효율적인 이동 객체 궤적 색인을 위한 최소 전파 TB-tree

고주일*O, 김명근*, 정원일*, 김재홍**, 배해영*
*인하대학교 전자계산공학과
**영동대학교 컴퓨터정보공학부
{jikoO}@dblab.inha.ac.kr

Minimal Propagation TB-tree for Efficient Indexing of Moving Objects Trajectories

Ju-Il Ko*O, Myung-Keun Kim*, Warnill Chung*, Jae-Hong Kim**, Hae-Young Bae*
*School of Computer Science and Engineering, Inha University
**Dept of Computer Science, Youngdong University

요 약

시간이 흐름에 따라 연속적으로 위치를 변경하는 객체를 이동 객체(Moving Objects)라고 한다. 이러한 이동 객체의 대용량 위치 정보를 효율적으로 검색하기 위하여 색인이 필요하며, 대표적인 색인으로 TB-tree가 제안되었다. 그러나 전통적인 R-tree 기반의 TB-tree는 엄격한 궤적 보존 정책에 의해 레코드가 삽입될 때마다 해당 레코드의 선행자(predecessor)를 포함하는 단말 노드를 검색해야 하며, 레코드 삽입으로 인한 단말 노드 MBB의 변경을 중간 노드들의 MBB에도 반영해야 하는 갱신 부하를 가지고 있다.

본 논문에서는 대용량 이동 객체 궤적 정보의 효율적인 색인을 위한 최소 전파 TB-tree를 제안한다. 본 기법은 앞으로 삽입될 이동 객체의 궤적을 포함하는 예상된 MBB(EMBB: Expected Minimum Bounding Box)를 트리에 먼저 반영한 후 레코드가 삽입될 때마다 중간 노드의 MBB를 갱신하지 않고, 객체가 EMBB을 벗어났을 때 중간 노드의 MBB를 조정하여 TB-tree의 MBB 조정 횟수를 줄이고, 또한 TB-tree에 별도의 테이블 구조를 둠으로써 레코드 삽입을 위한 단말 노드 검색 비용을 줄여 전체적인 TB-tree의 갱신 비용을 감소시킨다.

1. 서 론

최근 무선 통신과 측위 기술의 발달로 인하여 위치 기반 서비스(LBS: Location Based Services)에 대한 요구가 증가되고 있다. 위치 기반 서비스에서는 시간이 흐름에 따라 연속적으로 위치가 변경되는 이동 객체(Moving Objects)의 위치 정보를 효과적으로 저장 및 검색하기 위하여 색인이 필요하며, 대표적인 색인으로 TB-tree가 제안되었다.

그러나 전통적인 공간 색인인 R-tree 기반의 TB-tree는 레코드가 삽입될 때마다 발생하는 MBB의 변경을 해당 단말 노드에서부터 루트 노드까지 모두 전파해야 하며, 엄격한 궤적 보존 정책에 의하여 레코드가 삽입될 때마다 해당 레코드의 선행자(predecessor)를 포함하는 단말 노드를 검색해야 하는 부하를 가지고 있다[1,4]. 이러한 TB-tree의 갱신 부하는 이동 객체의 수가 증가하고 그 위치가 연속적으로 변경됨에 급격히 증가하게 된다.

따라서 본 논문에서는 이러한 TB-tree의 갱신 부하 감소를 위한 최소 전파 TB-tree(MPTB-tree: Minimal Propagation TB-tree)를 제안한다. 제안 기법은 예상된 MBB(EMBB: Expected MBB)를 이용하여 레코드가 삽입될 때마다 발생하는 중간 노드의 MBB 조정 부하를 감소시킨다. EMBB는 앞으로 이동 객체가 이동할 예상된 영역으로 처음 이동 객체의 레코드가 삽입될 때 설정되어 트리에 반영된다. 그 후 삽입되는 이동 객체의 레코드가 EMBB를 벗어나지 않으면 중간 노드의 MBB를 조정해 주지 않고, EMBB를 벗어날 경우 EMBB를 재설정하여 삽입된 레코드와 재설정된 EMBB를 포함하도록 트리의 MBB를 재조정한다. 또한 제안 기법은 별도의 Predecessor Link 구조를 유지하여 삽입되는 레코드의 선행자를 포함하는 단말 노드에 대한

빠른 접근을 가능하게 한다.

