주기억장치 상주형 다차원 색인 구조 설계

심정민[O] 송석일 유재수
충북대학교 정보통신공학과 및 컴퓨터정보통신연구소
jmshim@netdb.chungbuk.ac.kr, sisong@chungju.chungbuk.ac.kr, yjs@cbucc.chungbuk.ac.kr

Design of a Multi-dimensional Index Structure based on Main Memory

Jeong Min Shim[0] Seok Il Song Jae Soo Yoo
Dept. of Computer and Communication Enginnering, Chungbuk National University

요 약

최근 중앙처리장치와 주기억장치간의 병목 현상에 의한 성능 저하를 극복하기 위해 캐시를 고려한 색인 구조들이 제안되었다. 이런 색인 구조들의 궁극적인 목표는 엔트리 크기를 줄여 팬-아웃(fan-out)을 증가시키고, 캐시 접근 실패를 최소화하여 시스템의 성능을 높이는 것이다. 엔트리의 크기를 줄이는 기법에 따라 기존의 색인 구조들을 두 가지로 구분할 수 있다. 하나는 좌표 값을 고정된 비트로 양자화 함으로써, MBR 키를 압축하는 것이다. 또 다른 하나는 MBR들의 각 좌표 값 중에 그들의 부모 MBR과 같지 않은 좌표 값만을 저장하는 것이다. 본 논문에서는 두 기법의 특성들을 적절히 합한 새로운 색인 구조를 제안하고, 기존에 제시된 두 접근법을 따르는 주기억장치 상주형 다차원 색인 구조를 다양한 환경에서 성능 평가한다. 또한, 기존의 색인 구조와 비교를 통해 제안하는 색인 구조의 우수성을 보인다.

1. 서 론

최근, L2 캐시 접근 실패를 줄임으로서 주기억장치 상주형 데이터베이스 관리 시스템의 성능을 향상시키는 연구들이 활발히 진행되었다[1, 2, 3, 4, 5, 6]. 1990년대 말 이후로 캐시를 고려한 색인 구조는 주기억장치 상주형 데이터베이스 관리 시스템의 성능을 향상시키기 위한 주된 관심이 되었다. 특히, 2000년대 초에는 지리정보시스템(Geographical Information System : GIS), 위치기반시스템(Location Based System : LBS)과 같은 현대 응용 분야가 발전됨에 따라 캐시를 고려한 다차원 색인 구조가 제안되었다.

캐시를 고려한 다차원 색인 구조의 대표적인 것들은 CR-트리[5], PR-트리[6] 그리고 노드의 크기를 캐시 라인 또는 그 정수 배 크기로 사용하는 R-트리[7]가 있다.

본 논문에서는 CR-트리, PR-트리, R-트리와 같은 기존 색인 구조들의 성능을 실험한다. 그리고, PR-트리와 CR-트리의 특성들을 조합한 새로운 색인 구조를 제안한다. 본 논문에서는 실험을 통해 CR-트리와 PR-트리의 결합을 통한 성능 개선 방법을 연구한다. 마지막으로, 제안하는 색인 구조가 다양한 환경에서 CR-트리, PR-트리, R-트리 보다 우수함을 보인다.

본 논문은 다음과 같이 구성되어 있다. 2장에서는 기존의 캐시를 고려한 색인 구조에 대하여 기술한다. 3장에서는 본 논문에서 제안하는 PCR-트리에 대해 설명한다. 4장에서는 다양한 실험을 통해 제안하는 색인 구조의 우수성을 보이고, 마지막으로 5장에서 결론을 맺는다.

2. 관련 연구

1990년대 말에 Rao와 Ross는 주기억장치 상주형 데이터베이스 관리 시스템의 성능을 향상시키기 위해서 캐시를 고려한 두 개의 색인 구조를 제안했다. OLAP 환경을 위해 고려된 CSS-트리(Cache-Sensitive Search-트리)는 엔트리들이 매우 밀집되고, 공간 효율적인 B+-트리이다[1]. 색인에서 포인터를 제거하고 물리적으로 연속적인 공간에 키를 저장한다. CSB+-트리(Cache-Sensitive B+-트리)는 부모노드에 있는 포인터를 줄이기 위해 자식 노드들을 물리적으로 인접한 공간에 저장한다[2].

※ 본 연구는 2001년도 한국학술진흥재단 (KRF - 2001 - 041 -E00233)의 지원으로 수행되었음

반면에, [3]의 부분 키 기법은 전체 키 중에서 일부 고정된 크기의 부분 키만을 사용함으로서, 키 비교 비용과 캐시 접근 실패를 줄였다. 그리고, pB+-트리는 효과적으로 캐시 접근 실패를 줄이기 위해 프리페치 기법을 B+-트리에 도입했다[4]. 검색을 하는 동안 발생하는 캐시 접근 실패를 줄이기 위해 노드의 크기를 캐시 라인의 정수 배로 확장하였다. 따라서, 색인 구조의 높이는 낮아진다.

