# 큐보이드 전위트리를 이용한 빙산질의 처리
## (Iceberg Query Evaluation Technical Using a Cuboid Prefix Tree)

한 상 길 [†] 양 우 석 [††] 이 원 석 [†††]
(Sang Gil Han) (Woo Sock Yang) (Won Suk Lee)

**요 약** 무한한 데이터 스트림을 저장하는 것은 거의 불가능하기 때문에 데이터 스트림 환경에서 빙산질의를 수행하기 위해서는 새로운 데이터 구조와 알고리즘이 요구된다. 본 논문에서는 데이터 스트림 환경에서 빙산질의를 처리하기 위해 전위트리 구조에 기반한 큐보이드 전위트리(Cuboid prefix tree)를 제안한다. 큐보이드 전위트리는 빙산질의에 사용된 그룹항목으로 이루어진 항목집합만을 트리에서 관리하므로 전위트리보다 적은 메모리를 사용한다. 1-항목 관리를 통해서 빈발하지 않은 항목을 트랜잭션에서 제거함으로써 갱신 시 불필요하게 소요되는 시간을 줄일 수 있다. 또한 다중 빙산질의에서 공통적으로 사용된 그룹속성에 따라 노드를 공유함으로써 적은 메모리를 사용하여 효율적으로 다중 빙산질의를 처리할 수 있는 방법을 제안한다. 큐보이드 전위트리는 무한히 연속적으로 생성되는 데이터에 대하여 빙산질의를 처리하는데 있어서 메모리 사용량과 처리시간을 효과적으로 줄이며, 이를 여러 실험을 통해 확인하였다.

**키워드** : 빙산질의, 데이터스트림, 데이터 마이닝, 큐보이드 전위트리, 집계연산

***Abstract*** A data stream is a massive unbounded sequence of data elements continuously generated at a rapid rate. Due to the characteristics of a data stream, it is impossible to save all the data elements of a data stream. Therefore it is necessary to define a new synopsis structure to store the summary information of a data stream. For this purpose, this paper proposes a cuboid prefix tree that can be effectively employed in evaluating an iceberg query over data streams. A cuboid prefix tree only stores those itemsets that consist of grouping attributes used in GROUP BY query. In addition, a cuboid prefix tree can compute multiple iceberg queries simultaneously by sharing their common sub-expressions. A cuboid prefix tree evaluates an iceberg query over an infinitely generated data stream while efficiently reducing memory usage and processing time, which is verified by a series of experiments.

**Key words** : Iceberg query, Data Stream, Data mining, Cuboid Prefix Tree, Aggregation

## 1. 서 론

### 1.1 연구 배경

다차원 관계 데이터베이스나 데이터웨어하우스 시스템에서 GROUP-BY구문을 통한 집계 연산은 방대한 양의 데이터를 그룹속성에 따라 요약하기 위해 사용되는 중요한 연산이다. 그러나 집계함수 연산에 이용되는 차원의 수가 증가하거나 각 차원의 수가 증가한다면 생성되는 데이터 항목집합의 수는 기하급수적으로 증가하게 된다. 대부분의 경우에 있어서 사용자는 전체 질의 결과보다는 특정 임계값(threshold)을 넘어가는 데이터 항목집합에 관심을 갖는다. 이와 같이 GROUP-BY 질의를 적용하여 집계함수를 수행한 뒤, 사용자가 미리 정의한 임계값 이상인 데이터 항목집합만을 결과로 반환하는 연산을 빙산질의(Iceberg Query)라고 한다. 이러한 빙산질의는 데이터웨어하우징이나, 데이터마이닝의

· 이 논문은 2008년도 정부(교육과학기술부)의 재원으로 한국과학재단의 국가지정연구실사업(No.R0A-2006-000-10225-0)및 특정기초연구(R01-2006-000-11223-0)지원으로 수행된 연구임

† 정 회 원 : LG 전자
girihan@hotmail.com
†† 학생회원 : 연세대학교 컴퓨터과학과
bidol@database.yonsei.ac.kr
††† 종신회원 : 연세대학교 컴퓨터과학과 교수
leewo@database.yonsei.ac.kr

논문접수 : 2008년 12월 15일
심사완료 : 2009년 2월 17일

빈발항목 탐색이나 연관규칙 탐색, 클러스터링, 침입탐지 등 많은 응용프로그램에서 널리 사용되고 있다.

최근 많은 응용 도메인에서 생성되는 데이터들은 네트워크 트래픽 모니터링, 증권, 날씨, 콜 센터, 웹 클릭 데이터 등과 같이 빠른 속도로 생성됨과 동시에 지속적으로 발생되는 데이터로 구성된 데이터 스트림 형태이다. 이러한 데이터 스트림을 기존의 방법으로 저장하고, 그 저장된 데이터 위에서 빙산질의를 수행하기에는 저장 공간과 처리시간의 측면에서 많은 어려움이 따른다. 또한 데이터 스트림을 처리하기 위해서는 다음과 같은 제약 사항을 만족해야 한다[1]. 첫째 각 데이터 항목은 단 한번만 읽고 처리해야한다. 둘째 메모리는 유한하게 제한된다. 셋째 새로 생성되는 데이터 객체들은 가능한 빨리 처리되어야 한다. 마지막으로 최신 결과는 필요 시 즉시 사용가능해야 한다.

기존의 빙산질의에 관한 연구는 주로 저장된 데이터를 기반으로 정렬과 해싱방법을 이용하여 데이터를 여러번 반복해서 읽게 되는데 이때 발생하는 I/O 비용과 연산처리 비용을 최소화하기 위한 연구가 주를 이루었다. 그러나 빙산질의를 수행하기 위해서는 최소 2번이상의 데이터 테이블 스캔이 필요하며, 빙산질의를 수행하는 중간 단계에서 생성되는 후보항목집합의 수가 커지는 문제점이 있다. 앞서 언급했듯이 데이터 스트림의 특성상 기존의 방법으로는 데이터 스트림 환경에서 빙산질의를 수행하기가 불가능하다. 따라서 데이터 스트림 환경에서 빙산질의를 처리하기 위해서는 새로운 데이터 구조와 원 패스(one-path) 알고리즘이 필요하다.

