이화여자대학교 과학기술대학원
2001학년도
석사학위 청구 논문

웹 로그 데이터의 OLAP연산 시 희박성 형태 분석 및 성능 평가

컴 퓨 터 학 과

김 지 현

2002

웹 로그 데이터의 OLAP연산 시 희박성 형태 분석 및 성능 평가

이 論文을 碩士學位 論文으로 提出 함

2002 年 1月

梨花女子大學校 科學技術大學院

컴퓨터學科 金 智 賢

김 지 현의 碩 士 學 位 論 文을 認准 함

指導敎授 용 환 승

審査委員 김 명 _____

박 승 수 _____

용 환 승 _____

梨 花 女 子 大 學 校 科 學 技 術 大 學 院

목 차

그 림 목 차

표 목 차

論文概要

많은 기업이나 회사들은 하루에도 수십 수백 메가 바이트의 웹 로그 데이터를 수집하고 이러한 데이터들을 효율적으로 분석하고자 한다. 하지만 이러한 엄청난 데이터의 용량은 웹 로그 데이터를 분석하는데 장애요인이 된다. 따라서 웹 로그 데이터를 다차원적으로 실시간에 사용자의 질의에 응답하기 위해서는 OLAP의 적용은 꼭 필요한 일이라 할 수 있다. OLAP적용에 있어서 사용자 질의에 대해 빠른 응답 성능을 보장하기 위해서는 사전 연산결과를 저장해서 그 결과에 질의를 하는 것이 필요하다. 이러한 사전 연산 과정에서 OLAP 데이터의 희박성은 데이터 폭발(Data Explosion)현상을 발생시키고, 이는 또한 시스템 성능을 저하시키는 요인이 된다.

본 논문에서는 실제 웹 로그 데이터를 사용하여 웹 로그 데이터에서 발생되는 희박성의 원인에 대해 분석하고, 2,3차원 데이터 모델 내에서의 희박성 형태를 조사하였다. 조사된 희박성 형태를 바탕으로 시중에 나와 있는 OLAP시스템 중 MS SQL 2000 Analysis Service, Oracle Express Server와 희박성 처리를 위해 청크 방식을 사용하여 구현한 MOLAP시스템 세가지 시스템에 대해 성능 평가를 해 보았다. 성능 평가를 하기 위한 기초 단계로 각 희박성 형태에 따라 테스트 데이터를 생성하기 위해 2,3 차원의 데이터 모델과 질의 모델을 설계하고, 테스트 데이터 생성기를 구현하였다. 그리고 각 OLAP시스템을 위한 성능 평가 인터페이스를 구현하였다.

본 논문에서 수행된 OLAP 제품들의 성능 평가를 분석하고 이를 확장 시킨다면 웹 로그 데이터내의 희박성을 효율적으로 처리할 수 있는 저장구조와 인덱스 방식을 발견하는데 토대가 될 수 있을 것이다.

I. 서 론

1.1 연구 배경 및 목적

최근의 경쟁적 비즈니스 환경에서 CRM(Customer Relationship Management)을 위한 IT(Information Technology)기술의 발전이 급속도로 진전되고 있다.

CRM의 목적은 각 기업이나 회사들이 새로운 고객을 유치하고, 기존의 고객들을 유지하면서 장기간의 이익 관계를 이어나가기 위해 고객의 취향과 필요를 이해하는 것이다[1]. CRM을 위해 최근 많이 적용, 개발되고 있는 분야는 간단하게는 통계적 수단을 통해 웹 사이트를 분석하는 방법과 좀더 정교하게는 다차원 분석이 가능한 OLAP(On-Line Analysis Processing) 기술을 적용한 방법 그리고 Data Mining 알고리즘(신경 회로망, 의사결정 트리, 연관 규칙)을 이용한 방법들이 각광 받고 있다. 이에 적용되는 데이터 들로는 고객 등록, 구매, 공급업체, 경쟁사 등 외부 환경 자료 등이 사용되고 있다. 하지만 이러한 자료 중 CRM의 목적을 위해 가장 유용한 것은 고객 데이터라 할 수 있다. 특히 고객 데이터 중에서 각 기업이나 회사의 웹 서버에 저장되어 있는 웹 로그 데이터가 가장 쉽게 얻을 수 있고, 유용하게 사용될 수 있다.

