동적 Hilbert Curve 색인을 이용한 효율적인 *k*-NN 질의 처리

서동민[O†] 이승우[†] 김평[†] 정한민[†] 박용훈[††] 유재수[††]
[†]한국과학기술정보연구원 정보기술연구실 [††]충북대학교 정보통신공학과
{dmseo, swlee, pyung, jhm}@kisti.re.kr, {yhpark, yjs}@chungbuk.ac.kr

Efficient *k*-NN Query Processing using the Dynamic Hilbert Curve Index

Dongmin Seo[O†] Seungwoo Lee[†] Pyung Kim[†] Hanmin Jung[†] Yonghun Park[††] Jaesoo Yoo[††]
[†]Dept. of Information Technology Research, Korea Institute of Science and Technology Information
[††]Dept. of Information and Communication Engineering, Chungbuk National University

1. 서 론

무선 통신, 임베디드 시스템, GPS(Global Positioning System) 기술 그리고 유비쿼터스 컴퓨팅 기술이 빠르게 발전하면서, 이동객체 위치를 관리하는 데이터베이스 시스템을 기반으로 하는 응용들이 급속히 성장하였다. 이에 따라 데이터베이스 분야에서는 이동객체 응용들이 고부가가치 서비스를 제공하기 위해 필요한 요구조건들을 만족시킬 수 있도록 많은 연구가 진행되었다. 특히, 주요 연구 쟁점들 중 하나가 효과적인 이동객체 색인을 이용해서 이동객체 응용의 성능을 향상시키는 것이다. 기존에 제안된 이동객체 색인들은 R-트리 기반의 색인과 B^+-트리 기반의 색인으로 분류할 수 있다. TPR-트리와 TPR*-트리[1]는 대표적인 R-트리 기반의 이동객체 색인으로, R-트리의 MBR을 이용해 객체들을 클러스터링하기 때문에 우수한 검색 성능을 제공한다. 하지만, 객체의 위치 변경에 따른 갱신 연산을 처리하기 위해서는 MBR 갱신이 요구되며, 한 MBR 갱신은 다수의 다른 MBR 갱신을 요구하기 때문에 검색 성능의 저하를 가져온다. 즉, R-트리 기반의 이동객체 색인은 객체의 위치가 빈번하게 변경되는 환경에 최적화되지 못했다. 최근, R-트리 기반의 이동객체 색인의 문제를 해결하기 위해 B^+-트리 기반의 이동객체 색인이 제안되고 있다. B^+-트리는 R-트리에 비해 우수한 갱신 성능을 제공하고 상용DBMS에 적용할 수 있는 장점을 가지고 있다. B^x-트리와 ST^2B-트리[2]는 B^+-트리 기반의 이동객체 색인으로, Hilbert Curve를 이용해 이동객체의 2차원 위치 정보를 1차원 정보로 변환해 B^+-트리에 색인한다. 그리고 Hilbert Curve로 변환된 정보들은 지역성(Locality)을 가지고 색인되기 때문에, R-트리에 떨어질 수 있는 B^+-트리의 검색 성능을 향상시킨다. 하지만, 해상도가 고정된 Hilbert Curve 사용하기 때문에 그리드의 크기가 변경되지 않고 이동객체가 불균등 분포되면 검색 성능이 현저히 감소하는 문제를 가진다. 본 논문에서는 *k*-NN 질의를 효율적으로 처리할 수 있는 B^+-트리 기반의 색인을 제안한다. 제안하는 색인은 Hilbert Curve의 해상도를 객체의 분포와 개수에 따라 가변적으로 적용하는 기법을 사용한다. 이 기법은 객체가 불균등 분포되면, 특정 영역의 Hilbert Curve의 해상도를 증가시켜 전체 검색 성능의 저하를 가져오지 않으면서 공간 활용도가 높은 색인을 유지시킨다.

