중첩된-서브큐브: 전위-합 큐브를 위한 손실 없는 압축 방법

(Overlapped-Subcube: A Lossless Compression Method for Prefix-Sum Cubes)

강 흠 근 † 민 준 기 † 전 석 주 †† 정 진 완 †††
(Heum-Geun Kang) (Jun-Ki Min) (Seok-Ju Chun) (Chin-Wan Chung)

요 약 영역 질의는 의사결정에서 자주 사용되는 중요한 질의이다. 그러나, 영역 질의를 처리하기 위해서는 많은 점(cell)들이 검색되어야 하기 때문에 효율적인 처리가 쉽지 않았다. 이러한 문제를 해결하기 위해서 영역의 크기에 관계없이 일정한 시간에 영역 질의를 처리할 수 있는 전위-합 큐브(prefix-sum cube)가 제안되었다. 그러나, 전위-합 큐브는 영역 질의의 처리는 효율적으로 할 수 있지만, 그것을 저장하기 위해 매우 큰 저장 공간이 필요하다는 문제를 갖고 있다. 본 논문에서는 전위-합 큐브의 이 문제를 해결하기 위해서 손실 없이 전위-합 큐브를 압축하는 중첩된-서브큐브 압축 방법을 제안한다. 중첩된-서브큐브 압축 방법은 전위-합 큐브의 압축을 위해서 만들어진 것으로 압축된 상태에서 저장된 값을 검색할 수 있는 매우 유용한 특징이 있다. 이 특징으로 인해, 질의 처리 시 압축된 전위-합 큐브를 그대로 사용할 수 있다. 압축된 전위-합 큐브를 사용하면, 동일한 크기의 버퍼에 전위-합 큐브의 더 많은 부분을 저장할 수 있다. 이것은 질의 처리 시 디스크 입출력의 횟수를 획기적으로 감소시킨다.

키워드 : 온라인 분석처리, 영역-합 질의, 압축 방법, 전위-합 큐브

Abstract A range-sum query is very popular and becomes important in finding trends and in discovering relationships between attributes in diverse database applications. It sums over the selected cells of an OLAP data cube where target cells are decided by specified query ranges. The direct method to access the data cube itself forces too many cells to be accessed, therefore it incurs severe overheads. The prefix-sum cube was proposed for the efficient processing of range-sum queries in OLAP environments. However, the prefix-sum cube has been criticized due to its space requirement. In this paper, we propose a lossless compression method called the overlapped-subcube that is developed for the purpose of compressing prefix-sum cubes. A distinguished feature of the overlapped-subcube is that searches can be done without decompressing. The overlapped-subcube reduces the space requirement for storing prefix-sum cubes, and improves the query performance.

Key words : OLAP, range-sum query, compression method, prefix-sum cube

1. 서 론

온라인 분석처리(On-Line Analytical Processing: OLAP)는 축적된 많은 데이타를 쉽게 분석할 수 있도록 하여 사용자가 빠르고 정확한 의사결정을 할 수 있도록 도움을 주는 것을 목적으로 한다[1-3]. OLAP 데이타의 특징으로는 데이타 크기의 방대함과 희박성(sparseness)에 있다[4,5]. OLAP에서는 흔히 데이타 큐브(data cube)로 알려진 데이타 모델을 사용한다[1,6,7]. 데이타 큐브는 가능한 모든 조합에 대해서 미리 집계(aggregation)할 수 있도록 지원한다. 모든 가능한 조합에 대해서 집계를 하기 때문에 데이타 큐브의 크기는 원본 데이타에 비해서 훨씬 커지는 것이 보통이다. 수백 기가 바이트(gigabyte)의 데이타는 흔히 볼 수 있다[8]. 데이타 큐브의 다차원 공간에서 많은 부분은 저장된 데이타가 없다. 보통 데이타 큐브의 모든 점들 중 1~2%만이

· 본 연구는 정보통신부의 대학IT연구센터(ITRC) 지원을 받아 수행되었습니다.
† 비 회 원 : 한국과학기술원 전산학과
hgkang@islab.kaist.ac.kr
jkmin@islab.kaist.ac.kr
†† 정 회 원 : 안산1대학 인터넷정보과 교수
chunsj@mail.ansan.ac.kr
††† 종신회원 : 한국과학기술원 전산학과 교수
chungcw@cs.kaist.ac.kr
논문접수 : 2002년 11월 1일
심사완료 : 2003년 7월 30일

값을 갖는다[4]. 심한 경우는 모든 점들 중에서 0.0001% 만이 값이 갖는 경우도 있다[9].

OLAP은 데이타를 관계형 시스템에 저장하는 관계형 OLAP(Relational OLAP: ROLAP)과 데이타를 다차원 배열에 저장하는 다차원 OLAP(Multidimensional OLAP: MOLAP)으로 구분할 수 있다[10,11]. ROLAP에서는 존재하는 데이타만을 테이블에 튜플로 저장하기 때문에 희박성은 문제가 되지 않는다[12,13]. 그러나, MOLAP에서는 많은 부분이 비워있을 경우, 많은 공간을 낭비하게 된다. 그래서, MOLAP에서는 존재하는 데이타만을 저장하여 데이타의 희박성으로 인해 발생하는 공간의 낭비를 막는다.