제안 기법은 EMBB를 트리에 먼저 반영하여, 그 후 삽입되는 레코드가 EMBB를 벗어나지 않으면 중간 노드의 MBB가 조정되지 않으므로, 레코드가 삽입될 때마다 발생하는 MBB 조정 부하가 크게 감소되며, 별도의 Predecessor Link 구조를 유지하여, 레코드가 삽입될 때 해당 레코드의 선행자를 포함하는 단말 노드가 빠르게 검색되어 전체적인 갱신 비용이 감소된다.

2. 본 론

2.1 관련 연구

본 절에서는 먼저 이동 객체의 위치 정보 중 궤적 정보를 효율적으로 저장 및 검색하기 위하여 제안된 기존의 이동 객체 색인 기법들에 대해 설명하고, 다음으로 이동 객체의 빈번한 갱신 연산 비용을 줄이기 위하여 제안된 이동 객체 색인 기법에 대해 설명한다.

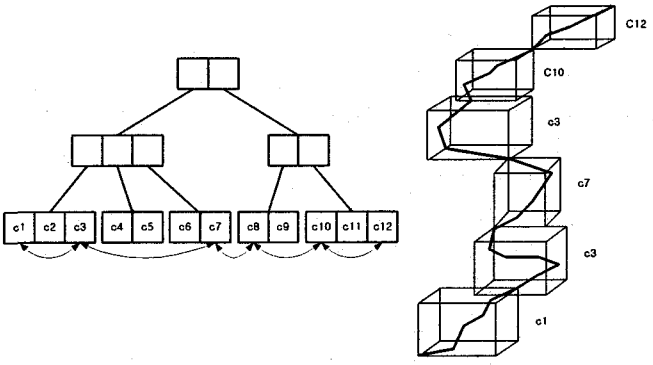
2.1.1 이동 객체 색인 기법

이동 객체 색인에 대한 연구는 크게 이동 객체의 현재 위치 색인에 관한 연구와 과거 위치(궤적) 색인에 관한 연구로 분류되며, 본 논문은 후자의 문제를 다룬다. 이동 객체의 과거 위치 즉 궤적 색인에 대해서는 많은 연구가 진행되어 왔으며, 대표적인 색인으로 3DR-tree, STR-tree, TB-tree 등이 제안되었다[3,5,6,7].

3DR-tree는 2차원 공간에 시간 차원을 고려한 3차원 R-tree 색인으로, 어떠한 R-tree의 변형들도 사용될 수 있다[7]. 3DR-tree는 불필요한 정보를 가지고 있지 않기 때문에 크기가 가장 작고, 3차원 영역 질의에 대해 우수한 성능을 보인다. 그러나 실제 궤적이 차지하는 부분에 비해 많은 양의 사장 공간(dead space)이 발생하기 때문에 겹치는 영역이 증가 되어 색인의 성능이 저하된다.

STR-tree(Spatio-Temporal R-tree)는 이동 객체의 궤적에 대한 효율적인 질의 처리를 지원하기 위하여 확장된 R-tree이다. 따라서 STR-tree는 공간 속성 즉 공간 인접성(spatial closeness)과 궤적 보존(trajectory preservation)의 두 가지 성질을 모두 고려하여 라인 세그먼트들을 구성한다(궤적 보존이란 소속된 궤적에 따라 세그먼트들을 그룹화하는 것으로 동일한 궤적에 속한 세그먼트들은 같은 그룹으로 분류된다). 전반적으로는 R-tree와 유사하며, 단지 삽입/분할 알고리즘이 다르다[4].

TB-tree(Trajectory-Bundle tree)는 앞서 소개한 두 색인 기법과 근본적으로 다른 단계를 거친다[4]. 즉 TB-tree는 각각의 라인 세그먼트들을 궤적의 일부분으로 보고, 단말 노드들은 동일한 궤적의 세그먼트들만을 저장하도록 하여 엄격하게 궤적을 보존하는 것에 중점을 둔다. TB-tree의 인덱스 구조는 [그림 1]과 같다. 단말 노드에는 하나의 궤적만 저장되고, 동일한 궤적을 저장하는 단말 노드들은 이중 연결 리스트(double linked list)로 연결되어 있어 선택된 궤적의 연속된 궤적을 빠르게 검색할 수 있다. TB-tree는 엄격한 궤적 보존 정책에 의해서 궤적 기반 질의에 대해서는 좋은 성능을 보이지만, 세그먼트의 공간적 특성을 고려하지 않으므로 단말 노드간의 중복이 심하여 영역 질의의 성능이 상대적으로 낮고, 또한 삽입 연산을 위한 FindNode의 비용 증가로 삽입 성능이 나쁘다.



