-2

[그림 5]와 같이 두 번째 궤적 세그먼트가 다른 객체 B이므로 다음과 같이 노드를 하나 더 생성하고 같은 궤적 세그먼트들은 같은 노드에 삽입하게 된다. D-Table에는 새로운 객체 B가 들어왔으므로 새로운 레코드를 생성하고 새로운 객체 B가 삽입된 노드를 가리킨다.

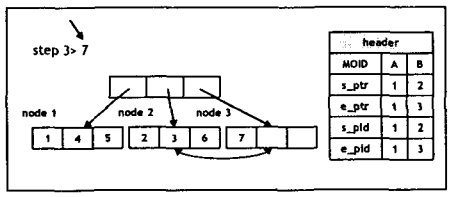


그림 6. 데이터 삽입 과정-3

[그림 6]과 같이 객체 B의 궤적 세그먼트 7이 들어왔을 때 Node 2가 오버플로가 발생하므로 새로운 노드를 생성하고 연결 리스트로 링크시킨다. 테이블에는 마지막 노드가 Node 3으로 바뀌었으므로 Node 3으로 수정한다.

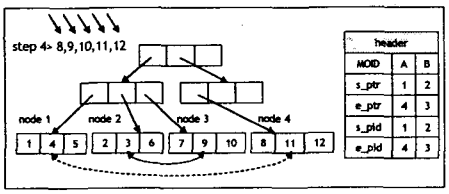


그림 7. 데이터 삽입 과정-4

[그림 7]과 같이 객체 A의 세그먼트 8이 들어 왔을 때 Node 1이 오버플로가 발생하므로 새로운 노드 Node 4를 생성하고 부모노드 또한 오버플로가 발생하므로 새로운 루트 노드와 비단말 노드를 생성한다. 그리고 D-Table의 객체 A의 마지막 노드를 Node 4로 수정한다.

Ⅳ. 실험 환경 및 성능 평가

1. 실험 환경

본 논문에서는 R*-트리, TB-트리, 그리고 제안하는 ETB-트리를 구현하여 삽입 성능과 궤적 질의에 대해서 성능 평가를 수행하였으며 수행 환경은 Pentium-Ⅳ 1.7GHz CPU와 메인 메모리 512MB, 윈도우 2000, 그리고 C++ 언어를 이용하였다. R*-트리, TB-트리, ETB-트리 모두 디스크 페이지의 크기는 4KB이다. 성능 평가를 위해 사용한 실험 데이터는 GSTD 알고리즘[11]을 이용해 데이터 분포도를 고려한 경우와 그렇지 않은 경우 2가지로 나누어 생성하였다. [표 1]은 데이터 분포도를 고려한 실험 데이터(이하 D_Dataset)를 나타내며, 이를 위해서는 각 분포도에 따라 객체의 수가 1000(1K)에서부터 10000(10K)까지 100초 동안 이동한 데이터를 toriod 좌표계를 사용해 생성하였다. [표 2]는 데이터 분포도를 고려하지 않고 단지 객체의 수와 각 객체의 라인 세그먼트 수를 고려한 실험 데이터(이하 N_Dataset)를 나타낸다. 성능 평가는 기존의 R*-트리, TB-트리, 그리고 ETB-트리에 대해서 삽입 성능과 검색 성능을 소요 시간(elapsed time)과 노드 접근 횟수(node access) 측면을 고려하여 수행하였다.

표 1. 분포도를 고려한 실험데이타(D_Dataset)

	초기 분포도 (initial distribution)	이동 방향성 (movement)
UR dataset	Uniform	Random
UD dataset	Uniform	Directed
GR dataset	Gaussian	Random
GD dataset	Gaussian	Directed

표 2. 분포도를 고려하지 않은 실험데이타(N_Dataset)

종류 / 파라미터	데이터셋 1 (D1)	데이터셋 2 (D2)	데이터셋 3 (D3)
전체 라인 세그먼트 수	200,000 (0.2M)	2,000,000 (2M)	20,000,000 (20M)
이동 객체의 수	100	1,000	10,000
객체 당 라인 세그먼트의 수	2,000	2,000	2,000

