색인 구조 예측을 통한 이동체의 지연 다량 삽입 기법†

Lazy Bulk Insertion Method of Moving Objects Using Index Structure Estimation

김정현*, 박순영**, 장용일**, 김호석**, 배해영***

Jeong-Hyun Kim, Sun-Young Park, Hyong-Il Jang, Ho-Suk Kim, Hae-Young Bae

요 약 본 논문은 이동체의 잦은 갱신에 의해 발생하는 색인 재구성에 대한 비용을 줄이기 위해 이동체의 지연 다량 삽입 기법을 제안한다. 기존 이동체 색인에 대한 연구는 주로 색인 구성 후에 발생하는 질의 처리 효율성에 초점을 두었다. 이들은 이동체 데이터의 갱신 연산에 의한 색인 재구성에 대한 디스크 접근 오버헤드를 거의 고려하지 않았다. 이러한 이동체 데이터의 갱신 연산에 대한 비용을 줄이기 위해 다량 삽입 기법과 여러 색인이 제안되었다. 하지만 제안된 기법들은 매우 가변적이고 대량인 데이터를 효율적으로 처리하는데 많은 디스크 I/O 비용을 필요로 한다.

본 논문에서는 빠른 데이터 생성 속도에 적합하도록 디스크 접근 오버헤드를 고려해서 R-트리를 관리할 수 있는 현재 색인에 대한 다량 삽입 기법을 제안한다. 이 기법에서는 다차원 색인 구조에서의 다량 삽입 기법을 위한 버퍼링 기법을 사용한다. R-트리의 단말 노드 정보를 관리하는 보조 색인을 추가하여 노드의 분할과 합병을 예상한다. 또한 연산을 종류에 따라 분류하여 불필요한 삽입과 삭제 연산을 줄인다. 노드의 변화를 최소화하는 방향으로 이동 객체의 처리 순서를 정하여 데이터 갱신에 따른 노드의 분할과 합병을 최소화한다. 실험을 통해 제안한 기법을 이용한 다량 삽입 기법이 기존의 삽입 기법들보다 색인의 갱신 비용을 감소시키는 것을 알 수 있다.

Abstract This paper presents a bulk insertion technique for efficiently inserting data items. Traditional moving object database focused on efficient query processing that happens mainly after index building. Traditional index structures rarely considered disk I/O overhead for index rebuilding by inserting data items.

This paper, to solve this problem, describes a new bulk insertion technique which efficiently indexes the current positions of moving objects and reduces update cost greatly. This technique uses buffering technique for bulk insertion in spatial index structures such as R-tree. To analyze split or merge node, we add a secondary index for information management on leaf node of primary index. And operations are classified to reduce unnecessary insertion and deletion. This technique decides processing order of moving objects, which minimize split and merge cost as a result of update operations. Experimental results show that this technique reduces insertion cost as compared with existing insertion techniques.

주요어 : 이동객체, 다차원 색인, 다량 삽입

KeyWords : moving object, multi-dimension index, bulk insertion

1. 서 론

이동체 데이터를 다루는 이동체 색인은 매우 가변적이고 대량인 데이터를 효율적으로 처리할 수 있어야 한다[1]. 시공간 데이터를 다루기 위해 효율적인 색인들이 제안되었다. 가장 간단한 방법으로 다차원

† 본 연구는 대학의 IT 연구센터 육성·지원사업의 연구결과로 수행되었음.

* 인하대학교 컴퓨터정보공학과 석사과정 k820230@dblab.inha.ac.kr

** 인하대학교 컴퓨터정보공학과 박사과정 {sunny, himalia, bluesnow}@dblab.inha.ac.kr

*** 인하대 컴퓨터공학부 교수 hybae@inha.ac.kr

공간 색인을 이용하여 이동체의 위치 값을 저장하는 색인들이 적용되었다[2]. 최근 많은 색인들이 이동체를 다루기 위해 제안되었다. 데이터를 다루는 방법에 따라 크게 두 가지 분류가 존재한다. 객체의 과거 정보 즉, 궤적을 다루는 색인과 현재 위치 색인이 있다. 전자에 대한 색인들은 다차원 공간 정보에 시간을 다루기 위한 차원을 추가하였다. Spatio-Temporal R-트리(STR-트리)와 Trajectory-Bundle 트리 (TB-트리)가 대표적이다[3]. 최근에 제안된 색인으로는 multi- version B-트리와 3D-R트리의 개념을 합쳐 Tao가 제안한 Multiversion 3D R-트리(MV3R-트리)가 있다[4]. 본 논문에서 제안하는 기법은 현재 위치 색인을 위한 것이며, 기존의 현재 위치 색인은 이동체의 현재 위치 변화를 간단한 함수의 변형으로 처리하는 것이 대부분이다. 현재 위치 색인으로는 PMR-Quard트리를 이용한 색인과 벡터를 이용한 time-parameterized R-트리(TPR-트리) 등이 대표적이다[5],[6].

기존 이동체 색인에 대한 연구는 색인 구성 후의 질의 처리 효율성에 초점을 두었다. 하지만 다수의 이동체 색인에서 이동체 데이터의 갱신 연산에 의한 색인 재구성에 따른 디스크 접근 횟수에 대한 갱신 오버헤드를 고려하지 않았다. 이를 위해 이미 존재하는 R-트리 색인에 다량의 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 한번에 대상 R-트리에 다량으로 삽입하는 방법을 이용한다[7]. 이 방법에서 하나의 클러스터는 공간상에서 가까운 데이터들의 집합으로 이루어진다. 각각의 클러스터가 공간상에서 작은 영역을 차지하도록 하여 대상 R-트리에 삽입이 되었을 때 삽입이 된 노드 MBR (Minimun Bounding Rectangle) 영역의 확장을 줄여보려는 시도이다. 그러나 R-트리의 구조를 전혀 고려하지 않기 때문에 클러스터가 삽입된 노드의 MBR이 확장하여 다른 노드들과 겹치는 영역의 증가로 인해 질의 성능 저하를 가져온다.

