다차원 데이터 분석을 위한 비트맵 인덱스

임윤선* 박영선 김 명
이화여자대학교컴퓨터학과 고성능인터넷지식공학연구실
{lys96, ys_park, mkim}@ewha.ac.kr

A Bitmap Index for Multi-Dimensional Data Analysis

Yoonsun Lim Youngsun Park Myung Kim
Dept. of Computer Science & Engineering, Ewha Womans University

요 약

다차원 데이터를 배열에 저장하는 Multidimensional OLAP (MOLAP) 시스템은 배열내의 위치 정보를 통해 데이터를 신속하게 엑세스할 수 있는 장점을 갖는다. 그러나 실생활의 다차원 데이터는 대체로 희박하여 저장될 때 압축되고, 데이터가 검색될 때는 원래의 위치 정보를 찾기 위해 인덱스를 필요로 하게 된다. 다양한 종류의 다차원 인덱스가 테이블 형태의 데이터를 대상으로 개발되어 있으나, 이들은 데이터의 삽입과 삭제에 유연하게 대처할 수 있도록 하기 위해서 인덱스 공간과 데이터 검색 시간에 약간의 낭비를 초래한다. 본 연구에서는 OLAP 데이터가 주기적으로 갱신되며, 분석에 필요한 집계 데이터도 점진적으로 갱신되기보다 실제로는 새로 생성되고 있다는 점을 고려하여, 읽기 전용 MOLAP 데이터를 위한 인덱스 구조를 제안한다. 데이터는 청크들로 나뉜 후 압축 저장되며, 각 청크는 위치 정보를 유지하면서 비트로 표현되어 인덱스에 저장되도록 하였다. 제안한 비트맵 인덱스는 높은 압축률을 보이며, 범위 질의(range query)를 포함한 OLAP 주요 연산들 처리에 특히 효율적이다.

1. 서론

대용량 데이터를 다차원적으로 분석하여 그 결과를 신속하게 제공하기 위해서 OLAP (Online Analytical Processing) 시스템들은 질의처리가 시작되기 전에 입력 데이터로부터 집계 연산을 하여 저장해 놓는다. 데이터와 집계 연산 결과가 다차원 배열에 저장될 때 이 배열을 OLAP 큐브(cube) 라고 부르며 이러한 저장구조 상에서 운영되는 OLAP 시스템을 MOLAP (Multidimensional OLAP) 시스템이라고 한다. 배열 방식으로 데이터를 저장하면 인덱스 없이 데이터를 엑세스할 수 있지만, 실생활 데이터로부터 생성되는 큐브는 대체로 데이터 밀도가 낮아 압축 저장되므로 배열의 위치 정보가 손실되고, 데이터에 신속하게 접근하기 위해서는 MOLAP 다차원 인덱스 구조가 필요하게 된다.

현재 제안된 대표적인 다차원 인덱스는 KDB-트리[1], MLGF(Multi-Level Grid File)[2], UB-트리[3] 등을 들 수 있다. 이들은 데이터 조회 뿐만 아니라 데이터 추가와 삭제 역시 효율적으로 처리하도록 설계되어 있어서 일반 운영계 데이터베이스 (operational database) 시스템의 다차원 데이터를 인덱싱하는 데에도 효율적이다. 반면에 OLAP 시스템의 데이터는 분석용 데이터로써 현재 운영 중인 데이터라기보다 과거의 데이터이다. 이 데이터로부터 집계 연산 등을 통해 각 차원별, 차원 계층별 집계 연산 결과를 분석함으로써 데이터를 분석하는 것이다. 따라서 OLAP 데이터의 갱신은 발생할 때마다 적용되는 것이 아니라 주기적으로 모아서 갱신되고, 데이터가 변경되면 집계 연산 결과들은 새로 생성되어 저장되는 경우가 많다. 본 연구는 이와 같은 읽기 전용 MOLAP 데이터를 위한 인덱스를 설계하는 것을 목표로 한다. 인덱스는 갱신을 허용하지 않는 대신, 높은 압축률을 보장하고 OLAP의 주요 연산인 슬라이스, 다이스, 범위 질의 (range query)와 같이 인접한 데이터를 필요로 하는 연산에 효율성이 뛰어나도록 하였다.

