저장뷰를 통한 빙산 질의 처리
(Iceberg Query Processing by Materialized View)

홍 석 진 † 이 석 호 ††
(Seokjin Hong) (Sukho Lee)

요 약 빙산 질의란 대용량의 데이타에 대해 집단 함수를 수행하여 특정 임계값 이상인 데이타를 결과로 반환하는 연산을 의미한다. 빙산 질의는 도메인의 크기가 대단히 큰 다차원, 대용량의 데이타에 대해 적용되므로 집단 함수의 수행을 위한 카운터를 전부 메모리에 적재할 수 없는 상황이 발생한다. 이 논문에서는 빙산 질의에 대한 저장뷰를 통해 효율적으로 빙산 질의를 수행하는 방법을 제시하였다. 빙산 질의의 임계값이 저장뷰 내에 포함되는 경우에는 즉각적으로 결과를 돌려줄 수 있으며, 그렇지 않은 경우에도 표본 추출 대신 저장뷰를 사용함으로써 빙산 질의 중간 단계의 후보 수를 크게 감소시키고, 질의 수행 시간 또한 단축시킬 수 있다. 또한 순위 빙산 질의를 수행하는 방법을 제시하여 사용자로 하여금 보다 직관적인 질의를 작성할 수 있도록 하였다.

Abstract Iceberg query means such operation that computes aggregate functions over large data set and returns the results satisfying the specified threshold. Because iceberg queries are mainly applied to very large data set, the situation may occur that all counters for computing iceberg queries cannot be located in main memory. This paper proposes a method which computes iceberg queries efficiently by materialized views storing the results of aggregate queries above the specified threshold. If the threshold of an iceberg query is above that of the materialized view, the result can be returned immediately, and if not, no sampling process is needed by using the materialized view. So the number of candidates and running time can be reduced. It also proposes a technique for computing ranking iceberg queries, which helps users ask more intuitive queries.

1. 서 론

다차원 관계 데이타베이스에서의 group-by를 통한 집단 함수 연산은 OLAP 환경에서 중요하게 사용되는 연산 중의 하나이다. 하지만, 많은 경우에 있어서 사용자는 전체 질의 결과보다는 특정 임계값을 넘어가는 결과에 대해 주로 관심을 갖게 된다. 이처럼 대용량의 데이타들에 대해 group-by를 통한 집단 함수를 수행한 뒤 임계값 이상인 데이타들을 결과로 반환하는 연산을 빙산 질의(Iceberg Query)라 하며 이는 [1]에서 처음으로 제기되었다. 이러한 빙산 질의는 다차원, 대용량의 데이타를 다루는 데이타웨어하우스[2][3]와 데이타마이닝[4] 등의 분야에서 사용된다.

빙산 질의에 대한 기본적인 SQL문의 형태는 그림 1과 같다.

여기서 테이블 R의 애트리뷰트 dim_1, dim_2, ..., dim_k는 group-by를 위한 차원 애트리뷰트라 하고, 차원 애트리뷰트의 실제 각 레코드 값들을 차원 벡터라 한다. 그리고 나머지 애트리뷰트 $value_1$, $value_2$, ... 는 실제 차원 벡터에 대한 값을 명시하는 값 애트리뷰트가 된다. 이러한 R과 임계값 T가 주어졌을 때 빙산 질의는 차원 벡터 각각의 값들에 대한 카운트나 합 또는 평균을 계산한 후 임계값 이상인 것들만 반환하게 된다.

빙산 질의를 처리하기 위한 가장 기본적인 방법은 각 차원 벡터마다 하나의 카운터를 메모리에 생성한 후, 데이타베이스를 스캔하면서 집단 함수를 수행하는 것이다. 그러나 차원이 커지고, 도메인의 크기 또한 대단히 커지게 되는 데이타웨어하우스 환경에서는 이러한 카운터를 메모리에 전부 유지할 수 없는 상황이 발생한다. [1]에

· 본 연구는 BK21의 지원을 받았음.
† 학생회원 : 서울대학교 전기·컴퓨터공학부
jinny@db.snu.ac.kr
†† 종신회원 : 서울대학교 컴퓨터공학부 교수
shlee@comp.snu.ac.kr
논문접수 : 2000년 4월 14일
심사완료 : 2000년 9월 20일

```
SELECT dim1, dim2, ..., dimk, count(value1, value2, ....)
FROM R
GROUP BY dim1, dim2, ..., dimk
HAVING count(value1, value2, ...) >= T
```

그림 1 기본적인 빙산 질의의 형태

서는 이를 표본 추출과 해싱의 방법을 통해 해결하고 있다.

현재까지의 방법은, 많은 수행시간을 필요로 하는 질의를 대용량의 데이타 전체에 대해 임계값이 변함에 따라 매번 수행하여야 하며, 중간 단계에 생성되는 후보 수가 커지는 문제점이 있다. 또한, 빙산 질의의 경우 사용자가 적절한 임계값을 예측하여 질의를 하여야 하는데, 실제 사용자는 기본 데이타의 분포를 정확히 알 수 없기 때문에, 임계값을 예측하여 질의를 만들기가 어렵다.

