Z-인덱스 기반 MOLAP 큐브 저장 구조
(A Z-Index based MOLAP Cube Storage Scheme)

김 명 † 임 윤 선 ††
(Myung Kim) (Yoonsun Lim)

요 약 MOLAP(multi-dimensional online analytical processing)은 데이타의 다차원적 분석 기술로서, 이는 질의 처리 속도를 높이기 위해 데이타를 큐브(cube)라고 불리는 다차원 배열에 저장하고 배열 인덱스를 사용하여 데이타를 엑세스한다. 큐브는 다양한 방식으로 디스크에 저장될 수 있으며 이 때 사용되는 방식에 따라 MOLAP의 주요 연산인 슬라이스와 다이스 연산 속도가 크게 영향을 받는다. 이러한 연산들을 효율적으로 처리하기 위해 다차원 배열을 작은 크기의 청크로 나누고 이 들 중에서 희박한 청크들을 압축하여 저장하는 기법이 [1]에 제안되어 있다. 이 방식에서는 청크들을 행우선 순서로 디스크에 저장한다. 본 연구에서는 청크들을 밀도와 인접도 기준으로 배치시킴으로써 슬라이스와 다이스 연산 속도를 향상시키는 방법을 제시한다. 청크 밀도를 이용하여 청크들을 디스크 블록 경계에 가능한 한 맞추었고, Z 인덱싱을 사용하여 인접한 저밀도 청크들을 군집화 함으로써 디스크 I/O의 속도를 높였다. 제안한 큐브 저장 방식은 일반적 비즈니스 데이타의 분석에 흔히 사용되는 3~5차원의 큐브 저장에 효율적이라는 것을 실험적으로 보였다.

키워드 : OLAP 큐브 저장, Z-인덱스, 청크기반 OLAP 큐브

Abstract MOLAP is a technology that accelerates multidimensional data analysis by storing data in a multidimensional array and accessing them using their position information. Depending on a mapping scheme of a multidimensional array onto disk, the speed of MOLAP operations such as slice and dice varies significantly. [1] proposed a MOLAP cube storage scheme that divides a cube into small chunks with equal side length, compresses sparse chunks, and stores the chunks in row-major order of their chunk indexes. This type of cube storage scheme gives a fair chance to all dimensions of the input data. Here, we developed a variant of their cube storage scheme by placing chunks in a different order. Our scheme accelerates slice and dice operations by aligning chunks to physical disk block boundaries and clustering neighboring chunks. Z-indexing is used for chunk clustering. The efficiency of the proposed scheme is evaluated through experiments. We showed that the proposed scheme is efficient for 3~5 dimensional cubes that are frequently used to analyze business data.

Key words : OLAP Cube Storage, Z-Index, Chunk-based OLAP Cube

1. 서 론

OLAP(online analytical processing)은 데이타를 다차원적으로 분석하여 그 결과를 사용자에게 온라인으로 신속하게 제공하는 기술[2, 3, 4]이다. 이는 지식공학(knowledge engineering) 시스템의 중요한 컴포넌트로써, 기업이 경영 전략을 세울 때나 개인화된 서비스를 제공하는 전자상거래 시스템의 구축을 위한 데이타 분석 엔진에 쓰이는 기술이다. 데이타의 다차원적 분석을 위해서는 분석할 데이타의 상당 부분이 반복적으로 스캔되어야 하므로 OLAP 시스템들은 질의처리 성능을 일정 수준 이상으로 보장하기 위해 분석 결과의 일부를 미리 계산해 두며, 이를 OLAP 큐브 생성(cube generation)이라고 한다[3, 4, 5]. OLAP 큐브 생성과 질의 처리 시간은 데이타의 저장 방식에 따라 크게 달라질 수 있다[1, 6, 7, 8]. 본 연구의 목표는 OLAP 데이타를 효율적으로 저장함으로써 주요 연산들의 속도를 향상시키려는 것이다.

· 본 연구는 한국과학재단 목적기초연구 (R04-2001-000-00191-0) 지원으로 수행되었음.
† 종신회원 : 이화여자대학교 컴퓨터학과 교수
mkim@ewha.ac.kr
†† 비 회 원 : 이화여자대학교 컴퓨터학과
lys96@ewha.ac.kr
논문접수 : 2001년 10월 23일
심사완료 : 2002년 6월 18일

OLAP 데이타의 주요 저장 방식으로는 크게 ROLAP (relational OLAP)과 MOLAP(multi-dimensional OLAP) 방식을 들 수 있다[3]. ROLAP 시스템들은 큐브를 관계형 데이타베이스에 튜플 형태로 저장하고, RDBMS 질의 처리 기술을 바탕으로 OLAP 질의를 처리한다. 튜플에 분석할 데이타와 그 데이타의 차원 정보가 함께 저장되므로 저장 공간이 낭비되고 RDBMS 기술이 OLAP 연산 처리에 최적화된 것이 아니므로 큐브 생성과 질의 처리 속도가 느리다는 단점이 있다[3, 5]. MOLAP 시스템들은 반면 큐브를 다차원 배열에 저장한다. 데이타가 배열 인덱스로 엑세스되기 때문에 데이타가 신속하게 엑세스된다. 배열에 유효(valid) 셀이 많지 않은 경우에는 데이타 압축 기술이 쓰인다.

MOLAP 방식의 큐브는 고속으로 생성될 수 있기 때문에, 이 방식은 ROLAP 시스템의 큐브 생성 속도를 높이는 데에도 쓰일 수 있다[1, 9]. 즉, 튜플 형태의 분석 대상 데이타를 MOLAP 데이타 저장 형태로 변환하고, MOLAP 엔진으로 큐브 생성을 마친 후에 큐브를 다시 튜플 형태로 변환하는 것이 튜플 형태의 분석대상 데이타 상에서 직접 ROLAP 큐브를 생성하는 것보다 현저하게 빠르다는 연구결과가 [1]에 발표되어 있다. [9]는 또한 MOLAP 큐브 생성 방식을 기반으로 메모리 재사용 효율을 높임으로써 고속 확장성 있는 ROLAP 큐브 생성 방법을 제시한다.

본 연구에서는 MOLAP 시스템의 데이타 저장구조에 초점을 맞추어 데이타를 효율적으로 저장함으로써 OLAP의 주요 연산인 슬라이스(slice) 연산과 다이스(dice) 연산의 속도를 높이는 방안을 고안하였다. 예를 들어 분석할 데이타가 특정 상점에서 특정 시각에 팔린 특정 상품의 판매액 집합이라고 하자. 이와 같은 데이타가 그림 1과 같은 3차원 배열에 저장되어 있다고 하자. 이 때 슬라이스 연산은 그림 1(a)와 같이 특정 차원 값을 고정하였을 때의 모든 판매액을 제시하는 것을 뜻하고, 다이스 연산은 그림 1(b)와 같이 각 차원 값에 범위를 주었을 때의 판매액의 집합을 제시하는 것을 뜻한다.

