LUR-tree에서 이동체의 지연 다량 삽입 기법

°김정현, 장용일, 배해영
인하대학교 컴퓨터 · 정보공학과
{k820230, himalia}@dblab.inha.ac.kr, hybae@inha.ac.kr

Lazy Bulk Insertion Method of Moving objects on LUR-tree

°Jung-Hyun Kim*, Yong-Il Jang*, Soon-Young Park*, Young-Hwan Oh**, Hae-Young Bae*
*Dept. of Computer Science & Information Engineering, Inha University
**Dept. of Information & Communication, Korea Nazarene University

요 약

지금까지의 이동체 인덱스에 대한 연구는 주로 인덱스 구성 후에 발생하는 질의 처리 효율성에 두고 있다. 다수의 이동체 인덱스에서 이동체 데이터의 갱신 연산에 의한 인덱스 재구성에 대한 디스크 접근 오버헤드를 고려하지 않았다. 이동체 데이터 처리를 위한 대표적 인덱스 구조인 R-tree는 이동체에 대한 갱신 연산 비용이 많이 든다. 이런 R-tree의 단점을 보완하기 위해 이동체가 가지는 MBR값이 동적으로 변화하는 환경에 맞추어 R트리의 갱신 비용을 절감하여 처리하는 LUR-tree가 제안되었다.

본 논문에서는 빠른 데이터 생성 속도에 적합하도록 디스크 접근 오버헤드를 고려해서 LUR-tree를 관리할 수 있는 현재 인덱스에 대한 다량 삽입 기법을 제안한다. 이 기법에서는 다차원 인덱스 구조에서의 다량 삽입 기법을 위한 간단한 버퍼링 기법을 사용한다. LUR-tree의 단말 노드 정보를 관리하는 보조 인덱스를 추가하여 갱신 연산에 따른 노드의 분할과 합병을 예측한다. 예측된 결과를 바탕으로 노드의 변화를 최소화하는 방향으로 데이터의 갱신 순서를 정하여 데이터 갱신에 따른 노드의 분할과 합병을 최소화한다. 실험을 통해 제안한 기법을 이용한 다량 삽입이 기존의 다량 삽입 기법들과 비교해 인덱스의 갱신 비용을 감소시키는 것을 알 수 있다.

1. 서론

이동체 데이터를 다루는 이동체 데이터베이스는 매우 가변적이고, 대량인 데이터를 효율적으로 처리할 수 있어야 한다. 하나의 갱신을 이동체 인덱스에 반영하기 위해서, 일반적으로 디스크의 여러 페이지에 대한 접근이 요구된다.

이동체 인덱스에 대한 연구는 인덱스 구성 후에 질의 처리에 효율성을 두었다. 하지만 다수의 이동체 인덱스에서 이동체 데이터의 갱신 연산에 의한 인덱스 재구성에 따른 디스크 접근 횟수에 대한 갱신 오버헤드를 고려하지 않았다. 이를 위해 이미 존재하는 R-tree 인덱스에 다량의 새로운 이동체 데이터를 빠르게 추가하는 다량삽입 문제의 효율적인 해결이 요구된다. 기존 기법들은 새로 추가할 데이터 집합을 몇 개의 클러스터로 분류한 뒤 각 클러스터를 하나의 단위로 해서 한 번에 대상 R-tree에 다량으로 삽입하는 방법을 이용한다. 이 방법에서 하나의 클러스터는 공간상에서 가까운 데이터들의 집합으로 이루어진다. 각각의 클러스터가 공간상에서 작은 영역을 차지하도록 하여 대상 R-tree에 삽입이 되었을 때 삽입이 된 노드 MBR(Minimun Bounding Rectangle) 영역의 확장을 줄여보려는 시도이다. 그러나 R-tree의 구조를 전혀 고려하지 않기 때문에 클러스터가 삽입된 노드의 MBR이 확장하여 다른 노드들과 겹치는 영역의 증가로 인해 질의 성능 저하를 가져온다.

