**0. Abstract**

데이터 큐브에 있는 Range Sum Queries(이하, 범위 합계 쿼리)는 분석을 위한 강력한 도구이다. 범위 합계 쿼리는 선택이 크기 수치 값의 범위를 제공함으로써, 지정된 데이터 큐브의 모든 선택된 셀을 통해 집계 동작 (예를 들어, SUM)을 적용한다. 대부분의 응용 프로그램 도메인 분석 도구에서 제공하는 정보는 현재 또는 "거의 현재 할 것“을 요구하고 있다. 데이터 큐브 ‘범위 합계 쿼리’ 기존 기술은 그러나, 데이터 큐브의 크기 순서에 따라 업데이트 비용을 초래할 수 있다. 데이터 큐브의 차원에 따라 데이터 큐브의 크기는 지수적으로 증가하기 때문에, 전체 큐브를 재구축하는 것은 매우 많은 비용을 초래할 수 있다. 데이터 덩어리의 크기가 전체 데이터 큐브를 다시 그것의 차원 수에 지수 때문에 매우 고가 일 수 있다. 우리는 업데이트 비용을 지불하면서 일정 시간 범위 합계 쿼리를 달성할 수 있는 접근 방법을 제시한다. 그리고 우리의 방법은 다양한 합 문제의 전체적인 복잡도를 줄인다.

**1. Introduction**

다차원 데이터베이스 [OLA96] AGS97]로 OLAP 커뮤니티에서 알려진 데이터 큐브 [GBLP96]는, 데이터베이스 및 데이터웨어 하우스의 내용을 분석 할 수 집계 정보를 제공하도록 설계된다. 데이터 큐브 데이터베이스 특성의 서브 세트로 구성된다. 특정 속성은 속성을 측정하도록 선택된다, 즉, 그 값이 관심있는 특성이 선택되는 것이다. 다른 속성은 치수 또는 기능적 속성으로 선택된다. 이 법안의 속성은 크기에 따라 집계된다. 예를 들어, 보험 회사에 의해 유지되는 가상의 데이터베이스를 고려해보자. 하나는 차원으로 측정 속성과 판매, 그리고 CUSTOMER\_AGE 및 DATE\_OF\_SALE와 데이터베이스에서 데이터 큐브를 구성 할 수 있다. 이러한 데이터 큐브는 지역 및 날짜의 모든 조합에 대한 집계 된 총 판매 수치를 제공한다.

데이터 큐브에 적용 할 때 범위 합계 쿼리 유용한 분석 도구이다. 범위 합계 쿼리는 쿼리의 범위 내에서 측정 특성을 요약; 예를 들어 지난 3 개월 동안, 52-37에서 나이를 고객의 총 매출에서 찾을 수 있다. 이 양식의 쿼리는 경향을 찾는 데이터베이스의 속성 사이의 관계를 발견하기에 매우 유용하다. 데이터 큐브를 통해 범위 합계 쿼리는 분석을 위한 유용한 도구를 제공한다. 효율적인 범위 합을 구하는 것과, 특히 온라인 분석 처리 (OLAP) 데이터베이스 분석에 대한 관심이 증가하고, 더 중요해지고 있다.

데이터 큐브의 도입 이후, 추정치를 구축하기 위해, [HRU96] [GHRU97]를 미리 계산하고 또한 데이터 큐브의 하위 집합을 선택하기위해 데이터 큐브 [AAD + 96]의 계산에 관한 데이터베이스에 대한 상당한 연구가 있었다. 다차원 집계 [SDNR96]와 미리 계산 된 요약 [SR96] [JS96]를 색인에 대한 크기로 나누었다. 연구진은 집계 쿼리의 [Lom95] 구체화 된 뷰의 유지 보수 및 처리 [CS94] [GHQ95] [YL95] [GEA + 98]을 살펴보았다. 온라인 분석 처리 연구 (OLAP) Cod93 또한 직접 데이터 큐브에 관한 것이다.

호, 아가 왈, 므깃도와 Srikant [Ho97]라는 사람들은, 우리가 접두사 합 방식을 호출 데이터 큐브의 범위 쿼리로 계산하기 위한 알고리즘을 제안하였다. 접두사 합 방법의 요지는 런타임 임시 쿼리에 응답하기 위해 사용될 수 있는 데이터 큐브의 많은 프리픽스 금액을 미리 계산하는 것이다. 또한, 프리픽스 합 방법은 일정 시간의 데이터 큐브의 모든 범위 합 쿼리의 평가를 허용한다. 접근 방식은 최악의 경우 전체 데이터 큐브와 같은 크기의 배열을 재 구축하는 요구도 발생하는 데 이는, 보통 업데이트 비용에 의해 방해된다.

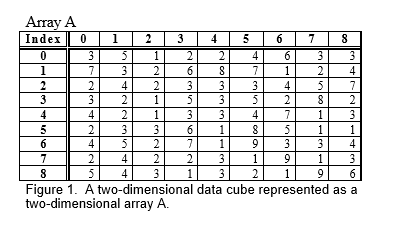
몇몇 문제의 경우에, 업데이트 비용이 중요한 고려 사항은 아니다. 데이터가 정적인 또는 업데이트되지 않는 경우가 존재하기 때문이다. 적당한 갱신 비용이 중요한 애플리케이션도 있긴 있다. 많은 회사들은 새로운 정보가 매일 도달 할 수 있는 현재의 판매 데이터를 추적 관련 데이터로 설정한다. 세계 시장에서의 경쟁이 증가함에 따라, 관리자는 분석 도구가 현재의 또는 "현재 거의"정보를 제공 할 것을 요구한다. 이 업데이트 합리적인 비용으로 새로운 정보를 수용 할 수 없는 경우 등의 애플리케이션의 경우, 데이터 큐브는 유용한 분석 도구로의 역할 수행에 실패 할 것이다. 우리는 데이터 큐브의 차원의 수가 증가함에 따라, 데이터 큐브의 크기는 기하 급수적으로 증가한다. 데이터 큐브 크기의 업데이트 비용이 특히 많은 큰 데이터 큐브에 대해서, 이러한 응용 프로그램은 실용적이지 않을 수 있습니다. 일정 시간의 범위 합계 쿼리를 달성하는 데 방법이 필요하지만, 업데이트 시 전체 데이터 큐브를 재구축 할 필요는 없습니다.

상대 접두사 합 방법 : 본 논문에서는 데이터 큐브 범위 합계 쿼리를 평가하기 위한 새로운 기술을 제시한다. 이 방법은 쿼리에 대한 일정 시간의 성능을 달성시키고 제약 비용을 업데이트시킨다. 또한, 상대 프리픽스 합 접근 범위 문제의 전체적인 복잡도를 줄인다.

**1.1. Paper Organization**

본 논문은 다음과 같이 구성되어있다. 제 2 절에서는 범위 합 문제의 모델을 제시한다.

