실시간 갱신을 통한 이동 객체의 색인 기법

Indexing Moving Objects with Real-Time Updates

복경수, 서동민, 유재수
충북대학교 정보통신공학과

Kyoung-Soo Bok(ksbok@netdb.chungbuk.ac.kr), Dong-Min Seo(dmseo@netdb.chungbuk.ac.kr),
Jae-Soo Yoo(yjs@cbucc.chungbuk.ac.kr)

요약

본 논문에서는 위치 기반 서비스에서 이동 객체의 연속적인 위치를 효과적으로 갱신하면서 미래 위치 검색을 지원하는 색인 구조를 제안한다. 이동 객체의 갱신 비용을 감소시키기 위해 제안하는 색인 구조는 보조 색인 구조를 통해 이동 객체가 존재하는 단말 노드를 직접 접근하고 노드의 정보가 변경될 경우 상향식으로 갱신을 수행한다. 실제 이동 객체의 위치는 주 색인 구조에 저장되며 중간 노드에는 노드의 팬아웃을 증가시키기 위해 kd-트리와 유사한 분할 정보를 기록한다. 또한 자식 노드에 포함된 이동 객체를 포함하는 속도 정보를 이용하여 미래 위치 검색을 지원한다.

■ 중심어 : | 이동 객체 | 갱신 | 미래 위치 | 위치 기반 서비스 |

Abstract

In this paper, we propose the index structure supporting the future position retrievals with efficiently updating continuous positions of moving objects in location based services. For reducing update costs of moving objects, our index structure directly accesses to the leaf node with moving objects using secondary index structure and performs bottom up update when node information is changed. Positions of moving objects are stored in primary index structure. In primary index structure, the split information similar to kd-tree is stored to internal node for increasing node's fanout. And the proposed index structure supports the future position retrievals using velocity of moving objects in the child node.

■ keyword : | Moving Objects | Update | Future Position | Location Based Service |

1. 서 론

최근 무선 통신 및 통신 기기의 발전과 더불어 위치 기반 서비스에 대해 많은 관심이 집중되고 있다. 위치 기반 서비스(LBS : Location Based Services)는 GPS 칩을 내장한 핸드폰, PDA와 같은 무선 통신 기기 사용자에게 위치 정보를 통해 다양한 서비스를 제공한다. 위치 기반 서비스의 시장 규모는 매년 200~300%의 성장률을 보이고 있으며 2006년 미국 25억 달러, 유럽 40억

* 본 연구는 2004년도 충북대학교 학술연구지원사업의 연구비 지원에 의하여 연구되었음

접수번호 : #041022-002
접수일자 : 2004년 10월 22일
심사완료일 : 2004년 11월 29일
교신저자 : 유재수, e-mail : yjs@cbucc.chungbuk.ac..kr

달러, 한국 4억 달러 이상이 될 것으로 예상된다. 이러한 위치 기반 서비스는 현재 개인 위주의 서비스에서 전자상거래, 교통, 환경, 의료 등 국가 전반적인 인프라 차원으로 확대 발전 추세에 있다.

이와 함께 90년 중반 이후 공간적인 위치를 이동하면서 서비스를 요청하거나 서비스의 대상이 되는 이동 객체(moving object)에 대한 많은 연구들이 진행되고 있다. 이동 객체는 시간의 변화에 따라 공간적인 위치를 변화하는 시공간 데이터의 특수한 형태이다. 위치 기반 서비스의 대상이 되는 이동 객체들은 일정 시간 주기로 위치 변화를 서버로 전송한다. 이로 인해 많은 갱신 비용을 소요한다[1].

대용량의 이동 객체에 대한 검색을 효과적으로 처리하기 위해서는 이동 객체를 관리하기 위한 색인 구조가 필수적이다. 초기의 시공간 색인 구조는 대부분 과거에서 현재까지 객체의 이동 경로 즉, 궤적을 관리하거나 현재의 위치만을 관리한다. 3DR-트리[2], STR-트리[3], HR-트리[4] 등은 이동 객체의 과거의 위치에서 현재의 위치를 공간적인 위치와 시간 정보를 사용하여 표현한다. 이러한 색인 구조들은 이동 객체의 궤적을 효과적으로 표현하지 못하기 때문에 이를 해결하기 위해 TB-트리[5], SETI-트리[6], SEB-트리[7] 등이 제안되었다.

최근 이동 객체에 대한 현재 위치를 기반으로 하여 미래 위치를 예측하기 위한 필요성이 증가되면서 미래 위치를 검색하기 위해 VCI-트리[8], TPR-트리[9], TPR*-트리[10] 등이 제안되었다. 이러한 색인 구조들은 이동 객체의 현재 위치와 속도 정보를 표현하여 미래 위치에 대한 검색을 가능하게 한다. 예를 들어, "10분 이내에 시청을 통과하는 자동차를 검색해라"와 같은 질의 요청을 수행한다면 현재 시점을 기준으로 노드에 저장된 속도 정보를 이용하여 이동 객체가 이동할 수 있는 영역을 확장하여 검색을 수행한다.

그러나 기존에 제안된 색인 구조는 대부분 R-트리 기반 색인 구조로 이동 객체들이 존재하는 최소의 영역을 생성한다. 따라서, 객체의 삽입이나 갱신으로 단말 노드에 존재하는 영역의 크기나 속도 정보가 변경될 경우 분할된 각 영역을 재조정하기 때문에 많은 갱신 비용을 소요한다. 실제 위치 기반 서비스의 대상이 되는 객체들은 동적인 변화 특성을 갖고 계속적인 위치 변화를 갖기 때문에 기존 R-트리 색인 구조를 사용할 경우 많은 갱신을 소요한다. 또한, 기존에 제안된 R-트리 기반의 색인 구조는 객체의 위치를 갱신하기 위해 루트 노드에서 단말 노드를 순회하면서 객체의 이전 객체의 위치 정보를 삭제하고 새로운 위치를 삽입하는 하향식(top-down) 갱신 기법을 사용한다. 이로 인해, 갱신을 위해 접근해야 할 노드의 수가 증가되고 많은 갱신 비용을 소용한다.

본 논문에서는 위치 기반 서비스의 중심이 되는 이동 객체의 계속적인 위치 변화를 효과적으로 갱신하면서 이동 객체의 미래 위치 검색을 지원하기 위한 색인 구조를 제안한다. 제안하는 색인 구조는 기존의 공간 분할 방식의 색인 구조를 변형하여 이동 객체의 미래 위치를 검색할 수 있으며 계속적인 이동 객체의 변화에 따른 갱신 비용을 감소시키기 위해 이동 객체가 저장된 단말 노드를 직접 접근하기 위한 보조 색인 구조를 사용한다.

