http://dx.doi.org/10.3745/KIPSTD.2011.18D.4.225

과거 위치 색인에서 입력/검색 비용 조정을 위한 가변 버퍼 노드 기법 설계

정 영 진[†] · 안 부 영[††] · 이 양 구[†††] · 이 동 규[††††] · 류 근 호[†††††]

요 약

무선 통신 기술의 발달과 컴퓨터의 소형화에 힘입어 사용자의 위치에 따라 맞춤형 서비스를 제공하기 위하여 다양한 위치 기반 서비스 응용들이 개발되고 있다. 그리고 대용량의 차량 위치 데이터를 효과적으로 처리하기 위하여 차량 위치 감지 및 전송, 데이터의 삽입 및 검색과 사용자 질의 처리 기술이 요구된다.

이 논문에서는 대용량의 과거 차량 위치 정보를 빠르게 입력, 검색하는 과거 위치 색인을 설계하고 상황에 따라 입력과 검색 비용을 조절할 수 있는 가변 버퍼 노드인 기법을 제안한다. 설계된 색인은 GIP+와 같이 효과적인 입력을 위해 버퍼 노드를 사용하고 빠른 검색을 위해 프로젝션 스토리지를 사용한다. 그리고 사용자가 지정한 시간 간격에 따라 버퍼 노드에 저장되는 데이터의 개수를 조절하여 입력과 검색 비용을 조절할 수 있다. 실험에서는 버퍼 노드 크기에 따라 비단말 노드 수가 달라지며, 이로 인해 입력과 검색 성능이 달라짐을 확인할 수 있다. 제안된 가변 버퍼 노드 방식은 위치 기반 서비스 응용에 따라 과거 위치 색인의 성능을 조절하는데 효과적으로 사용 가능하다.

키워드 : 위치 기반 서비스, 이동 객체, 과거 위치 색인, 입력/검색 비용 조정, 가변 버퍼 노드

Design of the Flexible Buffer Node Technique to Adjust the Insertion/Search Cost in Historical Index

Young Jin Jung[†] · Bu-Young Ahn[††] · Yang Koo Lee[†††] · Dong Gyu Lee[††††] · Keun Ho Ryu[†††††]

ABSTRACT

Various applications of LBS (Location Based Services) are being developed to provide the customized service depending on user's location with progress of wireless communication technology and miniaturization of personalized device. To effectively process an amount of vehicles' location data, LBS requires the techniques such as vehicle observation, data communication, data insertion and search, and user query processing.

In this paper, we propose the historical location index, GIP-FB (Group Insertion tree with Flexible Buffer Node) and the flexible buffer node technique to adjust the cost of data insertion and search. the designed GIP+ based index employs the buffer node and the projection storage to cut the cost of insertion and search. Besides, it adjusts the cost of insertion and search by changing the number of line segments of the buffer node with user defined time interval. In the experiment, the buffer node size influences the performance of GIP-FB by changing the number of non-leaf node of the index. the proposed flexible buffer node is used to adjust the performance of the historical location index depending on the applications of LBS.

Keywords : Location Based Service, Moving Objects, Historical Location Index, Insertion/Search Cost Adjustment, Flexible Buffer Node

1. 서 론

위치기반서비스(Location Based Services, LBS)는 객체의 변화하는 위치와 지도 등을 다루는 인류의 역사와 함께 탐험, 수송, 전쟁 등 다양한 분야에서 계속되어 왔다[1, 2]. 최근에는 GPS (Global Positioning System) 및 무선 네트워크 기술의 발달로 무선 단말기 사용자나 차량 운전자에게 경로 안내, 광고, 쇼핑, 등 위치에 따라 적절한 컨텐츠를 제공하는

※ 이 논문은 2011년도 정부(교육과학기술부)의 재원으로 한국연구재단의 지원을 받아 수행된 연구임(No.2011-0001044).
† 정 회 원 : 한국과학기술정보연구원 선임연구원
†† 정 회 원 : 한국과학기술정보연구원 책임기술원(교신저자)
††† 준 회 원 : 충북대학교 전자정보대학 Post-doc
†††† 준 회 원 : 충북대학교 전자계산학과 박사과정
††††† 종신회원 : 충북대학교 전자정보대학 소프트웨어전공 교수
논문접수 : 2010년 12월 17일
수 정 일 : 1차 2011년 3월 2일
심사완료 : 2011년 4월 9일

서비스가 발전하는 추세이다. 이러한 서비스를 제공하기 위해서는 시간에 따라 위치가 변화하는 객체 (이동 객체)[3, 4]의 방대한 위치 정보를 빠르게 저장, 갱신, 검색하는 색인 기술과 사용자 질의 처리 기술, 위치 관련 개인화 컨텐츠 검색 기술 등 대용량의 정보를 신속하게 다룰 수 있는 정보 관리 기술들이 필수적이다.

그 중 색인 기술은 B-트리, Quad-트리, R-트리[5]를 기반으로 다양한 색인들이 제시되고 있다. 이동 객체 정보는 시간의 흐름에 따라 끊임없이 생성되기 때문에, 위치 및 궤적을 검색하는 질의 처리 성능 뿐만 아니라 데이터 입력 비용을 줄이기 위해 많은 연구가 계속되고 있다. 위치 기반 서비스의 과거 위치 색인에서는 대량의 차량 데이터를 효과적으로 저장 검색하는 기술이 필요하다. 그리고 현재 및 미래 위치 색인에서는 차량의 현재 위치를 빠르게 저장, 갱신, 검색하는 기술과 이동성을 기반으로 추정된 미래의 위치에 따라 노드의 구성을 수정하여 데이터를 보다 신속하게 다루는 기술이 필요하다.

이 논문에서는 특정 시점 질의와 범위 질의에 초점을 맞춘 과거 차량 위치 색인 구조를 GIP+[1] 기반으로 설계하고 입력과 검색 성능을 효율적으로 조절하기 위한 가변 버퍼 노드 입력 방식을 제안한다. 제시한 방법은 사용자가 버퍼 노드의 시간 간격을 조절함으로써, 입력 비용과 검색 비용을 응용에 따라 적절하게 변경함을 보인다. 제시한 방법을 효과적으로 설명하기 위하여, 먼저 기존 이동 객체 색인들의 입력 방식 및 그 문제점들을 알아보고, 설계된 과거 차량 위치 색인 구조와 가변 버퍼 노드 방식을 소개한다. 그리고 가변 버퍼 노드 방식의 알고리즘을 살펴본 후, 제안된 버퍼 노드 기법과 기존 과거 위치 색인들의 성능 비교를 위해, 차량 위치 데이터 입력, 특정 시점 질의, 시공간 범위 질의에 대한 노드 접근 수를 측정하였다

2. 관련 연구

위치 기반 서비스는 이동하는 객체 (moving object)의 위치 및 상태에 따라 지도와 함께 다양한 서비스를 제공하는 시스템이다[6]. 예를 들어, 운전 중인 차량의 위치 및 상황이 시간의 흐름에 따라 다양하게 변화하기 때문에, 운전자에게 필요한 지도 및 교통 정보, 등은 이동하는 지점 및 시간에 따라 달라진다. 따라서 차량의 위치 정보를 효과적으로 다루는 기술이 필수적이며, 필수 기술 중 하나로 다양한 색인 기법들[7-14]이 제시되었다.

이러한 색인들은 R-트리나 B^+-트리를 기반으로 설계되며, 응용에 따라 과거 이력 정보 및 궤적 정보를 다루는 색인과 현재 및 가까운 미래의 위치를 다루는 색인으로 구분된다[15].

