차량 위치 정보 저장을 위한 버퍼 노드 기반 그룹 갱신 기법

(A Group Update Technique based on a Buffer Node to Store a Vehicle Location Information)

정 영 진 † 류 근 호 ††
(Jung YoungJin) (Ryu KeunHo)

요 약 GPS 및 무선 통신 기술의 진보와 네트워크의 활성화 및 단말기의 소형화를 통해 이동하는 차량의 위치 추적이 가능해지고, 위치를 기반으로 한 서비스 개발이 활발하게 이루어지고 있다. 이러한 위치 기반 서비스를 제공하기 위해서는 수많은 이동 객체 데이타를 빠르게 저장, 검색하기 위한 색인 기술이 필수적이다. 그러나 기존의 이동 객체 색인은 각각의 입력 받는 위치 정보를 직접 색인에 입력하기 때문에 많은 비용이 든다. 따라서 이 논문에서는 이와 같은 문제점을 해결하기 위해, 색인의 입력 비용을 효과적으로 줄이는 버퍼 노드 방식을 제안하고, 이를 활용한 GU-트리를 설계한다. 제안된 버퍼 노드 방식은 입력 받는 이동 객체 정보를 넌 리프 노드 단위로 그룹화하여 색인에 저장함으로써 데이타를 입력하는 시간을 효과적으로 줄인다. 그리고 기존의 색인과 비교 실험함으로써, 버퍼 노드 입력 방식이 이동 객체 색인의 데이타 입력 비용을 효과적으로 줄이고, 특정 시점 질의에서 검색의 성능을 높이는 것을 확인한다. 제안된 버퍼 노드 방식은 여행 가이드 및 물류 차량 관리 시스템과 같이 빈번한 위치 갱신이 이루어지는 환경에서 효과적으로 사용될 수 있다.

키워드 : 시공간 데이타베이스, 이동 객체, 이동 객체 색인, 버퍼 노드, 위치 갱신

Abstract It is possible to track the moving vehicle as well as to develop the location based services actively according to the progress of wireless telecommunication and GPS, to the spread of network, and to the miniaturization of cellular phone. To provide these location based services, it is necessary for an index technique to store and search too much moving object data rapidly. However the existing indices require a lot of costs to insert the data because they store every position data into the index directly. To solve this problem in this paper, we propose a buffer node operation and design a GU-tree(Group Update tree). The proposed buffer node method reduces the input cost effectively since the operation stores the moving object location data in a group, the buffer node as the unit of a non-leaf node. And then we confirm the effect of the buffer node operation which reduces the insert cost and increase the search performance in a time slice query from the experiment to compare the operation with some existing indices. The proposed buffer node operation would be useful in the environment to update locations frequently such as a transportation vehicle management and a tour-guide system.

Key words : Spatio-temporal database, Moving objects, Indexing of moving objects Buffer node, Location update

1. 서 론

지도 및 위치를 다루는 연구는 인류의 역사와 더불어 탐험 및 여행, 교통, 전쟁 등을 통해 계속되어 왔다. 최근 GPS(Global Positioning System)를 활용한 위치 측위 기술의 발달, 무선 컴퓨팅 기술의 진보, 그리고 휴대용 전화기 및 무선 단말기의 확산에 힘입어, 위치를 다루고 활용하는 연구가 매우 활발히 이루어지고 있다. 특

· 이 논문은 2004년도 충북대학교 학술연구지원사업의 연구비 지원에 의하여 수행되었음
† 학생회원 : 충북대학교 전자계산학과
yjjeong@dblab.cbu.ac.kr
†† 종신회원 : 충북대학교 전자계산학과 교수
khryu@dblab.cbu.ac.kr
논문접수 : 2004년 12월 1일
심사완료 : 2005년 9월 10일

히 차량 및 비행기와 같이 이동하는 객체를 추적하고, 객체의 변화하는 위치에 따라 적절한 서비스를 제공하는 위치기반서비스(Location Based Services, LBS)가 무선 인터넷 시장의 중요한 이슈가 되고 있다[1].

이와 같이 시간의 흐름에 따라 위치 및 모양이 변화하는 객체를 이동 객체(Moving Object)[2,3]라 하며, 이동 객체의 위치 정보를 다루는 기술이 LBS에서 가장 기본적이며 핵심이다. 특히 이동 객체에 대한 위치 정보는 시간의 흐름에 따라 그 변화량이 방대하게 증가되기 때문에 대용량 데이타를 관리하기 위한 데이타베이스 시스템의 활용이 반드시 필요하다. 또한 연속적으로 위치 등의 정보가 변경되는 이동 객체를 데이타베이스를 이용하여 저장 및 관리하기 위해서는 연속적인 객체의 위치 정보변화를 데이타베이스 내에 표현하고, 이를 검색 및 질의처리 할 수 있어야 한다. 따라서 저장된 대용량의 이동 객체 정보를 효율적으로 다루기 위한 색인 기법이 필요하며, 많은 연구가 진행되고 있다[2].

이동체 색인에 대한 많은 연구들은 공간 데이타나 그 밖의 멀티미디어 데이타 에 대한 색인들로부터 아이디어를 얻고 있으며, 그 중 B-트리의 확장한 높이 균형 트리인 R-트리[4]에 기반한 많은 색인들이 제안되고 있다[5,6]. 시간의 흐름에 따라 이동 객체 정보가 끊임없이 생성되는 이동 객체 환경에선 질의 처리 요청보다 데이타 입력에 대한 요청이 상대적으로 많기 때문에 사용자의 질의를 효과적으로 처리하는 것과 함께 데이타의 입력 비용을 줄이는 데에도 많은 연구가 계속되고 있으며, 기존의 이동 객체 색인은 크게 Top-down 방식과 Bottom-Up 방식을 사용한다. 그러나 위 방식에서 입력받는 데이타 하나하나를 리프 노드 단위로 입력하기 때문에, 데이타 입력이 매우 빈번하게 이루어지는 단점이 있다.

따라서 이 논문에서는 이와 같은 문제점을 해결하기 위하여 효과적인 데이타 입력을 위한 버퍼 노드 연산을 제안하고, 이를 활용한 GU-트리를 설계한다. 제안된 기법은 리프 노드 단위로 입력되는 기존의 방식과는 달리, 입력 받는 리프 노드를 넘 리프 노드 단위로 그룹화하여 입력함으로써, 불필요한 입력 비용을 줄일 수 있으며 이를 위해 많은 저장 공간을 필요로 하지 않는다.

이 논문의 전체적인 구성은 다음과 같다. 먼저 2장에서는 기존 이동 객체 색인들의 입력 방식 및 그 문제점들을 알아본다. 3장에서는 이 논문에서 제안한 버퍼 노드 연산과 GU-트리를 소개하고 그 효과에 대해 설명한다. 4장에서는 색인 입력에서 사용되는 버퍼 노드 입력 방식의 알고리즘에 대해서 살펴본다. 5장에서는 제안된 버퍼 노드 방식을 기존의 Top-down 방식 및 Bottom-Up 방식과 비교하여 그 효율성을 검토한다. 6장에서는 이동 객체 데이타를 입력하고 질의하여 GU-트리와 기존의 R-트리의 성능을 비교 실험한다. 마지막으로 7장에서는 결론을 맺는다.