본 논문에서 이동체에 대한 갱신 비용이 많이 드는 R-tree를 개선한 LUR-tree 구조를 기반으로 갱신 연산에 따른 노드의 분할과 합병을 예측하여 전체적인 갱신 비용을 줄이는 갱신 관리 기법을 제안한다. 다량의 데이터를 처리하기 위해 버퍼 구조를 활용하여 갱신되는 이동체 데이터들을 저장한다. 연결 리스트를 활용하여 이동체를 갱신 연산이 인덱스 구조 변화에 영향을 미치는 정도에 따라 두 부분으로 나누어 클러스터링한다. LUR-tree가 가지는 보조 인덱스는 이동체와 이동체가 속한 단말 노드의 정보를 가지고 있어 탐색 시간을 줄인다. 또한 LUR-tree에서 제안하는 확장MBR(Extended MBR)과 갱신 알고리즘은 이동체가 가지는 MBR값이 동적으로 변화하는 환경에 맞추어 R트리의 갱신을 지연시켜 처리한다. 이를 확장하여 LUR-tree가 가지는 보조 인덱스에 단말 노드가 가지는 빈 엔트리 공간에 대한 정보를 저장하는 필드를 추가한다. 이는 연결 리스트에 이동체 데이터를 인덱스에 갱신할 때 예상되는 노드의 분할과 합병을 예측하여 클러스터링에 반영한다.

다량 삽입은 데이터의 삽입 속도는 데이터의 생성 속도를 맞추어야 하는 것은 물론 질의를 효과적으로 처리해야 한다. 제안된 기법을 통한 다량 삽입 기법은 보조 인덱스 사용으로 이동체 탐색 시간과 인덱스 노드의 분할과 합병 수를 줄임으로써 인덱스 재구성 비용이 줄어듦을 실험을 통해 볼 수 있다. 성능 평가를 위해 본 논문에서는 이동체 데이터베이스 분야에서 널리 사용되는 실세계 데이터인 Tiger/Lines 데이터와 세 가지의 합성 데이터를 사용한다.

본 논문은 다음과 같이 구성된다. 2장에서, LUR-tree와 다량 삽입 기법에 대한 관련 연구를 설명한다. 3장에서, 제안 기법인 LUR-tree에서 이동체의 지연 다량 삽입 기법에 대한 내용을 다룬다. 4장에서, 실험 결과를 보이고 그 결과를 논한다. 5장에서, 결론을 내리고 향후 연구를 제시한다.

2. 관련 연구

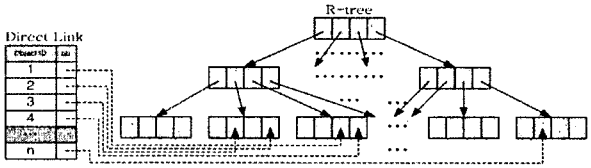
본 장에서는 본 논문에서 제안하는 기법과 관련된 LUR-tree와 다량 삽입 기법에 대한 관련 연구를 설명한다.

2.1 LUR 트리(Lazy Update R-tree)

R-tree와 같은 전통적인 다차원 인덱스는 지속적으로 갱신되는 이동체를 다루는데 갱신 비용이 많이 드는 문제점이 있다[1]. 갱신 연산의 수를 줄이기 위해 간단한 선형 함수를 이용한 많은 방법들이 존재한다. 그러나 이동체의 복잡한 움직임을 표현하기에는 다소 어려움이 있다. 이런 갱신 비용을 줄이고 다량의 이동체에 대한 현재 위치를 위한

본 연구는 대학 IT연구센터 육성 · 지원사업의 연구결과로 수행되었음

인덱스로 제안된 것이 [그림 1]에 나타난 LUR-tree이다[2]. 이동체의 위치를 갱신하는데 불필요한 인덱스의 수정을 줄이고, 이동체가 해당 MBR을 벗어났을 때의 경우에만 인덱스를 갱신한다. LUR-tree의 성능 향상을 위해 이동체를 포함하는 MBR의 크기를 약간 늘린 확장MBR이 제안되었다. LUR-tree의 기본 연산은 R-tree의 기본 연산과 동일하다.