본 논문의 구성은 다음과 같다. Ⅱ장에서는 이동 객체의 위치를 검색하기 위해 제안된 색인 구조에 대해 기술하고 Ⅲ장에서는 기존에 색인 구조의 문제점을 해결하기 위해 제안하는 새로운 색인 구조에 대해 기술한다. Ⅳ장에서는 기존에 제안된 색인 구조와의 성능 평가를 수행하고 마지막 Ⅴ장에서는 본 논문의 결론 및 향후 연구에 대해 기술한다.

Ⅱ. 관련연구

이동 객체에 대한 미래 위치를 검색하기 위한 관심이 집중되면서 이동 객체의 현재 위치를 기반으로 미래 위치를 검색하기 위한 VCI-트리[8], TPR-트리[9], TPR*-트리[10] 등이 제안되었다. S. Prabhakar는 이동 객체의 이동 변화에 따라 색인 구조에 대한 갱신 비용을 줄이기 위해 가능한 객체의 최대 속도를 이용하여 색인을 구성하는 VCI-트리(Velocity Constrained Indexing-Tree)라는 색인 구조를 제안하였다[8]. VCI-트리는 기존의 R-트리 기반 색인 구조에 미래를

검색하기 위한 V_{max} 필드를 추가하였다. V_{max} 필드는 자식 노드에 존재하는 이동 객체의 가장 빠른 속도 값을 나타내며 이를 통해 미래 위치를 검색한다.

S. Saltenis는 기존 R*-트리 기반의 색인 구조를 이용하여 이동 객체에 대한 현재 및 미래 위치에 대한 효율적인 검색을 지원하기 위해 TPR-트리(Time Parameterized R-tree)를 제안하였다. TPR-트리는 시간 파라미터 경계 영역(time parameterized bounding rectangle)을 이용하여 현재 및 미래 위치에 대한 검색을 지원한다. TPR-트리는 R-트리 기반 구조를 이용한 균형 트리로 단말 노드의 엔트리는 실제 이동 객체에 대한 위치와 포인터로 구성되며 중간 노드의 엔트리는 자식 노드에 대한 정보와 포인터로 구성된다. 이동 객체에 대한 정보는 참조 위치(reference position)와 속도 벡터(velocity vector)로 표현된다. TPR-트리는 VCI-트리와 유사한 방법으로 이동 객체를 효율적으로 색인하며 이동 객체에 대한 미래 위치 검색을 지원하는 색인 구조이다[9]. TPR-트리는 R-트리를 기반으로 하고 이동 객체는 색인 구조가 구성된 시간에 대한 MBR과 속도 정보로 표현한다.

이동 객체에 대한 효과적인 검색을 수행하기 위해서는 계속적인 이동 객체의 위치 정보를 갱신할 필요가 있다. 그러나 기존의 R-트리 기반의 색인 구조는 루트 노드에서 단말 노드까지 순회하면서 객체의 이전 객체의 위치 정보를 삭제하고 새로운 위치를 삽입하는 하향식(top-down) 갱신 기법을 사용하였다. 또한 중간 노드들 사이에 겹침 영역이 발생할 경우 루트 노드에서부터 단말 노드를 순회해야할 경로가 증가되어 갱신 비용을 증가시킨다. 이러한 문제를 해결하기 위해 D. S. Kwon은 다차원 데이터 색인 구조의 대표적인 R-트리 기반 색인 구조에서 동적인 변화 특성을 갖고 빈번한 갱신을 수행 요청을 효율적으로 처리하기 위해 LUR-트리를 제안하였다[11]. LUR-트리는 해쉬 테이블을 통해 이동 객체가 저장되어 있는 단말 노드를 접근하여 갱신을 수행한다. M. L. Lee는 R-트리 기반 색인 구조에서 발생하는 갱신 성능을 향상시키기 위한 상향식(bottom-up) 갱신 기법을 제안하였다[12]. 상향식 갱신 기법에서는 노드들에 대한 직접적인 접근을 제공하기 위해 메모리 기반 구조를 이용한다. M. L. Lee는 이동 객체에 대한 갱신을 상향식으로 수행하기 위해 해쉬 테이블(hash table)과 노드에 대한 요약 구조(summary structure)를 사용하였다. 해쉬 테이블은 LUR-트리와 같이 객체가 저장된 단말 노드를 직접 접근하기 위한 정보를 저장한다. 요약 구조는 R-트리 기반의 색인 구조의 중간 노드를 직접 접근하기 위한 직접 접근 테이블(direct access table)과 단말 노드의 상태를 저장하는 비트 벡터로 이루어져 있다.

III. 제안하는 색인 구조

1. 색인 구조

제안하는 색인 구조는 이동 객체의 계속적인 위치 변화를 효과적으로 갱신하면서 미래 위치 검색을 지원한다. 제안하는 색인 구조는 갱신 비용을 감소시키기 위해 공간 분할 방식에 의해 색인을 구성하고 이전에 삽입된 이동 객체에 대한 갱신은 색인 구조 전체를 순회하지 않고 상향식으로 수행한다. 그림 1은 제안하는 색인 구조를 나타낸 것이다. 제안하는 색인 구조는 주 색인 구조(primary index structure)와 보조 색인 구조(secondary index structure)로 구성되어 있다. 주 색인 구조는 미래 위치를 검색하기 위해 실제 이동 객체를 저장한다. 주 색인 구조는 기존의 공간 분할 방식의 색인 구조를 변형하여 이동 객체의 미래 위치를 검색할 수 있도록 한다. 주 색인 구조는 높이 균형 트리로 중간 노드는 공간 분할 방식의 대표적인 kd-트리 형태로 구성된다. 주 색인 구조의 단말 노드에는 객체의 실제적인 위치 정보를 저장하며 단말 노드가 존재하는 레벨은 동일하다. 중간 노드에는 단말 노드에 존재하는 이동 객체의 미래 위치를 검색하기 위한 영역 정보와 속도 정보를 표현한다. 보조 색인 구조는 이동 객체의 계속적인 위치 변화에 따른 갱신을 효과적으로 수행하기 위해 주 색인 구조의 단말 노드에 저장되어 있는 객체의 위치를 직접 접근하는 색인 구조이다. 보조 색인 구조는 해쉬 테이블로 구성되어 있으며 이동 객체의 위치가 갱신하기 위해서 보조 색인 구조를 통해 이동 객체의 이전 위

치가 저장된 주 색인 구조의 단말 노드를 검색한다. 새로운 이동 객체의 삽입은 전통적인 삽입 기법에 의해 주 색인 구조를 순회하면서 객체를 삽입하고 보조 색인 구조에 객체가 삽입된 단말 노드의 정보와 객체 식별자를 해쉬 테이블에 삽입한다.