본 논문에서는 *estDec* 방법에서 데이터스트림 환경에서의 빈발항목집합 탐색을 위해 사용한 전위트리를 빙산질의에 적합하게 변형한 큐보이드 전위트리(*Cuboid Prefix tree*)를 제안한다. 큐보이드 전위트리는 빙산질의의 GROUP BY 절에 사용된 그룹속성으로 구성된 전위 항목집합만을 관리하므로 트랜잭션 내의 모든 항목집합을 관리하는 전위트리에 비해 적은 메모리를 사용한다. 또한 데이터스트림 처리에 사용되었던 전위트리를 기반으로 함으로써, 데이터스트림에 대한 빙산질의를 가능하게 한다. 한편, 다중 빙산질의에서 공통적으로 사용되는 속성의 항목들을 트리의 상위 레벨에 위치시킴으로써 노드를 공유하여 다중 빙산질의를 처리한다.

논문의 구성은 다음과 같다. 2장에서는 관련연구를 살펴보고, 3장에서 빙산질의를 위한 데이터 스트림의 기본 정의 및 큐보이드 전위트리의 기본이 되는 전위트리와 *estDec* 방법을 소개한다. 4장에서는 큐보이드 전위트리의 구조 및 관리 방법에 대해 자세히 기술하고 5장에서는 다양한 실험을 통해 제안된 방법의 효율성을 검증한다. 끝으로 6장에서 본 논문의 결론을 맺는다.

## 2. 관련 연구

최근 데이터 스트림 환경에서 연속적인 집계함수 질의를 처리하기 위해 많은 알고리즘들이 제안되었다. [2]는 질의 조건과 윈도우의 크기가 변화하는 환경에서 집계함수 질의를 수행할 때 효율적으로 집계함수 질의를 공유할 수 있는 방법을 제안하였다. 서로 다른 크기의 윈도우에서 수행되는 질의들을 처리하기 위한 *Shared Time Slices* 기법과 서로 다른 조건을 가진 질의들의 처리하기 위해 *Shared Data Fragments*기법을 혼합한 *Shared Data Shards* 기법을 통해 서로 다른 조건과 윈도우 크기를 갖는 집계함수 질의를 효율적으로 공유하는 방법을 제안하였다. [3,4]는 연속 질의 처리 시스템의 필터에 다양한 조건 인덱스를 적용하여 자원을 공유하는 방법을 제안하였고, [5,6]은 데이터 스트림 환경에서 많은 수의 슬라이딩 윈도우 집계 연산을 수행할 때 자원을 공유하는 방법을 제안하였다. [7]에서는 LFTA와 HFTA 2레벨 구조를 갖는 시스템을 기반으로 데이터 스트림 환경에서 효율적으로 다중 집계 질의를 처리하는 방법을 제안하였다. 이 방법은 *phantoms*을 이용하여 다중 집계 질의를 처리하는데 집계 질의 사이에 공유할 수 있는 부분 연산을 공유함으로써 전체적인 연산 비용을 줄인다. 이러한 집계 질의에 관한 이전 연구들은 트랜잭션 발생 시간이나 트랜잭션의 수로 정해지는 작은 크기의 윈도우 범위 내에서 수행되며 대부분 질의 최적화 방법에 더 가깝다.

빙산질의(Iceberg Query)는 기본적으로 질의에 사용되는 그룹속성에 따라 분류된 항목집합에 COUNT, SUM, AVG 등과 같은 집계함수를 적용한 집계 값들 중에서 사전에 정의한 임계값을 넘는 값을 결과로 반환하는 연산이다. [8]에서 제시한 빙산질의의 아이디어는 사용자가 정의한 임계값 $T$를 만족하는 SQL 질의의 GROUP BY절에 대한 집계함수 결과를 찾는 것이다. 기본 알고리즘은 크게 두 단계로 나타낼 수 있다. 첫 번째 단계에서는 각 항목의 빈발횟수를 유지한다. 발생한 각 항목은 해쉬 함수를 통해 빈발횟수가 계산된다. 이렇게 계산된 각 항목의 빈발횟수는 비트맵으로 압축되며, 비트맵에서 1로 표시되는 항목이 빈발한 항목임을 나타낸다. 두 번째 단계에서는 첫 번째 단계에서 관리된 항목들 중 비트맵이 1로 표시된 항목들의 정확한 빈발횟수를 계산한다. 그러나 이 알고리즘은 데이터를 여러 번 반복적으로 읽어야 하기 때문에 데이터를 한번 읽고 처리해야 하는 데이터 스트림 환경에 적용하기 어렵다.

빙산질의와 데이터 마이닝 분야의 빈발항목 집합 탐색 알고리즘에서 빈발도는 사용자가 미리 정의한 임계값을 넘는 항목집합만을 결과로 반환한다는 점에서 공통점이 있다. 그러나 트랜잭션의 길이가 $n$일 때, 빈발항

목탐색에서는 트랜잭션내의 항목들의 멱집합에 해당하는 모든 항목집합들의 빈발횟수를 계산하여 이중 사용자가 미리 정의한 임계값을 넘는 모든 항목집합을 결과로 반환한다. 반면 빙산질의에서는 GROUP BY 절에 $n$개의 속성이 사용되었다면 $n$-항목집합의 빈발횟수를 계산하여 이중 임계값을 넘는 $n$-항목집합만을 결과로 반환하는 차이점이 있다.

기존의 빈발항목 집합탐색 방법에 관한 연구들을 살펴보면, 먼저 저장된 데이터 집합에서 빈발항목 집합을 탐색하기 위해 *Apriori* 알고리즘[9], *DIC*[10], *Partition*[11], *Carma* 알고리즘[12]등 많은 연구가 진행되었다. 또한 새로운 트랜잭션이 지속적으로 발생하는 데이터 집합에서는 *FUP-based* 알고리즘[13], *DAEMON* 알고리즘[14]과 같은 점진적 마이닝 알고리즘을 통해 효율적으로 빈발항목집합을 탐색할 수 있다. 그러나 이러한 기존의 알고리즘들은 데이터 집합을 여러번 검색해야 하고 트랜잭션 정보를 관리해야 하므로 데이터 스트림 환경에서 적용하기는 적합하지 않다.

