주 메모리 데이타베이스에서의 효율적인 테이블 수직 분할

박현진*, 차재혁**, 송병호***, 이석호*
genie@db.snu.ac.kr, jaehcha@email.hanyang.ac.kr, bhsong@sangmyung.ac.kr, shlee@cse.snu.ac.kr
*서울대학교 전기·컴퓨터공학부, **한양대학교 컴퓨터교육과, ***상명대학교 소프트웨어학과

Efficient Vertical Partitioning in Main Memory Databases

Hyunjin Park*, Jaehyuk Cha**, Byoungho Song***, Sukho Lee*
*School of Electrical Engineering and Computer Science, Seoul National University
**Dept. of Computer Education, Hanyang University
***Dept. of Software Science, Sangmyung University

요 약

주 메모리 데이타베이스 환경에서는 메모리 접근이 성능상의 병목으로 작용하므로 캐시접근 실패를 줄이는 것이 중요하다. 본 논문에서는 데이타베이스 디자인 단계에서 캐시를 고려하여 테이블의 수직 분할을 결정하는 방법을 제안한다. 캐시 접근 실패 횟수를 기반으로 하여 질의처리비용을 예상하는 비용식을 제안하고, 이를 최소로 하는 테이블 수직 분할을 찾는 휴리스틱을 제안한다.

1. 서론

CPU와 메모리 사이의 속도 차이가 점점 커지고 있는 현재의 하드웨어 발전 추세에서는 캐시의 중요성이 더욱 강조된다. 즉 많은 데이타를 처리하는 응용프로그램의 경우 CPU 연산의 수보다는 캐시 접근 실패(cache miss)정도에 의해 그 성능이 좌우되는 것이다. 이에 따라 응용프로그램의 메모리 접근 패턴을 최소한의 캐시 접근 실패를 유발하도록 변경해주는 연구가 진행되고 있다[2].

주 메모리 데이타베이스(Main Memory Databases)란 모든 데이타를 주 메모리에 상주, 디스크 I/O를 최소화하여 최대의 성능을 얻을 수 있도록 하는 시스템을 말한다. 과거의 주 메모리 데이타베이스 연구는 T-트리[5]와 같이 CPU 연산수를 최소로 하는 것을 목표로 하였다. 그러나 최근 연구는 CPU 연산수보다는 캐시 접근 실패를 줄이는데 역점을 두고 있다[1,3].

본 논문에서는 내장형 시스템과 같이 작업부하가 예측 가능한 주 메모리 데이타베이스 환경에서 효율적인 테이블 수직 분할을 찾는 기법을 제시한다. 작업부하에 대한 각 수직 분할 방식들의 질의 처리비용을 예측하는 비용식을 제안하고, 이 비용들의 합을 최소로 하는 분할 방식을 찾을 때 그 탐색 공간을 줄이는 휴리스틱을 제안한다.

기존의 디스크 기반 데이타베이스에서도 테이블 수직 분할에 대한 연구가 있었다[4,6]. 그러나 분할 비용이 크고, 분할된 레코드들을 다시 통합하는 비용이 커서 그 이득이 적었다. 따라서 그 분할의 이득이 명백한 경우에 한해서 부분적으로 사용되어 왔다. 그러나 주 메모리 데이타베이스에서는 레코드 분할 비용이나, 통합 비용이 매우 적을 뿐만 아니라 테이블 스캔 연산에서 큰 이득을 볼 수 있음을 참고 문헌 [1]에서 실험으로 보인바 있다.

본 논문의 구성은 다음과 같다. 2절에서는 기존 테이블 수직 분할 방법과 그 문제점을 살펴본다. 3절에서는 각 수직 분할의 질의 처리비용을 예측하는 비용식을 제시한다. 4절에서는 최적에 가까운 분할을 찾는 휴리스틱을 제안하고 5절에서 결론을 맺는다.