웹 로그 데이터는 최근 World Wide Web (WWW) 기술의 엄청난 발전과 사용자들의 수요가 많아 짐에 따라 그 양이 매일 수십 수백 메가 바이트까지 증가 하고 있다. 이러한 대규모 데이터를 사용해 실시간에 다차원 분석을 하기 위해서는 OLAP을 사용해야 한다. 하지만 OLAP을 사용하는데 있어 웹 로그 데이터 자체가 가지고 있는 특성에 의해 희박성이 발생되고, 이러한 데이터를 이용하여 사전 집계 연산을 수행

할 시 데이터의 폭발(Data Explosion)현상이 일어난다. 이는 저장공간의 낭비 뿐 아니라 다차원 질의 시 성능 저하를 발생 시킨다.[2,3,4]

따라서 본 논문에서는 웹 로그 데이터의 희박성에 대한 체계적인 접근을 위한 기초 단계로 희박성을 발생시키는 원인과 2,3 차원 내에서의 희박성 형태들에 대해 밝혀 보고, 이러한 분석을 기반으로 성능 평가를 위한 테스트 데이터 모델과 질의 모델을 설계하였다. 그리고 이를 바탕으로 희박성 처리를 위한 청크 방식을 사용한 MOLAP시스템을 구현해 보고, 이 시스템과 MS SQL 2000 Analysis Services, Oracle Express의 성능을 평가해 보았다.

1.2 연구 내용

본 논문에서는 웹 로그 데이터를 이용하여 OLAP적용 시 희박성이 발생되는 원인 및 발생 가능한 희박성 형태에 대해 알아본다. 이를 위해 실제 웹 로그 데이터들을 분석하기 위해 적당한 형식으로 만들기 위한 사전처리 모듈을 구현하고, 사전처리된 데이터를 사용하여 MS SQL 2000 Analysis Services 와 DBMiner 3D Cube Explore를 사용하여 실제 OLAP적용 시 발생되는 희박성 원인과 형태에 대해 분석 및 도식화 한다. 분석된 각각의 웹 로그 데이터의 희박성 형태에 따라 MS SQL 2000 Analysis Services, Oracle Express, 청크 방식을 기반으로 구현한 MOLAP시스템에 대해 성능 평가를 한다. 성능 평가를 하기 위해 테스트 데이터 생성 모듈을 구현하여, 각 OLAP시스템에 대한 성능 평가 모듈을 통해 큐브 생성 모듈에서 사전연산이 완료된 데이터들에 대해 성능을 평가하고, 결과를 분석해 본다. 테스트 데이터 생성 모듈은 실제 웹 로그 데이터를 사용하여 분석한 희박성 형태에 따른 테스트

데이터를 생성하는 모듈이며, 큐브 생성 모듈은 각 OLAP 시스템 내에 생성된 테스트 데이터들을 로딩하여 2,3차원 큐브를 생성하여 집계연산을 수행하는 모듈이다. 성능 평가 대상이 되고 있는 OLAP시스템 중 희박성 처리를 위해 청크 방식을 사용하고, 집계 연산을 수행 하는데 최소의 메모리를 사용해 한번의 디스크 스캔으로 전체 집계연산을 수행할 수 있는 MOLAP시스템을 구현하여[9], 성능 평가 모듈에서는 이 시스템을 포함하여 MS SQL 2000 Analysis Service 와 Oracle Express에 대해 성능 평가를 할 수 있는 인터페이스를 구현한다. 그리고, 성능 평가를 위해 정한 질의 모델에 대해 성능을 비교하고, 마지막 분석 모듈에 있어서는 평가 결과를 분석한다.

본 논문의 구성은 다음과 같다. Ⅱ장은 현재 OLAP시스템들의 희박 데이터와 그 형태에 대해 살펴 보고, 이러한 데이터가 OLAP의 성능에 미치는 영향에 대해 알아본다. 그리고, 현재까지의 희박 처리를 위한 기법들에 대해 기술 한다. Ⅲ장에서는 웹 로그 데이터를 OLAP적용 시 발생되는 희박성 원인에 대해 분석해 보고, 실제 웹 로그 데이터에 대해 간단히 설명한 후 OLAP시스템에 적용되기 전에 사전 처리하는 사전처리 모듈에 대해 기술 한다. 그리고 이를 사용하여 희박성 원인에 대해 확인하고, 2,3 차원 내에서의 희박성 형태에 대해 알아본다. Ⅳ장에서는 성능 평가를 위해 사용된 OLAP시스템 중 청크 방식을 사용한 C-MOLAP시스템의 특징 및 주요 알고리즘에 대해 설명 하고, Ⅴ장에서는 성능 평가를 위한 테스트 데이터 생성기에 대한 설명과 OLAP제품들의 성능을 측정하여 그 결과를 기술한다. 마지막으로 Ⅵ장에서는 본 연구의 결과 및 향후 연구 방향을 제시한다.