이와 같은 MOLAP 큐브를 그대로 디스크 파일에 저장하면 정해진 차원 순서로 1차원 배열화 되어 저장된다. 예를 들어, 1000×1000×1000 크기의 3차원 데이타가 그림 2(a)와 같이 화살표 방향으로 저장되어 있다고 가정하자. 디스크 블록을 4K 바이트로 하고, 각 데이타가 4 바이트 크기라고 한다면 디스크 한 블록에는 화살표 방향의 한 행이 저장된다. 여기서, 차원 C의 값을 고정한 2차원 평면을 떼어내는 슬라이스 연산을 했다고 하자. 이 슬라이스는 AB면과 평행한 경우로써, 디스크로부터 1000개의 블록을 읽게 되고, 읽은 데이타 전체가 슬라이스 연산에 필요한 데이타이므로 전혀 낭비가 없다. 그러나 슬라이스가 그림 2(a)의 진하게 표시된 부분처럼 AB면에 수직이고 BC면에 수평인 경우는 1개의 디스크 블록에는 슬라이스 연산에 필요한 셀(데이타)이 1개만 들어 있기 때문에 슬라이스 연산을 처리하기 위해서는 필요량의 1000배에 해당하는 데이타를 디스크로부터 읽게 된다. 즉, 그림 2의 예제의 경우는 큐브 전체가 읽히게 된다.

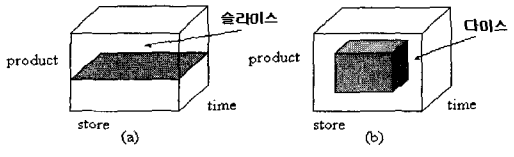


그림 1 슬라이스 연산과 다이스 연산

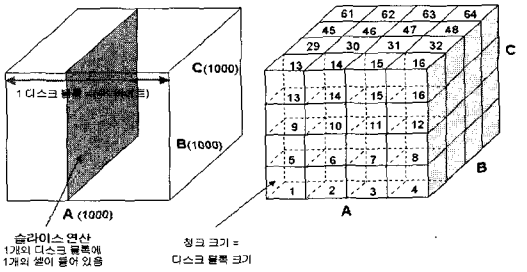


그림 2 디스크에 배열 저장하는 법 (a) 특정 차원 기준으로 1차원 배열화한 경우, (b) 청크 단위로 나누어 저장하는 경우

MOLAP의 슬라이스, 다이스 연산을 효율적으로 처리하기 위해 그림 2(b)와 같은 구조가 [1, 7]에 제안되어 있다. 즉, 배열은 디스크 블록과 비슷한 크기의 청크(chunk)들로 나뉘어 저장된다. 청크 1개가 10×10×10 셀로 구성된다면 2차원 슬라이스 연산을 할 때 방향에 관계없이 디스크로부터 읽혀 들이는 데이타의 양은 항상 필요량의 10배이고 모든 차원에 처리 속도가 공평하게 보장된다는 것을 알 수 있다. 이와 같은 큐브 저장 방식은 다이스 연산에도 효율적이라는 것을 알 수 있다. 청크에 유효 셀이 많지 않은 경우에는 청크를 압축하여 저장한다[1]. 압축 방법은 다음 절에서 자세히 소개하기로 한다. 비즈니스 데이타로부터 생성된 큐브는 일반적으로 밀도가 낮고 유효 셀들이 클러스터를 이루기 때문

에 압축된 청크들은 그 크기가 다양해지게 된다.

본 연구에서는 이와 같이 크기가 다양한 청크들을 밀도와 인접도를 고려하여 저장함으로써 청크 단위로 MOLAP 큐브를 저장하는 경우 슬라이스와 다이스 연산의 성능 개선을 위한 방안을 제시한다. 일반적으로 OLAP 과정을 살펴보면 먼저 큐브가 생성된다. 대형 큐브인 경우 이 작업은 장시간이 걸린다. 비즈니스 데이타의 경우, 일단 생성된 큐브는 여러 사용자가 동시에 반복적으로 여러 날에 걸쳐 사용된다. 따라서, 분석결과의 일부를 사전 연산하여 저장해 두는 것도 질의처리 속도를 높이는 방안이지만, OLAP 질의처리 과정이 시작되기 전에 큐브의 청크들을 질의에 최적화하여 재배치하는 일 역시 질의처리를 고속화하는 데 기여한다.

OLAP 데이타의 저장 방식과 질의 처리 속도 향상에 관한 연구는 현재까지 꾸준히 진행되고 있다. [10]은 ROLAP 사실테이블의 레코드들을 클러스터화함으로써 질의처리의 속도를 높인다. [11]은 앞선 질의를 통해 이미 메모리에 캐쉬되어 있는 데이타의 재사용을 통해 질의 처리 속도를 높인다. [12]는 MOLAP 큐브를 저장할 때 희박 차원들은 인덱스로 사용하고 큐브는 밀집차원들만으로 구성함으로써 저장 공간을 줄인다. 공간 조인 연산의 효율을 높이기 위해 공간 데이타를 Hilbert Curve 방식으로 저장하는 방식인 필터 트리(filter tree)도 데이타 웨어하우스를 저장하는 한 방식으로 [8]에 소개되어 있다. [6]은 여러 개의 희박 차원들을 모아 한 개의 합성 차원(composite dimension)으로 구성하여 저장 공간을 줄이는 방안을 제시한다. 그러나 [6, 12]의 경우에도 MOLAP 큐브의 논리적 구조에 관해 정의를 하는 것이며, [1, 7]이외에는 슬라이스와 다이스 연산이 적용되는 다차원 배열(MOLAP 큐브) 자체를 디스크에 효율적으로 저장하여 연산 속도를 개선하는 연구결과는 제시되지 않았다.

본 논문은 다음과 같이 구성된다. 2절에서 [1]이 제안한 청크단위 MOLAP 큐브 저장방식과 개선점을 관찰한다. 3절에서는 본 연구에서 청크 저장 순서에 사용할 다차원 배열의 Z 인덱싱에 관해 살펴본 후에 Z 인덱싱을 적용한 청크단위 MOLAP 큐브 저장방식을 설명하고 이를 통한 질의처리 고속화에 관해 설명한다. 4절에서는 이를 실제 데이타에 적용한 실험 결과를 제시하고 5절에서 결론을 맺는다.

2. 압축청크 기반의 MOLAP 큐브

우선 [13]이 제안한 MOLAP 큐브 저장 구조를 살펴보기로 한다. MOLAP 큐브는 그림 3과 같이 저장된다. 그림에는 A, B, C 세 개의 차원으로 구성된 베이스 큐브(base cube, base cuboid, 분석할 데이타가 있는 배열)를 저장한 모습이 보인다. 이 방식은 큐브 전체를 저장할 때도 사용된다.

큐브는 모든 차원에 공평한 기회를 부여하기 위해 그림 3과 같이 청크로 나뉘어져서 저장된다. 청크의 크기는 디스크 데이타 블록 크기 정도로 하여, 청크 1개를 메모리로 읽어들일 때, 디스크 블록 1개가 엑세스되도록 한다. 일반적으로 큐브는 그림과 같이 밀집 청크 (dense chunk, 진한 색 청크)들과 희박 청크 (sparse chunk, 연한색 청크)으로 구성된다. [1]은 청크에 유효 셀이 40% 이상 들어 있는 경우를 밀집 청크로 분류한다. 밀집 청크를 파일에 저장할 때는 청크를 배열형태 그대로 저장한다. 희박 청크는 유효 셀만을 골라서 그 셀의 청크 안에서의 주소(offset)와 셀 값(value) 쌍으로 만들어서 저장한다. 디스크 블록의 크기를 4K 바이트라고 볼 때, 주소를 저장하는데 2~4 바이트를 할당하면 되고, 셀 값을 저장하는데 4바이트를 사용한다면, 1개의 셀을 저장하는데 6~8바이트가 필요하다. 청크 크기가 일정하지 않기 때문에 청크의 디스크 시작주소와 청크의 크기는 따로 메타 데이타로 저장한다.