영역 질의(range query)는 OLAP에서 중요한 질의 중의 하나이다. 예를 들어서, 자동차 판매 정보를 저장하고 있는 데이타 큐브가 있다고 하자. 그 데이타 큐브는 "자동차 종류", "년도", "지역"의 차원으로, 자동차의 판매 매출액을 저장하고 있다고 하자. 이러한 경우 "2000년부터 2002년 사이에 서울지역에서의 승용차 판매 총액은 얼마인가?"는 그 데이타 큐브로 처리할 수 있는 영역 질의의 한 예이다.

본 논문에서는 영역 질의(range query) 중에서 영역 합 질의를 갖고 논의를 한다. 영역 질의의 처리는 질의의 영역에 포함된 모든 좌표를 검색해야 한다. 따라서, 영역의 크기가 커짐에 따라 질의의 처리에 필요한 시간도 증가한다. OLAP의 사용자는 보통 대화식으로 분석 처리를 한다. 따라서, 사용자의 원활한 의사결정을 위해서, 질의는 빠른 시간 안에 처리되어야 한다. 영역 질의의 효율적인 처리는 중요한 연구 과제이다.

전위-합 큐브(prefix-sum cube)는 영역 질의의 효율적인 처리를 위해서 제안이 되었으며 많은 주목을 받았다[6]. 전위-합 큐브는 모든 좌표에 대해서 큐브의 시작부터 그 곳까지의 합을 미리 계산하여 저장한다. 전위-합 큐브를 이용하면 영역의 크기에 관계없이 2^d(d: 차원의 수) 번의 디스크 액세스로 영역 질의를 처리할 수 있다. 그러나, 전위-합 큐브는 매우 촘촘하다고 비판을 받았다[14,15]. 매우 희박성이 높은 데이타 큐브를 사용하여 만들어진 전위-합 큐브도 모든 공간에 데이타가 존재하게 된다. 데이타 큐브의 경우에는 존재하는 데이타만을 저장하여 저장 공간을 많이 절약할 수 있지만 전위-합 큐브의 경우에는 데이타가 희박하지 않으므로 데이타 큐브에서 사용하는 방법을 이용할 수는 없다. 따라서, 전위-합 큐브를 저장하는 것은 큰 문제가 될 수 있다.

전위-합 큐브 압축에 관한 연구로는 [16]과 같은 손실 압축에 관한 것은 있었지만, 손실 없는 압축에 관한 연구는 없었다. 이에, 본 논문에서는 손실 없이 전위-합 큐브를 압축하는 방법을 제안한다. 제안된 방법은 중첩된-서브큐브(overlapped-subcube) 압축 방법이라고 이름한다. 중첩된-서브큐브 압축 방법은 압축된 상태에서 저장된 값을 검색할 수 있는 특징이 있다. 이로 인해, 중첩된-서브큐브 압축 방법을 사용하면 압축된 데이타를 검색하기 위해서 압축을 풀 필요가 없다. 따라서, 데이타의 저장을 위한 디스크 공간을 절약할 수 있으며, 또한 적은 메인 메모리로도 많은 데이타를 저장할 수 있다. 이러한 특징은, 실험에 의해서 확인되었듯이, 질의 처리의 성능을 획기적으로 높일 수 있다.

본 논문의 구성은 다음과 같다. 제2절에서는 관련 연구로 전위-합 큐브와 기존의 데이타 큐브 압축 방법들을 간략히 소개한다. 제3절에서는 중첩된-서브큐브 압축 방법의 개념을 설명하고, 압축 알고리즘과 검색 알고리즘을 소개한다. 제4절에서는 실험 환경 및 실험 결과를 제시한다. 제5절에서는 결론을 내린다.

2. 관련 연구

2.1 전위-합 큐브

전위-합 큐브는 영역 질의를 효율적으로 처리하기 위해서 제안된 것으로써 부분 합들을 저장한다[2]. 전위-합 큐브는 데이타 큐브의 각 점들에 대해서, 데이타 큐브의 시작에서부터 그 점까지의 합을 미리 계산하여 저장한다. 그림 1은 데이타 큐브의 예와 그 데이타 큐브로 생성한 전위-합 큐브를 보여 준다. 그림 1의 데이타 큐브는 일부의 점에만 값이 있음을 보여 준다. PC의 한 점인 PC[2,2]에는 A[0,0]부터 A[2,2]까지의 점들에 저장된 값들을 모두 합한 값을 저장한다. A에 저장된 모든 값들을 더한 합계는 PC의 마지막 점인 PC[4,4]에 저장되어 있다.

그림 2는 2차원 전위-합 큐브를 이용하여 영역 질의를 처리하는 과정을 보여주고 있다. 하나의 영역 질의를 4개의 부분 합을 사용하여 계산하는 과정을 도형으로 표시하였다.

index	0	1	2	3	4
0			4		
1					
2		3			
3				6	
4					

(가) 데이터 큐브 A

index	0	1	2	3	4
0	0	0	4	4	4
1	0	0	4	4	4
2	0	3	7	7	7
3	0	3	7	13	13
4	0	3	7	13	13

(나) 전방-합 큐브 PC

그림 1 데이타 큐브와 전위-합 큐브의 예

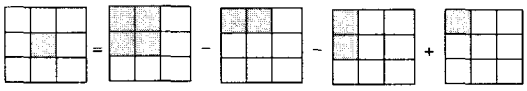

그림 2 2차원 전위-합 큐브에서 영역-합 질의 처리과정

2.2 손실 없는 데이타 큐브 압축 방법

MOLAP에서는 데이타 큐브를 다차원 배열로 저장한다. 일반적으로 데이타 큐브의 많은 점들은 비어있고 일부에만 값이 저장된다. 따라서, 저장할 값은 없으면서 공간만 차지하는 점들이 많다. 이러한 문제를 해결하기 위해서 MOLAP에서는 값이 있는 점만을 저장하는 방법을 사용한다.