이 논문에서는 집단 함수의 결과가 특정 임계값 이상인 데이타로 구성된 저장뷰[5]를 통해 임의의 임계값으로 구성된 빙산 질의를 효율적으로 수행하는 방법을 제안하였다. 빙산 질의의 수행 시 표본 추출 대신 저장뷰에 있는 데이타를 사용함으로써, 표본 추출 시간을 절약하고, 빙산 질의 중간 단계의 후보 개수를 크게 줄여, 기존의 방법에 비해 빠른 질의 수행이 가능하다.

그리고 순위를 통해 임계값을 지정하는 순위 빙산 질의를 제안하여, 사용자로 하여금 전체 데이타에 대한 정보 없이도 직관적인 질의를 작성할 수 있도록 하였다. 또한, 이러한 순위 빙산 질의를 빙산 질의로 변경하여 수행하는 방법을 제시하였다.

본 논문의 구성은 다음과 같다. 먼저, 2장에서는 [1]에서 제시한 빙산 질의를 수행하는 기법에 대해 살펴본다. 3장에서는 빙산 질의를 위한 저장뷰를 제안하고, 저장뷰를 통한 빙산 질의 기법에 대해 설명한다. 또한 순위 빙산 질의와 이의 처리를 위한 빙산 질의 히스토그램을 제안하며, 이를 통한 순위 빙산 질의의 수행 기법을 기술한다. 그리고 저장뷰의 갱신에 대해 살펴본다. 4장에서는 저장뷰를 통한 빙산 질의의 처리와, 기존 빙산 질의 처리를 실험을 통해 비교하고, 저장뷰를 통한 후보 개수 감소와, 질의 수행 시간 감소 효과를 확인한다. 마지막으로, 5장에서 결론과 추후 연구 과제에 대해 기술한다.

2. 관련 연구

대용량의 데이타들에 대해 group-by를 통한 집단 함수를 수행한 뒤 임계값 이상인 데이타들을 결과로 반환하는 빙산 질의는 [1]에서 처음으로 제기되었다. [1]에서 제시한 빙산 질의 처리 알고리즘의 기본 아이디어는, 카운터의 개수가 매우 많아져서 메모리 내에 카운터를 전부 적재할 수 없는 경우에 해싱을 통해 카운터를 공유하겠다는 것이다. 해싱을 통한 카운터 공유는 그림 2에서처럼 두 가지의 문제점을 수반한다. 첫째, 임계값을 넘지 않는 여러 개의 데이타가 하나의 카운터 버킷을 공유함으로써, 버킷 내의 데이타들이 모두 후보로 선정되는 경우이다. 둘째, 한 카운터 버킷 내에 임계값을 넘는 데이타와 임계값을 넘지 않는 데이타가 함께 들어가 있음으로 인해, 이 버킷내의 모든 데이타가 후보로 선정되는 경우로, 두 경우 모두 후보 개수를 증가시키는 문제점을 갖고 있다. [1]에서는 이러한 문제를 다중 해시 함수와 표본 추출을 통해 해결하고 있다.

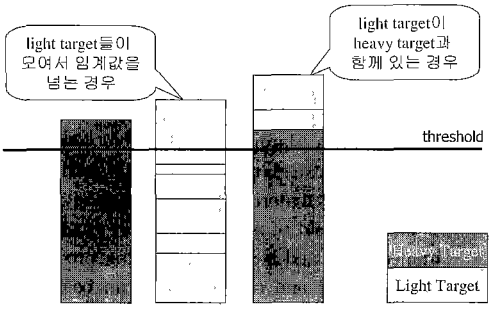

그림 2 해싱을 통한 카운터 공유의 문제점

첫 번째 문제는 여러 개의 해시 함수를 사용하여 해결한다. 각 데이타들을 여러 해시 함수를 거치도록 하여 모든 해시 함수에 대해 임계값을 넘는 데이타만 후보로 선정한다.

두 번째 문제를 해결하기 위하여 [1]에서는 표본 추출[6]과 해싱[7]을 함께 사용한다. 먼저 기본 데이타베이스에서 표본을 추출하여 임계값이 넘어가는 후보들을 선정하고 그 후보들에 대해서는 별도의 카운터를 통해 개수를 세어, 후보의 개수를 줄인다. 최종적으로 데이타베이스를 스캔하면서 그 후보들에 대한 카운팅을 하여 결과를 얻는다.

표본 추출을 통해 얻어진 후보는 최종 후보의 개수를 줄이는 역할을 한다. 표본 추출을 통해 얻어진 후보는 그 자체로 임계값을 넘을 가능성이 큰 데이타이다. 따라서, 해시를 통해 카운터를 공유한다면, 그들과 같은 해시 버킷을 사용하는 다른 데이타들은, 자신이 임계값을

넘는지 여부와 관계없이 후보로 선정될 가능성이 크다. 따라서 표본 추출을 통해 얻어진 후보를 별도로 관리함으로써 두 번째 문제를 어느 정도 해결할 수 있다.
그러나 이 경우, 표본 추출을 통해 얻어진 후보의 최종 결과 포함 여부는 결정된 것이 아니다. 따라서 이들 후보는, 최종 데이타베이스 스캔 시에 함께 카운팅하여야 하며, 이로 인해 최종 후보의 개수가 커지는 문제점이 있다.