최근에 제안된 *sticky sampling, Lossy Counting* 알고리즘[15] 및 *estDec* 알고리즘[16]은 데이터 스트림 환경에서의 빈발항목집합 탐색에 중점을 두고 있다. *LossyCounting*알고리즘[15]에서는 빈발항목집합 탐색 과정에서 메모리 사용량을 일정 범위로 한정하기 위해서 출현 빈도수 관리 대상 항목집합들을 보조 저장장치에 관리하며 메모리 상에는 일괄 연산 수행을 위한 버퍼만을 유지한다. 이 알고리즘에서는 보조기억 장치를 사용하기 때문에 수행시간이 늦어지는 단점이 있다. *estDec*알고리즘[16]에서는 데이터 스트림에서 발생한 항목집합들 중에서 사전 정의된 지연 추가 및 전지 임계값 $S_{sig}(S_{sig} \leq S_{min})$ 이상의 지지도를 갖는 항목집합들을 가까운 미래에 빈발항목집합이 될 수 있는 중요 항목집합으로 간주하여 이 항목집합들만을 메모리 상에서 관리한다. 이를 통해 온라인 데이터 스트림에 대한 빈발항목집합 탐색 과정에서 메모리 사용량을 감소시킨다. *estDec*방법에서 출현 빈도수 관리 대상 항목집합들은 전위트리 래티스 구조로 메모리상에서 관리된다. 본 논문에서는 최대빈발항목집합 탐색 방법인 *estDec* 방법을 응용하여 고안한 큐보이드 전위트리를 이용하여 빙산질의를 처리하는 방법을 제안한다.

## 3. 전위트리와 *estDec* 방법

빙산질의를 위한 데이터 스트림은 지속적으로 발생되는 트랜잭션의 무한집합이며, 다음과 같이 정의된다.

1. 항목집합 $I = \{i_1, i_2, i_3, \ldots, i_n\}$는 빙산질의에 사용된 그룹속성을 이루는 각 속성의 도메인(domain)의 집합이며 각 속성의 도메인을 항목(item)이라 한다.
2. $2^I$가 항목집합 $I$의 멱집합을 나타낼 때, $e \in (2^I - \{\varnothing\})$을 만족하는 $e$를 항목집합(itemset)이라 한다. $|e|$는 항목집합 $e$를 구성하는 항목의 수를 의미하며, 항목집합 $e$는 항목의 수에 따라 "$|e|$-항목집합"이라 정의한다. 일반적으로 3-항목집합 $\{a,b,c\}$는 간단히 $abc$로 나타낸다.
3. 트랜잭션은 $I$로 이루어져 있으며 각 트랜잭션은 식별자 $TID$를 갖는다. $k$번째 순서로 데이터 집합에 추가되는 트랜잭션을 $T_k$라 나타내며 $TID$는 $k$이다.
4. 새로운 트랜잭션 $T_k$가 추가되었을 때 현재의 데이터 집합은 현재까지 발생하여 추가된 모든 트랜잭션들 즉, $D_k = <T_1, T_2, T_3, \ldots, T_k>$로 구성된다. 따라서 $|D|_k$는 현재 데이터 집합에 포함된 트랜잭션의 총 수를 의미한다.

빈발항목탐색 방법에서 $T_k$를 현재 트랜잭션이라 할 때, 임의의 항목집합 $e$에 대한 현재 출현빈도수를 $C_k(e)$라 정의 하며 이는 현재까지의 $k$ 트랜잭션에서 $e$가 포함된 트랜잭션의 수를 나타낸다. 이와 마찬가지로, 항목집합 $e$의 현재 지지도 $S_k(e)$는 현재까지의 트랜잭션의 총 수 $|D|_k$대비 항목집합 $e$의 출현 빈도수 $C_k(e)$의 비율로 정의한다. 항목집합 $e$의 현재 지지도 $S_k(e)$가 사전 정의된 최소 지지도($S_{min}$) 이상일 때, 항목집합 $e$를 현재 데이터 스트림 $D_k$에서의 빈발항목집합이라 정의한다.

*estDec* 방법은 데이터 스트림을 구성하는 트랜잭션이 생성과 동시에 처리되며, 빈발항목집합 생성을 위한 후보집합 생성 없이 전위트리 구조[10]를 갖는 모니터링 트리를 이용하여 트랜잭션에 나타난 항목집합들의 출현 빈도수를 관리한다.

*estDec* 방법은 지연 추가와 전지 작업을 통해서 빈발항목집합이 될 가능성이 있는 항목집합들만을 관리한다. *estDec* 방법에서 새로운 항목집합과 해당 항목집합의 출현 빈도수가 관리되는 경우는 다음의 두 가지 경우이다. 첫째, 트랜잭션 $T_k$에서 최초로 발생된 길이 1인 항목집합의 경우, 해당 항목집합과 그 출현빈도수는 빈도수 추정 과정을 거치지 않고 전위트리 $P_k$에 추가하여 관리한다. 둘째, $T_k$에 전위트리에서 관리되지 않는 길이 $n(n \geq 2)$인 새로운 항목집합이 발생되었을 때에는 해당 항목집합이 가까운 미래에 빈발항목집합이 될 수 있을 정도로 큰 지지도를 가질 때 전위트리 $P_k$에 추가된다. 즉, $P_k$에서 관리되고 있지 않은 새로운 $n$-항목집합 $e$는 자신의 모든 $(n-1)$-부분항목집합들이 $P_k$에서 관리될 때, $e$의 지지도를 이들 $(n-1)$-부분항목집합의 출현 빈도수로부터 추정하며, 추정값이 사전 정의된 지연추가 임계값 $S_{ins}$ 이상일 때 $P_k$에 추가된다. 이 과정을 지연추가라 한다. 한편, 이미 $P_k$에서 관리되고 있는 항목집합들에 대해 현재 항목집합의 지지도가 전지 임계값

$S_{prn}$ 미만으로 감소할 때 해당 항목집합을 앞으로 빈발 항목집합이 될 가능성이 상대적으로 낮은 비중요 항목집합으로 간주하여 빈발항목집합의 안티모노톤(*antimonotone*)성질에 의해 해당 항목집합을 표현하는 노드와 그 노드의 모든 자손 노드들을 $P_k$로부터 제거한다. 이 과정을 항목집합의 전지 과정이라 한다.

