위치 기반 서비스를 위한 가상 셀 기반 B+-tree 이동객체 색인 기법

박용훈†, 서동민††, 송석일†††, 유재수†
† 충북대학교 정보통신공학과, †† 한국과학기술정보연구원 정보기술연구실,
†††충주대학교 컴퓨터공학과
yhpark1119@chungbuk.ac.kr, dmseo@kisti.re.kr, sisong@chungju.ac.kr, yjs@chungbuk.ac.kr

Virtual Cell based B+-tree Index Structure of Moving Objects for Location Based Services

Yonghun Park†, Dongmin Seo††, Seokil Song†††, Jaesoo Yoo†
† Dept. Information and Communication Engineering, Chungbuk National University, Korea
†† Dept. Information Technology research, Korea Institute of Science and Technology Information, Korea
††† Dept. Computer Engineering, Chungju National University, Korea

요 약

최근 위치 인식 기술과 휴대 장치의 발달로 인해 이동하는 객체를 기반으로 하는 위치 기반 서비스(Location Based Service, LBS)의 관심이 점점 증가하고 있고 그에 관련된 연구들이 활발하게 진행되고 있다. 이동 객체의 응용은 빈번하게 변경되는 이동객체의 위치정보를 효과적으로 처리할 수 있는 색인구조를 필요로 한다. 위치정보를 색인하기 위해 R-tree 기반의 색인들이 제안되었다. 하지만 R-tree는 변경보다는 검색 연산에 초점이 맞추어진 색인구조이기 때문에 잦은 변경을 다루어야 하는 이동객체 환경에 적합하지 못하다. 최근 이러한 객체의 빠른 위치 변경을 지원하는 그리드 기반의 색인 구조가 제안되었다. 하지만 셀의 객체 점유율에 따라 검색 속도가 저하되는 단점은 여전히 해결되지 못하고 있다. 이러한 단점은 객체들이 특정 영역에 몰리는 경우 또는 그리드의 해상도를 잘못 지정한 경우 더욱 부각된다. 본 논문에서는 이러한 단점을 해결하기 위해 가상 셀 기반의 색인 구조를 제안한다. 데이터 페이지에 객체의 점유율을 보장하기 위해 여러 개의 인접한 셀들의 데이터를 한 데이터 페이지에 함께 저장한다. 공간 채움 곡선을 기반으로 순서화된 셀들로 셀의 인접성을 결정한다. 또한 공간 채움 곡선의 차수를 동적으로 지정하여 객체가 집중된 셀에 대해서는 셀의 단위 크기를 작게 지정한다. 뿐만 아니라 셀을 표현하기 위한 식별자를 위해 비트를 이용한 표현식을 제안하였다. 이로 인해 노드의 팬아웃을 증가시켰고, 저장공간을 절약하였다. 실험을 통해서 제안하는 색인 기법의 우수성을 증명하였다.

1. 서 론

위치 인식 기술과 휴대 장치의 발달로 인해 이동하는 객체를 기반으로 하는 위치 기반 서비스(Location Based Service, LBS)의 관심이 점점 증가하고 있다. 위치 기반 서비스는 특정 객체의 위치에 따르는 정보를 제공하는 서비스이다. 인터넷을 통해 수화물의 현재 위치를 제공하거나, 현재 사용자 위치로부터 가장 근접한 주유소 정보를 제공하는 것을 예로 들 수 있다. 현재 위치 기반 서비스의 수준은 미비하지만 현재 GPS 모듈을 포함하는 휴대폰의 생산과 수요가 증가하고 있을 뿐만 아니라 지도 데이터를 기반으로 하는 다양한 콘텐츠가 개발되고 있기 때문에 앞으로 성장 가능성이 크다.

이동 객체를 대상으로 하는 응용은 이동 객체에 대한 지속적인 대량의 변경연산을 효율적으로 처리해야 한다. 게다가 대량의 변경연산을 처리하면서 동시에 다차원 질의를 효과적으로 처리해야 한다. 이러한 요구사항은 다차원 데이터 색인 기술에서 해결하기 쉽지 않은 문제이다. R-tree 계열의 색인구조는 저차원의 데이터를 색인하기 위해 쓰고 주로 대용량의 정적인 데이터 집합에 최적화되었으며 변경연산 처리에 있어서는 낮은 처리 성능을 보인다[1].