3. 빙산 질의 저장뷰를 통한 빙산 질의 알고리즘의 개선

3.1 빙산 질의 저장뷰

빙산 질의 저장뷰는 대용량 데이타에 대한 집단함수의 결과 중 특정 임계값을 넘는 데이타에 대한 값들을 명시적으로 유지하는 저장뷰이다.
저장뷰를 생성하기 위해서는 차원 애트리뷰트들의 집합과 임계값이 인자로 필요하다. 빙산 질의 저장뷰를 생성하기 위한 SQL문의 구조는 그림 3과 같다.

```
CREATE VIEW 뷰 이름 AS
SELECT dim1, dim2, ..., dimk, count(rest)
FROM 테이블 이름
GROUP BY dim1, dim2, ..., dimk
HAVING count(rest) >= 임계값
```

그림 3 빙산 질의 저장뷰를 생성하기 위한 SQL문

그림 4는 빙산 질의 저장뷰의 한 예로서, 전체 집단함수 결과 중 임계값 60을 넘는 결과들로 구성되어 있다.
많은 경우 이러한 빙산 질의 저장뷰는 빙산 큐브(Iceberg Cube)[8]를 사용하여 구성한다. 빙산 큐브란 특정 임계값을 넘는 데이타들로 구성한 데이타 큐브로 빙산 질의 수행과, 데이타마이닝에서의 다차원 연관 규칙 탐사, 그리고 데이타 큐브 구성 등에 사용된다. 대용량 다차원 데이타의 경우, 모든 데이타들에 대해 데이타 큐브를 구성할 경우, 그 크기가 너무 커질 수 있기 때문에[1], 빙산 큐브를 통해 특정 임계값을 넘는 결과만을 저장한다면 저장 공간을 크게 절약할 수 있다.
빙산 큐브를 통해 빙산 질의 저장뷰를 구성한다면, 가능한 모든 차원 애트리뷰트의 그룹에 대한 집단 함수 결과를 저장할 수 있기 때문에 다양한 빙산 질의를 처리할 수 있으며, 빙산 질의 큐브의 상향 연산(bottom-up computation) 기법을 통해 효율적으로 빙산 질의 저장뷰를 구성할 수 있다.

3.2 빙산 질의 저장뷰를 이용한 빙산 질의의 처리

빙산 질의 저장뷰을 이용한 빙산 질의 처리는 두 가지 형태로 생각해 볼 수 있다. 우선, T를 빙산 질의의 임계값이라 하고, count(rest)를 저장뷰에 저장된 차원 벡터의 카운트 값이라고 하자. 첫째, 'T ≥ min(count(rest))' 의 경우는 저장뷰 내에서 빙산 질의 결과를 바로 얻어 낼 수 있다. 예를 들어 그림 4와 같은 저장뷰가 있을 때 임계값이 90인 질의가 들어온다면, 결과 (a, b, 100), (a, c, 98), (b, b, 97), (a, a, 94), (a, f, 93), (b, a, 93), (c, b, 92), (b, d, 90) 을 저장뷰를 통해 바로 얻을 수 있다.

Rank	dim_1	dim_2	count(rest)
1	a	b	100
2	a	c	98
3	b	b	97
4	a	a	94
5	a	f	93
6	b	a	93
7	c	b	92
8	b	d	90
9	b	c	89
...	...	...	...
...	...	...	....
100	e	f	60

그림 4 빙산 질의 저장뷰

많은 경우에 있어서 사용자는 질의 결과 중 상위 일부분만을 원하기 때문에, 적절한 임계값 이상의 데이타를 빙산 질의 저장뷰를 통해 미리 갖고 있다면, 같은 데이타에 대한 반복적인 디스크 스캔의 회수를 효과적으로 줄일 수 있을 것이다.
둘째, 'T < min(count(rest))'의 경우는 두 단계로 질의가 수행된다. 우선, 빙산 질의 저장뷰에서 min(count(rest))까지의 결과를 얻어 사용자에게 바로 돌려준다. 그리고, 나머지 min(count(rest))부터 T사이의 결과를 기존 빙산 질의 알고리즘을 사용하여 계산한다. 이 때, 기존의 방법과는 달리, 저장뷰에 포함되어 있는 차원 벡터를 제외한 나머지 차원 벡터들에 대해서만 카운팅을 한다. 이로 인해, 한 카운터 버킷 내에 임계값을 넘는 데이타와 임계값을 넘지 않는 데이타가 함께 들어가서 후보의 개수가 커지는 해싱의 문제점을 많이 감소