2. 관련 연구

이동 객체[7]는 그 위치 및 속성이 매우 빈번하게 변화하는 특성을 가지고 있기 때문에, 이동 객체의 위치에 따른 서비스를 원활히 제공하기 위해서는 저장된 이동 객체 정보를 빠르게 검색해야 하며, 이를 위해 대용량의 이동 객체 정보를 효율적으로 다루기 위한 이동 색인 기법이 많이 연구되고 있다[2]. 기존의 이동 객체 정보를 다루는 색인들은 응용에 따라 크게 이동 객체의 과거 이력정보 및 궤적 정보를 다루는 색인과 이동 객체의 현재 위치 및 가까운 미래의 위치를 다루는 색인으로 나누어진다[8]. 이들은 대부분 R-트리를 기반으로 하며 이동 객체의 궤적 정보를 라인 세그먼트로 분해하여 다룬다. 그리고 이 논문에서는 이동 객체의 과거 이력 및 궤적 정보를 다루는 색인에 초점을 맞춘다.

그리고 대부분의 위치 기반 시스템에선, 질의 처리 요청보다 이동 객체 데이타 입력에 대한 요청이 보다 빈번하게 일어나기 때문에, 빠른 데이타 검색뿐만 아니라 데이타 입력 비용을 줄이기 위한 노력들이 계속되고 있다. 기존의 이동 객체 색인의 입력방식은 크게 Top-down 방식과 Bottom-Up 방식 두 가지로 나뉜다. Top-down 방식은 전형적인 입력 형태로 루트노드로부터 하위 노드로 검색하며 데이타 삽입, 삭제, 갱신이 이루어지는 진다. 이 방식을 사용하는 색인으로는 STR-트리(Spatio-Temporal R-tree)[9], TB-트리(Trajectory Bundle Tree)[8], TPR-트리(Time Parameterized R-tree)[10], MP-트리(Moving Point tree)[11] 등이 있다. STR-트리는 이동 객체의 좌표와 궤적 보호를 고려하여 노드를 분할하였으나 오히려 검색 성능이 R-트리보다 떨어진다. TB-트리는 노드간의 링크를 연결하여 이동 객체 궤적에 대한 성능을 높였다. MP-트리는 하위 노드를 각 축에 대한 순서대로 저장하는 Projection *Storage*를 활용하여 특정 시점에 대한 성능을 높였다. TPR-트리는 현재 및 미래 위치를 다루는 색인으로 선형함수를 활용하여 빈번한 갱신을 줄였다.

Bottom-Up 방식은 루트로부터 리프 노드를 찾는 비용을 줄이기 위한 방법으로 해시 테이블을 통해 리프 노드에 직접 이동 객체 데이타를 입력하며, 상위 노드로 갱신하는 방식이다. 이 방식은 [6]과 [5]에서 LUR-트리와 Bottom-Up update 방식으로 소개된 뒤, TB*-tree[12]에서와 같이 기존의 색인 기법에 이를 적용한 연구들이 활발히 계속되고 있다. 또한 갱신 효율을 위해 MBR을 확장하도록 하였지만 오히려 R-트리보다 검색

효율이 떨어지는 단점이 있다. 그리고 상위 노드를 탐색하기 위한 포인터와 해시테이블로 인해 색인의 크기 및 유지비용이 더 드는 문제점이 있다.

기존 이동 객체 색인들의 입력 방식들은 리프 노드에 입력되는 라인 세그먼트 즉, 리프 노드 단위로 입력된다. 이로 인해 각 라인 세그먼트가 입력되고 갱신 될 때마다, 색인의 입력 및 갱신 비용이 증가하며 색인의 크기 조정도 매우 빈번하게 이루어진다. 또한 공간 색인에서는 R-트리의 입력 비용을 감소시킨 STLT[13,14], 등이 있지만 별도의 클러스터링 비용이 필요하며 시간에 따른 데이타 값의 변화를 고려하지 않아 이동 객체 색인에 적용하기에는 적절치 못하다. 따라서 이 논문에서는 이러한 문제점을 해결하기 위하여 새로운 버퍼 노드 입력 방식을 제안하고, 제안한 버퍼 노드 방식을 적용한 GU-트리(Group Update Tree)를 설계한다. 그리고 기존의 Top-down 방식 및 Bottom-Up 방식과 비교하여 그 효율성을 알아본다.

3. 버퍼 노드를 활용한 GU-트리

이 논문에서는 시간에 따라 이동 객체 데이타가 끊임없이 서버로 전송된다고 가정하며, 이동 점 객체의 움직임만을 고려한다. 그리고 과거 이력 및 현재의 위치 정보를 다루는데 초점을 맞춘다. 이 장에서는 GU-트리의 구조와 빈번한 입력 횟수를 효과적으로 줄이기 위해 제안된 버퍼 노드 입력 방식에 대해 설명한다.

3.1 GU-트리의 구조

제안된 GU-트리는 기존의 이동 객체 색인에 버퍼 노드 연산을 사용하여 빈번한 갱신 비용을 줄이고, 특정 시점 질의 및 시간 범위 질의를 효과적인 처리하는 이동 객체 색인이다. 제안된 이동 객체 색인 구조는 그림 1과 같이 넌 리프 노드와 리프 노드 그리고 이동 객체 데이타를 모아서 한꺼번에 색인에 입력하는 버퍼 노드로 구성된다. 사용된 버퍼 노드 방식은 임시로 버퍼 역할을 하는 넌 리프 노드를 생성하고 이동 객체 데이타를 입력한 뒤, 버퍼 노드가 꽉 차면 이를 다시 색인에 입력하는 방식이다. 이와 같이 그룹화된 버퍼 노드 단위로 이동 객체 데이타를 입력하여 색인의 빈번한 데이타 입력 횟수를 효과적으로 줄인다. 더불어 색인의 노드들이 시간 순서대로 정렬되는 효과를 얻을 수 있기 때문에, 특정 시점 질의 및 시간 범위 질의에서 빠르게 노드를 검색할 수 있다. 이때 리프 노드는 이미 버퍼 노드에 저장이 되어 있기 때문에, 색인에 버퍼 노드를 입력할 때는 넌 리프 노드 중 가장 아래 단계 까지만 입력된다. 따라서 기존의 이동 객체 색인들이 각각의 차량 데이타를 일일이 리프 노드 단계까지 입력하는 것과 비교해 볼 때, 입력 비용이 효과적으로 감소함을 알 수 있다.

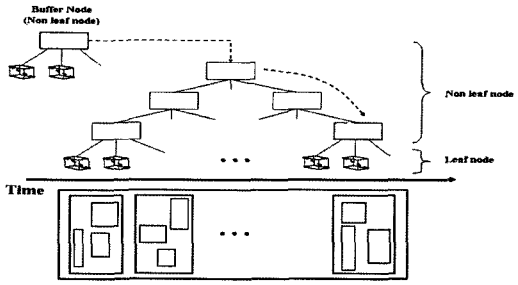

그림 1 버퍼 노드를 활용한 GU-트리의 구조

그림 1은 GU-트리의 구조를 나타낸다. 입력받는 차량의 데이타는 먼저 버퍼 노드에 저장되기 때문에, 버퍼 노드는 항상 최신의 현재 정보를 갖고 있게 된다. 따라서 색인에 입력될 때는 시간이 흐르는 쪽의 가장 앞에 저장된다. 그리고 노드가 가득차게 되면, 노드를 분할하지 않고 새로운 노드를 생성한다. 이와 같은 입력 방식으로 인하여 보통 리프 노드 단계에서 시작되는 새로운 노드의 생성 및 노드 분할이 GU-트리에선 넌 리프 노드 단계에서 고려됨으로써 데이타 입력 및 색인의 균형 조정 비용을 감소시킨다. 또한 이로 인해 트리의 노드들이 시간 순으로 정렬되는 효과를 얻을 수 있다. 그림 1에서는 Top-down 입력방식에 버퍼 노드를 적용하였지만, Bottom-Up 방식에도 역시 이와 유사하게 활용할 수 있다.