3 장에서는, 우리는 우리의 알고리즘에 사용되는 일반적인 방법을 개발한다. 우리는 오버레이 및 RP 배열의 개념을 도입하고, 그 동작을 설명한다. 제 4 절에서, 우리는 이러한 구조의 쿼리 및 업데이트 알고리즘을 제시한다. 또한, 방법의 성능 특성을 분석하고, 그 사용 범위 합 문제의 전체적인 복잡도를 감소시킨다는 것 또한 보여준다. 우리는 또한 오버레이 상자의 크기 조정 가능한 매개 변수를 고려하고, 성능 관련 문제를 논의한다. 제 5 장을 마친다.

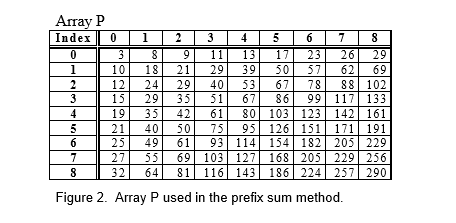


**2. The Model**

우리는 프리픽스 합 방법을 소개 [Ho97]에 제시된 범위 합 문제 동일한 모델을 사용한다.

데이터 큐브를 가정하는 것은 하나의 측정 속성과 D 치수를 가지고 있다. D가 = {1,2, ..., D} 크기의 세트를 나타낸다하자. 예를 들어, 측정 속성이 판매 될 수 있도록, 그리고 치수는 고객 AGE 및 판매의 날짜 수. 각 차원 차원의 고유 값의 개수를 나타내고 있는 크기를 갖는다. 이 크기는, 예를 들어 정적 인 것으로 가정 될 수 있으며, 1 년의 일수와 같은 선험적으로 알려져 있다. 따라서, 우리는 D 차원의 사이즈 N1 × 배열 A를 N2 × ... ×와, ni≥2, i∈D하여 D-차원 데이터 큐브를 나타낼 수 있다. 명확화를 위해, 일반성의 손실없이, 각 차원을 가정한다. 우리 비용 모델은 동일한 크기를 갖는다; 이것은 우리가 더 간결 식의 많은 것을 제공 할 수 있습니다. 따라서, 각 차원의 크기가 =의 N1의 =의 N2 = ... = 차에서, n은 할 수 있습니다. 우리는 각 배열 요소 셀 호출한다. 어레이 A의 총 크기는 셀이다. 우리는 배열의 각 차원에서 인덱스 0을 시작했다고 가정한다. 표기의 편의를 위해, 두 차원의 예에서 우리는 I가 수직 좌표이며, j는 수평 좌표이다. A [I,J]같은 배열 (A)에 셀을 참조한다. 마찬가지로, 우리는 P [I, J, RP]의 [I, J]으로 RP에 O [I, J]와 같은 오버레이에서 세포 및 세포로 배열 P 셀을 참조하라.

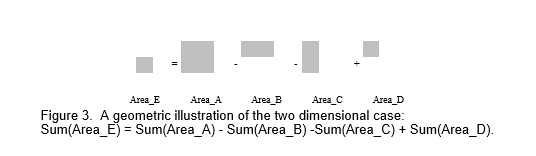
배열 A의 각 셀 크기에 의해 형성된 D-차원 공간에서 주어진 점에 해당하는 측정 속성 (예를 들어 총 판매)의 집계 값을 포함한다. 예를 들어,에서 측정 속성 판매 및 치수 고객 AGE 및 DATE\_OF\_SALE, 셀을 주어진 [37, 25] 일 25 일 37 세의 고객에게 총 매출이 포함되어 있다. 어레이 A의 범위 합 질의는 특정 범위 내에있는 모든 셀의 합으로 정의된다. 예를 들어, 일에서 37 세의 고객에게 총 판매를 요청하는 범위 합계 쿼리 (20) (22)에 셀 A [37, 20], A [37, 21]을 합산하여 대답 할 것이다. A [37, 22]. 우리는 이 문서의 나머지 부분 단순히 범위 쿼리로 범위-합계 쿼리를 참조한다. 호 등으로. 알. 지적, 여기에 제시된 기술도 COUNT, AVERAGE, ROLLING SUM, ROLLING AVERAGE, 및 이진 연산자를 얻기 위해 적용 +되는 역 이항 연산자가 존재 할 수 있다.



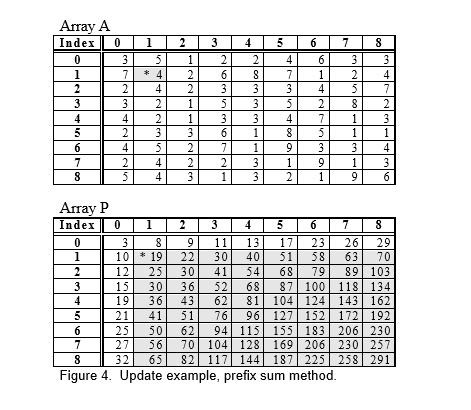
우리는 배열 A의 다음과 같은 특징을 관찰 어레이 A는 범위 합 질의를 해결하기 위해 단독으로 사용될 수 있다; 우리는 Naive 방법으로 참조됩니다. 배열 A의 임의의 범위 쿼리는 O (차)를 요할 수 : 전체 배열의 범위 범위 쿼리는 배열의 모든 세포를 합산이 필요하다. 셀에 대한 새로운 값이 주어진 업데이트 어레이의 셀의 값을 변경함으로써 간단하게 달성 될 수있다 : A와 어레이의 업데이트는 O (1)이 걸릴. 질의 및 업데이트 똑같이 가능성이 있다고 가정하면, 우리는 질의 및 업데이트 비용의 제품을 고려하여 방법의 전체적인 복잡도를 나타낼 수 있다. Naive 방법의 경우, 쿼리 및 업데이트 비용이 제품은 O (차)이다 \* O (1) = O (차).

프리픽스 합 방법은, 어레이 같은 크기의 배열 P, A (도 2)의 다양한 저장 미리 계산 프리픽스 합. 각 셀까지의 모든 셀의 합을 포함하고 어레이 A. 자체를 포함 셀 P [2,1] 모든 셀의 합을 포함하면서, 예를 들어, 셀 P [4,0]을 A [4,0], 또는 19의 범위에서 A [0,0]의 모든 셀의 합을 포함 어레이 A의 [2,1], 또는 24의 범위 A [0,0]이다. 전체 배열의 합이 마지막 셀에서 발견된다. P [8,8]. 공식적으로, P를 사용하여 모든 0≤x≤n 및 i∈D을 위해, D 치수에 대한 범위 쿼리는 상수 (2D) 셀 조회와 응답을 할 수 있다.

그림 3은 프리픽스 합 방법의 기본 개념을 제시 우리가 관심의 영역을 격리 할 때까지의 범위 쿼리의 영역에 대응하는 합계 추가 등 다양한 영역의 합계를 감산하여 판정 될 수 있다. 우리는 이러한 모든 영역은 셀 A [0,0]에서 시작하고 일부 셀에 확장 있다; 따라서, 이들 영역들의 합계는 P. 단일 셀의 값을 판독하여 찾을 수 있다. 프리픽스 합 방법은 따라서 P 배열 한 개인 셀을 판독하는 문제에 범위 합계 쿼리를 감소시켰다.



