데이터 웨어하우스에서 해쉬 테이블을 이용한 효율적인 데이터 큐브 생성 기법

김형선°*, 유병섭*, 이재동**, 배해영*
*인하대학교 컴퓨터·정보공학과
**단국대학교 정보컴퓨터학부
{sunnymsg°, subi}@dblab.inha.ac.kr and letsdoit@dankook.ac.kr and hybae@inha.ac.kr

Efficient Creation of Data Cube Using Hash Table in Data Warehouse

Hyungsun Kim°*, Byeongseob You*, Jaedong Lee**, Haeyoung Bae*
*Dept. of Computer Science & Information Engineering, Inha University
**Division of Information and Computer Science, Dankook University

요 약

데이터 웨어하우스는 축적된 대량의 데이터를 분석하여 의사결정을 지원하는 시스템이다. 의사결정을 위한 대량의 데이터 분석은 많은 비용을 요구하므로, 질의 처리 성능을 높이고 의사 결정자에게 빠른 응답을 제공하는 효율적인 데이터 큐브 생성 기법이 연구되었다. 기존 기법으로는 Multiway Array 기법과 H-Cubing 기법이 있다. Multiway Array 기법은 다차원 집계 연산에 필요한 모든 데이터를 배열로 저장하는 것으로 데이터의 양이 많아질수록 메모리 사용이 증가한다. H-Cubing 기법은 Hyper-Tree를 기반으로 튜플을 트리로 구축하므로 모든 튜플을 트리로 구축해야 하는 비용이 증가한다.

본 논문에서는 데이터 웨어하우스에서 해쉬 테이블을 이용한 효율적인 데이터 큐브 생성 기법을 제안한다. 제안 기법은 데이터 큐브 생성 시 필드 해쉬 테이블과 레코드 해쉬 테이블을 사용한다. 필드 해쉬 테이블은 저장될 레코드 순서 계산을 위하여 각 필드에 대해 레벨 값을 해쉬 테이블로 관리한다. 레코드 해쉬 테이블은 데이터 큐브 테이블에 저장될 레코드의 순서와 데이터 큐브 테이블에 저장하기 위한 임시 레코드의 위치를 관리한다. 필드 해쉬 테이블을 이용하여 다차원 데이터의 저장될 레코드 순서를 빠르게 찾아 저장함으로서 데이터 큐브의 생성속도가 향상된다. 또한 해쉬 테이블 만을 유지하면 되므로 메모리 사용량이 감소한다. 따라서 해쉬 테이블의 사용으로 데이터의 빠른 검색과 데이터 큐브 생성 요청에 빠른 응답이 가능하다.

1. 서 론

데이터 웨어하우스(Data Warehouse)는 축적된 대량의 데이터를 분석하여 의사결정을 지원하는 시스템으로 크게 서버(Server)와 구축기(Builder)로 이루어져있다[1, 2, 3]. 데이터 웨어하우스 서버는 OLAP(On-Line Analytical Processing) 질의 처리 성능을 높이고 의사결정자에게 빠른 응답을 제공하기 위해 구축기로부터 적재된 차원테이블의 다차원 분석 결과를 저장하고 있는 데이터 큐브를 생성한다[4, 5]. 다차원 데이터 집계 연산 결과를 계산하고 관리하는 데이터 큐브는 대용량 데이터 분석을 위해 많은 시간과 비용을 요구하여 효율적인 데이터 큐브 생성 기법이 연구되었다.

효율적인 데이터 큐브를 생성하기 위한 기존 기법은 Multiway Array 기법과 H-Cubing 기법이 있다[6, 7]. Multiway Array 기법은 모든 데이터를 배열에 저장하여 다차원 집계 연산을 하는 기법이다. 하지만 데이터양이 많아질수록 필요로 하는 배열이 증가하므로 다차원 집계 연산을 위한 메모리 사용이 증가한다. H-Cubing 기법은 Hyper-Tree를 기반으로 하며 다차원 집계 연산을 위한 데이터를 모두 트리로 구성하여 관리한다. 하지만 다차원 집계 연산을 위해 모든 튜플을 트리로 구축해야 하는 비용과 다차원 집계 연산을 위한 트리 검색 비용이 든다.