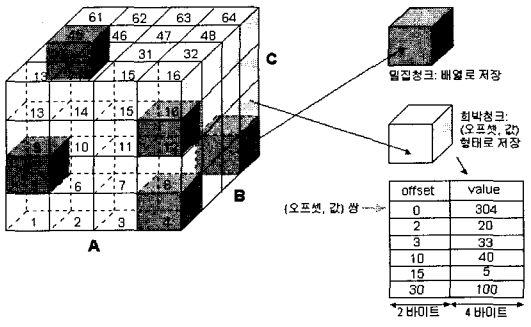


그림 3 청크 단위 MOLAP 큐브 저장 방식

이와 같은 큐브 저장방식을 살펴보면 큐브 상에서 OLAP 연산을 효율적으로 하기 위해 다음과 같은 개선 가능성을 찾을 수 있다. 첫째, 밀집 청크가 디스크 블록과 같은 크기라고 하자. 청크 1개를 읽을 때 디스크 블록 1개가 읽혀질 것이 기대된다. 그러나, 실제로는 밀집 청크들이 희박 청크들과 섞여 있기 때문에 청크의 경계가 디스크 블록의 경계와 일치할 확률이 매우 낮고, 이로 인해 밀집 청크 1개를 엑세스하면 디스크 블록 2개가 읽혀지는 경우가 대부분이다. 청크 저장 순서를 조절하여 이런 현상의 영향을 줄일 필요가 있다. 둘째, 베이

스 큐브는 데이타가 클러스터를 이루면서 평균 밀도가 1% 또는 그 이하로 낮은 경우가 흔하다. 이러한 경우 희박 청크 1개에는 평균 0~10개 정도의 셀이 속해 있고, 디스크 블록 1개에는 이러한 청크들이 수십 개 저장될 수도 있다. [1]처럼 row-major 순서로 청크들을 디스크에 저장한다면, 특정 차원을 기준으로 청크들이 디스크 블록에 모이게 된다. 청크 저장 순서를 조절하여 1개의 디스크 블록이나 연속한 디스크 블록에 저장되는 청크들을 모았을 때, 원래 청크보다 각 변이 2배 이상으로 큰 청크가 되는 효과, 즉 "큰 청크 효과 (big chunk effect)"를 볼 수 있을까? 이렇게 된다면 슬라이스 연산과 다이스 연산을 할 때 디스크 I/O 시간을 줄일 수 있게 된다.

3. Z 인덱스 기반의 MOLAP 큐브 저장구조

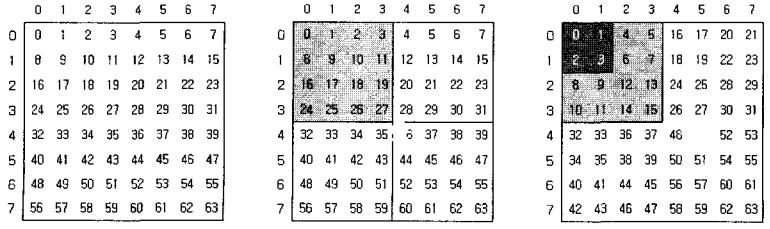
본 연구에서는 이 두 문제를 개선하는데 초점을 맞춘 MOLAP 큐브 저장 방식을 제안한다. 이 방식에 Z-인덱싱이 사용되므로, 우선 Z-인덱싱의 정의와 특징을 설명한 후에 이에 기반한 MOLAP 큐브 저장구조를 소개하기로 한다.

(1) 다차원 배열의 Z 인덱싱

Z 인덱싱은 2~3차원 영상의 픽셀에 번호를 매겨서 영상내의 각 컴포넌트를 신속하게 인식하면서 영상 저장공간을 줄이는데 효과적인 인덱싱 기법이다. 우선, Z-인덱싱을 2차원 배열에 적용하여 설명하기로 한다[13]. 그림 4에 8×8 배열이 있다. (a)는 배열 원소를 row-major 순서 (또는 linear 순서)로 번호 매긴 것이고, (c)는 배열 원소를 shuffled-row-major 순서로 번호 매긴 것이다. shuffled-row-major 인덱싱은 Z 인덱싱이라고도 불린다. 배열을 Z-인덱싱하는 방법은 다음과 같다. 배열의 크기를 $2^n \times 2^n$, $1 \le n$, 이라고 하자. 이 배열을 그림 4(b)와 같이 4개의 작은 배열인 $2^{n-1} \times 2^{n-1}$ 크기로 나눈다. 여기서, 왼쪽 위의 배열은 NW(northwest) 블록이라 하고, 오른쪽 위의 배열은 NE(northeast) 블록, 왼쪽 아래 배열은 SW(southwest) 블록, 오른쪽 아래 배열은 SE(southeast) 블록이라고 하자. Z-인덱싱을 하려면, NW 블록에 속한 원소들을 모두 번호를 매기고 나서, NE 블록에 속한 원소들을 번호 매기고, 그 다음에 SW와 SE 블록 순서로 번호를 매긴다. 이 각 4개의 블록은 다시 동일한 방법으로 순환적으로 분할(recursively decompose)되어 동일한 방식으로 번호가 매겨진다. 그림 4(c)는 NW 블록을 순환적으로 분할한 것을 보인다. 인접한 셀들이 멀리 떨어져 있는 셀들보다 우선적으로 번호가 매겨진다는 것을 알 수 있다.

배열 원소의 Z 인덱스는 쉽게 계산될 수 있다. 예를 들어, r행 c열에 있는 원소 $A(r,c)$의 row-major 인덱스와 Z-인덱스를 계산해 보자. 그림 4와 같이 8×8 배열의 경우, r과 c는 3 비트의 숫자로 표현된다: $r = r_2r_1r_0$, $c = c_2c_1c_0$. 이 경우, $A(r,c)$의 row-major 인덱스는 $r_2r_1r_0c_2c_1c_0$이고, Z 인덱스는 $r_2c_2r_1c_1r_0c_0$이 된다. 즉, row-major 인덱스는 행을 나타내는 3 비트를 앞에 쓰고, 열을 나타내는 3 비트를 뒤에 써서 계산된다. Z 인덱스는 행을 나타내는 비트들과 열을 나타내는 비트들을 한 개씩 번갈아 쓴 것이다. 예를 들어, 그림 4(b)의 $A(5,6)$은 $A(101,110)$으로 표현될 수 있고, 이 원소의 row-major 인덱스는 101110 = 46이고, Z 인덱스는 그림 4(c)와 같이 110110 = 54가 된다.