2 성능 평가

2.1 삽입 성능

[그림 8]은 삽입 성능을 노드 접근 횟수와 시간으로 나타낸 것이다. 기존의 TB-트리가 가장 많은 노드 접근 횟수를 보이고 있으며, 특히 데이타의 양이 많아짐에 따라 다른 색인 기법보다는 성능이 떨어지는 걸 알 수 있다. 이는 객체의 수가 많아질수록 해당 객체의 단말 노드를 탐색하는 데 더 많은 노드 접근을 요구하며 하나의 라인 세그먼트가 삽입될 때마다 객체의 이전 세그먼트를 탐색하기 위해 루트 노드에서부터 단말 노드까지 탐색해야 하는 오버헤드가 높기 때문이다. 그에 반해 R*-트리와 ETB-트리는 노드 접근 횟수가 그래프는 스케일로 인해 다소 유사해 보이지만, 실제로는 20만개의 라인 세그먼트인 D1 데이타 셋의 경우, ETB-트리는 10000번, R*-트리는 11681번으로 약 1600번의 차이를 보이며, 200만개의 D2의 경우는 ETB-트리가 100000, R*-트리는 194188번으로 약 90000번의 큰 차이를 보인다. 이는 R*-트리의 재삽입 정책 등으로 인해 ETB-트리보다는 노드 접근 횟수가 많다. TB-트리가 객체를 찾기 위해 전체 트리를 순회하는 횟수가 이동 객체의 수에 비례하는 반면에 ETB-트리는 객체를 찾기 위해 D-Table을 이용하기 때문에 노드 접근 횟수가 적다.

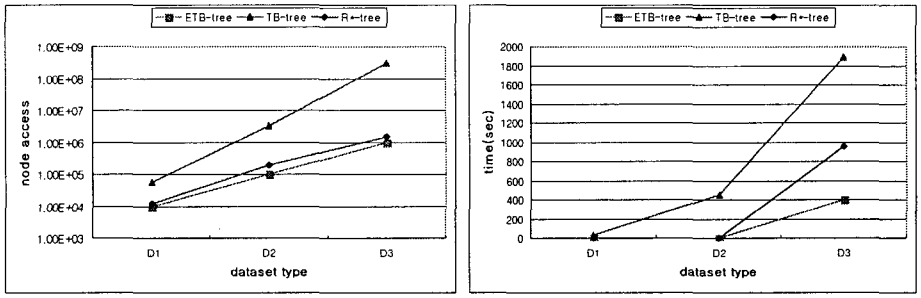


그림 8. N_Dataset에 대한 삽입 성능 결과

2.2 검색 성능

본 논문에서는 각 색인의 검색 성능을 평가하기 위해 이동 객체를 위한 다양한 질의 타입 중에서 궤적 질의(trajectory query)와 복합 질의(composite query)에 대해서 성능 평가를 수행하였다. 궤적 질의는 각각의 실험 데이타 집합에 대해서 임의로 100개의 질의 이동 객체 식별자(MOID)를 선별해서 해당 이동 객체의 전체 궤적시간 영역 중에 10% 구간을 검색하는 방식으로 처리했으며 노드 접근 횟수는 전체 결과에 대해서 평균을 구한 것이다.