2. 관련 연구

참고 논문 [1]에서는 테이블의 기본 저장 구조로 BAT(Binary Association Tables)을 제안하고 있다. BAT은 테이블들을 애트리뷰트별로 완전히 수직 분할하여 <OID, 애트리뷰트>의 쌍으로 그림 1과 같이 저장하는 것이다.

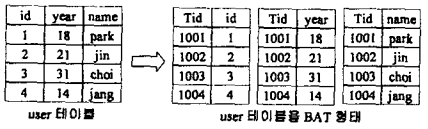

그림 1. BAT구조 [1]

이와 같이 각 애트리뷰트들을 별도로 모아서 저장하게 되면 애트리뷰트 값들 사이의 공간지역성이 증대된다. 따라서 이 애트리뷰트에 대해서 스캔이 있을 때 캐시 접근 실패가 줄어들어 성능이 향상된다.

그러나 이렇게 모든 애트리뷰트별로 완전히 분할하여 저장하는 것은 인덱스로 레코드를 임의 접근(random access)하는 경우, 여러 애트리뷰트를 동시에 접근하는 경우, 갱신이 자주 일어나는 환경에 대해서는 오히려 성능을 악화시키는 문제가 있다. 그러므로 여러 접근 패턴을 고려하여 함께 사용되는 애트리뷰트를 같은 분할에 저장하는 것이 더 효과적일 것이다. 이 논문에서는 이렇게 여러 접근 패턴을 고려하여 더 효율적으로 테이블을 수직 분할하는 방법을 찾는다.

3. 수직 분할의 비용식

본 절에서는 가능한 수직 분할에 따른 질의 처리비용을 예측하는 비용식(cost function)을 제시한다. 이 비용식은 각 분할 방식의 효율성을 예측하기 위한 것이므로 기본 테이블의 분할에 영향을 받지 않는 질의처리비용을 제외하였다. 예를 들어 조인된 테이블에 대해서 집단함수를 적용하는 질의가 있다고 할 때, 테이블의 분할 방식에 영향을 받는 비용은 조인을 하기 위해 각 테이블을 순차적 또는 임의로 접근하는 비용이다. 그 외의 실제 조인을 처리하는 비용이나 집단함수를 적용하는 비용은 기본 테이블의 분할과 관계가 없다. 따라서 기본 테이블에 임의 또는 순차 접근하는 비용만을 계산하였다. 이 비용은 다시 메모리 접근 비용과 CPU에서의 연산 비용으로 나눌 수 있다. 여기서 제시하는 비용식은 메모리 접근 비용 즉 캐시 접근 실패 횟수만으로 한다. 왜냐하면 분할된 레코드를 하나로 통합하는 비용을 제외한 CPU 연산은 분할의 수와 무관할 것이고, 통합을 위한 CPU 연산 비용 역시 매우 간단한 연산으로 이루어져 있기 때문에 캐시 접근 실패를 처리하는 비용과 겹쳐져서 전체 성능에 거의 영향을 주지 않을 것이기 때문이다.

이 절에서는 먼저 캐시 접근 실패에 큰 영향을 주는 바이트 정렬문제를 어떻게 비용식에서 처리했는지를 보이고, 각 질의 형태에 대한 비용식을 제시한다.

3.1. 바이트 정렬

캐시는 32 ~ 64 바이트의 블록 단위로 입출력이 이루어지게 되므로 캐시 블록에 대해서 바이트 정렬이 맞지 않을 경우 부가의 캐시 접근 실패가 발생할 수 있다. 즉 그림 2와 같이 31 바이트의 길이를 갖는 데이타가 있을 때 32 바이트의 블록 크기를 갖는 캐시의 경우 2번의 캐시접근 실패가 발생할 수 있는 것이다. 데이타베이스 연산은 매우 많은 자료를 대상으로 이루어지게 되므로 이와 같은 부가의 접근 실패는 전체 성능을 크게 저하시킬 수 있다.