1) 39Mbyte 크기를 갖는 9차원 데이타에 대해 9Gbyte가 넘는 데이타 큐브가 생성된다. [8]

```
Algorithm Iceberg_with_MaterializedView(R, dim1, dim2, ..., dimk, T, V)
Input:
    R: 빙산 질의가 수행되는 릴레이션
    dim1, dim2, ..., dimk : 빙산 질의의 차원 에트리뷰트
    T: 빙산 질의의 임계값
    V: 저장뷰
    V.T: 저장뷰의 임계값, V.T = min{t | t=r.count(rest), r ∈ V}
Output:
    result: count가 임계값을 넘는 차원 벡터와 해당 count 값으로 구성되는 레코드의 집합
Begin
    If T ≤ V.T Then
        result = result ∪ {r | r.count(rest) > T, r ∈ V};
    Else
        result = result ∪ {r | r ∈ V};
        sample = V;
        hash_counter = COARSE-COUNT[1](R, dim1, ..., dimk ,sample);
        temp = CANDIDATE-SELECTION[1](hash_counter, T);
        result = result ∪ temp;
    End If
    return result;
End
```

그림 5 저장뷰를 통한 빙산 질의 처리 알고리즘

시킬 수 있다. 따라서 이 경우 별도의 표본 추출과정을 생략할 수 있으며, 결과로 나온 후보의 수도 줄일 수 있다. 저장뷰를 이용한 빙산 질의 알고리즘은 그림 5와 같다.

저장뷰에 포함된 데이타는 이미 임계값을 넘는 데이타들이며, 이들과 해시 버킷을 공유하는 데이타들은 자신이 임계값을 넘는지 여부와는 관계없이 해시 카운팅 과정에서 무조건 후보로 선택된다. 따라서 이러한 저장뷰에 포함된 데이타를 해시 카운팅에 포함시키지 않음으로써, 최종 후보의 수를 상당 부분 감소시킬 수 있는 것이다.

[1]에서는 임계값을 넘을 가능성이 있는 후보들을 미리 표본 추출 기법을 통해 선정하고 해시 카운팅 시 이 후보들을 따로 관리함으로써, 임계값을 넘지 않는 데이타가 이 후보들과 함께 카운팅되어 후보의 개수가 커지는 문제를 해결하였지만, 빙산 처리 저장뷰가 있는 상황에서는 빙산 처리 저장뷰가 표본 추출을 통한 후보의 역할을 수행하여 표본 추출 과정을 생략할 수 있다. 기존 방법의 경우, 표본 추출을 통한 후보는 결과로 선정될 지의 여부가 아직 결정이 안된 상태이므로 최종 후보에 포함시켜 결과 선정 여부를 결정하여야 하지만, 빙산 처리 저장뷰의 데이타들은 이미 결과로 선정된 데이타들이므로 최종 후보에 포함시킬 필요가 없으며, 부가적인 처리가 불필요하다는 장점이 있다.

나머지 T부터 min(count(rest))사이의 결과를 위한 빙산 질의 알고리즘은 [1]에서 제시한 해싱 기법과 [9]에서 제시한 동적 분할 기법을 모두 이용할 수 있다.

3.3 빙산 질의 저장뷰를 이용한 순위 빙산 질의의 처리

기존의 빙산 질의의 경우, 상위 일부의 결과만을 돌려받기 위해, 사용자는 특정 임계값을 명시하여야 한다. 예를 들어, 제품별 총 매출액이 상위권에 드는 제품들을 조사하고 싶은 경우, 사용자는 임계값으로 특정 매출액을 지정해야 한다. 하지만, 질의 작성시점에는 매출액이 어느 분포를 갖는지를 파악하지 못하기 때문에, 원하는 결과를 돌려주는 질의를 쉽게 작성하기 어렵다. 따라서, 임계값으로 사용될 매출액을 구하기 위해, 전체 매출액의 분포를 구하는 질의를 미리 해 보거나, 아니면 임의의 적당한 값을 선택할 수밖에 없다. 이 경우, 만일 너무 높은 임계값을 선택한다면, 원하는 것보다 작은 개수의 결과가 반환되거나, 아니면 아예 아무런 결과가 나오지 않을 수도 있다. 또한, 너무 낮은 임계값을 사용하면, 너무 많은 결과가 반환될 뿐 아니라, 후보의 개수가 크게 증가하여, 해시 카운터를 사용하는 빙산 질의 처리 알고리즘에 상당히 좋지 않은 영향을 줄 수 있다.

따라서, 데이타의 분포에 영향을 받는 임계값에 의한 질의 보다, '상위 100개'와 같이 순위를 통한 질의가 가능하다면, 사용자는 보다 직관적인 질의를 작성할 수 있으며, 질의 처리도 보다 효율적으로 수행될 수 있을 것이다.