본 논문에서는 이동체에 대한 갱신 비용이 많이 드는 R-트리를 개선한 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 관리한다. 제안한 OperationResult 테이블을 사용하여 연산이 색인 구조 변화에 영향을 미치는 정도에 따라 이동체를 필터링한다. 다량 삽입 기법에서 사용되는 DirectLink는 이동체가 속한 단말 노드의 정보를 가지고 있어 탐색 시간을 줄인다. 이를 확장하여 DirectLink 외에 단말 노드가 가지는 빈 엔트리 공간에 대한 정보를 저장하는 테이블을 추가한다. 이는 버퍼에 저장된 이동체 데이터를 색인에 갱신할 때 예상되는 노드의 분할과 합병을 예측한다.

다량 삽입은 데이터의 삽입 속도와 데이터의 생성 속도를 맞추어야 하는 것은 물론 질의를 효과적으로 처리해야 한다. 제안된 기법을 통한 다량 삽입 기법은 보조 색인 사용으로 이동체 탐색 시간과 색인 노드의 분할과 합병 수를 줄인다. 노드의 분할과 합병 수를 줄임으로써 색인 재구성 비용이 줄어듦을 실험을 통해 볼 수 있다. 제안한 삽입 기법은 R-트리에 한정되지 않고 LUR-트리, 2-3 TR-트리, Hashing 등의 시공간 색인에 폭 넓게 적용될 수 있다[8],[9],[10].

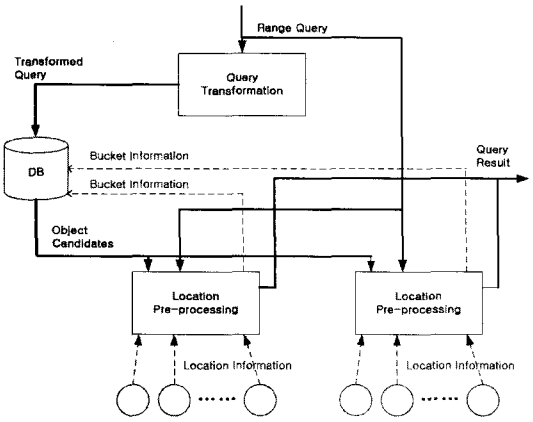
본 논문은 다음과 같이 구성된다. 2장에서, 현재 위치 색인에 대한 관련 연구를 설명한다. 3장에서, 제안 기법인 R-트리에서 이동체의 지연 다량 삽입 기법에 대한 내용을 다룬다. 4장에서, 성능 평가를 보이고 그 결과를 논한다. 5장에서, 결론을 내리고 향후 연구를 제시한다.

2. 관련 연구

본 장에서는 데이터베이스의 갱신 부하를 고려하여 제안된 현재 위치 색인인 Hashing Moving Objects, LUR-트리와 다량 삽입 기법에 대한 관련 연구를 설명한다.

2.1 Hashing Moving Objects

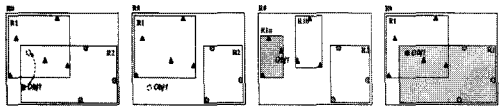
이동 객체는 그 수가 증가하고 연속적으로 움직임에 따라, 많은 데이터베이스의 갱신 연산을 발생시킨다. 이러한 데이터베이스의 갱신을 줄이기 위하여 해싱을 이용한 기법이 제안되었다. 이 기법은 전체 공간을 일정한 개수의 격자로 자르고, 객체가 속한 격자의 번호만을 데이터베이스에 저장한 후, 객체가 이 격자에 속한 동안은 위치의 변경을 수행하지 않고 다른 격자로 이동한 경우에만 데이터베이스에 변경 연산을 수행하여 데이터베이스의 갱신 비용을 감소시킨다. 이 기법의 전체적인 구조는 <그림 1>과 같다.



<그림 1> 해싱 기법의 구조

2.2 LUR-트리(Lazy Update R-트리)

R-트리와 같은 전통적인 다차원 색인은 지속적으로 갱신되는 이동체를 다루는데 갱신 비용이 많이 드는 문제점이 있다. 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 색인으로 제안된 것이 LUR-트리다. 이 기법은 이동 객체의 새로운 위치가 기존 단말 노드의 MBR 범위를 벗어나지 않는 경우, 트리의 구조를 변경하지 않고 단지 노드 내부의 위치만을 변경하여 R-트리에서의 갱신 비용을 크게 감소 시킨다. 기존 R-트리의 갱신 기법과 지연 갱신 R-트리의 갱신 기법의 예가 <그림 2>에서 보여진다.



<그림 2> 기존 R-트리 갱신 기법과 지연 갱신 R-트리 갱신 기법의 수행 예

이동체의 위치를 갱신하는데 불필요한 색인의 수정을 줄이고, 이동체가 해당 MBR을 벗어났을 경우에만 색인을 갱신한다. LUR-트리의 성능 향상을 위해 이동체를 포함하는 MBR의 크기를 약간 늘린 확장MBR이 제안되었다. LUR-트리의 기본 연산은 R-트리의 기본 연산과 동일하다.

2.3 다량 삽입 기법(Bulk Insertion Methods)

이동체가 갱신될 때, 기존의 이동체 색인 기법들은 간단하게 하나의 삭제 연산과 하나의 삽입 연산으로 이루어진 갱신 연산을 사용한다. 이러한 단순하고 일반적인 방법들은 하나의 갱신 이동체를 색인에 반영하기 위해서, 디스크의 여러 페이지 액세스를 요구한다. 따라서 시간이 흐를 때, 빈번한 갱신들을 이동체 색인에 반영하기 위해서 상당한 갱신 비용이 요구된다. 따라서 모든 이동체에 대한 연산을 삭제와 삽입 연산으로 나누어 처리하는 것은 비효율적이다. 만약 두 연산들이 동일한 단말 노드에서 이루어지면 두 연산 대신 하나의 수정(modify) 연산으로 대신 처리하여 일반적인 갱신 방법을 향상시킬 수 있다.

R-트리에서 다량 삽입에 관한 초기 연구에서는 삽입할 데이터를 먼저 공간상에서의 근접성에 의해 정렬한 후(예:Hilbert, Z-order) N개씩 묶어서 블록을 구성한다. 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-트리의 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-트리(small 트리)를 만들고 이것을 대상 R-트리(large 트리)에 삽입하는 STLT기법이 있다[11]. 또한 STLT(Small-Tree-Large-Tree)를 응용한 방법으로 입력 데이터 집합을 공간상에서 근접한 데이터끼리 분류하여 여러 개의 클러스터를 생성한 후 각 클러스터로부터 R-트리를 생성하고 마지막으로 이 R-트리들을 대상 트리에 한번에 하나씩 삽입하는 GBI(Generalized Bulk Insertion)기법이 있다[12]. 이들은 대상 R-트리의 노드들과 새로 삽입되는 R-트리들이 겹칠 수 있는 문제점이 있다. 노드의 겹침은 하나의 검색을 수행하기 위하여 여러 경로에 있는 트리 노드들을 방문하게 된다.