그러나, 배열 P에 대한 업데이트는 비용이 많이들 수 있다. 업데이트는 A가 [i, j]가 변경된 셀보다 인덱스 P에 더 큰 모든 세포의 재계산을 할 수 있다. 그림 4는 배열 (A)에 업데이트를 보여준다. 배열 P.에의 대응에 미치는 영향 배열 A의 셀 A [1,1] 3에서 4로 업데이트되었다. 앞서 언급 한 바와 같이, P 세포 배열 A의 영역의 일부 금액을 포함 배열 P의 음영 지역은 A [1,1]로 업데이트의 영향을 받는 세포를 보여준다. 구성에 의해, [1,1]보다 큰 인덱스 P의 모든 셀은 셀의 값 A [1,1]을 포함 할 것이다; 따라서,이 세포를 업데이트해야 할 때 A [1,1]의 값이 변경. 이 업데이트는 A1,1] 원인은 P 행렬에 걸쳐 업데이트를 계단식 셀 수 있다. 업데이트는 A [0,0], 최악의 경우 셀에, P.의 모든 셀의 재계산이 발생한다. 프리픽스 합 방법은, 따라서, O (ND)의 갱신의 복잡성을 갖는다. 접두사 합 솔루션 따라서 Naive한 방법으로 갱신 및 조회 특성을 거래하고 있다. 접두사 합 방법에 대한 쿼리 및 업데이트 비용 제품 (1) \* O (n)은 Naive 방법과 동일 O (과) = O이다.



**3. The relative prefix sum method**

접두사 합 접근 방식은 매우 강력하다. 관계없이 데이터 큐브의 크기를 일정 시간 범위 합 질의를 제공한다. 한편, 데이터 구조의 업데이트로 인해 계단식 업데이트 효과로 매우 고가이다. 앞서 언급 한 바와 같이, 일부 응용 프로그램 도메인은 정기적으로 업데이트해야한다; 데이터 큐브가 매우 크고, 많은 차원이 있을 때, O (차)의 업데이트 비용이 비현실적이 될 수 있다. 우리는 접두사 합 방식의 쿼리 능력을 가지고 있지만 업데이트 성능을 향상하는 방법을 추구하고 있다.

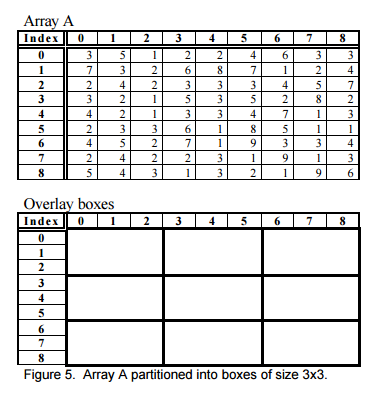
P의 값 비싼 비용 업데이트는 셀 값 사이의 의존 관계의 결과이다. 차례로 이러한 의존성은 계단식 업데이트 효과가 발생한다. 그러나 우리는 완전히 이러한 의존성을 제거 할 수 없다. 접두사 합 방법은 정확하게는 세포 사이의 종속성 때문에 작동한다. 일반적으로, 우리는 우리가 증가 쿼리 비용으로 지불 종속성을 제거 할 때 우리는 모든 종속성을 제거 할 경우, 우리는 그 쿼리 비용 O (차) 인 Naive한 접근 방식으로 남아있을 것이다. 우리가 완전히 종속성을 제거 할 수 있기 때문에, 우리는 또한 완전히 업데이트를 계단식 제거 할 수 있다. 그러나 아마도 우리는 고유 세포에 계단식 업데이트를 제한 경계를 만들 수 있다. 이 과정에서 우리는 일정 시간의 질의 특성의 손실을 초래 쿼리의 오버 헤드를 소개하지 않도록 주의해야한다.

상대 접두사 합 접근 방식은 데이터 큐브 범위 합계 쿼리에 응답하기위한 새로운 방법이다. 위의 방법은 추가하고도 3에 그려진 바와 같이, 쿼리 전체 영역의 합을 구하는 영역 합을 감산한다. 상기 방법은 Cascading 갱신을 제어하면서 일정 시간 조회를 제공 할 새로운 데이터 구조를 사용한다. 세포를 구별하기 위해 업데이트 계단식 제한 경계를 생성함으로써,이 방법은 범위 합 문제의 전체적인 복잡도를 줄인다.

오버레이와 상대 접두사 (RP) 배열 : 우리의 방법은 두 가지 구성 요소를 사용한다. 고정 된 크기의 영역으로 오버레이 파티션 배열 A는 오버레이 상자를 수행했다. 오버레이 상자는 그들 앞에 배열의 영역의 합에 대한 정보를 저장한다. RP는 어레이 A와 동일한 사이즈의 배열이며; 그것은 오버레이에 의해 정의 된 영역 내에서 상대 접두사 금액이 포함되어 있다. 우리는 "즉시"접두사의 합계를 구성한다. 여러 개 중에서 두 개의 구성 요소를 사용하여, 함께, 구성하는 요소는 세포를 구분하는 계단식 업데이트로 제한 할 수 있다. 우리는 먼저 오버레이, 다음 RP를 기술에 대해 설명하고자 한다.

**3.1 Overlay**

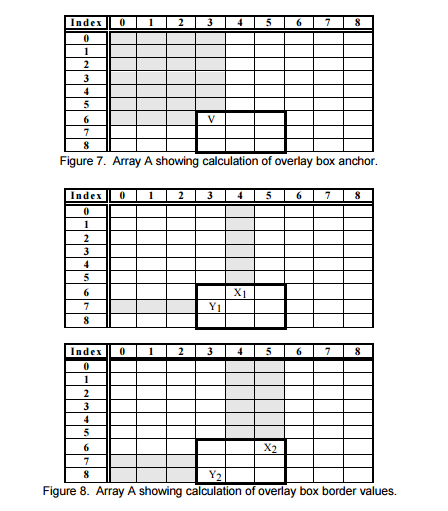
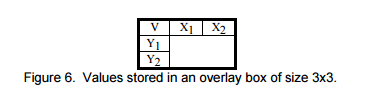
Overlay는 array A를 완벽히 같은 사이즈의 셀로 분리한 구역인, 독립된 거대한 사각형(이하 ‘box’)의 집합이라고 정의한다.(그림5) 이 그림에서 array A는 3x3크기의 overlay box들로 분할되어있다. 편의상 overlay박스의 각 차원의 길이를 k 라하자. Array A의 크기를 라 하면 overlay box의 총 개수는 이다.



이 논문에서 몇 가지 용어는 뒤에서 정의한다. Array A에서 overlay box의 첫번째 셀(각 차원에서 index가 가장 낮은 셀)이 (이면 overlay box가 (에 고정되어 있다(anchored at)고 말하고 overlay box를 라고 한다. 첫 번째 overlay box는 에 고정되어 있다. array A의 셀 (이 overlay box의 경계에 포함된다면 ‘array A의 셀을 overlay box 가 커버(cover)했다’고 한다.

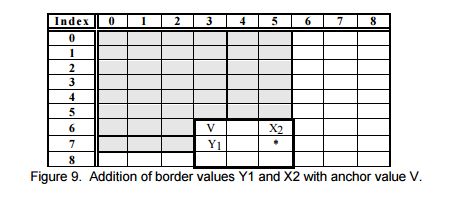
그림5를 보아라. 그림5에서 k=3이고 각 overlay box는 3x3 크기를 가진다. 총 overlay box의 갯수는 이다. box들은 (0,0), (0,3), (0,6), (3,0), (3,3), (3,6), (6,0), (6,3), (6,6)에 고정되어 있다. 각 overlay box들은 크기가 인 array A 범위 안에 해당되므로 이 예시에서는 각 overlay box들은 array A의 개의 셀을 커버한다.

