효율적인 이동 객체의 궤적 색인을 위한 TB-tree 갱신 기법

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

TB-tree Update Technique for Efficient Indexing Trajectories of Moving Objects

Juil Ko*○, Myung-Keun Kim*, Warnil 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 위치가 연속적으로 변경되는 객체를 이동 객체(Moving Objects)라고 한다. 이러한 이동 객체의 대용량 궤적 정보를 효율적으로 검색하기 위해서 색인이 필요하며 대표적인 색인으로 TB-tree가 있다. 그러나 전통적인 공간 색인인 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 해당 단말 노드에서부터 루트 노드까지 반영해야하는 갱신 부하를 가지고 있다.

본 논문에서는 대용량 궤적 정보의 효율적인 색인을 위한 TB-tree 갱신 기법을 제안한다. 본 기법은 앞으로 삽입될 이동 객체의 궤적을 포함하는 예상된 MBB(EMBB: Expected Minimum Bounding Box)를 트리에 먼저 반영한다. 그 후 새로운 레코드가 삽입될 때마다 중간 노드의 MBB를 갱신하지 않고, 삽입되는 레코드의 MBB가 EMBB을 벗어났을 때 EMBB를 재설정하여 실제로 삽입된 레코드의 MBB와 재설정된 EMBB를 포함하도록 중간 노드의 MBB를 조정하므로 TB-tree의 MBB 조정 횟수를 줄인다. 또한 TB-tree에 선행자를 포함하는 단말 노드를 직접적(direct)으로 접근하기 위하여 별도의 선행자 테이블(Predecessor Table) 구조를 두어 레코드 삽입을 위해 선행자를 포함하는 단말 노드의 검색 비용을 줄여 전체적인 색인 갱신 비용이 감소된다.

1. 서 론

최근 PDA, 휴대폰 등과 같은 이동 기기 기술이 빠르게 발전하여 그 가격이 저렴해짐에 따라 이동 기기들이 일반 사용자들에게 널리 보급되고 있다. 또한 GPS(Global Positioning System) 기술 및 무선 통신 기술의 발달로 측위 기술이 크게 발전하여 일상생활에서도 쉽게 위치 추적 기능의 이용이 가능하게 되어 위치 기반 서비스(LBS: Location Based Service)의 기반을 조성하게 되었다.

위치 기반 서비스에서는 시간에 따라 위치정보가 계속해서 변하는 이동 객체를 효율적으로 저장, 관리 및 검색하기 위해서 이동체 데이터베이스(Moving Objects Database)가 필요하다[4]. 전통적인 데이터베이스 시스템과 다르게 이동체 데이터베이스에서는 이동 객체의 지속적인 위치 정보의 변경으로 인한 빈번한 갱신 연산이 발생한다. 특히 이동 객체는 지속적으로 그 위치를 변경하기 때문에 데이터베이스가 처리해 주어야 하는 갱신 연산은 이동 객체의 수가 증가함에 따라 급격히 증가하게 된다[3]. 이런 이동 객체의 위치 정보를 효율적으로 관리하기 위한 많은 색인 기법들이 연구되었다. 하지만 이들은 공간 색인을 위한 전통적인 R-tree 기반의 색인 기법으로 데이터의 갱신 연산을 위해서 단지 데이터만 변경해주는 것이 아니라 데이터의 변경에 따른 인덱스 노드들의 분할, 합병 및 키(MBR) 값 조정 등의 트리의 구조적인 변경이 필요하다[1]. 기존의 GIS 응용이나 공간 데이터베이스 시스템에서 공간 데이터는 한번 입력을 받으면 크게 변하지 않고, 지속적인 삽입 요청이 발생하지 않으므로 R-tree 기반의 삽입, 삭제 등의 색인 갱신 부하는 크게 고려되고 있지 않지만, 연속적으로 움직이는 이동 객체 위치 정보의 빈번한 갱신을 R-tree 기반으로 색인 하기에는 큰 부하가 따른다.

이러한 문제를 해결하기 위하여 본 논문에서는 지속적으로 발생하는 이동 객체의 대용량 궤적 정보를 효율적으로 색인하기 위한 TB-tree 갱신 기법을 제안한다. 제안 기법은 이동 객체가 이동함에 따라 증가되는 예상된 MBB(EMBB: Expected MBB)와 새로이 삽입되는 이동 객체 레코드의 MBB를 고려하여 갱신 연산을 수행하고, TB-tree에 Predecessor Table이라는 별도의 테이블 구조를 둠으로써 레코드 삽입으로 인한 전반적인 색인 갱신 비용을 감소시킨다.