이와 같이, 대용량의 데이타에 대해 집단 함수를 수행한 결과에서, 특정 순위 이상의 결과만을 반환하는 질의를 순위 빙산 질의(Ranking Iceberg Query)라고 한다. 하지만 순위 빙산 질의 처리는 단순한 문제가 아니다. 빙산 질의의 특성상, 질의 수행시 전체 데이타에 대한 카운팅을 한 후에 그 중 임계값을 넘는 대상을 고를 수 있는 것이 아니라, 알고리즘 수행 중에 임계값을 넘을 가능성이 있는 후보들을 선정한 후 후보들을 다시 카운팅 해야 하기 때문이다. 이 때 임계값을 기준으로 후보를 선정하는 것은 수행 중에 가능하지만, 순위를 기준으로 후보를 선정하려면 전체 데이타의 분포를 알아야 하며, 이를 통해 순위에 해당하는 임계값을 알 수 있어야 한다. 하지만, 이 경우 순위 빙산 질의의 특성상 반드시 정확한 임계값을 구해야 할 필요는 없다.[10] 정확한 임계값보다 작은 임계값을 사용한다면 원하는 결과보다

많은 개수의 결과를 얻게 되며, 그 중 원하는 결과만을 사용하면 된다. 만일 정확한 임계값보다 큰 임계값을 사용한다면 원하는 결과보다 적은 개수의 결과를 얻게된다. 이 경우 얻어진 결과를 우선 사용자에게 돌려주고, 얻어진 결과와 저장뷰에 있는 결과를 제외한 나머지 데이타들에 대해 질의를 재수행하면 된다. 이처럼 순위로부터 반드시 정확한 임계값을 구해야 할 필요는 없지만, 가능하면 정확한 임계값보다 작고, 근접한 임계값을 추정하는 것이 질의 수행에 더욱 효과적이다.

순위로부터 임계값을 추정하는 첫 번째 방법으로, 아래의 식을 사용한다.

$$T = \alpha\left((R-1)\frac{\min(count(rest)) - \max(count(rest))}{\max(rank) - 1} + \max(count(rest))\right)$$

위의 식에서 T는 추정할 임계값, R은 순위 빙산 질의의 순위이다. $count(rest)$는 저장뷰 내의 각 차원 벡터의 카운트 값이며, $rank$는 각 차원 벡터의 순위이다. α는 $0<\alpha<1$ 의 수로, 제어 변수의 역할을 한다. α가 작을수록 임계값이 작아져서 질의를 재수행할 확률이 낮아지며, α가 크면 임계값이 커져서 적은 개수의 후보를 내는 효율적인 질의 수행을 하지만, 질의를 재수행해야 할 확률이 커질 수 있다.

순위로부터 임계값을 추정하는 두 번째 방법은 전체 데이타의 분포 정보를 나타내는 빙산 질의 히스토그램을 사용하는 방법이다. 이 방법은 앞의 방법과는 달리, 질의 재수행이 절대 일어나지 않음을 보장한다.

Rank	dim_1	dim_2	count(rest
10	a	b	92
20	c	f	83
30	h	i	73
50	k	m	65
100	s	o	50
200	t	l	27
500	v	x	14
1000	z	w	9
...	...	...	...

그림 6 빙산 질의 히스토그램

빙산 질의 히스토그램은 그림 6과 같이 구성한다. 전체 데이타 항목 중 일부에 대해 순위와 임계값을 미리 계산하여, 전체 데이타의 분포를 저장한다. 이러한 빙산 질의 히스토그램을 통해, 순위로 임계값이 주어지는 순위 빙산 질의를 임계값이 카운트 값인 일반 빙산 질의로 바꾸어 계산할 수 있다. 그림 6에서 순위 45까지의 순위 빙산 질의는 순위 50의 카운트 값에 해당하는 임계값 65인 빙산 질의로 변경하여 계산한다.

R을 빙산 질의의 순위라 하고, Rank를 저장뷰에 저장된 순위 값이라고 하자. 순위 빙산 질의에 있어서도, 'R ≤ max(Rank)'의 경우는, 빙산 질의 저장뷰에서 순위 빙산 질의 결과를 바로 얻어 낼 수 있으며, 'T > max(Rank)'의 경우는 저장뷰로부터 max(Rank)까지의 질의 결과를 얻고, max(Rank)부터 R까지의 결과는 위의 방법을 통해 순위로부터 임계값을 추정하여, 일반 빙산 질의로 변경한 후 처리할 수 있다.

3.4 빙산 질의 저장뷰의 갱신

이러한 빙산 질의 저장뷰는 데이타의 변화에 따라 갱신되어야 한다. 빙산 질의의 특성상 대용량 데이타웨어하우스 환경을 가정할 때, 데이타베이스의 갱신은 점진적인 데이타의 적재를 통해 이루어진다[2][11][12]. 또한 이러한 점진적인 적재는 주기적으로 일어나므로, 데이타웨어하우스 갱신 주기마다 저장뷰를 갱신하면 된다. 저장뷰의 갱신은 다음과 같은 순서로 이루어진다. 우선 새로 추가되는 데이타에 대한 카운터를 메모리에 만들어 새로 추가된 데이타들에 대한 카운팅을 한 후, 전체 데이타베이스를 한 번 스캔하면서, 기존 데이타에 대한 카운팅을 한다. 새로 추가되는 데이타의 종류는 그렇게 많지 않기 때문에 카운터를 메모리에 전부 적재할 수 있으나, 만일 그렇지 않을 경우는 새로 추가된 데이타를 몇 개로 나누어 각각 카운팅 하게된다. 그런 후, 새로 추가된 데이타 중 카운트 값이 가장 큰 데이타부터 기존 저장뷰에 합병시킨다. 새로 추가된 데이타의 카운트 값을 보고 알맞은 위치에 삽입시키는 방법으로 합병이 진행된다. 이 경우, 합병시키는 데이타의 차원 벡터가 이미 저장뷰에 존재한다면 기존 항목을 삭제하여야 한다.