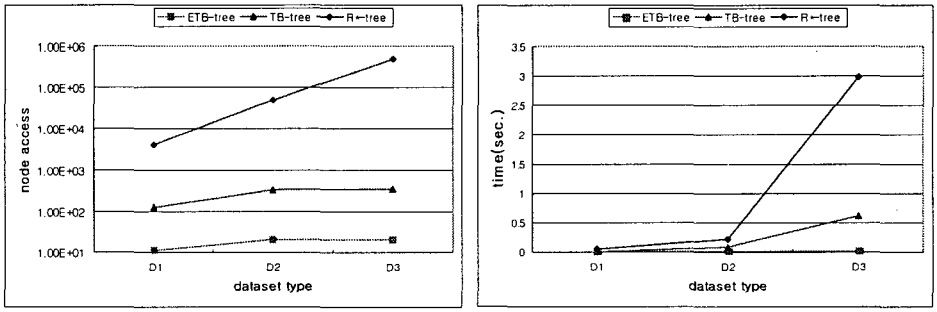


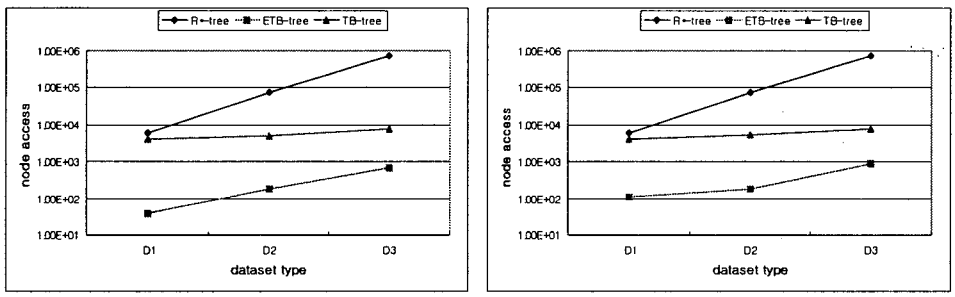
그림 9. N_Dataset에 대한 궤적 질의 결과

[그림 9]는 N_Dataset를 이용하여 궤적 질의에 대한

검색 성능을 노드 접근 횟수와 시간으로 나타낸 것이다. R*-트리가 가장 많은 노드 접근 횟수를 보이고 있으며, 특히 데이타의 크기가 증가할수록 노드 접근 횟수도 같이 증가하는 모습을 나타낸다. 이는 R*-트리의 특성상 동일한 이동 객체의 궤적을 검색하기 위해서는 각 라인 세그먼트마다 전체 트리를 모두 순회해야 하기 때문이다. 그에 반해 TB-트리와 ETB-트리는 R*-트리와 같이 데이타의 크기에 큰 영향을 받지 않는 편이며, 궤적 질의에 대해서는 성능이 매우 우수한 편이다. 특히 ETB-트리는 궤적 질의 처리시 TB-트리와는 달리 객체의 MOID(식별자)를 탐색하기 위해 트리를 탐색하는 오버헤드가 없고 단지 D-Table을 통해 해당 MOID를 검색한 후 연결 리스트로 링크되어 있는 단말 노드들만을 접근하면 되기 때문에 궤적 질의에 좋은 성능을 보이고 있다. 성능 평가 결과 약 10배 정도의 성능 향상을 나타내고 있다.

[그림 10]은 N_Dataset에 대하여 복합 질의에 대한 검색 성능을 노드 접근 횟수로 나타낸 것이다. 복합 질의는 궤적 질의와 마찬가지로 질의는 100번을 수행하였으며, 질의 영역(range)은 각각 전체 영역의 1%와 10%로 정했으며 영역을 지나는 이동 객체의 전체 궤적을 검색하는 방식으로 성능평가를 수행하였다. 즉, 먼저 설정된 1% 혹은 10%의 질의 영역에 대해서 영역 질의를 수행한 후, 이 영역에 포함되는 이동 객체의 식별자, MOID를 찾은 다음, D-Table을 통해 해당 이동 객체의 첫 번째 라인 세그먼트가 들어가 있는 단말 노드를 찾고 나서 나머지 전체 궤적을 검색하는 방식이다. 그림에서와 같이 복합 질의의 경우 ETB-트리가 가장 적은 노드 접근 횟수를 보이고 있으며, TB-트리와 R*-트리의 순으로 노드 접근 횟수가 많아짐을 알 수가 있다. 이는 복합 질의를 수행할 때에도 궤적을 검색을 해야 하므로 궤적질의를 수행했을 때의 성능의 차이가 지속됨을 알 수가 있다. 전체적으로는 TB-트리나 ETB-트리가 복합질의에서 R*-트리에 비해 좋은 성능을 보이고 있지만, 영역 질의시 노드 접근 횟수와 궤적 질의시 노드 접근 횟수를 분리해서 측정하면 오히려 영역 질의 측면에서는 R*-트리가 TB-트리나 ETB-트리에 비해 더 나은 성능을 보인다.

[그림 11]과 [그림 12]는 각각 D_Dataset을 이용하여 각 분포도에 따른 궤적 질의의 성능 평가 결과를 노드 접근 횟수와 시간으로 나타낸 것이다. 궤적 질의는 1,000개의 이동 객체의 궤적을 검색하도록 하였다. R*-트리는 궤적에 대한 특별한 정책이 없기 때문에 궤적



(a) (1% 영역, 전체궤적) (b) (10% 영역, 전체궤적)

그림 10. N_Dataset에 대한 복합 질의 결과

질의에 대해서 이동 객체의 개수가 많아질수록 페이지 접근 횟수가 점진적으로 증가함을 알 수 있다. R*-트리는 TB-트리와 ETB-트리보다 페이지 접근 횟수에서 많은 차이를 보이고 있다. 이는 앞에서 언급하였듯이 R*-트리의 경우에는 궤적 질의에 대한 특별한 정책이 없고 영역질의 성능을 향상시킨 색인 기법이기 때문에 궤적 질의에 대해서는 많은 페이지 접근 횟수의 차이를 보이고 있는 것이다. ETB-트리와 TB-트리사이에서는 ETB-트리가 TB-트리보다 약 10배 정도 좋은 성능을 보이고 있다. 그 이유는 ETB-트리의 경우 D-Table을 사용하여 거의 한 번의 노드 접근으로 궤적을 찾기 때문이다. 궤적 질의에서 ETB-트리와 TB-트리는 분포도에 영향을 받지 않고 똑같은 노드 방문 횟수를 보이