그림 6은 overlay box에 저장된 값들을 보여준다. 각 overlay box들은 하나의 고정값(anchor value)과 개의 경계값(border value)를 저장한다. V는 고정값이고 X1, X2, Y1, Y2는 경계값이다. Overlay box에 커버된 다른 셀들은 overlay에서 필요가 없으므로 저장되지 않는다.



Overlay box에 저장된 값들은 박스 밖 지역의 합을 나타낸다. 고정값은 A안에 있는 셀 중 V이하의 셀들을 제외한 A위의 있는 셀들의 합이다. 그림 7은 array A위에 나타난 overlay box를 나타낸다. Overlay box의 경계값은 array A의 색칠된 부분의 합과 같다. 일반적으로 말하자면, (에 고정된 overlay박스의 고정값은 과 값다. 그림 8 은 경계값의 계산을 보여준다. 경계값들은 array A에서 색칠된 값들의 합과 같다. 경계값 X1은 X1을 포함하는 열의 X1 위의 값들의 합이다. 경계값 X2는 X1 위의 셀들을 포함하여 자신 위의 셀들의 합이다. 경계값 Y1은 Y1을 포함하는 행의 Y1 왼쪽 값들의 합이다. 경계값 Y2는 Y1 왼쪽 셀들을 포함하여 자신 왼쪽 셀들의 합이다. 여기서 경계값들은 누적값인 것을 확인할 수 있다.(X2는 X1을 포함하므로 넓게보면 Xn은 X1,X2,…..,Xn-1 을 포함한다.) 다른 차원의 경계값들도 같은 방법으로 계산된다. 일반적으로 에 포함된 경계값들은

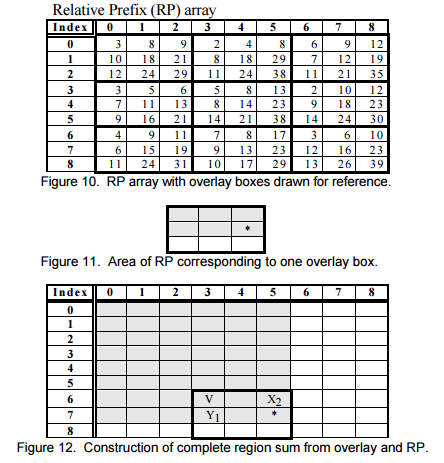
과 같다.



앞에서 언급했듯이 그림3의 영역은 모두 에서 시작해서 a의 임의의 셀에서 끝난다. 그림9는 \*이 표시된 셀을 보여주고 이 셀을 포함하는 overlay box가 array A위에 나타나 있다. 이 예시에서 이 셀은 (7,5)이다. 문제는 에서 시작해서 에서 끝나는 구역의 합을 구하는 것이다. 이때, 두 단계에 거쳐서 합을 구한다. 첫 번째로 overlay를 이용하고 두 번째로 RP를 이용한다. Overlay에 있는 값들을 이용하여 그림9의 색칠된 구역의 합을 구한다. 색칠된 부분은 문제에서 원하는 최종 목표 구역에서 overlay box의 바깥쪽 부분을 나타낸다. 이 구역을 구하기 위해 고정값에 Y1과 X2를 더한다. 고정값은 의 합에서 을 뺀 것을 의미한다. Y1는 까지의 합이다. X2는 까지의 합이다. 각 구역은 위 그림에 나와있다. 하나의 고정값과 두 경계값을 더하는 것은 색칠된 부분의 합을 나타낸다.

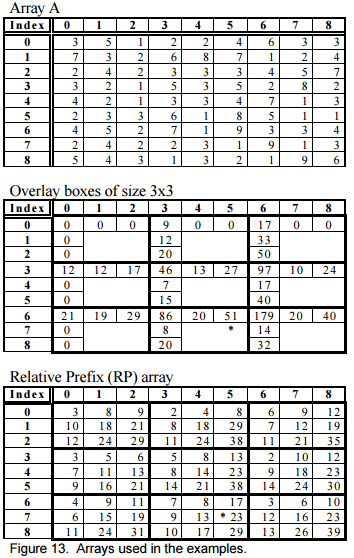
**3.2 Relative prefix array(RP)**

상대적 전위 행렬(RP)는 A와 크기가 같다. 이것은 overlay box들과 일치하는 구역으로 분할되어 있다. 그림 10이 overlay box가 그려진 RP를 나타낸다. 각 RP구역은 box를 둘러싼 부분과 관련된 전위합을 포함한다. RP의 각 구역은 다른 구역과 독립이다. 그림11은 한 overlay box와 일치하는 RP구역을 보여준다. 그림에서 \*는 박스 안에서의 셀의 위치를 보여준다. 셀\*의 A의 색칠된 부분의 합이다. 일반적으로 이 셀을 커버하는 overlay box의 와 고정셀 위치 가 주어졌을 때 에 저장된 값은 이다.



앞에서 언급했듯이 overlay 값은 총 구역 합의 일부분을 제공한다. RP는 그 나머지 부분을 제공한다. 그림12는 overlay와 RP를 이용한 완성된 구역을 나타낸다. 그림에서 \*는 A의 임의의 셀, 여기에선 를 나타낸다. 이 셀을 커버하는 overlay box가 array A위에 나타나 있다. 위에서 언급했듯이 overlay box의 고정값과 경계값이 overlay box 바깥 쪽 색칠된 부분의 합을 나타낸다. RP안에있는 셀\*는 overlay box의 색칠된 내부를 나타낸다. 두 합을 더하면 A의 색칠된 모든 구역의 합을 구할 수 있다. 이러한 과정으로 에서 시작하는 구역의 합은 overlay와 RP를 이용해 구할 수 있고 이것은 그림3에서 요구된 구역 합을 충분히 제공할 수 있다. 이 과정은 하나의 고정값, d개의 경계값, 한 RP값이 요구된다.

**3.3 Examples using the overlay and RP**



여기에선 overlay와 RP를 이용하는 구체적인 예를 소개한다. 그림13은 A, overlay, RP를 나타낸다. RP에 overlay 구역이 나타나있다. 다음 예는 그림을 설명한다. 우선 고정값과 경계값을 계산하는 예시를 제공한다. 최종적으로 완전한 구역합을 구한다.

Calculating anchor and border values

(3,3)에 고정된 overlay box의 고정값과 경계값을 구한다. Overlay 셀 O[3,3]의 고정값은 이다. 이것은 을 제외한 A이상 셀들의 합이다. 경계값 , 이다. 유사한 방법으로 overlay 셀인 경계값 , 경계값 이다.