연구에 사용된 MOLAP 데이터의 구조는 [4]에서 제안된 구조를 사용한다. 큐브는 selectivity를 높이기 위해 각 차원별로 동일한 크기의 청크(chunk)로 나누어 압축한 후에 디스크에 저장하였고[5], 청크들의 순서는 클러스터링 효과를 높이기 위해 밀집 청크들과 희박 청크들을 구별하여 Z 인덱스 순서로 저장하는 방식이다[4]. 본 연구에서는 이러한 데이터 구조 상에서 효율적인 비트맵 인덱스 구조를 설계하였다. OLAP 큐브는 압축되어 저장되기 때문에 각 데이터는 위치 정보를 잃을 수 있으나, 이 위치 정보를 인덱스 차원에서 비트맵을 통해 유지시키고, 디스크 한 블록에 가능한 한 많은 인접 청크에 대한 정보를 둠으로써 OLAP 연산시에 읽어야 하는 인덱스 블록의 수를 최소화하도록 하였다. 인덱스의 효율성은 저장공간, 인덱스 조회 시간 등의 파라미터들을 [5] 구조에 사용 가능한 기존의 인덱스 구조인 UB-트리와 비교하여 분석하였다.

본 논문의 구성은 다음과 같다. 2절에서 기존 연구결과를 소개하고, 3절에서 청크 기반 MOLAP 시스템에서의 새로운 인덱스 구조를 제시한다. 4절에서 인덱스 구조에 대해 분석 결과를 설명하고, 5절에서 결론을 맺는다.

2. 관련 연구

이 장에서는 기존에 연구된 다차원 인덱스와 비트맵 인덱스에 대해 살펴보기로 한다. 대표적인 다차원 인덱스로는 KDB-트리[1], MLGF[2], UB-트리[3] 등이 있다. KDB-트리와 MLGF는 다차원 공간을 계층적으로 분할하여 구축되는 인덱스이며 데이터 추가와 삭제를 허용한다. UB-트리 역시 유사한 인덱스 구조

* 본 연구는 한국과학재단 목적기초연구 (R04-2001-000-00191-0) 지원으로 수행되었음.

인데, 이는 다차원 데이터를 Z 인덱스 순서로 나열하여 1차원적으로 데이터를 나열한 후에 이 데이터 상에서 B-트리로 인덱스를 구축한 것이다. 이러한 구조는 데이터의 추가와 삭제를 효율적으로 처리하기 위해서 인덱스에 여유 공간을 두며, 데이터가 차원 축에 평행하게 나뉘어지는 것이 아니므로 데이터 검색 시에 질의 조건에 부합되는 셀을 찾는데 상대적으로 많은 시간이 걸린다.

비트맵 인덱스[6,7]는 낮은 카디널리티(cardinality)를 갖는 차원에 주로 쓰이는 것으로, 데이터의 해당 차원 값이 특정 값인가를 신속하게 찾는데 쓰인다. 카디널리티가 높은 데이터에도 확장 적용하기 위해 Encoded 비트맵 인덱스[8], 비트맵 조인 인덱스 [9]등 변형된 비트맵 인덱스 연구가 있다. 본 연구는 이러한 비트맵 인덱스의 장점을 다차원 데이터의 인덱싱에 확장 적용하는 방안을 제안하고자 한다.

3. 다차원 비트맵 인덱스 구조의 설계

3.1 청크 기반 MOLAP 큐브 저장 구조

우선 본 연구에서 사용하는 MOLAP 큐브 구조를 살펴보기로 한다. MOLAP 데이터 큐브는 질의 성능을 높이기 위해 그림 1(a)와 같이 청크라 부르는 조그마한 배열로 나눠 데이터를 저장한다[5]. 즉, 큐브는 특정 차원에 부당하지 않게 각 차원 길이가 동일하며, 디스크 블록 크기의 청크들로 나뉜다. 청크들 중에서 밀도가 대략 40% 이상인 청크들은 밀집 청크들로 분류되고 남은 청크들은 희박 청크들로 분류된다. 희박 청크들은 압축되어 저장되는데 이 때 유효 셀 들만이 선택되어 저장되며, 각 셀은 자신의 청크 내의 오프셋과 함께 저장된다. 데이터 클러스터링 효과를 높이기 위해 그림 1(b)와 같이 밀집 청크들과 희박 청크들은 분리되어 저장되며, 각 청크 그룹은 공간 분할에 쓰이는 "Z 인덱스" 순서로 저장된다[4].