3. 이동체의 지연 다량 삽입 기법

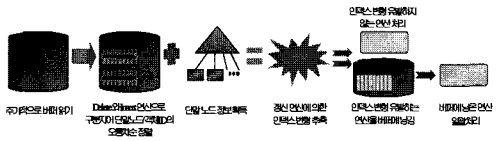
본 장에서는 제안 기법인 이동체의 지연 다량 삽입

기법을 정의하고, 이 색인에 대한 구조와 다량 삽입 과정에 대해 설명한다.

3.1 이동체의 지연 다량 삽입 기법

이동체의 지연 다량 삽입 기법은 다량의 갱신 연산들을 버퍼에 저장하여 일정 주기마다 이를 일괄적으로 처리하는 갱신 기법이다. 일괄적인 연산 처리를 위해 버퍼 정책이 제안되었으며, 검색 효율을 높이기 위해 DirectLink 테이블을 적용하였다. 연산 처리시 색인의 구조에 대한 변화를 예측하고 분석하기 위해서 LeafNode 테이블을 제안한다.

주기적으로 버퍼에 저장된 연산들을 종류에 따라 구분하여 일괄처리에 적합한 Delete/Insert 연산들만을 모아서 정렬한다. 정렬된 연산과 단말 노드에 대한 정보를 함께 분석하며 연산에 의한 색인의 변형을 예측하게 된다. 연산의 결과가 색인을 변형하지 않는 경우 버퍼의 연산을 처리하며, 색인의 변형을 유발하는 경우는 연산을 처리하지 않고 버퍼에 남긴다. 만약 주 색인에서 합병을 요구하는 단말 노드가 있을 경우 합병을 모두 수행한 후 버퍼에 남아 있는 나머지 연산들을 모두 처리한다. 본 논문에서 제안하는 지연 다량 삽입의 과정을 그림으로 간단하게 살펴보면 <그림 3>과 같다.

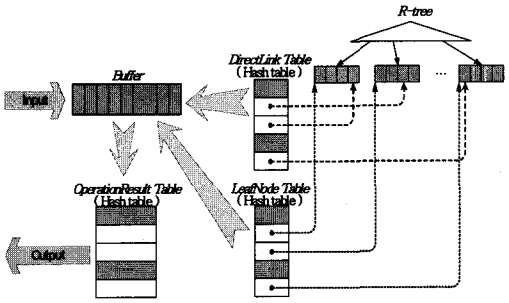


<그림 3> 지연 다량 삽입 기법의 처리 순서

본 논문에서 제안하는 삽입 기법은 연산이 들어올 때마다 색인의 구조를 변형시키는 비효율적인 갱신 비용을 줄이기 위한 것이다. 교환 갱신과 다량의 연산에 대한 일괄적인 처리, 색인의 구조에 대한 예측을 통하여 갱신 비용을 절감할 수 있다. 하지만 버퍼를 이용하여 연산을 정렬하고, 주 색인을 분석하기 위한 보조 색인을 구성하는데 비용이 드는 단점이 있다.

3.2 이동체의 지연 다량 삽입 기법을 위한 색인 구조

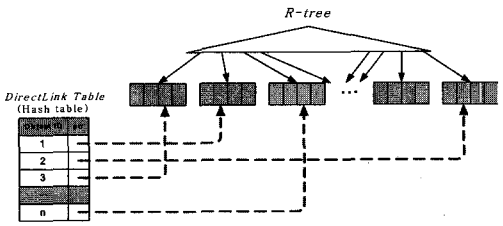
본 논문에서 제안하는 전체적인 색인의 구조를 살펴보면 <그림 4>와 같다.



<그림 4> 지연 다량 삽입 기법의 전체 색인 구성

입력된 질의는 버퍼에 저장되어 주기적으로 처리하게 된다. 버퍼에 저장된 질의들을 처리하기 위해서 DirectLink 테이블과 LeafNode 테이블로부터 객체 정보와 해당 객체 관련 단말 노드에 대한 정보를 얻는다. 처리된 결과는 OperationResult 테이블에 저장되며, 질의 결과에 따른 객체에 대한 정보 변경 및 단말 노드 정보의 변경을 각각 DirectLink 테이블과 LeafNode 테이블에 전달한다. 버퍼 내 질의가 모두 수행된 후에 사용자에게 OperationResult 테이블에 저장된 질의 결과를 반환하게 된다.

본 논문에서는 이동체에 대한 검색 비용을 줄이기 위해서 한번의 해쉬 연산으로 주 색인 내에 위치한 이동 객체의 위치를 얻어낼 수 있는 DirectLink 테이블을 사용한다. DirectLink 테이블에 대한 구조는 <그림 5>와 같다.



<그림 5> R-트리와 DirectLink 테이블

<그림 5>에 대한 각각의 세부적인 구조를 살펴보면 다음과 같다.

R-트리

제안 기법에서는 Guttman의 R-트리를 채택하였으며, 각각의 단말 노드는 고유의 색인 ID를 가지고 있다[13]. 색인 내의 모든 노드는 해당 노드 내의 모든 레코드를 포함하는 MBR을 헤더에 저장하고 있다.

DirectLink 테이블

색인 안에 저장되어 있는 모든 이동 객체에 대한 관리를 위한 구조이다. 각각의 객체에 대한 ID를 키로 하며, 해당 객체가 속한 단말 노드의 포인터를 갖는 해쉬 테이블이다. DirectLink 테이블은 주 색인 내에 객체가 존재하는지 여부 확인과 주 색인 내 해당 객체의 위치를 파악하는데 사용된다.

이동체의 지연 다량 삽입 기법에서 제안하는 단말 노드를 관리하기 위한 테이블의 구조는 <그림 6>와 같다.