R-tree 기반 색인구조들의 변경연산 성능을 향상시키기 위한 연구들이 제안되었다[2][3]. 이들 색인 기법들은 변경 성능 향상을 위해서 각각 변경 지연 기술과 상향식 변경기법을 제시하였다. 하지만

이 논문은 2007 년도 정부(교육인적자원부 학술연구조성사업비)의 재원으로 한국학술진흥재단의 지원(KRF-2007-314-D00221)과 2009 년도 정부(교육과학기술부)의 재원으로 한국연구재단의 지원을 받아 수행된 기초연구사업(No. 2009-0089128)의 결과임.

여전히 변경 성능을 향상해야 하는 과제가 남아 있었다. 특히, 대량의 변경 연산이 동시에 수행될 때 효과적으로 처리하지 못하는 동시성 제어 알고리즘이 문제를 악화시킨다. 노드 분할과 MBR변경의 반영을 위해서 트리를 거슬러 올라가는 연산이 빈번하게 되면 잠금 충돌이 빈번해지기 때문이다. 또 다른 문제는, 제안된 색인구조들이 기존의 데이터베이스 관리 시스템(DBMS)에 쉽게 통합되기가 어렵다는 점이다. 그 이유는 대부분의 상용 DBMS에서 R-tree 계열의 이동 객체 색인을 지원하지 않을 뿐만 아니라 R-tree 계열의 이동객체 색인구조를 통합하기 위해서는 DBMS의 커널을 상당부분 수정해야 하기 때문이다.

최근 공간 채움 곡선(Space Filling Curve)을 이용하여 다차원 데이터를 1차원으로 변경하고, 이를 B^+-tree를 통해 색인하는 방법들이 제안되었다[4][5][6]. B^+-tree를 이용하면 다름과 같은 장점을 얻을 수 있다. B+-tree는 기존의 상용 DBMS에서 폭넓게 사용되고 있을 뿐 아니라, 질의 및 변경에 대한 효율성과 안정성이 이미 증명되었다. 두 번째로 1차원 색인구조를 이용하게 되므로 앞에서 언급한 기존의 R-tree 계열의 색인구조의 문제가 발생하지 않는다.

기존에 제안된 공간 채움 곡선을 기반으로 하는 색인 기법들에서는 고정된 차수의 공간 채움 곡선을 사용한다. 다시 말해서 색인 공간을 동일한 크기의 셀들로 구성된 그리드 구조로 관리하고 각 셀을 공간 채움 곡선을 기반으로 정렬하여 B^+-tree로 셀들을 색인한다. 실세계에서 객체들은 균등하게 분포되지 않을 뿐만 아니라 이동객체의 수 역시 변경된다. 이러한 환경에서 셀의 객체 점유율에 따라 검색 속도가 저하되는 단점은 여전히 해결되지 못하고 있다. 이러한 단점은 객체들이 특정 영역에 몰리는 경우 또는 그리드의 해상도를 잘못 지정한 경우 더욱 부각된다.

본 논문에서는 이러한 단점을 해결하기 위해 가상 셀 기반의 B^+-tree 색인 구조를 이용한 VCB-tree를 제안한다. VCB-tree는 데이터 페이지에 고정된 크기의 한 셀 영역에 포함되는 객체 정보를 기록하지 않고, 여러 개의 인접한 셀들의 데이터를 한 데이터 페이지에 함께 저장하여 객체의 점유율을 보장한다. 또한 공간 채움 곡선의 하나인 z-ordering 기법을 기반으로 순서화된 셀들을 기준으로 인접성을 결정하기 때문에 한 데이터 페이지 내의 데이터의 인접성을 보장한다. 게다가 공간 채움 곡선의 차수를 고정하지 않고 동적으로 지정할 수 있는 비트기반의 셀 식별자 표현 기법으로 B^+-tree에서 중간 노드의 팬아웃을 증가시켰다. 제안하는 기법은 데이터 페이지에 객체 데이터 점유율과 인접성을 보장하고 중간 노드의 팬아웃을 증가 시킴으로서 질의 처리 성능을 향상 시킨다. 또한 z-ordering 기법의 1차원 순서를 기반으로 데이터 영역이 관리되기 때문에 분할, 병합 연산이 간단하다. 마지막으로 비트기반 식별자 표현 기법으로 색인의 저장 공간을 절약하여 불필요한 자원의 낭비를 방지한다.