이상 설명한 색인 구조들은 B+-트리와 같은 일차원 색인 구조들이다. 다차원 데이터를 위한 방법들로는 CR-트리와 PR-트리가 있다.

2.1 PR-트리

주기억장치 상주형 데이터베이스 관리 시스템에서 R-트리의 노드 크기는 보통 캐시 라인과 같거나 그 정수 배이다. PR-트리는 R-트리에서 부모 MBR의 좌표값과 겹침이 발생하는 자식 MBR의 좌표값을 제거해 노드의 크기를 줄이는 부분 MBR 기법을 사용하였다.

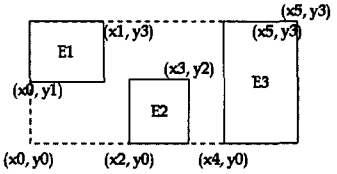


그림 1. 부분 MBR 기법의 예

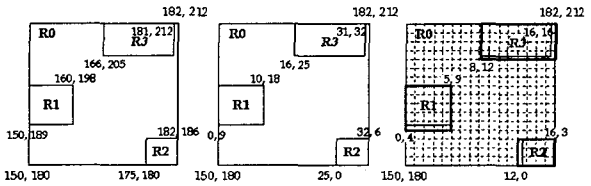
그림 1은 PR-트리의 부분 MBR 기법의 예를 보인 것이다. 그림 1에서 E1, E2, E3를 포함하는 점선은 부모 MBR이다. E1의 x0와 y3는 부모 MBR의 좌표 값과 같다. 이것은 부모 MBR을 알고 있을 경우 E1의 MBR은 2개의 좌표 값 x1과 y1만으로 표현이 가능하다. 같은 방법으로 E2는 x2, x3, y2만이 저장되어 지고, E3에 대해서는 x4만이 저장되어 진다. 저장되지 않은 값은 부모 MBR로부터 얻을 수 있다.

PR-트리는 MBR중 어떤 좌표 값이 저장되어 있는지를 나타내기 위해 각 엔트리에 대한 비트-필드(bit-field)를추가 하였다. 비트-필드는 4비트로 구성하고, 각각의 비트는 E1의 좌표 값들이 저장되어 있는지를 나타낸다. 그림 1에서 E1은 두 개의 좌표 값

x1, y1을 저장하고, B1에서 이 좌표 값들에 대응하는 두 개의 비트를 1로 설정한다. 나머지 두 개의 비트는 좌표 값이 저장되지 않았음을 표시하기 위해 0으로 설정하고, 이에 대한 좌표 값은 부모 MBR로부터 얻는다.

2.2 CR-트리

PR-트리에서 사용하는 부분 MBR 기법의 궁극적인 목적은 노드 안의 엔트리로부터 중복되는 정보를 제거 함으로서 노드의 팬-아웃을 증가시키는 것이다. CR-트리 또한 PR-트리와 목적은 같지만 접근 방법은 매우 다르다.



(a) (b) (c)

그림 2. MBR 압축 기법의 예

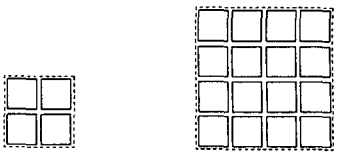
CR-트리는 MBR 압축 기법을 사용하여 노드의 크기를 줄인다. 그림 2는 CR-트리의 MBR 압축 기법의 예를 보여준다. 그림 2(a)는 R0~R3의 MBR들을 절대 좌표로 표현하고 있다. 그림 2(b)는 R0의 좌측-아래(150, 180)를 기준으로 하는 R1~R3의 MBR들을 상대 좌표로 표현하고 있다. 상대 좌표는 절대 좌표들보다 훨씬 작은 수로 표현이 가능하다. 그림 2(c)는 16레벨 또는 4비트로 양자화 된 R1~R3의 좌표 값들을 보인다. 이렇게 양자화된 MBR을 QRMBR(quantized relative representation of MBR)이라고 부른다. 그림 2(c)에서처럼 QRMBR은 양자화로 인하여 원래의 MBR보다 약간 커질 수 있다.

CR-트리는 색인 키로 QRMBR을 사용한다. MBR을 압축하는 양자화 레벨은 모든 노드에서 같다. 노드에 저장된 QRMBR을 계산하기 위하여 노드에는 엔트리의 모든 자식 MBR을 포함하는 최소 크기의 참조 MBR을 유지한다.