Calculating a complete region of sum

현재 에서 까지 구역의 합을 구하려고 한다. 셀 (7,5)는 (6,3)에 고정된 overlay box에 있다. 고정값 하나와 경계값 두개를 요구하는 overlay box의 바깥쪽 색칠된 부분의 합을 구하자. Overlay box의 고정값은 86이다. 이제 경계값을 구하자. (7,5)는 고정값에서 밑으로 한칸, 오른쪽으로 두칸 옆에 위치하므로 경계값 Y1,X2가 필요하다. 경계값 Y1은 overlay cell O[7,3]=8임을 알 수 있다. 경계값 X2는 overlay cell O[6,5]=51임을 알 수 있다. 따라서 overlay box의 바깥쪽 색칠된 부분은 86+51+8과 같다. 이제 overlay box 내부의 색칠된 구간을 구한다. 합은 RP[7,5]=23임을 알 수 있다. 따라서 총 면적의 합은 이다.

**요약**

데이터 큐브에서 Range sum quere들은 분석을 위한 강력한 도구이다.

Range sum query는 지원한다. 데이터 큐브내에있는 모든 선택된 셀들을 통합지원(ex.sum).

여기서 선택된 데이터란 숫자크기에 대한 값의 범위를 제공함으로써 그에 선택되는 셀들이다.

많은 응용프로그램 도메인은 “분석도구들에 의해 제공된 정보들이 현재 또는 “near-current” 상태여야 한다”.라는 것을 요구한다.

**4. Operations using the relative prefix sum approach**

우리는 range sum queries 와 updates를 이용한 the relative prefix sum 접근방식을 이용한 알고리즘을 보여주려 한다. 우리는 큰 선명도(grater clariry)에 대한 알고리즘을 앞에서 배웠었다.

**4.1 Range Sum Queries**

Range Sum Queries 에 대한 알고리즘은 Appendix 1 (부록 1)이다. 알고리즘은 두 부분으로 구분되어 진다. RangeSumQuery() 와 GetRegionSum() 이다. RangeSumQuery()는 최상위함수이다. 즉 figure 3의 내용에서 했던 것과 비슷할 것이다. 이 함수는 입력 값으로 low 또는 high의 쌍으로 두 개의 매개변수를 사용한다. 이것은 정수의 배열이고 the query range의 경계인 하한선, 상한선으로 저장된다. 배열들은 C 라는 것으로 첨자로 기입된다. 즉, low[0]은 첫 번째 차원에 있는 query의 하한선이다. RangeSumQuery()은 A[0,0,...,0]부터 낮고, 높은 모든 가능한 조합까지의 지역의 부분합을 계산하기 위해 GetRegionSum()을 반복적으로 요구한다. 따라서 2d부분합들을 필요로 한다. 최종 합계를 얻기 위해 부분합을 조합한다. sign변수는 각각의 부분합이 최종 합계로부터 더하거나 빼주는 것인지에 대하여 계속해서 추적한다.

GetRegionSum()은 (0,0,..,0) 부터 시작해 정해진 셀 c에 끝나는 어떤 지역에 있는 부분합을 계산하기 위해서 the overlay와 Relative Prefix를 이용한다. 이 함수는 c라는 하나의 parameter를 가지고 있다. c는 한 정수 배열에 존재하는 c의 지역에 저장된다. 부분합은 0으로 초기화, 처음에 이 함수는 c를 포함하는 overlay box를 결정한다. Overlay anchor는 부분합에 더해진다. 그리고 이 함수는 overlay box에서 cell c 의 offset을 계산한다. Offset을 이용해서, d (경계 값)가 선택되고 부분합이 더해진다. Cell이 주어진 차원에서 offset 0일 때, 코드를 간단히 하기 위해 경계값은 불필요하다. 이전부터 우리는 offset zero에서 모든 경계값을 0으로 설정해왔었다. Relativ Prefix로부터 one value를 부분합에 더했다.

GetRegionsum() 함수는 하나의 +d 라는 고정값과 RP로부터 나온 하나의 +값을 필요로 한다. [Ho97] 와 같이 우리는 주어진 data cube에서 나온 차원들의 수를 고정적이라고 가정한다. 따라서 본 절차는 다소 일정시간이 필요하다. 그 이유는 각 region의 합은 일정시간 내에서 얻어지고, region 합들의 상수들을 data cube에서 range sum queries 응답하는데 필요로 하기 때문이다. 우리의 방법은 일정시간 동안 query 수행을 제공하는 것이다.

**4.2 Updates**

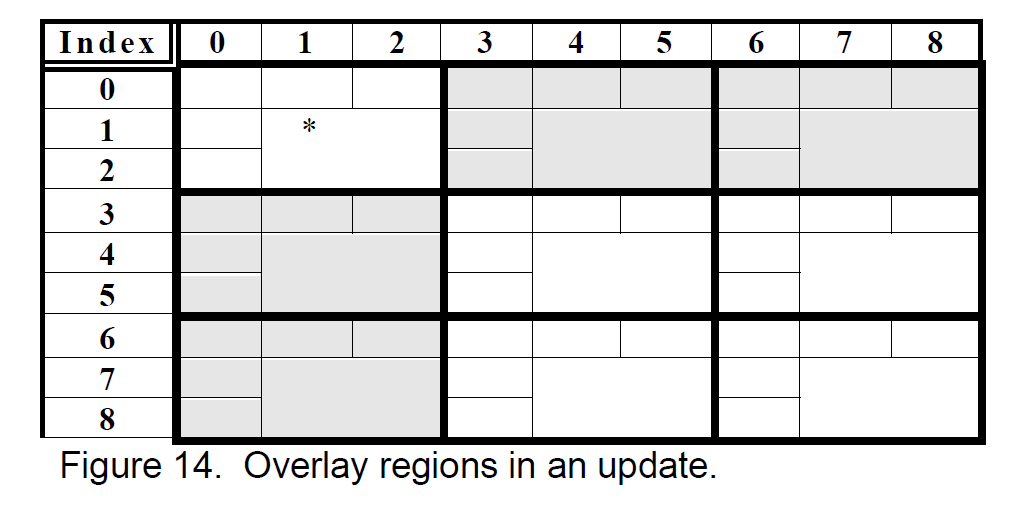
데이터 큐브에 셀 C를 업데이트하려는 것은 RP 및 overray에 대한 업데이트가 필요합니다. updates에 관한 알고리즘은 Appendix 2에서 볼 수 있다. 이 알고리즘은 UpdateCell()라는 함수를 사용한다. UpdateCell()은 두개의 parameter를 가지고 있다. 하나는 정수 array c이다. 이것은 상기 cell의 좌표를 update하기 위해 저장하는 것이다. 그리고 하나는 newvalue이다. 이것은 cell의 새로운 값을 저장한다. 먼저 RangeSumQuery(c,c)를 통해 구했던 cell c의 이전 값을 얻을 수 있다. 그리고 나서 cell의 previouse 값과 new값의 차이를 계산할 수 있다. (new value-previouse value).