본 논문의 구성은 아래와 같다. 2장에서는 관련연구를 기술한다. 3장에서는 제안하는 기법인 VCB-tree의 특징과 질의 처리 기법을 기술하고 4장에서는 성능 평가를 통해 그 우수성을 보여준다. 5장에서 결론과 향후 연구를 기술하고 본 논문의 기술을 마친다.

2. 관련연구

이동 객체와 같은 공간 데이터를 색인하기 위해 R-tree와 그 변형의 색인들이 널리 사용되고 있다. 공간 색인 기법으로 R^*-tree[7], R^+-tree[8] 이외에 R-tree 계열의 많은 기법들이 공간 데이터를 대상으로 하는 질의를 효율적으로 처리하기 위해 제안되었다. TPR-tree[9]는 자주 이동하는 객체의 갱신 비용을 줄이기 위해 제안되었다. 객체의 위치뿐만 아니라 갱신 시간과 속도를 함께 저장하여, 객체가 현재 위치 갱신을 하지 않더라도 객체의 위치를 추정하여 질의를 처리한다. 그러나 사장 영역의 증가로 인해 질의 처리 성능 저하를 초래한다.

이러한 이유로 그리드와 B^+-tree기반의 공간 색인이 최근 들어 제안되었다. B^x-tree[4]는 공간을 동일한 크기의 셀로 구성된 그리드로 분할한다. 그리고 각 셀에 식별자를 부여한 뒤, 그 식별자를 키(key)로 하는 B^+-tree를 통해 각 셀을 관리한다. 그리드 구조를 쓰기 때문에 객체의 갱신 속도는 향상되었지만 사장 영역으로 인한 질의 처리 속도의 저하는 여전히 발생한다. 또한 전체 공간에 객체의 밀집도가 균등하지 않을 경우, 질의 처리 속도 저하는 더욱 심해 진다. 이를 해결하기 위해 ST^2B-tree[6]가 제안되었다. 이미지 처리 시 동일한 색을 찾는 알고리즘을 바탕으로 비슷한 밀집도를 가지는 영역을 구별하여, 그 중심점의 보로노이 다이어그램(Voronoi Diagram)을 통해 영역을 분할한다. 그리고 영역별로 동일한 해상도의 그리드가 아닌 해당 영역의 객체 밀집도에 따른 적절한 그리드 해상도를 계산하여 B^x-tree를 구축한다. 이 방식은 객체의 쏠림 현상이 발생할 경우 발생하는 질의 처리 성능 저하 문제를 해결한다.

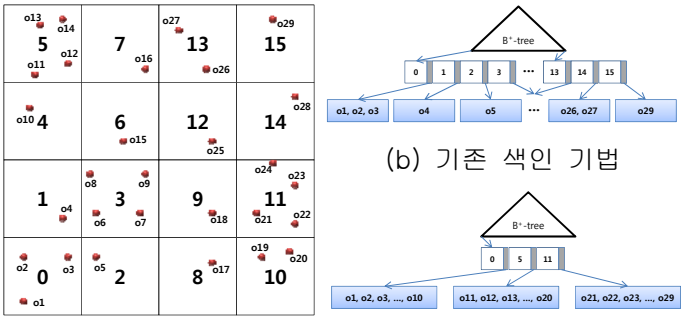
이들 방법은 힐버트 곡선을 이용해서 2차원 이상의 데이터를 1차원으로 변환하고 있다. 변환을 위해서는 먼저 힐버트 곡선의 차수를 결정해야 한다. 이때, 차수를 너무 크게 주면 저장 공간이 많이 소요되고 너무 작게 주면 같은 값을 갖는 객체들이 많아져서 검색 성능이 저하된다. 따라서, 데이터 분포에 따라서 차수가 변경될 수 있다면 색인구조의 성능이 향상 될 것으로 예측할 수 있다. 결정된 차수는 변경이 될 경우 색인구조를 다시 구성해야 하므로 색인구조를 유지하는 동안 변경하기 어렵다.