3.2 버퍼 노드 입력 방식

기존의 이동 객체 색인에서 빈번하게 이루어지는 데이타 갱신과 색인의 구조 조정 비용을 줄이기 위해, 제안된 버퍼 노드 입력 방식은 그림 2와 같이 기존의 이동 객체 색인에서 입력 받는 라인 세그먼트 단위의 MBB(Minimum Bounded Box)로 입력하던 것을 버퍼 노드라는 넌 리프 노드 단위로 그룹화하여 입력함으로써, 입력비용을 줄이는 방법이다.

그림 2는 입력 받는 데이타를 그룹화하여 색인에 입력하는 버퍼 노드 방식을 나타낸다. 차량의 데이타는 버

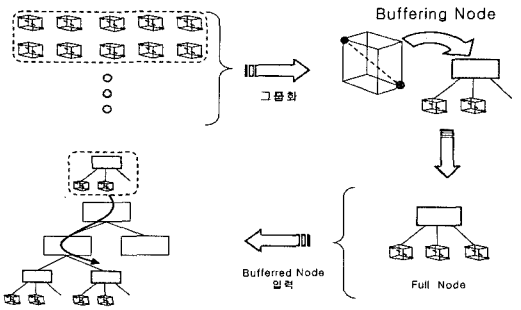

그림 2 버퍼 노드를 활용한 입력 방식

퍼 노드가 가득 찰 때까지 모아진 후, 색인에 입력된다. 따라서 실제로 트리에 입력하는 노드 개수는 전체 차량 데이타를 넌 리프 노드의 차수로 나눈 만큼 줄어들다. 그리고 버퍼 노드는 이미 넌 리프 노드이기 때문에, 리프 노드의 바로 위 상위 노드까지 입력됨으로써, 기존의 색인보다 데이타가 입력되는 깊이를 한 단계 줄였다. 각 객체의 OID는 리프 노드에 저장되어 질의 처리할 때 활용된다. 다음 장에서는 이러한 버퍼 노드를 구현하기 위해 사용된 알고리즘에 대하여 설명한다.

4. GU-트리의 알고리즘

이동 점 객체의 과거 및 현재 위치를 다루는 GU-트리에서 사용된 버퍼 노드 삽입 알고리즘은 먼저 버퍼 노드라는 임시 노드를 만들어 입력받는 라인 세그먼트를 리프 노드 MBB로 입력 받은 후, 이동 객체 색인의 가장 최근의 시간대에 버퍼 노드를 입력한다. 그리고 노드가 가득 차게 되면, 노드 분할을 하지 않고 새로운 노드를 만들어 입력한다.

```
Algorithm insert_buffer node(class node *root, class node *entry)
input: root     // The node of a Tree
       entry // The information of moving objects
method:
 if BufferNode is null then              // BufferNode is global variable pointer
    make new BufferNode
 else
    if the entry's boundary is not included in BufferNode's boundary then
       modify BufferNode's boundary with the entry's boundary
    endif

    if BufferNode's count < M then
       insert the entry into BufferNode
    else
       insert the entry into BufferNode // BufferNode is full
       insert_node(root, BufferNode) // insert full BufferNode into R-tree
       make BufferNode null
    endif
 endif
end
```

알고리즘 1. 버퍼 노드 삽입 알고리즘

GU-트리의 버퍼 노드 입력 과정은 알고리즘 1을 통해 알 수 있다. 이는 일반적인 R-트리 알고리즘에 버퍼 노드라는 임시 저장장치를 추가한 형태이다. 이러한 버퍼 노드를 활용함에 따라 데이타 입력 비용이 줄어들게 되고, 특정 시점 질의 및 시공간 범위 질의에서 대체적으로 기존의 R-트리보다 좋은 성능을 보였다. 이는 버퍼 노드를 활용하여 단순화된 알고리즘으로 인해, 색인의 노드들이 시간 축에 대해 좀 더 정렬이 되기 때문으로 보인다. 또한 버퍼 노드의 크기가 너무 크면, 색인의 입력 효율은 좋아질 수 있으나 버퍼 노드를 검색하는데 오랜 시간이 걸리기 때문에 적당한 크기를 유지하는 것이 중요하다.

```
Algorithm search_node(class node *root, class node *query)
input: root     // The node of a Tree
       query // The ranges of query
method:
 if the root's boundary overlap query's boundary then
    if the root is a leaf node then
       return root
    else
       search_node( the intersected child node, query)
    endif
 endif

 // BufferNode is global variable pointer
 if the query's boundary overlap BufferNode's boundary then
       return the intersected child nodes of BufferNode
 endif
end
```

알고리즘 2. GU-트리의 검색 알고리즘

알고리즘 2는 GU-트리의 검색 과정을 보여준다. GU-트리의 버퍼 노드는 주로 입력 과정에서 쓰이기 때문에, 버퍼 노드 검색 부분을 제외한 나머지 과정은 일반적인 R-트리와 유사하다. 단, 이 논문에서는 시간은 무한하고, 공간은 유한하다고 가정하고 실험을 하기 때문에, 질의 범위를 체크할 때 시간 축의 범위를 먼저 검사하여 상대적으로 불필요한 노드 검색을 줄인다. 그리고 보통 이동 객체 데이타는 시간에 따라 연속적으로 들어오기 때문에, 이동 객체의 과거 및 현재의 위치를 다루는 색인은 시간 축으로 점점 자라나는 형태를 가진다. 따라서 GU-트리에서 질의를 처리할 때는 시간 값을 먼저 체크하여 상대적으로 긴 시간 축에서 부적합한 후보들을 우선적으로 제거한다.

5. 버퍼 노드 방식과 기존 방식과의 비교

이 장에서는 제안된 버퍼 노드 입력 방식의 효율성을 알아보기 위해, 기존의 Top-down 방식 및 Bottom-Up 방식과 비교하여 설명한다. 이를 위해 정의된 기호들을 다음과 같다. K는 입력받는 차량의 라인 세그먼트 수이고 M은 노드의 최대 값이다. 버퍼 노드는 M 만큼 이동 객체 데이타를 입력받은 후에 색인에 저장된다. 그리고 이동 객체 색인을 구성하는 레코드 수 N과 R-트리의 최대 높이인 $\lceil \log_m N \rceil - 1$ [4]를 사용하여 입력 비용을 분석한다.

5.1 Top-down 기법과 버퍼 노드 기법의 비교

Top-down 방식은 일반적으로 널리 쓰이는 방법으로, 루트 노드에서부터 리프 노드까지 적절한 하위 노드를 찾아 데이타를 입력한다. 그리고 버퍼 노드 방식은 리프 노드들을 모아 한꺼번에 저장하는 방식이다. 두 색인의 비교를 N 개의 레코드를 가진 색인에 K 개의 데이타를

입력한다고 가정하고 아래 표 1과 같이 입력 비용을 계산하였다.

표 1은 Top down 방식과 버퍼 노드 방식의 입력 비용 계산식을 나타낸 것이다. Top-down 방식은 입력받는 모든 데이타를 그대로 색인에 입력하기 때문에, 모든 데이타가 트리의 높이만큼의 노드를 거쳐 저장된다. 그러나 버퍼 노드 방식은 먼저 버퍼 노드에서 데이타 개수(K)만큼 입력 비용이 들고, 생성된 버퍼 노드의 수(K/M)를 그대로 색인에 입력하는 비용이 추가된다. 이 때, 버퍼 노드는 넌 리프 노드이므로 트리의 높이가 한 단계 줄어들게 된다. 제안된 버퍼 노드 기법의 효율성을 알아보기 위해, 일반 Top-down 입력 비용에서 버퍼 노드 입력 비용을 빼면, 아래와 같은 결과를 얻을 수 있다.