본 논문에서는[1] 데이터 웨어하우스에서 해쉬 테이블을 이용한 효율적인 데이터 큐브 생성 기법을 제안한다. 제안 기법은 데이터 큐브 생성 시 필드 해쉬 테이블과 레코드 해쉬 테이블을 사용한다. 필드 해쉬 테이블은 저장될 레코드의 순서를 위해 사용하는 것으로, 각 필드 레코드를 중복 없이 해쉬 테이블로 작성하고 순차적인 레벨 값을 할당하여 관리한다. 레코드 해쉬 테이블은 데이터 큐브에 저장하기 위한 레코드 순서와 다차원 집계 연산 결과를 저장하기 위한 임시 레코드의 위치를 관리한다. 레코드 해쉬 테이블의 순서는 각 필드의 필드 해쉬 테이블 레벨 값을 계산하여 얻으며, 실제 저장된 레코드의 위치 값을 저장한다. 제안 기법을 적용한 데이터 큐브 생성은 해쉬 테이블의 사용으로 데이터 큐브 생성 요청에 대한 빠른 응답과 다차원 집계 연산 결과에 대한 빠른 처리가 가능하다.

본 논문의 구성은 다음과 같다. 2장에서는 관련 연구에 대해 설명하고, 3장에서는 제안하는 해쉬 테이블을 이용한 데이터 큐브 생성 기법에 대해 기술한다. 4장에서는 성능 평가를 한 후, 5장에서 결론 및 향후 연구에 대하여 기술한다.

2. 관련 연구

1) 본 연구는 정보통신부 및 정보통신연구진흥원의 대학 IT연구센터 육성 지원사업의 연구결과로 수행되었음.

2.1 Multiway Array를 사용한 데이터 큐브 생성

Multiway Array 기법은 배열을 이용한 데이터 큐브 생성 기법이다. 데이터 큐브 생성 시 필요한 모든 다차원 테이블의 튜플을 배열에 저장하여 다차원 집계 연산을 수행한다[6]. 데이터 큐브 생성 시 필요로 하는 필드가 N개라면, 모든 튜플을 저장하기 위해 N차원 배열 생성한다. 따라서 데이터 큐브 생성 시 필요한 필드가 증가하면 필요한 배열의 크기도 증가하므로 메모리 사용이 증가한다. 또한 다차원 집계 연산이 완료되면, 배열에 저장된 다차원 집계 값을 데이터 큐브 테이블에 저장하는 비용이 발생한다.

2.2 H-Cubing을 사용한 데이터 큐브 생성

H-Cubing 기법은 H-Tree(Hyper-Tree) 자료구조를 이용한 데이터 큐브 생성 기법이다[7]. H-Cubing 기법은 데이터 큐브를 생성하기 위해 필요한 모든 필드를 트리로 구성한다. 이때 각 튜플은 하나의 간선으로 표현되며, 같은 필드 값을 가지는 튜플에 대해서는 서브 간선으로 표현된다. 이와 같이 H-Cubing을 사용한 데이터 큐브 생성 기법은 모든 튜플을 트리로 구축해야 하므로, 트리 구축을 위한 튜플 검색 비용이 증가한다. 또한 다차원 집계 연산을 하기 위해서는 트리를 완성하고, 모든 트리를 검색해야하므로 연산비용이 증가한다.

3. 해쉬 테이블을 이용한 데이터 큐브 생성 기법

3.1 데이터 큐브 생성을 위한 필드 해쉬 테이블

제안 기법을 적용한 데이터 큐브 생성은 우선적으로 필드 해쉬 테이블이라고 하는 해쉬 테이블을 생성한다. 필드 해쉬 테이블은 저장할 레코드의 순서를 계산하기 위한 것으로 각 필드마다 중복 없는 레코드 값을 이용해 각각의 독립적인 해쉬 테이블로 만든다. 필드 해쉬 테이블에 입력된 레코드 값에는 레코드 순서 계산을 위한 레벨 값으로 순차적인 값이 할당되며, 이러한 레벨 값을 이용해서 저장 레코드 순서를 계산한다. 그림 1은 각 필드 해쉬 테이블에 '0'부터 순차적인 값을 할당하여, n까지 레벨 값이 저장된 것을 보여준다.

