OLAP 환경에서 다중점 MAX/MIN 질의의 효율적인 처리기법

(Efficient Processing of Multipoints MAX/MIN Queries in OLAP Environment)

양 우 석 † 김 명 호 ††

(Woo Suk Yang) (Myoung Ho Kim)

요 약 OLAP(Online analytical processing)은 의사지원시스템을 효과적으로 지원하기 위한 핵심 요소이며 주로 집단함수를 포함한 분석 질의를 처리한다. 이러한 질의를 효율적으로 처리하기 위한 연구들이 많이 이루어져 왔으나, 기존의 연구들은 어떤 범위 내의 모든 값을 대상으로 하는 집단함수의 처리 방법들을 다루고 있다. 그러나 이러한 범위 질의 외에도 범위 내의 특정 값들, 즉 다중점에 대한 질의도 많이 사용될 수 있으며, 이러한 질의에는 기존의 연구가 적용되기 어렵다. 본 논문에서는 다중점 MAX/MIN 질의를 효율적으로 처리하는 방법으로 순위 색인과 순위 결정 트리를 제안한다. 최대/최소값을 구하기 위해, 비트맵 형태의 노드로 이루어진 순위 결정 트리를 사용하여 결과의 순위를 구하고, 순위 색인을 통하여 질의의 결과를 얻는다. 그리고 실험을 통하여 제안한 방법이 대부분의 MAX/MIN 질의에 대해 안정적으로 높은 성능을 나타낸다는 것을 보였다. 또한, 단일 선계산 자료만으로 MAX와 MIN 질의를 모두 처리할 수 있다는 것도 제안한 방법의 주요 장점이다.

Abstract Online analytical processing (OLAP) systems are introduced to support decision support systems. Many researches focussed on efficient processing of aggregate functions that usually occur in OLAP queries. However, most previous researches in the literature are deal with the situation in which aggregate functions are applied to all the values in a given range. Since those approaches utilize characteristic of aggregate functions applied to a range, they are difficult to be applied to a multipoint query that is a query considering only some points in a given range. In this paper, we propose the Ranking Index and the Ranking Decision Tree (RDT) for efficient evaluation of multipoints MAX/MIN queries. The ranking of possible MAX/MIN values are computed with RDT. Then MAX/MIN values can be acquired from the Ranking Index. We show through experiments that our method provides high performance in most situations. In other words, the proposed method is robust as well as efficient. A single common set of precomputed results for both MAX and MIN values is another advantage of the proposed method.

1. 서 론

최근 들어 정보의 관리 및 그에 대한 분석의 중요성이 강조되고 있다. 각 기업들은 자사의 업무 내용을 분석하고 이해하기 위해서 데이타 웨어하우스(Data Warehouse)를 기반으로 의사 지원 시스템을 사용한다. 데이타웨어하우스 시스템을 사용하여 축적되는 자료는 기업의 업무 내용 대부분을 포함하는 데이타들의 거대한 집합이며, 의사 지원 시스템은 이들 데이타를 다양한 집단함수가 포함된 질의를 사용하여 짧은 응답 시간 안에 분석해서 가치 있는 정보로 가공해낸다. 과거 대부분의 관계형 데이타베이스 시스템들은 OLTP(On-Line Transaction Processing) 처리를 위해 설계되었기 때문에, 방대한 자료를 빠른 시간 내에 분석하기 위한 업무에는 적합하지 않았다. OLAP(On-Line Analytical

† 비 회 원 : 한국과학기술원 전산학과
wsyang@dbserver.kaist.ac.kr
†† 종신회원 : 한국과학기술원 전산학과 교수
mhkim@dbserver.kaist.ac.kr
논문접수 : 1999년 6월 1일
심사완료 : 1999년 11월 3일

Processing) 데이타베이스는 짧은 응답시간 안에 방대한 양의 자료를 분석하기 위한 요구를 만족시키기 위하여 설계되었다[1].

이미 여러 OLAP 시스템이 상용화 되었는데, 이들 시스템들은 특별한 인덱스를 사용하거나 예상되는 많은 수의 집단함수 질의에 대해 선 계산 함으로써 빠른 응답시간을 제공한다[2][3][4]. 본 논문에서 우리가 관심을 가지는 부분은 집단함수 질의 처리 방법이다. 일반적으로, OLAP 시스템에서는 데이타 큐브(data cube) 모델을 사용하여 질의에 포함되는 여러 애트리뷰트들을 차원(dimension)으로 표현하고, 각 차원에 대해 집단함수의 결과를 선 계산 해두는 방식을 사용한다[5][6][7]. 그러나, 많은 경우 차원 전체에 대한 집단 함수 외에도 특정 범위가 적용된 집단 함수 질의가 존재한다. 예를 들어 기온 정보 데이타에 대해 데이타 큐브가 "도시, 일자" 형태인 2개의 차원(애트리뷰트)으로 구성되어 있다고 하자. 차원별로 기온의 최대값을 선계산 해 놓았을 때 이를 이용하여 빠르게 처리할 수 있는 질의는 도시별로 현재까지 가장 높았던 기온을 표현하거나, 일자별로 전국에서 가장 높은 기온을 표현하는 질의, 그리고 전체 자료 중 가장 높은 기온에 대한 질의들뿐이다. 그러나 매년 12월중 가장 기온이 높았던 날에 대한 질의라든지, 충청지역에서 가장 기온이 높았던 날에 대한 질의 등 각 차원에 대한 범위가 주어지고 이에 따른 집단함수의 값을 요구하는 질의 등도 많이 사용될 수 있다. 이러한 전체가 아닌 일부분의 범위가 주어진 집단함수를 빠르게 처리하기 위해서 모든 범위에 대한 집단함수의 값을 선계산 해두는 것은 현실적으로 불가능하므로, 적당한 추가 저장 공간을 이용하여 효율적으로 사용될 수 있는 선계산 방법이 필요하다.

SQL 표준에서는 집단함수로서 SUM, AVG, COUNT, MAX, MIN 의 5가지를 지정하고 있다. 그리고 이 함수들은 그 특성에 따라 SUM 계열과 MAX 계열로 나눌 수 있다. AVG는 SUM과 COUNT를 사용하여 구할 수 있고, COUNT는 SUM의 특별한 형태로 볼 수 있다. 또, MAX와 MIN은 최대치를 구하는 집단 함수로서 같은 특성을 가진다. 그래서, 대부분의 연구들이 SUM과 MAX에 초점을 두고 수행되었으며, 이들을 다루는 방법은 무리 없이 AVG, COUNT 나 MIN 등의 다른 집단함수 문제의 해결에도 사용될 수 있다.

