동적 분할에 의한 평균 빙산 질의 처리

배진욱 °, 이석호
oblody@db.snu.ac.kr, shlee@comp.snu.ac.kr
서울대학교 컴퓨터공학과

Computing Average Iceberg Query by Dynamic Partition

Jinwook Bae °, Sukho Lee
Dept. of Computer Engineering, Seoul National University

요 약

평균 빙산 질의란 대용량의 데이타들에 대해 avg 집단 함수를 수행한 뒤 임계값 이상인 데이타들을 결과로 출력하는 연산을 의미한다. 이 때 데이타 도메인의 크기가 메모리에 생성할 수 있는 카운터의 수보다 크기 때문에 연산 처리가 어렵다. 지난 연구에서 빙산 질의에 대해 제안한 해시 카운터는 avg 연산의 경우 착오누락이 발생한다는 문제점이 존재한다. 그래서 이런 문제점들을 해결하며 효율적으로 연산을 수행하기 위해, 데이타베이스를 분할하며 카운터를 관리하는 '메모리 Full 분할', '후보 Full 분할'의 두 알고리즘을 제안한다. 실험결과 두 알고리즘은 메모리 크기, 데이타 분포, 데이타 순서에 영향을 받았는데, 데이타들이 정렬이 되어있거나 데이타분포가 정규분포를 이룰 때 우수한 성능을 보였다.

1. 서론

빙산 질의(Iceberg Query)란 도메인의 크기가 큰 대용량 데이타들에 대해 집단 함수를 수행한 뒤 임계값 이상인 데이타들을 결과로 반환하는 연산을 의미한다. 이때, 도메인의 크기가 메모리에 생성할 수 있는 카운터의 수보다 크기 때문에 연산 처리의 어려움이 생겨난다. 일반적인 집단 함수 연산이라면 각 데이타들에 대해 하나의 카운터를 생성한 뒤 데이타베이스를 스캔하며 집단 함수를 수행하면 되겠지만, 빙산 질의는 각 데이타들에 대해 하나의 카운터를 생성할 수가 없으므로 효과적인 카운터 처리가 필요하다. [1]에서는 해시에 의한 카운터를 두었으며, [2]에서는 분할을 통해 카운터를 관리하는 기법을 제시하였다. 이와 같은 빙산 질의는 주로 대용량의 데이타를 처리하는 데이타마이닝[3], 데이타웨어하우스[6] 분야에서 자주 수행되고 있다.

평균 빙산 질의는 빙산 질의 중에서 집단 함수가 avg일 경우를 의미한다. 평균은 count와 sum 모두 고려되어야 하므로, count나 sum만을 다루는 연산과 처리 방법이 달라진다.

평균 빙산 질의를 SQL문으로 표현하면, 테이블 R(target_1, target_2, ..., target_k, value)과 임계값 T가 주어졌을 때

```
select target_1, target_2, ..., target_k, avg(value)
from R
groupby target_1, target_2, ..., target_k
having avg(value) >= T
```

와 같다. 예를 들어, 그림 1의 R1 테이블에 대해 k=2, T=20인 질의를 수행하면, <a, e, 20>, <c, f, 25>가 결과로 얻어진다.

target_1	target_2	value
a	e	10
b	f	5
a	e	30
b	d	17
a	e	15
c	f	25

그림 1 R1 테이블

이 논문에서는 평균 빙산 질의를 효과적으로 수행하기 위한 '메모리 Full 분할'과 '후보 Full 분할'의 두 가지 알고리즘을 제안한다. 두 알고리즘 모두 분할을 통해 카운터를 관리하는데, 분할 시점에 따라 달라진다.

논문의 구성은 2절에서 관련연구를 살펴보고, 3절에서 동적 분할을 통한 두 가지 알고리즘을 소개한다. 4절에서 실험을 통해 두 가지 알고리즘을 비교하고, 5절에서 결론짓는다.