전위트리의 크기는 테이터 스트림에 나타나는 $S_{sig}$ 이상의 지지도를 갖는 중요 항목집합의 수에 의존하므로 중요항목집합을 관리하는 전위트리의 크기가 한정된 메모리의 크기보다 큰 경우 이후에 발생하는 새로운 중요 항목집합을 모니터링 할 수 없다는 단점을 갖는다. 이러한 단점으로 인해, 전위트리의 크기가 한정된 메모리의 크기보다 큰 경우, *estDec*방법을 이용한 빈발항목집합 탐색 결과의 정확도는 크게 감소하게 된다.

## 4. 큐보이드 전위트리

### 4.1 큐보이드 전위트리

전위트리가 트랜잭션내의 모든 항목집합에 해당하는 노드를 생성하는데 반해, 큐보이드 전위트리는 빙산질의에 사용되는 그룹속성으로 이루어진 전위 항목집합에 해당하는 노드만을 생성하기 때문에 훨씬 적은 메모리를 사용한다. 그림 1(c)와 (d)는 그림 1(a)의 데이터 집합을 이용하여 항목집합의 빈발횟수가 2이상인 경우에 트리에 추가 된다고 할 때, 생성되는 전위트리와 그림 1(b)에 주어진 빙산질의를 처리하기 위해 생성한 큐보이드 전위트리이다. 그림 1(c)와 (d)에서 볼 수 있듯이 전위트리와는 달리 큐보이드 전위트리는 질의의 GROUP BY절에 사용된 속성들의 순서를 가지고 노드를 생성한다. 사용된 속성이 다르거나 속성간 순서가 변경된 결과를 원할 경우 새로운 큐보이드 전위트리를 생성해야한다. 이러한 제약사항은 기존의 SQL과도 같아서, 기존 SQL의 GROUP BY에서 속성 개수나 순서가 변경된 결과를 얻고 싶은 경우 질의를 새로 만들어 실행한다.

큐보이드 전위트리의 노드는 빙산질의의 GROUP BY절에 사용된 그룹속성의 순서와 같은 순서로 생성된다. 예를 들어, 그림 1(b)에서 주어진 빙산질의와 같이 GROUP BY 절에 사용된 그룹속성의 순서가 $dim_1$, $dim_2$, $dim_3$인 경우, 생성되는 노드들은 그림 1(d)와 같이 GROUP BY절에 사용된 속성들의 순서와 동일한 순서로 노드간의 상하관계를 가지며 큐보이드 전위트리를 생성한다. 이러한 상하관계를 가지고 구성된 루트노드로부터 단말노드까지의 경로인 <1>→<100>→<1000>을 **전위 경로**라고 한다. 큐보이드 전위트리는 루트노드에서 단말노드까지의 모든 경로가 **전위 경로**로 이루어져 있다.

**정의 1. 전위 항목집합(*prefix-itemset*)**

큐보이드 전위트리의 ***prefix-path*** 상에 존재하는 노드들이 나타내는 모든 항목집합들을 **전위 항목집합**이라 한다.

예를 들어, 그림 1(d)에서 루트노드부터 단말노드 <1000>까지의 경로 상에 존재하는 노드들은 항목집합 <1>, <1,100>, <1, 100, 1000>을 나타낸다. 이 항목집합들을 전위 항목집합이라 한다.

**정의 2. 큐보이드 전위트리(*Cuboid prefix tree*)**

큐보이드 전위트리는 GROUP BY절에 사용된 그룹속성으로 이루어진 전위 항목집합만을 가지고 생성되는 전위트리 구조이다. 큐보이드 전위트리 다음과 같은 특징을 갖는다.

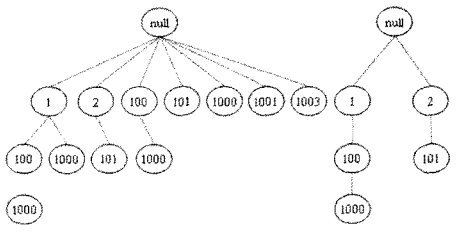
1. 큐보이드 전위트리는 "*null*"값을 가지는 하나의 루트노드 $r$을 가지며 $r$을 제외한 다른 노드들은 하나의 항목 $i \in I$를 갖는다.
2. 큐보이드 전위트리의 노드는 그룹속성의 순서에 따른 노드간의 상하관계를 갖는다.
3. 루트노드 r로부터 단말 노드(leaf node) $v_n$까지의 *prefix-path*상에 존재하는 노드들이 $r \rightarrow v_1 \rightarrow v_2 \rightarrow \cdots \rightarrow v_n$의 순서를 이루고 경로상의 임의의 노드 $v_j$가 항목 $i_j$를 갖는다고 할 때, 노드 $v_n$는 그룹속성으로 이루어진 항목집합 $e_n = i_1 i_2 \cdots i_n$를 표현하며 $e_n$의 현재 출현 빈도수 $C_k(e_n)$을 관리한다.
4. 큐보이드 전위트리의 단말노드는 빙산질의의 모든 그룹속성에 의해 집계된 결과 값을 관리한다.
5. 빙산질의의 GROUP BY절에 사용된 그룹속성이 $n$

| $dim_1$ | $dim_2$ | $dim_3$ |
|---|---|---|
| 1 | 100 | 1000 |
| 1 | 100 | 1000 |
| 2 | 101 | 1001 |
| 1 | 100 | 1000 |
| 2 | 101 | 1003 |

(a) 테이블 R

```
SELECT   dim1, dim2, dim3, COUNT(*)
FROM     R
GROUP BY dim1, dim2, dim3
HAVING   COUNT(*) ≥ 2
```

(b) 빙산질의 예




(c) 전위트리 (d) 큐보이드 전위트리

그림 1 전위트리와 큐보이드 전위트리의 비교

개이고, 각 속성의 항목 수가 각각 $k_i$개($k_1,k_2,\cdots,k_n$)일 때, 큐보이드 전위트리가 가질 수 있는 최대 노드 수는 식 (1)과 같다.