연속된 영역을 범위로 하는 집단함수의 처리에는 각 집단함수가 영역에서 보여지는 특성들을 사용하여 빠르게 계산되는 방법들이 제시되었다. [8] 은 연속된 영역을 가지는 범위에 대한 SUM 함수를 전방 부분 합 배열(prefix sum array)을 사용하여 O(1) 시간으로 계산할 수 있는 방법을 제시하였다. 이 방법은 질의 영역에 대한 합을 전방 부분 합 배열로 미리 계산 해둔 몇 개의 값을 더하거나 빼서 구하는 것으로 영역에 대한 합의 특성을 이용한 것이다. [9]는 최대 포함영역(maximal cover) 개념을 사용하여 연속된 영역에 대한 MAX 질의를 효율적으로 처리하는 방법을 제시하였다. MAX 값이 영향을 미치는 범위를 선계산하여 계층 트리 형태로 표현하고, 영역 질의가 주어졌을 때 질의 영역을 포함하는 계층 트리를 검색하여 MAX 값을 구해내는 방법이다.

그러나, 이들 논문에서 제안된 방법은 연속된 영역을 범위로 하는 질의에 대한 집단함수의 특성을 사용하여 질의를 처리하기 때문에, 연속되지 않은 영역에 있는 데이타들, 즉 비연속 다중점(multipoints)에 접근해야 하는 질의에는 적용하기 어렵다. 예를 들어, 매주 토요일에 대한 결과를 원하는 질의나 전국의 모든 도시에 대한 정보를 가지고 있는 데이타베이스에서 광역시에 대한 질의등에는 적용될 수 없다. 이러한 비연속 다중점 질의는 연속 영역 질의에서 보여진 집단함수의 특성을 보이지 않기 때문에 연속 영역 질의 처리와는 다른 방법이 필요하다.

[10] 은 비연속 다중점에 대한 SUM 질의를 효율적으로 처리하는 방법을 제시하였다. 다중점 질의에 대해 에러수정코드(error correcting code)에서 사용되는 포함 코드(covering code)를 사용하여 다중점 질의에서 참조해야 하는 셀 수보다 적은 수를 사용하여 해당 질의의 SUM 값을 구하는 방법이다. 이 방법은 합에 대해서만 적용되며, 합과는 특성이 다른 MAX 질의에는 적용될 수 없다.

본 논문에서는 비연속 다중점 질의에 대한 MAX와 MIN을 효율적으로 계산하는 방법으로 순위 결정 트리(Ranking Decision Tree)를 제안한다. 순위 결정 트리는 MAX/MIN 질의에서 필요한 순서 정보만을 비트맵으로 요약해서 관리하므로, 보다 적은 수의 디스크 접근으로 결과값의 순위 구간을 결정하고 순위 색인을 사용하여 비연속 영역 범위 MAX/MIN질의를 처리한다. 본 논문의 구성은 다음과 같다. 2장에서는 다중점 질의 문제를 정의하고 데이타들의 순서 관계를 표현하기 위한 구조인 순위 색인을 기술하며, 3장에서 다양한 질의 분포에 대해서도 안정적인 성능을 보이는 순위 결정 트리를 제안한다. 그리고, 제안된 기법의 분석과 성능 평가 결과를 4장에서 기술한다. 마지막으로 5장에서 결론을 맺는다.

2. 순위 색인(Ranking Index)

일 차원 비연속 다중점 MAX/MIN을 구하는 문제는 다음과 같이 정의된다.

0 부터 m-1 까지 색인되어 있는 크기 m 의 배열을 A 라고 하자. 그리고 집합 $M = \{ 0, 1, \cdots, m-1\}$ 은 A의 색인 도메인이다. 질의는 A 의 색인 도메인 M의 부분 집합인 I로 표현된다. 예를 들어 크기가 k 인 질의 I는 $\{ i_1, i_2, \cdots, i_k\}$ 이며 i_j 는 A에 있는 셀의 색인이다. 이 때, 질의 I 에 의해 선택된 A의 셀들 중 최대 (또는 최소값)은 $\text{MAX}_{i \in I}(A[i])$ (또는 $\text{MIN}_{i \in I}(A[i])$) 이다.

MAX와 MIN의 결과는 질의 범위에 있는 셀의 값이 결과 값들 중에서 가장 크거나 작은 값만이 의미를 갖는다. 따라서, 어떠한 값이 질의 범위 안의 나머지 셀들의 값보다 크거나 작다는 사실을 알 수 있다면, 나머지 셀들은 조사할 필요가 없다. 전체 데이타에 대한 크기 순서를 표현하는 방법으로 다음과 같은 순위 색인 구조를 정의한다.

정의 1. 1차원 데이타 큐브를 표현한 크기 m인 배열 A의 셀들을 내림차순으로 정렬했을 때, i 번째에 위치하는 셀을 $A[k_i]$ 라 하자. 이 때 $R[i]$ 의 값이 k_i인 배열 R을, $0 \le i \le m$-1, A의 *순위 색인* 이라 정의한다. ■

관찰 1. 주어진 크기 m 인 배열 A의 순위 색인 R 이 있을 때, 모든 $0 \le i < m$-1 인 정수 i 에 대해 $A[R[i]]$ 는 $A[R[i+1]]$ 보다 크거나 같다. ■

그림 1은 12개의 셀을 가진 배열 A에 대한 순위 색인 R을 보인다. 이 예에서 $A[1]$이 가장 큰 값인 99를 가지고 있으므로 $R[0]$은 1 이며, 두 번째로 큰 값은 $A[7]$인 84이므로 $R[1]$은 7, 그리고 제일 작은 값은 $A[9]$인 14이므로 $R[11]$은 9 가 됨을 보인다.