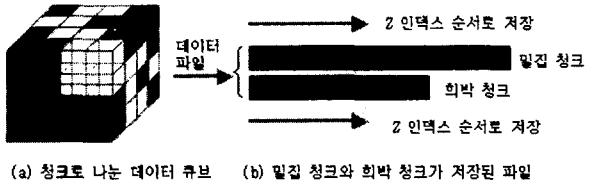

그림 1. MOLAP 큐브 저장 구조

3.2 다차원 비트맵 인덱스 구조

MOLAP 큐브가 3.1절에서와 같이 저장되는 경우에 사용 가능한 인덱스를 제안하고자 한다. 제안하는 인덱스는 압축률이 높으며, OLAP의 주요 연산인 슬라이스, 다이스 질의 시에 해당 데이터의 인덱스 블록을 신속하게 메모리로 읽어 들일 수 있도록 하기 위해 하나의 인덱스 블록이 가능한 많은 데이터 블록의 정보를 포함하도록 하였다.

본 논문에서 제안한 인덱스는 청크를 인덱싱하는 다차원 인덱스 구조로써, 이를 청크 비트맵 인덱스 또는 CBM 인덱스(Chunk Bitmap Index)라고 부르기로 한다. CBM 인덱스의 기본 개념은 데이터 큐브를 저장할 때 희박 청크가 압축되고 널(null) 청크가 제거되면서 손실된 청크의 위치 정보를 인덱스 차원에서 복원하려는 것이다. CBM 인덱스는 다음과 같다: 그림 1(a)의 청크들을 Z 인덱스 순서로 나열한 후에 그 순서대로 각 청크에 대해 인덱스를 만들고, 이를 디스크 한 블록 단위로 잘라 저장한다. 4K 바이트 크기의 디스크 한 블록에는 10K 개의 청크에 관한 인덱스가 포함되어 있고, 청크들이 Z 인덱스 순서로 나열되어 있기 때문에 인접한 청크들이 하나의 디스크 블록으로 인덱싱이 가능하여 클러스터링 효과를 나타낸다.

CBM 인덱스 디스크 블록은 그림 2와 같은 구조를 갖는다. 이 블록에는 크게 두 가지 정보가 포함되어 있다. 첫째는 이 인덱스 디스크 블록에 포함된 밀집 청크들의 시작 주소 (또는 카운트)와 희박 청크들의 시작 주소가 저장된 인덱스 엔트리이다. 둘째는 세 그룹의 비트 시퀀스이다. 각 시퀀스의 길이는 인덱스 디스크 블록에 포함된 청크 개수 (여기서는 10K개)이다. 각 비트 그룹의 i번째 비트는 이 디스크 블록에 포함된 청크들 중에서 i번째 청크에 관한 정보이다. 첫째 비트 그룹 (그림에서 그룹 A)은 해당 청크가 널 청크인가를 나타낸다. 둘째 비트 그룹은 해당 청크가 밀집 청크인지 희박 청크인지를 구분한다. 셋째 비트 그룹은 해당 청크가 희박 청크인 경우 그 청크의 주소를 찾는데 쓰인다.

데이터 검색은 다음과 같이 이루어진다. 검색할 데이터의 각 차원 정보가 주어지면 이로부터 해당 청크 번호를 계산하고, 이 번호로부터 해당 인덱스 디스크 블록을 메모리로 읽어 들인다. 검색할 청크가 인덱스 디스크 블록내에서 k번째 비트에 해당하는 경우 다음과 같은 연산을 통해서 해당 데이터 블록의 주소를 구한다. 우선, 비트 그룹 A의 k를 통해 그 청크가 널 청크인가를 확인한다. 널 청크가 아닌 경우는 비트 그룹 B의 k번째 비트를 조사한다. 밀집 청크이면 비트 그룹 B내의 첫 k비트 중에서 1로 세팅된 것들을 카운트한다. 이 카운트와 인덱스 엔트리의 밀집 청크 시작 주소를 사용하여 해당 청크의 주소를 계산한다. 희박 청크이면 비트 그룹 C의 첫 k비트 중에서 1로 세팅된 것들을 카운트한다. 이 카운트와 인덱스 엔트리의 희박 청크 시작 주소를 사용하여 해당 청크의 주소를 계산한다. 이 그룹의 비트 i는 i번째 청크와 $i-1$번째 청크가 동일 디스크 블록에 저장되어 있는가를 나타낸다. 희박 청크의 경우는 해당 디스크 블록을 스캔하면서 찾을 수 있다.