본 논문의 구성은 다음과 같다. 2장에서는 관련 연구를 간단히 살펴보고, 3장에서는 본 논문에서 제안한 갱신 기법을 소개한다. 마지막으로 4장에서 결론 및 향후 연구 방향을 기술한다.

2. 관련 연구

이동체 데이터베이스 시스템에서 이동 객체의 위치 정보를 효율적으로 저장, 관리 및 검색하기 위해서 시공간 색인이 필요하다. 기존의 이동 객체 색인에 관한 연구는 처리하는 데이터에 따라 크게 2가지로 분류된다. 첫째는 이동 객체의 과거 이력 정보 및 궤적의 색인에 관한 연구로 3DR-tree, STR-tree, TB-tree가 이에 속한다[2][6]. 3DR-tree는 시간 차원을 고려한 R-tree로 영역 질의에 우수하며, STR-tree는 이동 객체의 궤적 보호 및 공간 속성을 고려한 색인 구조이다. TB-tree는 하나의 단말 노드에 동일한 이동 객체의 세그먼트만 저장되도록 하는 엄격한 궤적 보존 정책으로 궤적 질의 특히 복합질의에 우수한 색인이다. 두 번째는 이동 객체의 현재 위치와 가까운 미래 위치의 색인에 관한 연구로, TPR-tree가 이에 속한다[5]. TPR-tree는 이동 객체의 위치를 단순한 선형 함수로 가정하고 객체의 속도와 방향만을 저장하는 기법이다. 이들 이동 객체 색인의 대부분은 R-tree에 기반한 구조이기 때문에 데이터의 갱신에 따른 트리 구조의 변경이라는 부하를 그대로 가지고 있다.

최근 이런 이동 객체 위치 정보의 갱신 비용을 줄이기 위하여 해쉬를 이용한 기법과 R-tree 지연 갱신 기법이 제안되었다[7][3]. 하지만 이들은 이동 객체의 현재 위치만을 색인하는 구조에서 갱신 비용을 줄이는 기법이다.

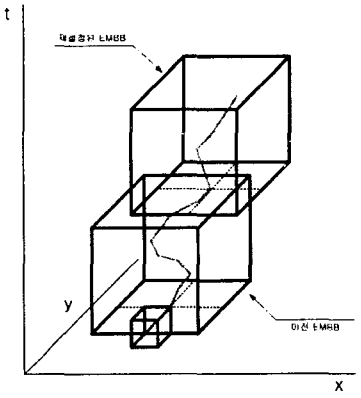
3. 제안 기법

3.1 효율적인 TB-tree 갱신 기법

관련 연구에서 설명하였듯이 R-tree 기반의 이동 객체 색인 기법들은 데이터의 갱신 부하라는 문제점을 가지고 있다. 이런 갱신 부하를 줄이기 위해 최근 여러 기법들이 제안되었지만 이들은 이동 객체의 현재 위치만을 색인하기 위한 구조에서 부하를 해결한 기법들이다. 하지

만 위치 기반 서비스에서는 이동 객체의 현재 위치뿐만이 아니라 과거 위치 정보(궤적) 역시 중요하다. 따라서 본 논문에서는 이동 객체의 현재 위치뿐만 아니라 과거 위치에 대한 검색도 가능한 TB-tree에서의 갱신 비용을 줄이기 위한 효율적인 색인 갱신 기법을 제안한다.