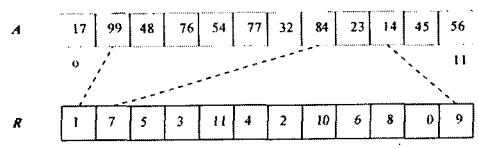

그림 1 순위 색인의 예

관찰 2. 1차원 데이타 큐브를 표현한 배열 A 에 대하여 질의 I 가 주어질 때, $R[k] \in I$ 이고, k보다 작은 모든 정수 j에 대해서 $R[j] \notin I$ 이면 질의 I 에 대한 최대값은 $A[R[k]]$ 이다. ■

관찰 3. 1차원 데이타 큐브를 표현한 배열 A 에 대하여 질의 I 가 주어질 때, $R[k] \in I$ 이고, k보다 큰 모든 정수 j에 대해서 $R[j] \notin I$ 이면 질의 I 에 대한 최소값은 $A[R[k]]$ 이다. ■

관찰 2와 3에 따라, 주어진 질의 I에 대한 최대값 또는 최소값은 순위 색인 배열을 순차적으로 검사하여 구할 수 있다.

3. 순위 결정 트리 (Ranking Decision Tree)

순위 색인을 순차적으로 검사하여 최대값 또는 최소값을 찾는 방법은 질의의 성격에 따라 순위 색인 전체를 모두 검사해야 하는 경우가 발생한다. 주어진 질의에 대한 최대값 또는 최소값의 전체 중 순위를 효과적으로 찾을 수 있다면 순위 색인을 통하여 바로 결과값을 찾을 수 있다. 결과값에 대한 전체 데이타상의 순위인 *rank-max*와 *rank-min*을 다음과 같이 정의한다.

정의 2. 1차원 데이타 큐브를 표현한 배열 A 에 대하여 주어진 질의 I 가 있다고 하자, *rank-max*(I)는 I 에 의해 선택되는 셀 중에서 최대값을 가지는 $A[R[i]]$ 의 i이며, *rank-min*(I)는 최소값을 가지는 $A[R[i]]$ 의 i이다. 만약, i가 여러 개 존재할 때는 (즉 동일한 최대값/최소값이 여러 개 있을 때는) *rank-max*(I)는 이 중 가장 작은 색인 i이며, *rank-min*(I)는 가장 큰 색인 i이다. ■

관찰 4. 배열 A에 대하여 순위 색인 R과 주어진 질의 I 가 있을 때, 질의에 대한 최대값은 $A[R[$*rank-max*(I)$]]$ 이며, 최소값은 $A[R[$*rank-min*(I)$]]$ 이다. ■

관찰 4에 의해, 순위 색인 구조가 있을 경우 질의에 대한 *rank-max*나 *rank-min*값을 구하면, 바로 최대값, 최소값을 구할 수 있음을 알 수 있다. 즉, 다중점 MAX/MIN 을 구하는 문제를 질의 결과의 순위를 구하는 문제와, 순위 정보를 사용하여 실제 결과를 얻는 문제로 나눌 수 있다. 이 장에서는 주어진 질의에 대한 최대값의 전체 데이타 상의 순위를 효율적으로 결정하기 위해 순위 이분 요약으로 이루어진 순위 구간 결정 트리를 제안하고, 이 방법을 수행하기 위한 비용을 분석한다. 최소값 구하기는 특성상 최대값 구하기와 유사하므로, 설명의 편의상 먼저 주로 최대값에 대해서 기술하고, 최소값에 대해서는 후반부인 3.3절에서 별도로 언급한다.

3.1 순위 이분 요약 (Ranking Bisection Signature)

검색에 필요한 질의와 데이타의 특성을 보다 효율적으로 표현하기 위하여 다음과 같이 비트 벡터 표현을 사용한다.

길이 m인 비트 벡터 $V = (b_0 b_1..b_{m-1})$는 m개의 비트들로 구성되어 있으며 각 비트 b_i의 값은 "0" 이나 "1" 이다. 이때 비트 벡터의 *무게(weight)* 는 값이 "1" 인 비트들의 개수이다. 크기 m인 배열 A에 대한 질의 I는 0 과 m-1 사이의 중복되지 않은 값들로 이루어 진다. 그러므로, 이 집합은 길이가 m 인 비트 벡터로 표현될 수 있다. 질의 I 를 표현하는 비트 벡터 V_I 는 (이후 질의 벡터라고 부르겠다) 다음과 같이 정의된다.

질의 벡터 $V_I = (b_0 b_1..b_{m-1})$ 에서, $k \in I$ 인 모든 정수 k 에 대해 $b_k = 1$이고 그렇지 않을 경우 $b_k = 0$ 이다.

데이타 웨어하우스나 OLAP데이타베이스 등은 고속의 데이타베이스 질의 처리가 필요하므로 비트맵 인덱스(bitmap index)와 비트맵 조인 인덱스(bitmap join index)가 개발되어 현재 Oracle, RedBrick, Sybase등의 많은 상용 DBMS에서 지원하고 있다[11] . 비트맵 인덱스나 비트맵 조인 인덱스를 사용하면 전체 테이블을 검색하지 않고 비트맵 데이타들에 대한 비트 연산으로 필요한 튜플들의 리스트를 하나의 비트 벡터로 표현할 수 있다. 비트맵 인덱스와 비트맵 조인 인덱스가 대량의 데이타에 대한 고속의 질의 처리에 효과적이라고 알려져 있으므로[3], 이러한 질의 처리 과정에서 얻어진 비트 벡터를 본 논문에서 언급한 질의 벡터로 사용한다.

순위 이분 요약은 크기가 m인 데이타 배열 A에 대해 순위별로 1등에서 $\lceil m/2 \rceil$등까지의 값들과 $\lceil m/2 \rceil$+1등에서 m등까지의 값들을 구분하는 이진 요약코드로 다음과 같이 정의된다.