$$MAX_{node} = \sum_{i=1}^{n}\left(\prod_{j=1}^{i} k_j\right) \quad (1)$$

### 4.2 큐보이드 전위트리 생성 방법

그림 2는 큐보이드 전위트리를 생성하는 알고리즘이다. 큐보이드 전위트리 알고리즘은 노드의 빈발도와 단말노드에 집계함수 결과 값을 갱신하는 단계(line 4-9)와 새로운 항목이 발생하였을 때, 트리에 새로운 노드를 생성하는 지연 추가 단계(line 10-20)로 나누어진다. 항목집합의 빈발횟수와 집계함수 결과 값 갱신 단계에서는 새로운 트랜잭션이 발생하였을 때 트랜잭션내의 각 전위항목에 해당하는 노드가 큐보이드 전위트리 내에 존재하면 각 노드의 빈발횟수를 1증가 시킨다. 이때 갱신된 항목의 지지도($cnt_k/|D|_k$)가 사용자가 미리 정의한 전지 임계값($S_{sig}$)보다 작은 경우에는 해당 노드와 모든 자식노드를 큐보이드 전위트리에서 삭제한다(line 6-7). 이 작업을 전지 작업이라고 한다. 또한 갱신하려는 노드가 단말노드인 경우 해당 노드의 빈발도 갱신과 함께 집계함수 결과도 같이 갱신한다(line 8).

이 후에 가까운 미래에 빈발한 항목집합이 될 가능성이 높은 새로운 항목집합을 찾기 위해 지연 추가 작업이 이루어진다. 큐보이드 전위트리의 새로운 1 레벨 노드를 생성하기 위해서는 해당 1-항목의 지지도를 1-항목 관리 테이블에서 관리하다가 지지도가 지연 추가 임계값($S_{sig}$)이상이 될 경우에 큐보이드 전위트리에 추가된다. $n$ 레벨 노드부터는 ($n$-1)레벨 노드의 지지도가 지연 추가 임계값 이상이고 해당 1-항목의 지지도 또한 지연 추가 임계값 이상인 경우 트리에 추가된다.

### 4.3 큐보이드 전위트리를 이용한 빙산질의 처리

새로운 트랜잭션이 발생할 때마다 트랜잭션 내의 항목집합 중 빈발횟수가 지연 추가 임계값 이상인 항목집합은 큐보이드 전위트리에 추가된다. 그림 3(a)의 구조를 갖는 데이터 스트림에서 그림 3(b)의 빙산질의를 수

```
     Input: A data stream D
     Output:: CPT
     CPT: A Cuboid Prefix tree
1:   CPT = ∅;
2:   For each transaction in D{
3:     read current transaction Ti;
       //Count and Aggregation value(sum, count, etc) updating phase
4:     For all prefix-itemset e st e ∈ (2^R-{all non prefix-itemset}) and e ∈ CPT{
5:       cnt = cnt+-;
6:       If(cnt/|D|k<Ssig)
7:         Eliminate e and it's child node from CPT;
8:       Computing aggregation value M;
9:     } // of For in line 4
       //Delayed-insertion phase
10:    tree_level=0;
11:    For all prefix-itemset e s.t.e∈(2^R-{all non prefix-itemset}) and e ∉ CPT{
12:      if (|e|=1 and tree_level =0){
13:          Insert e into CPF; cnt = 1;
14:      }else{
15:        if(|e|>1 and C(|e|-1) ≥ Ssig)
16:          Insert e into CPT; cnt=1;
17:        } // of If in line 14
18:      } // of If in line 11
19:      tree_level++;
20:    } // of For in line 10
21:  } // of For in line 2
```

그림 2 큐보이드 전위트리 생성 알고리즘

행할 경우, 빈발도가 지연추가 임계값 이상인 항목집합을 추가함으로써 그림 3(c)와 같은 큐보이드 전위트리를 생성할 수 있다. 그림 3(b)에서 주어진 빙산질의는 그룹 속성 <$dim_1$, $dim_2$, $dim_3$>으로 그룹화된 후 집계함수($COUNT$)를 적용한 빈발도가 임계값 $T$이상인 항목집합을 결과로 반환한다. 큐보이드 전위트리에서 질의에 사용된 그룹속성인 <$dim_1$, $dim_2$, $dim_3$>으로 그룹화된 항목집합의 정보는 단말 노드에 저장되어 있다. 따라서, 큐보이드 전위트리를 이용하여 그림 3(b)의 결과를 제공하기 위해서는 그림 3(c)의 큐보이드 전위트리에서 3-항목집합들 중 빈발도가 주어진 임계값 $T$이상인 3-항목집합들을 탐색한다.

### 4.4 공유노드를 통한 다중 빙산질의 처리

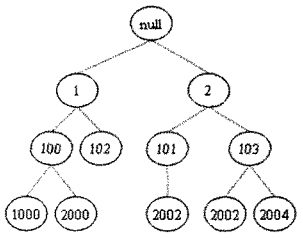
큐보이드 전위트리를 이용하여 빙산질의를 수행할 때, GROUP BY절에 사용된 그룹속성에 따라 여러 빙산질의가 하나의 큐보이드 전위트리를 이용하여 처리될 수 있다. 그림 4(b)는 4개의 속성을 가지고 있는 데이터 집

| | $dim_1$ | $dim_2$ | $dim_3$ |
|---|---|---|---|
| tid 1 | 1 | 100 | 1000 |
| tid 2 | 2 | 101 | 1002 |
| tid 3 | 1 | 100 | 1000 |
| tid 4 | 1 | 100 | 1000 |
| tid 5 | 2 | 101 | 1003 |
| ... | ... | ... | ... |