[그림 1] LUR-tree의 구조

2.2 다량 삽입 기법(Bulk Insertion Methods)

이동체가 갱신될 때, 기존의 이동체 인덱스 기법들은 간단하게 하나의 삭제 연산과 하나의 삽입 연산으로 이루어진 갱신 연산을 사용한다. 이러한 일반적인 갱신 방법을 향상시키기 위해서, 만약 두 연산들이 동일한 단말 노드에서 이루어지면, 두 연산 대신 하나의 수정(modify) 연산으로 대신 처리할 수 있다. 그러나 그 갱신 방법은 모든 이동체들을 위한 Direct Link라 불리어지는 디스크 기반의 인덱스를 사용하기 때문에, 상당한 공간 오버헤드를 추가적으로 요구한다. 이러한 단순하고 일반적인 방법들은 하나의 갱신 이동체 인덱스에 반영하기 위해서, 디스크의 여러 페이지 액세스를 요구한다. 따라서 시간이 흐를 때, 빈번한 갱신들을 이동체 인덱스에 반영하기 위해서 상당한 갱신 비용이 요구된다. 다량 삽입은 새롭게 추가된 이동체들을 위해서 이미 존재하는 인덱스에 해당되는 엔트리들을 빠르게 추가하는 것이다.

R-tree에서 다량 삽입에 관한 초기 연구에서는 삽입할 데이터를 먼저 공간상에서의 근접성에 의해 정렬한 후(예:Hibert, Z-order) N개씩 묶어서 블록을 구성한다[3]. 이 블록들을 한 번에 하나씩 변형된 표준 삽입 알고리즘을 이용해 삽입한다. 이 기법은 한번에 N개의 데이터가 삽입되므로 N배의 삽입 속도 향상을 가져온다는 것을 알 수 있다. 그러나 이 블록과 기존 R-tree의 노드 사이의 겹치는 면적은 증가할 가능성이 높고 결과적으로 질의 성능 저하를 가져 올 수 있다.

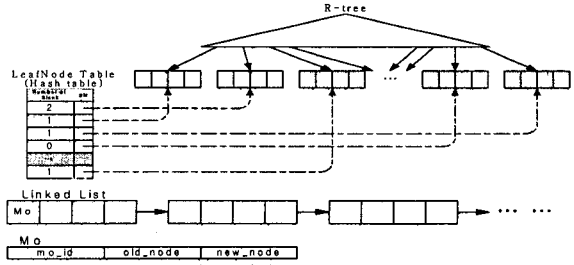
또 다른 다량 삽입에 대한 연구로 입력 데이터로부터 하나의 입력 R-tree(small tree)를 만들고 이것을 대상 R-tree(large tree)에 삽입하는 SLTL(Small-Tree-Large-Tree) 기법이 있다[5]. 그리고 STLT를 응용한 방법으로 입력 데이터 집합을 공간상에서 근접한 데이터끼리 분류하여 여러 개의 클러스터를 생성한 후 각 클러스터로부터 R-tree를 생성하고 마지막으로 이 R-tree들을 대상 트리에 한번에 하나씩 삽입하는 GBI(Generalized Bulk Insertion)기법이 있다[4]. 이들은 대상 R-tree트리의 노드들과 새로 삽입되는 R-tree들이 겹칠 수 있는 문제점이 있다. 노드의 겹침은 하나의 검색을 수행하기 위하여 여러 경로에 있는 트리 노드들을 방문하게 된다.

3. LUR-tree에서의 이동체의 지연 다량 삽입 기법

본 장에서는 제안 기법인 LUR-tree에서의 이동체의 지연 다량 삽입 기법을 정의하고, 이 인덱스에 대한 구조와 다량 삽입 과정에 대해 설명한다.

3.1 LUR-tree의 확장

확장된 LUR-tree의 전체적인 구조는 [그림 2]와 같다.



[그림 2] 확장된 LUR-tree의 구조

각각의 세부적인 구조를 살펴보면 다음과 같다.

LeafNode Table

LUR-tree를 확장하기 위해 본 논문에서 제안된 구조이며, 이는 각각의 단말노드가 가지고 있는 빈 엔트리의 수와 각각의 단말 노드를 가리키는 포인터의 쌍으로 구성된다.