예를 들어, 그림 8과 같은 저장뷰가 있는 상황을 생각해 보자. 데이타웨어하우스의 갱신 주기 동안 추가된 새로운 데이타 아이템들에 대해, 추가된 값에 대한 카운팅을 한 후, 데이타베이스를 한 번 스캔하며 기존 값에 대한 카운팅을 한다. 그 결과로 그림 9와 같이 새로 추가된 데이타에 대한 카운트를 얻는다. 이 결과를 기존의 저장뷰와 합병함으로써 그림 10과 같은 갱신된 저장뷰를 얻게 된다. 예를 들어 (a, c)의 경우 새로 카운팅한 결과 113이란 값을 얻었으며, 기존의 저장뷰에 이미 있던 항목이기 때문에, 기존의 항목을 삭제하고, 새로운 순위인 1위로 삽입된다. (b, c)의 경우 새로 카운팅한 결과 95란 값을 얻었으며, 새로운 항목이기 때문에 저장뷰 내의 적당한 위치를 찾아 4위로 삽입된다. 이 때, 삽입 위치의 아래에 있는 항목은 순위가 하나씩 밀리게 된다. 저장뷰의 갱신 알고리즘은 그림 7과 같다.

```
1  Algorithm Update_MaterializedView (R, V, N)
2  Input:
3      R: 빙산 질의가 수행되는 릴레이션
4      V: 저장뷰
5      N: 새로 추가된 데이타
6  Output:
7      V: 갱신된 저장뷰
8  Begin
9      For All tuple t ∈ N Do
10         If t.dim d ∈ counter Then
11             counter[d]++;
12         Else
13             add new counter[d] in counter;
14             counter[d]++;
15         End If
16     End For
17     For All tuple t ∈ R Do
18         If t.dim d ∈ counter Then
19             counter[d]++;
20     End For
21     merge V and counter;
22 End
```

그림 7 저장뷰의 갱신 알고리즘

Rank	dim_1	dim_2	count(rest)
1	a	b	100
2	a	c	98
3	b	b	97
4	c	b	91
5	d	b	90

그림 8 기존의 저장뷰

dim_1	dim_2	count(rest)
a	c	113
a	a	10
b	c	95
c	d	5

그림 9 새로 추가된 데이타

Rank	dim_1	dim_2	count(rest)
1	a	c	113
2	a	b	100
3	b	b	97
4	b	c	95
5	c	b	91

그림 10 새로 갱신된 저장뷰

주기적인 저장뷰의 갱신시, 새로 추가되는 데이타를 메모리 내에서 카운트 할 수 있는 일반적인 경우에는 데이타베이스를 한 번 스캔하는 만큼의 비용이 소요되며, 이는 빙산 질의를 수행하는 비용보다 일반적으로 작다.

3.5 SUM 연산

지금까지 저장뷰를 통한 빙산 질의 수행시, 집단 함수 중 count에 대한 경우만 살펴보았으나, 비슷한 방법으로 sum에 대한 처리도 가능하다. count의 경우, 주어진 데이타 아이템에 해시 함수를 적용한 해당 카운터를 하나 증가시키는 식으로 질의 수행을 하였지만, sum의 경우는 카운터에 실제 값을 더하는 방식을 취하면 된다. 해당 임계값 또한 실제 값에 대한 합계로 주어지며, 역시 주어진 임계값 이상의 결과만 고려하면 된다.

4. 실험 및 분석

3차원 데이타 1,000,000개에 대해서 저장뷰가 없는 상황과 저장뷰가 있을 경우에 메모리 크기와 임계값을 변화시켜 가면서 최종 후보의 개수를 비교하였다. 저장뷰를 사용하는 경우 표본 추출 과정 없이 실행시켰다. 실험에 사용된 기계는 Ultra Sparc II 333MHz이며, OS는 Solaris 7이다.

4.1 후보 개수의 비교

먼저 'T ≥ min(count(rest))' 경우는 저장뷰를 통해 바로 결과를 얻을 수 있으므로, 항상 빙산 질의를 수행해야 하는 기존 방법에 비해 절대적으로 좋은 성능을 보인다. 따라서 'T < min(count(rest))'인 경우에 대해서만 메모리 크기와 임계값을 변화시켜 가면서 기존의 방법과의 비교를 하였다.