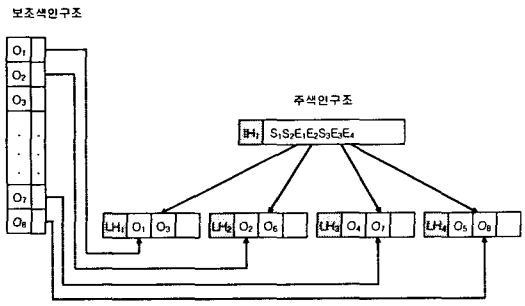


그림 1. 제안하는 색인 구조

2. 노드 구조

2.1 주 색인 구조

주 색인 구조의 단말 노드는 주기적으로 전송된 이동 객체 o_i에 대한 최신의 위치 정보를 저장한다. 단말 노드는 헤더 LH(Leaf Header)와 실제 이동 객체의 위치를 저장하는 엔트리로 구성되어 있다. 단말 노드의 헤더 LH_i는 $\langle sp, pptr \rangle$로 구성되며, 단말 노드에 새로운 객체의 삽입으로 노드에 오버플로우가 발생하였을 때 공간 분할을 수행한다. 이동 객체의 갱신으로 노드에 변경이 발생하였을 경우 부모 노드를 직접 접근하여 변경된 노드의 정보를 반영한다. 제안하는 색인 구조는 이동 객체의 계속적인 위치 변화에 따른 MBR 영역의 재구성 시간을 제거하기 위해 공간 분할에 의해 색인을 구성한다. sp는 단말 노드에 존재하는 객체들을 포함하는 영역으로 공간 분할에 의해 생성된 영역으로 $\langle sp^-, sp^+ \rangle$와 같이 표현된다. sp^-는 분할된 n차원 공간의 각 차원 i에 대한 하한 영역 $\langle sp_1^-, sp_2^-, \ldots, sp_n^- \rangle$이고 sp^+는 분할된 n차원 공간의 각 차원 i에 대한 상한 영역 $\langle sp_1^+, sp_2^+, \ldots, sp_n^+ \rangle$이다. 이러한 sp는 단말 노드에 오버플로우가 발생하여 분할을 수행할 때 공간 분할을 수행할 기준 영역이다. 이동 객체의 위치 변화에 따라 색인 구조에 갱신을 요청할 경우 보조 색인 구조를 통해 객체의 이전 위치가 존재하는 단말 노드를 직접 접근한다. 이동 객체의 이전 위치가 존재하는 단말 노드 영역 내에 새로운 위치가 존재하는지를 판별하기 위해서는 단말 노드에는 공간 분할에 의해 생성된 영역 정보 sp을 이용한다.

단말 노드의 헤더에 존재하는 $pptr$는 단말 노드에 존재하는 이동 객체의 갱신으로 노드에 대한 변경이 발생할 경우 하향식으로 부모 노드를 직접 접근하여 갱신하기 위한 부모 노드에 대한 포인터이다. 제안하는 색인 구조는 이동 객체의 삽입이나 기존에 삽입된 이동 객체에 대한 갱신으로 단말 노드에 대한 정보가 변경되었을 경우 $pptr$을 이용하여 부모 노드를 직접 접근할 수 있도록 한다.

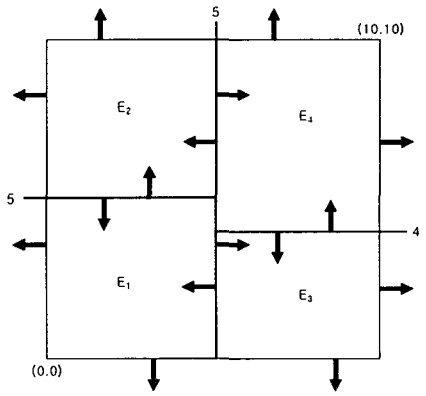
단말 노드에 존재하는 엔트리는 서버로 전송된 이동 객체의 실제 위치를 저장한다. 따라서, 단말 노드의 엔트리는 서버로 전송되는 이동 객체 o_i의 정보 $\langle id, p_{upt}, v_{upt}, t_{upt} \rangle$을 저장한다. 이때, id는 이동 객체 o_i에 부여된 고유한 식별자를 나타내며 p_{upt}와 v_{upt}는 이동 객체의 위치 정보를 나타내는 것으로 이동 객체의 갱신을 수행한 t_{upt} 시점의 위치와 속도를 나타낸다. 기존에 제안된 색인 구조에서는 이동 객체의 실시간적인 갱신을 수행하지 않기 때문에 색인을 한번 구성하면 특정 시간 동안 갱신을 수행하지 않거나 이동 객체의 변화가 고정되어 있다고 생각한다. 따라서, 단말 노드에는 객체의 식별자와 위치 정보만을 기록하였다. 그러나 제안하는 색인 구조에서는 이동 객체의 실시간적인 갱신을 수행하기 때문에 색인된 객체들에 대한 갱신 시간이 서로 다를 수 있다. 따라서, 단말 노드에는 이동 객체들에 대한 갱신 시간을 포함해야 한다. 이를 통해 보다 정확한 객체의 위치를 검색할 수 있다.

주 색인 구조의 중간 노드에는 이동 객체에 대한 현재 및 미래 위치를 검색하기 위해 자식 노드에 존재하는 이동 객체를 포함하는 영역과 속도 정보를 표현한다. 중간 노드는 단말 노드와 유사하게 헤더 IH(*Internal Header*)와 실제 자식 노드에 대한 영역 및 속도 정보

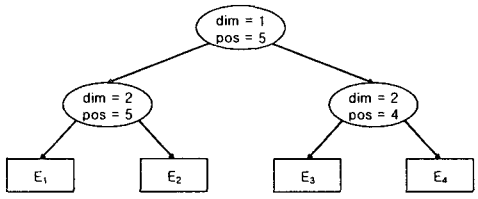
를 표현하기 위한 *PI*(*Partition Information*)로 구성되어 있다. 중간 노드의 헤더 IH_i는 〈*sp*, *pptr*〉로 자식 노드에 분할 정보를 표현하기 위한 기준 정보 *sp*와 부모 노드를 접근하기 위한 포인터 *pptr*로 구성된다. *sp*는 자식 노드에 대한 영역을 포함하는 영역으로 공간 분할에 의해 생성된 영역이다. 이러한 *sp*는 겹침이 발생하지 않으며 중간 노드에 대한 오버플로우가 발생할 경우 공간 분할을 수행하기 위한 기준 정보이다. 중간 노드는 분할된 영역에 대한 실제 영역을 표현하는 것이 아니라 분할된 정보만을 표현한다. 따라서, *sp*는 각 분할된 영역을 표현하기 위한 기준 영역으로 사용한다. *pptr*는 상향식 갱신을 위한 부모 노드의 포인터이다.