3. 제안하는 VCB-tree(Virtual Cell based B$^+$-tree)

3.1 VCB-tree의 기본 개념

기존의 기법들은 전체 색인 공간을 적절한 크기의 그리드 형태로 분할하고 각 셀을 한 데이터 페이지에 할당한다. 공간 채움 곡선을 이용하여 셀에 순서화된 식별자를 부여하고 식별자를 기반으로 B$^+$-tree를 구성한다. 이러한 경우 데이터 페이지에 데이터 점유율을 보장하지 못하기 때문에 한 셀에 데이터가 적게 존재하거나 또는 존재하지 않더라도 한개의 데이터 페이지가 할당되어 저장 비용뿐만 아니라 질의 처리 비용을 증가 시킨다. 또한 객체가 한 셀에 집중되는 경우 고정된 크기의 셀을 가지기 때문에 여러 데이터 페이지가 한 셀에 할당되어 질의 처리시 불가피하게 여러 데이터 페이지를 탐색해야 하여 질의 처리 비용을 증가 시킨다.

VCB-tree는 전체 색인 공간을 그리드 구조로 분할하고 공간 채움 곡선을 이용하여 셀에 순서화된 식별자를 부여한다. 그리고 그 식별자를 기반으로 B$^+$-tree를 구성한다. 기존 기법과 차이점은 전체 색인 공간을 가상의 그리드 셀로 분할하고 한 데이터 페이지에 그 셀들의 식별자 범위를 지정하는 방식으로 여러 셀을 할당한다. 만약 셀에 데이터 점유율이 높으면 여러 개의 셀이 한 데이터 페이지에 할당되고, 만약 셀에 데이터 점유율이 낮다면 한개 또는 몇개의 셀이 한 데이터 페이지에 할당된다. 그림 1은 기존 기법과의 차이를 나타낸다. 그림 1(a)는 색인 공간에서 데이터 분포를 나타낸다. 색인 공간은 4x4 해상도로 분할되어 있고 셀의 식별자는 공간 채움 곡선의 하나인 z-ordering 기법을 이용하여 할당하였다. 그림 1(b)는 기존의 색인 기법을 나타낸다. 각 셀은 한 데이터 페이지에 할당되고, 셀의 식별자를 기준으로 B$^+$-tree를 구성한다. 한 데이터 페이지의 팬아웃을 10이라고 가정한다. 한 셀의 객체 점유율에 관계 없이 한 데이터 페이지를 할당하기 때문에 16개의 데이터 페이지가 할당되고 또한 이 데이터 페이지를 가리키기 위한 키(key)도 16개가 유지된다. 그림 1(c)는 제안하는 기법인 VCB-tree를 나타낸다. VCB-tree는 셀에 데이터 점유율에 따라 객체의 합이 팬아웃 10을 넘기지 않는 셀 범위를 한 데이터 페이지에 할당한다. 셀 0~4 까지 10개의 객체를 포함하기 때문에 한 데이터 페이지에 셀 0~4을 할당하고 그 데이터 페이지의 키를 0으로 지정한다. 이러한 방식으로 셀 5~10을 한 데이터 페이지에 할당하고 그 키를 5로 지정한다. 셀 11~15를 한 데이터 페이지에 할당하고 그 키를 11로 지정한다. 그러면 키가 각각 0, 5, 11인 데이터 페이지 3개가 생성되고 그 키를 이용하여 B$^+$-tree를 구성한다. 이러한 환경에서 기존 기법의 경우 16개의 데이터 페이지를 이용하지만 제안하는 VCB-tree의 경우 단지 3개의 데이터 페이지를 이용한다.