과거 특정 시점의 차량 위치나 및 과거 특정 시간 간격의 차량의 궤적, 등을 빠르게 검색하는 과거 위치 색인으로는 STR-트리(Spatio-Temporal R-tree)[15], TB-트리(Trajectory Bundle Tree)[16], MP-트리(Moving Point tree)[17], GIP[1]. The B_x-tree 구조[11], ST^2B-tree[12], 등이 있다. 과거 위치 색인들은 대용량의 이동 객체 위치 데이터를 효과적으로 다루기 위하여 데이터 삽입 및 검색 성능을 높이는데 초점을 맞춘다. STR-트리[15]는 R-트리를 기반으로 이동 객체의 위치 좌표와 궤적 보호를 중점으로 설계되었다. TB-트리는 색인의 단말 노드들 사이를 링크로 연결하여 이동 객체 궤적에 대한 검색 성능을 높였다. MP-트리는 노드들의 겹치는 영역을 모아 순서대로 저장하는 프로젝션 스토리지를 활용하여, 특정 시점에 대한 성능을 높였으나 입력 비용 및 트리의 크기가 커지는 단점이 있다. TB^*-tree[12], B_x-tree[18]는 이동 객체의 움직임을 벡터로 다루며, B-트리를 활용하여 시간에 따른 검색을 용이하게 하고, Hilbert와 Z-커브[19]를 통해 공간을 분할하여 검색 성능을 높였다.

현재의 이동 객체 위치를 파악하거나 미래의 위치를 빠르게 다루기 위한 현재 및 가까운 미래 색인은 TPR-트리(Time Parameterized R-tree)[20], LUR-트리[21]와 Bottom-Up update 방식[22], TPR^*-트리[23], 등이 있다. 현재 및 미래 위치 색인들은 주로 이동 객체의 현재 및 미래 위치를 효과적으로 갱신하고 검색하는 방법을 제시한다. TPR-트리는 현재 및 미래 위치를 다루는 색인으로 이동 객체의 움직임에 따라 선형함수를 활용하여 노드를 구성함으로써 빈번한 갱신을 줄였다. LUR-트리와 Bottom-Up 갱신은 노드를 갱신할 때 필요한 검색 비용(루트로부터 단말 노드를 찾는 비용)을 줄이기 위해서, 해시 테이블을 활용하여 데이터를 색인의 단말 노드에 직접 입력했다.

많은 량의 데이터를 효과적으로 입력하기 위한 버퍼 노드 입력 방식은 STLT (Small Tree and Large Tree)[24]를 기반으로 설계되었다. STLT는 공간 데이터를 클러스터링해서 작은 트리(small tree)를 만들고, 그 작은 트리를 다시 큰 트리(large tree)로 넣는 과정을 통해, 입력 비용을 효과적으로 감소시켰다. 그러나 데이터를 클러스터링하는 비용과 만들어진 작은 트리가 색인에 입력될 때, 작은 트리가 이미 넓은 범위를 갖고 있기 때문에, 노드 사이의 오버랩을 증가시킬 수 있다. 이로 인해, 검색 질의에 따라 R-트리보다 검색 성능이 떨어지기도 한다. 이 방식은 효율적인 데이터 입력을 위하여 다양한 색인에서 활용되어졌다. 이 외에도 PMR-쿼드 트리의 대용량 데이터 삽입(the bulk loading of the PMR quad tree)[25], BIOR(Bulk Insertion in Oracle R-trees)[26], 등의 방식이 제시되었다.

이런 이동 객체 색인 기법들은 시공간 질의를 처리하는데 매우 효과적이다. 그러나 응용에 따라 데이터 검색 질의 처리 뿐만 아니라 차량의 위치 데이터 입력에 대한 요청도 매우 빈번하게 일어나기 때문에, 상황에 따라 색인에서 데이터의 입력과 검색 비용의 균형을 적절히 유지하는 것이 필요하다. 이 문제점을 해결하기 위하여 이 논문에서는 색인의 입력과 검색 비용을 조정하는 가변 버퍼 노드 방식을 제안한다.

3. 버퍼 노드를 활용한 GIP-FB

이 논문에서는 차량의 위치 데이터가 서버로 계속해서 전송된다고 가정하고, 그 과거 위치 데이터를 효과적으로 저장, 검색하는데 초점을 맞춘다. 데이터를 검색할 때는 특정 시점 질의와 범위 질의를 다룬다.

3.1 GIP-FB의 구조

가변 버퍼 노드는 버퍼 노드의 크기에 따라 색인의 입력과 검색 비용을 조절하기 위하여 제시되었으며, 이를 과거 위치 색인 GIP*[1]에 활용하여 가변 버퍼 노드를 활용한 그룹 삽입 색인 (GIP-FB, Group Insertion Tree with Flexible Buffer Node)을 아래 그림과 같이 설계하였다.

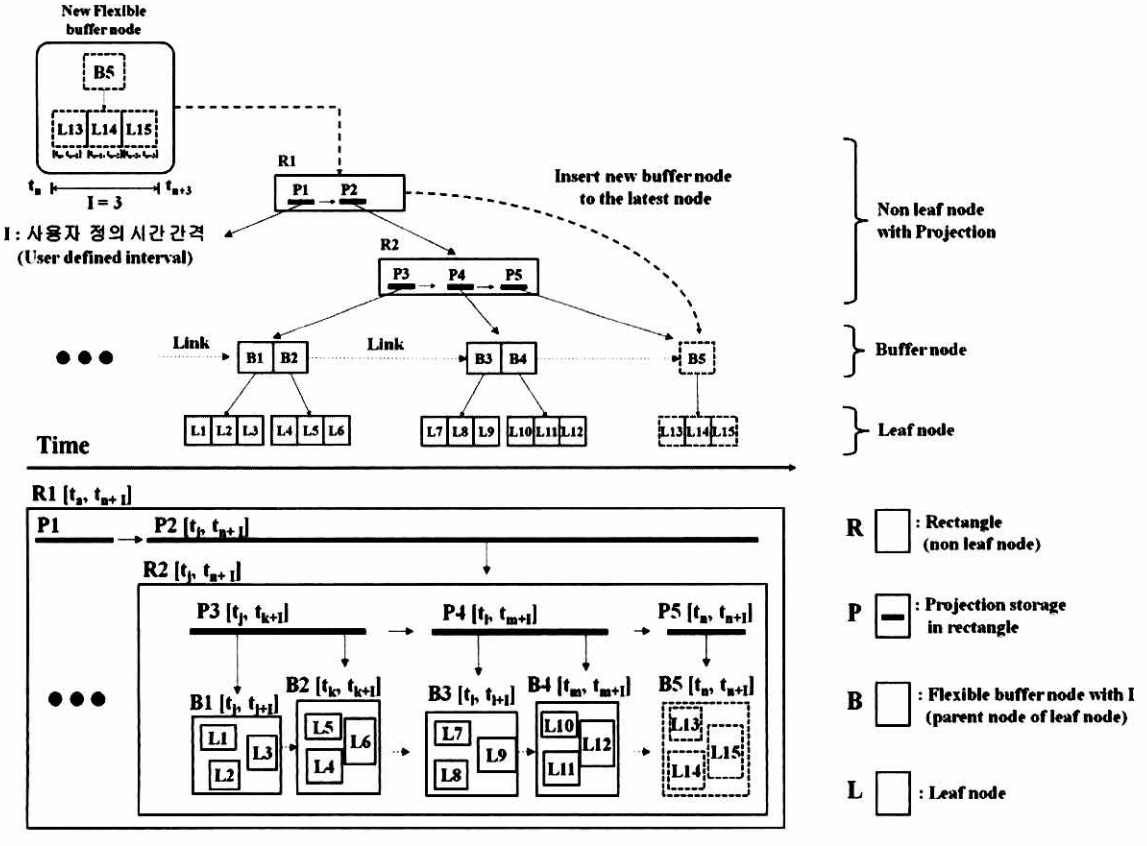
(그림 1)은 차수 (fan-out)가 3인 GIP-FB의 구조를 나타낸다. 노드의 구성은 R (비단말 노드), P (프로젝션 스토리지), B (가변 버퍼 노드), L (단말 노드)로 이루어져 있으며, 입력 비용을 줄이기 위하여, 버퍼 노드 입력 방식[1]에 기반한 가변 버퍼 노드를 설계하고, 검색의 효율을 높이고 색인의 크기를 줄이기 위하여 비단말 노드에만 프로젝션 스토리지[1, 17]를 사용한다.