ID	Number of Blank	ptr
1	11	●
2	7	●
3	8	●
...	...	...
m	0	●

<그림 6> LeafNode 테이블(Hash 테이블)

LeafNode 테이블

단말 노드를 관리하기 위해 본 논문에서 제안된 구조이다. 이는 각각의 단말 노드 ID를 키로 하고, 각각의 단말 노드가 가지고 있는 빈 엔트리의 수와 해당 단말 노드를 가리키는 포인터의 쌍을 레코드로 하는 해쉬 테이블이다. 단말 노드에 대한 빈공간 수에 대한 정보를 관리하여 갱신 연산에 의한 단말 노드의 변형을 예측한다.

OperationResult 테이블

버퍼 내에 저장된 연산들을 분석하여 해당 객체에 대한 특정 시점에서의 객체의 정보를 변경하는 마지막 연산의 결과를 저장하는 해시 기반 테이블이다.

- ID는 이동체의 ID
- IsExist Flag는 연산의 결과로 이동체가 주 색인 내에 존재하는지 여부를 true/false로 표시
- TimeStamp는 해당 객체의 정보를 변경한 마지막 연산에 대한 타임스탬프 값
- Result는 연산에 의해 변경된 객체의 정보

이동체의 지연 다량 삽입 기법에서 제안하는 OperationResult 테이블의 구조는 <그림 7>과 같다.

ID	IsExist Flag	Time Stamp	Result
1	true	10	(136, 34)
2	false	39	null
3	false	4	null
...	...	...	...
m	true	67	(824, 72)

<그림 7> OperationResult 테이블(Hash 테이블)

OperationResult 테이블은 버퍼에 쌓인 연산들 중에서 search 연산들을 처리할 때 다른 연산에 의한 결과의 순차성을 보장하기 위해 사용된다. search 연산보다 선행되는 연산에 대한 결과를 테이블에 저장하여 객체의 존재 유무와 순차성을 보장하게 되는 것이다. search 연산에서 얻고자 하는 객체에 대한 정보는 Result 필드를 이용하여 얻어낼 수 있다.

3.3 다량 삽입 기법

본 장에서는 제안 기법인 교환갱신 기법과 이동체의 지연 다량 삽입 기법을 위한 버퍼 관리 기법을 이용하여 이동체의 검색 연산을 처리하는 방법을 살펴본 후 갱신 연산 처리 과정에 대해 설명한다.

3.3.1 교환 갱신

교환 갱신은 하나의 객체에 대한 삭제와 삽입 연산을 해당 객체에 대한 레코드 값만을 수정함으로써 색인의 구조의 변형을 줄이는 방법이다. 교환 갱신은 해당 객체가 속해 있던 노드와 수정되어 속하게 될 노드가 동일한 경우에만 수행된다. 이는 색인의 구조를 변형시키지 않을 뿐만 아니라 DirectLink 테이블과 LeafNode 테이블에 대한 수정을 필요로 하지 않아 색인의 갱신 비용을 절감시킬 수 있다.

3.3.2 버퍼 관리

다량의 갱신할 이동체를 버퍼에 모아 놓고 일정 주기에 따라 이들을 분류하여 관리한다. 이동체를 버퍼에 모아 놓을 때 각각에 타임스탬프와 연산의 유효성을 표시할 수 있는 플래그를 붙여 저장한다. 우선 버퍼에 저장된 search연산에 대한 처리를 선행한다. Search 연산의 경우 즉시 해당 객체를 찾아 결과를 반환하며, 버퍼에서 제거한다. 이때 연산의 순차성을 보장하기 위해서 본 논문에서 제안된 OperationResult

테이블을 사용한다. search 연산보다 선행되는 연산에 대한 결과를 테이블에 저장하여 객체의 존재 유무와 순차성을 보장하기 위해 사용된다. search 연산에서 얻고자 하는 객체에 대한 정보는 테이블 내 Result 필드를 이용하여 얻을 수 있다.

특정 객체에 대한 연산을 처리하기 위해서는 먼저 OperationResult 테이블에 해당 객체ID가 있는지 확인한다. 만약 해당 객체에 대한 정보가 들어 있지 않을 경우 DirectLink 테이블을 통하여 해당 객체에 대한 정보를 얻어온다. 이때의 TimeStamp의 값은 버퍼링 주기 내에 가장 작은 값으로 정한다. 만약 객체가 주 색인에 존재하면 IsExistFlag필드를 true로 표시하고, 존재하지 않으면 false로 표시한다. 만약 현재 IsExist Flag필드가 true이고 바로 다음에 들어온 질의가 해당 객체에 대한 insert 연산일 경우, 이 연산의 결과는 failure가 된다. 또한 현재 IsExistFlag필드가 false이고 바로 다음에 들어온 질의가 해당 객체에 대한 delete 또는 update 연산일 경우, 이 연산의 결과도 마찬가지로 failure가 된다. 이처럼 연산의 결과로 failure를 발생시키는 연산들은 OperationResult 테이블을 만드는 과정에서 TimeStamp와 IsExistFlag의 사용으로 버퍼로부터 해당 연산들을 제거할 수 있다. 연산에 의해 OperationResult 테이블이 변경될 경우 이전에 OperationResult 테이블에 저장된 연산은 버퍼에서도 지워지게 된다. Result 필드의 값은 연산에 의해 변경된 값으로 대체된다. 만약 해당 객체에 대한 정보가 존재하지 않을 경우 null로 표시하게 된다. Search 연산의 결과는 OperationResult 테이블의 Result 필드를 참조하여 얻을 수 있다.

이동체를 버퍼에 남겨두는 연산으로는 Delete, Insert, Update 연산이 존재한다. 하지만 이들은 선행된 search 연산 처리 과정에 의해 객체 정보의 변경에 영향을 주는 연산들만 버퍼에 남아있게 된다. 남아있는 연산들에 대해 각각의 연산의 결과가 주 색인의 어느 단말 노드에 영향을 주는지 분석한다. Update 연산의 경우 DirectLink 테이블에서 찾은 단말 노드(삭제될 객체가 속한 단말 노드)와 R-트리를 탐색하여 이동체가 추가 또는 갱신될 위치에 해당하는 단말 노드가 동일하다면 교환 갱신을 사용하여 연산을 수행하며 해당 연산을 버퍼에서 삭제한다. 동일한 단말 노드에서 연산이 이루어지지 않을 경우 Update 연산은 하나의 Delete 연산과 하나의 Insert 연산으로 나누어 처리된다. Update 연산을 나눈 두 연산 중 insert 연산에 대한 TimeStamp값을 해당 Update 연산의 Time-Stamp보다 임의로 크게 두어 delete 연산이 먼저 처리되도록 한다. 이는 Update 연산에 의한 결과를 보장하기 위한 것이다.