Ⅱ. 관련 기술 및 연구동향

본 장에서는 희박성에 대한 정의와 이러한 희박성이 OLAP시스템의 성능에 미치는 영향에 대해 알아 본다. 그리고 이러한 희박성이 OLAP 큐브 내에서 발생 될 수 있는 형태들에 대해 알아본다. 마지막으로 현재 까지 나와 있는 희박 데이터들을 처리 하기 기법들에 대해 살펴 보도록 한다.

2.1 희박성에 대한 정의

OLAP에 있어서 셀이라 함은 데이터가 입력 되고 저장 될 수 있는 공간이다. 큐브상의 모들 셀들은 잠재적으로 데이터를 가질 수 있다. 그러나 현실적으로 모든 셀들이 데이터를 가지는 경우는 극히 드물다. 따라서 희박성이라 함은 다차원 모델에서 데이터가 입력되고 저장될 수 있는 공간인 셀에 해당되는 값이 없어서 비어있는 셀들이 전체 데이터의 대부분을 차지할 경우를 말한다[7]. 예를 들어 다음과 같은 P전자의 경우 잠실 매장에서는 냉장고와 세탁기만을 판매하고 청소기를 판매하지 않을 수 있다. 마찬가지로 8월 15일이 공휴일 이어서 매출이 전혀 일어나지 않을 수 도 있다. 그림 2.1은 이를 시각적으로 보여주고 있다.

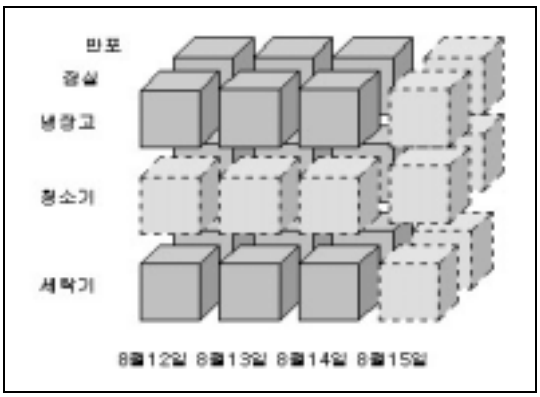

그림 2. 1 희박성의 예

2.2 희박성이 OLAP시스템의 성능에 미치는 영향

OLAP 제품은 데이터 저장 구조에 따라 크게 ROLAP(Relational OLAP), MOLAP(Multidimensional OLAP), HOLAP(Hybrid OLAP) 3종류로 나눌 수 있다. ROLAP의 경우 데이터들을 관계형 데이터베이스에 저장한다. 테이블에 저장하기 때문에 큐브 생성이나 연산속도가 느리고, 데이터를 각 차원의 키 값들과 함께 저장해야하는 단점이 있다. 이에 반해 MOLAP의 경우 데이터를 다차원 배열에 저장하고, HOLAP의 경우 ROLAP과 MOLAP의 두 가지 방법을 혼용하고 있다[13]. MOLAP은 앞에서 언급한 것처럼 다차원 배열 형태로 저장됨으로 각 차원의 축은 데이터가 저장 되기 전에 모든 항목들의 조합에 의해 저장 위치가 결정되어져 있다. 따라서 데이터를 저장하거나 분석 하는 데는 빠른 성능을 보이나, 데이터 값이 포함되어있지 않을 경우 무수한 무효 셀들을 생성 하게 된다. 이렇게 무수한 무효 셀들을 저장하고 있는 큐브에 사전 연산을 수행할 경우 로딩된 데이터 양에 비해 연산된 데이터 양이 심각하게 커져 데이터 폭발 현상이 일어나게 된다. 이러한 데이터 폭발 현상은 계층구조를 구성하는 레벨이 많아지고, 차원의 수가 증가하고 희박성이 커짐에 따라

심각하게 발생한다. 그림 2.2는 희박 차원의 수가 증가 함에 따라 데이터 폭발 현상이 심각해 짐을 나타낸다.