비용 차이 = Top-down 비용 - 버퍼 노드 비용

$$= (K - \frac{K}{M}) \times (\lceil \log_m N \rceil - 2)$$

위 식을 풀이하면, K-K/M > 0 그리고 $\lceil \log_m N \rceil - 2 > 0$ 하면, 버퍼 노드 방식의 효율이 좋다고 생각할 수 있다. K 는 입력 데이타 개수이고, M은 노드의 최대 값이므로 항상 K - K/M > 0가 성립함을 알 수 있다. 그리고 $\lceil \log_m N \rceil > 2$가 성립하려면 $N \geq m^3$ ($\therefore \lceil \log_m m^3 \rceil - 1$)를 만족하면 된다. 이로 인해, 트리의 높이가 2 이상 되면, 버퍼 노드 방식이 효과가 있음을 알 수 있다.

표 2는 Top down 방식과 버퍼 노드 방식의 입력 비용 차이를 샘플 값을 넣어 확인한 것이다. 입력 비용 및 비용 차이를 나타낸 항목에서 버퍼 노드 방식이 매우 효과적으로 입력 비용을 줄이는 것을 확인할 수 있으며, 결과를 정리해보면 K, M의 값이 클수록 효율이 높아지는 것을 확인 할 수 있다. 즉, 입력받는 데이타의 수(K)와 노드의 차수(B)가 커지면 커질수록, 버퍼 노드 기법을 사용하는 색인의 효율은 높아진다는 것을 알 수 있다.

그림 3은 표 2에서의 가정을 기반으로 데이타가 입력되는 과정을 나타내며, 그에 따른 비용을 분석한 것이다. Top-down 방식은 입력받는 데이타를 그대로 색인에 입력하기 때문에, 각각의 데이타마다 트리의 높이만큼의 색인의 입력 비용이 필요하다. 따라서 입력받는 데이타의 개수(K) * 트리의 높이만큼의 입력 비용이 필요하며, 대부분의 색인에서도 입력받는 데이타 개수에 따라 입력 비용과 색인 유지 비용이 비례적으로 증가한다. 그러나 버퍼 노드 방식은 입력 받는 데이타를 버퍼 노드에 모아 저장함으로써, 입력 받는 데이타를 버퍼 노드의 차수로 나눈 만큼만 색인의 입력 비용이 필요하게 된다.

5.2 Bottom-Up 기법과 버퍼 노드 기법의 비교

LUR-트리 및 Bottom-Up 방식은 리프 노드 및 넌 리프 노드를 가리키는 해시 테이블을 활용하여 직접 적절한 리프노드에 데이타를 입력함으로써 입력 비용을

표 1 Top-down 방식과 버퍼 노드 방식의 입력 비용 계산 식

입력 방식	비용 계산
Top down 방식	입력 비용 = 입력 데이타 개수 × 트리의 높이 $= K \times (\lceil \log_m N \rceil - 1)$
버퍼 노드 방식	입력 비용 = 버퍼 노드 입력비용 + 트리 입력 비용 $= K + \frac{K}{M} \times (\lceil \log_m N \rceil - 2)$ $\therefore$ 트리 입력 비용 = 입력 버퍼 노드 개수 × 트리의 높이 $= \frac{K}{M} \times \{ (\lceil \log_m N \rceil - 1) - 1 \}$

표 2 Top-down 방식과 버퍼 노드 방식의 입력 비용 차이 비교

입력 방식	비용 계산
가 정	K = 100,000, m = 10, M = 20, N = 1,000,000 $\lceil \log_m N \rceil - 1 = \lceil \log_{10} 1000000 \rceil - 1 = \lceil \log_{10} 10^6 \rceil - 1 = 5$
Top down 방식	입력 비용 = $K \times (\lceil \log_m N \rceil - 1)$ = 100,000 × 5 = 500,000
버퍼 노드 방식	입력 비용 = $K + \frac{K}{M} \times (\lceil \log_m N \rceil - 2)$ = 100,000 + (100,000/20) × (6-2) = 120,000
Top-down 방식과 버퍼 노드 방식의 비용 차이	비용 차이 = $(K - \frac{K}{M}) \times (\lceil \log_m N \rceil - 2)$ = (100,000 - 100,000/20) × 4 = 380,000

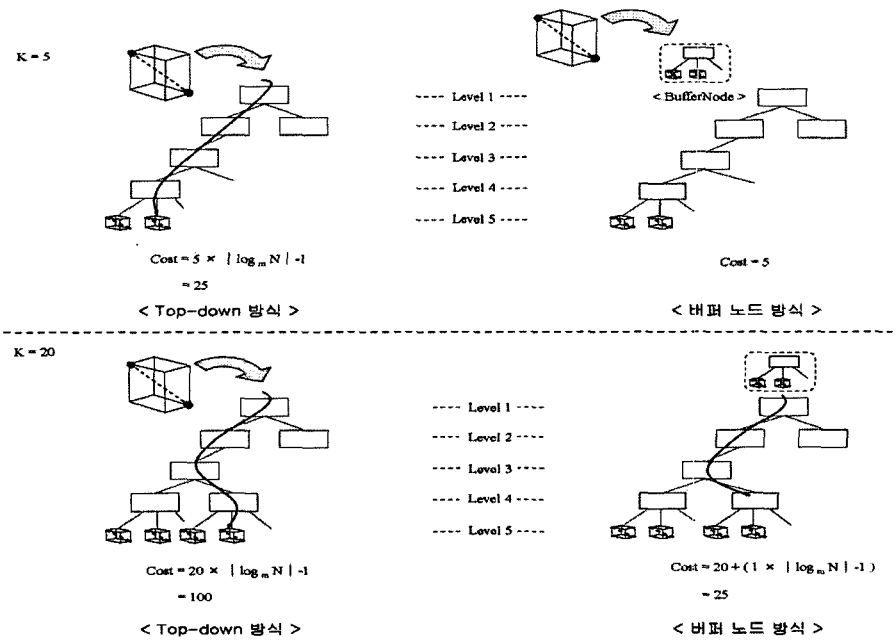

그림 3 Top-down 방식과 버퍼 노드 방식에서의 입력 비용 분석

줄이는데 그 목적을 갖고 있다. 그러나 Bottom-Up 방식은 다음과 같은 문제점이 있다.

1. 해시 테이블을 갖기 때문에 색인의 크기가 커진다.
2. 리프 노드의 경계를 갱신하면, 그 효과가 상위 노드까지 계속 이어져, 가장 안 좋은 경우에는 Top-down 방식과 같은 입력 비용이 든다.
3. 상위 노드를 가리키기 위한 포인터가 있어야 한다. 역시 색인의 크기를 크게 한다.
4. 경계의 Overlap으로 인하여 검색 성능이 Top-down 방식보다 떨어진다.

이 논문에서는 LUR-트리 및 Bottom-Up 방식이 제안되었던 현재 데이타를 다루는 색인 환경이 아닌, 과거 이력 및 현재 위치를 다루는 색인에서 Bottom-Up 기법을 적용하였기 때문에, 한번 저장된 정보는 바뀌지 않는다. 따라서 리프 노드의 경계가 상위 노드의 경계보다 더 커지면, 무조건 상위 노드의 경계를 갱신하는 Bottom-Up 기법을 사용하였다. 이로 인해 위 문제점 중 4번은 고려 대상에서 제외된다.

표 3은 Bottom Up 방식과 버퍼 노드 방식의 입력 비용 계산식을 나타낸 것이다. Bottom-Up 방식은 입력받는 데이타를 해시 테이블을 통해 리프 노드에 입력하고, 리프 노드의 경계가 상위 노드를 벗어나게 되면, 상위 노드를 갱신하는 것으로 입력 과정을 마친다.