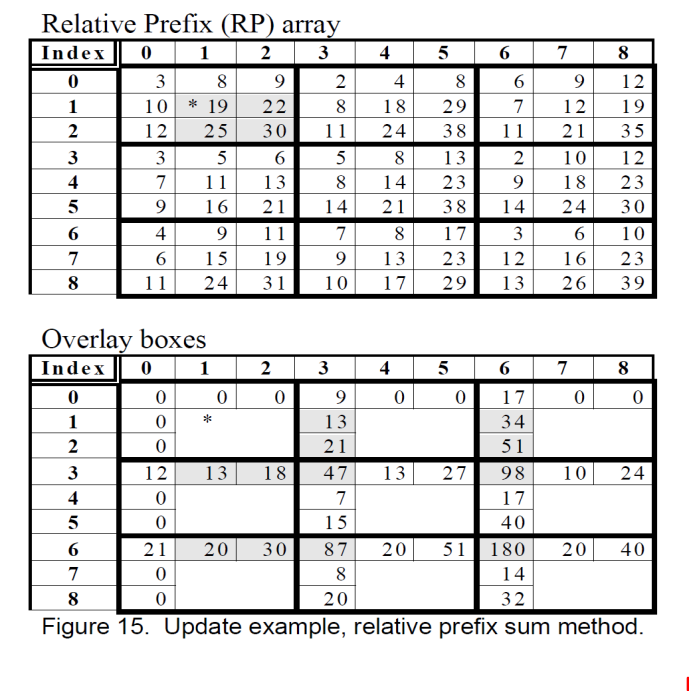
우리는 변경된 cell을 포함한 overlay box를 결정한다. 그리고 overlay box내에있는 cell의 offset을 계산한다. 또한 각 다음의 overlay 경계의 시작을 말해주는 차원에 있는 좌표를 결정한다.

우선 이 알고리즘은 RP에 영향을 받은 cell을 업데이트 한다.

그림 15는 overlay와 RP.에 대한 업데이트를 보여준다. \* 은 Array A 안에 변경된 셀의 위치를 나타낸다. Relative Prefix의 독립적인 각각의 영역들을 확인해보자. 하나의 overlay box에 의해 변경된 지역에 대해 RP내의 업데이트가 제한된다. Update cell보다 overlay boundary 이 큰 index에서 RP cell들은 영향을 받는다. 다음 우리는 overlay를 업데이트 한다. Overlay를 업데이트하려는 것은 여러 단계를 걸친다. Figure 14와 같이 update하려는 overlay는 몇몇 범주로 분류된다. Figure 14에서 \* 는 변경하려는 cell의 위치를 표시한다. overlay box의 경계 값은 update를 필요로 할 수 도 있는 차원마다 overlay box들의 한 그룹이 있다 변경된 cell의 오른쪽에 있는 음영 영역은 그 경계 overlay box 를 보여준다. 첫 번째 차원에서 값이 업데이트에 의해 영향을 받을 것이다. 변경된 cell 아래의 음영 영역은 그 경계 overaly box를 보여준다. 두 번째 차원에서 값이 엡데이트에 의해 영향을 받을 것이다.

아래 변경된 cell의 오른쪽에 있는 비 음영부분은 그 고정값들을 포함하여 혼자 업데이트를 받을 것이다. 이처럼 혼자만의 업데이트를 하는 것을 우리는 interior 라고 한다. 이 알고리즘은 overlay box의 각 그룹을 업데이트 한다. 각 차원에 대한 영향을 받는 경계 값이 업데이트 된다. 차례로 \* 보다 크거나 같은 열에 업데이트의 영향을 받은 경계값 을 확인한다. 마지막으로, interior overlay box들의 고정값이 업데이트된다.





**Update Example**

Figure 15는 overlay 및 RP의 업데이트를 보여준다 \*은 array A내에서 변화된 것의 위치이다. RP의 각 영역은 독립적인 것을 기억하자. 오직 변한 overlay box와 같은 RP 의 cell만 수정될 것이다.

이 경우에는 네 개의 음영부분 cell만 변할 것이다. Overlay box경계내의 RP에서 계속해서 업데이트한다. 하지만 경계에서는 멈춘다. 다른 overlay 박스들로 커버된 RP내의 cell은 변경되지 않는다. 또한 overlay box들의 몇몇의 그룹 내에 있는 cell들은 영향을 받는다.

Overlay boxes directly to the right of the change will require updates to border

values in rows greater than or equal to the changed cell (e.g., overlay cells [1,3],[2,3] and [1,6],[2,6]).

Overlay boxes directly below the change will require updates to all border values in columns greater than

or equal to the changed cell (e.g., overlay cells [3,1],[3,2] and [6,1],[6,2]).

Interior overlay box들은 하나의 바뀐 고정값을 가진다. 이를 테면 (e.g., overlay cells [3,3], [3,6], [6,3] and [6,6]) 이 예를 통해 열 두 개의 overlay cell들이 바뀔 것이다.

고정값 바로 아래에 있는 cell의 업데이트가 발생할 때를 참고해보자. ex) cell[0,0]

이것은 다른 overlay box들의 고정 cell들을 업데이트 하는 것만 필요로 한다. 어떤 경계값들은 어떠한 overlay box에서 변경될 필요가 없을 것이다. The relative prefix sum 방법은 cell들은 구분하기 위해 계단식 업데이트를 제한한다. 예를들면, the overlay 알고리즘에 대한 총 업데이트 비용은 16cells이다. (12개의 overlay cells + 4개의 RP에서의 cell) =16 이것은 Figure 4에서 보여주는 것처럼 prefix sum방법인 64개의 cell들을 다 더하는 것과 비교된다.

이제 우리는 성능특성 및 relative prefix sum method의 바꿀만할 parameter들을 검토할 것이다.

**4.3 Choosing the overlay box size**

**As noted earlier, finding a value in P using RP and the overlay requires one anchor value + d border**

**values + one value from RP.**

앞에서 언급한 바와 같이, RP와 overlay를 이용하여 P값을 찾는 것은 하나의 고정값 + d경계값 +RP로부터의 한 값이다.

**Range sum queries using the overlay box method are thus achieved in constant time.**

Overlay box 방법을 이용한 Range sum queries는 일정시간에 따라서 실행된다.

**This is irrespective of the overlay box size.**

Overlay box크기에 관계없이 실행된다.

**However, the overlay box size does affect update costs.**

그러나 overlay box 크기는 업데이트 비용에 영향을 미치지 않는다.

**Assume a data cube with d dimensions, each dimension having size n, and an overlay box of size k on each dimension.**

d 차원을 갖는 Data-cube를 가정해보자, 크기를 n으로하는 각각의 차원, 각각 차원에 해당하는 box의 크기는 k이다.

**For convenience, assume n is evenly divisible by k.**

편의상, n은 k로 나눠 진다고 가정하자.

**Further assume that the cost of writing an overlay cell is equal to the cost of writing a cell in RP, as is the case when both structures are in main memory (RAM).**

또한 두 구조 메인 메모리에 있는 경우와 같이 overlay cell을 쓰는 비용은 RP에서 cell을 쓰는 비용과 같다고 하자.

**In the worst case, an update to the data cube will affect (k-1)d cells in the RP array + d(n/k)(kd-1) overlay border cells.**

최악의 경우, Data-cube에 대한 업데이트는 RP배열의 (k-1)/d cells + d(n/k)(kd-1) overlay border cells + (n/k-1)d overlay 고정 cells에 영향을 미칠 것이다.