정의 3. 크기 m 인 데이타 배열 A에 대한 순위 색인 배열 R이 있을 때, $0 \; k < \lceil m/2 \rceil$ 인 모든 정수 k 에 대하여 $b_{R[k]} = 1$, $\lceil m/2 \rceil \le j < m$ 인 모든 정수 j 에 대하여 $b_{R[j]} = 0$ 인 비트 벡터 $V_S = (b_0 b_1..b_{m-1})$ 을 A의 *순위 이분 요약(RBS)*이라 한다. ■

관찰 5. 질의 벡터 V_I 에 대하여 상위 50% 순위에 해당되는 질의를 표현하는 길이가 m인 비트 벡터 V_I^{upper} 는 $V_I \wedge V_S$이며, 하위 50% 순위에 해당되는 비트벡터 V_I^{lower}는 $V_I \wedge \sim V_S$이다. (여기서 $\wedge$ 는 이진곱(AND) 연산이며 ~ 는 1의 보수를 나타내는 연산이다) ■

그림 2는 크기 m이 14 인 데이타 집합에 대하여 질의 I가 주어졌을 때, 순위 이분 요약 코드 V_S와 질의 벡터 V_I , 그리고 두 벡터의 연산의 결과인 V_I^{upper} , V_I^{lower} 의 예를 보인다. V_I^{upper} 의 무게가 1 이상이므로, 질의 I의 최대값은 상위 50%이내라는 것을 알 수 있다. 상위 50%에 해당되는 인덱스 I_{upper} = {3,13}, 하위 50%에 해당되는 인덱스 I_{lower} = {8, 10, 11} 를 주어진 비트 벡터에 대해 단순한 이진연산을 사용하는 것만으로 알 수 있다.

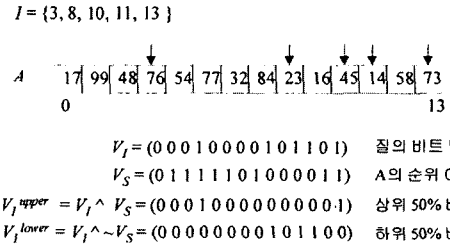

그림 2 순위 이분 요약의 예

3.2 순위 결정 트리(Ranking Decision Tree)

순위 결정 트리는 전체 데이타 상에서 주어진 질의에 대한 최대값의 순위를 효율적으로 알아내기 위한 방법이다. 기본적인 개념은 순위 이분 요약을 균형 이진 트리로 구성하여, 하위 레벨의 노드일수록 더욱 좁은 범위의 순위를 결정하는 순위 이분 요약을 배치하고, 이를 탐색하여 세부적인 순위를 결정하도록 하는 것이다. 그러나, 앞서 정의한 순위 이분 요약은 데이타 배열의 크기와 같은 길이의 비트 벡터가 필요하다. 같은 크기의 순위 이분 요약을 사용하여 트리를 구성한다면 깊이가 N 인 이진 트리일 경우 $m \times 2^{N-1}$비트의 선 계산 공간이 필요하고 이진 검색 시에도 $m \times N$ 비트의 정보를 읽어야만 한다. 그러나, 하위 레벨에 있는 RBS의 경우, 이미 상위 레벨에 있는 RBS에서 순위 구간의 일부가 결정되므로, 상위 레벨의 RBS에 의해 결정된 구간에 대한 RBS를 다음과 같이 정의하여 크기가 상위 레벨 RBS의 반인 RBS로 표현이 가능하다.

정의 4. 크기 m 인 데이타 배열 A 대한 순위 색인 배열 R이 있을 때, 주어진 순위 범위 $(l:u)$, $u > l$ 에 대해 배열 $A_{l:u}$ 는 $A[R[k]]$, $l \le k \le u$ 인 원소들로 구성되며, 배열 A 에서의 순서를 유지하는 크기 u-l+1 인 배열이다. 이 때, 주어진 순위 범위 $(l:u)$ 에 대한 순위 이분 요약 $RBS_{l:u}$ 은 $A_{l:u}$ 에 적용된 순위 이분 요약이다. ■

정의 5. 크기 $m > 1$인 데이타 배열 A에 대하여, 순위 결정 트리는 다음과 같이 정의된다.

1) 트리의 루트 노드는 $RBS_{0:m-1}$ 이다.
2) 트리의 리프 노드는 u-l = 1 인 $RBS_{l:u}$ 이다.
3) $RBS_{l:u}$인 노드에 대해, $RBS_{l:u}$ 의 무게를 w라 하면 왼쪽 자식 노드는 $RBS_{l:l+w-1}$ 이다. (모든 RBS에서 "1" 의 개수, 즉 무게는 그 RBS크기의 반에

해당하므로 $l+w-1$은 구간 $(l:u)$를 반으로 나누는 지점이 된다.)

4) $RBS_{l:u}$인 노드에 대해, $RBS_{l:u}$ 의 무게를 w라 하면 오른쪽 자식 노드는 $RBS_{l+w:u}$ 이다. 단 $u-l$ = 2 일 경우 오른쪽 자식 노드는 없다. ■

그림 3은 순위 결정 트리의 예를 보인다. 레벨 2 의 왼쪽 노드는 원 데이타 배열중 상위 50% 에 해당하는 데이타를 순서를 유지시킨 상태로 추출하여 그 중 상위 50% 에 해당하는 위치에 비트 1을 표기한 형태이다. 즉, 상위 50% 의 데이타 중 상위 25% 에 해당되는 정보의 위치를 표현한 순위 이분 요약이다.

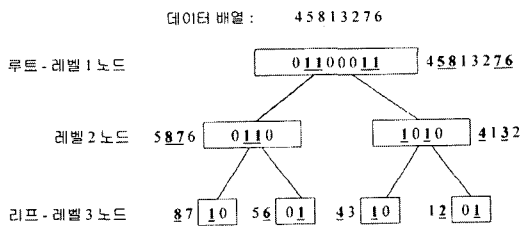

그림 3 순위 결정 트리의 예

최대값을 찾는 질의 I에 대해 순위 결정 트리를 탐색하는 과정은 상위 노드에서 결정된 V_I^{upper}, V_I^{lower} 비트 벡터를 하위 노드의 질의 벡터로 사용하는 것이다. 그러나, 하위 노드의 순위 이분 요약은 전체 데이타에 대한 요약 정보가 아닌 상위 노드에 의해 양분되어 제한되어진 순위 범위내의 요약 정보이므로, 질의 벡터도 이에 맞게 변환하여야 한다. 다음 정의 6에서 비트 벡터 연산자 HALF-SELECT를 정의한다. 그림 4는 HALF-SELECT 연산자의 수행 예이다.