2.2.1 Chunk-offset 압축 방법

Chunk는 대용량의 다차원 배열을 효율적으로 저장하기 위해서 제안된 개념이다[17]. 다차원 배열을 디스크(disk)에 저장하는 일반적인 방법은 행 우선(row-major) 순서 또는 열 우선(column-major) 순서를 이용하는 것이다. 그러나, 대용량의 다차원 배열을 행 우선이나 열 우선 순서로 저장하면 배열에서 이웃에 위치한 점들이 실제로 저장될 때에는 멀리 떨어지는 현상이 발생한다. 이러한 문제를 해결하기 위해서 제안된 방법이 chunk 파일이다. Chunk 파일에서는 다차원 배열을 작은 다차원 배열인 chunk로 나누어서 저장하며, 검색 및 저장의 단위는 chunk가 된다. 그림 3은 chunk 파일의 예이다. 그림 3의 (가)는 원래의 배열을 나타내며 (나)는 chunk로 나뉘어진 것을 나타낸다. 그림 3의 (나)에 있는 chunk는 하나가 9개의 점을 포함한다. 보통, chunk의 크기는 한 페이지의 크기가 되도록 정한다.

하나의 chunk에 포함된 점들 중에는 값이 있는 것이 있고 값이 없는 것이 있다. Chunk-offset 압축 방법은 값이 있는 점만을 저장한다[18]. 이때, 각 점들은 chunk의 시작에서 그 점까지의 거리(offset)와 그 점에 저장된 값이 쌍으로 저장된다. 예를 들어 그림 3의 (가)에서 '*'가 표시된 점의 좌표는 (8, 5)이다. 그 점은 chunk 6에 저장된다. Chunk 6의 시작에서 점(8,5)까지의 거리를 행 우선으로 하면 다음의 식으로 계산할 수 있다.

(1) (5%chunk_y_size)(chunk_x_size + (8%chunk_x_size)

위의 식에서 "%"는 나머지를 구하는 연산자이다. chunk_x_size는 한 chunk에서 x축의 크기를 나타내며, chunk_y_size는 y축의 크기를 나타낸다.

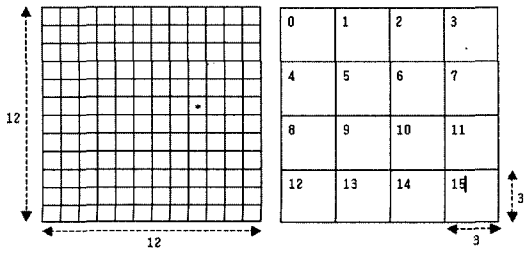

그림 3 chunk 파일의 예

2.2.2 header 압축 방법

header 압축 방법은 1차원 배열에서 필요한 값만을 저장하는 방법을 제공한다[19]. 다차원 배열인 경우는 1차원으로 변경한 다음 압축한다. 이 방법에서는 정수의 벡터인 header를 이용한다. Header에서 홀수 위치에 있는 정수는 저장할 필요가 있는 연속된 값들의 개수를 나타내고, 짝수 위치에 있는 정수는 저장할 필요가 없는 연속된 값들의 개수를 나타낸다. Header에 있는 각 정수는 누적된 값이다. 물리적 파일(physical file)은 원래의 파일(original file)에서 저장할 필요가 있는 값만을 선택하여 저장한다.

original file : 4 0 0 0 0 7 3 0 0 0 0 0 0 9
header : 1 4 3 10 4
physical file : 4 7 3 9

그림 4 header 압축의 예

그림 4는 header 압축의 예를 보여주고 있다. 원래의 파일을 입력으로 하여 헤더와 물리적 파일로 압축을 하였다. 이 예에서는 원래의 파일에 있는 반복되는 0을 제거하여 압축하는 것을 보여주고 있다. header의 홀수 위치에 있는 1, 3, 4는 원래의 파일에서 물리적 파일에 저장될 값의 개수를 누적하여 나타낸다. 짝수 위치에 있는 4, 10은 저장하지 않을 값의 개수를 누적하여 나타낸다. Header의 마지막 홀수 위치에 있는 4는 저장할 필요가 있는 값의 총 개수이다. 마지막 짝수 위치에 있는 10은 저장할 필요가 없는 값의 총 개수이다. 물리적 파일은 원래의 파일에 저장된 값들 중 0을 제외한 나머지를 저장한다.

2.3 Space-efficient Relative Prefix-Sums (SRPS) 방법

전위-합 큐브는 영역-합 질의를 영역의 크기에 관계없이 매우 효율적으로 처리할 수 있다. 그러나, 데이타 큐브에 있는 하나의 점에 저장된 값이 변경될 때, 그 데이타 큐브로 만든 전위-합 큐브의 무수히 많은 점들의 전위-합들이 새로이 계산되어야 하는 문제를 갖고 있다.