중간 노드의 헤더를 제외한 나머지 부분은 중간 노드의 헤더에 존재하는 *sp*을 기준으로 자식 노드를 접근하기 위한 분할 영역을 표현하기 위해 하나의 kd-트리로 구성된다. 중간 노드에 기록되는 kd-트리의 중간 노드는 자식 노드의 분할 영역을 나타내기 위한 분할 정보를 표현하고 kd-트리의 단말 노드에는 실제 자식 노드에 대한 미래 위치를 검색하고 접근하기 위한 정보를 표현한다.

제안하는 색인 구조는 자식 노드에 대한 분할 영역을 표현하기 위해 kd-트리의 중간 노드는 자식 노드에 대한 분할 정보를 나타내기 위한 〈dim, *pos*〉에 의해 나타낸다. dim는 헤더에 존재하는 *sp*을 기준으로 분할을 수행한 차원을 나타내며 *pos*는 분할 차원에서 분할을 수행한 위치를 나타낸다. 예를 들어, 그림 2의 (a)와 같이 2차원 공간에서 분할된 영역 4개 영역이 존재한다고 하자. 중간 노드의 헤더에 존재하는 *sp*에는 (0, 0, 10, 10)이 되어 있다고 할 때, 자식 노드는 첫 번째 차원의 5에 의해 분할을 수행하고 분할된 각 영역은 다시 두 번째 차원에 대해 5인 위치와 4인 위치에 의해 분할되어 있다. 그림 2의 (a)와 같이 분할된 영역은 그림 2의 (b)와 같이 표현된다. 그림 2의 (b)에서 분할된 각 영역 정보는 kd-트리의 분할 정보에 의해 표현되고 kd-트리의 단말 노드에는 실제 자식 노드에 대한 속도 정보와 자식 노드에 대한 포인터를 표현한다.



(a) 중간 노드의 분할 영역



(b) 분할 정보

그림 2. 중간 노드의 분할 정보

그러나 kd-트리 형태의 분할 정보를 노드에 기록할 경우 분할된 영역을 접근하기 위해서는 포인터가 필요하다. 이러한 포인터를 제거하기 위해서 중간 노드에서는 분할 정보에 대해 kd-트리를 깊이 우선 탐색 방식에 의해 트리를 순회한 순서대로 노드에 기록한다. 예를 들어, 그림 2와 같이 분할된 4개의 영역이 존재한다고 할 때, 실제 노드에는 kd-트리를 깊이 우선으로 탐색한 순서대로 기록된다. 그림 3은 깊이 우선 방식에 의해 기록된 중간 노드의 분할 정보를 나타낸 것이다.

LH	〈1, 5〉	〈2, 5〉	E_1	E_2	〈2, 4〉	E_3	E_4

그림 3. 중간 노드의 실제 분할 정보

kd-트리의 단말 노드에는 이동 객체의 미래 위치를 검색할 수 있도록 자식 노드의 속도 정보와 함께 분할된 영역에 존재하는 객체들이 분할 영역을 벗어나는 시

간을 표현한다. 또한, 검색 과정에서 자식 노드를 접근할 수 있도록 자식 노드에 대한 포인터를 포함한다. 만약 kd-트리에 의해 분할된 영역 sp가 존재한다고 할 때 kd-트리의 단말 노드 E_i에는 분할된 영역에 대한 정보 $\langle sp_{vec}, t_{upt}, ptr\rangle$로 구성된다. sp_{vec}는 분할 영역에 대한 속도 정보, t_{upt}는 분할 영역에 대한 갱신 시간, ptr는 자식 노드에 대한 포인터를 나타낸다. 미래 위치를 검색하기 위해 필요한 속도 정보 sp_{vec}는 n차원의 공간에서 자식 노드를 포함하는 영역에 대한 최대 속도로 $\langle sp^-_{vec}, sp^+_{vec}\rangle$와 같다. sp^-_{vec}는 각 차원 d의 하한 영역 sp^-로 이동하는 객체에 대한 최소 속도 $\langle v_1^-, v_2^-, \ldots, v_n^-\rangle$이고 sp^+_{vec}는 각 차원 d의 상한 영역 sp^+에 대한 최대 속도 $\langle v_1^+, v_2^+, \ldots, v_n^+\rangle$이다.

공간 분할에 의해 생성된 분할 영역 sp는 분할된 영역 내에 사장 공간을 포함하고 있다. 또한 중간 노드에 존재하는 자식 노드에 대한 속도는 각 차원 d에 대해 최대로 확장할 수 있는 최대 속도에 의해 표현된다. 공간 분할에 의해 생성된 영역은 실제 객체들이 존재하지 않는 사장 공간을 포함하고 있다. 특정 시간 구간 동안은 늦게 이동하고 있는 객체가 영역의 테두리 부분에 존재하고 빠르게 이동하고 있는 객체가 영역의 중심 부분에 존재하는 경우 실제 부모 노드에서는 미래의 어느 시점에서 검색을 수행할지 판단할 수 없기 때문에 최대의 속도로 이동하고 있는 객체의 속도를 노드의 속도로 사용한다. 따라서 불필요한 영역의 확장을 수행할 수 있다. 이러한 현상은 R-트리 계열의 색인 구조에서도 발생한다.

중간 노드에서는 실제 단말 노드에 존재하는 미래 시점에 대한 검색을 수행하는 과정에서 빠르게 이동하고 있는 객체에 영역의 중심 부분 또는 반대편에 존재할 경우 불필요한 노드의 확장을 수행한다는 문제점이 있다. 따라서, 이러한 문제점을 해결하기 위해 중간 노드에서는 t_{esp}을 이용하여 불필요한 노드의 확장을 방지한다. t_{esp}는 노드에 존재하는 객체들이 분할 영역 sp을 벗어나는 시간이다. t_{esp}는 (1)과 같이 자식 노드에 포함된 객체 o_i들이 각 차원 d에 대해 분할 영역을 벗어나는 시간 $t^d_{esp}(o_i)$의 최소 값에 의해 표현한다.

$$t_{esp} = MIN\{t^d_{esp}(o_i)\} \quad (1)$$

이동 객체의 위치가 변경될 때마다 계속적인 갱신을 수행하기 때문에 색인 구조 내에는 서로 다른 갱신 시간을 갖는 이동 객체들이 존재할 수 있다. 만약 단말 노드의 갱신 시간을 중간 노드에 반영하지 않을 경우 이동 객체의 위치를 검색하는 과정에서 불필요한 영역의 확장을 발생시킬 수 있다. 이로 인해 검색해야 할 노드의 수를 증가시켜 검색 성능을 저하시킬 수 있다. 따라서, 중간 노드에는 자식 노드에 대한 갱신 시간 t_{upt}을 표현해야 한다. 이러한 t_{upt}는 단말 노드에 존재하는 객체의 위치가 갱신되면 (2)을 이용하여 자식 노드에 존재하는 객체 o_i에 대한 갱신 시간 $t_{upt}(o_i)$의 가장 과거의 갱신 시간을 부모 노드에 반영한다.