정의 6. 주어진 비트 벡터 V와 V_S에 대하여 HALF-SELECT(V, V_S) 는 V_S 의 비트 벡터가 1인 위치의 V 의 비트를 원래 순서를 유지한 상태로 가지며, 길이가 V_S 의 무게인 비트벡터를 생성시키는 연산자이다. ■

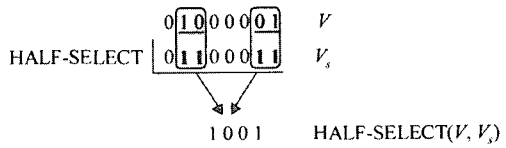

그림 4 HALF-SELECT 연산자의 수행 예

한 노드의 순위 이분 요약 V_S 에 따라 V_I^{upper}와 V_I^{lower} 가 구해졌을 때, 왼쪽 자식 노드에 적용되는 질의 벡터는 HALF-SELECT(V_I^{upper}, V_S) 이며, 오른쪽 자식노드에 적용되는 질의 벡터는 HALF-SELECT(V_I^{lower}, $\sim V_S$)이다. 다음 그림 5는 주어진 질의 I 에 대해 순위 결정 트리를 사용하여 최대값의 전체 데이타 상의 순위인 *rank_max(I)* 를 구하는 알고리즘이다.

```
    GetRankMax(I)
    I : query index
    begin
1)      V_I = ConvertBitVector(I);
2)      return(Search_RD_Tree(V_I, Root RBS of RD_Tree, 0))
    end

    Search_RD_Tree(V_q, V_S, L)
    V_q : query bit vector
    V_S : ranking bisection signature
    L : lower rank of RBS
    begin
1)      V_q' = V_q ^ V_S;
2)      if weight(V_S) = 1 and weight(V_q') = 1 then
          return L;
3)      if weight(~V_S) = 1 and weight(V_q') = 0 then
          return L+weight(V_S)
4)      if weight(V_q') > 0  then
5)        val = Search_RD_Tree(HALF_SELECT(V_q',V_S),
                    Leftchid RBS, L)
6)      else
7)        val = Search_RD_Tree(HALF_SELECT(V_q'~^V_S,
                    ~V_S), RightChild RBS, L+weight(V_S))
8)      return val;
    end
```

그림 5 *rank_max*(I) 를 구하는 알고리즘

알고리즘 설명 :

GetRankMax의 1) 은 질의를 질의 비트 벡터로 바꾸며, 2)에서 재귀함수의 형태로 순위 결정 트리를 검색하는 함수를 사용하여 순위를 결정한다. 함수의 초기 값으로 질의 벡터와 순위 결정 트리의 루트 노드, 그리고 함수가 결정하는 순위 범위의 시작 값을 사용한다.

Search_RD_Tree는 재귀 함수로서 순위 결정 트리를 백 트래킹 없이 단일 경로로 탐색한다. 재귀 함수의 종료 조건은 마지막 리프 노드에 도달했는가이며 2)와 3)이 나타내고 있다. 2) 는 V_I^{upper}의 무게가 1 이상인 경우로서 결과는 순위 범위의 시작 값을 가지게 되며, 3)은 V_I^{upper}의 무게가 0이고 V_I^{lower}의 무게가 1 이상인 경우이며 하위의 순위 범위 시작 값을 가진다. 5)는 V_I^{upper}의 무게가 1 이상일 때 순위 범위를 상위로 더욱 좁혀서 탐색을 진행하며, V_I^{upper} 의 무게가 0일 때는 7) 에서 순위범위를 하위로 좁혀서 탐색을 진행하게 된다. 다음 레벨의 질의 벡터를 구할 때 그 레벨에서의 질

의벡터 V_I와 그 레벨에서의 RBS 인 V_S 의 논리곱, 즉 $V_I \wedge V_S$의 무게가 0일 때는 $V_I \wedge \tilde{}V_S$ 를 이용한다는 사실에 주의하라. 그림 6는 그림 3에서 예로 들은 데이타 배열 및 순위결정 트리를 사용하여 주어진 질의 벡터에 대해 순위를 결정하는 과정을 보인다. 이 예에서는, 앞서 설명한 알고리즘을 사용하여 단일 경로로 순위 결정 트리를 탐색하여 최대값이 순위 범위 (0:7) 중에서 2 순위임을 알아낸다. 결국, 위 질의 벡터에 대한 최대값은 $R[2]$ 인 6 이다.

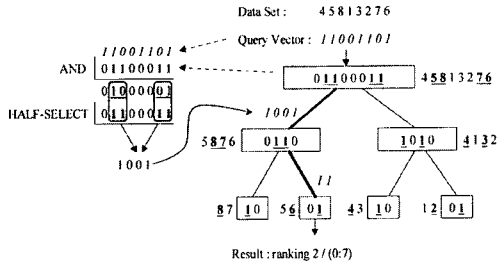

그림 6 순위 결정 트리를 통한 순위 결정 과정의 예

관찰 6. 하위 노드의 RBS 크기를 n_{child}, 상위 노드 RBS 의 크기를 n_{parent} 라 할 때, $n_{child} \leq \lceil n_{parent}/2 \rceil$ 이며 하위 두 노드의 RBS 크기의 합은 n_{parent} 이다. ■

관찰 7. 크기 m인 데이타 배열에 대한 순위 결정 트리의 깊이는 $\lceil \log_2 m \rceil$ 이다. ■

관찰 6,7에 의해 앞에서 기술한 순위 결정 트리는 최대 $m\lceil \log_2 m \rceil$ 비트의 선계산 공간을 차지하며 순위를 정확하게 알아내기 위하여 이진 검색시 약 $2m$ 비트의 정보를 읽어야 한다. 하지만, 실제 구현 시 순위 결정 트리는 디스크에 존재하게 되며, 한 노드의 정보를 읽기 위하여 한번의 디스크 접근 비용이 필요하다면, 하위 레벨의 작은 크기의 순위 이분 요약을 탐색하는 과정은 효율성이 떨어진다. 다음 절에서는, 이를 고려하여 순위 결정 트리를 생성하고 이에 따라 최대값을 구하는 방법을 기술하고 성능을 분석한다. 그리고, MAX 를 위해 선계산 되어진 순위 결정 트리와 순위 색인을 사용하여 MIN 문제에 적용하는 방법에 대해서 언급한다.

3.3 구현 및 분석

3.3.1 변형된 순위 결정 트리

순위를 정확하게 결정하기 위해서 필요한 선계산 공간과 디스크 접근 횟수를 고려할 때, 순위 결정 트리를 일부 변형할 필요가 있다. 우선 리프 노드를 한 블록 정도 크기를 가진 노드가 되도록 순위 결정 트리의 정의 5의 2) 항을 다음과 같이 수정하고 이를 *순위 구간 결정 트리*라 부른다.