2. 관련 연구

[1]에서 최초로 빙산질의에 대해 문제를 제기하고, 해시 카운팅[4][5]을 이용한 방법들을 제안하였다. 그러나, 해시 카운팅의 경우 avg 연산에 대해 착오누락이 발생할 수 있다는 문제점으로 인해, count와 sum 두 연산 밖에 지원하지 못한다는 한계를 가지고 있다. 그림 2는 착오누락이 발생하는 예를 보여주고 있다. (b)의 데이타들이 (a)의 해시값 1인 카운터에 해당할 때, 데이타 aa는 임계값 이상으로 결과로 출력되어야 하나, 해시 카운터에서 임계값 미만이므로 결과에서 제외된다.

그래서, [2]에서는 분할에 의해 빙산 질의를 수행하는 방법을 제안하였으며, 이 논문은 특히 avg 빙산 질의를 수행하는

두 알고리즘을 제안하고 있다.

임계값 : 4

해시값	1	2	3
sum	7	3	4
count	2	2	1

(a) 해시 카운터

데이타	aa	ab
sum	5	2
count	1	1

(b) 해시값이 1인 데이타들

그림 2 해시 카운터에 의한 착오누락의 예

3. 동적 분할 알고리즘

이 논문에서 제안하는 알고리즘들이 평균 빙산 질의를 수행하는 과정은 두 단계로 이루어진다. 먼저, 데이타베이스를 스캔하고 분할하여 후보를 찾는 단계와, 다시 데이타베이스를 스캔하며 전 단계에서 찾아진 후보들의 정확한 값을 계산하여 최종 결과를 찾는 단계로 구성된다. 이때 처음 데이타베이스 스캔시 데이타 도메인의 크기가 작아 모든 데이타들의 카운터가 메모리에 거주가능하다면 두 번째 단계없이 최종 결과를 얻을 수 있다.

3.1. 후보 선택의 원리

3.3과 3.4에서 설명되는 알고리즘들에서 각 분할들의 후보 선택은 다음 원리에 따라 이루어진다.

(원리) 평균에 대한 임계값이 T이고, N개의 데이타를 n개의 분할(분할1, 분할2, ... 분할n)로 나누었을 때, 임계값 이상인 데이타들은 적어도 하나의 분할i($1 \le i \le n$)에서 T 이상의 평균값을 가진다. 역은 성립하지 않는다.

(증명) 임계값 이상인 데이타 d가 모든 분할에서 T 미만의 평균값을 가진다고 하자. 이때, d의 평균값은 $\sum^{N} value/N$이고, 이 값은 T보다 작다.

(∵ 모든 분할에서 T 미만의 평균값을 가지므로

$\forall i : \sum^{N} value/N_i < T$ (i = 1, ..., n) ..... ①

이 때, $\sum^{N} value/N = \sum_{i=1}^{n} \sum^{N} value/N$ (분자에 ① 적용)

$< (TN_1+...+TNn)/N = T$)

(N_i : i번째 분할의 데이타 개수)

이는, 데이타 d가 임계값 이상이라는 가정에 모순이다. 그러므로, 적어도 하나의 분할에서 T 이상의 평균값을 가진다. (증명 끝)

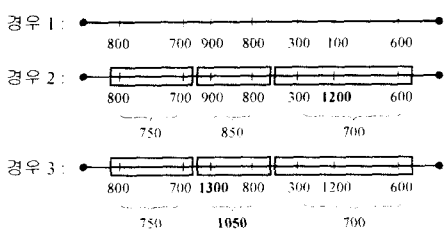
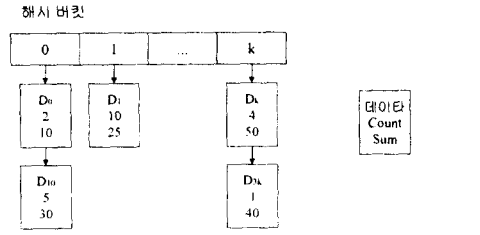


그림 3 후보 선택의 원리가 적용되는 경우들

그림 3은 후보 선택의 원리가 적용된 여러 가지 경우의 예를 보여주고 있다. 경우 1은 D의 평균값도 임계값 미만이고, 모든 분할들의 평균값도 임계값 미만이므로 D는 후보로 선택되지 않는다. 경우 2는 세 번째 분할에서 임계값 이상인 값이 한 번 나오긴 하였지만 분할의 평균값이 임계값 미만이므로 역시 후보가 되지 못한다. 경우 3에서는 두 번째 분할에서 평균값이 임계값 이상으로 후보로 선택되나, 평균값은 임계값 미만으로 최종 결과에 포함되지 못한다.