$$t_{upt} = MIN\{t_{upt}(o_i)\} \quad (2)$$

2.2 보조 색인 구조

일반적으로 이전에 삽입된 이동 객체의 위치를 갱신하기 위해서는 색인 구조의 루트 노드에서 단말 노드를 순회하면서 이동 객체의 이전 위치가 존재하는 노드를 탐색하여 이전 위치를 삭제하고 새로운 위치를 다시 삽입하는 과정을 수행한다. 따라서, 이동 객체의 계속적인 위치 변화에 따라 최대 두 번의 색인 구조를 순회하게 된다. 이러한 갱신 과정은 색인 구조의 갱신 비용을 증가시키게 된다.

이동 객체의 계속적인 위치를 변화를 효과적으로 갱신하기 위해서는 이동 객체의 이전 위치를 빠르게 검색하고 객체의 새로운 위치를 삽입하기 위한 단말 노드를 선별해야 한다. 제안하는 색인 구조에서는 이동 객체가 최신 위치를 저장하고 있는 주 색인 구조의 단말 노드를 직접 접근하기 위한 해쉬 테이블을 사용한다. 해쉬 테이블은 이동 객체가 삽입된 단말 노드의 위치를 직접 접근하여 갱신을 수행할 수 있도록 $\langle id, ptr\rangle$로 구성

된다. *id*는 단말 노드에 저장된 이동 객체의 식별자를 나타내며 *ptr*는 *id*을 갖는 이동 객체가 저장된 단말 노드에 대한 포인터를 나타낸다.

제안하는 색인 구조에서는 이전에 삽입된 이동 객체의 위치가 변경되어 갱신을 수행할 경우 보조 색인 구조를 통해 이동 객체의 이전 위치가 저장된 단말 노드를 접근하여 상향식으로 갱신을 수행한다. 따라서, 이동 객체가 저장된 이전 위치를 찾기 위해 주 색인 구조를 순회할 필요가 없다.

3. 삽입

제안하는 색인 구조는 공간 분할 방식을 이용하여 색인을 구성하기 때문에 분할된 영역들 사이에 겹침이 발생하지 않으며 주 색인 구조의 중간 노드에는 자식 노드에 대한 속도 정보를 저장하기 때문에 이동 객체의 미래 위치를 검색할 수 있다. 또한, 이동 객체의 미래 위치를 검색하기 위해 불필요한 영역의 확장을 방지하기 위해 t_{esp}을 이용하여 색인을 구성한다. 그림 4는 제안하는 색인 구조의 삽입 알고리즘을 나타낸 것이다. 객체 e을 삽입하기 위해서 먼저 *SearchHashTable*()을 수행하여 서버로 전송된 객체의 위치가 기존에 삽입된 이후 갱신을 요청하는 것인지 새로운 객체의 삽입인지를 판별한다. 만약 객체가 이전 위치에 대한 갱신을 요청하는 것이라면 *UpdateNode*()을 수행하여 객체의 위치를 갱신한다. 그러나 현재 서버로 전송된 객체가 새로 삽입되는 객체라면 *FindNode*()을 수행하여 객체를 삽입하기 위한 단말 노드를 탐색하고 *InsertNode*()를 수행하여 단말 노드에 객체를 삽입한다.

현재 서비스를 수행 중인 이동 객체의 위치는 주기적으로 서버에 전송된다. 특정 시간 동안 서버로 위치를 전송하지 않는 객체는 객체에 장애가 발생하거나 서비스를 중지한 것으로 판별한다. 서비스를 중지한 이동 객체에 대한 삭제를 수행하기 위해 제안하는 색인 구조에서는 객체를 삽입할 단말 노드를 접근하면 단말 노드에 존재하는 객체에 대해 DeleteExpObject()를 수행한다. DeleteExpObject()는 단말 노드에 존재하는 객체 o_i에 대해 현재 서버로 전송된 객체 e의 갱신 시간을

```
Algorithm Insert(e, root)
/* 객체 o를 삽입할 단말 노드를 검색하여 객체를 삽입 */
{
   node=ReadNode(root); /* 루트 노드를 메모리를 읽어 들임 */
   if(node==LEAF){ /* 루트 노드가 단말 노드일 경우 */
      InsertObject(node, e); /* 단말 노드에 객체를 삽입 */
      return ;
   }
   /* 해쉬 테이블에서 이동 객체에 대한 이전 위치를 검색하여 삽입 */
   if(node=SearchHashTable(e)){ /* 해쉬 테이블에 객체가 존재 */
      leaf=ReadNode(node); /* 단말 노드를 읽음 */
      DeleteExpObject(leaf); /* 서비스를 중지한 객체를 삭제 */
      UpdateNode(e, leaf); /* 단말 노드에서 객체의 위치를 갱신 */
   }
   else{ /* 해쉬 테이블에 이동 객체가 존재하지 않는 경우 */
      leaf=FindNode(e, node); /* 객체를 삽입할 단말 노드를 탐색 */
      DeleteExpObject(leaf); /* 서비스를 중지한 객체를 삭제 */
      InsertNode(e, leaf); /* 단말 노드에 객체를 삽입 */
   }
}
```

그림 4. Insert( ) 알고리즘

비교하여 $MaxT$을 초과하도록 서버로 자신의 위치를 전송하는지 않는 객체 o_i는 서비스를 중지한 것으로 판별하고 삭제한다. (3)는 단말 노드에서 서비스를 중지한 객체를 삭제하는 조건을 나타낸 것이다.

$$(t_{upt}(e) - t_{upt}(o_i)) < MaxT \quad (3)$$

객체의 이전 위치를 갱신하는 것이 아니라 새로 삽입되는 객체일 경우에는 주 색인 구조를 순회하면서 객체를 삽입할 단말 노드를 검색하게 된다. 중간 노드에는 분할된 영역 자체를 저장하는 것이 아니라 중간 노드에 존재하는 헤더를 기준으로 분할된 영역 정보가 저장되어 있다. 따라서, 중간 노드에서 새로운 객체를 삽입하기 위한 자식 노드를 탐색하기 위해서는 kd-트리와 유사하게 중간 노드에 존재하는 분할 정보를 통해 순회하면서 객체를 삽입할 자식 노드를 탐색한다. 이동 객체의 위치를 삽입하기 위한 단말 노드를 탐색하는 *FindeNode*()는 헤더를 기준으로 kd-트리 형태의 분할 정보를 통해 객체를 삽입할 자식 노드를 검색한다.