2) 한 블럭의 비트 수를 h라 할 때, 트리의 리프 노드는 $RBS_{l:u}$, $u-l < h$ 이다.

그리고, 아래의 코드는 이에 따라 그림 5의 알고리즘에서 재귀 함수의 종료 시점을 RBS 가 한 블록 미만인 경우로 바꾼 것이다.

2) if $|V_S| <= h$ and $weight(V_q') > 0$ then return L;

3) if $|V_S| <= h$ and $weight(V_q') = 0$ then return L+weight(V_S)

그림 7은 이러한 구조를 그림으로 도식화 한 것이다. 순위 구간 결정 트리는 리프노드가 한 개의 블록으로 표현되는 깊이까지 구성된다. 이 때, 리프 노드에 의해서 결정되는 순위 구간은 $h/2$의 크기를 갖는다. 이 순위 구간에 대해 순위 색인을 순차적 검색을 통하여 실제 MAX 값을 찾는다.

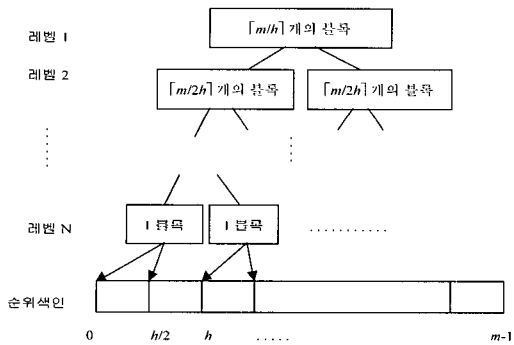

그림 7 순위 구간 결정과 순위 색인

3.3.2 분석

관찰 8. 한 블록의 비트 수를 h라 하고, 데이타 배열의 한 셀이 c바이트의 저장공간을 차지한다면, 크기 m인 데이타 배열을 저장하기 위한 공간 M은 cm바이트가 필요하고, 순위 구간 결정 트리를 위해 필요한 선 계산 공간의 크기는 $(\lceil \log_2 M - \log_2 ch - 1 \rceil / 8c)M$바이트이다.

관찰 9. 한 블록의 비트 수를 h라 하고, 순위 색인의 한 색인이 i 바이트의 저장공간을 차지한다면 크기 m인 데이타 배열에 대한 순위 구간 결정 트리가 있을 때, 최대값을 찾기 위한 디스크 접근 비용은 최대 $2\lceil m/h \rceil + 4i - 1$ 블록, 최소 $\lceil 2m/h \rceil$ 블록이 된다. ■

관찰 8에서 데이타 크기가 증가할수록 필요한 선계산 공간의 크기 비율도 증가함을 알 수 있다. 일반적인 상황에 맞추어, 한 블럭의 크기 h를 32768비트(4k바이트)

라 하고, 데이타 배열의 한 셀은 4 바이트를 차지한다고 하자. 식에 대입하면 100만개 정도의 데이타에 대하여 원래 데이타의 18.7%의 추가 선계산 공간이 필요하고 1억개 정도로 증가하면 38% 정도의 추가 선계산 공간이 필요하다. 관찰 9는 검색 성능을 다루고 있다. 한 블록의 비트수를 32768 (즉 4k바이트)라 하고 색인은 4 바이트의 크기를 갖는다고 하자. 1024k개 가량의 데이타에 대해 최대값을 구하려 할 때 최악의 경우 79 블록을 읽고 최선의 경우 64 블록만을 읽고 결과를 얻을 수 있다. 데이타 배열의 한 셀이 4바이트를 차지한다고 하면, 전체 데이타가 1024 블록의 크기를 가지는 것과 비교할 때 디스크 접근 비용 면에서 최고 16배 적다.

3.3.3 MIN 문제의 적용

순위 색인 구조와 순위 이분 요약, 그리고 순위 결정 트리는 모두 대칭(symmetric) 형태를 가지고 있다. 순위 색인 구조의 경우 최대값을 찾기 위한 순차적 검색의 반대방향 검색을 수행하면 최소값을 구할 수 있고, 순위 이분 요약은 1의 보수를 취했을 때 상, 하위 순위를 바꿀 수 있다. 또한 순위 결정 트리 역시 좌우를 바꾸어 검색함으로써 최대값을 위한 선계산 자료를 사용하여 최소값을 찾을 수 있다. 그림 8은 최소값의 전체 데이타들 중의 순위를 결정하는 알고리즘이다. 이와 같이, 최소값 구하기를 위한 별도의 선계산 자료 없이, 대칭적인 알고리즘을 최대값을 위해 생성된 선계산 자료들에 적용하여 최소값을 구할 수 있다.

```
    GetRankMin(I)
    I : query index set
    begin
1)      V_I = ConvertBitVector(I);
2)      return(Search_RD_Tree_for_Min(V_I, Root RBS of
        RD_Tree, m-1))
    end

    Search_RD_Tree_for_Min(V_q, V_S, u)
    V_q : query bit vector
    V_S : ranking bisection signature
    u : upper rank of RBS
    begin
1)      V_q' = V_q ^ ~V_S;
2)      if weight(~V_S) = 1 and weight(V_q') = 1 then
          return u;
3)      if weight(V_S) = 1 and weight(V_q') = 0 then
          return u - weight(~V_S)
4)      if weight(V_q') > 0  then
5)        val = Search_RD_Tree_for_Min(HALF_SELECT
              (V_q',~V_S), Rightchild RBS, u)
6)      else
7)        val = Search_RD_Tree_for_Min(HALF_SELECT
              (V_q^V_S,V_S), LeftChild RBS, u - weight
                  (~V_S))
8)      return val;
    end
```

그림 8 *rank_min(I)* 를 구하는 알고리즘

4. 성능 평가

이 장에서는 앞에서 제안한 방법들에 대해 실험을 통하여 성능을 비교한다. 실험 데이타로서 균일 분포를 갖는 랜덤 함수를 이용하여 1048576개의 데이타[1]를 생성하였다. 이미 서론에서 설명한 바와 같이 기존의 연구들은 비연속 다중점 MAX/MIN질의에 적용할 수 없으므로 이 실험에서는 순위 색인을 순차 검색 하는 방법(Ranking Index), 순위 결정 트리를 사용하여 순위 구간을 결정하고 순위색인을 통해 최대값을 구하는 방법(RDT+Ranking Index), 그리고 MAX나 MIN을 구하려는 차원 애트리뷰트에 대해 프로젝션 인덱스[3]를 사용하는 방법(Projection Index)이 비교되었다. 일반적인 상황에 맞추어 한 블록의 크기는 4096 바이트이며 하나의 데이타는 4 바이트의 저장공간을 차지하는 것으로 설정되었다.