(a) 객체 분포 (b) 기존 색인 기법 (c) VCB-tree

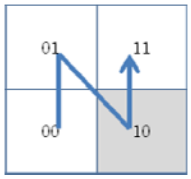
그림 1. 기존의 색인 기법과 VCB-tree의 차이점

3.2 비트 식별자 표현

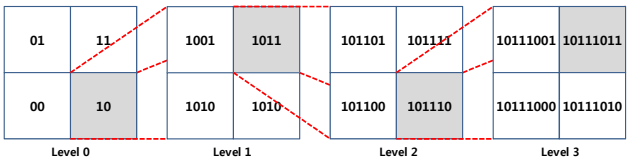
VCB-tree는 데이터 페이지에 할당된 키의 크기를 줄여서 중간 노드의 데이터 팬아웃을 늘리기 위해 가상 셀 기반의 비트 식별자 표현 방식을 제안한다. 만약 전체 색인 공간이 2x2의 해상도로 표현할 경우 2개의 비트로 각 셀을 식별할 수 있다. 4x4 해상도로 표현할 경우 4개의 비트로 각 셀을 식별할 수 있다. 일반적으로 식별자를 할당하기 위해 4바이트 정수를 사용하지만 상황에 따라서 32비트 길이의 식별자가 필요하지 않다. 그래서 가상으로 분할된 그리드의 해상도에 따라 사용되는 비트 식별자의 길이가 결정된다. 그림 2는 비트 식별자의 생성 패턴을 나타낸다. 비트 식별자는 부모 셀의 비트 P_{bit}와 부모 셀에서 분할된 4개의 셀을 표현하기 위한 지역 셀의 비트 L_{bit} 로 구분된다. 한 셀의 식별자 C_{bit}는 P_{bit} 뒤로 L_{bit}를 추가하여 결정된다. 그림 2(a)는 L_{bit}를 할당하는 방법이다. 그림 2(b)는 한 셀의 전체 비트 식별자를 결정하는 것을 보여준다. 전체 색인 공간을 2x2로 분할한 것을 level 0로 한다. 이때 나타나는 4개의 셀은 L_{bit}를 할당받지만 최상위 노드이기 때문에 P_{bit}는 존재하지 않는다. 그래서 L_{bit}가 곧 C_{bit}가 된다. level0에서 어둡게 표현된 셀(C_{bit}=10)을 분할하여 level1의 셀로 표현한다면 그 셀들의 식별자를 계산하기 위해서 다시 L_{bit}를 할당한다. 그리고 그 셀들의 P_{bit}는 10이기 때문에 level1의 셀 식별자는 '10'+L_{bit}로 결정된다. 이러한 방식으로 객체의 점유율에 따라 level의 깊이를 동적으로 지정한다.

비트 식별자 표현 기법을 이용해 각 셀의 식별자를 결정하고 이는 곧 데이터 페이지를 구분하는 키로 사용된다. 데이터 페이지의 키로 B$^+$-tree가 구성된다. 비트 식별자의 길이가 동적으로 결정되기 때문에 이에 따른 노드의 새로운 자료구조가 필요하다. 노드에서 데이터 정보를 표현하기 위해 사용되는 데이터는 <L_{bit}, *Key*, *Add*>이다. *key*는 비트 식별자로 표현된 하위 노드의 키, *Add*는 하위 노드의 정보가 저장된 디스크 페이지 주소 그리고 L_{bit}는 키의 비트 식별자 level을 나타낸다. L_{bit}는 곧 비트 식별자의 길이를 나타낸다. 그림 3은 노드에 저장된 자료의 예를 나타낸다. 그림

3(a)는 데이터 구조를 나타내고 그림 3(b)는 level 값에 따른 비트 길이를 나타낸다.



(a) z-ordering 기법을 이용한 비트 할당 패턴



(b) 비트 식별자 결정

그림 2. 비트 식별자 생성 패턴

L_{bit}	C_{bit}	Add
0001	0110	...
0010	011101	...
0011	01100110	...

(a) 데이터 구조

L_{bit}	Length of C_{bit}
0000	2
0001	4
0010	6
0011	8
...	...