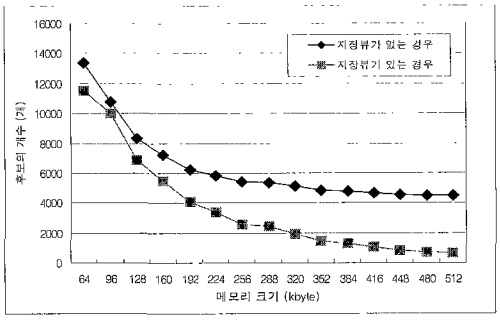

그림 11 메모리 크기에 따른 후보 개수의 변화

그림 11은 메모리 크기에 따른 후보 개수의 변화를 나타내고 있다. 실험에 사용된 임계값은 56이며, 저장뷰는 임계값 68 이상인 결과를 저장하고 있다. 실험 결과를 통해 알 수 있듯이, 메모리 크기가 커질수록 한 버킷을 공유하여 카운팅 되는 차원벡터의 수가 줄어들기 때문에 후보의 개수는 감소하게 된다. 또한, 전반적으로 저장뷰가 있을 경우, 저장뷰가 없는 경우 보다 후보의

개수가 적게 나타나는 것을 알 수 있다. 이는, 기존 방법의 경우 표본 추출을 통해 임계값을 넘을 가능성이 큰 후보를 미리 선택하지만, 이 후보들은 임계값을 넘는다는 확실한 보장이 없으므로 최종 후보에 함께 포함시켜야 하는 반면, 저장뷰를 표본 추출 대신 사용할 경우, 저장뷰에 있는 결과들은 이미 임계값을 넘는 결과들로, 최종 후보에 포함시킬 필요가 없기 때문이다.

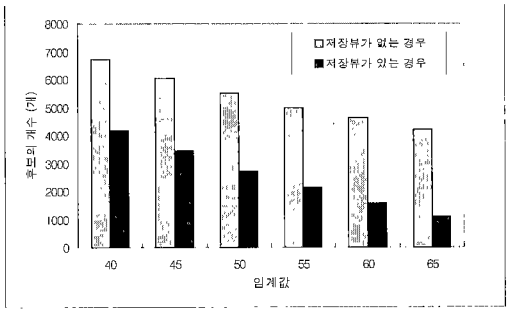

그림 12 임계값에 따른 후보 개수의 변화

그림 12는 임계값에 따른 후보 개수의 변화를 나타내고 있다. 임계값을 40에서 65까지 변화시키며 저장뷰가 없는 경우와, 저장뷰를 사용한 경우에 대해 빙산 질의를 수행하였다. 빙산 질의 수행에 사용된 메모리 크기는 274.62 kbyte이며, 저장뷰에서는 임계값 70 이상인 결과를 저장하고 있다. 저장뷰에 임계값이 70 이상인 결과가 저장되어 있으므로, 임계값 70이상인 빙산 질의는 수행하지 않았다. 역시 실험 결과를 통해 알 수 있듯이, 전반적으로 저장뷰를 통해 후보의 개수가 감소된 것을 알 수 있으며, 임계값이 커질수록 후보의 개수가 줄어들고 있음을 알 수 있다.

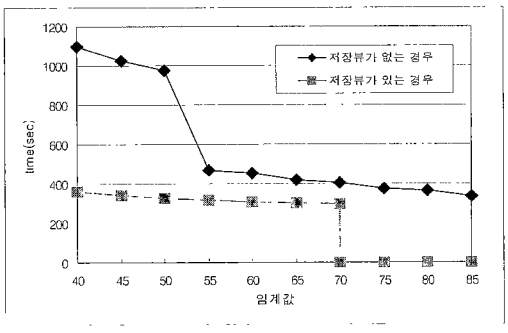

그림 13 임계값에 따른 수행 시간의 변화

4.2 수행 시간의 비교

그림 13은 임계값에 따른 수행 시간의 변화를 보여주고 있다. 그림에서 알 수 있듯이 임계값이 커질수록 수행 시간이 작아지며, 임계값 70이상의 결과는 저장뷰를 통해 바로 돌려주기 때문에 수행시간이 0에 가까워지고, 임계값 70이하에서도 저장뷰가 없는 상황보다는 빠른 수행시간을 보여주고 있다. 저장뷰가 없는 경우, 임계값 55와 50 사이에서 급격한 변화를 보이는데, 이는 후보의 개수가 너무 많아져서 최종 카운트시 후보들을 메모리에 적재할 수 없는 상황이 발생하기 때문이다. 이 경우 후보를 두 그룹으로 나누어 카운트해야 하기 때문에, 한 번의 전체 데이타베이스 스캔이 더 필요하다.