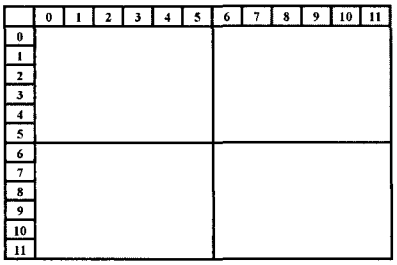

그림 5 SRPS 큐브의 예

이러한 문제를 해결하기 위해서 SRPS 방법이 제안되었다. 이 방법은 전위-합 큐브를 동일한 크기의 박스(box)들로 분할하여 새로 계산해야 되는 전위-합의 개수를 제한하는 방법이다. 그림 5는 4개의 박스로 나누어진 SPRS 큐브의 예이다.

3. 중첩된-서브큐브 압축 방법

본 절에서는 먼저 중첩된-서브큐브 압축 방법의 기본 개념을 설명한다. 이를 위해서, 서브큐브의 정의를 소개하고 중첩된-서브큐브 압축 방법을 이용하여 전위-합 큐브를 압축하는 원리를 설명한다. 그리고, 전위-합 큐브를 압축하는 효율적인 알고리즘을 소개한다. 또한, 압축을 풀지 않고 직접 압축된 큐브를 검색하는 알고리즘을 소개한다. 본 논문에서는 명료한 설명을 위해서 이차원의 큐브를 대상으로 하여 모든 논의를 진행한다.

3.1 기본 개념

중첩된-서브큐브 압축 방법은 전위-합 큐브를 서브큐브들의 리스트로 바꾸는 것이다. 서브큐브들은 리스트에서 앞에 있는 서브큐브를 포괄한다.

정의 1 서브큐브는 전위-합 큐브에 포함된 작은 큐브이다. 서브큐브는 다음과 같은 특징을 갖는다.

(1) 서브큐브의 차원의 수는 그 서브큐브를 포함하는 전위-합 큐브의 차원의 수와 같다.

(2) 하나의 서브큐브에 속한 모든 점들은 같은 값을 갖는다. 그 값을 $V_{subcube}$라고 한다.

(3) 서브큐브의 첫 점의 주소를 서브큐브의 첫 주소라고 한다. 서브큐브의 마지막 점의 주소를 서브큐브의 마지막 주소라고 한다. □

일반적으로 데이타 큐브는 값을 갖고 있는 점의 수가 적기 때문에, 그러한 데이타 큐브로부터 만들어진 전위-합 큐브는 같은 값이 반복하는 경향이 있다. 중첩된-서브큐브 압축 방법에서는 그 반복되는 값을 서브큐브로 표현한다. 서브큐브의 점들은 모두 같은 값을 갖기 때문에 각 점들이 갖는 값을 따로 저장할 필요가 없다. 그림 6에 표시된 다섯 개의 서브큐브들은 그림 1의 (나)에

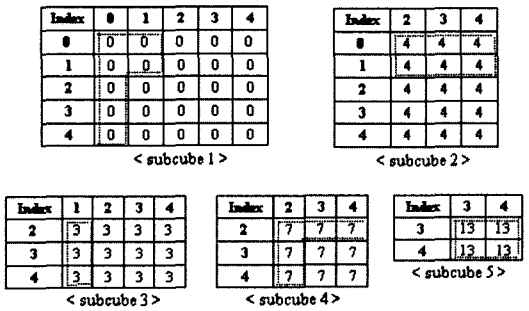

그림 6 서브큐브들

표 1 서브큐브의 간략한 표현

서브큐브	Vsubcube	첫 주소	마지막 주소
1	0	(0, 0)	(4, 4)
2	4	(2, 0)	(4, 4)
3	3	(1, 2)	(4, 4)
4	7	(2, 2)	(4, 4)
5	13	(3, 3)	(4, 4)

표시된 전위-합 큐브로부터 추출된 것이다. 다섯 개의 서브큐브들은 각기 하나의 값만을 저장하고 있다. 서브큐브 1의 점들은 0만을 저장하며 서브큐브 2는 4만을 저장한다. 그림 6의 모든 서브큐브들은 첫 주소는 다르지만 마지막 주소는 같다. 그러므로, 하나의 서브큐브는 첫 주소, 마지막 주소, 그리고 $V_{subcube}$로 표현될 수 있다. 표 1은 그림 6의 서브큐브들을 간략히 표현한 것이다.

정의 2 모든 서브큐브는 하나의 비공유 영역(private region)을 갖는다. 서브큐브의 비공유 영역은 서브큐브의 리스트에서 뒤에 위치한 다른 서브큐브에 의해서 포개지지 않는 부분이다. □

그림 6에서 점선으로 표시된 부분이 각 서브큐브의 비공유 영역이다. 서브큐브 1을 직접적으로 포개는 서브큐브는 2와 3이다. 따라서, 서브큐브 1의 비공유 영역은 서브큐브 2와 3에 의해서 포개지지 않는 영역이다. 서브큐브 5는 서브큐브들의 리스트에서 마지막에 위치함으로 전체가 비공유 영역이다. 비공유 영역은 압축된 큐브에서 검색할 때 사용된다.