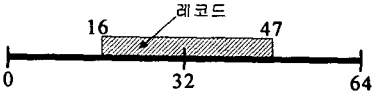

그림 2. Byte Align이 맞지 않는 경우

그림 3은 일정한 길이의 레코드들이 연속적으로 있는 메모리 영역에 임의로 접근할 때 각 레코드 길이에 따른 캐시 접근 실패 수의 기대값을 계산한 것이다.

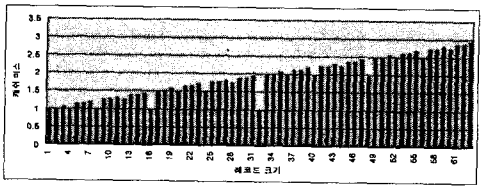

그림 3. 캐시 접근 실패 수의 기대값

레코드의 길이가 $32\times n$, $32\times n+4$, $32\times n+8$, $32\times n+16$일 때는 부가적인 접근 실패가 발생하지 않지만 그 외의 경우 기대값이 점차 증가하는 것을 볼 수 있다. 그러므로 비용식에서는 그 레코드의 길이에 대한 함수의 형태로 반영한다.

3.2 검색 질의의 비용식

```
SELECT id  FROM user
WHERE year > 18 AND name like '박%';
```

위의 질의를 그림 1과 같이 모든 애트리뷰트가 분할 저장된 상태에서 인덱스를 사용하지 않고 처리하는 방법을 생각해보면, year 애트리뷰트를 포함하고 있는 분할을 스캔하여 조건(year > 18)을 만족하는 레코드를 찾고, 이 레코드의 TID(Tuple ID)를 이용하여 이 레코드의 name 애트리뷰트에 접근하여 다른 조건(name

like '박%')을 만족하는지를 검사하고, 만족할 경우 같은 방법으로 id 애트리뷰트를 반환하는 형태가 된다. 즉 선택도가 가장 높은 분할에 대해서 스캔을 하면서 조건을 만족하는 레코드에 대해서만 그 다른 분할에 있는 애트리뷰트를 접근해 가는 것이다. 이를 처리하는 비용은 스캔비용과 다른 애트리뷰트를 임의 접근하는 비용으로 나눌 수 있다. 스캔되는 분할의 레코드의 길이가 L이고 레코드의 수가 m, 캐시 블록 크기를 C라고 하면 하나의 분할을 스캔하는 비용은 $\lceil m \times L/C \rceil$ 이 된다. 이 스캔의 선택도가 S_1이라 하면 다음 조건을 다른 분할에 있는 애트리뷰트를 찾아서 적용하는 비용은 $\lceil m \times S_1 \rceil \times (F(l_2)+n)$이 된다. 여기서 n은 TID를 이용하여 다른 분할에 존재하는 애트리뷰트를 찾는 비용으로 상수 값을 가진다. $F(l_2)$는 레코드의 길이가 l_2인 분할에 접근할 때의 캐시 접근 실패 수의 기대값이다. 이를 일반화하면 질의 Q_1을 처리하기 위해 접근해야 하는 분할이 <1..n>이 있고, 각 분할의 선택도가 $<S_1..S_n>$, 각 분할에서의 레코드의 길이가 $<l_1..l_n>$일 때 모든 i에 대해서 $S_i < S_{i+1}$이 성립한다고 할 때 다음과 같은 식을 얻을 수 있다.

$$\begin{aligned} \textit{Cost Function} = & \lceil (m \times l_1)/c \rceil + \\ & \lceil m \times S_1 \times (F(l_2)+n) \rceil + \ldots . + \\ & \lceil m \times S_1 \times S_2 \times \ldots S_n \times (F(l_n)+n) \rceil \end{aligned}$$

위에서 예로든 질의에서 year 애트리뷰트에 인덱스가 있다면 이를 통해서 해당 조건을 만족하는 RID의 리스트를 구할 수 있다. 그러므로 인덱스에 의해서 스캔이 발생하지 않는 경우는 위의 비용식에서 스캔 비용 부분을 제거한 형태가 된다.