Delete 연산의 경우 DirectLink 테이블에서 해당 이동체를 찾아 단말 노드 정보와 함께 버퍼에 저장한다. Insert 연산의 경우 주 색인에서 객체가 삽입될 단말 노드를 찾아 단말 노드 정보와 함께 버퍼에 저장한다.

<알고리즘 1> 버퍼링과 검색 연산 알고리즘

```
Procedure Buffering(oid, newvalue)
    Input : an object's id oid, newvalue newvalue
    Begin
    01 : SetBuffer(oid, newvalue, timestamp);
    02 : for( Until All entries of buffer are read )
    03 :     SetOperationResultTable (oid, calculate
                        _timestamp(), calculate_value());
    04 :     if( VerificateOperation(operation) == true )
    05 :         if( operation == search )
    06 :             SetOperationResultTable(oid, timestamp,
                                            newvalue);
    07 :         RemoveOperationInBuffer(timestamp);
    End
```

버퍼 관리 기법을 이용한 일괄처리 방식은 다수 연산들의 조합에 의해 발생하는 불필요한 연산들을 수행하지 않고 제거할 수 있다. 또한 제안한 버퍼 관리 기법은 대용량의 데이터를 처리하는데 있어 검색연산에 대한 우선 순위를 부여하여 갱신연산에 의한 연산 대기 시간을 줄여준다. 입력되는 연산들을 저장하고 관리하기 위한 버퍼의 공간이 필요하며, 검색 연산 수행 시 연산들에 대한 순차성을 보장하기 위해 OperationReslut 테이블을 관리하는 비용과 검색 성능 향상을 위해 관련 데이터에 직접 접근이 가능하도록 DirectLink 테이블의 사용으로 추가적인 비용이 요구된다. 이는 버퍼를 사용하지 않고 DirectLink 테이블만을 사용하는 순차적인 질의 처리 방법보다는 메모리 I/O의 수가 늘지만 연산에 불필요한 객체 제거와 같이 버퍼를 이용해서 얻을 수 있는 디스크 I/O 비용의 절감의 큰 장점이 있다. 본 논문에서 제안하는 버퍼 관리 기법은 전체적인 연산 성능을 향상시키기 위해 많은 추가적인 공간 비용을 필요로 한다.

다량의 이동체 갱신을 위한 버퍼의 크기는 갱신 이동체의 수, 갱신 연산 수행 시간 간격에 의해 결정된다. 본 논문에서는 수행한 실험에서 버퍼의 크기를 이

동체가 모두 삽입된 색인의 단말 노드 개수의 절반에 해당하는 수만큼의 페이지 크기(페이지의 크기는 4KB로 고정)로 정한다.

3.3.3 갱신 연산 처리 과정

제안 기법의 연산 처리 과정은 버퍼에 저장된 모든 연산을 단말 노드 ID와 객체 ID의 오름차순으로 정렬한다. 버퍼 내에 정렬되어 있는 연산들을 단말 노드의 ID순으로 탐색한다. 해당 단말 노드에 대한 정보는 LeafNode 테이블로부터 얻으며 이 테이블은 각 단말 노드가 가지는 공백 레코드의 수를 저장하고 있다. 각 버퍼에서 해당 단말 노드의 ID를 포함하는 레코드의 수를 비교하면서 연산을 수행한다.

<알고리즘 2> 갱신 연산 알고리즘

```
Procedure Update(oid, newvalue)
Input : an object's id in buffer oid, newvalue in buffer
        newvalue
Begin
01 : SortRecordInBuffer(LeafNodeID, OID);
02 :  for( Until All LeafNodeID of buffer are read )
03 :    for( Until All OID of buffer are read )
04 :      ChooseValidOperation(oid);
05 :      if( OccurSplit(oid) == false )
06 :        ProcessOperaion(oid);
07 :        RemoveOperationInBuffer(oid);
08 :  Merge(index);
09 : ProcessAllOperationInBuffer();
End
```

단말 노드에 들어갈 수 있는 레코드의 수는 한정되어 있으며 단말 노드가 가질 수 있는 최소 레코드의 수도 정해져 있다. 만약 저장될 레코드의 수가 노드의 최대 수용 한계치를 넘으면 노드의 분할(Split)이 일어나며, 저장될 레코드 수가 노드의 최소 수용 한계치에 미치지 못하면 합병(Merge)되어야 한다. 합병과 분할에 대한 기준은 R-트리의 분할/합병 정책에 따른다.

동일 단말 노드 ID에 대한 Delete/Insert 연산의 수 비교는 다음과 같은 3가지 경우로 분류된다.

- Delete 연산 수가 Insert 연산 수보다 많을 경우
- Delete 연산 수가 Insert 연산 수와 동일할 경우
- Delete 연산 수가 Insert 연산 수보다 적을 경우

위의 각 경우에 대한 연산 처리 방법은 다음과 같다.

- Delete 연산 수가 Insert 연산 수보다 많을 경우

이는 해당 단말 노드에 삽입될 객체보다 삭제될 객체의 수가 더 많은 경우이다. 해당 노드에 대한 삽입/삭제 연산을 모두 처리한 결과는 해당 단말 노드의 빈 공간(Number of Blank)을 늘리게 된다. 해당 노드에 대한 모든 연산을 처리하고 난 후 늘어난 빈 공간의 수를 LeafNode 테이블에 반영한다. 처리한 모든 연산은 버퍼에서 지운다.

- Delete 연산 수가 Insert 연산 수와 동일할 경우

이는 해당 단말 노드에 삽입될 객체와 삭제될 객체의 수가 동일한 경우로 연산의 결과는 전체 색인의 구조나 LeafNode 테이블에 영향을 주지 않으며, 처리한 모든 연산은 버퍼에서 제거된다.