중첩된-서브큐브 압축 방법에서의 압축 결과는 서브큐브들의 리스트이다. 이 때 서브큐브들의 순서는 전위-합 큐브의 시작에서부터 서브큐브의 첫 주소까지의 거리로 결정된다. 거리가 작은 서브큐브는 앞에, 거리가 큰 서브큐브는 뒤에 위치한다. 두 점의 거리는 큐브를 1차원으로 저장할 때의 거리를 말한다. 예를 들어, 전위-합 큐브의 시작에서 (x, y)까지의 거리는 $x + y \times x$축의 크기로 구할 수 있다. 표 2는 전위-합 큐브의 시작에서부터 그림 5의 서브큐브들의 첫 주소까지의 거리이다.

표 1에서 모든 서브큐브들은 전위-합 큐브와 같은 마지막 주소를 갖기 때문에 이를 저장할 필요가 없다. 따라서, 하나의 서브큐브를 위해서 첫 주소와 Vsubcube 값만을 저장하면 된다. 첫 주소도 주소를 구성하는 각 차원의 값을 따로 저장하지 않고 그 주소의 거리를 저장한다. 그림 7은 그림 1의 (나)에 표시된 전위-합 큐브

표 2 전위-합 큐브로부터 서브큐브까지의 거리

서브큐브	1	2	3	4	5
첫 주소	(0, 0)	(2, 0)	(1, 2)	(2, 2)	(3, 3)
거 리	0	2	11	12	18

(0, 0), (2, 4), (11, 3), (12, 7), (16, 13)

그림 7 - 압축된 결과

를 압축한 결과이다.

3.2 압축 알고리즘

그림 8이 보여주고 있는 것은 2차원 전위-합 큐브를 압축하는 중첩된-서브큐브 압축 알고리즘이다. 이 알고리즘은 2차원 전위-합 큐브를 입력으로 받아서 서브큐브 리스트를 리턴한다. 이 알고리즘은 전위-합 큐브를 첫 주소에서부터 마지막 주소까지 모든 주소를 한 번 스캔한다. 스캔하면서, 모든 주소마다 새로운 서브큐브를 생성할지 검사한다. 각 주소에 저장된 값이 첫 주소 방향으로 인접한 주소에 저장된 값들과 다르면 새로운 서브큐브를 생성한다.

```
Algorithm Overlapped-subcube compression
Input : prefix_cube
Output : compressed prefix-sum cube
Begin
1.  for ( i = 0 ; i < X_SIZE ; i++ )
2.    for ( j = 0 ; j < Y_SIZE ; j++ )
3.      if ( ( i = 0 and j = 0 ) or
4.           ( i = 0 and prefix_cube [i][j] ≠ prefix_cube [i][j-1] ) or
5.           ( j = 0 and prefix_cube [i][j] ≠ prefix_cube [i-1][j] ) or
6.           ( prefix_cube [1][j] ≠ prefix_cube [i-1][j] and
7.                prefix_cube [i][j] ≠ prefix_cube [i][j-1] ) ) {
8.          make a pair with the value prefix_cube [i][j] and the offset
9.          append the pair into the result
10.     }
End.
```

그림 8 압축 알고리즘

3.3 검색 알고리즘

본 절에서는 검색 알고리즘을 제시한다. 검색 알고리즘은 서브큐브의 리스트 형태인 압축된 큐브와 검색하려는 주소를 입력으로 받는다. 검색 알고리즘에서는 압축을 풀지않고 직접 압축된 큐브에서 특정 주소의 값을 검색하여 리턴한다. 압축된 전위-합 큐브에서의 검색은 다음의 두 단계로 이루어진다. 첫 단계는 검색하려는 주소를 포함하는 비공유 영역을 갖고 있는 서브큐브를 찾는 것이다. 다음 단계는 그 서브큐브의 $V_{subcube}$ 값을 리턴한다. 위의 두 단계에 의해서 올바른 값을 검색할 수 있다는 것은 쉽게 알 수 있다.

비공유 영역은 서브큐브의 영역에서 다른 서브큐브에 의해서 포개지지 않는 부분이다. 압축된 큐브에서는 비공유 영역을 따로 저장하지 않는다. 따라서, 주어진 주소를 포함하는 비공유 영역을 찾으려면, 서브큐브들을 하나씩 검사하여 다른 서브큐브에 의해서 포개지지 않는 부분이 그 주소를 포함하는지 조사하여야 한다. 이 방법은 많은 계산을 필요로 하므로 좀 더 효율적인 다른 방법을 제시한다. 보조정리 1에 따라, 검색 알고리즘은 서브큐브들의 비공유 영역을 계산하지 않고 주어진 주소를 포함하는 서브큐브들 중에서 가장 큰 첫 주소를 갖는 서브큐브를 찾는다.

보조정리 1 만일 점 c1이 서브큐브 S1의 비공유 영역에 포함된다면, S1은 c1를 포함하는 서브큐브들 중에서 가장 큰 첫 주소를 갖는다.