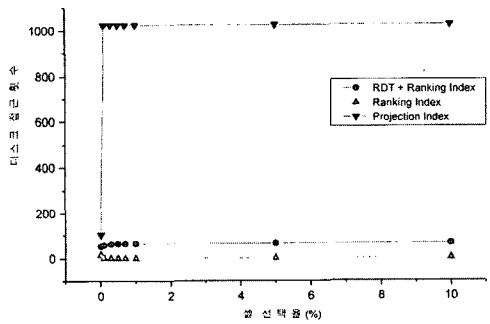

그림 9 균일 분포 질의일 때 셀 선택율에 대한 비용 그래프

그림 9는 균일분포를 갖는 랜덤함수를 이용하여 질의를 생성하고, 생성된 질의의 크기에 대한 각 알고리즘의 비용을 그래프로 표현한 것이다. x축은 전체 데이타에 대한 질의의 셀 선택율을 나타내며, 선택율 10% 이상은

1) 데이타의 크기인 1048576은 실험의 편의성 및 직관적인 평가 결과를 보이기 위하여2 의 n제곱수 중 실험에 적당한 값을 선택한 것이다. 제안된 알고리즘 및 분석은 데이타 개수의 제약이 없으며, 관찰 9에 따라 데이타의 개수가 2의 n 제곱수가 아닌 경우에도, 성능평가 결과는 크게 달라지지 않는다.

변화가 없으므로 그래프에서는 10% 미만을 선택 표현하였다.

셀 선택율이 0.1% 이상일 경우 질의가 적용되는 값을 클러스터링 하여 최대값을 구하는 방법은 거의 모든 디스크 블록을 읽어야 하지만, 본 논문에서 제안한 두가지 다른 방법들은 16배 이상 적은 디스크 블록 접근 비용을 보인다. 특히, 순위 색인의 경우 셀 선택율이 0.1% 이상이라면 한번의 디스크 접근 만으로 최대값을 구할 수 있다. 그러나, 위 실험은 질의가 데이타와 관계 없는 균일한 분포의 셀을 선택하였을 때의 결과로서, 실제로 적용되는 질의에 대한 결과를 나타낸다고 볼 수는 없다.

질의가 자료의 특성과 연관된 경우를 생각할 때, 질의에 의해 요구되는 최대값이 전체 데이타 중 매우 낮은 값 중의 하나가 될 수 있다. 예를 들어, 겨울 중 가장 더웠던 날의 기온이라든지, 여러 물품들을 판매하는 가게에서 겨울동안 에어컨을 가장 많이 팔았던 기록을 요구하는 것처럼 전체 데이타들에 비교할 때 매우 낮은 값들을 가지는 셀들에 대한 질의가 주어질 수 있다. 이러한 상황에서 순위 색인 만을 사용할 경우 앞에서 언급한 바와 같이 순위 색인의 상당 부분을 순차 검색해야 하므로 순위 결정 트리의 역할이 크게 중요해 진다. 다음 그림 10은 전체 데이타에 대한 질의 결과값의 순위에 따라 해당 질의에 대한 처리 비용을 표현한 것이다. 여기서 질의의 크기는 인덱스 1000 개로 실험하였다.

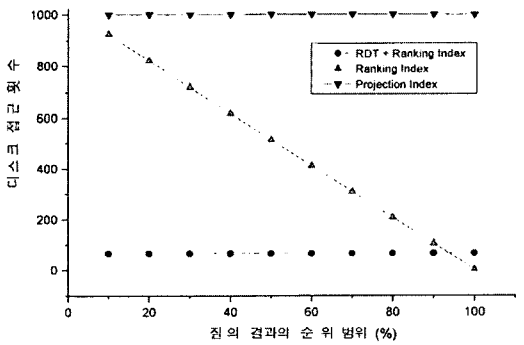

그림 10 최대값이 특정 범위로 제한된 질의에 대한 비용

실험 결과로 순위 색인을 순차적 검색하는 방법은 낮은 순위를 최대값으로 가지는 질의에 대해 좋지 않은 성능을 보이는데 비해 순위 구간 결정 트리를 사용하는 방법은 처리 비용이 결과의 순위에 대해 거의 영향을 받지 않는다는 것을 알 수 있다. 즉, 안정된 성능을 보이기 위해서는 순위 결정 트리가 사용되어야 한다. 그리고, 순위 결정 트리를 사용한 방법은 다른 방법들에 비해 최고 16배 빠른 성능을 보인다.

이러한 선계산 구조를 위하여 필요한 저장 공간의 크기를 살펴본다. 그림 11은 프로젝션 인덱스를 사용하는 경우와 비교한 순위 결정 트리와 순위 색인을 위한 선계산 공간의 추가 부담을 전체 데이타 갯수에 대해 표현한 그림이다. 약 100만개의 데이타 갯수에 대해서도 18% 정도의 추가 저장공간이 필요하다. 데이타의 크기가 클수록 추가 저장 공간의 비율이 더 커지는 특성을 보이고 있기는 하지만, 샘플 데이타에 비해 100배 큰 데이타(약 1억개)에 대해서도 39%의 추가 저장 공간이 필요한 정도이다.

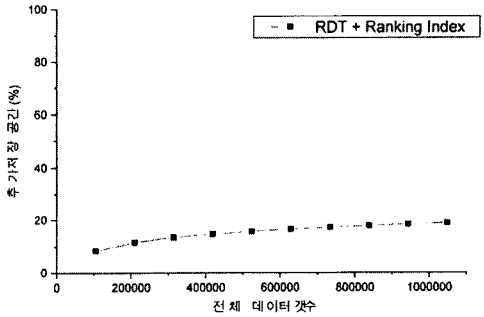

그림 11 데이타 갯수에 대해 선계산에 필요한 추가 저장 공간 비율

5. 결 론

OLAP 시스템에 주어지는 분석질의는 대부분 집단함수를 포함한다. 보통 자료 분석은 데이타 집합 전체를 대상으로 하는 것 외에도 특정한 조건을 가지는 데이타 집합에 대해 적용될 수 있으며, 이러한 경우 집단함수는 이 조건을 만족하는 범위의 데이타 집합에 대해 적용된다. 조건에 맞는 데이타의 범위는 숫자에 대한 조건일 경우 연속된 영역일 가능성이 있지만, 그렇지 않은 경우 대부분은 비연속 다중점 질의 형태를 가질 수 있다.