그리고 상위 노드 갱신 비용은 들어오는 데이타의 좌표 값 및 시간 값에 따라 달라지므로 정확하게 그 비용을 계산하기 어렵다. 따라서 이 논문에서는 $C_{UpdateMBB}$로 그 값을 표현한다. 다만 경계 갱신 비용 값의 범위가 $0 \leq C_{UpdateMBB} \leq |\log_m N| - 2$ (∴ 트리의 높이 - 1)인 것은 알 수 있다.

표 3 Bottom-Up 방식과 버퍼 노드 방식의 입력 비용 계산 식

입력 방식	비용 계산
Bottom-Up 방식	입력 비용 = 입력 데이타 개수 × 트리 입력 비용 $= K \times (1 + 1 + C_{UpdateMBB}) = K \times (2 + C_{UpdateMBB})$ ∴ 트리 입력 비용 = 해시체크 + 리프노드 입력 + 상위노드 경계 갱신
버퍼 노드 방식	입력 비용 = 버퍼 노드 입력 비용 + 트리 입력 비용 $= K + \frac{K}{M} \times (1 + C_{UpdateMBB})$ ∴ 트리 입력 비용 = (해시체크 + 리프노드 입력 + 상위노드 경계 갱신) × 입력 버퍼 노드 개수 $= \frac{K}{M} \times (1 + 1 + C_{UpdateMBB} - 1)$

버퍼 노드 방식은 Top-down 방식과 마찬가지로 먼저 버퍼 노드에서 데이타 개수(K)만큼 입력 비용이 들고, 생성된 버퍼 노드의 수(K/M)를 그대로 색인에 입력하는 비용이 추가된다. 이때, 버퍼 노드가 넌 리프 노드이므로 상위 노드 경계 갱신 비용이 한 단계 줄어들어 $C_{UpdateMBB}$ - 1이 된다. 그러므로 경계 갱신 비용의 값의 범위는 $0 \le C_{UpdateMBB} \le |\log_m N| - 3$ (∴ 트리의 높이 - 2)이다. 제안된 버퍼 노드 기법의 효율성을 알아보기 위해, 일반 Bottom-Up 입력 비용에서 버퍼 노드 기법 비용을 빼면, 아래와 같은 결과를 얻을 수 있다.

비용 차이 = Bottom-Up 비용 - 버퍼 노드 비용

$$= (K - \frac{K}{M}) \times (C_{UpdateMBB} + 1)$$

위 식을 풀이하면, $K - K/M > 0$ 그리고, $C_{UpdateMBB} + 1 > 0$하면, 버퍼 노드 방식의 효율이 좋다고 생각할 수 있다. 항상 $K - K/M > 0$가 성립하는 것은 앞서 Top-down 방식과의 비교에서 알 수 있었다. 그리고 $C_{UpdateMBB}$ 값은 입력 받는 데이타에 따라 변화하므로 1 보다 클 때를 정확히 나타내기 어렵다. 다만 노드의 경계 값은 시간의 흐름에 따라 계속 증가하고, 과거 및 현재의 위치를 다루는 이동 객체 색인에서는 많은 데이타 수로 인해 대부분 트리의 높이가 2 이상이 되기 때문에, 버퍼 노드 방식이 효과적인 가능성이 많아 보인다. 이 $C_{UpdateMBB}$ 값의 모호함 때문에, 샘플 값을 통한 비교 평가는 하지 않는다. 그러나 6 장의 입력 비용 실험 평가에서 버퍼 노드 방식이 Bottom-Up 방식보다 효과적인 것을 확인할 수 있다.

5.3 버퍼 노드의 크기 분석

버퍼 노드는 넌 리프 노드이며, 버퍼 노드의 크기는 넌 리프 노드의 차수(M)와 같다. 이 차수의 크기는 트리의 높이를 변화시키기 때문에, 색인의 입력 비용에 영향을 주게 된다. 예를 들어, 입력받는 차량의 데이타 수(K)가 10만이고 이동 객체 색인을 구성하는 레코드 수(N)가 100만인 경우, m은 10으로 두고 M을 10에서 100으로 변경하면, 버퍼 노드의 수(K/M)가 5,000에서 1,000으로 줄어든다. 이를 표 3의 입력 비용 식에 대입해 보면 색인의 입력 비용은 120,000에서 104,000으로 줄어들게 되지만, 색인의 리프 노드에서 원하는 데이타를 찾는 검색 비용은 늘어난 M만큼 증가하게 된다.

$$M = \frac{(\text{입력비용} - K)M^2}{(|\log_m N| - 2)K} \quad (1)$$

식 (1)은 입력비용과 M의 관계를 보여주고 있으며, 표 3의 입력 비용 식으로부터 유도된 것이다. M의 크기 변화에 따른 입력 비용 변화는 위 식과 앞의 예를 통해 알 수 있다. 정리하면, 데이타의 입력 요청이 검색 요청보다 월등히 많을 경우, M 값을 늘이는 것을 고려해 볼 수 있으며, 이 점은 실 세계의 시스템에서 차량의 개수와 사용자의 검색 활용도에 따라 달라 질 것이다. 그리고 만약 버퍼 노드의 크기가 넌 리프 노드보다 크다면, 버퍼 노드를 그대로 색인에 저장하질 못해 버퍼 노드의 하위 노드들을 분할하고 색인의 리프 노드로 다시 복사하는 비용이 필요하다. 즉, 분할 비용 외에도 입력된 데이타의 개수(K)만큼의 복사 비용이 더 들게 되어 버퍼 노드의 활용성이 떨어진다. 이는 버퍼 노드가 넌 리프 노드보다 작을 경우에도 마찬가지이다. 따라서 버퍼 노드가 넌 리프 노드의 크기를 유지하는 것이 입력 비용을 줄이는 데 도움이 된다. 또한, 공간 색인의 STLT [13,14]와 같이 단순히 1단계의 버퍼 노드가 아닌 여러 단계의 노드를 합친 작은 트리를 입력하는 방식도 고려되었지만, 이와 같은 입력 단위에 대한 설정은 실세계의 차량 관리 시스템에서 사용자의 요구 분석을 통해 적절히 조절되어야 할 것이다.

6. 실험 및 평가

이 장에서는 버퍼 노드 연산을 활용한 GU-트리의 효율성을 알아보기 위하여, 윈도우2000 및 xp 환경에서 C 언어로 색인을 구현하여 실험하였으며, 컴퓨터 성능 등에 따라 결과 값이 달라지는 것을 지양하여 검색 속도에 대한 측정이 아닌, 입력 또는 검색할 때 체크되는 노드 수를 측정하였다. 그리고 이동 객체 데이타 입력 및 특정 시점 질의 및 시공간 범위 질의 등을 처리할 때의 비용을 기존의 Top-down 방식, Bottom-Up 방식과 비교하여 실험하였다.

6.1 실험 환경

실험에서는 기존 R-트리의 Top-down 방식과 Bottom-Up 방식 그리고, Top-down 버퍼 노드 입력 방식과 Bottom-Up 버퍼 노드 입력 방식에 대해 노드 접근 횟수를 체크하여 입력과 검색 비용을 분석하였다. 또한 특정 시점 질의 및 시공간 범위 질의를 처리할 때, 노드 접근 횟수가 어떻게 달라지는지를 알아보았다.