Linked List

이동체들의 속성 정보(Mo)를 가지고 있는 필드로 구성된 연결 리스트 구조이다.

Mo

이동체들의 속성 정보를 가지고 있는 필드이다.
mo_id는 이동체의 ID이다.
old_node는 이동체가 속해 있던 단말 노드의 주소이다.
new_node는 이동체가 갱신되어 속할 단말 노드의 주소이다.

3.2 다량 삽입 기법

본 장에서는 제안하는 기법인 LUR-tree에서의 이동체의 지연 다량 삽입 기법을 위한 버퍼링 기법과 버퍼를 이용한 단말 노드 분석 알고리즘, 이동체 분류와 클러스터링에 살펴본 후 이동체 각각의 분류에 해당하는 갱신 기법과 갱신 연산 처리 순서에 대해 설명한다.

3.2.1 버퍼링 기법

다량의 갱신할 이동체를 버퍼에 모아 놓고 일정 주기에 따라 인덱스에 이들을 반영한다. 다량의 이동체 갱신을 위한 버퍼의 크기는 갱신 이동체의 수, 갱신 연산 수행 시간 간격에 의해 결정된다. 본 논문에서 수행한 실험에서 버퍼의 크기를 이동체가 모두 삽입된 인덱스의 단말 노드 개수의 절반에 해당하는 수만큼의 페이지 크기(페이지의 크기는 4KB로 고정)로 정한다.

3.3.2 단말 노드 분석 알고리즘

버퍼에 있는 이동체를 차례대로 검사한다. Direct Link에서 해당 이동체를 찾아 저장된 단말 노드를 저장한 후, R-tree를 탐색하여 이동체가 갱신될 위치에 해당하는 단말 노드를 저장한다. 버퍼에 있는 모든 이동체들에 대한 탐색을 마치면 저장된 이동체의 정보를 가지고 인덱스의 단말 노드 구조의 변화를 예측한다.

갱신될 노드가 이동체가 속해있는 노드와 일치한다면 LeafNode 테이블의 해당 단말 노드의 빈 공간 수(number of blank)에는 변화를 주지 않는다. 만약 이동체가 속해 있던 노드와 이동체가 속할 일치하지 않으면 LeafNode 테이블에서 이동체가 속해 있던 단말 노드의 빈 공간 수를 하나 늘이고, 이동체가 속할 단말 노드의 빈 공간 수를 하나 줄인다.

갱신될 모든 이동체에 대한 위의 연산을 수행한 결과 LeafNode 테이블에 존재하는 단말 노드의 빈 공간 수를 확인한다. 빈 공간 수가 0보다 작으면 노드 분할이 일어날 노드이며, 주어진 팬-아웃값의 절반을 넘으면 합병이 일어날 노드이다.

3.2.3 이동체 분류에 따른 갱신 기법

- LUR-tree 갱신 기법

LUR-tree에서 제안하는 갱신 기법과 동일하다. 단말 노드에 있는 이동체의 값을 수정하여 인덱스 구조를 변형하지 않는다.

- 교환 갱신 기법

갱신 이동체들 서로 간의 교환으로 인덱스의 구조를 변형을 줄인다. 이동체들을 교환할 수 있는지를 판단하는 기준으로 위에서 설명한 **단말 노드 분석 알고리즘**을 이용한다.

- R-tree 갱신 기법

R-tree에서 제안하는 갱신 기법과 동일하다. 인덱스 노드의 분할과 합병을 요구하는 이동체들의 갱신 연산을 위해 사용한다. 연결 리스트를 이용한 이동체의 갱신은 관련 단말 노드로의 직접 접근이 가능하기 때문에 LUR-tree의 경우보다 갱신을 위한 인덱스 순회 비용을 절감한다.