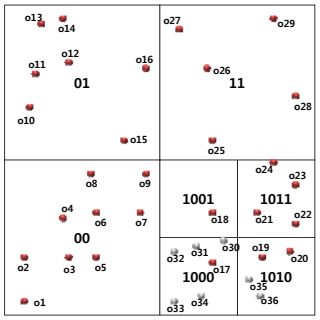
(b) level 값에 따른 비트 길이

그림 3. 노드에 저장된 데이터 표현

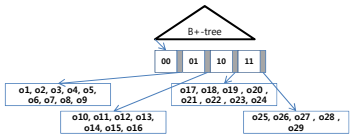
3.3 노드 분할

객체가 삽입되어 한 노드에 오버플로우(overflow)가 발생하면 노드 분할을 수행한다. VCB-tree는 z-ordering 기법의 1차원 순서를 기반으로 데이터 영역이 관리되기 때문에 분할이 간단하다. 오버플로우가 발생한 노드의 셀 범위를 포함하는 객체의 개수가 동일하도록 두개의 셀 범위로 분할한다. 분할이 수행되면 분할된 노드의 기존의 키를 그대로 유지하는 노드와 분할된 노드의 셀 영역 중 특정 위치의 셀의 식별자를 키로 갖는 노드로 구성된다. 그리고 새로운 키를 갖는 노드를 B^+-tree에 삽입한다.

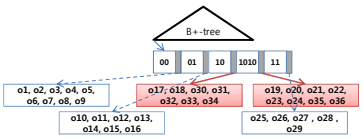
그림 4는 데이터 삽입으로 인한 노드 분할의 예를 보여준다. 데이터 페이지의 팬아웃이 10이라고 가정한다. 시간 t0에 객체 o1~29가 존재하고 데이터 분포는 그림 4(a)에서 나타낸다. 시간 t0에서 VCB-tree의 구조는 그림 4(b)에서 나타낸다. 시간 t1에 객체 o30~38이 삽입 된 경우 키가 '10'인 노드에서 오버플로우가 발생한다. 비트 식별자가 '10'인 셀을 분할하여 객체를 균등하게 분할하는 두개의 셀 식별자 영역 '10'~'1000'과 '1001'~'1010'을 생성한다. 그리고 '1010'를 키로 하는 데이터 페이지를 추가하고 B^+-tree에 삽입한다. 그림 4(c)는 시간 t1에 노드 분할 후의 VCB-tree의 구조를 나타낸다.



(a) 객체 분포



(b) 시간 t0에서 VCB-tree



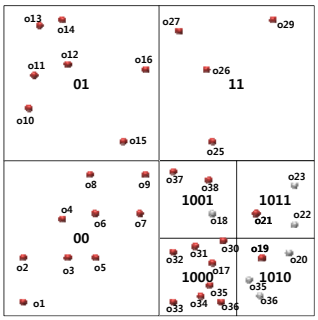
(c) 시간 t1에서 VCB-tree

그림 4. 데이터 삽입으로 인한 노드 분할

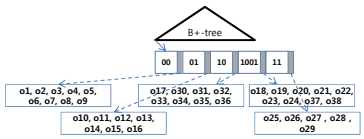
3.4 노드 병합

객체가 삭제되어 한 노드에 언더플로우(underflow)가 발생하면 이웃 노드와 비교하여 노드 병합 및 영역 재조정을 수행한다. 식별자 순서상의 앞 노드와 뒷 노드 객체 개수를 총합하여 두 노드로 병합이 가능하면 병합을 수행한다. 언더플로우가 발생한 노드는 삭제하고 그 노드가 포함하는 객체 데이터와 셀 영역은 양쪽의 이웃 노드 또는 한쪽의 이웃 노드에게 양도한 후 삭제된다.