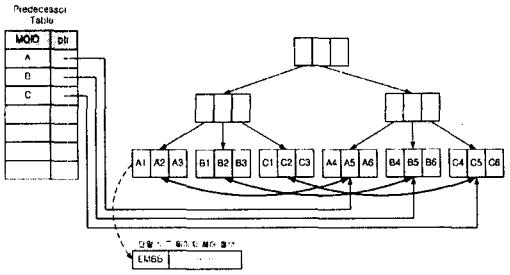
이동 객체의 궤적 색인을 위한 대표적인 색인 구조인 TB-tree 역시 R-tree 기반의 색인으로 삽입 연산시 크게 두 가지의 부하를 가지고 있다. 첫째로, 레코드를 삽입할 때 삽입되는 레코드의 선행자(predecessor)를 포함하는 단말 노드를 찾는 FindNode 알고리즘이다[2]. 이 알고리즘은 삽입될 레코드 세그먼트(E)의 MBB와 겹치는 단말 노드 중 E와 연결(connected)되는 엔트리를 포함하는 단말 노드를 반환한다. 따라서 중간 노드에서 E와 겹치는 모든 서브 트리들이 검색 대상이 되므로 레코드 삽입 시 부하로 작용한다. 본 논문에서는 TB-tree에 Predecessor Table을 추가적으로 두어 선행자를 포함하는 단말 노드에 대한 빠른 검색이 가능하도록 하여 레코드 삽입 비용을 줄인다. TB-tree의 두 번째 부하는 레코드 삽입시 단말 노드로부터 루트 노드까지의 모든 MBB를 재조정해 주어야 한다는 것이다. 특히 이동 객체의 수가 많아지고 지속적으로 위치를 변경하게 되면 MBB 조정 부하가 급격히 증가하게 된다. 따라서 본 논문은 예상된 MBB(EMBB: Expected MBB) 삽입을 통한 MBB 조정 기법을 제안한다. [그림 1]에서 굵은 검정색 실선으로 표현된 것이 EMBB이다. 이동 객체의 이동에 따라 처음 세그먼트 단위의 레코드가 삽입될 때 EMBB를 설정하여 중간 노드에 반영을 한다. 그 후에 삽입되는 세그먼트 레코드들이 EMBB 내에 존재할 경우에는 상위 중간 노드의 MBB 조정 과정이 필요 없고, 단지 EMBB를 벗어날 경우에만 EMBB를 다시 재설정하여 중간 노드의 MBB에 새로운 EMBB 반영하여 개개의 레코드 삽입시의 MBB 갱신 부하를 줄이게 된다. 하지만 EMBB로 인한 사장 공간이 발생할 수 있게 되는데 이것은 시간이 지남에 따라 과거 궤적 정보는 실제 레코드의 MBB로 재조정되고 단지 최근에 삽입되는 레코드에 대해서만 사장 공간이 발생하므로 기존의 궤적 질의 처리 비용이 크게 증가 되지는 않는다.



[그림 1] TB-tree 지연 갱신 구조

3.2 인덱스 구조

본 논문에서는 제안 기법을 위해서 수정된 TB-tree 인덱스 구조를 제안한다. 제안된 인덱스 구조는 기존의 TB-tree와 대부분 유사하다. 단지 선행자를 포함하는 단말 노드를 직접적(direct)으로 접근하기 위해 Predecessor Table을 유지하고, EMBB 값을 저장하기 위하여 단말 노드 헤더가 확장된다. Predecessor Table을 위해서는 기존의 해쉬나 B-tree 인덱스가 사용될 수 있다. Predecessor Table의 키는 이동 객체의 아이디(MOID)이고, ptr은 선행자를 포함하는 최근에 해당 이동 객체의 레코드가 삽입된 단말 노드를 가리키는 포인터이다. [그림 2]가 제안된 인덱스 구조를 설명하고 있다. 삽입될 레코드가 (MOID, (x_{begin}, y_{begin}, t_{begin}), (x_{end}, y_{end}, t_{end}))로 구성되어 있다고 하면, 레코드 삽입시 MOID 값으로 Predecessor Table을 통하여 기존의 삽입되는 레코드와 겹치는 모든 서브트리를 검색하는 방식 보다 빠르게 단말 노드를 검색하여 해당 단말 노드에 레코드를 삽입하게 된다.

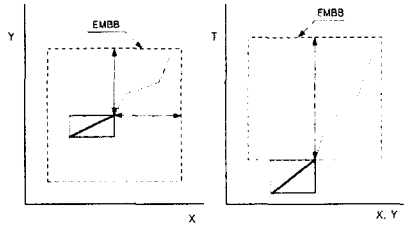


[그림 2] 수정된 TB-tree 인덱스 구조

제안 기법을 위하여 단말 노드 헤더에 EMBB 필드가 추가된다. 단말 노드는 최근에 설정된 EMBB를 유지하고 있어서 삽입되는 레코드의 MBB가 EMBB를 벗어날 때 새로운 EMBB를 재설정하여 저장한다.