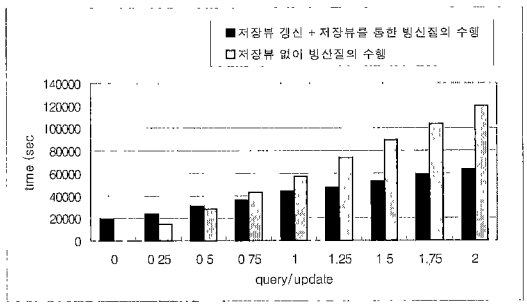

그림 14 저장뷰의 갱신과 빙산질의의 수행 비율에 따른 수행시간

4.3 갱신 비용

그림 14는 저장뷰의 갱신에 대한 실험 결과이다. 저장뷰의 갱신과 빙산 질의의 비율을 변경시켜 가면서 전체 소요시간을 조사하였다. 저장뷰가 있는 경우의 전체 소요시간은 저장뷰의 갱신 시간과 저장뷰를 통한 빙산질의 수행 시간의 합이며, 저장뷰가 없는 경우는 빙산질의를 수행하는 시간이 된다. 빙산 질의의 수행에 비해 저장뷰의 갱신이 많은 경우에는 저장뷰가 없는 상황이 시간이 적게 걸리며, 갱신에 비해 빙산질의의 수행이 많아질수록 저장뷰가 있는 상황의 수행시간이 상대적으로 작아진다. 매 갱신 주기마다 빙산질의의 수행이 한번씩 일어나는 경우에도 저장뷰가 있는 상황의 수행시간이 상대적으로 작은 이유는, 임계값이 저장뷰에 포함되는 질의의 경우 저장뷰를 통해서 데이타베이스 스캔 없이 결과를 바로 돌려줄 수 있기 때문이다.

또한, 데이타 웨어하우스의 특성상 이러한 갱신 작업은 주기적으로 일어나게 되며, 이러한 갱신 작업은 사용자의 질의에 영향을 미치지 않으므로[2], 데이타 웨어하

우스 갱신 주기마다 저장뷰의 갱신을 수행하게 되면, 큰 부가 비용 없이 저장뷰를 유지할 수 있다.

5. 결 론

이 논문에서는 대용량 데이타에 대한 집단함수의 결과 중 특정 임계값을 넘는 데이타에 대한 값들을 빙산 질의 저장뷰로 유지하고, 이를 이용해 빙산 질의를 효율적으로 수행하는 방법을 제시하였다.

이 저장뷰는 빙산 질의 수행 시 후보의 개수를 효과적으로 줄일 수 있으며, 별도의 표본 추출 없이 빙산 질의를 수행할 수 있게 하여, 수행 시간을 단축시킬 수 있다. 그리고 많은 수행 시간을 필요로 하는 큰 임계값으로 구성된 질의에 대해서, 질의를 매번 실행하지 않고도 결과를 즉시 돌려줄 수 있으며, 그렇지 않은 경우에도 저장뷰를 통해 결과의 일부를 사용자에게 신속히 돌려줄 수 있기 때문에, 연산 수행에 많은 시간이 걸리는 데이타웨어하우스 환경에서 보다 효율적으로 사용될 수 있을 것이다. 또한 순위 빙산 질의를 빙산 질의로 변환시켜 수행하는 방법을 제시하여 사용자가 보다 직관적인 질의를 작성할 수 있도록 하였다.

추후 연구로는, 순위 빙산 질의 수행 시 빙산 질의 저장뷰에 있는 데이타를 통해 데이타 분포를 추정하여 순위로부터 좀 더 정확한 임계값을 알아내는 방법에 대한 연구가 필요하다. 그렇게 되면 별도의 빙산 질의 히스토그램 없이도 효과적으로 순위 빙산 질의를 수행할 수 있을 것이다.

참 고 문 헌

[1] M. Fang, N. Shivakumar, H. G. Molina, R. Motwani, J. D. Ullman, "Computing iceberg queries efficiently," VLDB 1998

[2] S. Chaudhuri and U. Dayal, "An overview of data warehousing and OLAP technology," ACM SIGMOD Record 26(1), 1997

[3] M.C. Wu, A.P. Buchmann, "Research Issues in Data Warehousing," BTW'97, Ulm, March, 1997

[4] A. Savasere, E. Omiecinski, and S. Navathe, "An efficient algorithm for mining association rules in large databases," VLDB, 1995

[5] N. Roussopoulos, "Materialized Views and Data Warehouses," SIGMOD Record, 27(1), p21-26, March 1998.

[6] F. Olken, "Random sampling from databases," PhD thesis, University of California, Berkeley, 1993

[7] J.S. Park, M.S. Chen, and P.S. Yu, "An effective hash based algorithm for mining association rules," In Proceedings of ACM SIGMOD Conference, 1995

[8] Kevin S. Beyer, Raghu Ramakrishnan, "Bottom-Up Computation of Sparse and Iceberg CUBEs," SIGMOD Conference 1999

[9] 배진욱, 이석호, "빙산 질의 처리를 위한 동적 분할", '99 봄 학술발표논문집(B), 제26권 1호, pp.164-166, 1999.4

[10] S. Chaudhuri, L. Gravano, "Evaluating Top-k Selection Queries," VLDB 1999

[11] D.Quass, J.Widom, "On-Line Warehouse View Maintenance," SIGMOD 1997

[12] W.H.Inmon, R.D.Hackathorn, "Using the Data Warehouse," John Wiley & Sons, 1994

홍 석 진
1998년 서울대학교 컴퓨터공학과 졸업(공학사). 2000년 서울대학교 대학원 컴퓨터공학과 졸업(공학석사). 2000년 ~ 현재 서울대학교 대학원 전기·컴퓨터공학부 박사과정. 관심분야는 데이타웨어하우스, 데이타마이닝 등.

이 석 호
정보과학회논문지 : 데이타베이스
제 27 권 제 2 호 참조

2) 일반적인 데이타웨어하우스 시스템에서는 사용자의 질의가 없는 밤 시간에 갱신 작업을 수행한다.[11]