- Delete연산 수가 Insert 연산 수보다 적을 경우

이는 해당 단말 노드에 삭제될 객체보다 삽입될 객체의 수가 더 많은 경우이다. 해당 노드에 대한 삽입/삭제 연산을 모두 처리한 결과는 해당 단말 노드의 빈 공간(Number of Blank)을 줄이게 된다. 단말 노드 정보를 파악하여 만약 객체를 삽입할 공간이 부족한 경우 수행하지 못하고 남아 있는 연산은 버퍼에 남게 된다. 해당 노드에 대한 모든 연산을 처리하고 난 후 처리한 연산들을 버퍼에서 제거하고 줄어든 빈 공간의 수를 LeafNode 테이블에 반영한다.

위와 같은 과정을 거치고 난 뒤 LeafNode 테이블을 검사한다. 만약 노드가 가지는 객체의 수가 노드의 최소 한계치보다 작을 경우 R-트리의 합병 연산을 수행한다. 합병 연산 수행에 의해 변경되는 객체의 정보는 DirectLink 테이블에 반영한다. 합병 연산 수행이 모두 끝난 후 버퍼에 남아 있는 객체에 대한 연산을 수행한다. 합병과 분할에 관한 연산은 Guttman의 R-트리의 연산과 동일하다.

갱신 연산을 검색 연산과 분리하여 처리하는 것은 검색 연산에 대한 응답 시간 감소와 유효한 연산만을 추출해내기 위한 것이다. 버퍼를 이용한 검색 연산 과정에서 연산의 결과가 failure인 연산들과 추후에 오는 연산들에 의해 수행하지 않아도 될 연산들을 걸러내어 유효한 갱신 연산만을 수행하도록 기반을 마련한다. 갱신 연산 처리를 위한 단말 노드 분석 과정은 색인을 변형시키지 않는 연산들을 우선 처리한 후 색인 변형을 야기하는 연산들을 일괄처리 하도록 정보를 제공한다. 색인을 변형시키지 않은 연산은 교환 갱신을 이용한 객체의 정보 수정만으로 연산을 수행하기 때문에 갱신 비용을 절감할 수 있다. 갱신 연산 도중에 발생하는 합병을 요구하는 노드들은 버퍼에 저장된 연산들을 처리한 뒤에 수행된다. 색인의 변형을 요구하는 연산들은 분할을

발생시키는 연산들로 합병에 의한 색인 재구성 후에 일괄적으로 처리한다. 이를 통하여 객체의 갱신에 의해 발생하는 다수의 색인 분할/합병의 색인 재구성 비용을 줄일 수 있다. 갱신 연산 과정에서는 단말 노드 관리를 위한 LeafNode 테이블을 추가적으로 사용한다. 따라서 LeafNode 테이블 사용을 위한 공간 비용과 디스크 접근 비용을 필요로 하지만 색인 재구성의 비용을 현저하게 줄일 수 있도록 하는 이점을 가진다.

4. 성능평가

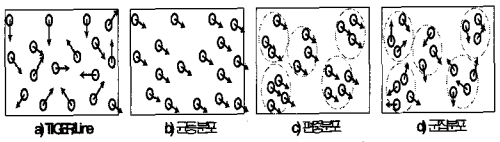
본 장에서는 실험을 통하여 본 논문에서 제시한 다량 삽입 기법을 구현하여 기존의 기법과 비교 분석한다.

4.1 실험 환경

실험은 Pentium IV 3.0GHz CPU, 1GB의 메모리, 160GB E-IDE HDD를 가진 Windows XP OS상에서 수행되었다. 실험에서 사용되는 R-트리의 한 노드는 4KB의 디스크 블록에 해당한다. 버퍼의 크기는 50개의 페이지를 저장할 수 있는 크기로 정한다. 본 논문에서 제안한 지연 다량 삽입 기법은 성능 향상을 위한 방법으로 버퍼 관리 기법과 DirectLink 테이블, OperationResult 테이블, LeafNode 테이블을 사용하였다. 본 논문에서 사용한 OperationResult 테이블과 LeafNode 테이블은 기존의 공간 색인에서 사용되지 않았다. 다량 삽입을 위한 현재 위치 색인을 관련해서 버퍼 관리 기법과 DirectLink 테이블을 사용한 기존의 기법과 성능 비교 평가를 수행한다. 본 논문의 기반이 되는 R-트리를 사용하여 한번에 하나의 연산을 처리하는 R-트리의 기본 삽입 기법(OBO; One By One), 버퍼 관리를 통하여 다량의 데이터를 클러스터링해서 처리하는 GBI 기법, DirectLink 테이블과 교환 갱신 기법을 사용한 Leaf-Update(LUR-트리)기법을 성능 비교 대상으로 정하였다.

삽입 비용과 검색 비용, 갱신 비용, 복합 질의 비용은 각각 초당 시스템의 처리량(TPS)으로 측정하였다. 본 논문에서 제시한 지연 다량 삽입 기법은 LBI(Lazy Bulk Insertion)라 표시하겠다. 'Leaf-Update' 뒤의 숫자는 확장 MBR의 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[14]. 이동체의 수는 약 50만개의 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 30만개의 데이터를 포함한다. 4가지의 데이터 셋은 이동체의 초기 분포와 이동성을 고려하였다. 이는 현실에서 일어날 수 있는 이동체의 움직임을 4가지의 특징으로 분류한 것이다. 실험에서 사용되는 데이터 셋에 대한 특징은 <그림 8>에서 살펴볼 수 있다.

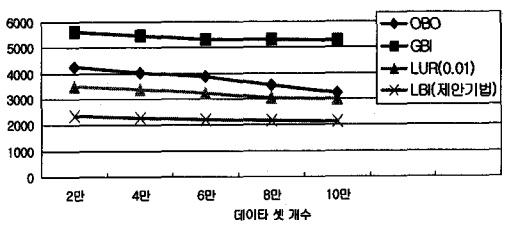


<그림 8> TIGER/Line 데이터와 합성 데이터
(균등분포, 편중분포, 군집분포)

실험 평가한 방법은 다음과 같다. 10만개의 임의의 이동체로 기 구축된 색인에 각각의 데이터 셋을 적용하여 그 성능을 평가하였다. 각각의 데이터 셋에는 search, delete, insert 그리고 update연산이 고르게 분포되어 있고, GBI와 LBI의 경우 갱신 주기를 0.5초로 정하였다.