[그림 1] TB-tree의 구조

이들 기존 이동 객체 색인 기법들은 전통적인 R-tree 기반의 색인으로 레코드 삽입에 따른 분할, 합병 및 MBB 재조정 등의 갱신 부하를 가지고 있어 실시간 위치 정보 색인에는 부적합하다.

2.2.2 갱신 부하를 고려한 이동 객체 색인 기법

전통적인 데이터베이스 시스템과는 다르게 이동 객체 데이터베이스 시스템에서는 이동 객체가 지속적으로 움직임에 따라 많은 갱신 연산이 발생한다. 이러한 갱신 부하를 고려하여 제안된 기법으로 해쉬을 이용한 기법과 LUR-tree(Lazy Update R-tree)가 있다[2,8,10].

해쉬을 이용한 기법은 전체 공간을 일정한 개수의 격자로 자르고, 객체가 속한 격자의 번호만을 데이터베이스에 저장한 후, 객체가 이 격자에 속한 동안은 위치의

변경을 수행하지 않고 다른 격자로 이동한 경우에만 데이터베이스에 변경 연산을 수행하여 데이터베이스의 갱신 비용을 감소시킨다[10].

LUR-tree는 이동 객체의 새로운 위치가 기존 단말 노드의 MBR 범위를 벗어나지 않는 경우, 트리의 구조를 변경하지 않고 단지 노드 내부의 위치만을 변경하여 R-tree에서의 갱신 연산 비용을 크게 감소시키는 기법이다[2].

이들 기법은 이동 객체 색인에서 갱신 부하를 고려한 기법들이지만, 단지 현재 위치만 색인이 가능하다는 한계를 가지고 있다.

2.2 최소 전파 TB-tree

2.2.1 기본 원리

이동 객체의 궤적 색인을 위한 대표적인 기법인 TB-tree는 전통적인 공간 색인인 R-tree에 기반한 구조이다. R-tree는 레코드를 삽입하기 위해서 단지 단말 노드에 인덱스 엔트리(MBB: Minimum Bounding Box, tuple-identifier)를 삽입하는 것으로 끝나지 않고, 레코드 삽입에 따른 MBB의 변경을 상위 중간 노드에 전파하여, 레코드가 삽입된 단말 노드로부터 루트 노드까지 모든 중간 노드의 MBB 키 값을 재조정해주어야 하는 갱신 부하를 가지고 있다. 또한 TB-tree는 엄격한 궤적 보존 정책에 의해, 단말 노드는 동일한 이동 객체의 궤적만을 저장하므로, 레코드를 삽입할 때마다 삽입될 레코드의 선행자(predecessor)를 포함하는 단말 노드를 항상 검색해야 하는 부하를 가지고 있다. 이러한 TB-tree의 갱신 부하는 이동 객체의 수가 증가하고 그 위치가 연속적으로 변경됨에 따라 급격히 증가하게 된다.