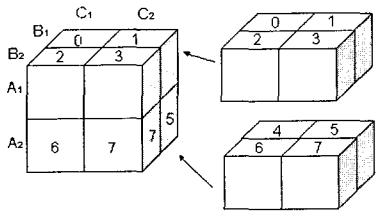
Z 인덱싱을 3차원 배열에 적용할 때는 각 차원을 2등분하여 $2 \times 2 \times 2 = 8$개의 블록이 만들어지고, 그림 4의 2차원 배열에 인덱스를 매겼던 것과 같은 방식으로 각 블록에 인덱스가 매겨진다. 즉, 배열의 각 축을 A, B, C라 하고, 각 차원을 2등분하여 만들어진 8개의 블록을 그림 5와 같이 $A_1B_1C_1$, $A_1B_1C_2$, $A_1B_2C_1$, $A_1B_2C_2$, $A_2B_1C_1$, $A_2B_1C_2$, $A_2B_2C_1$, $A_2B_2C_2$이라 하자. 블록들은 이 순서대로 번호가 매겨진다. 즉, 첫째 블록의 모든 셀의 번호가 매겨진 후에 둘째 블록의 셀들이 번호 매겨지고, 이어서 셋째 블록의 셀들이 번호 매겨진다. 각



(a) row-major 순서 (b) NW 블록 (c) Z 인덱싱 순서

그림 4 8×8 2차원 배열 원소를 Z 인덱싱으로 번호를 매긴 예제

블록의 셀들은 동일한 방법으로 순환적으로 번호가 매겨진다. 따라서, 8×8×8 3차원 배열 $A(r,c,h)$의 Z 인덱스는 $r_2c_2h_2r_1c_1h_1r_0c_0h_0$으로 계산된다. Z 인덱싱은 n 차원에도 확장되어 적용될 수 있는 인덱싱 기법이다. Z 인덱싱으로 셀의 번호를 매기면, 인접한 셀들이 우선적으로 번호가 매겨지므로, 특정 셀을 기준으로 클러스터가 형성된경우, 그 클러스터내의 셀들이 우선적으로 번호가 매겨진다는 것을 알 수 있다.

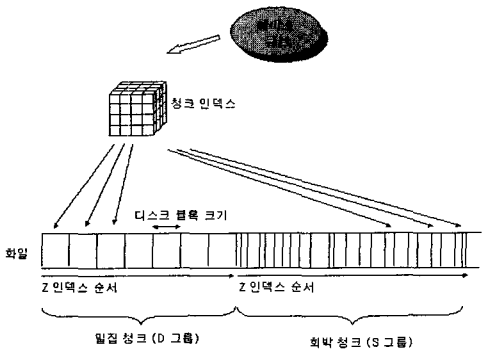


(a) (b)

그림 5 3차원 배열에 적용된 Z 인덱싱.

(2) Z 인덱스 기반의 MOLAP 큐브 저장구조

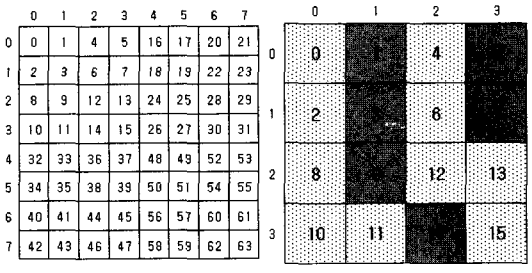
이제 Z 인덱스 기반의 MOLAP 큐브 저장구조를 소개한다. 큐브 저장 방식은 분석할 데이타가 저장되는 베이스 큐브를 중심으로 설명한다. 베이스 큐브로부터 계산된 사전 연산 결과 역시 다차원 배열이므로 베이스 큐브와 동일한 방법으로 저장하면 된다. 그림 6에 파일에 저장된 베이스 큐브의 예제가 있다. 베이스 큐브는 [1] 방식과 같이 청크들로 나뉜다. 각 청크는 Z 인덱스 순서로 번호가 매겨진다. 밀집 청크들은 배열 형태로 저장되고, 희박 청크들은 {offset, value} 쌍으로 변환되어 저장된다. 파일에 청크들이 저장될 때는 그림 6과 같이 D 그룹과 S 그룹으로 나뉘어 저장된다. D(dense) 그룹에는 밀집 청크들이 배열 형태로 저장된다. 따라서 각 청크는 크기가 일정하며 디스크 블록과 크기가 같다. S(sparse) 그룹에는 밀도가 낮은 희박 청크들이 저장된다. 이 들은 {offset, value} 형태로 변환되어 저장된다. 각 그룹내의 청크들은 모두 청크의 Z 인덱스 오름차순으로 저장된다. 각 청크를 가리키는 청크 인덱스는 그림 6에서와 같이 베이스 큐브 형태를 그대로 보존하면서 저장된다. 즉, 원래의 베이스 큐브가 $a\times b\times c$개의 청크를 갖는 3차원 배열이라면, 청크 인덱스 역시 $a\times b\times c$개의 인덱스를 저장하는 3차원 배열이 된다. 이러한 인덱스는 슬라이스 연산이나 다이스 연산 시에 해당 청크의 위치를 신속하게 찾도록 한다.



(a) (b)

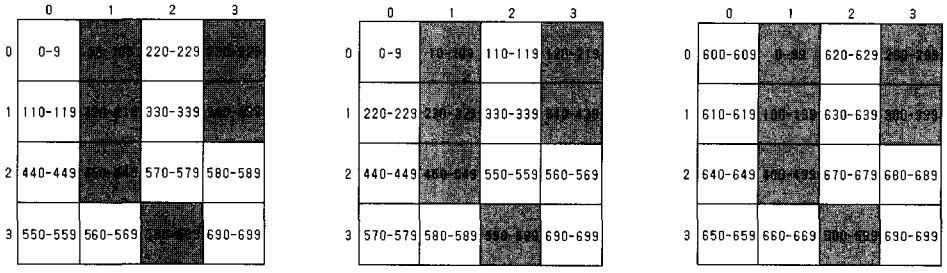
그림 6 Z 인덱스 기반 MOLAP 큐브 저장 구조

제안한 큐브 저장구조를 분석해 보기로 한다. 이 기법이 2절에서 제기한 문제점들을 개선하는데 도움이 되는지 살펴보자. 첫째 문제점은 밀집 청크의 크기가 실제 디스크 블록의 크기와 같지만 청크가 디스크 블록의 경계에 맞지 않아서 청크 1개를 디스크로부터 읽을 때 디스크 블록 2개가 읽히는 경우가 많이 발생한다는 것이었다. 이 문제는 밀집 청크들을 파일의 한 쪽에 모아 따로 저장함으로써 해결된다. 둘째 개선할 점은 큐브의 밀도가 낮은 경우, 1개의 디스크 블록 또는 여러 개의 인접한 디스크 블록에 인접한 희박 청크들을 저장하여 "큰 청크 효과"를 보려는 것이었다. 예를 들어, 그림 7과 8을 보자. 그림 7(a)는 8×8개의 셀로 구성된 2차원 큐브이다. 이를 2×2 셀 단위로 묶어서 청크로 만든 것이 그림 7(b)이다. 청크들은 Z 인덱스로 순서가 부여 되어 있다. 청크 1, 3, 5, 7, 9, 14와 같이 진한 색으로 표시된 청크들은 밀도가 40% 이상인 밀집 청크들이고, 점으로 표시된 청크들은 S 그룹에 속하는 희박 청크들이다. 그림 8은 이와 같은 청크들을 linear 순서, Z 인덱스 순서, 그리고 밀집 청크들과 희박 청크들을 분리한 후



(a) 큐브의 셀 (b) Z 인덱스 번호 매겨진 청크들

그림 7 8×8 큐브의 청크단위 저장(Z 인덱스 순서)