2. 제안하는 B^{dH}-트리 기반의 *k*-NN 질의 처리 기법

2.1 B^{dH}-트리 색인구조

그림 1은 객체가 그림 2(b)와 같이 분포된 상황에 대해, 최대 Hilbert Curve의 해상도가 $2(01_b, 10_b)$이고 한 페이지에 최대 저장될 수 있는 객체 수가 3인 B^{dH}-트리의 색인구조를 보여준다. B^{dH}-트리는 B^{link}-트리를 기반으로 한다. B^+-트리와는 다르게 B^{link}-트리는 단말노드뿐아니라 중간노드들도 링크로 연결이 되어 있다. 중간노드는 디렉토리 노드 역할을 수행하고 오른쪽 형제노드를 가리키는 포인터를 포함한다. B^{dH}-트리에서 단말노드에는 색인 대상이 되는 이동객체의 위치에 대한 Hilbert Curve 값을 저장한다. 단말 노드 엔트리는 (*cv*, *pid*) 쌍으로 구성된다. *cv*는 객체의 위치 정보에 대한 Hilbert Curve 값이고 *pid*는 객체 페이지에 대한 페이지 ID(identifier)이다. 객체 페이지란 이동 객체들의 실제 정보를 저장하는 페이지이다. B^{dH}-트리는 그림 2(b)와 같이 *cv*가 같은 객체들이 되도록 한 객체 페이지에 저장될 수 있도록 데이터의 분포에 따라 지역적으로 다른 Hilbert Curve 해상도를 부여한다. 이를 통해서, 색인의 크기를 줄일 수 있을 뿐 아니라 객체의 위치가 변경되더라도 Hilbert Curve 값이 변경되지 않을 확률이 높아져서 잦은 변경에도 유연하게 대처할 수 있다. 반면에 ST^2B-트리는 한 참조 점에 포함된 모든 그리드는 동일 Hilbert Curve 해상도를 갖는 그리드를 사용하기 때문에 공간 활용도가 떨어져도 그림 2(a)와 같이 한 객체 페이지에 한 객체를 할당하는 형태로 분할된다.