3.2. 카운터의 구조



(a) 전체 카운터 구조 (b) 하나의 카운터 노드 구조

그림 4 카운터의 자료구조

그림 4의 (b)에 보여지듯이 하나의 카운터 노드는 <데이타, count, sum>으로 구성된다. 이런 카운터 노드들이 데이타의 해시값에 따라 (a)와 같이 리스트를 이루고, 각 리스트들은 데이타에 따라 정렬이 되어있다. 이와 같은 해시를 이용한 동적 카운터 구조를 사용함으로써 데이타베이스로부터 읽혀지는 임의의 데이타에 대해 빠른 카운트 작업을 수행할 수 있다.

3.3. 메모리 Full 분할

'메모리 Full 분할' 알고리즘은 다음과 같다.

① 데이타베이스에서 데이타를 읽는다. 더 읽을 데이타가 없다면 ④를 수행한다.

②-1 읽은 데이타에 대한 카운터가 이미 메모리에 있으면 count, sum을 갱신한다.

②-2 메모리에 없으면, 새로운 카운터를 생성할 수 있는지 검사한다.

③-1 새로운 카운터를 생성할 수 있다면, count는 1로, sum은 데이타의 값으로 설정한 카운터를 생성한다.

③-2 새로운 카운터를 생성할 수 없다면, 카운터의 값들을 검사하여 임계값 이상을 후보로 파일에 기록하고, 모든 카운터를 삭제한다. 그리고, ①을 수행한다.

④-1 후보 기록이 한 번도 없었다면 카운터들을 검사하여 임계값 이상을 결과로 출력한다.

④-2 기록된 후보들을 메모리에 읽어들인 후, 데이타베이스를 다시 스캔하며 각 후보들의 정확한 값을 찾는다. 데이타베이스 스캔이 끝나면 카운터들을 검사하여 임계값 이상을 결과로 출력한다.

3.4. 후보 Full 분할

분할의 길이가 짧다면, 우연히 한 데이타가 그 분할에서 임계값 이상으로 후보로 선택될 가능성이 생긴다. 그러므로 분할

의 길이를 길게 해준다면 각 후보들의 신뢰도는 높아진다. '후보 Full 분할'은 '메모리 Full 분할'에서 각 분할의 길이를 길게 하여 분할의 개수를 줄인 방법이다.

즉, '메모리 Full 분할'에서는 카운터들로 메모리가 가득 찼을 때 후보를 기록하고 메모리를 초기화하였지만, '후보 Full 분할'에서는 메모리가 가득 찼을 때 후보가 아닌 카운터들은 삭제를 하고, 메모리가 후보로만 가득 찼을 때 후보를 기록한다.

'후보 Full 분할' 알고리즘은 다음과 같다.

① 데이타베이스에서 데이타를 읽는다. 더 읽을 데이타가 없다면 ⑤를 수행한다.

②-1 읽은 데이타에 대한 카운터가 이미 메모리에 있으면 count, sum을 갱신한다.

②-2 메모리에 없으면, 새로운 카운터를 생성할 수 있는지 검사한다.

③-1 새로운 카운터를 생성할 수 있다면, count는 1로, sum은 데이타의 값으로 설정한 카운터를 생성한다.

③-2 새로운 카운터를 생성할 수 없다면, 카운터의 값들을 검사하여 임계값 미만의 카운터들을 삭제한다.

④-1 삭제되는 카운터들이 있다면, ①을 수행한다.