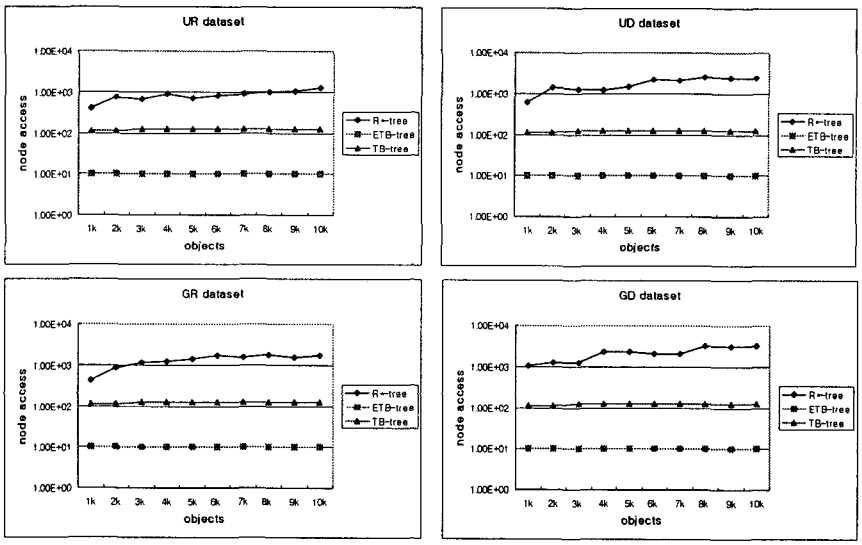


그림 11. D_Dataset에 대한 궤적 질의 결과(노드 접근 횟수)

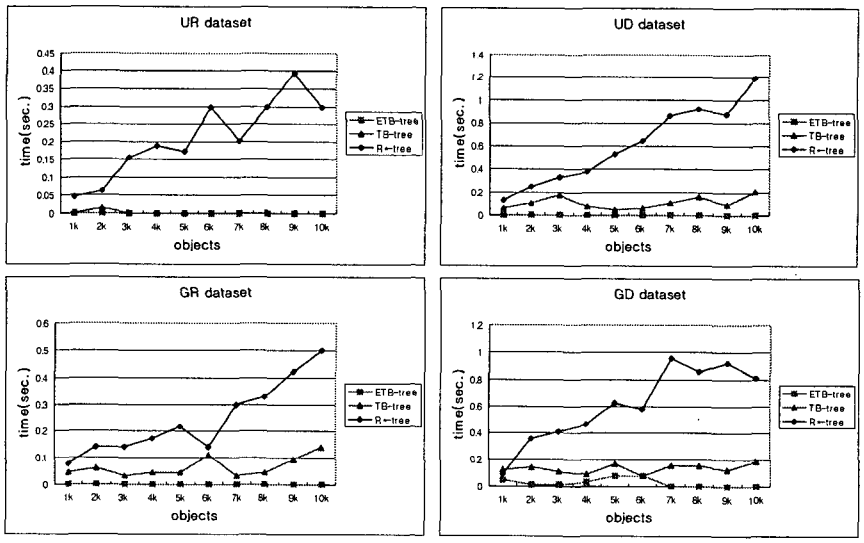


그림 12. D_Dataset에 대한 궤적 질의 결과(시간)

고 있다. 이것은 ETB-트리와 TB-트리 모두 궤적 질의를 위해 이동 객체를 찾을 때에 데이터가 어떻게 분포하는가와는 상관관계가 없다는 것을 의미한다.