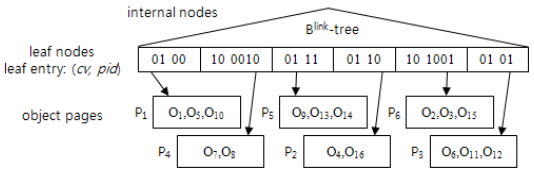

그림 1. B^{dH}-트리의 색인구조

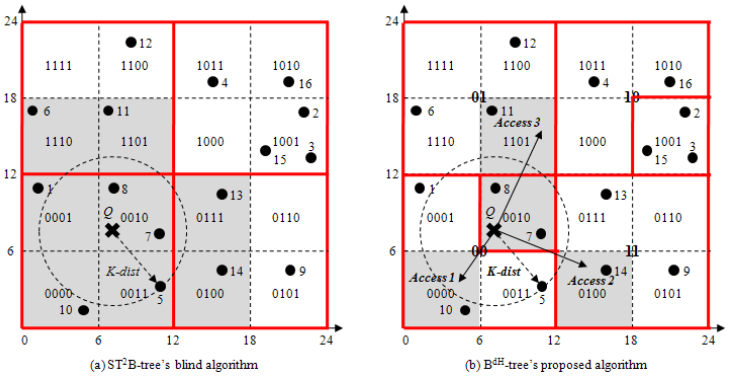

그림 2. ST^2B-트리와 B^{dH}-트리의 k-NN 처리 방법

2.2 B^{dH}-트리의 k-NN 질의 처리 알고리즘

그림 3은 B^{dH}-트리에서 제안하는 k-NN 알고리즘을 보여준다. 그림 2는 ST^2B-트리와 B^{dH}-트리에서 3-NN 처리에 대한 예제를 보여준다. 그림 2(a)와 같이, ST^2B-트리에서 위치가 X:7, Y:7인 질의 Q를 포함하는 영역에 대한 키는 100010_b이다. 그리고 키 100010_b에 대한 페이지에는 두 객체 밖에 존재하지 않기 때문에 맹목적 알고리즘을 통해서는 주변 키(그림 2(a)에서 어두운 영역)들에 대한 페이지 접근이 요구된다. 키 101000_b에 대한 페이지는 *KDist*와 겹치지 않기 때문에, NN 검사 시 제외된다. 이와 같은 방법은 많은 객체 페이지 접근이 요구된다. 하지만, 그림 2(b)와 같이, B^{dH}-트리는 공간 활용도를 최대화하기 위해 Hilbert Curve 해상도가 동적인 키를 사용하기 때문에 ST^2B-트리에 비해 적은 페이지 접근을 요구한다. 예를 들어, ST^2B-트리에서의 100000_b, 100001_b 그리고 100011_b 키들은 B^{dH}-트리에서 0100_b로 표현되기 때문에, NN 검사 시 한 페이지 접근만으로 이들 영역에 포함된 모든 객체 정보를 획득할 수 있다.

Algorithm: *KNN_BdHtree(a k-NN query)*

Output: A result set about k-NN
01: // *H.C.R* is Hilbert Curve Resolution
02: Compute KEY_{query} with the highest *H.C.R* about a query
Search KEY_{page} that is same with KEY_{query} or involve KEY_{query} in B^{dH}-tree
03: *KDist* = 0; $Level_{extend}$ = 0; $Queue_{KEY}$ = empty;
04: **WHILE** (TRUE)
05: Access the page with KEY_{page}
06: Compute *NN* about the objects in the page and set the result of *k-NN* and *KDist*
07: **IF** (*KDist* is covered with the page and find all *kNN* and $Queue_{KEY}$ is not empty)
08: **BREAK**
09: **END IF**
10: **IF** ($Queue_{KEY}$ is empty)
11: After $Level_{extend}$++, find keys that are the highest *H.C.R* and are covered with the extended range about $Level_{extend}$
12: The keys insert in $Queue_{KEY}$ by a descending order
13: **END IF**
14: Search KEY_{page} that is same with $Queue_{KEY}.POP$ or involve $Queue_{KEY}.POP$ in B^{dH}-tree
15: **END WHILE**

그림 3. B^{dH}-트리에서 제안하는 k-NN 알고리즘

3. 결 론

제안하는 k-NN 질의 처리 성능을 비교하기 위해서 모든 실험은 팬티엄 Ⅳ 2.93GHz CPU, 4GB의 주기억장치를 갖는 컴퓨터에서 수행되었다. 구현하는 색인구조의 페이지 크기는 모두4Kbyte이고 실험에서 사용된 데이터 셋은 [1]에서 사용한 데이터 셋을 사용했다. 그림 4는 k 값을 변경하며 100개의 NN 질의를 처리하기 위해 요구되는 페이지 접근 수를 보여준다. B^{dH}-트리는 지역적으로 데이터의 분포에 따라 Hilber Curve의 해상도가 동적으로 변경되는 키를 사용하기 때문에 주기적으로 데이터의 분포를 계산해 새로 색인을 구축하는 ST^2B-트리에 비해 공간 활용도가 높다. 즉, B^{dH}-트리는 팬-아웃(fan-out)이 증가해 질의 처리 시 적은 페이지 접근을 요구해 검색 성능이 증가한다.

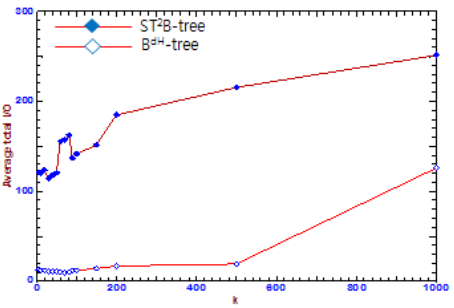

그림 4. k-NN 처리 성능평가

4. 참고문헌

[1] Y. Tao, D. Papadias, and J. Sun, "The TPR*-Tree: An Optimized Spatio-Temporal Access Method for Predictive Queries," *Proc. of VLDB'03*, pp.790-801, 2003.
[2] S. Chen, B.C. Ooi, K.L. Tan, and M.A. Nascimento, "ST^2B-tree: A Self-Turnable Spatial-Temporal B+-tree Index for Moving Objects," *Proc. of SIGMOD'08*, pp.29- 42, 2008.