국가별 경계 및 건물의 위치 등과 같은 공간 객체 들은 인터넷 등에서 관련 실제 데이타를 구할 수 있지만, 자동차나 비행기의 이동 경로 등과 같은 이동 객체 데이타는 공개된 자료를 구하기 어렵다. 따라서 이동 객체와 관련한 응용 분야의 실험에서는 GSTD(Generator of Spatio-Temporal Datasets)[15,16]나 City Simulator [17], Oporto[18] 등과 같은 이동 객체 데이타 생성기를 사용한다. 따라서 이 논문에서도 일반적으로 많이 쓰이는 GSTD와 유사한 데이타 생성기를 사용하였다. 실험에 쓰인 데이타 생성기는 이동하는 차량이 10분 간격으

로 자신의 위치를 보낸다고 가정하고, 10분 사이의 최대 이동거리는 30 Km로, 이동 방향은 16 방위로 설정하여 자유롭게 이동하는 차량의 위치데이타를 생성시켰다.

6.2 실험 및 성능 평가

이 장에서는 제안한 GU-트리와 기존 R-트리를 구현하여, 아래 표의 다양한 실험 평가 항목에 따라 실험하고, 각 과정을 처리하는데 필요한 노드 접근 수를 체크하여 처리 비용을 비교, 평가하였다.

실험은 표 4에서 명시한 것처럼, 차량의 위치 데이타를 10만부터 최대 100만까지 입력시키면서, 그에 따른 색인의 데이타 입력 비용을 비교한다. 또한 특정 시점 질의 및 시공간 범위 질의에서 질의 시점 및 시간과 공간의 범위를 변화시켜가며, 노드 검색 수를 체크한다.

6.3 데이타 입력 질의 평가

데이타 개수에 따른 각 입력 방식의 비용 비교를 위해, Top-down 방식의 R-트리와 GU-트리 그리고, Bottom-Up 방식의 R-트리와 GU-트리에 차량의 위치 데이타를 10만~100만개를 입력시켜, 각 트리의 노드 접근 수를 비교하였다.

각 트리의 입력 비용은 그림 4에서와 같이 입력 데이타 개수가 많을수록 그에 따른 비용도 늘어나는 것을 알 수 있다. 실험에서 Top-down 방식의 R-트리가 가장 많은 비용이 들었고, Bottom-Up 방식의 R-트리가 보다 나은 성능을 보였다. Bottom-Up 방식은 리프 노드로 직접 데이타를 입력하기 때문에, 상위 노드 경계 갱신비용이 Top-down 방식보다 적었던 것으로 생각된다. 그리고 버퍼 노드를 활용한 GU-트리는 Top-down 과 Bottom-Up 방식 모두, 기존의 R-트리 방식보다 좋은 성능을 보였다. 이로 인해 5 장에서 수식으로 나타낸 바와 같이, 라인 세그먼트 하나하나를 고려하는 것보단, 이를 그룹지어 입력하는 버퍼 노드 방식이 뛰어난 것을 알 수 있었다. 그리고 GU-트리(Top-down)가 GU-트리(Bottom-Up)보다 약간 적은 노드 입력 비용을 보였는데, 이는 버퍼 노드를 입력할 때마다 상위 노드 경계가 루트까지 갱신된 것과 해시 테이블을 사용하여 입력

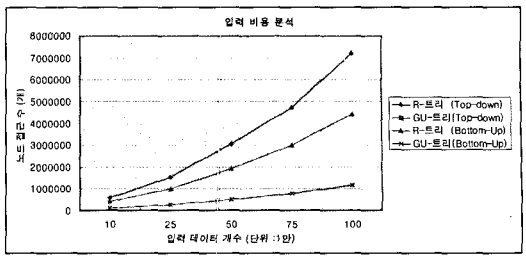

그림 4 데이타 개수에 따른 입력 비용 비교

단계가 늘어났기 때문으로 생각된다. Bottom-Up방식에서 데이타가 입력될 때마다 항상 상위 노드 경계를 갱신하게 되면, Top-down방식과 유사한 입력 비용이 들게 되고, 또한 Bottom-Up 방식은 해시 테이블이라는 단계를 한번 거치기 때문에 Top-down 방식보다 입력 단계가 한 단계 더 많다. 따라서 입력 데이타 개수에 따른 노드 접근 비용은 GU-트리(Top-down) ≒ GU-트리(Bottom-Up) < R-트리(Bottom-Up) < R-트리(Top-down) 이다.

6.4 검색 질의 비용 평가

버퍼 노드 방식을 사용한 GU-트리가 데이타 입력에 효과적인 것은 그림 4에서 알 수 있었다. 그러나 LUR-트리처럼 입력 비용이 좋더라도 오히려 검색 비용이 안 좋은 경우가 있기 때문에, 이 장에서는 표 5의 실험 평가 항목에 따라 특정 시점 질의 및 시공간 범위 질의를 통해, 각 색인의 검색 효율을 알아본다. 이 장에서는 평가 항목에 따른 모든 실험 중 일부분만을 설명하였다.

6.4.1 특정 시점 질의 비교

사용자가 지정하는 어느 한 시점에서 이동 객체의 위치를 검색하는 특정 시점 질의를 처리할 때의 검색 비용을 살펴보기 위해, 질의 시점 및 공간 범위를 변화시켜 실험한다. 아래 그림은 공간 범위를 50%로 고정 시키고, 질의 시점만을 변화시켜 실험한 결과이다. 그림에서 GU-T는 Top-down 방식의 GU-트리, GU-B는 Bottom-Up 방식의 GU-트리를 나타내고, R-트리도 입력방식에 따라 R-T와 R-B로 표현하였다.

표 4 실험 평가 항목

비교 항목	매개 변수	실험 데이타 범위				
데이타 입력 질의	위치 데이타 개수	10 만	25 만	50 만	75 만	100 만
특정 시점 검색 질의	공간 범위	10 %	25 %	50 %	75 %	100 %
	질의 시점	10 %	25 %	50 %	75 %	90 %
	공간 범위, 질의 시점	공간 10 % 시점 10 %	공간 25 % 시점 25 %	공간 50 % 시점 50 %	공간 75 % 시점 75 %	공간 100 % 시점 90 %
시공간 범위 검색 질의	공간 범위	10 %	25 %	50 %	75 %	100 %
	시간 범위	1 %	5 %	10 %	15 %	20 %
	시간, 공간 범위	시간 1 % 공간 10 %	시간 5 % 공간 25 %	시간 10 % 공간 50 %	시간 15 % 공간 75 %	시간 20 % 공간 100 %