3.2.4 이동체 분류와 클러스터링

위에서 제시한 단말 노드 분석 알고리즘에 의해 이동체를 분류한다. 이동체의 갱신이 인덱스 구조의 변경에 영향을 미치는 정도에 따라 이동체를 아래와 같이 구분한다.

a. 이동체가 변경되어 속할 노드가 이동체가 속해있던 노드와 일치하거나 이동체가 속해있던 노드를 확장한 확장MBR안에 이동체가 포함될 수 있는 경우 (LUR-tree 갱신 기법)

b. a를 만족하지 못하며, 이동체의 갱신이 다른 이동체들과의 교환으로 인덱스 구조의 변경에 영향을 주지 않는 경우 (교환 갱신 기법)

c. b를 만족하지 못하며, 이동체의 갱신이 인덱스 구조의 변경을 반드시 필요로 하는 경우 (R-tree 갱신 기법)

a는 갱신될 노드가 이동체가 속해있는 노드와 일치하기 때문에 이동체의 갱신이 단말 노드의 구조에 변화를 주지 않는 경우이다. b와 c의 경우에는 이동체가 속해 있던 노드와 갱신될 노드가 일치하지 않기 때문에 갱신이 단말 노드의 구조의 변화를 필요로 한다. 이런 경우 b에 속하는 이동체들끼리 묶어 b가 가지는 연결 리스트([그림 2] 참조)로 저장한다. c의 경우에도 마찬가지로 해당 이동체들을 묶어서 c가 가지는 연결 리스트로 저장한다.

3.2.5 갱신 연산 처리 순서

갱신 연산은 인덱스 구조의 변경을 요구하지 않는 갱신 연산부터 수행한 후에 인덱스 구조의 변경을 가하는 갱신 연산을 수행한다. 인덱스 구조의 변경에 영향을 주지 않는 갱신 연산으로 LUR-tree 갱신 기법과 교환 갱신 기법이 있다. 인덱스 구조의 변경에 영향을 주는 갱신 연산은 R-tree 갱신 기법이 있다.

LUR-tree 갱신 기법의 경우 단말 노드 분석을 실행함과 동시에 수행하여 가장 먼저 처리된다. 교환 갱신 기법과 R-tree 갱신 기법의 경우 공통적으로 각각에 대하여 생성된 연결 리스트를 이용한다. 교환 갱신 기법을 위한 연결 리스트에 저장된 이동체의 갱신을 수행한 후에 R-tree 갱신 기법을 위한 연결 리스트에 저장된 이동체의 갱신을 수행한다. 이러한 연산 순서를 이용함으로써 인덱스 구조의 변형을 최소화하고 전체적인 갱신 비용을 줄인다.

4. 실험

본 장에서는 실험을 통하여 본 논문에서 제시한 다량 삽입 기법을 구현하여 기존의 기법과 비교 분석한다.

4.1 실험 환경

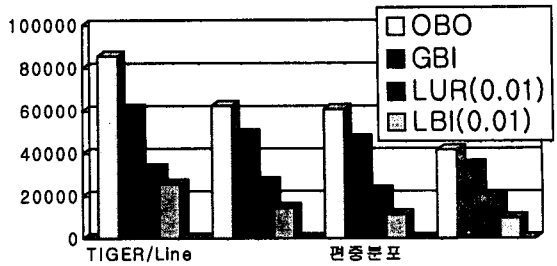
실험은 Pentium 2.0GHz CPU, 1GB의 메모리, 80GB E-IDE HDD를 가진 Windows 2000 OS상에서 수행되었다. 한 노드는 4KB의 디스크 블록에 해당한다. 버퍼의 크기는 50개의 페이지를 저장할 수 있는 크기로 정한다. 성능 비교는 하나의 데이터를 삽입하는 R-tree의 기본 삽입 방식(OBO; One By One), GBI 기법과 Leaf-Update(LUR-tree)기법에 대해 수행하였다. 삽입 비용은 평균적인 디스크 접근 회수로 측정하였다. 본 논문에서 제시한 지연 다량 삽입 기법은 LBI라 표시하겠다. 'Leaf-Update'와 LBI' 뒤의 숫자는 확장MBR의 확장 길이 ε를 의미한다.

실험에 사용할 실제 데이터는 공간 데이터베이스 분야에서 널리 사용되는 표준 벤치마크 데이터인 TIGER/Line 데이터를 이용하여 이동체들의 초기 공간 위치를 고려한다[6]. 이동체의 수는 약 120만개의 개체 정보를 추출하여 사용하였다. 합성 데이터로는 세 개의 서로 다른 분포를 가지는 데이터 집합을 사용하였다. 세 가지 분포는 각각 균등분포, 편중분포 그리고 군집분포이다. 각 데이터 집합은 약 90만개의 데이터를 포함한다.