(a) linear 순서로 저장하는 경우 (b) Z 인덱스 순서로 저장하는 경우 (c) Z 인덱스 순서, 청크 밀도를 고려하여 저장하는 경우

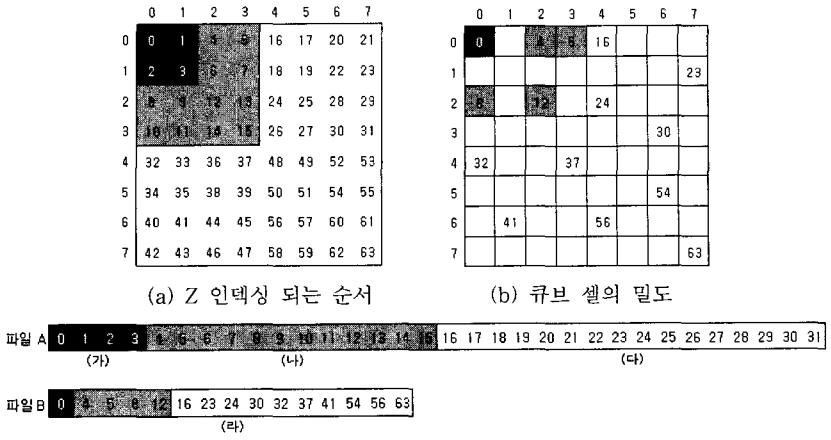
그림 8 2차원 큐브의 다양한 저장 방식이 슬라이스 연산 속도에 미치는 영향

각 그룹을 Z 인덱스 순서로 저장하는 경우를 보여 준다. 계산의 편의를 위해 여기서는 디스크 한 블록이 100개의 셀로 구성되어 있다고 가정하고 밀집 청크는 100셀을 차지하고 희박 청크는 10%에 해당하는 10개의 셀을 차지한다고 가정하였다. 그림 8(a)~8(c)의 각 청크에 표시된 숫자는 그 청크가 디스크에 저장될 곳의 위치를 나타낸다. 예를 들어, 10-119가 나타내는 것은 디스크의 셀 10번부터 119번 사이에 그 청크가 저장된다는 뜻이고, 디스크 블록 1개에 100개의 셀이 포함된다고 가정하였으므로 이는 2개의 블록에 걸쳐서 저장되어 있다는 것을 뜻한다. 즉 100단위의 숫자가 디스크 블록 번호를 나타낸다고 보면 된다.

이와 같이 저장된 큐브에서 1차원 슬라이스 연산을 실행해 보자(그림 8 참조). 0열을 지나는 슬라이스 연산을 하면 linear 순서로 저장한 경우 디스크의 4 블록(0, 2, 4, 5번 블록)이 읽히고, Z 인덱스 순서로 저장한 경우 디스크의 4 블록(0, 1, 4, 5번 블록)이 읽히지만 밀집도와 Z 인덱스를 함께 적용한 경우는 디스크의 1 블록(6번 블록)만이 읽힌다. 3열을 지나는 슬라이스 연산을 하면 linear 순서로 저장한 경우는 블록 6개, Z 인덱스 순서로 저장한 경우는 블록 5개가 읽히고, 밀집도와 Z 인덱스를 함께 고려한 경우는 블록 3개만이 읽힌다. 이 큐브에서 0, 1, 2, 3행과 0, 1, 2, 3열을 각각 지나는 8개의 슬라이스 연산을 하고 이 때 메모리로 읽혀들인 디스크 블록의 수를 계산해 보면, linear 순서로 저장한 경우 블록 30개가 읽히고, Z 인덱스 순서로 저장한 경우 블록 29개가 읽히고, 밀집도와 Z 인덱스를 동시에 적용한 경우는 블록이 20개만 읽힌다는 것을 알 수 있다. 이 데이타는 물론 밀집도와 Z 인덱스를 동시에 적용하는 경우의 효과가 부각되어 나타나도록 만든 데이타이기는 하지만 이를 통해서 희박 청크들을 모아 Z 인덱스 순서로 저장한다면 이 들이 디스크의 한 블록 또는 인접한 블록들에 모여 있기 때문에 "큰 청크 효과"를 볼 수 있다는 점을 파악할 수 있게 된다.

Z 인덱싱을 통한 "큰 청크 효과"를 좀 더 자세히 살펴보기로 한다. 예를 들어, 그림 9를 보자. 그림 9(a)는 64개의 청크로 구성된 큐브를 나타낸다. 그리고, 이들을 Z 인덱스 순서대로 파일에 저장한 것이 데이타 파일 A이다. 청크 0~3, 0~15, 0~63이 파일의 인접한 곳에 순서대로 저장되어 있다는 것을 알 수 있다 (파일 A의 가, 나, 다 부분). 즉, Z 인덱스 순서로 청크들을 저장하면 2×2, 4×4, 8×8 크기의 청크 클러스터들이 디스크의 인접한 블록들에 저장된다는 것을 알 수 있다. 청크들이 매우 희박하여 디스크의 한 블록에 수십 개의 청크가 저장되는 경우는 이와 같은 Z 인덱스의 특징이 OLAP 연산 속도를 향상시키는데 도움을 준다. 이 효과는 그림 9(b)를 데이타 파일 B에 저장하는 경우에 볼 수 있다. 그림 9(b)의 번호있는 청크들은 희박한 청크들이고 번호없는 청크들은 비어있는 (empty chunk) 들이다. 이 경우 2×2, 4×4, 8×8 크기의 청크들이 클러스터를 이루면서 파일 B에 저장되어 있는 것을 볼 수 있다. 이를 보면 밀집 청크들을 따로 저장하는 경우 상당히 많은 희박 청크들을 모아 한 개 또는 인접한 디스크 블록에 모아 놓을 수 있다는 것을 알 수 있다.

청크의 크기는 청크 인덱스 저장 공간에 큰 영향을 준다. 예를 들어, 저밀도 큐브를 저장할 때, 밀집 청크의 크기를 디스크 블록 1개의 크기에 맞춘다면 저밀도 청크들은 10개 이하의 셀을 포함하는 경우가 흔히 발생할 것이고, 청크마다 존재하는 인덱스의 분량이 상대적으로 커진다. Z 인덱싱 순서로 청크들을 저장시키면 이러한 점의 보완에 도움이 된다. 예를 들어, 3차원 큐브를 본 연구에서 제안한 방식대로 파일 A와 B에 각각 저장



(a) Z 인덱싱 되는 순서 (b) 큐브 셀의 밀도

(c) 파일에 저장된 큐브

그림 9 Z 인덱싱을 통한 "큰 청크 효과"

하였다고 하자. 파일 A에는 청크 한 변의 길이를 x 셀로 정해서 저장하였고, 파일 B에는 청크 한 변의 길이를 $2x$ 셀로 정해서 저장하였다고 하자. 그렇다면 파일 B의 한 청크에는 파일 A의 연속한 최대 8개의 청크들이 Z 인덱스 순서로 저장되어 있다. 이 점을 고려하여 파일 A의 희박 청크 인덱스 공간을 줄일 수 있다. 8개의 청크마다 1개의 인덱스를 둘 수 있으며 이 때 각 청크에 속한 셀들의 offset 에는 각 차원마다 1 비트씩 추가하여 청크의 상대적 위치를 나타내면 된다. 청크 각 변의 길이가 m배로 늘어나는 경우 각 변마다 $\log m$비트를 사용하면 된다. 이는 큐브 생성시에 희박 청크들을 압축할 때 함께 하면 되므로 이러한 방법을 구현하는 데에는 추가적인 시간 공간적 오버헤드가 거의 없다.