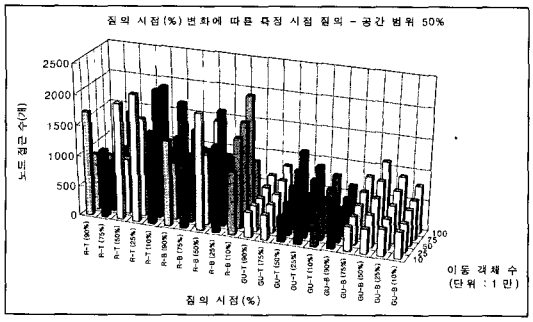

그림 5 질의 시점 변화에 따른 특정 시점 질의 검색 비용

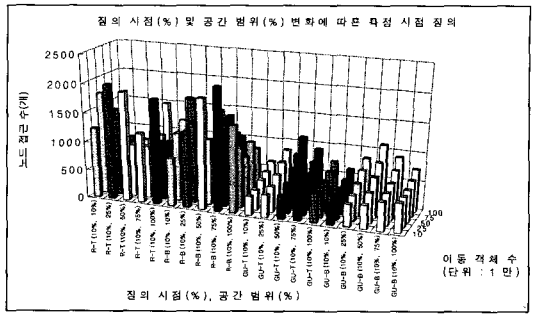

그림 6 질의 시점과 공간 범위 변화에 따른 특정 시점 질의 검색 비용

그림 5는 색인의 전체 시간 축에서 질의하는 시점(%)을 변화시키면서 특정 시점 질의를 처리한 결과이다. R-T와 R-B의 평균이 비슷하고, GU-T와 GU-B도 매우 유사한 검색 결과를 보였다. 그리고 버퍼 노드를 활용한 GU-트리가 R-트리보다 좀 더 좋은 성능을 보였다. 이는 공간 활용도가 60% 정도인 R-트리[9]보다 공간 활용도가 100%인 GU-트리가 보다 적은 노드 수를 갖기 때문으로 보인다. 또한 리프 노드가 아닌 넌 리프 노드 단위로 데이타를 입력받고 분할 연산을 하지 않으므로, 각 데이타를 입력할 때마다 적절한 하위 노드를 고려하는 R-트리보다 입력 과정이 단순해졌다. 그리고 입력 받는 데이타는 시간순서대로 노드에 저장되기 때문에 질의를 처리할 때 시간 축에 대한 검색을 돕는 것으로 생각된다. 그러나 이로 인해 공간 검색에 대한 효율은 나빠질 것이다. 질의 시점 변화에 따른 특정 시점 질의에 대한 노드 접근 수는 GU-T ≒ GU-B < R-T ≒ R-B이다.

특정 시간대에서 공간 범위를 10%, 25%, 50%, 75%, 100%로 변화시키면서 얻는 결과에서는 R-트리와 GU-트리 모두, 공간 범위가 넓어질수록 점차 검색 비용도 높아져갔다. 그리고 공간 범위에 따른 특정 시점 질의에서도 R-트리보단 GU-트리가 좀 더 좋은 성능을 보였다. 이는 GU-트리가 R-트리보다 공간 검색 효율이 좋은 것이 아니라 특정 시점에 대한 시간 검색이 빠르기 때문으로 보인다. 따라서 공간 검색만을 따질 경우, R-트리보다 성능이 좋지 않을 수 있다. 공간 범위 변화에 따른 특정 시점 질의에 대한 노드 접근 수는 GU-T ≒ GU-B < R-T ≒ R-B이다.

그림 6에서는 질의하는 시간대와 공간 범위의 변화에 따라 검색 결과가 어떻게 달라지는지 알 수 있다. R-트리와 GU-트리 둘다, 그림 5의 결과와 어느 정도 유사하다. 이는 시간은 무한하고 공간은 한정되어 있다는 가정 아래, 시간축->공간축의 순서로 검색하기 때문에, 공간 범위를 변화시키는 것이 검색에 많은 영향을 미치지 않는 것으로 보인다. 그리고 버퍼 노드 연산이 노드들을 시간 축에 따라 정렬시키는 효과를 내어, 특정 시점 질의에서의 GU-트리의 성능을 높였다고 생각된다. 질의 시점 및 공간 범위의 변화에 따른 특정 시점 질의에 대한 노드 접근 수는 GU-T ≒ GU-B < R-T ≒ R-B이다.

6.4.2 시공간 범위 질의

시공간 범위 질의는 요청된 질의의 특정 범위 안에 포함된 이동 객체의 위치 및 궤적을 검색하는 질의이다. 색인의 검색 비용을 자세히 알아보기 위해, 시간 범위 및 공간 범위를 변화시켜가며 그 처리 비용을 비교 분석하였다. 먼저 공간 범위를 50%로 고정시키고, 질의하는 시간 범위를 1%~20%까지 변화시켜 실험한 결과는 아래 그림과 같다.

그림 7은 색인의 전체 시간 축에서 질의하는 시간 범위(%)를 변화시켜가며 시공간 범위 질의를 처리한 결과이다. 이 실험에서는 모든 트리가 매우 유사한 검색 결과를 보였다. 질의 처리를 위한 노드 접근 수를 살펴보면, 각각의 검색 비용 중 약 1%~7% 정도로 GU-트리가 R-트리보다 적은 비용이 든 것을 알 수 있었다. 질의 시간 범위 변화에 따른 시공간 범위 질의 처리 비용은 GU-T ≒ GU-B ≤ R-T ≒ R-B이다. 또한 공간 범위에 따른 특정 질의 시점 검색 결과에서도 모든 색인의 검색비용이 매우 유사했다.

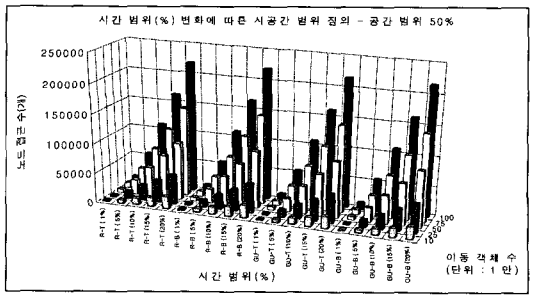

그림 7 시간 범위 변화에 따른 시공간 범위 질의 검색 비용

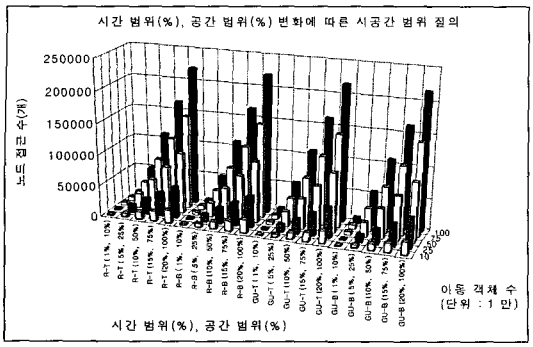

그림 8 시간 범위와 공간 범위 변화에 따른 시공간 범위 질의 검색 비용

질의 시간 범위와 공간 범위 변화에 따른 검색 비용의 차이는 그림 8에서 볼 수 있다. 모든 색인이 유사한 비용으로 질의를 처리했지만, R-B가 약간 더 많은 노드 검색 수를 보였다. 시간 범위 및 공간 범위의 변화에 따른 시공간 범위 질의에 대한 노드 접근 수는 GU-T ≒ GU-B ≒ R-T ≒ R-B이다. 시공간 범위 질의에 대한 검색 비용을 종합해 보면, 공간 범위 및 시간 범위가 커지면 커질수록 색인의 처리 비용은 서로 비슷해지고 많은 차이를 보이지 않는다.

6.5 실험 결과 분석 및 고찰

제안된 버퍼 노드를 활용한 GU-트리는 빈번한 이동 객체의 입력 횟수를 줄이기 위하여, 전송 받는 차량의 데이타를 버퍼 노드로 그룹화시켜 저장하는 이동 객체 색인이다. 제안된 색인의 효율성을 알아보기 위하여, 데이타 수를 증가시켜 색인에 입력한 실험 결과를 정리하면, 5장에서 수식으로 계산된 비용과 유사하게 GU-트리가 기존의 Top-down, Bottom-Up 방식보다 입력 비용이 매우 적은 것을 알 수 있다. 그리고 LUR-트리처럼 색인의 입력 효율이 뛰어나도 검색 성능이 안 좋은 경우가 있기 때문에, 특정 시점 질의 및 시공간 범위 질의에 대해 질의 시점, 공간 범위, 시간 범위를 변화시키면서 질의 처리 비용을 체크하였다. 질의 처리에 대한 평가에서는 대체적으로 제안된 GU-트리가 기존의 R-트리보다 약간 더 좋은 성능을 보였다. 특히 특정 시점 질의에서는 눈에 띄게 검색 비용을 줄였다. 이로 인해 입력 받는 라인 세그먼트를 넌 리프 노드에 모아 한꺼번에 저장하는 버퍼 노드 방식은 색인의 입력 비용을 줄일 뿐만 아니라, 시간 축으로도 약간 정렬된 효과를 내어 특정 시점 질의 및 시간 범위 질의에 도움을 주는 것을 알 수 있다. 따라서 이 버퍼 노드 방식은 시간 축이 상대적으로 길며 데이타의 갱신이 빈번하게 이루어지는 이동 객체 관리 시스템[17] 등에서 유용하게 사용할 수 있을 것이다. 그러나 현재 및 미래 위치를 다루는 색인에서는 신속한 갱신 연산이 필요하기 때문에, 이 버퍼 노드 방식을 그대로 사용하기엔 어려움이 있을 것으로 보이며, 다양한 응용에 대한 연구가 필요하다. 또한 색인의 저장 공간에 대해 고찰해보면 GU-트리는 사용된 버퍼 노드를 그대로 색인에 입력하기 때문에, 기존의 색인에 버퍼 노드 방식을 적용할 경우 추가적으로 많은 저장 공간을 필요로 하지 않는다.