증명: c1이 S1의 비공유 영역에 속해있다고 하자. 만일 S1의 첫 주소가 c1를 포함하는 서브큐브들 중에서 가장 크지 않다면, c1를 포함하는 서브큐브들 중에서 가장 큰 첫 주소를 갖고 있는 S2이 존재한다. S2의 첫 주소가 S1보다 크므로 S2는 S1를 덮는다. 따라서, S1의 c1은 S2의 c1에 의해서 포개진다. 그러므로, S1의 비공유 영역은 c1를 포함하지 못한다. 이것은 가정에 위배된다. 따라서, c1이 S1의 비공유 영역에 속해 있다면, S1은 c1을 포함하는 서브큐브들 중에서 가정 큰 첫 주소를 갖는다. □

그림 9는 압축된 큐브에서 한 점을 검색하는 검색 알고리즘을 보이고 있다. 서브큐브들은 첫 주소의 크기 순서대로 리스트에 위치한다. 그러므로, 검색하려는 주소를 포함하는 서브큐브들 중에서 리스트의 가장 뒤에 위치한 것을 찾으면 된다. 따라서, 리스트의 뒤에서부터 검색하려는 주소를 포함하는지 검사하여 처음으로 그 주소를 포함하는 서브큐브를 찾으면 된다. 이 알고리즘에서는 모든 서브큐브를 검사하는 대신에 먼저 검사할 서브큐브들의 범위를 좁힌 다음에, 좁혀진 범위에 포함된 서브큐브들만 하나씩 조사한다.

```
Algorithm Lookup
Input : compressed_cube, x, y
Output : V_subcube
Begin
1.  low = 1
2.  high = the number of pairs in the compressed_cube
3.  while ((low + 1) < high) {
4.     mid = (low + high) / 2
5.     (x1, y1) = the first address of mid's pair in the compressed_cube
6.     if (x ≤ x1 and y ≤ y1)
7.        high = mid
8.     else if (x1 ≤ x and y1 ≤ y)
9.        low = mid
10.    else
11.       break ;
12. }
13. for (i = high; low ≤ i ; i--) {
14.    (x1, y1) = the first address of ith pair in the compressed_cube
15.    if (x1 ≤ x and y1 ≤ y)
16.       return V_subcube of the ith pair in the compressed_cube
17. }
End.
```

그림 9 - 검색 알고리즘

4. 실 험

본 절에서는 중첩된-서브큐브 압축 방법의 효과를 측정하기 위해서 수행된 실험의 결과를 보인다. 실험에 사용된 데이타 큐브는 3차원으로 구성되었으며 무작위로

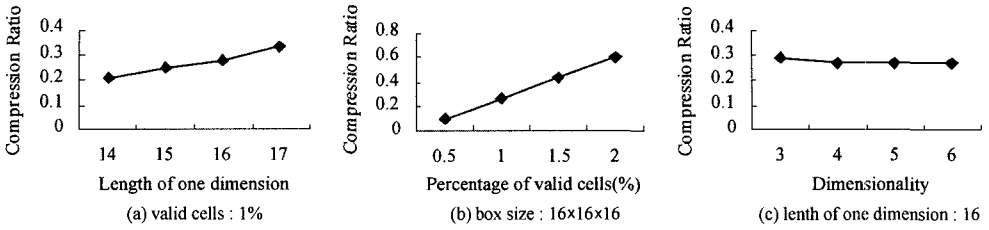

그림 10 압축 비율

선택된 일부의 점에 값을 부여하였다. 생성된 데이타 큐브를 이용하여 전위-합 큐브를 계산하였다. 이 전위-합 큐브를, [20]에서 제안한 SRPS 기법을 사용하여 여러 개의 서로 중복되지 않는 박스(box)들로 분할하였다. 실험은 두 부분으로 구성된다. 첫 부분은 여러 개의 박스를 압축하여 박스의 평균 압축률을 조사하였고, 다음 부분은 전위-합 큐브를 구성하는 모든 박스들을 압축한 다음 영역-합 질의의 수행에 필요한 디스크 입출력 횟수를 조사하였다. 실험이 진행된 컴퓨터에는 700MHz 펜티엄 프로세서, 512Mbytes의 메인 메모리, 20Gbytes의 하드 디스크 등이 장착되어 있다.

먼저, 우리는 실험에 사용된 전위-합 큐브를 구성하는 박스들을 압축하여, 압축 비율을 측정하였다. 압축 비율은 원래의 데이타 크기에 대한 압축된 데이타의 크기의 비율을 말한다. 따라서, 압축 비율이 적을수록 많이 압축되었음을 나타낸다. 그림 10의 (a)는 박스의 크기에 따른 압축 비율을 보여준다. 박스는 3차원으로 이루어졌고 모든 차원은 같은 크기를 갖는다. 박스의 모든 점들 중에서 1%만이 값을 저장하고 있다. 압축 비율은 박스의 크기에 비례한다. 박스의 크기가 작으면 압축 비율도 적으며, 박스의 크기가 크면 압축 비율도 높아진다. 박스의 크기가 커지면 서브큐브들의 교차에 의해서 생기는 서브큐브의 수가 증가하기 때문에 위와 같은 현상이 발생한다. 이것은 전위-합 큐브의 특성에 기인한 것이다.