4. 실험을 통한 큐브 저장구조의 효율성 검증

본 연구에서 제안한 OLAP 큐브 저장 구조가 [1]의 큐브 저장방식에 비하여 OLAP의 대표적인 연산인 슬라이스와 다이스 연산 속도를 얼마나 향상시키는가를 실험을 통해 비교 분석하였다. 각 연산을 실행하였을 때 메모리로 읽혀지는 디스크 블록의 개수와 디스크 헤드 탐색 시간 (seek time)을 성능 측정 기준으로 삼았다. 디스크 I/O 오버헤드는 읽히는 블록 개수 뿐 아니라 읽을 데이타가 디스크 블록들에 불연속적으로 저장되어 있는 정도에도 영향을 받기 때문이다. 헤드 탐색 시간은 연속한 디스크 블록들의 덩어리 숫자로 측정하였다. 큐브는 청크들이 정해진 순서대로 디스크의 인접한 블록들에 저장되었다고 가정하였다. 실험은 시뮬레이션을 통해 이루어졌고, 실험을 통해 분석한 항목들은 다음과 같다:

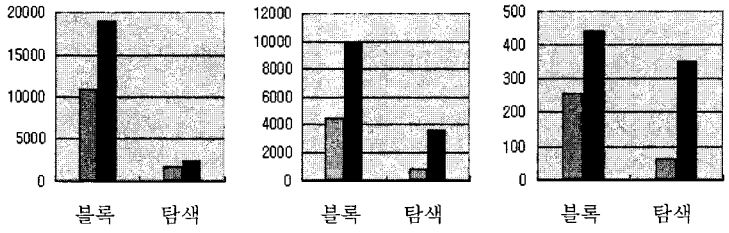
- 희박 청크들을 Z 인덱스 순서로 저장하는 경우의 큰 청크 효과 분석
- 밀집 청크들을 Z 인덱스 순서로 저장하는 경우의 성능 분석
- 클러스터를 이루는 희박 큐브를 Z 인덱스 순서로 저장하는 경우의 큰 청크 효과 분석

(1) 희박 청크들을 Z 인덱스 순서로 저장하는 경우의 큰 청크 효과 분석

희박 청크들을 Z 인덱스 순서로 디스크에 저장하면 인접한 청크들이 디스크 블록 1개에 여러 개씩 저장된다. 이 실험에서는 이러한 구조의 효율성을 검증하기 위해서 표 1과 같은 데이타 세트를 사용하였다. 균일 분포 데이타를 사용하였고 큐브 밀도는 5%로 하였다. 즉, 디스크 블록 1개에 청크가 10개 정도 모여 있는 경우를 실험한 것이다. 그림 10은 슬라이스 연산의 속도를 나타내고 그림 11은 다이스 연산의 속도를 나타낸다. 그림에서 '블록'으로 표시된 막대는 메모리로 읽혀지는 블록의 개수를 나타내고 '탐색'으로 표시된 막대는 디스크 탐색 시간인 인접한 청크들의 덩어리 개수를 뜻한다. 슬라이

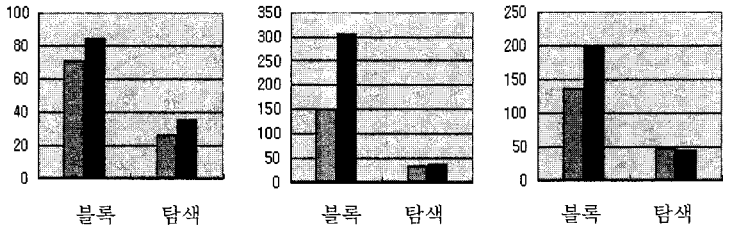
표 1 밀도 5%인 균일 분포 데이타 세트

데이타 세트	큐브 차원 수	각 차원 멤버 수	큐브의 청크 개수	큐브 셀 개수
1.1	3	320	32,768	32,768,000
1.2	4	160	1,048,576	655,360,000
1.3	5	64	1,048,576	1,073,741,824



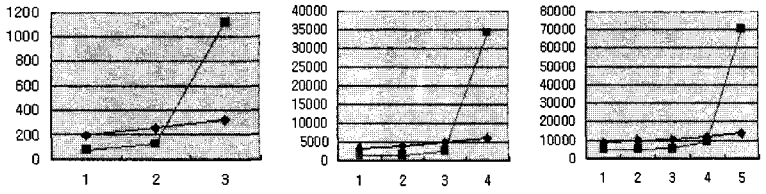
(a) 3차원 (b) 4차원 (c) 5차원

그림 10 데이타 세트 1.1~1.3에서 슬라이스 연산 속도(진한 막대: linear, 연한 막대: Z)



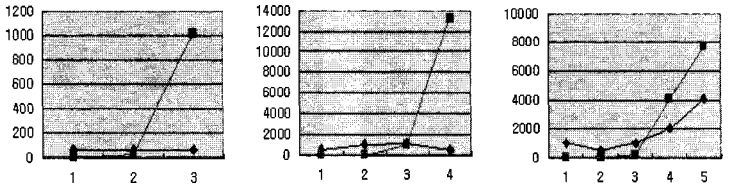
(a) 3차원 (b) 4차원 (c) 5차원

그림 11 데이타 세트 1.1~1.3에서 다이스 연산 속도(진한 막대: linear, 연한 막대: Z)



(a) 3차원 (b) 4차원 (c) 5차원

그림 12 데이타 세트 1.1~1.3에서 슬라이스 연산할 때 읽는 디스크 블록 수(진한 선: Z, 연한 선: linear)



(a) 3차원 (b) 4차원 (c) 5차원

그림 13 데이타 세트 1.1~1.3에서 슬라이스 연산할 때 드는 디스크 탐색 시간(진한 선: Z, 연한 선: linear)

스 연산은 실험 전반에 걸쳐 각 차원별로 무작위로 10개씩 만들었고 차원별 평균 값을 다시 평균 내어 그래프에 표시하였다. 다이스 연산은 차원에 무관하게 10개씩 무작위로 생성하였다. 모든 데이타 세트에 대해 linear 순서보다 Z 인덱스 순서로 저장하는 것이 효율적임을 알 수 있다.

슬라이스 연산은 큐브에서 특정 차원 멤버의 값이 고정된 셀들을 찾는 것이다. 슬라이스 연산의 속도는 그림 2에서와 같이 어떤 차원을 기준으로 실행되는가에 큰 영향을 받는다. 그림 12와 13은 이를 분석한 차트이다. 서로 다른 차원을 기준으로 슬라이스 연산을 했을 때 각각 읽혀지는 디스크 블록의 수를 나타낸 것이 그림 12이고 디

스크 헤드 탐색 시간을 나타낸 것이 그림 13이다. 그림에서 수평축의 레이블 $d, 1 \le d \le 3$, 는 d차원을 기준으로 슬라이스 연산을 했다는 것을 나타낸다. 진한 선으로 표시된 것이 Z 인덱스 순서로 저장된 경우인데 모든 차원에 공평하게 속도를 보장한다는 것을 알 수 있다.