7. 결 론

시간에 따라 끊임없이 전송되는 이동 객체의 위치를 다루는 이동 객체 관리 시스템에서는 질의 처리 요청보다 데이타 입력 요청이 상대적으로 빈번하기 때문에, 효과적인 데이타의 입력 및 갱신 기술이 필수적이다. 신속한 위치 데이타 갱신을 위해, 기존의 이동 객체 색인에서는 크게 Top-down 입력 방식과 Bottom-Up 입력 방식을 사용한다. 그러나 두 방식 모두 데이타를 입력할 때 입력받는 데이타를 직접 색인에 입력하기 때문에, 입력 비용이 전송 받는 데이타 수에 따라 크게 증가하는 단점이 있다.

따라서 이 논문에서는 이와 같은 문제점을 해결하기 위하여 입력받는 데이타를 넌 리프 노드 단위로 모아 한꺼번에 저장하는 버퍼 노드 입력 방식을 제안하였고, 아울러 제안한 버퍼 노드 입력 방식을 이용한 색인의 성능 향상을 위하여 GU-트리를 제안하였다. 제안된 GU-트리는 이동 객체의 과거 및 현재 위치를 다루며, R-트리와의 비교 분석을 통해 입력 비용 및 특정 시점 질의 처리 비용을 효과적으로 줄임을 확인하였다. 앞으로 남은 문제는 데이타가 기하급수적으로 증가하는 시공간 분야의 스트림 응용에서 사용할 수 있도록, 제안한 버퍼 노드 연산과 GU-트리의 개념을 색인의 응용 및 저장 단계별로 확장하여 성능을 극대화시키는 것이다.

참 고 문 헌

[1] J. H. Reed, K. J. Krizman, B. D. Woerner, T. S. Rappaport, "An Overview of the Challenges and Progress in Meeting the E-911 Requirement for Location Service," IEEE Communication Magazine, pp. 33-37, 1998.

[2] M. F. Mokbel, T. M. Ghanem, W. G. Aref, "Spatio-temporal Access Methods," IEEE Data Engineering Bulletin, Vol. 26, No. 2, pp. 40-49, 2003.

[3] R. H. Guting, M. H. Bohlen, M. Erwig, C. S. Jensen, N. A. Lorentzos, M. Schneider, M. Vazirgiannis, "A Foundation for Representing and Querying Moving Objects," ACM Transactions on Database Systems, Vol. 25, No. 1, pp. 1-42, 2000.

[4] A. Guttman, "A.:R-trees: a Dynamic Index

Structure for Spatial Searching," ACM-SIGMOD, pp. 47-57, 1984.
[5] M. L. Lee, W. Hsu, C. S. Jensen, B. Cui, K. L. Teo, "Supporting Frequent Updates in R-Trees: A Bottom-Up Approach," VLDB, pp. 608-619, 2003.
[6] D. S. Kwon, S. J. Lee, S. H. Lee, "Indexing the Current Positions of Moving Objects Using the Lazy Update R-Tree," Mobile Data Management, pp. 113-120, 2002.
[7] L. Forlizzi, R. H. Guting, E. Nardelli, M. Schneider, "A Data Model and Data Structures for Moving Objects Databases," ACM SIGMOD, pp. 319-330, 2000.
[8] D. Pfoser, Y. Theodoridis, C. S. Jensen, "Indexing Trajectories of Moving Point Objects," CHORO-CHRONOS TECHNICAL REPORT CH-99-03, October 1999.
[9] D. Pfoser, C. S. Jensen, Y. Theodoridis, "Novel Approaches in Query Processing for Moving Objects," CHOROCHRONOS TECHNICAL REPORT CH-00-03, 2000.
[10] S. Saltenis, C. Jensen, S. Leutenegger, M. Lopez. "Indexing the Positions of Continuously Moving Objects," ACM-SIGMOD, pp. 331-342, 2000.
[11] Y. J. Jung, E. J. Lee, K. H. Ryu, "MP-tree : An Index Approach for Moving Objects in Mobile Environment," ASGIS, pp. 104-111, 2003.
[12] E. J. Lee, K. H. Ryu, K. W. Nam, "Indexing for Efficient Managing Current and Past Trajectory of Moving Object," Apweb 2004, pp. 781-787, Hangzhou, 2004.
[13] R. Choubey, L. Chen, E. A. Rundensteiner, "GBI: A Generalized R-Tree Bulk-Insertion Strategy," Symposium on Large Spatial Databases, pp. 91-108, 1999.
[14] L. Chen, R. Choubey, and E. A. Rundensteiner, "Bulk Insertions into R-trees using the Small-Tree-Large-Tree Approach," Procedings of ACM GIS Workshop, 1998, pp. 161-162, 1998.
[15] Y. Theodoridis, M. A. Nascimento, "Generating Spatiotemporal Datasets," SIGMOD Record, Vol. 29, No. 3, pp. 39-43, 2000.
[16] D. Pfoser, C. S. Jensen, "Querying the Trajectories of On-Line Mobile Objects," CHOROCHRONOS TECHNICAL REPORT CH-00-57, 2000.
[17] T. Brinkhoff, "Generating Traffic Data," IEEE Data Engineering Bulletin, Vol. 26, No. 2, pp. 19-25, 2003.
[18] J. M. Saglio, J. Moreira, "Oporto: A Realistic Scenario Generator for Moving Objects," DEXA Workshop, pp. 426-432, 1999.
[19] K. H. Ryu and Y. A. Ahn, "Application of Moving Objects and Spatiotemporal Reasoning," Time-Center TR-58. 2001.

정 영 진

2000년 충북대학교 전자계산학과(이학사) 2002년 충북대학교 대학원 전자계산학과(이학석사). 2003년~현재 충북대학교 대학원 전자계산학과 박사과정. 관심분야는 이동 객체 데이타베이스, 이동 객체 색인, Temporal GIS, 유비쿼터스 컴퓨팅 및 질의 처리

류 근 호

1976년 숭실대학교 전자계산학과 졸업 1980년 연세대학교 공학대학원 전자계산학 석사. 1988년 연세대학교 대학원 전자계산학 박사. 1976년~1986년 육군군수지원사전산실(ROTC 장교), 한국전자통신연구소(연구원), 한국방송통신대, 전산학과(조교수) 근무. 1989년~1991년 Univ. of Arizona 연구원(TempIS Project). 1986년~현재 충북대학교 전기전자 및 컴퓨터공학부 교수. 관심분야는 시간 데이타베이스, 시공간 데이타베이스, Temporal GIS, 객체 및 지식베이스 시스템, 지식기반 정보검색시스템, 데이타 마이닝, 데이타베이스 보안 및 Bio-Informatics