3.3 EMBB(Expected MBB)

본 논문의 제안 기법인 EMBB를 통한 MBB 조정 갱신 기법의 기본 아이디어는 삽입될 궤적 레코드들을 포함하는 EMBB를 미리 트리에 반영함으로써 EMBB에 속한 궤적 레코드가 삽입될 때 트리의 MBB 조정 과정을 줄이는 것이다. 따라서 EMBB를 효과적으로 주는 것이 매우 중요하다. EMBB 설정 방법은 [그림 3]과 같이 삽입된 이동 객체의 세그먼트의 끝점에서 고정된 일정 영역을 EMBB 값으로 주는 고정 EMBB 설정 방법과 레코드의 속도와 방향을 고려하여 EMBB를 설정하는 가변 EMBB 설정 방법이 있다.



[그림 3] Expected MBB

3.4 알고리즘

제안 기법의 구체적인 알고리즘은 [알고리즘 1]과 같다. 이동 객체의 레코드 삽입 시, 레코드가 삽입될 단말 노드를 찾기 위하여 FindNode를 호출한다. FindNode는 Predecessor Table을 검색하여 해당 이동 객체의 레코드가 삽입될 단말 노드를 빠르게 검색한다. 만일 단말 노드가 존재하지 않으면 새로운 단말 노드를 생성하고, ChooseNodeForNewMO를 호출하여 생성된 단말 노드의 부모 노드를 찾아 삽입한다. FindNode를 통하여 단말 노드가 검색되면, 단말 노드에 새로운 레코드를 위한 공간이 있는지를 검사한다. 있다면 단말 노드에 레코드를 삽입하고 레코드의 MBB가 EMBB를 벗어나거나, 단말 노드가 가득찰 경우에만 AdjustTree를 호출한다. 새로운 레코드 삽입으로 단말 노드가 가득찰 경우 단말 노드에 존재하는 실제 레코드의 MBB만을 포함하는 MBB를 중간 노드에 반영하여 사장 공간으로 인한 검색 비용 증가를 예방한다. 그리고 레코드의 MBB가 EMBB를 벗어날 경우에 새로운 EMBB를 설정하여 AdjustTree를 호출함으로 중간 노드의 MBB 값을 조정한다. 만일 검색된 단말 노드에 레코드 삽입을 위한 공간이 없을 경우는 기존 TB-tree와 동일하게 새로운 단말 노드를 생성하고 right-most path 방식으로 부모 노드를 찾아 삽입한다. 제안 기법을 위해서 전통적인 R-tree의 SplitNode와 AdjustTree 알고리즘을 수정하였다. SplitNode는 노드의 엔트리들을 재분배하지 않고 단지 새로운 노드를 생성하고 들어온 엔트리만을 삽입하여 생성된 노드를 반환한다. AdjustTree 알고리즘은 기존의 알고리즘과 대부분 유사하고, 단지 삽입될 레코드의 MBB가 EMBB를 벗어날 경우 다시 설정된 EMBB를 포함하도록 중간 노드의 MBB를 조정한다.

```
Algorithm Insert(N, E)
INS1  Invoke FindNode(E) to select a leaf node L which has a
      predecessor of E.
INS2  IF node L is not found,
        Create a new leaf node NL containing E,
        Set new EMBB.
        Create a new entry EN with ENp pointing to NL and ENl enclosing
        all rectangles and EMBB in NL,
        Invoke ChooseNodeForNewMO(NL) to select non-leaf node(level
        1) P which is parent of NL,
        IF P has room,
          Install EN,
        ELSE
          Invoke SplitNode(EN) to produce PP,
          AdjustTree(P, PP),
        IF node split propagation caused the root to split,
          Create a new root whose children are the two resulting
          nodes.
        return.
INS3  IF L has room,
        Insert E.
        IF L is full OR E's MBB isn't contained in L's EMBB,
          Set new L's EMBB,
          AdjustTree(L).
      ELSE
        Create a new leaf node NL containing E,
        Set new EMBB,
        Create a new entry EN with ENp pointing to NL and ENl enclosing
        all rectangles and EMBB in NL,
        Invoke ChooseNodeForOverflow(L) to select non-leaf node(level
        1) P which is parent of NL,
        IF P has room
          Install EN
        ELSE
          Invoke SplitNode(P) to produce P and PP containing all P's
          old entries and EN
        AdjustTree(P, PP)
        IF node split propagation caused the root to split,
          Create a new root whose children are the two resulting
          nodes.
      return.

Algorithm SplitNode(E)
SN1   Create a new non-leaf node N containing E,
      return N

Algorithm FindeNode(E)
FN1   Searches Predecessor Table for E's MOID
FN2   IF MOID is found,
        return leaf node is indicated by ptr of the Predecessor
        Table's entry.
      ELSE
        return NULL.

Algorithm ChooseNodeForOverflow(L)
CNO1  IF N is the root
        Create a new root node NR,
        Create a new entry EN with ENp pointing to N and
        ENl enclosing all rectangles in N,
        Install EN in NR,
        return NR,
CNO2  Set N to be the parent node of N
CNO3  IF N has room OR N is the root node
        Invoke ChooseRightMostPath(N) to select non-leaf
        node(level1) P,
        return P.
      ELSE
        repeat from CNO2

Algorithm ChooseNodeForNewMO(L)
CNNM1 Set N to be the root
CNNM2 IF N is a leaf node
        Create a new root node NR,
        Create a new entry EN with ENp pointing to L and ENl
        enclosing all rectangles in L,
        Install EN in NR,
        return NR.
CNNM3 IF N is level 1,
        return N
CNNM4 Let F be the entry in N whose rectangle Fi needs least
      enlargement to include L,
CNNM5 Set N to be the child node pointed to by Fp and repeat from
      CNM3.

Algorithm ChooseRightMostPaht(L)
CRMP1 Set N = L
CRMP2 IF N is level 1,
        return N,
CRMP3 Choose the right-most entry E
CRMP4 Set N to be the child node pointed to by Ep and repeat
      from CRMP2
```

```
Algorithm AdjustTree(L, LL)
AT1   Set N = L
      IF L was split previously,
        Set NN = LL
      IF N is a leaf node AND N isn't full,
        Let P be the parent node of N, and let EN be N's entry in P,
        Adjust ENl so that it tightly encloses all entry rectangle and
        EMBB in N.
AT2   IF N is the root,
        return.
AT3   Let P be the parent node of N, and let EN be N's entry in P,
      Adjust ENl so that it tightly encloses all entry rectangle in N.
AT4   IF N has a partner NN resulting from an eailer split,
        create a new entry ENN with ENNp pointing to NN and ENNl
        enclosing all rectangles in NN,
        IF P has room
          add ENN to P
        ELSE
          Invoke SplitNode(ENN) to produce PP
AT5   Set N = P
      IF a split occured,
        Set NN = PP
      repeat from AT2
```