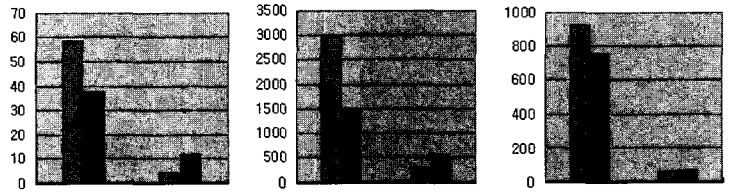
(2) 밀집 청크들을 Z 인덱스 순서로 저장하는 경우의 성능 분석

밀집 청크들은 디스크 블록 1개를 차지한다. 따라서 Z 인덱스 순서로 저장한다고 해도 이들은 디스크 블록 안에서 큰 청크 효과를 주지는 않는다. 또한 Z 인덱스 순서나 linear 순서 중에서 어떤 순서로 저장된다고 해도 읽혀지는 디스크 블록의 수는 같다. 여기서 실험하려는 것은 비즈니스 데이타의 경우와 같이 큐브 자체는 매우 희박하지만 데이타가 클러스터를 이루는 경우이다. 클러스터에는 밀집 청크들이 비교적 많이 포함되어 있다. 이러한 큐브의 밀집 청크들을 따로 모아서 Z 인덱스 순서로 저장하는 것이 얼마나 효율적인가를 점검해 보았다. 실험에 사용한 데이타가 표 2에 있다. 클러스터 내의 청크들의 평균 밀도를 40% 이상으로 하여 그러한 청크들이 배열 형태 그대로 저장되도록 하였다.

표 2 밀집 청크들이 클러스터를 이루는 데이타 세트

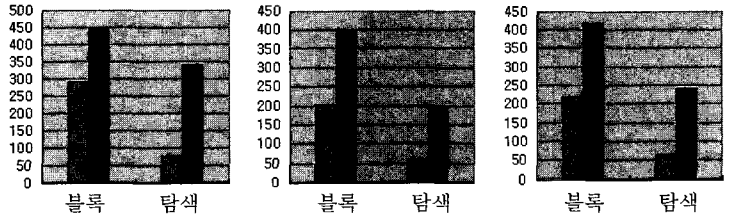
데이타 세트	큐브 차원 수	각 차원 멤버 수	클러스터를 이루는 청크들의 비율	클러스터 내의 청크 밀도	전체 큐브 밀도
2.1	3	320	8%	40%	3%
2.2	4	160	16%	40%	6.4%
2.3	5	64	16%	40%	6.4%

실험결과는 그림 14에 나타나 있다. 슬라이스를 하는 경우는 디스크 탐색 시간이 linear 순서로 저장하는 경



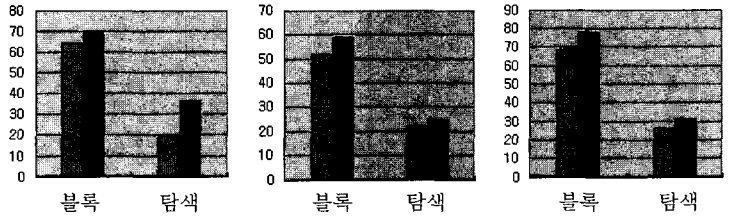
(a) 3차원 (b) 4차원 (c) 5차원

그림 14 데이타 세트 2.1~2.3에서 연산들의 디스크 탐색 시간(연한 막대: Z, 진한 막대: linear)



(a) 10% (b) 20% (c) 30%

그림 15 데이타 세트 3.1~3.3에서 슬라이스 연산 속도(연한 막대: Z, 진한 막대: linear)



(a) 10% (b) 20% (c) 30%

그림 16 데이타 세트 3.1~3.3에서 다이스 연산 속도(연한 막대: Z, 진한 막대: linear)

우보다 비교적 많으나 다이스 연산의 경우는 Z 인덱스 순서가 효율적임을 알 수 있다. 슬라이스 연산의 경우는 그림 2를 예로 들면 *AB* 평면에 수평인 슬라이스를 하는 경우 청크들이 적어도 2개 이상 인접하여 디스크에 저장될 확률이 높고 이로 인해 평균적으로 Z 인덱스 순서로 청크들을 저장하는 것보다 linear 순서로 저장하는 것이 헤드 탐색 시간을 줄인다고 볼 수 있다. 다이스 연산의 경우는 클러스터 모양의 청크들이 메모리로 읽혀 들이고 Z 인덱스 순서가 청크들을 그러한 순서로 저장하는 경향이 있기 때문이라고 본다. 큐브의 데이타를 모든 차원 멤버의 값에 범위를 주어 잘라내는 것이 다이스 연산이고 이러한 연산은 OLAP을 할 때 사용 빈도가 매우 높기 때문에 밀집 청크들을 저장하는 경우에도 Z 인덱스 순서를 사용하는 것이 효율적이라고 본다.

(3) 클러스터를 이루는 희박 큐브를 Z 인덱스 순서로 저장할 때의 큰 청크 효과 분석

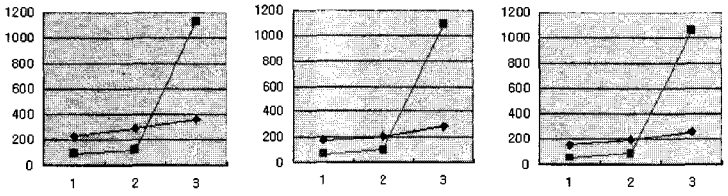
이 실험에서는 다양한 크기의 희박 청크들이 Z 인덱스 순서로 저장되어 있는 경우 어떠한 성능을 보이는가를 분석하였다. 실험에는 표 3과 같은 9개의 데이타 세트를 사용하였다. 비즈니스 데이타와 유사하도록 하기 위해 큐브의 밀도는 낮게 정하였다. 데이타는 클러스터를 이루도록 하였고 클러스터내의 청크 밀도를 10%, 20%, 30%로 변화해 가면서 슬라이스와 다이스 연산의 속도를 측정하였다. 클러스터의 크기와 위치 역시 무작위로 정하였다.

3차원 큐브 상에서의 연산 속도가 그림 15~18에 나타나 있다. 그림 15와 16은 각각 데이타 세트 3.1~3.3 상에서 슬라이스와 다이스를 실행하였을 때 읽는 디스크 블록 수와 헤드 탐색 시간을 나타낸다. 일반적으로 Z 인덱스 순서가 linear 순서보다 효율적임을 나타낸다.

그림 17~18은 슬라이스 연산을 서로 다른 차원을 기준으로 실행한 결과를 보인다. Z 인덱스 순서로 큐브가 저장된 경우 모든 차원에 공평한 속도를 보장한다는 것을 알 수 있다. 데이타 세트 3.4~3.9 에 대해서도 비슷한 추세를 보인다. 그림 19~22는 데이타 세트 3.4~3.9로 실험한 결과이다. 모든 경우에 Z 인덱스 순서로 청크들을 저장하는 것이 linear 순서로 저장하는 것보다 연산 속도를 크게 향상시킨다는 것을 알 수 있다.