항상 최신의 차량 위치 정보를 갖고 있는 버퍼 노드는 색인의 시간 축에서 가장 앞부분에 저장되며, 이때, 버퍼 노드는 사용자가 지정하는 시간 간격(I)에 따라 버퍼 노드의 시간 간격$[t_n, t_{n+I}]$에 포함되는 단말 노드만을 저장한다. 이 가변 버퍼 노드 저장 방식의 시간 간격 (I)에 따라 입력과 검색 성능이 순차적으로 변하고, 사용자는 그 중 응용에 따라 적절한 간격 (I)을 선택하여 입력 및 검색의 효율을 높일 수 있다.

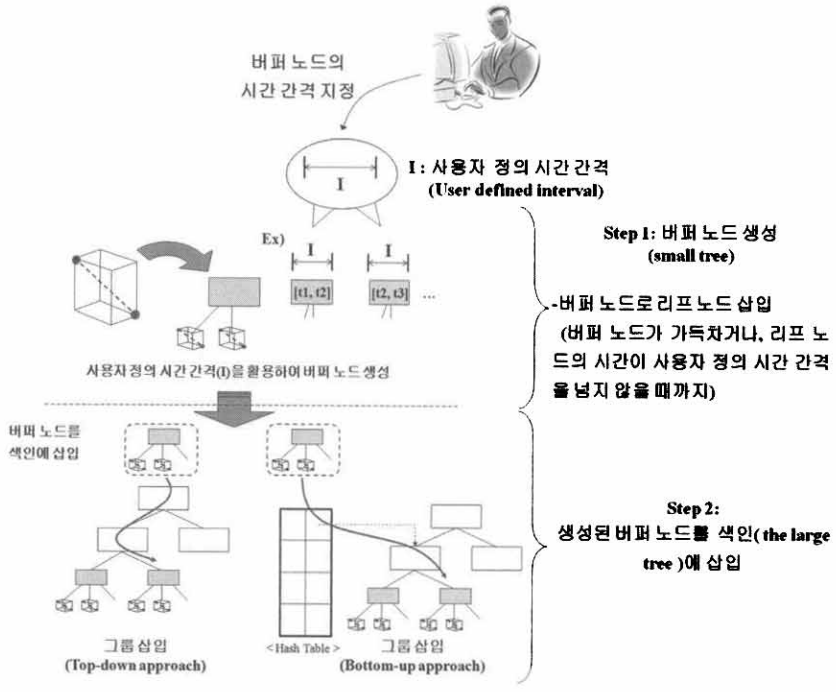
3.2 가변 버퍼 노드 입력 방식

기존의 이동 객체 색인에서는 일반적으로 위치 데이터에 대한 라인 세그먼트를 색인에 직접 입력하는 OBO (one by one) 방식을 사용하기 때문에, 데이터 입력 및 갱신을 위한 많은 비용이 필요하다. 이를 해결하기 위해 STLT[24]를 기반으로 버퍼 노드를 활용한 응용들[1]이 제시되고 입력 및 검색 효율을 높였다. 그러나 버퍼 노드 크기 설정에 대한 실험 평가가 드물기 때문에, 입력과 검색 비용을 적절히 고려하여 색인을 설계하고 활용하기 어렵다. 이 논문에서는 이를 해결하기 위하여 사용자 지정 시간 간격에 따라 버퍼 노드의 크기를 변경하는 기반 가변 버퍼 노드 방식을 설계하고 평가한다.

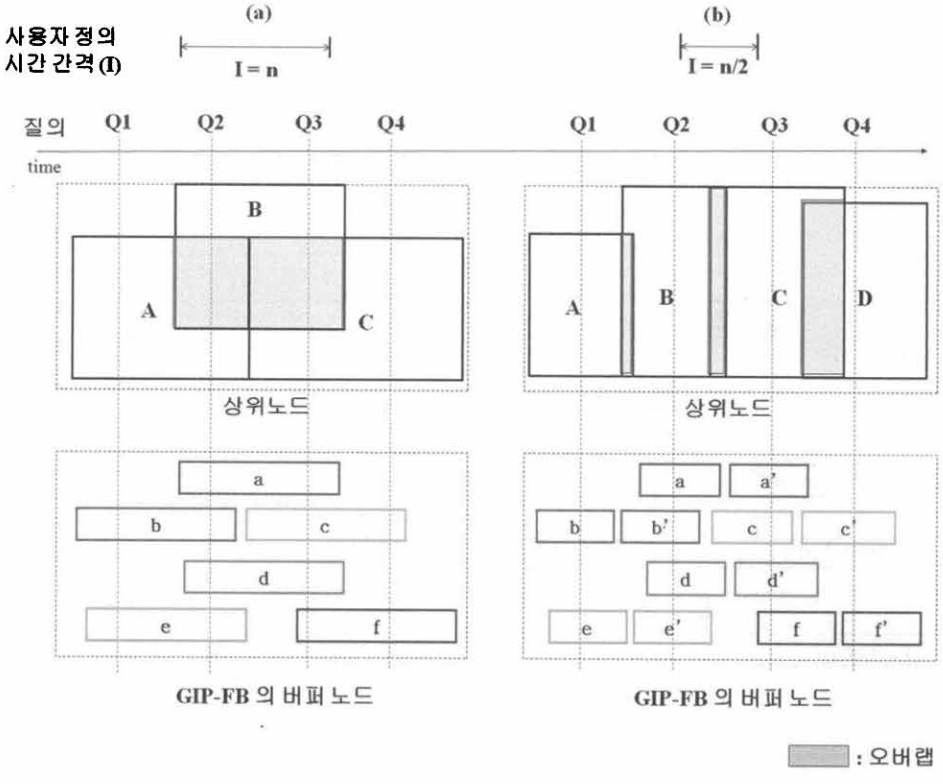
(그림 2)는 입력 받는 데이터를 사용자가 지정한 시간 간격 (I)에 따라 그룹화하여 색인에 입력하는 가변 버퍼 노드 방식을 보여준다. 트리에 입력하는 버퍼 노드 개수는 전체 차량 데이터를 버퍼 노드의 차수로 나눈 만큼 줄어들게 되지만, 그만큼 검색 성능이 떨어진다. 이를 해결하기 위하여, GIP*[1] 에선 특정 시점 질의의 효율을 높이는 프로젝션 스토리지를 비단말 노드에만 배치시켜 색인의 크기를 줄이고, 버퍼 노드의 크기를 최소한으로 줄여, 특정 시점 및 범위 질의의 검색 효율을 높였다. 그러나 버퍼 노드의 크기를 조절하는 옵션이 없어, 사용자가 원하는 입력과 검색의 성능을 응용에 따라 조절하는데 어려움이 있다.



(그림 1) 가변 버퍼 노드를 활용한 GIP-FB의 구조



(그림 2) 사용자 정의 시간 간격을 활용한 버퍼 노드 입력 방식



	Searched nodes
Q1	A→b→e
Q2	B→a→d→A→b→e
Q3	B→a→d→C→c→f
Q4	C→c→f

	Searched nodes
Q1	A→b→e
Q2	B→a→b'→d→e
Q3	C→a'→c'→d'→f
Q4	D→c'→f'

(그림 3) 가변 버퍼 노드의 크기 변화에 따른 노드 검색 예

(그림 3)은 가변 버퍼 노드 크기 변화에 따른 노드의 분포와 특정 시점 질의(Q1 ~ Q4) 처리를 위해 검색하는 노드들을 보여준다. (그림 3(a))에서 사용자 지정 시간간격이 길어 버퍼 노드가 커질수록, 오버랩이 커지고 질의를 처리하기 위해 검색하는 노드수가 증가함을 알 수 있다. 게다가 버퍼 노드가 클수록 저장된 하위 단말 노드가 많기 때문에, 단말 노드를 검색하기 위한 검색 비용까지 증가한다. 그러나 버퍼 노드가 커지면 색인에 입력되는 버퍼 노드 수가 적어지기 때문에, 검색 비용과는 반대로 입력비용은 점점 감소한다. 이를 <표 2>의 입력 비용 비교 예에서 확인할 수 있다. (b)에서는 사용자 지정 시간간격이 1/2로 줄어들고 버퍼 노드 크기가 작아져서 노드간의 오버랩이 줄어들고 노드 접근 수(검색 비용)가 감소함을 알 수 있다. 물론 입력 비용은 검색비용과 반대로 늘어난 버퍼 노드 수만큼 증가한다.