**This formula can be reasonably approximated as kd+d(n/k)(kd-1)+(n/k)d**

이 수식은 다음과 같이 적당하게 근사화 될 수 있다. **kd+d(n/k)(kd-1)+(n/k)d**

**which can be simplified as kd+dnkd-2+(n/k)d.**

이것은 **kd+dnkd-2+(n/k)d**. 로 단순화 시킬 수 있다.

**In the formula, note that the term kdaccountsforthetotalcostofupdatesinRP;theremainingcosts**

**are associated with updating the overlay.**

이 공식을 통해서 kd는 RP에서 업데이트에 대한 총 비용을 설명한다. 나머지 비용들은 the overlay를 업데이트하는 것과 관련이 있다.

**We are considering the case where the cost of updating cells in the overlay and cells in RP are identical;**

우리는 RP에서 cell들과 overlay cell들을 업데이트 비용이 동일한 경우를 고려해보자.

**therefore, we can minimize the total update cost by minimizing the equation as a whole.**

따라서, 우리는 전체 방정식을 최소화함으로써 전체 업데이트 비용을 최소화 할 수 있다.

**By using approximation, we find that the cost is minimized when the overlay box size is chosen to be**

**k=n1/2.**

근사를 이용해서 overlay box 크기가 k=n1/2되도록 선택될 때 비용이 최소화되는 것을 알 수 있다.

**When the box size is larger, fewer overlay cells will be updated, but more cells in RP will be updated,**

**leading to a larger total number of affected cells.**

**Box** 크기가 클 때, 영향을 받은 cell들의 총 개수가 커지는 것이 이어지면서 더 적은 overlay cell들이 업데이트되지만 RP에서 cell들은 더 업데이트 된다**.**

**When the box size is smaller, fewer cells in RP are updated, but more cells in the overlay are updated, again resulting in a larger total number of affected cells.**

Box 크기가 작은 경우, RP에서 cell들은 더 적게 업데이트 된다. 하지만 overlay에서 cell들은 더 많이 업데이트 된다. 그 결과 영향을 받은 총 cell의 개수가 늘어난다.

**Choosing a box size of n results in the lowest total number of cells in overlays and RP being changed.**

box 크기 n을 선택하려는 것은 변경된 overlay 및 RP 에서 cell들의 총 개수를 최소화시키는 결과이다.

**This is desirable when the cost of accessing cells in overlays is equal to the cost of accessing cells in RP.**

Overlay에서 cell들에 접근하는 비용이 RP에서 cell들에 접근하는 비용과 동일할 때 바람직하다.

**With a box size of k=n1/2,theworst-caseupdateformulayieldsO(nd/2).**

k=n1/2의 box 크기를 가질 때, 최악의 경우 업데이트 공식은 O(nd/2)이다.

**The product of the query cost and update cost is thus O(1) \* O(nd/2)=O(nd/2).**

쿼리 비용 및 업데이트 비용의 곱은 O(1) \* O(nd/2)=O(nd/2).이다.

**This is in contrast to the prefix sum algorithm and the naive method, both of which have a total cost of O(nd).**

위의 방법은 O(nd)의 총비용을 가지는 다음 두 가지 The prefix sum 알고리즘 과 the naïve method 와는 대조적이다.

**The relative prefix sum approach thus reduces the overall cost of the problem.**

The relative prefix sum 방법은 문제의 전체 비용을 감소시킨다.

**4.4 Practical considerations**

많은 실제 application 에서, RP의 큰 크기는 disk에 저장되는 것을 필요로 한다. RP가 매우 큰 경우에도 overlay들이 main memory에 유지될 수 있다. k는 각각의 차원에서 overlay box의 길이라는 것을 상기해보자. 앞에서 언급된 the overlay box들은 kd개의 cell들과 같이 RP배열의 영역을 커버한다.

**Each overlay box requires (kd - (k-1)d) cells of storage.**

각각의 overlay box들은 (kd–(k-1)d)개의 cell들의 저장을 필요로 한다.

**The storage required by the overlay is thus significantly smaller than that used by the corresponding area of array**

The overlay에 의해 필요로 하는 이 저장은 array의 해당하는 영역보다 훨씬 더 작다.

**RP, and space savings grow larger as the size of the overlay box grows.**

The overlay box가 커지면서 RP 및 공간저장은 점점 커질 것이다.

**For example, consider a two dimensional array RP and an overlay size of 100×100 cells**

예를 들면, 이차원의 array RP 와 100\*100 개의 cell들을 갖는 overlay의 size를 생각해보자,

**The overlay box needs (100^2– 99^2)=199 cells of storage, while the region of RP covered by the overlay box requires 10,000 cells; the overlay box requires less than 2% of the storage of the corresponding region of RP.**

The overlay box는 (100^2– 99^2)=199 의 저장 cell을 필요로 할 것 이다. The overlay box에 의해 커버된 RP의 영역은 RP의 해당영역에 저장하는 것보다 2% 작게 필요로 한다.

**Given suitable box sizes, it may be feasible to keep all of the overlay boxes in main memory, while RP resides on disk.**

RP는 디스크에 있는 동안 적절한 box 크기를 고려하면, 메인 메모리에 있는 overlay box들을 모두 유지 할 수 있을 것이다.

**Figure 16 shows the storage requirements of overlay boxes as a percentage of the storage required by the region.**

Figure 16은 영역에 의해 저장공간을 비율로 나타내는데 요구되는 overlay box들이 필요로 하는 저장 공간이다.

**they cover in RP as d and k are varied.**

변수 (d 및 k)의 를 RP에서 커버한다.

**Note that, as the overlay box size grows, overlay boxes use dramatically less storage than the region in RP they cover**

The overlay Box size(k)가 증가하면서, overlay box는 RP가 커버하는 RP의 영역보다 적은 저장공간을 사용한다.

**Maintaining overlays in main memory would have a substantial positive impact on query and update performance.**

메인 메모리에서 overlay들을 유지하는 것은 쿼리 및 업데이트에 상당한 긍정적인 영향을 미칠 것이다.

**In particular, much of the method's update cost is incurred in updating overlay box**

**values.**

특히, 위의 업데이트 비용의 대부분은 overlay box 값들을 업데이트하면서 발생한다.

**The optimal box size formula assumes that the cost of accessing overlay cells is the same as the cost of accessing cells in RP; but when overlays are in memory, their access cost is much smaller than the cost of accessing cells in RP on disk.**

최적 box 크기 식은 “overlay cell들에 접근하는 비용은 RP의 cell들에 접근하는 비용과 같다” 라고 가정한다. 그러나 overlay들이 메모리에 있을 때, overlay cell들의 비용은 디스크에 있는 RP의 cell들에 접근하는 비용보다 적다.

**Consequently, we can expect the optimal overlay box size in this configuration to be larger than the formula predicts.**

따라서, 우리는 최적의 overlay box 크기가 수식으로부터 예측한 것보다 클 것으로 예상 할 수 있다.

**Larger box sizes are desirable because they save more storage, and result in fewer overlay boxes being modified during the update process.**

큰 box 크기일수록 바람직하다. 왜냐하면 더 많이 저장하고 그 결과 업데이트를 진행하는 동안 overlay box들이 더 적게 변경되기 때문이다.