그림 10의 (b)는 박스의 크기는 고정되어 있는 상태에서 박스에 저장된 값의 비율을 변경하면서 측정한 압축 비율을 보여 주고 있다. 박스의 크기는 16×16×16로 고정되어 있다. 저장된 값의 비율이 적으면 압축 비율도 매우 적다. 그러나 저장된 값의 비율이 높아지면 덩달아 압축 비율도 높아진다. 이것의 원인은 앞에서 언급한 것과 같다. 그림 10의 (c)는 박스의 차원의 수에 따른 압축 비율의 변화를 보여주고 있다. 한 차원의 크기를 16으로 고정하였으며, 차원이 증가할수록 희박성은 증가한다. 희박성이 증가할수록 압축 비율이 감소해야 하지만, 차원이 증가할수록 서브큐브들 간의 간섭에 의해 더 많은 서브큐브들이 생기므로, 압축 비율은 큰 변화가 없다.

다음의 실험에서는 압축된 데이타를 사용하는 검색의 성능을 측정하였다. 표 3은 이 실험에서 사용한 파라미터 값들을 보여주고 있다. 그림 11은 여러 개의 질의를 처리하는 과정에서 발생하는 페이지 부재(page fault)의 수를 측정한 결과이다. 여기서의 질의는 한 점에 저장된 값을 검색하는 것이다. 영역 질의는 여러 개의 점(2^d)을 검색하는 것으로 처리되기 때문에 여기에서 보여주고 있는 성능은 영역 질의를 처리하는 성능과 유사한 패턴을 갖는다.

그림 11의 (a)는 질의가 전위-합 큐브의 전 영역에 걸쳐 고른 분포를 보일 경우의 성능 측정 결과이다. 실행한 질의의 개수가 증가할수록 디스크 입출력의 개수의 차이가 벌어짐을 볼 수 있다. 그림 11의 (b)는 질의가 전위-합 큐브의 특정 부분에 집중할 경우의 성능 측정 결과이다. θ가 0.8인 Zipf 분포를 사용하여 질의의 영역이 특정 부분에 편중되도록 질의를 생성하였다. 실행된 질의의 수가 증가할수록 압축된 전위-합 큐브를 사용하는 질의 처리와 원래의 전위-합 큐브를 사용하는 질의 처리의 성능 차이는 매우 커진다.

실험에 사용되는 버퍼는 1,500개의 페이지로 구성하였다. 압축을 하지 않은 경우 하나의 페이지에 하나의 박스를 저장하여 1,500개의 박스를 버퍼에 저장할 수 있다. 압축을 한 경우에는 하나의 페이지에 2.4개의 박스를 저장할 수 있으므로 버퍼에 3,600개의 박스를 저장할 수 있다. 따라서, 압축을 한 경우에는 같은 버퍼에 더 많은 박스를 저장할 수 있으므로 더 적은 수의 페이지 부재가 발생한다. 질의의 영역이 특정 부분에 편중되는

표 3 실험에 사용된 파라미터 값

파라미터	원래의 데이타	압축된 데이타
데이타 큐브의 크기	640 (640 (640	
저장된 값의 비율	1%	
박스의 크기	16 (16 (16	
박스의 개수	64,000	
페이지의 크기	16Kbytes	
버퍼의 크기	1,500 pages	
디스크에 저장된 데이타의 크기	1000Mbytes	470Mbytes
한 페이지에 저장되는 박스의 개수	1	2.4(average)

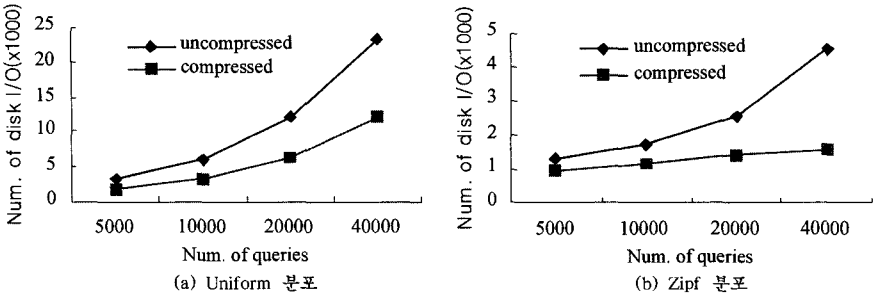

그림 11 질의 처리의 성능

경우에는 대부분의 경우 버퍼에 저장된 박스만으로 질의 처리를 할 수 있다. 따라서, 페이지 부재는 훨씬 감소한다. 이러한 경우에도, 압축한 데이타를 사용하면 자주 질의되는 부분을 더 많이 버퍼에 저장할 수 있기 때문에 압축하지 않은 경우 보다 훨씬 적은 페이지 수의 부재가 발생된다.

5. 결 론

전위-합 큐브는 영역 질의의 효율적 처리에 도움을 줄 수 있기 때문에 많은 연구의 대상이 되어왔다. 그러나, 전위-합 큐브를 저장하는 것은 방대한 공간을 필요로 하기 때문에 현실에서 사용하는 것은 쉽지 않았다. 본 논문은 전위-합 큐브에서의 중복된 값들을 효율적으로 저장할 수 있는 중첩된-서브큐브 압축 방법을 제안하였다. 제안된 방법의 중요한 특징 중의 하나는 압축을 풀지 않고도 저장된 값을 검색할 수 있다는 것이다. 이러한 특징으로 인해서 중첩된-서브큐브 방법으로 압축된 전위-합 큐브를 질의 처리에 사용하면, 디스크 공간을 절약할 수 있을 뿐더러 페이지 폴트의 횟수를 줄일 수 있다. 페이지 폴트의 감소는 질의 처리의 성능을 획기적으로 높일 수 있다.