이동 객체를 삽입할 단말 노드를 검색하면 먼저 객체의 삽입으로 오버플로우가 발생하는지를 검사한다. 객체의 삽입으로 오버플로우 발생하지 않는 경우에는 객체를 삽입하고 객체의 삽입으로 변경된 노드의 정보를 부모 노드에 반영한다. 만약 객체의 삽입으로 오버플로

우가 발생할 경우에는 직접 분할을 수행하지 않고 오버플로우가 발생한 단말 노드와 합병이 가능한 형제 노드를 검색한다. 오버플로우가 발생한 노드와 합병이 가능한 노드가 존재하는 경우에는 단말 노드에 존재하는 일부 객체들을 형제 노드에 삽입하여 오버플로우에 대한 처리를 수행한다. 이에 반해, 오버플로우가 발생한 단말 노드와 합병이 가능한 형제 노드가 존재하지 않는 경우에는 분할을 수행하고 분할된 정보를 부모 노드에 반영한다. 예를 들어, 2차원 데이터 공간에 o_1에서 o_8의 8개의 객체가 존재하고 단말 노드에 최대 3개의 객체를 저장할 수 있다고 하자. 그림 5와 같이 새로운 객체 e가 삽입된다고 가정하면 노드에 오버플로우가 발생한다. 현재 객체 e가 삽입된 노드에 이웃한 형제 노드에 오버플로우가 발생하지 않으면서 병합이 가능한 형제 노드가 존재한다. 이러한 경우에는 그림 6과 같이 오버플로우가 발생한 노드에 존재하는 객체의 일부를 형제 노드에 재삽입하여 오버플로우를 처리할 수 있다.

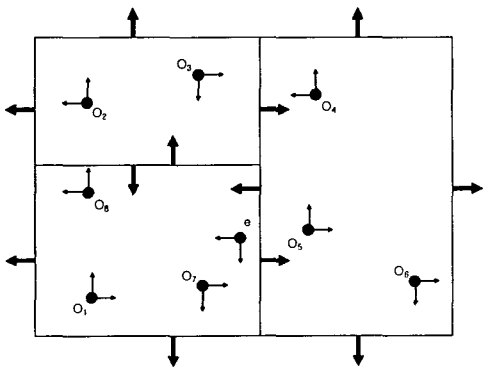


그림 5. 병합이 가능한 경우

이동 객체의 삽입 또는 위치 변화에 따른 갱신을 수행하는 과정에서 노드에 오버플로우가 발생하면 오버플로우가 발생한 노드의 형제 노드를 검색하여 병합을 수행하여 오버플로우가 발생한 노드에 존재하는 객체의 일부를 형제 노드에 삽입하여 오버플로우에 대한 처리를 수행한다. 그러나 형제 노드와 병합을 수행하여 오버플로우를 처리할 수 없을 경우에는 분할을 수행한다. 제안

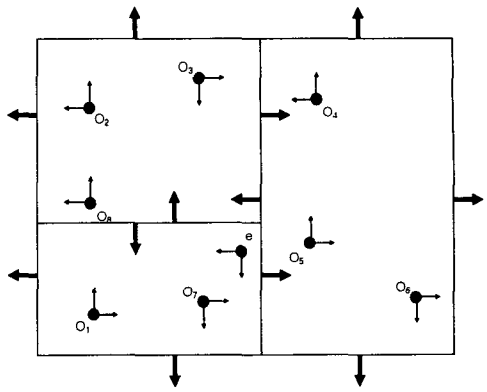


그림 6 노드의 재조정

하는 색인 구조는 미래 위치를 검색할 때 탐색해야 할 노드의 수를 감소시킬 수 있도록 분할 위치를 선택해야 한다. 단말 노드에 대한 분할을 수행하기 위한 분할 위치를 선택하기 위한 기준은 다음과 같다. 먼저, 분할된 영역에 대한 미래 위치에 대한 검색을 효과적으로 수행하기 위해 분할 영역에 대한 속도 sp_{vec}을 최소로 생성할 수 있는 위치를 선택한다. 만약 sp_{vec}을 최소로 생성할 수 있는 다수의 차원이 존재하는 경우에는 분할된 노드에 대한 t_{esp}과 t_{upd}의 값을 최대로 생성할 수 있는 위치를 선택한다.

중간 노드에 오버플로우가 발생할 경우에도 단말 노드와 유사하게 분할된 형제 노드와 합병이 가능한 노드가 존재하는지 판별하여 오버플로우에 대한 처리를 수행한다. 만약 오버플로우가 발생한 중간 노드와 합병이 가능한 중간 노드가 존재하지 않을 경우에는 중간 노드에 대한 분할을 수행한다. 공간 분할 방식의 색인 구조에서 중간 노드의 분할은 자식 노드들에 대한 계속적인 분할을 수행시키는 연속 분할(Cascading Split)을 발생시킬 수 있다. 따라서, 중간 노드에 대한 분할 위치를 판별하기 위해서는 먼저 기존에 분할 위치를 통해 자식 노드에 대한 연속 분할을 발생시키지 않는 위치에 대해 분할 위치를 선택한다.

4. 갱신

기존에 삽입된 이동 객체에 대한 갱신을 수행하기 위

해서는 해쉬 테이블을 통해 이동 객체가 저장된 단말 노드에 직접 접근하고 *UpdateNode*()을 수행하여 이동 객체의 위치를 갱신한다. 그림 7은 *UpdateNode*() 알고리즘을 나타낸 것이다. 이동 객체의 이전 위치가 저장되어 있는 단말 노드에 접근하면 새로운 위치가 단말 노드의 영역 내에 포함되는지를 판별하기 위해 *ReadHeader*()를 수행하여 단말 노드의 헤더를 읽고 헤더에 포함된 단말 노드의 영역을 이용하여 새로운 객체의 위치가 단말 노드에 포함되는지를 판별한다. 객체의 새로운 위치가 단말 노드 내에 포함되는 경우에는 *ReplacePosition*()을 수행하여 객체의 이전 위치를 삭제하고 객체의 새로운 위치를 단말 노드에 삽입한다. 만약 객체의 갱신으로 단말 노드에 대한 sp_{vec}, t_{esp}, t_{upt}가 변경되는 경우 *AdjustNode*()를 수행하여 부모 노드에 변경된 내용을 반영한다. 만약 객체의 새로운 위치가 단말 노드의 영역 내에 존재하지 않는 경우에는 *DeleteObject*()를 수행하여 기존 객체의 위치를 삭제하고 *FindNode*()를 통해 객체의 새로운 위치를 삽입할 단말 노드를 검색하여 객체를 삽입한다.

```
Algorithm UpdateNode(e, leaf)
/* 기존 객체에 존재하는 단말 노드 node에서 객체 o의 위치를 갱신 */
{
    leafhd=ReadHeader(leaf); /* 단말 노드에 대한  헤더를 읽음 */
    leafinfo=CurrentNodeInfo(leaf);
    if(CaluateContains(leafhd, e)){ /* 영역 내에 포함되는 경우 */
        newe=ReplacePosition(leaf, e); /* 객체의 새로운 위치를 갱신 */
        AdjustNode(newe, leafinfo); /* 부모 노드에 변경된 반영 */
    }
    else{ /* 객체 o의 위치가 영역 내에 포함되지 않는 경우 */
        DeleteObject(node, e); /* 객체의 이전 위치를 삭제 */
        DeletehashTable(e); /* 해쉬 테이블에서 객체를 삭제 */
        /* 루트 노드에서 부터 객체 o를 삽입할 단말 노드를 탐색 */
        leaf=FindNode(e, node);
      /* 단말 노드에 객체 o를 삽입 */
        InsertNode(e, leaf);
    }
}
```