④-2 삭제되는 카운터들이 없다면, 모든 카운터들을 파일에 기록하고, 모든 카운터들을 삭제한다. 그리고, ①을 수행한다.

⑤-1 카운터 삭제 작업이나 후보 기록이 한 번도 없었다면 카운터들을 검사하여 임계값 이상을 결과로 출력한다.

⑤-2 기록된 후보들을 메모리에 읽어들인 후, 데이타베이스를 다시 스캔하며 각 후보들의 정확한 값을 찾는다. 데이타베이스 스캔이 끝나면 카운터들을 검사하여 임계값 이상을 결과로 출력한다.

4. 실험

4.1. 실험 환경 및 내용

데이타A와 데이타B 모두 100만개의 데이타로 구성되었다. 데이타A의 도메인 크기는 2267개이고, 각 데이타들이 데이타베이스에 나타나는 회수가 정규분포를 이루도록 생성되었다. 데이타B의 도메인 크기는 10000개이고, 각 데이타들이 데이타베이스에 나타나는 회수는 거의 비슷하도록 생성되었다. 그리고, 데이타A와 데이타B를 각각 정렬한 데이타A'와 데이타B'를 생성하였다. 이 때, 메모리에 생성되는 카운터의 수와 도메인 크기의 비율을 변경시키며, 네 종류의 데이타에 대해 평균 빙산 질의를 실행하였다. 실제로 메모리가 부족한 상황을 만들기는 어려워 메모리에 생성되는 최대 카운터의 수를 조절하였다.

실험은 Intel Celeron 366MHz CPU와 Windows 2000에서 이루어졌다.

4.2. 실험 결과 및 분석

그림 5와 그림 6을 살펴보면 '메모리 Full 분할'보다 '후보 Full 분할'이 우수한 성능을 보였음을 알 수 있다. 또한, 데이타베이스에 나타나는 회수가 비슷할 때보다는 편차가 있을 때 '후보 Full 분할'이 빠르게 연산을 수행하였다.

미리 정렬된 데이타A'와 데이타B'에 대해서는 '카운터수/도메인크기'가 1보다 작을 때 비율에 상관없이 30초 정도의 일정한 성능을 보였다.

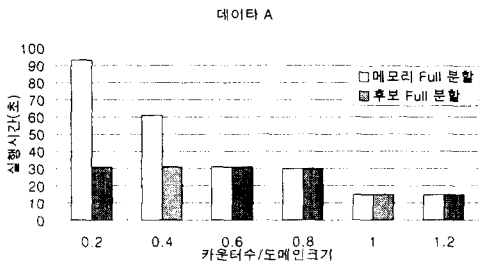


그림 5 데이타A에 대한 평균 빙산 질의 실행

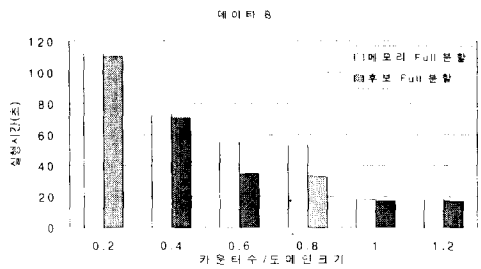


그림 6 데이타B에 대한 평균 빙산 질의 실행

5. 결론

이 논문은 최초로 평균 빙산 질의에 대해 데이타베이스 분할을 통해 효율적으로 카운터를 처리하는 두 알고리즘을 제안하였다. 실험 결과 두 개의 동적 분할 알고리즘들은 메모리의 크기, 데이타의 분포, 데이타의 순서에 영향을 받는데, '후보 Full 분할' 알고리즘이 보다 우수한 성능을 보였다.

참고 문헌

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

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

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

[4] P. Flajolet and G.N. Martin, "Probabilistic counting algorithms for database applicatoins", Journal of Computer System Sciences, 31:182-209, 1985

[5] K. Whang, B.T. Vander-Zanden, and H.M. Taylor, "A linear-time probabilistic counting algorithm for db applications", ACM Transactions on Database Systems, 15(2):208-229, 1990

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