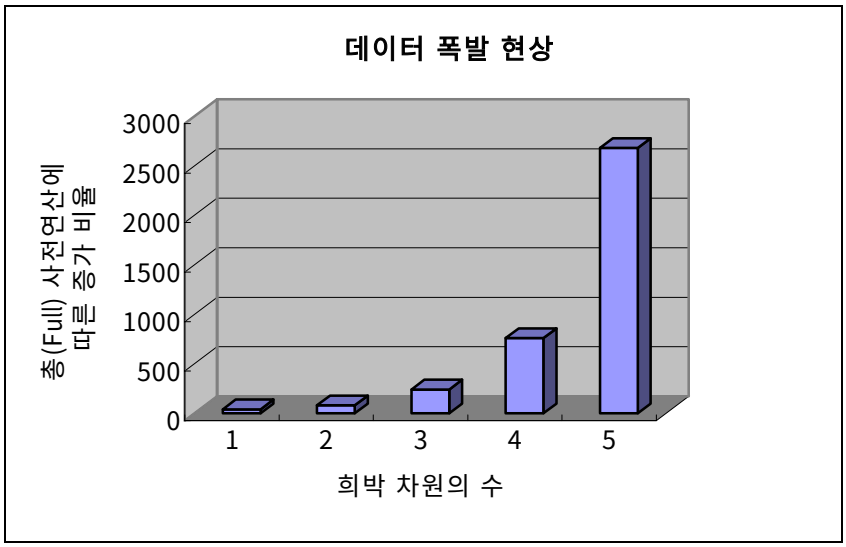

그림 2. 2 희박차원 수의 증가에 따른 데이터 폭발현상

2.3 OLAP 데이터의 희박성 형태

본 절에서는 2,3차원 데이터 모델에서의 데이터 희박성이 어떻게 나타나는지 살펴보고 그 형태를 분석해 본다.

2.3.1 2차원 모델의 희박성 형태

제품과 매장으로 이루어지고, 측정값은 판매 데이터를 갖은 그림 2.3과 같은 2차원 데이터 모델을 살펴 보자. 이러한 2차원 데이터 모델에 있어서는 그림 2.4와 같이 무작위, 줄, 클러스터 형태의 희박성 형태가 가능하다[14].영등포 매장에서는 청소기와 비디오가, 잠실에서는 세탁기가 반포에서는 TV가 판매되었다고 가정하자.