4. 가변 버퍼 노드 알고리즘 및 비용 분석

가변 버퍼 노드 입력 방식은 사용자가 임의대로 버퍼 노드의 크기를 조절하여 과거 위치 색인의 입력/검색 비용을 응용에 따라 조절할 수 있게 해준다. 버퍼 노드의 크기가 커질수록 입력 효율은 좋아지지만, 검색 성능은 감소되기 때문에, 응용에 따라 적당한 크기를 유지하는 것이 중요하다. 이 과정을 아래 알고리즘과 비용 분석 예에서 살펴본다.

4.1 가변 버퍼 삽입 알고리즘

버퍼 노드라는 임시 노드에 사용자가 정한 시간 간격에 따라 데이터를 넣고, 생성된 버퍼 노드를 GIP-FB의 가장 최근 시간대에 삽입한다. GIP-FB에서 노드가 가득차면, 분할하지 않고 새로운 노드를 생성한다. GIP-FB의 노드들은 시간의 흐름에 따라 노드가 죽 늘어선 모습을 하고 있다.

색인에서 시간 간격을 고려하지 않는 풀 버퍼 노드를 활용할 경우, 입력 비용이 OBO 방식보다 12/50 만큼 줄어드는 것을 알 수 있다[1]. GIP-FB의 검색은 GIP+와 같이 프로젝션 스토리지와 버퍼 노드 사이의 링크를 활용하여 특정 시점 및 범위 질의의 성능을 높였다. 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 데이터는 시간에 따라 연속적으로 들어오기 때문에, 이동 객체의 과거 및 현재의 위치를 다루는 색인은 시간 축으로 점점 자라나는 형태를 가진다.

4.2 가변 버퍼 노드 입력 비용 분석 예

버퍼 노드 방식은 단말 노드들을 모아 한꺼번에 저장하는 방식으로, N 개의 레코드를 가진 색인에 K 개의 데이터를 입력한다고 가정할 때, 아래와 같이 입력 비용이 계산된다[27].

```
Algorithm  Insert_buffer_node(class node *root, class
node *entry, int interval)
input :  root     // node of a tree
         entry    // information of moving objects
         interval // user defined interval
method :
    if buffer node is full or entry 's time is beyond the
    defined interval then
        insert_node(root, the buffer node)              //
        insert the buffer node into GIP-FB-tree
        make new buffer node
    endif
    insert the entry into the buffer node
end
```

[Algorithm 1] 가변 버퍼 노드에 데이터 삽입

```
Algorithm  Insert_node(class node *root, class node *entry)
input : root  // node of a tree
        entry // the buffer node
method :
    if root is full then
        // Put the new entry into new node and keep the
        old one as it is
        make new root for inserting entry and adjust tree
    endif
    find the latest T projection in root
    if entry's boundary does not intersect the T projection's
    boundary then
        make new T projection storage
    endif

     insert entry into the T projection
     find the latest node in the T projection
     if entry's level is the child node level of the latest node
     then
        insert entry into the latest node
        make a link between last buffer node and new
        inserted entry      // for range queries
     else
        insert_node(the closest node, entry)
     endif
end
```

[Algorithm 2] GIP-FB에 가변 버퍼 노드 삽입

〈표 1〉 Top-down 방식과 버퍼 노드 방식의 입력 비용 계산 식

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

〈표 2〉 가변 버퍼 노드 방식의 입력 비용 비교 예

입력 내용	비용 계산
가 정	K = 100,000, m = 10, M = 20, N = 1,000,000 $\lvert log_m N \rvert - 1 = \lvert log_{10} 1000000 \rvert - 1 = \lvert l_{10} 10^6 \rvert - 1 = 5$
Top down 방식	입력 비용 = K × ($\lvert log_m N \rvert - 1$) = 100,000 × 5 = 500,000
버퍼 노드 방식	입력 비용 = K + $\frac{K}{M}$ × ($\lvert log_m N \rvert$ - 2) = 100,000 + (100,000/20) × (6-2) = 120,000
버퍼 노드 크기 확장에 따른 버퍼 노드 수 감소 (버퍼 노드 수 / 2)	입력 비용 = K + $\frac{K}{2M}$ × ($\lvert log_m N \rvert$ - 2) = 100,000 + (100,000/20/2) × (6-2) ≒ 110,000
버퍼 노드 크기 축소에 따른 버퍼 노드 수 증가 (버퍼 노드 수 * 2)	입력 비용 = K + $\frac{2K}{M}$ × ($\lvert log_m N \rvert$ - 2) = 100,000 + (2*100,000/20) × (6-2) = 140,000

이 논문에서 제안한 가변 버퍼 노드 방식을 사용할 경우, 사용자 지정 시간 간격과 비례하여 버퍼 노드 크기가 확장되지만, 확장된 버퍼 노드에 보다 많은 단말 노드가 저장되는 것 만큼 버퍼 노드 수는 오히려 적어진다. 이처럼 사용자 지정 시간 간격에 따라 버퍼 노드 수가 변화할 때를 가정하여 아래 표에서 입력 비용의 변화를 살펴본다.

<표 2>는 가변 버퍼 노드 입력 방식에 따른 입력 비용 변화를 예를 들어 설명한다. 사용자 지정 간격이 2 배로 증가될 때 버퍼 노드 크기도 2배로 증가되고 이에 따라 입력 받는 버퍼 노드 수는 1/2로 감소한다고 가정할 때, 이에 따른 입력 비용이 감소됨을 알 수 있다. 또한, 사용자 지정 간격이 절반으로 감소했을 때, 버퍼 노드 크기가 1/2로 감소되고 입력되는 버퍼 노드 수가 2배로 증가한다고 가정하면, 입력 비용도 증가되는 버퍼 노드 수에 비례하여 증가되는 것을 알 수 있다. 위의 예처럼 버퍼 노드의 크기 조절을 통해 입력 비용이 변화하는 것을 알 수 있다.

그리고 버퍼 노드 크기가 확대되어 버퍼 노드 수가 감소될 경우, 색인 안에서 버퍼 노드들이 서로 오버랩(overlap)될 가능성이 높아지고, (그림 7)의 실험처럼 검색 비용이 증가함을 알 수 있다. 이와 반대로 버퍼 노드의 크기가 축소되고, 버퍼 노드의 수가 증가할 경우, 입력 비용은 증가하지만, 버퍼 노드의 오버랩 가능성이 적어지고 검색 비용이 감소된다. 검색 비용은 색인내의 노드 분포, 검색 시점 및 범위에 따라 다양하게 달라지기 때문에 표 2의 입력 비용처럼 수식을 통한 분석을 하지 않지만, 5장의 실험 및 평가를 통해 결과를 확인할 수 있다.