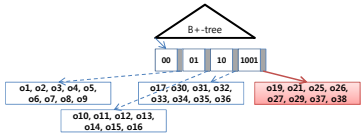
그림 5는 데이터 삭제로 인한 노드 병합의 예를 보여준다. 데이터 페이지의 팬아웃이 10이라고 가정한다. 시간 t0에 객체 o1~38이 존재하고 데이터 분포는 그림 5(a)에서 나타낸다. 시간 t0에서 VCB-tree의 구조는 그림 5(b)에서 나타낸다. 시간 t1에 어두운 색으로 표시된 객체가 삭제 된 경우 키가 '1001'인 노드에서 언더플로우가 발생한다. 키가 '1001'인 노드의 이웃 중 키가 '11'인 노드에 포함된 객체의 수가 한 노드의 팬아웃 보다 적어서 병합을 수행한다. 두 노드의 데이터를 한 데이터 페이지로 이동하고 그 데이터 페이지의 키는 '1001'로 지정한다. 그림 5(c)는 시간 t1에 노드 병합 후의 VCB-tree의 구조를 나타낸다.



(a) 객체 분포



(b) 시간 t0에서 VCB-tree



(c) 시간 t1에서 VCB-tree

그림 5. 데이터 삭제로 인한 노드 병합

3.5 질의 처리 기법

VCB-tree에서 영역질의 처리는 B^+-tree의 특성을 이용하여 효과적으로 수행이 된다. 질의의 형태는 사각형으로 가정한다. 질의가 내려지면 질의 영역 좌측

하단 꼭지점 좌표를 포함하는 가장 작은 가상 셀의 비트 식별자를 계산을 통해서 만들어 낸다. 그리고 그 식별자를 키로 하여 VCB-tree를 순회하여 그 탐색 키를 포함하는 영역을 가진 노드를 탐색한다. 해당 노드에서 질의 영역과 겹치는 모든 객체를 알아낸다. 계속해서 아직 검색되어 지지 않은 인접한 영역의 가상 식별자를 계산하고 그 식별자를 포함하는 영역의 노드를 탐색하는 방식으로 질의를 처리한다. 이 과정을 질의영역을 포함하는 모든 영역을 데이터 페이지를 순회할 때까지 반복한다.

4. 성능 평가

본 논문에서는 가장 최근 발표된 B-tree 기반의 이동객체 색인 기법인 ST^{2}B-tree, 그리고 R-tree 기반의 색인 구조인 TPR-tree와 제안하는 VCB-tree를 다양한 실험을 통해서 비교한다. 구현하는 색인구조의 페이지 크기는 모두 1Kbyte로 고정하였다. 데이터 공간은 2차원이며 각 차원의 영역은 1~1000으로 하였다.

제안하는 색인구조와 TPR-tree, ST^{2}B-tree의 저장공간을 비교한 하였다. 저장공간의 크기는 전체 객체의 개수를 변경시켜 가면서 측정하였다. VCB-tree의 저장 비용이 다른 색인 구조에 비해 현저히 적다. 이것은 노드의 객체의 점유율을 고려하고 동적인 분할 기준으로 노드를 분할하고 병합하기 때문이다.

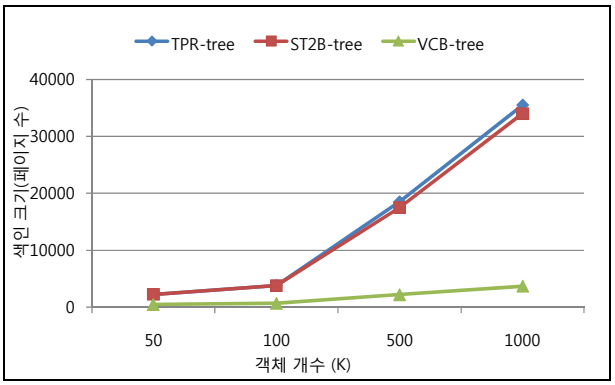


그림 6. 데이터 크기에 따른 저장 비용

객체의 이동에 따른 색인 갱신 비용을 비교하였다. 실험에서는 객체의 개수를 50,000개에서 1,000,000개로 증가시키면서 페이지 접근 횟수를 측정하였다. 총 5,000개의 변경연산을 수행하였고 제시된 결과는 평균 페이지 접근 횟수이다. 제안하는 색인 기법은 ST^{2}B-tree와 VCB-tree의 경우 TPR-tree보다 현저한 성능 향상을 보였고 VCB-tree의 경우 ST^{2}B-tree보다 30% 나은 성능을 보였다. VCB-tree의 경우 노드에 대한 객체 점유율을 보장하고 또한 비트 식별자를 이용하여 노드 팬아웃을 증가시키는 데서 B+-tree의 레벨차이가 나고 또한 객체가 다른 노드로 이동하는 연산이 줄어들기 때문이다.