또한 필드 레코드에 대한 레벨 값을 할당한 후에는 필드의 일반화 값인 'Total'레코드와 'Total'레코드의 레벨 값을 추가한다. 필드 해쉬 테이블에 'Total'레코드를 포함하는 이유는 다차원 집계 연산에서 일반화에 필요하기 때문이다.

이러한 필드 해쉬 테이블은 큐브 연산 결과 순서를 맞추기 위해 필드를 이름순으로 정렬해야한다. 이름순으로 정렬된 필드는 필드 해쉬 테이블 생성 시 중복되지 않는 레코드 값을 구분하는 비용을 줄여준다. 필드 해쉬 테이블은 저장하기 위한 레코드 순서를 계산하는데 있어, 각 레코드의 값을 빠르게 찾아 계산할 수 있다.

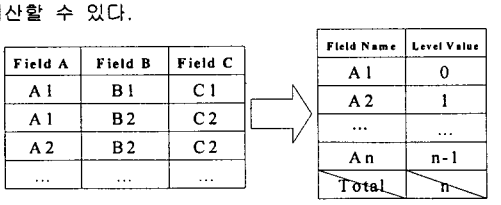

그림 1 데이터 큐브 생성을 위한 필드 해쉬 테이블

3.2 데이터 큐브 생성을 위한 레코드 해쉬 테이블

필드 해쉬 테이블이 생성되면, 다음으로 필드 해쉬 테이블을 이용한 레코드 해쉬 테이블을 생성하게 된다. 레코드 해쉬 테이블은 필드 해쉬 테이블의 레벨 값으로 계산된 레코드 순서에 대해 실제 레코드 저장 위치를 관리하는 해쉬 테이블 이다. 그림 2와 같이 레코드 해쉬 테이블은 데이터 큐브 테이블에 저장될 순서와 임시 레코드의 주소를 저장하는 구조로 구성된다. 이때 저장 가능한 임시 레코드의 최대 개수는 데이터 큐브 생성에 필요한 필드의 '중복되지 않는 레코드 개수 +1'의 값을 모두 곱한 값이다. 이는 데이터 큐브를 생성하기 위해 모든 경우에 대한 다차원 집계 연산을 저장 가능하도록 하기 위함이다. 그림 2의 레코드 해쉬 테이블을 보면 특정 레코드에 대해 저장할 레코드 순서 값에 실제 저장위치가 매핑되어 있는 것을 볼 수 있다. 레코드의 실제 저장은 계산된 레코드들 순서로 하기 때문에 레코드 해쉬 테이블의 순서 값으로 결과의 순서를 보장한다. 따라서 레코드 해쉬 테이블은 큐브 연산의 결과 순서를 유지하면서 저장 공간을 효율적으로 사용할 수 있다.

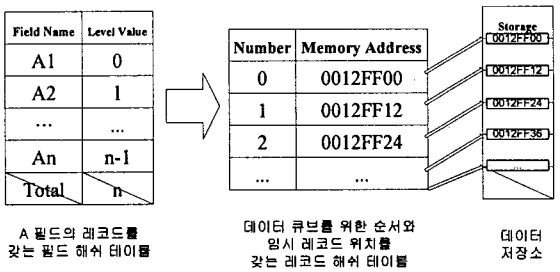

그림 2 데이터 큐브 생성을 위한 레코드 해쉬 테이블

3.3 데이터 큐브 생성을 위한 다차원 튜플의 위치 계산

필드 해쉬 테이블과 레코드 해쉬 테이블이 생성되면 다차원 분석의 기반 데이터를 읽어 연산을 수행한다. 다차원 분석을 위해 하나의 레코드마다 필드 해쉬 테이블의 레벨 값을 사용하여 레코드 해쉬 테이블의 순서 값을 계산한다. 하나의 튜플을 계산하기 위해서는 각 필드에 따라 다른 계산을 한다. 이는 큐브 연산이 필드 순으로 각 필드의 레벨 값은 뒤에 존재하는 각 필드들의 [최고 레벨 값+1]을 곱한다.