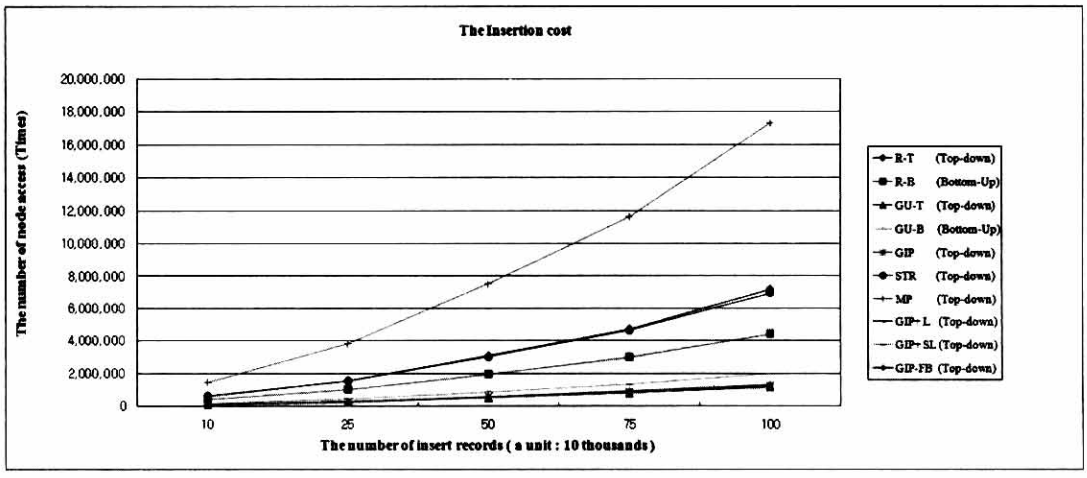
5. 실험 및 평가

이 장에서는 제안한 GIP-FB 의 성능을 비교하기 위하여 기존의 R-T (R-tree, Top-down), R-B (R-tree, Bottom-Up), GU-T (Gu-tree, Top-down), GU-B (Gu-tree, Bottom-Up), GIP (Top-down), STR (Top-down), MP (Top-down), GIP$^+$ L (Top-down), GIP$^+$ SL (Top-down)와 데이터 입력 질의, 특정 시점 질의, 시공간 범위 질의를 처리하는 비용을 실험 평가하였다. 실험에 사용된 GIP-FB는 GIP$^+$ L (풀 버퍼 노드 사용)과 GIP$^+$ SL (최소 시간 간격 버퍼 노드 사용) 사이의 버퍼노드 크기를 선택하기 위해 버퍼 노드 시간 간격을 2 (20분)로 잡았다. 색인의 질의 처리 시간은 실험하는 시스템의 성능, 등에 따라 달라질 수 있으므로 이 논문에서는 질의를 처리할 때 필요한 노드 접근 수를 체크하였다.

실험에 사용된 데이터는 GSTD와 유사하게 100개의 차량 데이터를 매 10분마다 랜덤하게 생성하는 데이터 생성기를 사용하였으며, 각 차량의 위치는 2차원[40,000 x 40,000]의 공간에서 [0 ~ 30]의 범위를 랜덤하게 움직인다. 실험에서는 색인의 입력 비용 비교를 위해, 차량의 위치 데이터를 [100,000 ~ 1,000,000]까지 입력시키고, 특정 시점 질의 및 범위 질의에서 공간 범위(%), 시점(%), 시간 범위(%)에 따라 질의 비용을 체크하였다. 예를 들어, 시공간 범위 질의에서 시간 범위가 25%, 공간 범위가 10%라면, 색인의 전체 시간 범위 중 중간 25% 범위를 지정하고, 10%씩 분할된 공간 범위마다 검색 질의를 수행하고, 그에 대한 평균을 기술하였다.

〈표 3〉 실험 평가 항목

비교 항목	매개 변수	실험 데이터 범위
데이터 입력 질의	위치 데이터 개수	100,000, 250,000, 500,000, 750,000, 1,000,000
특정 시점 검색 질의	공간 범위 (%)	1, 3, 5, 7, 10, 25, 50, 75, 100
	질의 시점 (%)	10, 25, 50, 75, 90
시공간 범위 검색 질의	공간 범위 (%)	1, 3, 5, 7, 10, 25, 50, 75, 100
	시간 범위 (%)	1, 5, 10, 25, 50, 75, 90



(그림 4) 데이터 개수에 따른 입력 비용 비교

5.1 데이터 입력 비용 분석

색인의 입력 비용 분석을 위해, 입력 데이터의 개수를 변화하면서, 데이터를 입력하기 위한 노드 접근 수를 비교한다.

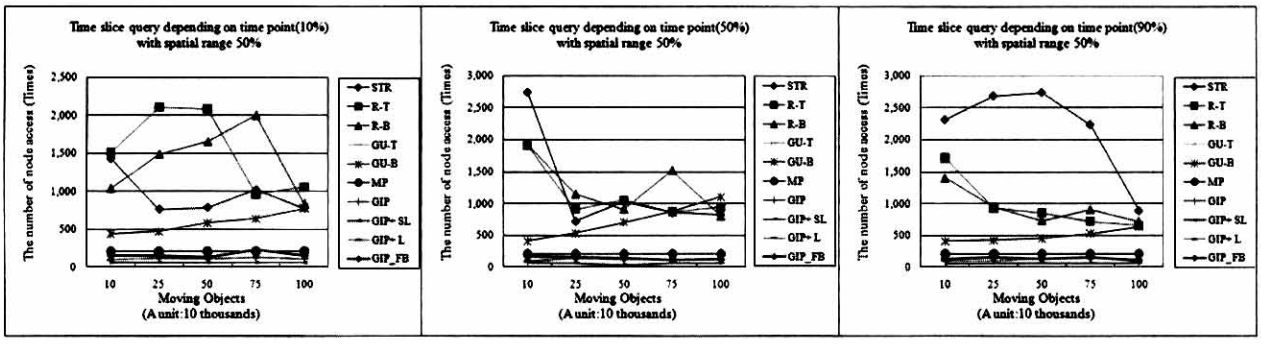
(그림 4)는 실험에서 데이터 입력 비용을 비교한 결과를 보여준다. 이중, MP-트리가 매우 높은 입력 비용을 보였는데, 이는 검색 비용을 낮추는 프로젝션 스토리지가 데이터 입력시엔 추가적인 노드 접근을 필요로 하기 때문이다. 그리고, GU-T와 GU-B[27]가 기본적인 버퍼 노드 방식을 사용하여 가장 낮은 입력 비용을 기록했다. 버퍼 노드와 프로젝션 스토리지를 적절하게 배합한 GIP, GIP$^+$ L, GIP$^+$ SL의 성능은 GU-트리보다 약간 못 미쳤지만, R-T, R-B보단 좋은 성능을 보였다. 가변 버퍼 노드를 활용한 GIP-FB는 GIP$^+$ L와 GIP$^+$ SL 사이의 성능을 보였으며, 60% 이상의 결과가 GIP$^+$ L에 가까운 입력 비용을 보였다. 입력 데이터 개수에 따른 노드 접근 비용은 GU-T ≒ GU-B <≒ GIP ≒ GIP$^+$ L <≒ **GIP-FB** < GIP$^+$ SL < R-B < STR <≒ R-T < MP 이다.

5.2 검색 질의 비용 분석

대부분의 색인들이 입력 비용과 검색 비용 사이에 상반관계(trade-off)가 있으며, 응용에 따라 적절하게 이 관계를 조절하여 효율을 높인다. (그림 5)는 특정 시점 질의를 질의 시점과 공간 범위를 변화시켜가면서 비용을 분석한 결과이다. 그래프의 STR, R-T, R-B에서 노드 접근 수의 변화가 심한데 이는 이 색인들이 공간 좌표를 고려하여 노드를 분할하기 때문에, 랜덤한 차량 위치 데이터를 분할 할 때, 노드의 분포가 고르지 않은 것으로 보인다.

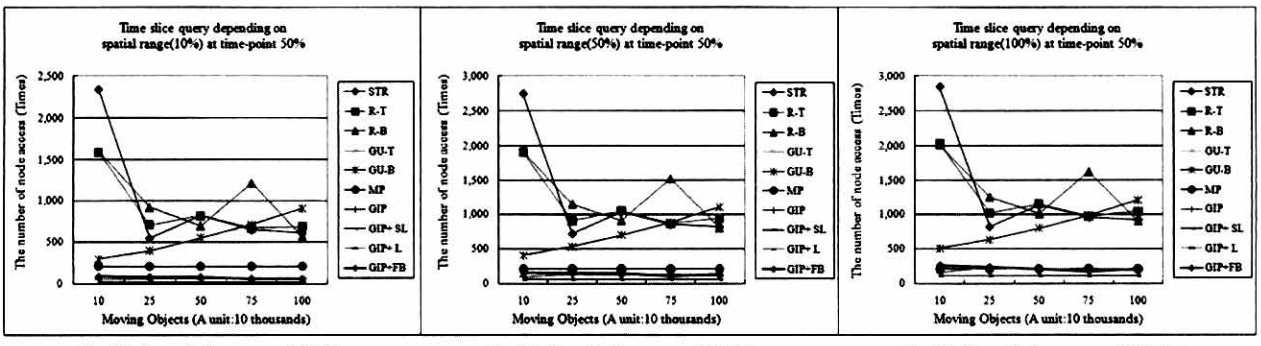
GU 트리[27], GIP 트리들은 대부분 노드를 분할하지 않고 시간에 따라 새로운 노드를 생성하기 때문에, 대부분의 노드들이 시간의 순서에 따라 늘어선 형태로 비교적 균등하게 노드가 분포된다. 그리고, GIP는 하위 노드들을 정렬하여 저장하는 프로젝션 스토리지를 활용하여, 특점 시점 질의를 보다 적은 비용으로 처리할 수 있다. 제시된 GIP-FB는 GIP보다 약간 뒤처지는 성능을 보여준다.