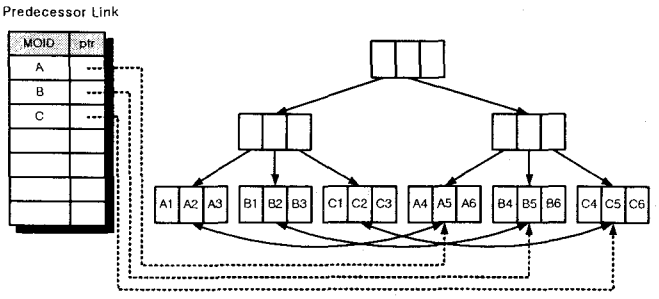
따라서, 본 논문에서는 이동 객체 궤적 색인의 갱신 부하를 고려한 최소 전파 TB-tree(MPTB-tree: Minimal Propagation TB-tree)를 제안한다. 제안 기법은 예상된 MBB(EMBB: Expected MBB)를 이용하여, 레코드가 삽입될 때마다 발생하는 MBB의 변경을 상위 노드에 전파(propagation)하는 횟수를 줄여 레코드 삽입에 따른 색인의 갱신 비용을 감소시킨다. EMBB는 앞으로 이동 객체가 이동할 예상된 영역으로, 처음 이동 객체의 레코드가 삽입될 때 해당 이동 객체의 위치로부터 일정한 영역이 EMBB로 설정되어 트리에 반영된다. 그 후에 삽입되는 이동 객체의 레코드들이 EMBB를 벗어나지 않으면, 상위 노드로 MBB의 변경 전파하여 중간 노드의 MBB를 조정하지 않고, 단지 EMBB를 벗어날 경우 새로운 EMBB를 재설정하여 실제 삽입된 레코드의 MBB와 재설정된 EMBB를 포함하도록 MBB의 변경을 트리에 전파하여 상위 중간 노드들의 MBB를 재조정한다. 또한 제안 기법은 TB-tree의 엄격한 궤적 보존 정책에 의해 레코드 삽입시 삽입될 레코드의 선행자를 포함하는 단말 노드를 검색하는 비용을 줄이기 위하여 별도의 Predecessor Link를 유지한다. 제안 기법은 Predecessor Link를 통하여, 레코드가 삽입될 단말 노드(삽입될 레코드의 선행자를 포함하는 단말 노드)에 대한 빠른 검색을 가능하게 한다.

2.2.2 확장된 인덱스 구조

본 절에서는 제안 기법인 최소 전파 TB-tree를 위하여 확장된 TB-tree 구조를 설명한다. 확장된 TB-tree 구조는 제안 기법을 위하여 Predecessor Link를 유지하고, 단말 노드의 인덱스 페이지 헤더에 EMBB를 위한 필드를 추가한다.

가) Predecessor Link

최소 전파 TB-tree는 기존 TB-tree의 FindNdoe 알고리즘의 비용을 줄이기 위해 Predecessor Link 구조를 유지한다. Predecessor Link의 추가로 인해 확장된 TB-tree의 구조는 [그림 2]과 같다.

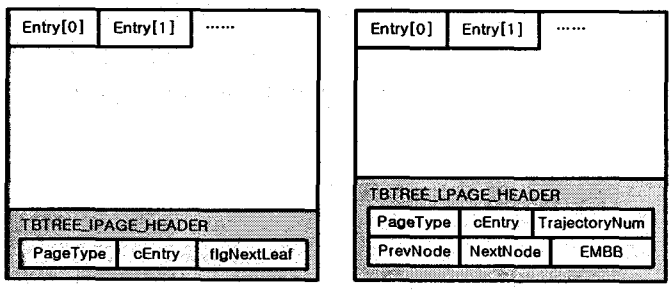


[그림 2] 확장된 TB-tree 구조

Predecessor Link는 TB-tree를 위한 보조 인덱스로써, 이동 객체의 아이디(MOID)가 키(key)가 된다. Predecessor Link의 엔트리들은 해당 이동 객체(MOID)의 선행자를 포함하는 단말 노드에 대한 포인터(ptr)를 가지고, 포인터는 선행자를 포함하는 단말 노드가 오버플로우로 인하여 변경되었을 경우에만 갱신된다. Predecessor Link를 위해서는 기존의 해쉬 인덱스 또는 B-tree가 사용될 수 있다. 본 논문에서는 해쉬를 이용하여 Predecessor Link를 구현하였다.

나) 페이지 구조

제안 기법인 최소 전파 TB-tree의 페이지 구조는 [그림 3]과 같다. 비단말 노드 페이지는 기존 TB-tree와 유사하며, 단말 노드 페이지는 EMBB를 이용한 갱신 정책으로 인하여 페이지 헤더에 EMBB 필드가 추가되었다.