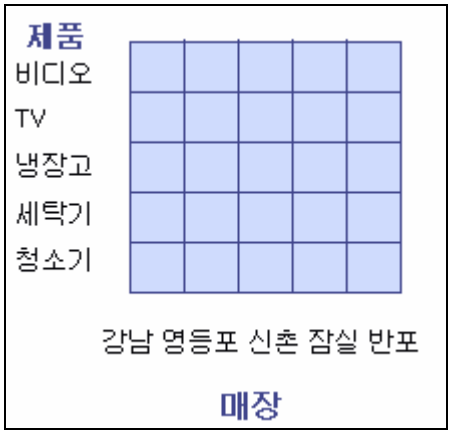

그림 2. 3 제품과 매장차원으로 이루어진 2차원 데이터 모델

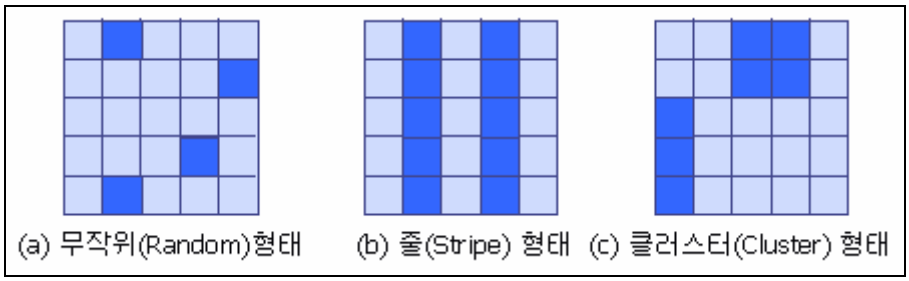

그림 2. 4 2차원모델의 희박성 형태

이러한 경우 희박성 형태는 어떤 특정 형태를 띄지 않는 무작위 형태(Random)를 이루게 된다. 반면에 강남, 신촌, 반포 매장이 어느 특정 일자에 휴무를 하였거나 처음부터 그 매장들에 가전 제품이 취급되지 않았을 경우 줄무늬 형태가 생성된다. 마지막으로 클러스터 형태의 경우는 잠실과 반포매장에서 TV와 비디오가 강남 매장에서는 냉장고, 세탁기와 총소기가 판매 되었고, 영등포와 신촌 매장에서는 당일 매출이 없을 경우 발생된다.

2.3.2 3차원 모델의 희박성 형태

2.3.1 절에서의 2차원 모델에 기간 차원을 추가한 매장, 제품, 기간으로 이루어진

3차원 데이터 모델을 아래 그림 2.5로 가정 한다.

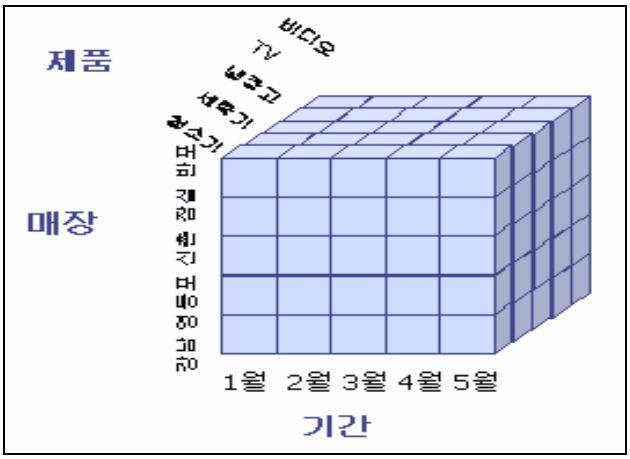

그림 2. 5 제품, 매장, 기간으로 이루어진 3차원 데이터 모델

2차원 희박성 형태를 기반으로 3차원 모델에서의 희박성 형태는 무작위, 줄 클러스터, 슬라이스 형태가 가능하다[14].

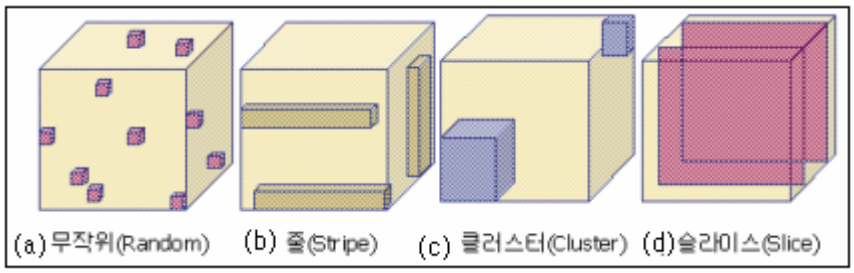

그림 2. 6 3차원 데이터 모델의 희박성 형태

3차원 모델에서 추가된 슬라이스 형태는 전 기간에 걸쳐 전 매장에서 청소기와 TV가 꾸준히 판매된 경우 발생한다.

위와 같이 2,3 차원 데이터 모델에서 나타나는 희박성 형태는 서로 완전히 독립적인 것이 아니고 차원 항목들의 위치 조정을 통해 물리적으로 저장 될 때 서로 다른 희박성 형태로의 전환이 가능하다.

2.4 희박 데이터 처리 기법

본 절에서는 위에서 언급했던 희박한 데이터들을 처리 하기 위한 기법들로 Oracle사의 Oracle Express와 Hyperion사의 EssBase제품에 대해 살펴 보도록 하겠다.

2.4.1 청크 기반 구조

다차원 배열 형태로 데이터를 저장하는 MOLAP 시스템의 경우 데이터를 하나의 튜플로 저장하는 ROLAP의 경우와는 달리 희박한 데이터를 저장하는데 특별한 기법이 사용되어야 한다. 배열 기반의 희박한 데이터를 처리하기 위한 방법으로 전체 배열을 청크 단위로 나누고, 각 청크를 청크 내부의 값이 존재하는 셀의 비율에 따라 희박 청크와 밀집 청크로 구분한다. 밀집 청크에 대해서는 배열의 값만을 저장하고, 희박 청크의 경우는 셀의 색인 값으로 저장하는 기법을 사용한다[9]. 희박한 청크를 저장하기 위한 방식들로 Index-Value Pairs, Offset-Value Pairs, Compressed Sparse Row등을 사용한다[15].