(a) 질의 시점 변화에 따른 특정 시점 질의 검색 비용은 GIP$^+$ SL <≒ GIP$^+$ L≒ GIP < **GIP-FB** < MP < GU-T ≒ GU-B < R-T ≒ R-B <≒ STR 이고, (b) 공간 범위 변화에 따른 특정 시점 질의 검색 비용은 GIP$^+$ SL <≒ GIP$^+$ L ≒ GIP <≒ **GIP-FB** < MP < GU-T ≒ GU-B < R-T ≒ R-B <≒ STR, (c) 질의 시점과 공간 범위 변화에 따른 특정 시점 질의 검색 비용은 GIP$^+$ SL <≒ GIP$^+$ L ≒ GIP <≒ **GIP-FB** < MP < GU-T ≒ GU-B < R-T ≒ R-B <≒ STR이다. GIP-FB는 GIP$^+$SL 과 GIP$^+$L 사이의



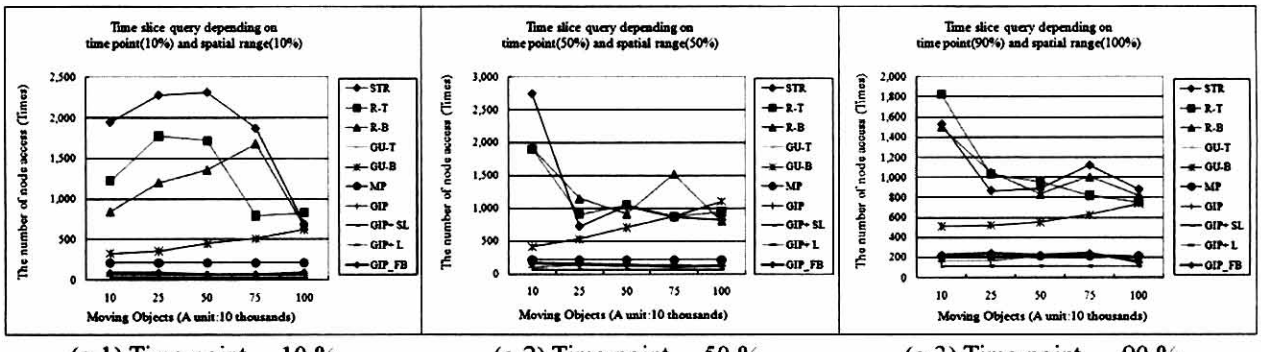
(a-1) Time point 10 % (a-2) Time point 50 % (a-3) Time point 90 %

(a) Time slice query depending on time point (%) with spatial range 50 %



(b-1) Spatial range 10 % (b-2) Spatial range 50 % (b-3) Spatial range 100 %

(b) Time slice query depending on spatial range (%) at time point 50 %



(c-1) Time point 10 % Spatial range 10 % (c-2) Time point 50 % Spatial range 50 % (c-3) Time point 90 % Spatial range 100 %

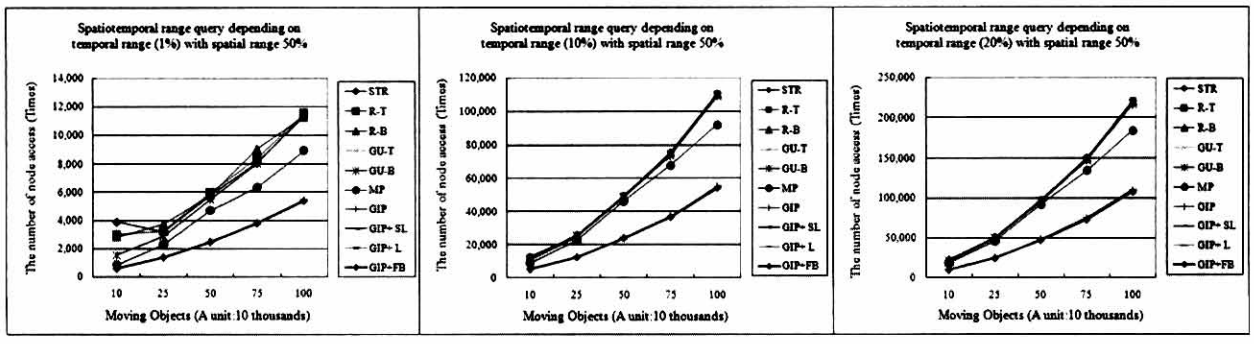
(c) Time slice query depending on spatial range (%) and time point (%)

(그림 5) 질의 시점과 공간 범위 변화에 따른 특정 시점 질의 검색 비용

성능을 낼 것으로 기대되었으나, 사용자 시간 간격 2 (20분)에 따라 노드의 오버랩을 좀 더 허용한 것으로 보인다.

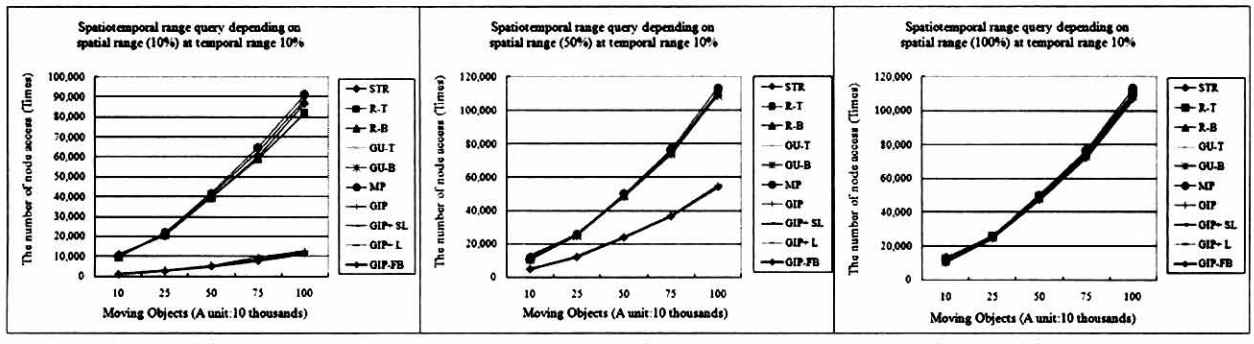
시공간 범위 질의는 요청된 특정 범위 안에 포함된 차량의 위치 및 궤적을 검색하는 질의이다. (그림 6)은 시공간 범위 질의에서 시간 간격과 공간 범위를 변화시켜가면서 비용을 분석한 결과이다. 모든 그래프에서 데이터 개수에 따라 검색 비용도 증가 했으나 GIP 계열은 그 증가 비용을 매우 단축시켰다. 이는 프로젝션 스토리지를 통해 특정 시점 질의를 빠르게 수행하고 버퍼 노드 사이의 링크를 통해 질의 범위를 검색했기 때문이다. (b)에서 볼 수 있듯이 GIP 계열은 공간 범위에 따라 그 비용이 크게 증가한다. 시간축에 초점을 맞추었기 때문에 상대적으로 공간 축에 대한 검색 비용이 쉽게 늘어나는 것을 알 수 있다.