3 PCR-트리 : 제안하는 알고리즘

PR-트리는 각 노드마다 엔트리의 수가 3~7개로서, 엔트리 수가 적을 때 좋은 성능을 보인다. PR-트리에서는 노드의 엔트리 수가 증가함에 따라 부모 MBR과 겹치는 좌표의 비율은 줄어든다. 예를 들어, 2차원 데이터 공간에서 최대 엔트리 수가 3일 경우 부모 MBR과 겹치는 최소의 좌표 수는 4개이다. 만약 하나의 좌표 값이 4바이트를 차지한다면, 3개의 엔트리를 표현하기 위한 공간은 48바이트만큼 필요하다. 이 경우 겹치는 비율은 33%가 되고, 32바이트의 저장 공간이 사용되어 진다. PR-트리에서는 2차원의 경우, 각 엔트리마다 추가적인 정보로써 4비트의 비트-필드가 필요하다. 따라서 3개의 엔트리를 표현하기 위한 공간의 총계는 (32바이트+12비트) = 33.5바이트가 된다. 그러나, 노드의 최대 엔트리 수가 40개 일 경우에도 겹치는 최소 좌표 수는 4개이다. 결과적으로, 엔트리를 표현하기 위한 공간은 (640-16)바이트 + (4*40)비트 = 644바이트가 된다. 이 경우 공간적인 이득을 거의 얻을 수 없다.

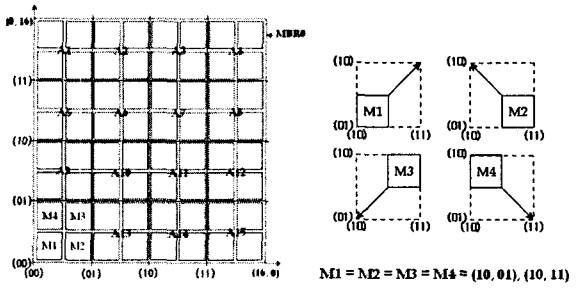
그림 3(a)는 점선으로 표현된 부모 MBR에 같은 크기의 MBR 4개가 균등하게 분포되어 있는 것을 표현하고 있다. 그림 3(a)의 노드에 포함된 좌표는 총 16개이고, 그 중 8개가 부모 MBR과 겹친다. 이 경우 50%의 저장공간이 절약된다. 반면에, 그림 3(b)의 노드에 포함된 좌표은 총 64개이고, 그 중 16개의 좌표가 부모 MBR과 겹친다. 이것은 전체 좌표의 25%에 해당한다. 부모 MBR의 좌표와 겹치는 좌표의 비율은 노드의 엔트리 수가 증가함에 따라 감소한다.



(a) (b)

그림 3. 같은 크기의 MBR을 각각 4, 8개 포함하는 두 노드

CR-트리의 MBR 압축 기법은 MBR 정보의 크기를 감소시켜 노드의 팬-아웃을 증가시키는 것이다. 캐시 라인의 크기가 64바이트이고 하나의 좌표가 4비트를 차지한다면, 하나의 엔트리는 2바이트로 표현이 가능하고 QRMBR 이외의 정보들이 25바이트를 차지하므로 노드의 최대 엔트리 수는 19개이다. CR-트리에서는 MBR이 양자화 됨으로서 MBR영역이 확장된다. 따라서, 노드의 부모 MBR과 겹치는 MBR의 비율이 증가할 것이다. 그림 4는 CR-트리에서 MBR을 양자화한 후 겹치는 좌표의 비율이 증가하는 예를 나타낸 것이다.



(a) (b)

그림 5. CR-트리에서 좌표 값의 증가

그림 4(a)에서 정수(4바이트)로 표현된 MBR0는 (0, 0)에서 (16, 16)의 영역을 차지하고, 작은 사각형의 MBR들은 모두 MBR0에 포함되어 있다. 만약 2비트로 MBR을 압축한다면, MBR들은 00, 01, 10, 11과 같이 새로운 좌표 값에 따라 확장될 것이다. 이런 결과로 M1, M2, M3, M4는 같은 좌표값(00, 00), (01, 01)으로 표현된다. 유사한 방법으로, 각각의 영역에서 MBR들은 확장되어지고, 그림 4(b)에서처럼 각 영역의 MBR들은 같은 값을 갖게 된다.