(a) 비단말 노드 페이지 (b) 단말 노드 페이지

[그림 3] 최소 전파 TB-tree의 페이지 구조

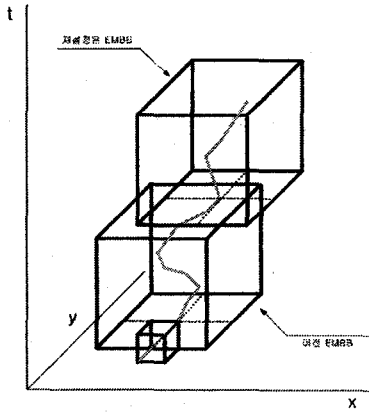
비단말 노드와 단말 노드 페이지의 엔트리 구조는 [그림 4]와 같다.

(a) 비단말 엔트리 (b) 단말 엔트리

[그림 4] 최소 전파 TB-tree의 페이지 엔트리 구조

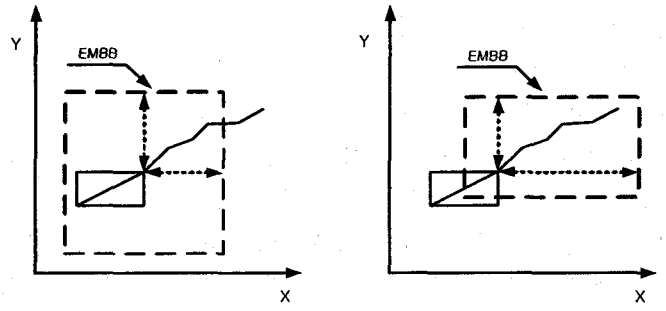
2.2.3 EMBB

EMBB(Expected MBB)는 이동 객체의 예상된 궤적 세그먼트들을 포함하는 MBB이다. EMBB는 이동 객체가 앞으로 이동할 예상된 영역으로, 처음 레코드가 삽입되거나 삽입되는 레코드가 앞서 설정된 EMBB의 영역을 벗어날 때, 단말 노드에서 설정되어 트리에 반영된다. EMBB 설정 예는 [그림 5]와 같다.



[그림 5] EMBB를 이용한 MPTB-tree

제안 기법에서 EMBB를 설정하는 방법은 다음과 같다. 이동 객체의 예상된 이동 영역은 속도와 방향과 같은 이동 객체의 속성을 고려해야 한다. 제안 기법의 EMBB 설정은 세부적으로 2가지 방식으로 구분된다. 삽입된 이동 객체의 속도를 기준으로 regular EMBB과 rectangular EMBB 방식이 있다. 최근 삽입된 이동 객체의 속도가 제한 속도보다 빠르면 rectangular EMBB 방식을, 제한 속도보다 느리면 regular EMBB 방식을 사용한다. Regular EMBB 방식은 최근 획득된 이동 객체의 위치를 기준으로 정사각형의 EMBB를 설정하는 방식이고, rectangular EMBB 방식은 이동 객체의 방향을 고려하여 해당 방향으로 기울어진 사각형의 EMBB를 설정하는 방식이다. 이유는 이동 객체의 속도가 빠를수록 이동 방향이 급격하게 변화하지 않기 때문이다. [그림 6]이 regular EMBB와 rectangular EMBB 방식을 설명하고 있다.



(a) regular EMBB (b) rectangle EMBB

[그림 6] EMBB 설정 방식

2.2.4 알고리즘

TB-tree에서의 갱신 연산은 삽입 연산을 의미한다. 따라서 본 절에서는 최소 전파 TB-tree의 삽입 알고리즘에 대해서 자세히 설명한다.

제안 기법의 구체적인 삽입 알고리즘은 [알고리즘 1]과 같다. 먼저 Predecessor Link를 통하여 레코드를 삽입할 단말 노드를 검색한다(FindNode). 단말 노드가 발견되지 않는다면, 새로운 이동 객체의 레코드가 위치를 보고한 경우이다. 따라서, 삽입될 레코드를 포함하는 새로운 단말 노드를 생성하고, Predecessor Link에 등록한다. 그리고 새로이 생성된 단말 노드의 삽입을 위하여 부모 노드(level 1)를 검색하여 생성된 단말 노드를 삽입한다(InsertFirstNewLeafNode). 만일 검색된 단말 노드가 존재하고, 단말 노드에 공간이 있으면 레코드를 삽입하고 공간이 없으면 삽입될 레코드를 포함