4.2 갱신 질의 성능

[표 1] 데이터 집합별 삽입 비용(디스크 I/O수)

	LBI(0.01)	LUR(0.01)	GBI	OBO
TIGER/Line	25,798	32,447	60,472	85,586
균등분포	14,586	26,785	48,662	62,363
편중분포	11,112	22,132	46,034	60,585
군집분포	9,883	19,936	34,486	41,381



[그림 3] 데이터 집합별 삽입 비용(디스크 I/O수)

[그림 3]의 실험 결과에서 볼 수 있듯이 본 논문에서 제안하는 LBI가 GBI, OBO나 LUR에 비해 삽입 처리에서 크게 앞서는 것을 알 수 있다. GBI의 경우 클러스터링에 드는 비용이 커서 삽입 비용도 뒤떨어진다. LUR의 경우 위에서 설명한 이동체 분류 b.와 c.의 경우 해당 이동체를 R-tree의 갱신 연산과 동일하게 수행하기 때문에 LBI보다 삽입 비용이 많다.

5. 결론 및 향후 연구

매우 빈번하게 이동하는 이동체 데이터를 다루기 위해, 본 논문에서는 LUR-tree를 이용한 지연 다량 삽입 기법을 제안한다. LUR-tree는 변경된 값과 이전의 값이 연관성을 지니는 변경 요청의 지역성이 존재함을 이용하여 갱신 비용을 줄인다. 다량 삽입 기법은 다량의 갱신들을 버퍼링과 클러스터링을 이용해서 인덱스에 대한 접근 비용을 줄임으로서 전체적인 인덱스의 갱신 비용 줄인다. 또한 제안 기법은 기존의 LUR-tree의 알고리즘을 그대로 이용할 수 있어서 기존 응용 환경에 쉽게 적용할 수 있는 장점이 있다. 본 논문에서는 제안 기법이 기존 기법에 대해여 가지는 갱신 연산 비용의 이득을 실험을 통하여 확인하였다.

향후 연구로서, 다양한 상황에서 버퍼 크기에 따른 성능 비교를 통하여 효율적인 버퍼 크기를 얻어낼 것이다. 본 논문에서 제안하는 기법을 주기억 장치 기반의 환경에 적용하였을 경우 성능의 변화 및 LUR-tree의 다량 삽입 기법 향상을 위한 Bottom-Up방식[7] 적용에 대해 연구하겠다. 마지막으로 다수의 이동체 다량 삽입에 따른 동시성 제어에 대한 연구가 필요하다.

참고 문헌

[1] A. Guttman, "R-tree: a dynamic index structure for spatial searching," ACM SIGMOD, pp. 47~57, 1984.

[2] Dongseop Kwon, Sangjun Lee, Sukho Lee, "Indexing the C-urrent Positions of Moving Object using the Lazy Update R-tree," IEEE MDM '02, pp. 113~120, 2002.

[3] I. Kamel, M. khalil and V. Kouranmajian, "Bulk insertion in dynamic R-trees," SDH '96, pp. 3B.31~3B.42, 1996.

[4] R. Choubey, L. Chen and E. A. Rundeensteiner, "GBI: A Generalized R-tree Bulk-Insertion Strategy," Advances in Spatial Databases, pp. 91~108, 1997.

[5] L. Chen. R, Choubey and E. A. Rundeensteiner, "Bulk-Insertion into R-trees using the small-tree-large-tree approach," ACM GIS, pp. 161~162, 1998.

[6] TIGER/Line Files, 2000 Technical Documentation, U.S. Bureau of Census, Washington DC, accessible via URL http://www.census.gov/geo/www/tiger/tigerua/ua_tgr2k.html

[7] Mong Li Lee, Wynne Hsu, Christian S. Jensen, Bin Cui, Keng Lik Teo, "Supporting Frequent Updates in R-Trees: A Bottom-Up Approach," VLDB, pp. 608~619, 2003.