그림 4(a)에서 MBR0에 포함된 MBR들의 좌표 중 12.5%가 MBR0와 겹친다. 그렇지만, MBR을 압축한 후에는 좌표의 25%가 MBR0와 겹치게 된다. 이것은 PR-트리에서 노드의 엔트리 수를 3개로 하였을 때와 유사한 결과이다. CR-트리에서 좌표의 크기를 4비트와 8비트로 압축하였을 경우 얼마나 많은 좌표들이 부모 MBR과 겹치는가를 알아보기 위해 간단한 실험을 하였다. 표 1은 실험 결과이다.

표 1. 4, 8비트 압축 레벨에서 부모 MBR과 겹치는 자식 MBR의 비율

	4비트 압축	8비트 압축
균등 분포 데이터	43.75%	28.125%
실제(real) 데이터	50%	34.375%

4비트로 압축한 경우 MBR의 좌표 중 45%가 부모 MBR의 좌표 값과 중복되었다. 8비트로 압축한 경우에는 30%가 중복되었다. 이 결과에서는 추가적인 정보(각 좌표에 대한 비트필드)는 고려하지 않았다. 추가적인 정보를 고려하더라도 4비트 압축의 경우에는 20%, 8비트 압축의 경우에는 18%의 겹침이 발생하였다. 따라서, 부분 MBR 기법과 MBR 압축 기법을 결합하여 사용하는 것은 공간 이득을 얻을 수 있다. 이것은 R-트리의 성능을 향상시킬 수 있는 동기를 부여한다.

그림 5은 본 논문에서 제안하는 PCR-트리의 노드 구조를 보인다. NE(Number of Entry)는 노드가 포함하는 엔트리의 수이고, CP(Child Pointer)는 자식 노드 그룹에 대한 포인터이다. AMBR(Absolute MBR)은 노드의 엔트리들을 포함하는 MBR의 절대 좌표이고, 노드에 포함된 엔트리들의 MBR은 AMBR에 의해서 다시 계산된다. BF는 엔트리들의 비트-필드이고, ENTRIES는 AMBR의 좌표 값들과 겹치지 않는 엔트리들의 양자화 된 MBR 좌표 값이다.

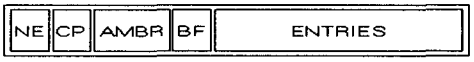


그림 5. PCR-트리의 노드 구조

노드의 엔트리 수를 나타내는 NE는 또 다른 중요한 역할을 수행한다. 모든 삽입과 삭제 연산 과정에서, 부분 MBR 기법과 MBR 압축 기법을 결합하여 노드를 구성하는 방법이 MBR 압축 기법만을 사용하였을 때보다 많은 저장 공간을 차지하는지 확인한다. 만약 MBR 압축 기법만을 사용했을 때, 더 작은 저장 공간을 차지한다면 NE 값을 음수로 바꾼다. NE이 음수 값을 가질 때는 BF에 아무런 값도 갖지 않음을 나타낸다. 즉, 부분 MBR 기법이 적용되지 않은 노드의 BF값은 사용되지 않는다.

4. 성능평가

본 논문에서 제안하는 PCR-트리의 성능을 평가하기 위해 CR-트리, PR-트리, R-트리와 비교하였으며, 노드 크기는 캐시 라인 크기와 그 정수 배로 하였다. 성능 평가에 사용된 시스템은 펜티엄 4 1GHz프로세서에 256Mbytes의 메모리를 가지며, 캐시 라인의 크기는 64bytes이고, 운영체제는 윈도우 2000을 사용하였다. CR-트리, PR-트리, R-트리와 제안하는 PCR-트리는 공통적으로 QS(quadratic splitting algorithm)분할 알고리즘을 사용하였다. 또한, STR(sort tile recursive)[8]벌크로딩 알고리즘을 사용하여 벌크로딩 기법을 구현하였다.

실험에 사용된 데이터들은 균등 분포 데이터와 실제 데이터(TIGER)[9]를 사용하였으며, 검색 성능과 삽입 성능을 측정하였다. 검색 성능을 평가하는데 있어서 10,000개의 범위 질의를 사용하였으며, 데이터 전체 영역의 0.001%~1%에 해당하는 범위 질의에 대해서 접근하는 노드 수, 검색 시간, 캐시 접근 실패 수를 측정했다.

그림 6은 실제 데이터(TIGER)에 대한 검색 성능 결과이다. 그림 6에서 모든 색인 구조가 노드 크기가 증가함에 따라, 검색 시간이 증가하는 것을 볼 수 있다. 노드 크기가 512바이트 이하에서 제안하는 색인 구조인 PCR-트리(8)가 가장 좋은 성능을 보인다.