그림 7. UpdateNode() 알고리즘

예를 들어, 2차원 데이터 공간에 o_1에서 o_7까지 7개의 객체가 존재하고 분할된 3개의 단말 노드가 존재한다고 가정하자. 이전에 삽입된 o_1 객체의 위치가 o'_1로 변경되었다고 할 때 그림 8과 같이 동일한 노드에 객체의 새로운 위치가 변경되었을 경우에는 단말 노드에 존재하는 이동 객체의 위치를 새로운 위치로 변경한다. 만약 객체의 위치의 변경으로 노드의 정보가 변경되었을 경우에는 각 노드에 존재하는 부모 노드에 대한 포인터를 이용하여 상향식으로 갱신을 수행한다. 이에 반해, 그림 9와 같이 o_1 객체의 새로운 위치 o'_1가 기존 노드에 존재하지 않을 경우에는 객체의 이전 위치를 삭제하고 색인 구조를 순회하면서 이동 객체의 새로운 위치를 삽입한다.

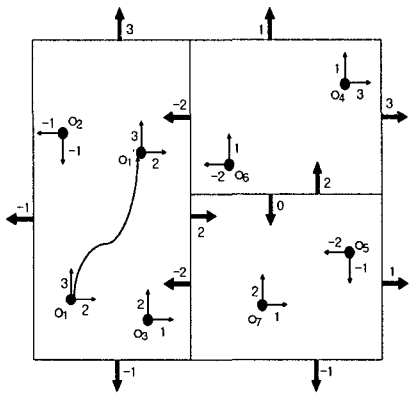


그림 8 기존 노드에 존재하는 경우

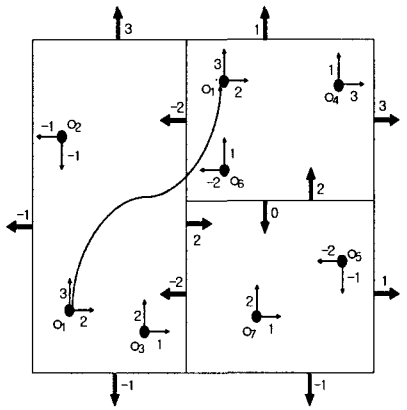


그림 9 다른 노드에 존재하는 경우

Ⅳ. 실험 및 성능 평가

1. 실험 환경

성능 평가를 위해 제안하는 색인 구조와 TPR-트리

와 제안하는 색인 구조를 256M의 메모리는 갖는 펜티엄 Ⅳ 1.7GHz에서 C 언어를 통해 구현하였다. 실험을 위해 사용된 데이터는 GSTD[13]을 통해 10만개의 이동 객체를 생성한다. 표 1은 성능 평가에 사용된 파라미터를 나타낸다. 제안하는 색인 구조는 편의상 SP(Space Partition)-트리라 한다.

표 1 성능평가를 위한 파라미터

항 목	값
이동 객체 수	10,000~100,000
노드의 크기	4Kbyte
객체에 대한 갱신 수	1000~10,000
질의 시간 간격	5~30
질의 영역 크기	0.1~5

2. 성능 평가

제안하는 색인 구조의 이동 객체의 계속적인 위치 변화에 따른 갱신 비용을 최소화하면서 이동 객체의 미래 위치를 지원하기 위한 색인 구조이다. 먼저 제안하는 색인 구조와 TPR-트리에 대한 삽입 성능 및 갱신 성능을 비교한다. 삽입 성능을 비교하기 위해 1만개에서 10만개의 이동 객체를 삽입하는 시간을 비교한다. 또한, 갱신 성능은 10만개의 이동 객체를 삽입하고 기존에 삽입된 이동 객체에 대해 1천개에서 1만개까지의 위치를 갱신한 시간을 비교한다. 그림 10은 삽입 성능을 수행한 결과이고 그림 11은 갱신 성능을 수행한 결과이다. TPR-트리는 데이터 분할 방법을 사용하기 때문에 데이터의 계속되는 위치 이동으로 인해 색인의 변경이 빈번히 발생하기 때문에 많은 갱신 시간이 요구한다. 이에 반해 제안하는 색인 구조는 공간 분할 방법을 사용하여 데이터를 색인하기 때문에 데이터 분할 방법을 사용하는 색인 구조의 문제점인 빈번한 색인 구조의 변경 비용을 줄일 수 있고 보조 색인 구조를 통한 빠른 객체의 검색을 지원함으로 갱신 비용을 효율적으로 줄일 수 있다. 검색 성능은 10만개의 객체를 삽입하고 5~30분까지 미래 위치에 대한 범위 검색을 수행한다. 그림 12는 검색 성능의 비교는 1000번의 범위 검색을 수행한 평균시간

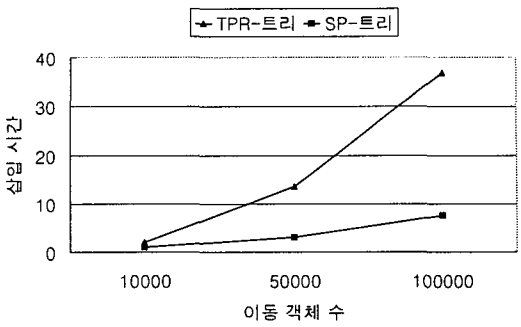


그림 10 삽입 성능

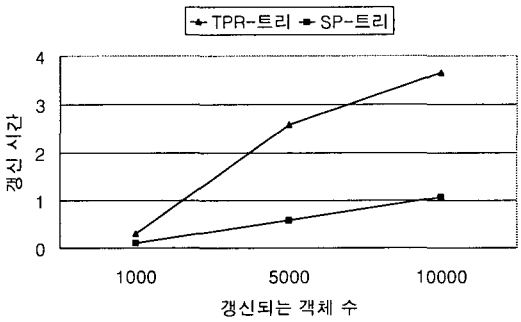


그림 11. 갱신 성능

을 나타낸 것이다. 제안하는 색인 구조의 중간 노드는 자식 노드에 대한 분할 영역을 kd-트리 형태로 표현하기 때문에 노드의 팬아웃을 증가시킬 수 있다. 또한, 분할된 각 영역에 대한 갱신 시간 t_{upd}과 분할된 영역들이 현재 노드를 벗어나는 t_{esp}을 표현하기 때문에 불필요한 노드의 확장을 감소시킬 수 있다. 이로 인해 검색 성능을 향상시킬 수 있다.