[알고리즘 1] EMBB를 이용한 MBB 지연 갱신 알고리즘

4. 결론

이동 객체의 위치가 지속적으로 변경됨에 따라 이동 객체의 궤적을 저장하는 TB-tree의 경우 삽입 연산이 지속적으로 발생한다. 이때 레코드가 삽입될 때마다 중간 노드의 MBB 조정 및 선행자를 포함하고 있는 단말 노드를 찾아야 하는 부하가 발생한다. 이 문제를 해결하기 위해서 본 논문은 EMBB를 이용한 MBB 지연 갱신 기법과 수정된 TB-tree 인덱스 구조를 제안한다. EMBB를 통하여 레코드가 삽입될 때마다 발생하는 MBB 조정을 감소시키고, Predecessor Table을 유지하여 선행자를 포함하는 단말 노드에 대한 빠른 검색이 가능하게 하여 삽입 부하를 감소시킨다. 또한 노드 헤더에 TMBB, TAC 등의 값을 유지하여 EMBB에 의한 사장공간을 감소시킨다.

향후 연구로 실험을 통한 효율적인 EMBB 값 설정 및 본 기법의 성능평가가 필요하다.

참고문헌

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

[3] 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

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

[5] S. Saltenis, C. Jensen, S. Leutenegger and M. Lopez. "Indexing the Positions of Continuously Moving Objects", In Proc. of the 19th ACM SIGMOD Int. Conf. on Management of Data, Dallas Texas, 2000

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

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

[8] 류근호, 안윤애, 이준욱, 이용준, "이동 객체 데이터베이스와 위치 기반 서비스의 적용", 데이터베이스 연구회지, VOL. 17 NO. 2001