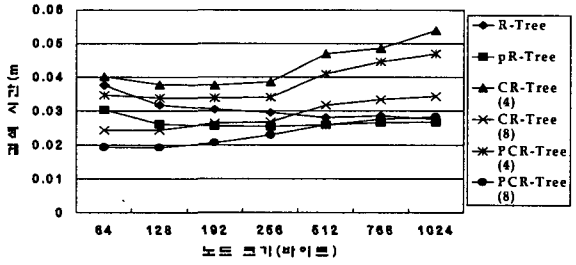


그림 6. 노드 크기에 따른 검색 시간 (실제 데이터)

삽입 성능을 측정하기 위해 10,000개의 엔트리를 삽입하였다. PCR-트리는 상대적인 좌표를 계산하고 압축 기법을 사용하는 것이 이득인가를 확인하여, 그에 따라 노드를 구성한다. 그림 7을 보면 PCR-트리(8)가 R-트리를 제외한 색인 구조들에 비해 좋은 성능을 보인다. PCR(8)의 삽입 성능은 64바이트 일 때 가장 좋은며, 노드 크기가 64바이트와 128바이트 일 때는 R-트리의 삽입 시간과 유사하다.

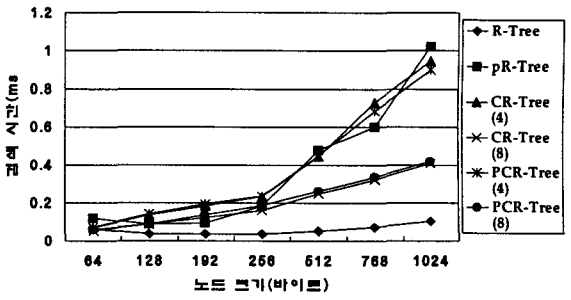


그림 7. 노드 크기에 따른 삽입 시간

5. 결론

본 논문에서는 캐시를 고려한 주기억 장치 상주형 다차원 색인 구조를 제안했다. 제안하는 색인 구조에서는 부분 MBR 기법과 MBR 압축 기법을 결합하여 엔트리의 크기와 캐시 접근 실패를 줄였다. 두 기법을 동시에 적용하여 구성한 노드가 MBR 압축 기법만을 적용한 경우보다 공간 효율이 떨어지는 경우에는 MBR 압축 기법만을 사용함으로써, 두 기법을 동시에 적용한 경우 발생할 수 있는 단점을 보완하였다. 그리고, 다양한 환경에서 폭넓은 성능 평가를 통해 제안하는 PCR-트리가 기존의 트리들 보다 우수함을 보였다. PCR-트리(8)가 검색의 모든 부분에 걸쳐 다른 색인 구조들에 비해 뛰어난 결과를 보였다. 본 논문에서는 기존의 캐시를 고려한 다차원 색인 구조들에 대한 자세한 분석을 통해 향상된 성능의 색인 구조를 제안하였고, 제안하는 색인 구조가 기존의 색인 구조들 보다 우수함을 보였다.

6. 참고 문헌

[1] Jun Rao and Kenneth A. Ross, "Cache Conscious Indexing for Decision-Suppoert in Main Memory", In Proceedings of VLDB Conference, pp. 78-79, 1999.

[2] Jun Rao and Kenneth A. Ross, "Making B+-Trees Cache Conscious in Main Memory", In Proceedings of ACM SIGMOD Conference, pp. 475-486, 2000.

[3] Philip Bohannon, Peter Mcllroy and Rajeev Rastogi, "Main-Memory Index Structures with Fixed-Size Partial Keys", In Proceedings of ACM SIGMOD Conference, pp. 163-174, 2001.

[4] Shimin Chen, Phillip B. Gibbons and Todd C. Mowry, "Improving Index Performance through Prefetching", In Proceedings of ACM SIGMOD Conference, 2001.

[5] Kihong Kim, Sang K. Cha and Keunjoo Kwon, "Optimizing Multidimensional Index Trees for Main Memory Access", In Proceeding of ACM SIGMOD Conference, pp. 139-150.

[6] I. Sitzmann and P.J. Stuckey, "Compacting discriminator information for spatial trees.", In Proceedings of the Thirteenth Australasian Database Conference, pp. 167-176, 2002.

[7] Guttman, A., "R-trees: a Dynamic Index Structure for Spatial Searching." In Proceedings of ACM SIGMOD Conference, pp. 47-47, 1984.

[8] Scott T. Leutenegger, J. M. Edgington, Mario A. Lopez, "STR: A Simple and Efficient Algorithm for R-Tree Packing.", In Proceedings of ICDE Conference, pp. 497-506, 1997.

[9] http://www.cs.du.edu/~leut/MultiDimData.html