4.2 삽입 질의 성능

본 실험에서 삽입 질의 성능에 대한 평가는 기 구축된 색인에 임의의 10만개의 TIGER/LINE 데이터 셋을 삽입하여 평가하였다. <그림 9>은 TIGER/Line 데이터를 사용하여 OBO, GBI, LUR(0.01)와 LBI에서 삽입 질의를 수행하였을 때 데이터 셋 2만개 입력마다 변화하는 삽입 질의에 대한 초당 시스템의 처리량(TPS)의 변화를 나타낸 것이다.

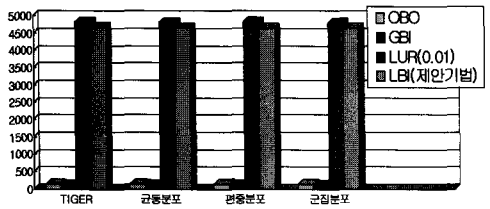


<그림 9> 삽입 데이터 개수에 따른 삽입 비용 변화
(초당 시스템의 처리량 변화)

삽입 성능 실험 결과 OBO방식을 기준으로 GBI는 약 34% 성능 향상, LUR은 약 18% 성능 저하, LBI는 약 34%의 성능 저하를 보였다. LUR과 LBI는 R-트리와 추가적인 색인을 사용하여 데이터 삽입 질의 처리 시 더 많은 비용을 요구한다. 하지만 데이터 셋 개수가 늘어나면서 나타나는 성능 변화를 살펴보면 OBO나 LUR보다 LBI가 더 완만한 기울기를 가지는 것을 알 수 있다. 이는 LBI에서 색인의 분할을 최소화하는 방향으로 질의 처리 순서를 정하기 때문이다. 즉, LBI는 보다 많은 양의 데이터를 처리할수록 상대적인 성능 향상을 가진다.

4.3 검색 질의 성능

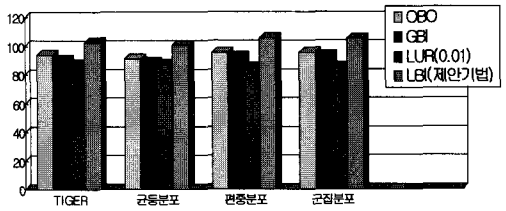
본 실험에서 검색 질의 성능에 대한 평가는 특정 객체를 찾는 질의와 영역을 중심으로 해당 객체를 찾는 질의로 구성된 데이터 셋을 사용하였다. <그림 10>은 TIGER/Line 데이터와 3가지 합성 데이터를 사용하여 OBO, GBI, LUR(0.01)와 LBI에서 삽입, 삭제, 수정 및 검색 질의를 혼합한 질의를 수행하였을 때 특정 객체에 대한 검색 질의의 초당 시스템 처리량(TPS)을 나타낸 것이다.



<그림 10> 데이터 집합 별 특정 객체에 대한 검색 비용(초당 시스템의 처리량)

OBO와 GBI의 경우 특정 객체에 대한 검색 성능이 저하되는 것을 확인할 수 있다. LBI와 LUR은 DirectLink 테이블을 사용하는데 있는 공통점을 가지고 있어 OBO와 GBI보다 특정 객체에 대한 질의 처리에 뛰어난 성능을 보인다. GBI의 경우 OBO보다 약간 저하된 검색 성능을 보이는데 이는 버퍼 관리와 클러스터링을 위한 시간을 추가적으로 필요로 하기 때문이다. LBI와 LUR는 비슷한 검색 성능을 보였는데, LBI가 LUR에 비해 약 2.7%정도의 성능저하를 보였다. 이는 LBI에서 버퍼 관리를 위한 추가적인 시간 비용이 소요되기 때문이다.

다음은 TIGER/Line 데이터와 3가지 합성 데이터를 사용하여 OBO, GBI, LUR(0.01)와 LBI에서 삽입, 삭제, 수정 및 검색 질의를 혼합한 질의를 수행하였을 때 전체 영역의 10%에 해당하는 영역에 대한 검색 질의에 대한 질의 처리 결과를 나타낸 것이다.

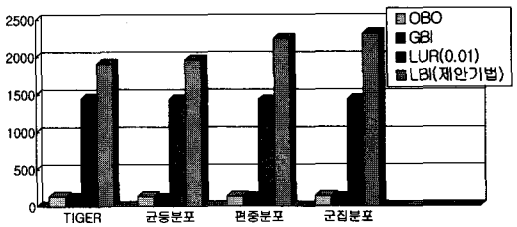


<그림 11> 데이터 집합 별 10% 영역에 대한 검색 비용(초당 시스템의 처리량)

영역 질의의 경우 4가지 기법 모두 동일한 알고리즘으로 질의 처리를 하였으나 그 결과는 약간씩 차이가 있다. LUR의 경우 확장 MBR을 사용하기 때문에 다른 기법들보다 특히 LBI는 OBO보다 약 9%정도의 성능 향상을 보인다. 두 기법에 대한 각각의 연산 처리 수행 시간을 비교 하였을 때 아주 비슷한 성능을 가졌지만 질의 처리 결과를 반환하는 과정에서 LBI는 전송 비용의 이득을 보였다.

4.4 갱신 질의 성능

<그림 12>는 TIGER/Line 데이터와 3가지 합성 데이터를 사용하여 OBO, GBI, LUR(0.01)와 LBI에서 특정 객체에 대한 갱신연산을 수행하였을 때 발생한 초당 시스템의 처리량(TPS)을 나타낸 것이다.