(a) 데이터 스트림 R

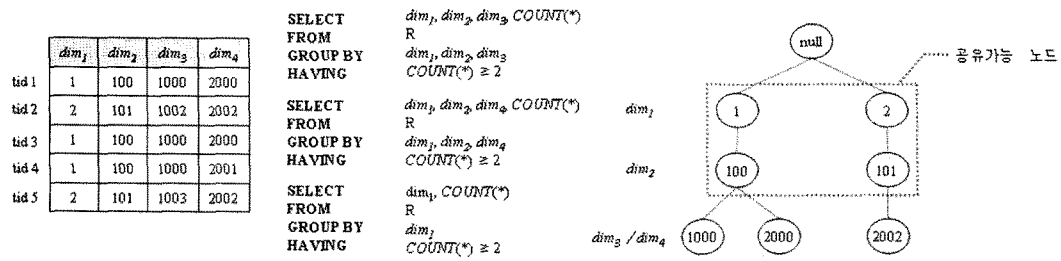
**SELECT** $dim_1, dim_2, dim_3, COUNT(*)$
**FROM** R
**GROUP BY** $dim_1, dim_2, dim_3$
**HAVING** $COUNT(*) \geq T$

(b) 빙산 질의




(c) 큐보이드 전위트리

그림 3 큐보이드 전위트리를 이용한 빙산질의 처리




(a) 데이터 집합 $R$ (b) 빙산 질의 (c) 다중 빙산질의 처리를 위한 큐보이드 전위트리

그림 4 노드 공유를 통한 다중 빙산질의 처리

합 R을 가지고 수행될 서로 다른 그룹속성을 갖는 3개의 빙산질의다. 그림 4(b)의 질의들을 살펴보면 3개의 빙산질의에서 GROUP BY절에서 공통적으로 사용하는 그룹속성은 $<dim_1>$다. 첫 번째 질의와 두 번째 질의에서 공통적으로 사용되는 그룹속성은 $<dim_1, dim_2>$이다. 따라서 큐보이드 전위트리에서 항목집합 $<dim_1>$, $<dim_1, dim_2>$의 정보를 담고 있는 노드들을 트리의 상위레벨에 위치시킴으로써 노드를 공유할 수 있다. 그림 4(c)는 (a)의 데이터 집합에 대해 (b)의 빙산질의들을 수행하기 위해 생성한 큐보이드 전위트리이다.

그림 4(b)의 그룹속성 $<dim_1, dim_2, dim_3>$을 이용해 처리되는 첫 번째 질의에 대한 응답은 (c)의 큐보이드 전위트리에서 전위 경로 1→100→1000의 단말 노드 <1000>에 저장된다. 이와 마찬가지로 그룹속성 $<dim_1, dim_2, dim_4>$을 이용해 처리되는 두 번째 질의에 대한 결과는 전위 경로 1→100→2000이나 2→101→2002에 존재하는 단말 노드 <2000>과 <2001>에 저장된다. 끝으로 그룹속성 $<dim_1>$을 이용해 수행되는 세 번째 빙산질의의 결과는 1레벨 노드인 <1>, <2>에 저장된다. 이와 같이 다중 빙산질의 연산에서 GROUP BY절에 사용된 그룹속성 중 공통적으로 사용된 속성에 해당하는 큐보이드 전위트리를 공유함으로써 빙산질의를 처리하기 위해 필요한 트리 생성시간과 메모리 사용량을 감소시킬 수 있다.

## 5. 성능 평가

Zip 분포에 의해 생성한 여러 데이터 집합을 가지고 제안한 큐보이드 전위트리를 이용한 빙산질의 방법의 효율성을 검증하였다. 트랜잭션은 순서대로 하나씩 처리되었으며 질의처리를 위한 큐보이드 전위트리는 빈발도가 사용자가 정의한 지연 추가 임계값($S_{sig}$)인 항목집합들로 생성되었다. 각 실험 그래프의 라벨은 표 1과 같은 의미를 가진다. 예를 들어, D1000K.A7.I500.Z2는 1,000,000개의 트랜잭션으로 구성되었고 속성이 7개이며 각 속성의 도메인이 500개이고 zip분포가 2인 데이터셋을 가리킨다. 모든 실험은 1GB의 메모리를 가진 2.8GHz 팬티엄 컴퓨터와 리눅스 환경에서 실험되었으며 모든 프로그램은 C언어로 구현되었다.

표 1 라벨의 의미

| 기호 | 의미 |
|---|---|
| D | 트랜잭션의 수 (단위, K = 천개) |
| A | 속성의 갯수 |
| I | 속성의 도메인 개수 |
| Z | zip 분포 값 |

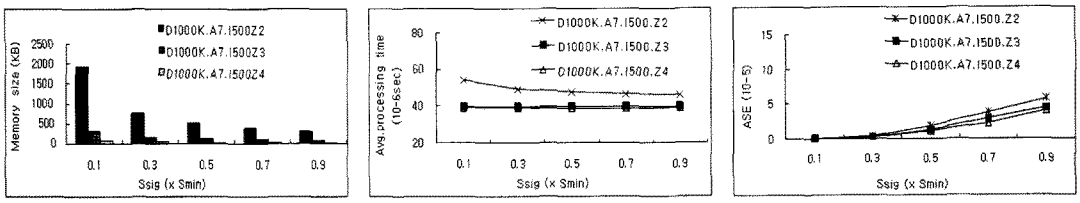
제안된 큐보이드 전위트리의 정확도를 표현하기 위해 [16]와 같은 방법으로 두 집합 $R_1 = \{(e_i, S'_k(e_i) \mid S'_k(e_i) \geq S_{min}, |e|=n)\}$과 $R_2 = \{(e_j, S''_k(e_j) \mid S''_k(e_j) \geq S_{min}, |e|=n)\}$에 대해 *average support error* $\boldsymbol{ASE(R_2|R_1)}$를 식 (2)과 같이 정의한다.

$$ASE(R2 \mid R1) = \frac{\sum_{e_i \in R_1 - R_1 \cap R_2} S'_k(e_i) + \sum_{e_i \in R_1 - R_1 \cap R_2} \{|S'_k(e_i) - S''_k(e_i)|\} + \sum_{e_i \in R_2 - R1 \cap R2} S''_k}{|R_1|} \quad (2)$$

$|R_1|$은 결과 집합 $R_1$의 항목집합의 개수를 나타낸다. $ASE(R_1|R_2)$이 작을수록 $R_2$의 결과 집합이 $R_1$의 결과 집합과 유사하다. 실험에서 빙산질의 결과의 정확도를 표현하기 위해 $ASE(R_{cp\text{-}tree}|R_{DB})$를 사용하였다. $R_{cp\text{-}tree}$와 $R_{DB}$는 각각 큐보이드 전위트리를 이용한 빙산질의 결과와 상용 DBMS인 MS-SQL SERVER를 이용하여 산출한 빙산질의의 결과를 나타낸다.