예를 들어, 그림 3과 같이 다차원 테이블에 필드 'A, B, C'의 튜플 'A2, B2, C2'를 계산하여 레코드 해쉬 테이블의 위치를 찾는다고 하자. 우선 A필드의 필드 해쉬 테이블을 참조하여 'A2'레벨 값 '1'을 가져온다. 그 뒤에 'B2'와 'C2'의 필드에 대해서는 각 필드 해쉬 테이블의 [최고 레벨 값+1]을 참조하게 된다. 그렇게 해서 나온 레코드 해쉬 테이블의 순서는 1×3×3 = '9'가 된다. 다음으로 필드 B에 대한 계산을 하면 1×3='3'이 된다. 또한 필드 C는 마지막 필드이므로 레코드 값인 '1'이 된다. 따라서 저장할 레코드 순서는 [필드 A의 순서 값]+[필드 B의 순서 값]+[필드 C의 순서 값]인 '13'이 된다.

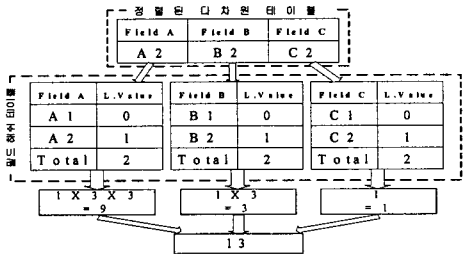

그림 3 다차원 튜플의 위치 검색

3.4 데이터 큐브 생성

데이터 큐브 생성은 필드 해쉬 테이블과 레코드 해쉬 테이블을 생성한 뒤, 연산에 필요한 레코드를 읽어 처리한다. 하나의 레코드를 읽으면 3.3장의 계산 방법을 이용하여 레코드의 순서 값을 계산하고 실제 저장한 위치 값을 레코드 해쉬 테이블의 계산된 순서 위치에 저장한다. 이때 하나의 레코드를 읽으면 큐브에 관련된 일반화 레코드에도 적용하게 된다. 일반화 레코드란 큐브 연산의 필드 전체적으로 'Total'값이 하나 이상 존재하는 레코드를 말한다. 즉 'A1, B1, C1'레코드의 경우 'A1, B1, Total'이 하나의 일반화 레코드이다. 이때 큐브 연산의 각 필드마다 'Total'값이 올 수 있으므로 하나의 레코드는 2^n-1개의 일반화 레코드를 생성하게 된다. 각 일반화 레코드에 대하여 레코드 해쉬 테이블의 순서 값을 계산하고 저장 위치 값을 해당 버켓에 저장한다. 이때 일반화 레코드는 여러 번 저장하는 경우가 발생한다. 예를 들어 'A1, B1, C1'레코드와 'A1, B1, C2'레코드는 'A1, B1, Total'이라는 일반화 레코드가 발생한다. 이때 레코드 해쉬 테이블에서 레코드 순서에 해당하는 버켓을 찾고, 이미 저장되어 있었다면, 실제 저장된 위치 레코드를 읽어 다차원 집계 값을 변경한다.

4. 성능 평가

본 장에서는 앞서 기술한 관련 연구의 Multi-Way Array 기법을 이용한 데이터 큐브 생성기법과 H-Cubing 기법 그리고 본 논문에서 제안한 해쉬 테이블을 이용한 데이터 큐브 생성 기법을 이용한 성능 평가를 하였다.

성능 평가를 하기 위한 시스템 환경은 Pentium4 3.0 GHz CPU와 1 Gigabyte Memory, 160 Gigabyte HDD로 구성하였으며 Windows 2000 Professional 운영체제를 사용한다. 성능 평가에 사용된 테스트셋은 약 5000개의 레코드를 가진 10개의 차원 테이블로 구성하였다.

성능 평가는 테스트셋을 기반으로 1개부터 10개까지의 차원 테이블을 이용한 데이터 큐브 생성 시간의 변화를 측정하였다.