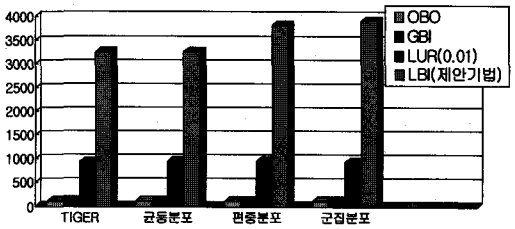
<그림 12> 데이터 집합 별 갱신 비용 (초당 시스템의 처리량)

위의 실험 결과에서 볼 수 있듯이 본 논문에서 제안하는 LBI가 GBI, OBO나 LUR에 비해 삽입 처리에서 크게 앞서는 것을 알 수 있다. 버퍼 관리 기법을 사용

하는 GBI의 경우 클러스터링에 드는 비용이 커서 갱신 비용도 많이 든다. 검색 성능의 저하는 갱신 성능에도 영향을 준다. LUR은 DirectLink 테이블을 사용하여 검색 성능을 향상 시키고, 확장 MBR과 교환 기법을 사용하여 색인 변형을 유발하는 갱신 연산의 수를 줄인다. 하지만 이동체의 갱신 연산이 색인 구조 변형을 요구하는 경우 이동체를 R-트리의 갱신 연산과 동일하게 수행하기 때문에 OBO 방식과 비슷한 삽입 비용을 필요로 한다. LBI기법은 버퍼 관리를 이용하여 갱신 연산의 수를 줄이며, DirectLink 테이블을 사용하여 검색 비용도 줄인다. LBI와 LUR의 가장 큰 차이점은 LBI에서는 유효한 연산만을 추출해내어 전체 연산 수를 줄이는데 있다. LBI 기법은 LUR기법과 비교하여 각각의 데이터 셋에 대해 32%, 36%, 57%, 62%의 성능 향상을 보임을 실험에서 확인할 수 있다.

4.5 복합 질의 성능

<그림 13>은 TIGER/Line 데이터와 3가지 합성 데이터를 사용하여 OBO, GBI, LUR(0.01)와 LBI에서 특정 객체에 대한 삽입, 삭제, 수정 연산을 동일한 비율로 혼합한 복합 질의를 수행하였을 때 발생한 초당 시스템의 처리량(TPS)을 나타낸 것이다.



<그림 13> 데이터 집합 별 복합 질의 비용
(초당 시스템의 처리량)

위의 실험 결과에서 LBI는 OBO방식에 비해 약 30배 이상의 질의 처리 성능을 확인할 수 있다. 이는 삽입과 삭제, 갱신 질의의 혼합으로 인해서 질의 처리과정이 불필요한 모든 질의를 제거함으로써 분할과 합병에 대한 비용은 물론 수행할 질의 수를 줄여 비용을 절약하기 때문이다. 따라서 LBI에서 버퍼 주기 내에 동일한 객체에 대한 연산이 많이 분포할수록 더 좋은 성능을 나타낼 수 있다.

5. 결론 및 향후 연구

매우 빈번하게 이동하는 다량의 이동체 데이터를 다루기 위해, 본 논문에서는 R-트리를 이용한 지연 다량 삽입 기법을 제안하였다. R-트리는 변경된 값과 이전의 값이 연관성을 지니는 변경 요청의 지역성이 존재함을 이용하여 갱신 비용을 줄인다. 다량 삽입 기법은 다량의 갱신들을 버퍼링과 연결 리스트를 이용한 클러스터링을 사용해서 색인에 대한 접근 비용을 줄임으로써 전체적인 색인의 갱신 비용 줄인다. 또한 제안 기법은 기존의 R-트리의 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 변화에 대해 연구하겠다.

참고문헌

[1] T. Abraham and J. F. Roddick, "Survey of spatio-temporal databases," GeoInformatica, pp.61-99, 1999.

[2] 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.

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

[4] 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.

[5] V. Gaede and O. Gunther, "Multidimensional access methods,"ACM Computing Surveys, pp.170-231, 1998.

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

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

[8] M. Abdelguerfi, J. Givaudan, K. Shaw and R. Ladner, "The 2-3TR-tree, A Trajectory Oriented Index Structure for Fully Evolving Valid-Time Spatio-Temporal Datasets," ACM GIS, pp. 29-34, 2002.

[9] D. Pfoser, C. S. Jensen, and Y. Theodoridis, "Novel approaches in query processing for moving object trajectories,"VLDB, pp.395-406, 2000.

[10] S. Saltenis, C. S. Jensen, S. T. Leutenegger, and M. A. Lopez, "Indexing the positions of continuously moving objects," ACM SIGMOD, pp.331-342, 2000.

[11] Y. Tao and D. Papadias, "MV3R-tree: a spatio-temporal access method for timestamp and interval queries,"VLDB, pp.431-440, 2001.

[12] J. Tayeb, O. Ulusoy, and O. Wolfson, "A quadtree -based dynamic attribute indexing method,"The Computer Journal, pp.185-200, 1998.

[13] Z. song, and N. Roussopoulos, "Hashing Moving Objects," International Conference on Mobile Data Management, pp. 161-172, 2001.

[14] 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

김정현

2004년 인하대학교 컴퓨터정보공학과 (공학사)

2004년 ~ 현재 인하대학교 컴퓨터 정보공학과 석사 과정

관심분야 : 공간 데이터베이스, 지리정보시스템, 공간 데이터베이스 클러스터 등

박순영

1989년 인하대학교 전자계산학과 졸업 (이학사)

1991년 인하대학교 전자계산공학과 졸업 (공학석사)

1991년 ~ 현재 한국전자통신연구원 디지털홈연구단 선임연구원

2002년 ~ 현재 인하대학교 대학원 컴퓨터·정보공학과 박사과정

관심분야 : 저장 시스템, XML, GML, 분산 데이터베이스 시스템, 센서 데이터베이스

장용일

1997년 인하대학교 전자계산공학과 (공학사)

2001년 인하대학교 컴퓨터공학부 (공학석사)

2003년 ~ 현재 인하대학교 컴퓨터공학부 박사과정

관심분야 : 웹 & 모바일 GIS, 공간 데이터베이스 클러스터, 위치기반서비스, 그리드 데이터베이스

김호석

2001년 인하대학교 전자계산공학과 (공학사)

2003년 인하대학교 전자계산공학과 (공학석사)

2003년 ~ 현재 인하대학교 컴퓨터정보공학과 박사과정

관심분야 : 공간데이터베이스, GIS, LBS

배해영

1974년 인하대학교 응용물리학과 (공학사)

1978년 연세대학교 대학원 전자계산학과 (공학석사)

1989년 숭실대학교 대학원 전자계산학과(공학박사)

1985년 Univ. of Houston 객원교수

1992년~1994년 인하대학교 전자계산과 소장

1982년 ~ 현재 인하대학교 컴퓨터공학부 교수

1999년 ~ 현재 지능형GIS연구센터 센터장

2000년 ~ 현재 중국 중경우전대학교 대학원 명예교수

2004년 ~ 현재 인하대학교 정보통신대학원 원장

관심분야 : 분산 데이터베이스, 공간 데이터베이스, 지리정보 시스템, 멀티미디어 데이터베이스 등