**Since disks are block-based devices, the cost of accessing a cell in RP is related to the cost of accessing a disk block.**

디스크는 block 기반 장치이므로, RP의 셀을 접근하는 비용이 disk block에 접근하는 비용과 관련이 있다.

**In this configuration, it would be preferred to set the overlay box size such that the corresponding region of RP fits exactly into a constant number of disk pages;**

이 구성에서, RP의 대응 영역이 디스크 페이지의 일정한 개수로 정확하게 부합되도록 overlay box 크기를 설정하는 것이 바람직할 것이다;

**both queries and updates will then require only a constant number of disk reads or writes.**

쿼리와 업데이트 모두 디스크를 읽고 쓰는 일정한 수를 필요로 할 것이다.

**When both the overlay and RP are stored on disk, one could still expect the optimal overlay box size to be somewhat larger than the formula predicts.**

Overlay 및 RP 모두 디스크에 저장되는 경우, 하나는 여전히 최적의 overlay box크기가 수식예측보다 다소 크게 기대된다.

**Since overlay boxes in general require less storage than the area of RP they cover, they can presumably be packed more efficiently into disk blocks.**

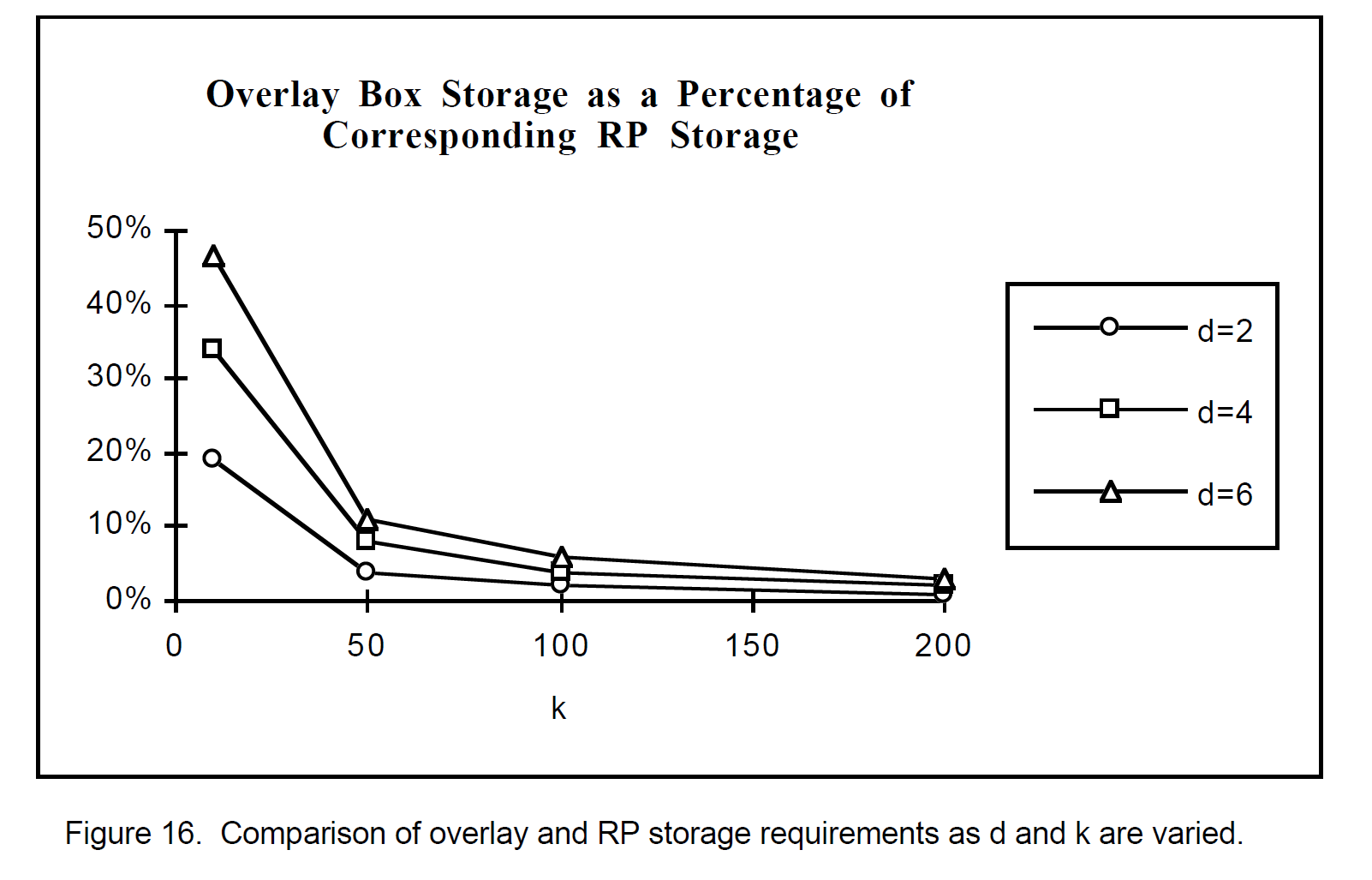
일반적으로 overlay box들은 RP의 영역보다 적은 저장공간을 요구하기 때문에, disk block 에서 효율적으로 압축 될 수 있다.

**Recall that updates only effect one region of RP, but potentially many overlay boxes.**

업데이트는 하나의 RP의 영역에 영향을 미친다. 그러나 많은 overlay boxes들에 영향을 미친다는 것을 기억하자.

**When overlay boxes are packed efficiently into disk blocks, the aggregate cost of accessing them should be less than the cost of accessing a region in RP**.

Overlay box들을 disk block에 효율적으로 압축할 때, 이들 접근의 총 비용은 RP 영역에 접근하는 비용보다 작아야 한다.



**5. Conclusion**

최근에 Range Sum Quries 의 범위 계산은 더욱 중요해지고 있다. <OLAP>와 <Data cube의 개발> 의 수요증가 때문이다. 수 많은 business applications 은 정기적을 최신화가 이루어지는 data cube를 필요로 한다. 매주 또는 매일 업데이트 되는 대용량 data cube에 대해 단순한 접근과 the prefix sum 접근방식으로는 처리하기 불충분하다. 단순한 접근방식은 queries 의 nd를 필요로 한다. 이 방법은 바람직하지 못하다. 그러나 오직 업데이트만 1만 필요하다. The prefix sum 방법도 단순방법과 똑같은 비용이 든다. 그러나 많은 차원을 가지고 있는 방대한 data cube에서는 업데이트 빈도에 심각한 제한을 받을 수 있다. The naive method 와 The prefix sum method에 대한 The product of the query와 업데이트 비용은 O(nd)이다.

The relative prefix sum method 는 range sum queries의 일정시간평가를 제공한다. 이 방법은 the prefix sum 접근 방법에서 마주하는 업데이트 문제들을 줄여준다. 그 결과 RP의 업데이트 복잡도는 O(nd/2)로 줄어든다.

the relative prefix sum method는 The product of the query 와 업데이트 비용은 O(nd/2)이다.

따라서 the relative prefix sum method는 The prefix sum method와 비교해서 업데이트 비용도 감소뿐만 아니라 range sum problem에 대한 전반적인 복잡도도 줄여준다..