3.3 갱신을 위한 비용식

갱신은 INSERT, DELETE, UPDATE 질의로 나누어 생각할 수 있다.

INSERT 비용에서 분할에 의해 좌우되는 것은 레코드를 각 분할에 나눠서 삽입하기 위해 각 분할을 한 번씩 읽고, 쓰는 비용이다. 이를 3.2절에서의 표기법으로 표현하면 $F(l_1)+(F(l_2)+n)+\ldots+(F(l_n)+n)$와 같은 비용식을 얻을 수 있다. DELETE, UPDATE 질의의 경우 WHERE절의 조건으로 레코드를 찾는 비용과 이를 실제로 갱신, 삭제하는 비용으로 나눌 수 있다. WHERE절의 조건을 만족하는 레코드를 고르는 비용은 3.2절의 비용식을 그대로 사용할 수 있으며, 삭제, 갱신 연산을 하는 비용은 위의 INSERT질의의 비용식을 그대로 사용할 수 있다.

4. 최적의 분할 탐색

최적의 분할은 3절에서 살펴본 비용식을 모든 질의에 적용한 값들의 합을 최소로 하는 것일 것이다. 그러나 m개의 애트리뷰트로 된 테이블에 대해서 가능한 분할의 경우의 수는 거의 m^m이다[6]. 그러므로 모든 가능한 경우에 대해서 비용식을 구해서 최적의 것을 찾는 것은 그 비용이 너무 크다. 이 절에서는 그 최적에 가까운 분할을 찾는 휴리스틱을 제안한다.

1. 초기에 모든 애트리뷰트가 서로 다른 분할에 나눠져 있다고 가정한다.

2. 각 애트리뷰트에 대한 스캔 연산의 수를 조사해서, 스캔되지 않는 애트리뷰트를 하나의 분할로 묶는다.

3. 분할들의 쌍 중에서 결합하여 하나의 분할로 만들었을 때 가장 전체 질의처리비용을 줄이는 것을 선택해서 하나의 분할로 만드는 과정을 반복한다. 이때 각 분할에 대한 캐시 접근 실패 수의 기대값들의 최대값과 결합한 분할에 대한 캐시 접근 실패 수의 기대값의 정수 부분이 바뀌지 않도록 한다.

4. 스캔 빈도가 적은 분할부터 바이트 정렬을 맞추어 주었을 때 전체 질의처리비용을 구해서 비용이 줄어드는 경우 그 길이를 $32\times n$, $32\times n+4$, $32\times n+8$, $32\times n+16$으로 만들어 준다.

5. 결론

주 메모리 데이타베이스 환경에서는 테이블을 효율적으로 수직 분할하여 저장하는 것이 질의처리 비용을 줄이는데 크게 도움을 준다. 이 논문에서는 각 수직 분할 방식에 따른 질의처리비용을 예측할 수 있는 비용식을 제안하였고, 이를 기반으로 하여 효율적인 수직 분할을 찾는 휴리스틱을 제안하였다.

추후 연구로는 실제 구현을 통해 DBMS 모델과 하드웨어에 따른 수직 분할 비용식의 정확도를 측정하고 이를 보정하는 것과 데이타베이스 튜닝 단계에서 효율적인 수직 분할을 찾는 연구가 필요하다.

참고 문헌

[1] S. Manegold, P. Boncz, M. Kersten, "Database Architecture Optimized for the new Bottleneck: Memory Access" VLDB 1999

[2] T. Chilimbi, M. Hill, J. Larus, "Cache Conscious Structure Layout", PLDI 1999

[3] A. Shatdal, C. Kant, J. Naughton "Cache Conscious Algorithm for Relational Query Processing", VLDB 94

[4] D. Cornell, P. Yu, "Vertical Partitioning Algorithm for Relational Databases", ICDE 1987

[5] T. Lehman, "A Study of Index Structures for Main Memory Database Management System", VLDB 1986

[6] M. Hammer, B Niamir, "A Heuristic Approach to Attribute Partitioning" SIGMOD 1979