V. 결 론

본 논문에서는 이동 객체의 궤적을 색인하기 위해 기존에 제안되었던 TB-트리의 성능을 개선시킬 수 있는 확장된 TB-트리 즉 ETB-트리를 구현하였다. 제안하는 ETB-트리의 특징을 정리하면 다음과 같다. 첫째, ETB-트리는 기존의 TB-트리 구조에 연결 리스트로 구현된 직접 테이블을 결합한 구조를 가지고 있다. 둘째, ETB-트리는 D-Table을 이용하여 기존의 TB-트리가 가지고 있는 삽입 오버헤드를 줄일 수 있다. 셋째,

ETB-트리는 D-Table을 통해 궤적 질의 처리시 원하는 궤적의 첫 번째 세그먼트를 한 번의 노드 접근으로 탐색할 수 있다. 이를 통해 궤적 질의 검색 성능을 향상시킬 수 있다. 넷째, ETB-트리는 디스크 기반의 트리 구조로서 메모리에 상주된 트리의 정보가 변경되는 즉시 디스크에 반영함으로써 트리의 일관성을 유지하도록 설계되어 있다. 다섯째, 시스템의 가용메모리의 제약으로 인해 트리 전체가 메모리에 상주되지 못하는 경우를 위해서 트리의 일부분만을 메모리 상주시키고 LRU 정책을 이용해 트리의 노드를 메모리와 디스크로 교체할 수 있도록 하였다. 여섯째, 성능 평가 결과 삽입 및 검색 특히 궤적 질의에는 기존의 TB-트리보다 더 나은 성능을 보이고 있다.

참 고 문 헌

[1] Z. Song, and N. Roussopoulos, "Hashin Moving Objet," International Conference on Mobile Data Management, pp.161~172, 2001.

[2] D. Kwon, S. Lee, and S. Lee, "Indexing the Current Positions of Moving Objects Using the Lazy Update R-Tree," Conference on Mobile Data Management, pp.113~120, 2002.

[3] Tayeb J., Ulusoy O., and Wolfson O., "A Quadrate Based Dynamic Attribute Indexing Method," The Computer Journal, Vol.41, No.3, pp.185~200, 1998.

[4] M. A. Nascimento, and silva J. R. O., "Towards historical R-trees," ACM symposium on Applied Computing, pp.235~240, 1998.

[5] Tao Y., Papadias D., "MV3R-Tree: A Spatio-Temporal Access Method for Timestamp and Interval Queries," International Conference on VLDB, pp.431~440, 2001.

[6] Y. Theodoridis, M. Vazirgiannis, and T. Sellis K., "Spatio-Temporal Indexing for Large Multimedia Applications," IEEE International Conference on Multimedia Computing and Systems, pp.441~448, 1996.

[7] N. Beckman, and H. P. Kriegel, "The R*-tree:An Efficient and Robust Access Method for Points and Rectangles," In Proc. of ACM SIGMOD, pp.302~331, 1990.

[8] Y. Heodoridis, and C. S. Jensen, "Indexing Trajectories in Query Processing for Moving Objects," Chorochronos Technical Report, CH-99-3, 1999.

[9] C. S. Jensen, and Y. Theodoridis, "Novel Approaches in Query Processing for Moving Objects," International Conference on VLDB, pp.395~406, 2000.

[10] A. Guttman, R-Trees: A Dynamic Index Structure for Spatial Searching. In Proc. of the 1984 ACM SIGMOD Conference on Management of Data. pp.47~57. 1984.

[11] Y. Theodoridis, J. R. O. Silva, and M. A. Nascimento, "On the generation of spatiotemporal datasets," International Symposium on Spatial Datasets, pp.147~164, 1999.

저 자 소 개

심 춘 보(Choon-Bo Shim) 정회원

- 1996년 2월 : 전북대학교 컴퓨터공학과(공학사)
- 1998년 2월 : 전북대학교 컴퓨터공학과(공학석사)
- 2003년 2월 : 전북대학교 컴퓨터공학과(공학박사)
- 2005년 2월~현재 : 순천대학교 정보통신공학부 교수

<관심분야> : 멀티미디어 IR & DB, 이동 DB, LBS

신 용 원(Yong-Won Shin) 정회원

- 1992년 2월 : 인제대학교 의용공학과(공학사)
- 1996년 2월 : 인제대학교 의용공학과(공학석사)
- 2000년 2월 : 인제대학교 의용공학과(공학박사)
- 2004년 3월~현재 : 부산가톨릭대학교 병원경영학과 교수

<관심분야> : 의료콘텐츠, 의료 데이타베이스

박 병 래(Byung-Rae Park) 정회원

- 1992년 2월 : 인제대학교 의용공학과(공학사)
- 1994년 2월 : 동의대학교 전자공학과(공학석사)
- 2002년 2월 : 부산대학교 의과대학 의공학협동(공학박사)
- 2003년 2월~현재 : 부산가톨릭대학교 방사선학과 교수

<관심분야> : 의료 방사선영상, 의료전문가 시스템