본 논문에서, 우리는 비연속 다중점 질의에 대한 MAX와 MIN 집단함수를 효율적으로 처리하는 방법을 제안하였다. 데이타들의 순서 관계를 표현하기 위한 구조인 순위 색인(ranking index)을 정의하고, 이를 순차적으로 검색하는 방법으로 최대/최소값을 구할 수 있음을 보였다. 그러나, 이 방법은 균일하게 분산된 다중점 질의에 대하여 좋은 성능을 보이지만 낮은 순위의 자료

들에 대한 최대값을 구하려 하는 질의에 대해서는 매우 안 좋은 성능을 보인다. 순위 이분 요약(ranking bisection signature)으로 구성된 순위 결정 트리(ranking decision tree)는 주어진 질의에 대한 결과값의 전체 데이타 상의 순위를 효율적으로 결정할 수 있는 구조이다. 비트맵 형태인 순위 이분 요약을 사용하여 질의를 상위 순위의 질의와 하위 순위의 질의로 구분하며, 순위 이분 요약을 균형 이진트리(balanced binary tree)로 구성함으로써 질의의 결과가 어떠한 순위 구간에 존재하는지를 알 수 있다. 순위 구간이 결정되면, 순위 색인을 사용하여 보다 적은 비용으로 질의의 결과를 얻는다. 실험을 통하여 우리가 제안한 기법이 모든 형태의 MAX 질의에 대해서도 안정적(robust)인 성능을 보임을 보였다. 100만 여건의 자료에 대하여 수행된 실험 결과로 MAX나 MIN을 구하려는 차원 애트리뷰트에 대해 프로젝션 인덱스를 사용하는 방법에 필요한 저장 공간 비용에 비해 18%만을 추가 사용하면서, 약 16배 적은 디스크 접근 비용을 사용함을 보였다. 질의가 실험에서 가정한 것 보다 많은 공간을 차지하는 LONG이나 DOUBLE등의 값들에 적용된다면, 비율면에서 더욱 적은 디스크 접근 비용을 보일 것이다. 또한, 이 기법은 MAX 와 MIN 에 대해 각각의 선계산을 두지 않고, 단일 선계산 데이타를 사용하여 모두 처리할 수 있다는 장점을 가지고 있다.

본 연구에서는 디스크 접근 횟수만을 비용으로 사용하는 단순한 비용모델을 사용하여 성능 평가를 수행했다. 제안된 기법의 정확한 성능 평가를 위해, 실제 데이타베이스 시스템에 본 논문에서 제안한 기법을 구현하여 여러 데이타 집합에 대한 성능 분석을 진행중이다. 그리고, 적용 시스템의 환경에 따라 선계산 자료의 관리 비용이 중요시 될 수 있으므로, 관리 비용을 줄이기 위한 연구도 필요하다.

참 고 문 헌

[1] E.F. Codd, S. B. Codd and C. T. Salley, "Beyond decision support," Computerworld, 27(30), July 1993.

[2] A. Gupta, V. Harinarayan, and D. Quass. "Aggregate-Query Processing in Data Warehousing Environments". In Proceedings of the 21th VLDB, 1995

[3] P. O'Neil and D.Quass, "Improved Query Performance with Variant Indexes." In Proceeding of the ACM SIGMOD International Conference on Management of Data, 1997

[4] Yannis Kotidis and Nick Roussopoulos, "An Alternative Storage Organization for ROLAP Aggregate Views Based on Cubetrees," SIGMOD Conference, 1998

[5] J. Gray, A. Bosworth, A. Layman, and Pirahesh, "Data cube: A relational aggregation operator generalizing group-by, cross-tabs and sub-totals," In Proceedings of VLDB, pp 358-369, 1995.

[6] V. Harinarayan, A. Rajaraman, and J.D. Ullman, "Implementing Data Cubes Efficiently.," ACM SIGMOD 1996, pp 205-216, 1996

[7] Surajit Chaudhuri and Umeshwar Dayal, "An Overview of Data Warehousing and OLAP technology," ACM SIGMOD Record, 26(1):65-74, 1997

[8] Ching-Tien Ho, Rakesh Agrawal, Nimrod Megiddo and Ramakrishnan Srikant, "Range queries in OLAP data cubes," In Proc. Of the ACM SIGMOD Conference on Management of Data, 1997

[9] Dong Wook Kim, Myoung Ho Kim, Yoon Joon Lee, "An Efficient Processing of Range Min/Max Queries over Data Cube," Information Science 112, 1998, pp.223-237.

[10] Ching-Tien Ho, Jehoshua Bruck, Rakesh Agrawal, "Partial-Sum Queries in OLAP Data Cubes Using Covering Codes," In PODS, 1997

[11] Chee-Yong Chan, Yannis E. Ioannidis, "Bitmap Index Design and Evaluation," ACM SIGMOD 1998, pp 355-366

양 우 석

1993년 한국과학기술원 전산학과 학사. 1995년 한국과학기술원 전산학과 석사. 1995년 ~ 현재 한국과학기술원 전산학과 박사과정 재학중. 관심분야는 OLAP, 멀티미디어 데이타베이스, 능동 데이타베이스, 웹드 와이드웹

김 명 호

1982년 서울대학교 컴퓨터 공학과 학사. 1984년 서울대학교 컴퓨터 공학과 석사. 1989년 MICHIGAN 주립대 전산학과 박사. 1989년 MICHIGAN 주립대 연구원. 1989년 ~ 1993년 한국과학기술원 조교수. 1993년 ~ 1999년 한국과학기술원 부교수. 1999년 ~ 현재 한국과학기술원 교수. 1993년 개방형 컴퓨터 통신 연구회(OSIA) 분산트랜잭션처리 분과위(TG-TP)의장. 1993년 ~ 1994년 한국통신기술협회(TTA) 분산 트랜잭션처리 실무위원회 의장. 관심분야는 분산데이타베이스, 분산트랜잭션, 멀티미디어 데이타베이스