(a) 질의 시간 간격 변화에 따른 시공간 범위 질의 검색 비용은 GIP* L ≒ GIP-FB <≒ GIP* SL<≒ GIP <≒ MP < GU-T ≒ GU-B <≒ R-T <≒ R-B <≒ STR 이며, GIP* L과 GIP-FB는 매우 유사한 검색 성능을 보였다. GIP-FB는 시간 간격이 짧을수록 좋은 성능을 보였고, 시간 간격이 길수록 GIP* L의 성능이 좋았다. (b) 공간 범위 변화에 따른 시공간 범위 질의 검색 비용은 GIP-FB <≒ GIP* L < GIP* SL<≒ GIP <≒ MP < GU-T ≒ GU-B <≒ R-T <≒ R-B <≒ STR 이다. 공간 범위의 변화에 따



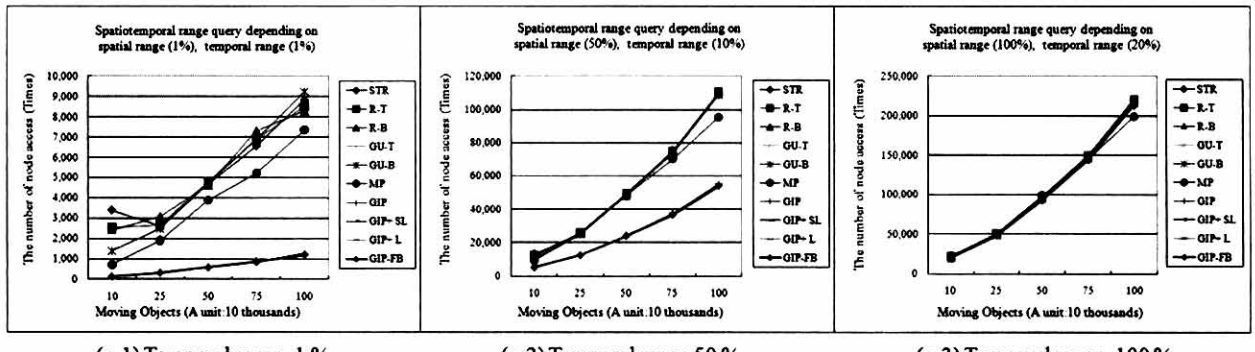
(a-1) Temporal range 1 % (a-2) Temporal range 50 % (a-3) Temporal range 100 %

(a) Spatiotemporal range query depending on temporal range (%) with spatial range 1 %



(b-1) Spatial range 1 % (b-2) Spatial range 5 % (b-3) Spatial range 7 %

(b) Spatiotemporal range query depending on spatial range (%) at temporal range 100 %



(c-1) Temporal range 1 % Spatial range 1 % (c-2) Temporal range 50 % Spatial range 5 % (c-3) Temporal range 100 % Spatial range 7 %

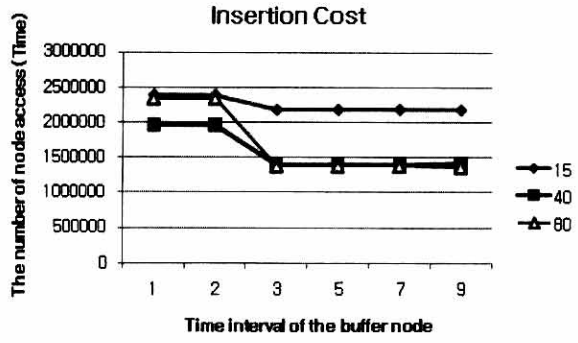
(c) Spatiotemporal range query depending on spatial range (%) and temporal range (%)

(그림 6) 질의 시간 간격과 공간 범위 변화에 따른 시공간 범위 질의 검색 비용

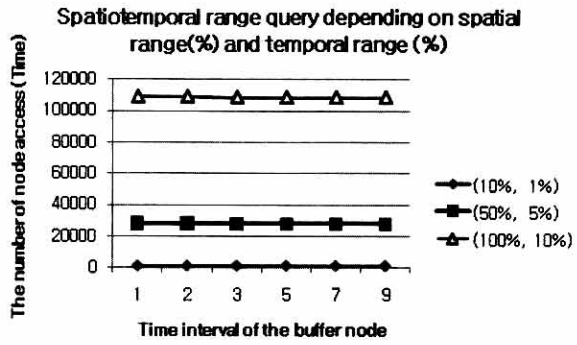
른 범위 질의에선 GIP-FB가 대부분 좋은 성능을 보였으며, GIP* SL가 GIP 보다 약간 더 좋은 검색 성능을 보였다. GU-트리, GIP, 등 버퍼 노드를 활용한 색인들은 시간 범위 질의를 얼마나 빨리 처리할 수 있는지에 중점을 두었기 때문에, 크게 차이가 나진 않지만, 노드의 공간 활용에 되도록 낭비가 없는 색인 구조가 조금씩 좋은 성능을 보인 것으로 보인다. (c) 질의 시간 간격과 공간 범위 변화에 따른 시공간 범위 질의 검색 비용은 GIP-FB <≒ GIP* L < GIP* SL<≒ GIP <≒ MP < GU-T ≒ GU-B <≒ R-T <≒ R-B <≒ STR 이다. 질의 시간 간격이 20% 이고 공간 범위가 100% 일 때, MP-트리가 가장 좋은 성능을 보이지만, 대부분의 경우에 GIP-FB가 좋은 성능을 보이고 있다.

위 실험과 같이 버퍼 노드의 크기를 변경하는 경우도 질의에 따라 고정된 버퍼 노드보다 좋은 성능을 보일 수 있기 때문에, 사용자가 원하는 데로 버퍼 노드의 크기를 변환시켜 입력 비용과 검색 비용의 적절한 균형을 맞추는 방식이 응용에 따라 유용하게 쓰일 것으로 기대된다.

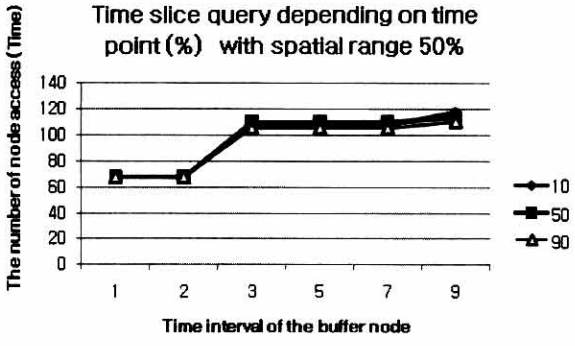
버퍼 노드 크기에 따라서 GIP-FB의 데이터 입력 및 검색 비용의 변화를 살펴보기 위하여, 색인에 1,000,000개의 데이터를 입력하고 버퍼 노드의 시간 간격을 변화시켜 질의 처리 비용을 체크하였다. (a) 입력 비용에서는 노드의 차수가 15, 40, 80 일 때, 입력 비용이 어떻게 변화하는지 체크하



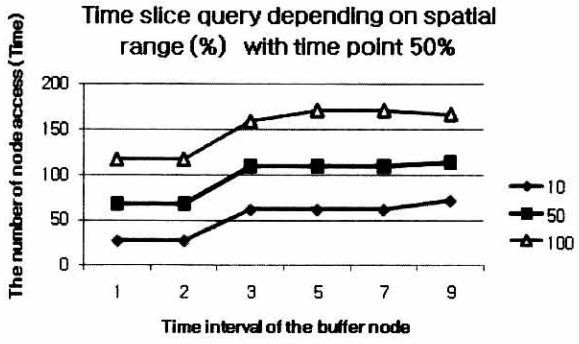
(a) Insertion cost



(b) Spatiotemporal range query with spatial range 1%



(c-1) Time slice query with spatial range 50%



(c-2) Time slice query with time point 50%

(그림 7) 버퍼 노드 크기 변화에 따른 입력, 검색 비용

였다. 시간 간격이 커질수록 입력 비용이 감소되었고, 특히 시간 간격이 2에서 3으로 변경될 때, 많이 감소하였다. 대체로 노드의 차수가 커질 때, 더 비용이 감소하는 경향을 보이는데 이는 노드에 더 많은 데이터가 들어가기 때문에 상대적으로 입력 비용이 더 감소하는 것으로 보인다. 이렇게 입력 비용이 감소하는 이유는 버퍼 노드의 크기가 커지면서, 색인의 비단말 노드의 수가 줄어들기 때문이다. 하지만, 이로 인해 (b), (c-1), (c-2)의 경우와 같이 검색 비용이 커지는 것을 확인할 수 있다. 시공간 범위 질의(b)의 경우는 비용이 크게 변화하지 않는데, 이는 버퍼 노드간의 링크를 사용하는 GIP+ 구조로 인해, 한번 질의를 만족하는 단말 노드를 찾은 후에는 링크를 따라 노드를 검색하여 비단말 노드의 수가 범위 질의 처리에 크게 영향을 미치지 않기 때문으로 보인다.