하는 새로운 단말 노드를 생성한다. 그리고 새로이 생성된 단말 노드의 부모 노드를 검색하여 단말 노드를 삽입하고(InsertNextNewLeafNode), Predecessor Link의 해당 이동 객체 엔트리를 갱신시킨다. 단말 노드에 레코드가 처음 삽입되거나 삽입되는 레코드의 MBB가 기존의 EMBB를 벗아나면, EMBB를 재설정한다. EMBB 재설정 후 삽입되는 레코드에 대해서는 레코드 세그먼트가 EMBB에 포함되는지를 비교한 후, 기존 EMBB에 포함되지 않으면 AdustTree를 호출하여 MBB 변경을 상위 노드에 전파한다. 따라서 기존 TB-tree 기법보다 MBB의 전파 횟수가 크게 감소하여 갱신 부하가 줄어든다.

[알고리즘 1] Insert

```
Input: Index Entry E
Procedure Insert(E)
BEGIN
Invoke FindNode(E) to select a leaf node L which has a
predecessor of E
IF node L is not found
  Create a new leaf node NL containing E
  Set new EMBB in NL
  Add new entry (E's MOID, NL) to Predecessor Link
  Invoke InsertFirstNewLeafNode(NL)
ELSE
  IF L has room,
    Install E
    IF EI isn't included in L's EMBB
      Set new EMBB in L
      AdjustTree(L)
    END IF
  ELSE
    Create a new leaf node LL containing E
    Set new EMBB in LL
    Update entry of Predecessor Link which has E's MOID
    Invoke InsertNextNewLeafNode(L, LL)
  END IF
END IF
END
```

2.3 성능 평가

본 장에서는 제안 기법인 최소 전파 TB-tree와 기존 TB-tree의 갱신(삽입), 검색 성능을 실험을 통하여 평가한다

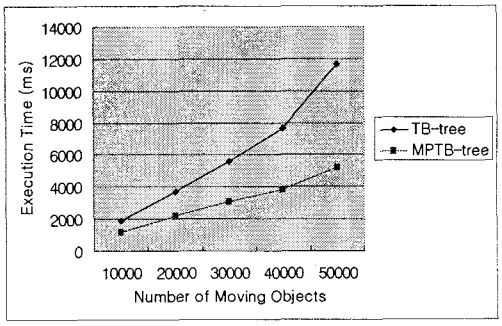
2.3.1 평가 환경

성능 평가를 위하여 실제 이동 객체에 대한 데이터 셋이 공개되어있지 않기 때문에, CitySimulator를 이용하여 데이터 셋을 생성하였으며, 단말 노드와 비단말 노드를 위한 인덱스 페이지의 크기는 1KB로 설정하였다. 기존 TB-tree의 비단말 노드와 단말 노드의 팬아웃은 36, 31이고, 최소 전파 TB-tree의 비단말 노드와 단말 노드의 팬아웃은 36, 30 이다[10].

2.3.2 평가

가) 갱신 연산

제안 기법과 기존 TB-tree의 삽입 연산 성능을 평가하였다. MEC 값은 단말 노드 최대 엔트리 개수의 20%, 제한 속도(RV: Restricted Velocity)는 60km로 하여 평가하였다.



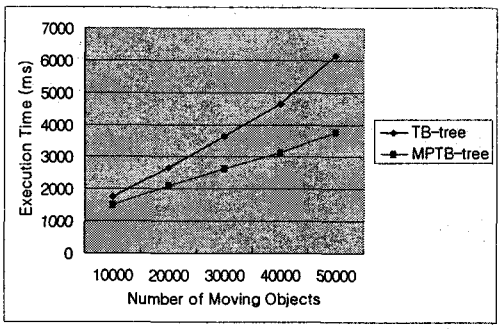
[그림 7] 삽입 연산

실험 결과에서 제안 기법이 기존 TB-tree보다 나은 삽입 성능을 나타내는 것을 볼 수 있다. 특히, 이동 객체의 수가 증가함에 따라 제안 기법이 기존 기법보다 더 좋은 결과를 보여주는데, 이것은 이동 객체의 수가 증가하게 되면, 기존 기법에서는 MBB 변경 전파 횟수와 높은 비용의 FindNode의 호출이 급격히 증가하기 때문이다. 그리고 기존 기법은 이동 객체의 수가 40000~50000개 사이에서 갱신 부하가 크게 증가됨을 실험을 통해서 볼 수 있는데, 이것은 색인 레벨의 증가로 인해 MBB를 재조정해주어야 하는 노드 수가 증가하기 때문이다. 실험 결과에서 50000개 이동 객체의 레코드 삽입 연산시, 제안 기법이 기존 기법보다 약 50% 정도까지 빠른 응답 시간을 나타내고 있다.