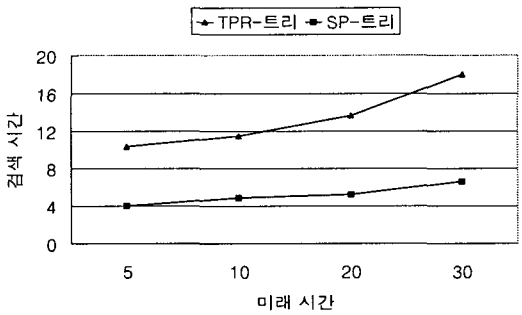


그림 12 미래 위치 검색

V. 결론 및 향후 연구

본 논문에서는 이동 객체의 연속적인 위치 변화에 대한 갱신을 효과적으로 수행하면서 이동 객체의 미래 위치를 지원하는 색인 구조를 제안하였다. 제안하는 색인 구조는 이동 객체의 계속적인 위치를 변화에 따른 갱신을 효과적으로 수행하기 위해 공간 분할 방식에 의해 색인을 구성하고 각 노드에는 부모 노드를 직접 접근하여 상향식 갱신을 수행한다. 또한, 보조 색인 구조를 이용하여 이동 객체가 저장되어 있는 단말 노드를 직접 접근할 수 있도록 하였다. 실험을 통해 제안된 색인 구조는 기존 TPR-트리에 비해 갱신 및 검색 성능이 향상됨을 보였다. 향후 연구 방향으로 색인 구조를 통한 다양한 질의 처리 기법에 대한 연구를 수행할 예정이다.

참 고 문 헌

[1] B. C. Ooi, K. L. Tan and C. Yu, "Fast update and efficient retrieval: an oxymoron for moving object indexes?," Proc. 3rd International Conference on Web Information Systems Engineering Workshops, pp. 3-12, 2002.

[2] M. Vazirgiannis, Y. Theodoridis and T. Sellis, "Spatio-Temporal Composition and Indexing for Large Multimedia Applications," ACM Multimedia Systems, Vol.6, No.4, pp. 284-298, 1998.

[3] D. Pfoser, C. S. Jensen and Y. Theodoridis, "Novel Approaches to the Indexing of Moving Object Trajectories," Proc. 26th International Conference on Very Large Databases, pp. 395-406, 2000.

[4] M. A. Nascimento and J. R. O. Silva, "Towards Historical R-trees," Proc. ACM Symposium on Applied Computing, pp. 235-240, 1998.

[5] D. Pfoser, "Indexing the Trajectories of Moving Objects," Data Engineering Bulletin, Vol.25, No.2, pp. 4-10, 2002.

[6] V. Prasad Chakka, A. Everspaugh and J. M. Patel, "Indexing Large Trajectory Data Sets With SETI", Proc. Conference on Innovative Data ..stems Research, 2003.

[7] Z. Song and N. Roussopoulos, "SEB-tree: An Approach to Index Continuously Moving Objects," Proc. 4th International Conference on Mobile Data Management, pp. 340-344, 2003.

[8] S. Prabhakar, Y. Xia, D. Kalashnikov, W. Aref and S. E. Hambrusch, "Query Indexing and Velocity Constrained Indexing : Scalable Techniques for Continuous Queries on Moving Objects," IEEE Transactions on Computers, Vol.51, No.10, pp. 1124-1140, 2002.

[9] S. Saltenis, C. S. Jensen, S. T. Leutenegger and M. A. Lopez, "Indexing the Positions of Continuously Moving Objects," Proc. ACM SIGMOD International Conference on Management of Data, pp. 331-342, 2000.

[10] Y. Tao, D. Papadias and J. Sun, "The TPR*-Tree: An Optimized Spatio-Temporal Access Method for Predictive Queries," Proc. 29th Very Large Data Bases Conference, pp. 790-801, 2003.

[11] D. S. Kwon, S. J. Lee and S. H. Lee, "Indexing the Current Positions of Moving Objects Using the Lazy Update R-tree," Proc. International Conference on Mobile Data Management, pp. 113-120, 2002.

[12] M. L. Lee, W. Hsu, C. S. Jensen, B. Cui, and K. L. Kenglik, "Supporting Frequent Updates in R-Trees : A Bottom-Up Approach," Proc. 29th International Conference on Very Large Data Bases, pp. 608-619, 2003.

[13] http://db.cs.ualberta.ca:8080/gstd

저 자 소 개

복 경 수(Kyoung-Soo Bok) 정회원

- 1998년 2월 : 충북대학교 수학과 (이학사)
- 2000년 2월 : 충북대학교 정보통신공학과(공학석사)
- 2000년 3월~현재 : 충북대학교 정보통신공학과 박사과정

<관심분야> : 데이터베이스 시스템, 자료 저장 시스템, 멀티미디어 검색 시스템, 이동 객체 데이터베이스, 고차원 색인 구조, 시공간 색인 구조 등

서 동 민(Dong-Min Seo) 정회원

- 2002년 2월 : 충북대학교 정보통신공학과(공학사)
- 2004년 2월 : 충북대학교 정보통신공학과(공학석사)
- 2004년 3월~현재 : 충북대학교 정보통신공학과 박사과정

<관심분야> : 데이터베이스 시스템, 에이전트 시스템, XML, 이동 객체 데이터베이스, 고차원 색인 구조, 시공간 색인 구조 등

유 재 수(Jae-Soo Yoo) 종신회원

- 1989년 2월 : 전북대학교 컴퓨터공학과(이학사)
- 1991년 2월 : 한국과학기술원 전산과(공학석사)
- 1995년 2월 : 한국과학기술원 전산과(공학박사)
- 1995년 3월~1996년 8월 : 목포대학교 전산통계학과 전임강사
- 1996년 9월~현재 : 충북대학교 전기전자컴퓨터공학부 및 컴퓨터정보통신연구소 부교수

<관심분야> : 데이터베이스 시스템, 정보 검색, 멀티미디어 검색 시스템, 분산 객체 시스템, 자료 저장 시스템 등