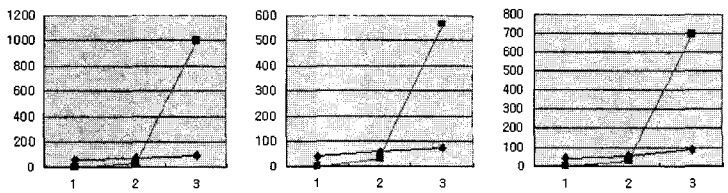
표 3 데이타가 클러스터를 이루는 희박 큐브 데이타

데이타 세트		각 차원 멤버 수	클러스터에 포함된 청크들의 비율	클러스터 내의 청크 밀도	전체 큐브 밀도
3.1	3차원	320	8%	10%	2.8%
3.2				20%	3.6%
3.3				30%	4.4%
3.4	4차원	160	16%	10%	3.6%
3.5				20%	5.2%
3.6				30%	6.8%
3.7	5차원	64	16%	10%	3.6%
3.8				20%	5.2%
3.9				30%	6.8%



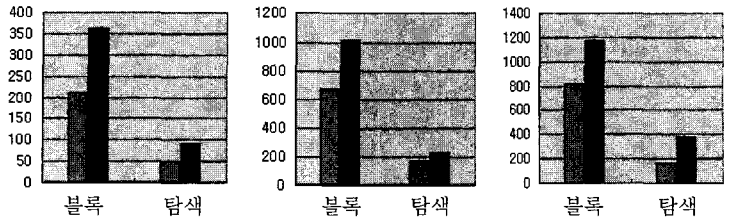
(a) 10% (b) 20% (c) 30%

그림 17 데이타 세트 3.1~3.3에서 슬라이스 연산할 때 읽는 디스크 블록 수(진한 선: Z, 연한 선: linear)



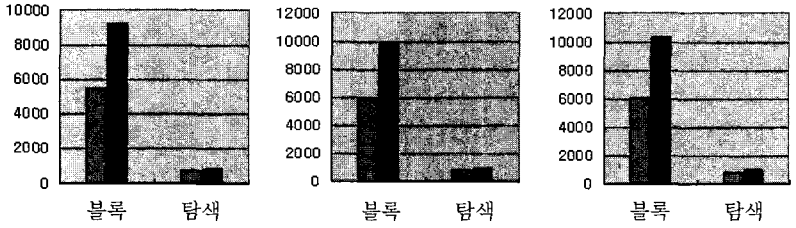
(a) 10% (b) 20% (c) 30%

그림 18 데이타 세트 3.1~3.3에서 슬라이스 연산할 때 드는 탐색 시간(진한 선: Z, 연한 선:lienar)



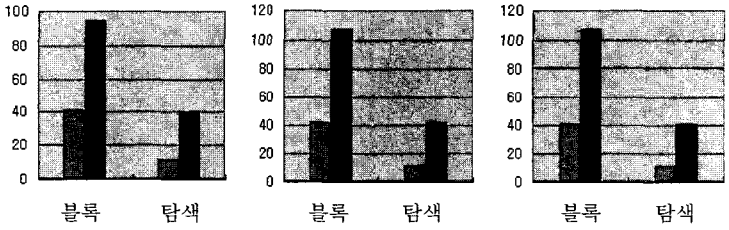
(a) 10% (b) 20% (c) 30%

그림 20 데이타 세트 3.4~3.6에서 다이스 연산 속도(연한 막대: Z, 진한 막대: linear)



(a) 10% (b) 20% (c) 30%

그림 21 데이타 세트 3.7~3.9에서 슬라이스 연산 속도(연한 막대: Z, 진한 막대: linear)



(a) 10% (b) 20% (c) 30%

그림 22 데이타 세트 3.7~3.9에서 다이스 연산 속도(연한 막대: Z, 진한 막대: linear)

5. 결 론

OLAP 큐브의 저장 방식은 OLAP 연산 속도에 크게 영향을 미친다. 본 연구에서는 청크 기반의 MOLAP 큐브를 효율적으로 저장하는 방법을 제시하고, 실험을 통해 이 방법이 주요 OLAP 연산인 슬라이스와 다이스 연산 속도를 향상시킨다는 것을 입증하였다. 이 방법은 청크들을 밀집도 기준으로 분리하여 인접한 희박 청크들을 하나의 디스크 블록에 가능한 한 많이 모이도록 함으로써 디스크 I/O를 줄이는 데 초점을 맞추고 있다. 인접 청크들을 클러스터링하는 데는 Z 인덱싱이 사용되었다. 제안한 방법이 대부분의 경우 연산 속도를 크게 향상시키고, 큐브의 모든 차원에 공평하게 슬라이스 연산의 성능을 보장한다는 것을 보였다. 본 연구에서는 또한 희박 청크들이 많은 경우에 인덱스 공간을 줄이는 간단한 방안을 제시하였다.

참 고 문 헌

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

[2] Won Kim and Myung Kim, "Performance and Scalability in Knowledge Engineering: Issues and Solutions," *Journal of Object-Oriented Programming*, Vol. 12, No. 7, pp. 39-43, Nov/Dec. 1999.

[3] Pilot Software, "An Introduction to OLAP: Multidimensional Terminology and Technology," http://www.pilotsw.com/olap/olap.htm.

[4] Erik Thomsen, *OLAP Solutions: Building Multidimensional Information Systems*, John Wiley

& Sons, New York, 1997.
[5] OLAP Counsil, http://www.olapreport.com/DatabaseExplosion.htm.
[6] Oracle Corp., "Sparsity Management System for Multi-dimensional Databases," *United States Patent* 5,943,677 , Aug. 24, 1999.
[7] Sunita Sarawagi and Michael Stonebraker, "Efficient Organization of Large Multidimensional Arrays," *Proc. of 1994 Data Engineering Conference*, Feb. 1994.
[8] Kenneth C. Sevcik and Nikos Koudas, "Filter Trees for Managing Spatial Data over a Range of Size Granularities," *Proc of the 1996 International Conference on Very Large Databases (VLDB)*, pp.16-27, 1996.
[9] Myung Kim and Jisook Song, "Efficient Summary Table Generation for ROLAP," *Ewha Institute of Science and Technology, EIST TR-2001-01*, 2001.
[10] H. V. Jagadish, Laks V. S. Lakshmanan, and Divesh Srivastava, "Snakes and Sandwiches: Optimal Clustering Strategies for a Data Warehouse," *Proc. of the 1999 ACM-SIGMOD Conference*, pp. 37-48, 1999.
[11] Prasad M. Deshpande, Karthikeyan Ramasamy, Amit Shukla and Jeffrey F. Naughton, "Caching multidimensional queries using chunks," *Proc. of the 1999 ACM-SIGMOD Conference*, pp. 259-270, 1999.
[12] Arbor Software Corporation, "Method and Apparatus for Storing and Retrieving Multi-dimensional Data in Computer Memory," *United States Patent* 5,359,724, Oct. 25, 1994.
[13] Hanan Samet, *Application of Spatial Data Structures-Computer Graphics, Image Processing, and GIS*, Addison Wesley, 1990.

김 명
정보과학회논문지 : 데이타베이스
제 29 권 제 2 호 참조

임 윤 선
1987년 중앙대학교 수학과 학사. 1992년 ~ 현재 (주)아리스트 개발이사. 2000년 ~ 현재 이화여자대학교 컴퓨터학과 석사과정. 관심분야는 OLAP, Data Mining, Knowledge Management