나) 검색 연산

본 절에서는 제안 기법과 기존 TB-tree의 레코드 삽입 과정 중 검색 연산 성능을 평가하였다. 검색 질의는 영역

질의로 한정하였으며, 질의 영역을 각 차원당 5%로 고정하였다.



[그림 7] 갱신 연산

실험 결과에서 제안 기법이 기존 기법보다 삽입 과정 중 검색 응답 시간이 더 나음을 보여준다. 이는 제안 기법의 갱신 부하가 기존 기법보다 감소되어 갱신 연산이 빠르게 수행되고, 기존 기법보다 갱신 연산으로 인한 시스템 자원이 적게 소모되기 때문이다. 이동 객체의 수가 증가하고 위치 보고 주기 짧아지게 되면 더욱 갱신 부하가 증가하게 되는데, 실험 결과에서 이동 객체의 수가 증가할수록 제안 기법의 검색 응답 시간이 더욱 빠름을 보여준다. 실험 결과에서 50000개의 이동 객체의 삽입 과정 중 제안 기법의 검색 성능이 기존 기법보다 약 40% 정도까지 향상됨을 볼 수 있다.

3. 결 론

전통적인 R-tree 기반의 TB-tree는 레코드가 삽입될 때마다 레코드의 삽입으로 인한 MBB의 변경을 전파해 주어야하는 하는 부하와 엄격한 궤적 보존 정책에 의해 삽입될 레코드의 선행자를 포함하는 단말 노드를 검색해야하는 부하를 가지고 있어 실시간 대용량 이동 객체 위치 정보 색인에는 적합하지 않다.

따라서, 본 논문에서는 이러한 TB-tree의 갱신 부하를 고려한 최소 전파 TB-tree를 제안한다. 본 기법은 별도의 Predecessor Link 구조를 유지하여, 기존 TB-tree의 FindNode 비용을 줄이고, EMBB를 이용하여 MBB 변경의 전파 횟수를 감소시켜 TB-tree의 전체적인 갱신 연산 성능을 향상시킨다. 본 기법은 갱신 부하의 감소로 실시간 레코드의 삽입과 검색 연산에 적합하다. 향후 연구로 이동 객체의 다양한 속성을 고려한 보다 효율적인 EMBB 설정 방법에 대한 연구와 실험이 필요하다.

참고 문헌

[1] A. Guttman, "R-Trees: A Dynamic Index Structure for Spatial Searching", In In Proc. of the 1984 ACM SIGDB Int'l. Conf. on Management of Data, 1984

[2] 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

[3] M. A. Nascimento, J. R. O. Silva, and Y. Theodoridis, "Evaluation of Access Structures for Discretely Moving Point", Spatio-Temporal Database Management, 1999

[4] D. Pfoser, Christian S. Jensen, Yannis Theodoridis, "Novel Approaches to the Indexing of Moving Objects", In Proc, of the 26th VLDB Conf, 2000

[5] D. Pfoser. Y. Theodoridis, and C. S. Jensen, "Indexing Trajectories of Moving Point Objects", Chorochronos Technical Report, CH-99-3, Octorber, 1999

[6] D. Pfoser, "Issues in Management of Moving Point Objects", Ph. D. thesis, Department of Computer Science, Aalborg University, Denmark, 2000

[7] Y. Theodoridis, Micheal Vazirgiannis, Timos Sellis, "Spatio-Temporal Indexing for Large Multimedia Applications", In International Conf, on Multimedia Computing and Systems, 1996

[8] O. Wolfson, B. Xu, S. Chamberlania, L. Jiang. "Moving objects databases: Issues and solutions" In Proc. of 10th Int'l. Conf. on Scientific and Statistical Database Management, 1998

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

[10] http://www.alphaworks.ibm.com/tech/citysimulator