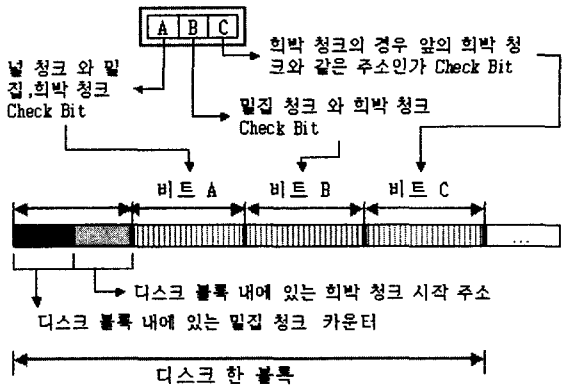

그림 2. CBM 비트맵 인덱스 구조

CBM 인덱스에는 밀집 청크와 희박 청크의 주소를 구하기 위해 해당 비트 시퀀스를 스캔해야 하는 오버헤드가 있다. 이를 해결하는 방법으로 매핑 테이블을 사용할 수 있다. 예를 들어, 16개의 비트 시퀀스 안에 들어 있는 1의 개수를 카운트하는 매핑테이블을 메모리에 상주시켜 연산 속도를 높일 수 있다. 이 경우 매핑 테이블의 엔트리 수는 2^{16}으로 메모리에 충분히 상주 시킬 수 있다.

4. 성능 분석

본 논문에서 제안한 인덱스는 다차원 데이터를 효과적으로 인덱싱하고 분석 시스템에 적합한 UB-트리와 비교하여 공간 및 OLAP 연산의 효율성을 분석하였다.

4.1 저장 공간 분석

본 논문에서 제안한 CBM 인덱스는 차원과 차원의 애트리뷰트에 의해 전체 청크 수가 정해지면 인덱스의 크기는 고정된다. 한 청크에 대하여 3 비트가 필요하고, 디스크 한 블록을 4K 바이트로 했을 경우 10K개의 청크 정보를 비트로 표시하고 10K개 청크에 대하여 밀집 청크 인덱스 엔트리 4바이트와 희박 청크 인덱스 엔트리 4바이트가 필요하다. 그림 4는 3차원, 5차원에 대하여 UB-tree와 CBM 인덱스의 저장공간을 데이터 밀도를 다양하게 하여 비교한 것이다. 그림 4(a)에서 분석한 데이터는 3차원 데이터로 각 차원이 1,000개의 애트리뷰트를 갖고 전체 청크 수는 1M개이다. 그림 4(b)에서 분석한 데이터는 5차원 데이터로 각 차원이 128개의 애트리뷰트를 갖고 전체 청크 수는 33M개이다. 분석 결과를 살펴보면 CBM 비트맵 인덱스가 데이터 밀도가 1% 이상일 때 인덱스 저장 공간을 효율적으로 사용하고 있음을 알 수 있다. 저장 공간의 비율을 살펴보면 데이터 밀도가 10%인 경우 CBM 인덱스는 UB-트리에 비해 1/8 정도의 저장 공간이 필요하고, 40%인 경우 약 1/34의 저장 공간이 필요하게 된다.

실제 데이터에 대한 CBM 인덱스의 압축율을 살펴보자. 그림 1(a)의 경우 1% 데이터 밀도를 갖고 있을 때 최악의 경우 모든 청크에 대해 1% 데이터가 있다고 할 때 실제 데이터 크기는 40M 바이트가 된다. 이 때 CBM 인덱스는 370K 바이트이므로 100배 정도의 압축률을 가진다. 따라서 10% 데이터 밀도일 경우 1000배 정도의 압축률을 갖게 되고 데이터 밀도가 높아지면 더욱 높은 압축률을 갖게 된다.

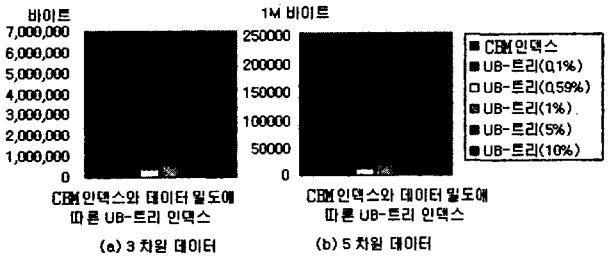

그림 3. 인덱스 저장 공간 비교

4.2 조회 성능 분석

인덱스의 효율성은 인덱스 저장 공간과 검색 시간으로 측정할 수 있다. UB-트리와 비교하여 한 개의 데이터를 검색하기 위한 디스크 엑세스 시간을 알아보기로 한다. CBM 인덱스는 데이터를 검색하기 위해 데이터가 속한 청크 번호를 구하여 청크 번호가 있는 디스크 블록을 한번 접근하여 청크 주소를 구한다. 그러나 B+-트리 구조를 갖고 있는 UB-트리 경우 탐색을 위한 최대 디스크 접근 수는 루트를 접근할 때의 I/O와 단말 노드를 접근할 때의 I/O를 합해 $\log_m(n+1)$시간이 걸린다(m은 한 노드에 표현될 인덱스 키의 개수, n은 전체 인덱스 키의 개수). 즉 CBM 인덱스는 항상 한번의 디스크 접근을 보장하는 반면 UB-트리는 데이터 밀도에 따라 생기는 인덱스 키의 개수에 의해 절대적 영향을 받는다.