그림 4를 보면, 제안 기법이 기존 기법보다 최대 약 65% 성능 향상된 것을 볼 수 있다. 이는 Multi-way Array 기법의 경우, 데이터 큐브 생성 시 메모리에 모든 튜플을 적재하여 다차원 집계 연산 수행으로 데이터 큐브에서 필요로 하는 차원 테이블의 개수가 증가하면 많은 메모리를 요구하여 가상 메모리를 사용하므로 데이터 큐브의 생성 시간 증가하며, H-Cubing 기법의 경우, 데이터 큐브 생성 시 모든 튜플을 트리로 구성하여 다차원 집계 연산 수행으로 데이터 큐브에서 필요로 하는 차원 테이블의 개수가 증가하면 트리 구축비용과 다차원 집계 연산을 위한 트리 검색 비용이 증가한다. 제안 기법의 경우 해쉬 테이블만 메모리에 유지하면 되므로 메모리 사용량이 적고 빠르게 연산이 가능하다.

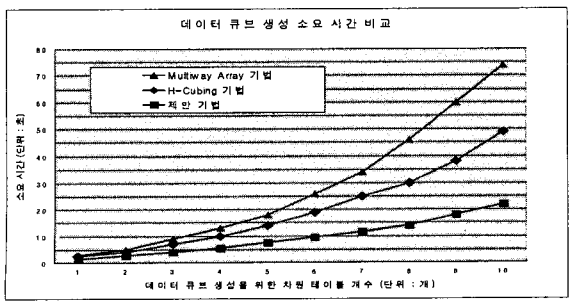

그림 4 데이터 큐브 생성 소요 시간 비교

5. 결론 및 향후 연구

본 논문에서는 데이터 웨어하우스에서 해쉬 테이블을 이용한 효율적인 데이터 큐브 생성 기법에 대하여 제안하였다.

제안 기법은 데이터 큐브의 빠른 생성을 위해 필드 해쉬 테이블과 레코드 해쉬 테이블을 사용한다. 필드 해쉬 테이블은 중복되지 않는 레코드 값으로 구성하여 레벨 값이 저장된다. 이는 데이터 큐브 생성 시 각 레코드의 저장할 레코드 순서 값을 계산하는데 이용한다. 레코드 해쉬 테이블은 레코드의 순서와 실제 저장 위치를 관리한다. 따라서 해쉬 테이블을 이용하여 레코드 순서를 빠르게 계산할 수 있으며 큐브 결과의 순서를 유지할 수 있다. 또한 실제 저장을 결과 순서와 상관없이 저장하므로 저장 공간의 효율성을 증가시킨다. 이는 레코드의 빠른 검색과 데이터 큐브의 빠른 생성을 가능하게 하며 메모리 공간의 사용을 줄일 수 있다.

향후 연구로는 데이터 웨어하우스에서 기반이 되는 다차원 테이블 데이터의 수정과 삭제 등의 변화를 구축된 데이터 큐브로의 빠르게 반영하는 연구가 필요하다.

참고 문헌

[1] 김형선, 유병섭, 박순영, 이재동, 배해영, "공간 데이터 웨어하우스 구축기에서 추출된 데이터의 효율적인 적재를 위한 테이블 단위의 데이터 관리 기법", 한국정보과학회, pp. 79-81, 2005.

[2] S. Chaudhuri and U. Dayal, "An Overview of Data Warehousing and OLAP Technology", ACM SIGMOD Record 26(1), pp. 65-74, 1997

[3] J. Gray, S. Chaudhuri, A. Bosworth, A. Layman, D. Reichart, M. Venkatrao, F. Pellow, and H. Pirahesh. "Data Cube: A relational aggregation operator generalizing group-by, cross-tab, and sub-totals", Data Mining and Knowledge Discovery, pp. 1:29-54, 1997

[4] D.Quess, A. Gupta, I. S. Mumick, and J. Windom. "Making Views Self-Maintainable for Data Warehousing", Proc. of Conf. on Parallel and Database Information Systems, pp. 158-169, 1996.

[5] Paulraj Ponniah, "Data Warehousing Fundamentals: A Comprehensive Guide for IT Professionals", John Wiley, 2002

[6] Yihong Zhao, Prasad M. Deshpande, Jeffrey F. Naughton, "An Array-Based Algorithm for Simultaneous Multidimensional Aggregates", ACM SIGMOD, pp. 159-170, 1997

[7] Jiawei Han, Jian Pei, Guozhu Dong, Ke Wang, "Efficient Computation of Iceberg Cubes with Complex Measures", ACM SIGMOD, pp. 1-12, 2001