참 고 문 헌

[1] R. Agrawal, A. Gupta, S. Sarawagi, "Modeling Multidimensional Databases," In Proceedings of the 13th International Conference on Data Engineering, pages 232-243, 1997.

[2] E.F. Codd, S.B. Codd, and C.T. Salley, "Providing OLAP (n-line analytical processing) to user-analysts: An IT mandate," Technical report, 1993.

[3] A. Shoshani, "OLAP and Statistical Databases: Similarities and Differences," In Proceedings of ACM PODS, pages 185-196, 1997.

[4] Ralph Kimball, The Data Warehousing ToolKit, John Wiley and Sons, 1996.

[5] T. Niemi, J. Nummenmaa, P. Thanisch, "Functional Dependencies in Controlling Sparsity of OLAP Cubes," In Proceedings of the second International Conference on DaWaK, pages 199-209, 2000.

[6] C. Ho, R. Agrawal, N. Megiddo, R. Srikant, "Range Queries in OLAP Data Cubes," In Proceedings of ACM SIGMOD, pages 73-88, 1997.

[7] J. Gray, A. Bosworth, A. Layman, and H. Pirahesh, "Data Cube: A Relational Aggregation Operator Generalizing Group-by, Cross-Tab and Sub Totals," In Proceedings of the 12th International Conference on Data Engineering, pages 152-159, 1996.

[8] BMC Software, Constructing a Data Warehouse, White Paper, www.bmc.com/products/documents/9322.pdf.

[9] M. Riedewald, D. Agrawal, A. E. Abbadi, "pCube: Update-Efficient Online Aggregation With Progressive Feedback and Error Bounds," In Proceedings of the 12th International Conference on SSDBM, pages 95-108, 2000.

[10] S. Chaudhuri and U. Dayal, An Overview of Data Warehousing and OLAP Technology, SIGMOD Record, 26(1), 1997.

[11] M. Gyssens and L.V.S. Lakshmanan, "A foundation for multi-dimensional databases," Proceedings of the 23th International Conference on VLDB, pages 106-115, 1997.

[12] Y. Kotidis, N. Roussopoulos, "An Alternative Storage Organization for ROLAP Aggregate Views Based on Cubetrees," Proceedings of ACM SIGMOD, pages 249-258, 1998.

[13] C. Li and X.S. Wang, "A data model for supporting on-line analytical processing," In Proceedings of the 5th International Conference on Information and Knowledge Management, pages 81-88, 1996.

[14] S. J. Chun, C. W. Chung, J. H. Lee, S. L. Lee, "Dynamic Update Cube for Range-Sum Queries," In Proceedings of the 27th International Conference on VLDB, pages 521-530, 2001.

[15] J. S. Vitter, M. Wang, "Approximate Computation of Multidimensional Aggregates of Sparse Data Using Wavelets," In Proceedings of ACM SIG-

MOD, pages 193-204, 1999.
[16] J. S. Vitter, M. Wang, B. Lyer, "Data Cube Approximation and Histograms via Wavelets," In Proceedings of the 7th International Conference on Information and Knowledge Management, pages 96-104, 1998.
[17] S. Sarawagi, M. Stonebraker, "Efficient Organization of Large Multi-Dimensional Arrays," In Proceedings of the 10th International Conference on Data Engineering, pages 328-336, 1994.
[18] Y. Zhao, K. Ramasamy, K. Tufte, and J. F. Naughton, "Array-Based Evaluation of Multi-Dimensional Queries in Object-Relational Database Systems," In Proceedings of the 14th Internaltional Conference on Data Engineering, pages 241-249, 1998.
[19] J. Li, D. Rotem, J. Srivastava, "Aggregation Algorithms for Very Large Compressed Data Warehouses," In Proceedings of the 25th International Conference on VLDB, pages 651-662, 1999.
[20] M. Riedewald, D. Agrawal, A. E. Abbadi, R. Pajarola, "Space-Efficient Data Cubes for Dynamic Environments," In Proceedings of the second International Conference on DaWaK, pages 24-33, 2000.

강 흠 근
1992년 한국과학기술원 전산학과(석사) 1992년~1995년 한국전자통신연구원 연구원. 1995년~현재 한국과학기술원 전산학과 박사과정. 관심분야는 데이타웨어하우스, OLAP, 분산시스템

민 준 기
1997년 2월 한국과학기술원 전산학과(석사). 2002년 8월 한국과학기술원 전산학전공(박사). 2002년 9월~현재 한국과학기술원 박사후과정 연구원. 관심분야는 데이타베이스, XML, 시공간DB, OLAP

전 석 주
1989년 2월 경북대학교 대학원 전자공학과(석사). 2002년 8월 한국과학기술원 정보및통신공학과(박사). 1989년 2월~1995년 9월 현대중공업 중앙연구소 주임연구원. 1997년 3월~현재 안산1대학 인터넷정보과 조교수. 관심분야는 멀티미디어 데이타베이스, 데이타웨어하우스, OLAP, 데이타마이닝

정 진 완
정보과학회논문지 : 데이타베이스
제 30 권 제 1 호 참조