**실험 1. 큐보이드 전위트리의 성능 분석**

이 실험에서는 임계값($T$)인 최소 지지도($S_{min}$) 값을 0.0001로 설정한 뒤 $S_{sig}$값을 0.1에서 0.9변경해 가면서 실험을 진행하였다. 그림 5(a)는 메모리 사용량의 변화를 보여주며 그림 5(b)는 트랜잭션 1000건당 평균 처리시간을 보여준다. 그림 5(c)는 빙산질의를 결과집합의 정확도를 나타낸다. 동일한 $S_{min}$값에 대해 $S_{sig}$값이 증가할수록 큐보이드 전위트리에서 관리되는 항목집합의 수가 감소하게 된다. 따라서 큐보이드 전위트리의 크기가 감소하게 되므로 사용되는 메모리 사용양은 감소하고 새로운 트랜잭션이 발생했을 때, 트리를 탐색하여 트리의 노드 정보를 갱신하는데 필요한 시간도 감소하게 된




(a) 메모리 사용량 (b) 평균 처리시간 (c) 결과집합의 정확도

그림 5 Ssig 변화에 따른 큐보이드 전위트리 성능 분석

다. 또한 관리하는 항목집합의 수가 감소함에 따라 정확도는 감소한다. 이는 전체 트랜잭션을 기준으로 하였을 경우에는 항목집합의 빈발횟수가 주어진 임계값인 $S_{min}$을 만족하지만 항목집합이 빈발하게 발생하지 않을 경우에는 큐보이드 전위트리에 추가되지 않거나 추가되었다가 삭제되기 때문이다.

**실험 2. 데이터 집합의 밀도에 다른 성능 분석**

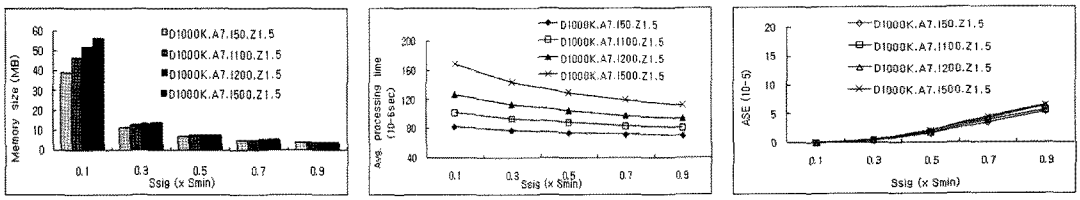
사용된 $S_{min}$값은 0.00001이며, 데이터 집합의 속성이 갖는 항목 수를 50, 100, 200, 500으로 변경해 가며 큐보이드 전위트리의 성능 변화를 실험하였다. 질의에 사용된 속성의 수가 동일한 경우 속성내 도메인의 수가 증가할수록 나타날 수 있는 항목집합의 수가 증가하기 때문에 그림 6(a)에서 나타나듯이 속성 도메인 수가 증가할수록 발생할 수 있는 항목집합의 수가 많아지므로 큐보이드 전위트리 내에서 관리하는 항목의 수가 증가하여 메모리 사용량이 증가하는 것을 볼 수 있다. 또한 그림 6(b)와 같이 속성 도메인 수가 증가할수록 트리의 사이즈가 커지게 되므로 트리를 갱신하는데 소요되는 시간인 평균 처리시간도 증가한다.

결과집합의 정확도는 그림 6(c)와 같이 속성 도메인의 수가 증가하면 발생할 수 있는 항목집합의 수가 증가하므로 데이터 집합내의 항목집합들의 빈발도가 낮아지게 되고 따라서 $S_{min}$ 근처의 지지도를 갖는 항목집합들이 결과집합에 포함되지 않는 경우가 발생하여 결과집합의 정확도가 약간 감소한다.

**실험 3. 속성 수에 따른 성능 분석**

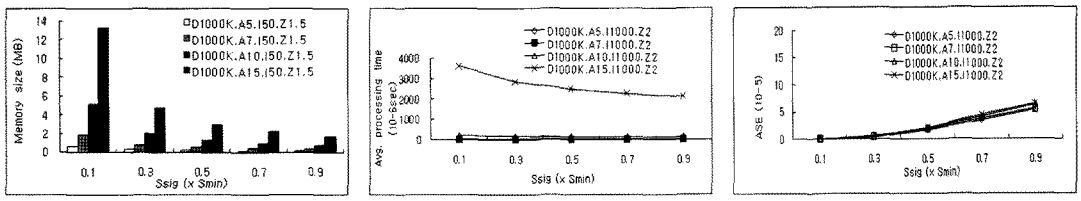
그림 7은 빙산질의에 사용된 속성 수에 따른 큐보이드 전위트리의 성능을 실험한 그래프이다. $S_{min}$값은 0.0001로 설정하였다. 그림 7(a)에서 보듯이 속성의 수가 많을수록 큐보이드 전위트리 내에 생성되는 중간노드의 수가 증가하게 되므로 큐보이드 전위트리에서 사용되는 메모리 사용량이 증가한다. 이때 메모리 사용량을 줄이기 위해서는 $S_{sig}$값 조절을 통해 약간의 오차를 허용하면서 메모리 사용량을 감소시킬 수 있다.

그림 7(b)와 (c)는 평균 처리시간과 결과 집합의 정확도를 나타낸다. 평균 처리시간은 이전 실험들과 마찬가




(a) 메모리 사용량 (b) 평균 처리시간 (c) 결과 집합의 정확도

그림 6 속성 도메인 밀도에 따른 큐보이드 전위트리의 성능 분석




(a) 메모리 사용량 (b) 평균 처리시간 (c) 결과집합의 정확도

그림 7 속성 수에 따른 큐보이드 전위트리의 성능 분석

지로 $S_{sig}$ 값이 증가함에 따라서 큐보이드 전위트리에서 관리하는 항목집합의 수가 감소하게 되므로 큐보이드 전위트리의 크기가 감소해 트리 갱신을 위해 소비되는 시간이 감소하게 된다. 또한 앞서 설명했듯이 데이터 집합의 속성 수가 증가하게 되면 큐보이드 전위트리의 크기가 커지게 되므로 $S_{sig}$ 값을 증가시켜 그림 7(c)와 같이 결과집합에 약간의 오차를 허용하면서 사용되는 메모리 사이즈를 조절할 수 있다.