CBM 인덱스의 범위 질의에 대한 효율성을 살펴보기로 한다. CBM 인덱스는 OLAP 연산을 효율적으로 하기 위해 Z 인덱스 순서로 나열한대로 청크 비트맵을 구현하므로 범위 질의에 필요한 청크들이 서로 인접해 있게 된다. 또한 청크를 인덱싱하기 위해 비트를 사용하여 정보를 저장함으로서 한정된 메모리에 많은 청크 정보들이 탑재하게 된다. 이것은 인덱스 디스크 블록을 한 번 엑세스함으로 필요한 청크의 정보들이 대부분 메모리에 존재하게 된다는 것을 의미한다. 3차원의 예를 들어보기로 한다. 디스크 한 블록을 4K 바이트라 했을 경우 한 인덱스 디스크 블록 안에는 10K 개의 청크 정보가 들어오게 된다. 디스크 블록 안에 있는 청크는 Z 인덱스 순으로 클러스터링 되어 있으므로 각 차원에 대해 22개의 청크가 클러스터링되어 뭉쳐 있게 된다. 따라서 한 면을 슬라이스 할 경우 최대 500 개의 청크 정보가 클러스터링 되어 있어 한 번의 인덱스 디스크 블록을 엑세스함으로써 높은 청크들의 클러스터링 효과를 보게 된다.

5. 결론

다차원 데이터를 배열에 저장하는 MOLAP 시스템은 배열내의 위치 정보를 통해 데이터를 신속하게 엑세스할 수 있다. 그러나 대부분의 OLAP 데이터는 희박하여 압축 저장하게 되면서 다차원 인덱스가 필요하다. 이에 본 논문은 읽기 전용인 MOLAP 데이터를 효율적으로 인덱싱하는 CBM 인덱스(청크 비트맵 인덱스, Chunk Bitmap Index)를 제안하였다. CBM 인덱스는 청크의 위치 정보를 유지하면서 데이터 밀도에 따른 청크 정보를 비트로 표현하여 인덱스에 저장하였다. 제안한 비트맵 인덱스는 높은 압축률을 보이며, 범위 질의(range query)를 포함한 OLAP 주요 연산들 처리에 특히 효율적임을 분석을 통해 입증하였다. 향후에는 각 차원에 계층이 있는 데이터 큐브에서 요약 정보를 만들기 위해 계층 데이터 정보를 효율적으로 가져올 수 있는 인덱스 연구를 진행한다.

6. 참고문헌

[1] John T. Robinson, "The K-D-B-Tree: A Search Structure For Large Multidimensional Dynamic Indexes," SIGMOD Conference 1981: pp. 10-18.

[2] Kyu-Young Whang, Ravi Krishnamurthy, "The Multilevel Grid File - A Dynamic Hierarchical Multidimensional File Structure", DSFAA 91, pp. 449-459.

[3] R. Bayer, "The Universal B-Tree for Multidimensional Indexing", Technische Universitat Munchen, TUM-I9637, Nov. 1996.

[4] Myung Kim, Yoonsun Lim, "A Z index based MOLAP Storage Scheme", Journal of Korea Information Science Society, Vol.29, No.4, pp. 262-273, August 2002.

[5] Yihong Zhao, Prasad Deshpande, Jeffrey Naughton, "An Array-Based Algorithm for Simultaneous Multidimensional Aggregates," Proc. ACM SIGMOD 1997, pp.159-170.

[6] C.Y. Chan and Y. Ioannidis, "Bitmap index design and evaluation," In Proc. ACM SIGMOD Intl. Conf. on Management of Data, pages 355--366, June pp. 1-4 1998.

[7] P. ONeill and D. Quass. Improved Query Performance with Variant Indexes. Proc. of ACM SIGMOD Conf., Tucson, Arizona,1997, pp. 38-49.

[8] O'Neil P., Grafe G. "Multi-Table Joins through Bitmapped Join Indices," SIGMOD Record, Sep 1995.

[9] M-C.Wu and A.Buchmann. "Encoded bitmap indexing for data warehouses," In Proc. 14th ICDE, Orlando, Florida, pp.220-230, 1998.