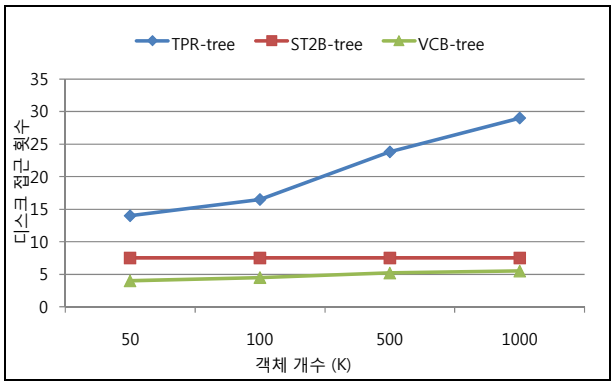


그림 7. 데이터 크기에 따른 갱신 비용

5. 결론 및 향후 연구

본 논문에서는 가상 셀을 이용한 B$^+$-tree 기반의 새로운 이동객체 색인 기법인 VCB-tree를 제안하였다. VCB-tree는 데이터 페이지에 고정된 크기의 한 셀 영역에 포함되는 객체 정보를 기록하지 않고, 여러 개의 인접한 셀들의 데이터를 한 데이터 페이지에 함께 저장하여 객체의 점유율을 보장한다. 또한 공간 채움 곡선의 하나인 z-ordering 기법을 기반으로 순서화된 셀들을 기준으로 인접성을 결정하기 때문에 한 데이터 페이지 내의 데이터의 인접성을 보장한다. 게다가 공간 채움 곡선의 차수를 고정하지 않고 동적으로 지정할 수 있는 비트기반의 셀 식별자 표현 기법으로 B$^+$-tree에서 중간 노드의 팬아웃을 증가시켰다. 실험을 통해서 VCB-tree의 우수성을 증명하였다. 실험 결과 다른 색인들 보다 변경 연산이나 질의 성능 모두 우수함을 확인하였다.

참고 문헌

[1] A.Gutmann, “R-trees: A dynamic index structure for spatial searching”, In Proc. ACM SIGMOD intl. conf. Management of Data, 1984.

[2] D. Kwon, S. Lee, and S. Lee, "Indexing the Current Positions of Moving Objects Using the Lazy Update R-tree," In Proceedings of MDM, pp.113-120, 2002.

[3] M. L. Lee, W. Hsu, C. S. Jensen, and K. L. Teo, "Supporting Frequent Updates in R-trees: A Bottom-up Approach," In Proceedings of VLDB, pp.608-619, 2003.

[4] C. S. Ensen, D. Lin, and B. C. Ooi, "Query and Update Efficient B$^+$-Tree Based Indexing of Moving Objects," In Proceedings of VLDB, pp.768-779, 2004.

[5] M. L. Yiu, Y. Tao, and N. Mamoulis, "The B^{dual}-tree: Indexing Moving Objects by Space Filling Curves in the Dual Space," submitted to VLDB Journal, 2006.

[6] S. Chen, B. C. Ooi, K. L. Tan, and M. A. Nascimento, "ST^{2}B-tree: A Self-Tunable Spatio-Temporal B+-tree Index for Moving Objects", In Proc. intl. conf. SIGMOD, 2008.

[7] N. Beckmann, H.-P. Kriegel, R. Schneider and B. Seeger, “The R*-tree: an efficient and robust access method for points and rectangles”, In Proc. ACM SIGMOD intl. conf. Management of Data, 1990.

[8] T. Sellis, N. Roussopoulos, and C. Faloutsos, “The R$^+$-Tree: A dynamic index for multi-dimensional objects”, In Proc. intl. conf. VLDB, 1987.

[9] Y. Tao, D. Papadias, and J. Sun, "The TPR-Tree: An Optimized Spatio-Temporal Access Method for Predictive Queries," In Proceedings of VLDB, pp.790-801, 2003.