## 6. 결 론

본 논문에서는 데이터 스트림 환경에서 빙산질의를 수행하기 위해 큐보이드 전위트리를 제안하였다. 큐보이드 전위트리는 질의의 그룹항목에 의해 데이터 스트림의 항목집합을 그룹화한 뒤 미리 정의된 임계값 이상인 항목집합만을 제한된 메모리를 사용하여 관리한다. 또한 지연 추가 및 전지 임계값을 사용하여 큐보이드 전위트리에 의해 사용되는 메모리 사용량을 조절할 수 있다. 또한 다중 빙산질의에서 공통된 속성을 사용하는 경우, 큐보이드 전위트리의 중간노드를 공유함으로써 적은 메모리를 이용하여 다중 빙산질의를 처리할 수 있는 방법을 제안하였다. 하나의 빙산질의를 처리하기 위해 하나의 큐보이드 전위트리를 생성해야 하는데 반해 다중 빙산질의들 간에 노드를 공유하여 질의를 처리하게 되면 큐보이드 전위트리를 생성하는데 필요한 처리시간과 메모리 사용량을 효과적으로 줄일 수 있다.

## 참 고 문 헌


[1] M. Garofalakis, J. Gehrke and R. Rastogi., "Querying and mining data streams: you only get one look," In the tutorial notes of the 28th International Conference on Very Large Databases, TUTORIAL SESSION: Tutorial 1, pp. 635-635, 2002.

[2] S. Krishnamurthy, C. Wu, and M. Franklin., "On-the-Fly Sharing for Streamed Aggregation," In Proceedings of the 2006 ACM SIGMOD International Conference on Management of Data, pp. 623-634, 2006.

[3] J, Chen, D.J. DeWitt, F. Tian, and Y. Wang., "NiagaraCQ: A scalable continuous query system for internet databases," Proceedings of the 2000 ACM SIGMOD International Conference on Knowledge Discovery and Data Mining, pp. 379-390, 2000.

[4] S. Madden, M.A. Shah, J.M. Hellerstein, and V. Raman., "Continuously adaptive continuous queries over streams," In Proceedings of the 2002 ACM SIGMOD International Conference on Management of Data, pp. 49-60, 2002.

[5] S. Chandrasekaran, M.J. Franklin., "Streaming queries over streaming data," In Proceedings of 28th International Conference on Very Large Data Bases pp. 203-214, 2002.

[6] A. Arasu, J. Widom., "Resource Sharing in Continuous Sliding-Window Aggregates" In Proceedings of 30th International Conference on Very Large Data Bases, pp. 336-347, 2004.

[7] Rui Zhang, Nick Koudas, Beng Chin Ooi, Divesh Srivastava, "Multiple Aggregations Over Data Streams," In Proceedings of the 2005 ACM SIGMOD International Conference on Management of Data, pp. 299-310, 2005.

[8] M. Fang, N. Shivakumar, H. Garcia-Molina, R. Motwani, J.D. Ullman., "Computing Iceberg Queries Efficiently" In Proceedings of 24rd International Conference on Very Large Data Bases, pp. 299-310, 1998.

[9] R. Agrawal, R. Srikant., "Fast Algorithms for Mining Association Rules in Large Databases," In Proceedings of 20th International Conference on Very Large Data Bases, pp. 487-499, 2004.

[10] S. Brin, R. Motwani, J.D. Ullman, and S. Tsur., "Dynamic itemset counting and implication rules for market basket data," In Proceedings of the ACM SIGMOD International Conference on Management of Data, pp. 255-264, 1997.

[11] A. Savasere, E. Omiecinski, and S. Navathe., "An Efficient Algorithm for Mining Association Rules in Large Databases," In Proceedings of 20th International Conference on Very Large Data Bases, pp. 432-444, 1995.

[12] C. Hidber., "Online association rule mining," In Proceedings of the ACM SIGMOD International Conference on Management of Data, pp. 145-156, 1999.

[13] D. Cheug, J.Han, V. Ng, and C.Y. Wong., "Maintenance of Discovered Association Rules in Large Databases: An Incremental Updating Technique for Maintaining Discovered Association Rules," In Proceedings of the 12th International Conference on Data Engineering, pp. 106-114, 1996.

[14] V. Ganti, J. Gehrke, and R. Ramakrishnan., "DAEMON: Mining and Monitoring Evolving Data," In Proceedings of the 16th International Conference on Data Engineering, pp. 439-448, 2000.

[15] G.S. Manku and R. Motwani., "Approximate Frequency Counts over Data Streams," In Proceedings of the 28th International Conference on Very Large Data Bases, pp. 346-357, 2002.

[16] J.H. Jang and W.S. Lee., "Finding recent frequent itemsets adaptively over online data streams," In Proceedings of the 2003 ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 487-492, 2003.

한 상 길

2005년 2월 단국대학교 전자컴퓨터공학부 학사. 2007년 2월 연세대학교 컴퓨터과학과 석사. 2007년 1월~현재 (주)LG전자. 관심분야는 Data warehouse, OLAP, Data Stream Mining

양 우 석

2007년 8월 연세대학교 컴퓨터과학과 학사. 2007년 9월~현재 연세대학교 컴퓨터과학과 석사 과정. 관심분야는 Data warehouse, OLAP, Data Mining

이 원 석

1985년 7월 Boston University 컴퓨터과학 공학학사. 1987년 7월 Purdue University 컴퓨터과학 공학석사. 1990년 7월 Purdue University 컴퓨터과학 공학박사. 2004년 3월~현재 연세대학교 컴퓨터과학과 정교수. 관심분야는 Database, Data Mining