질의 시점 변화에 따른 특정 시점 질의(c-1)와 공간 범위 변화에 따른 특정 시점 질의(c-2)에선 입력 비용(a)이 감소한 만큼 검색 비용이 증가하는 것을 볼 수 있다. 노드 접근 수를 고려하면 입력 비용을 줄이는 편이 더 효과적이지만, 응용에 따라 적은 검색 비용을 원할 때에는 버퍼 노드의 시간 간격을 줄여 사용할 수 있다.

6. 결 론

다양한 위치 기반 서비스 응용에 따라 색인의 입력과 검색 비용을 적절히 유지하는 것이 필요하다. 이 논문에서는 과거 위치 색인에서 가변 버퍼 노드를 활용하여 사용자가 입력과 검색 비용의 상반관계의 균형을 조절하는 방법을 제시하고 이를 활용한 GIP-FB를 설계하였다. 가변 버퍼 노드 방식의 효과를 검증하기 위하여 수식을 통해 버퍼 크기에 따른 입력 비용을 비교하였으며, 실험을 통해 과거 위치 색인의 입력 및 검색 성능을 효과적으로 조절할 수 있음을 확인하였다. 앞으로는 실시간 센서 데이터 및 대용량 데이터를 다루는 다양한 응용에서 실험 평가를 수행하고 자동으로 버퍼 노드의 크기를 조절하는 기법과 색인 구조를 설계할 예정이다.

참 고 문 헌

[1] Y. J. Jung, K. H. Ryu, M. S. Shin, S. Nittel, "Historical Index Structure for reducing Insertion and Search Cost in LBS," The Journal of systems and software, Vol.83, No.1, pp. 1500-1511, 2000.

[2] 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.

[3] O. Wolfson, B. Xu, S. Chamberlain, L. Jiang, "Moving Objects Databases: Issues and Solutions," SSDBM, 1998, pp.111-122.

[4] L. Forlizzi, R. H. Guting, E. Nardelli, M. Schneider, "A Data Model and Data Structures for Moving Objects Databases," ACM SIGMOD, 2000, pp.319-330.

[5] A. Guttman, "A.:R-trees: a Dynamic Index Structure for Spatial Searching," ACM-SIGMOD, pp.47-57, 1984.

[6] 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.

[7] 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.

[8] K. U, Kalipsiz O., "A comparison study of moving object index structures, " Journal of Computer Science and Technology, Vol.24, No.6, pp.1098 - 1108, 2009

[9] C. S. Jensen, D. Lin, B. C. Ooi, "Query and Update Efficient B+tree based Indexing of Moving Objects, " 30th International Conference on Very Large Data Bases (VLDB), pages 768-779, 2004.

[10] D. Lin, "Indexing and Querying Moving Objects Databases," PhD thesis, National University of Singapore, 2006.

[11] C. S. Jensen, D. Tiesyte, N. Tradisauskas, "Robust B+-Tree-Based Indexing of Moving Objects," Seventh International Conference on Mobile Data Management, pp. 12, 2006.

[12] S. Chen, B. C. Ooi, K. L. Tan, M. A. Nacismento, "ST2B-tree: A Self-Tunable Spatio-Temporal B+-tree for Moving Objects," ACM SIGMOD International Conference on Management of Data (SIGMOD), pp.29-42, 2008.

[13] M. Zhang, S. Chen, C. S. Jensen, B. C. Ooi, Z. Zhang, "Effectively Indexing Uncertain Moving Objects for Predictive Queries," 35th International Conference on Very Large Data Bases (VLDB), Vol.1, No.1, pp.1198-1209, 2009.

[14] S. Chen, D. Lin, C. S. Jensen, "A Benchmark for Evaluating Moving Objects Indexes," 34th International Conference on Very Large Data Bases (VLDB), pp.1574-1585, 2008

[15] D. Pfoser, Y. Theodoridis, C. S. Jensen, "Indexing Trajectories of Moving Point Objects," CHOROCHRONOS TECHNICAL REPORT CH-99-03, October, 1999.

[16] D. Pfoser, C. S. Jensen, Y. Theodoridis, "Novel Approaches in Query Processing for Moving Objects," CHOROCHRONOS TECHNICAL REPORT CH-00-03, 2000.

[17] 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.

[18] C. S. Jensen, D. Lin, B. C. Ooi, "Query and Update Efficient B+-Tree Based Indexing of Moving Objects," VLDB, 2004, pp.768-779.

[19] Samet, H., "The Design and Analysis of Spatial Data Structures," Addison-Wesley, pp.47, 1990.

[20] S. Saltenis, C. Jensen, S. Leutenegger, M. Lopez. "Indexing the Positions of Continuously Moving Objects," ACM-SIGMOD, pp.331-342, 2000.

[21] 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.

[22] 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.

[23] Y. Tao, D. Parpadias, J. Sun, "The TPR*-tree: an optimized spatio-temporal access method for predictive queries," VLDB, pp.790-801, 2003.

[24] 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.

[25] G. R. Hjaltason , H. Samet, "Speeding up construction of PMR quadtree-based spatial indexes," VLDB Journal, Vol.11, No. 2, 2002, pp.109-137.

[26] N. An , R. Kanth , V. Kothuri , S. Ravada, "Improving performance with bulk-inserts in Oracle R-trees," the 29th VLDB, 2003, pp.948-951.

[27] 정영진, 류근호, "차량 위치 정보 저장을 위한 버퍼 노드 기반 그룹 갱신 기법," 한국정보처리학회 논문지, 33-D 권, 1호, pp. 1 ~ 11, 2006.

정 영 진

e-mail : yjjung@kisti.re.kr

2000년 충북대학교 전자계산학과(이학사)

2002년 충북대학교 전자계산학과(이학석사)

2007년 충북대학교 전자계산학과(이학박사)

2007년~2010년 미국 Univ. of Maine Visiting Scholar

2010년~현 재 한국과학기술정보연구원 선임연구원

관심분야 : 워크플로우, 센서 데이터 추상화, 센서 데이터 융합, 이동 객체 데이터베이스, 이동 객체 색인, Temporal GIS

안 부 영

e-mail : ahnyoung@kisti.re.kr

2003년 공주대학교 교육정보대학원(교육정보학 석사)

2009년 충남대학교 문헌정보학과(문헌정보학 박사)

1982년~현 재 한국과학기술정보연구원 책임기술원

관심분야 : 비문헌정보, 메타데이터, 데이터베이스

이 양 구

e-mail : leeyangkoo@dblab.chungbuk.ac.kr
2002년 청주대학교 컴퓨터정보공학과(학사)
2004년 충북대학교 전자계산학과(이학석사)
2010년 충북대학교 전자계산학과(공학박사)
2009년~2010년 일본 University of Aizu 연구 학생
2010년~현 재 충북대학교 전자정보대학 Post-doc
관심분야 : 시공간 데이터베이스, 센서 네트워크 데이터베이스, 스트림 데이터 처리, 데이터 마이닝 등

이 동 규

e-mail : dglee@dblab.chungbuk.ac.kr
2005년 서원대학교 정보통신공학과(공학사)
2007년 충북대학교 전자계산학과(공학석사)
2007년~현 재 충북대학교 전자계산학과 박사과정
2010년~현 재 일본 University of Aizu 연구생
관심분야 : 데이터마이닝, 바이오메디칼인포매틱스, 공간정보시스템

류 근 호

e-mail : khryu@dblab.chungbuk.ac.kr
1976년 숭실대학교 전산학과(이학사)
1980년 연세대학교 전산전공(공학석사)
1988년 연세대학교 전산전공(공학박사)
1976년~1986년 육군군수 지원사 전산실 (ROTC 장교), 한국전자통신연구원 (연구원), 한국방송통신대 전산학과(조교수) 근무
1989년~1991년 Univ. of Arizona Research Staff(TempIS 연구원, Temporal DB)
1986년~현 재 충북대학교 전자정보대학 소프트웨어전공 교수
관심분야 : 시간 데이터베이스, 시공간 데이터베이스, Temporal GIS, 지식기반 정보검색 시스템, 데이터마이닝, 데이터베이스 보안, Biomedical 및 Bioinformatics