2.4.2 EssBase에서 사용하고 있는 희박성 처리기법

EssBase는 희박성을 가진 데이터를 효과적으로 저장하기 위해 우선 차원을 희박 차원과 밀집 차원으로 구분한다. 그리고 밀집 차원에 속하는 항목들의 조합으로 블록을 만드는데, 이 블록은 다차원 배열로 생각할 수 있다. 이러한 블록의 수는 희박 차원 항목들의 조합 수 만큼 생성 될 수 있다. 이러한 희박 차원의 어트리뷰트의 조합을 통해 블록 포인터 배열을 만들어 밀집 차원의 데이터를 가리키도록 한다[7,

16]. 예를 들어 그림 2.7과 같은 매출 분석 모델을 살펴 보자.

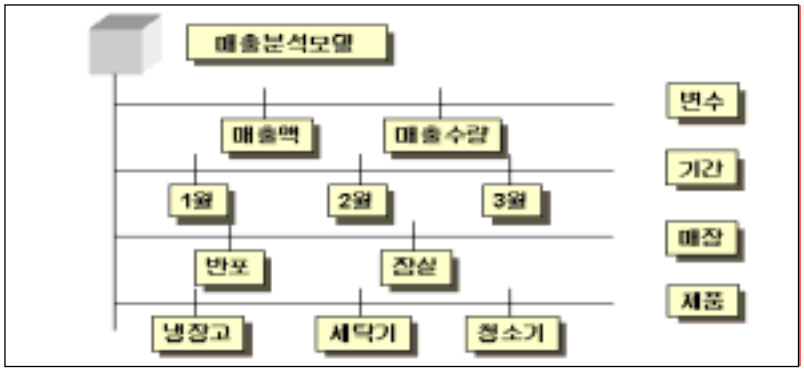

그림 2. 7 매출 분석 모델

반포 매장에서는 청소기를 판매하지 않고 잠실 매장에서는 냉장고와 세탁기를 판매하지 않는다고 가정하자. 그림 2.8은 매장 차원과 제품 차원을 밀집차원으로 나머지를 희박차원으로 설정한 경우 생성되는 블록들을 나타낸 것이다. 이에 반해 그림 2.9는 기간 차원과 변수차원을 밀집차원으로 설정하고 나머지를 희박차원으로 설정하여 생성된 블록을 나타낸다. 첫번째의 경우는 각 블럭은 6개의 셀로 구성되고 모두 50%의 희박성을 갖는다. 한편 두번째 경우는 반포와 청소기 잠실과 냉장고 잠실과 세탁기로 표현된 블록의 경우 데이터가 발생된 셀이 하나도 없으므로 이 블록은 물리적으로 생성되지 않고 각 블록은 0%의 희박성을 갖는다.

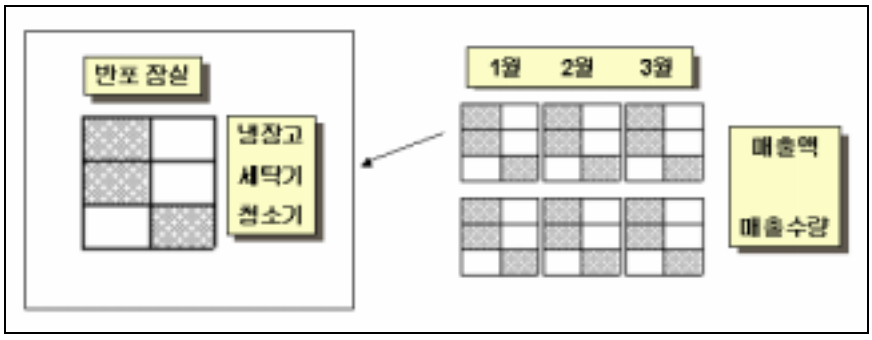

그림 2. 8 매장차원과 제품차원을 밀집차원으로 한 경우

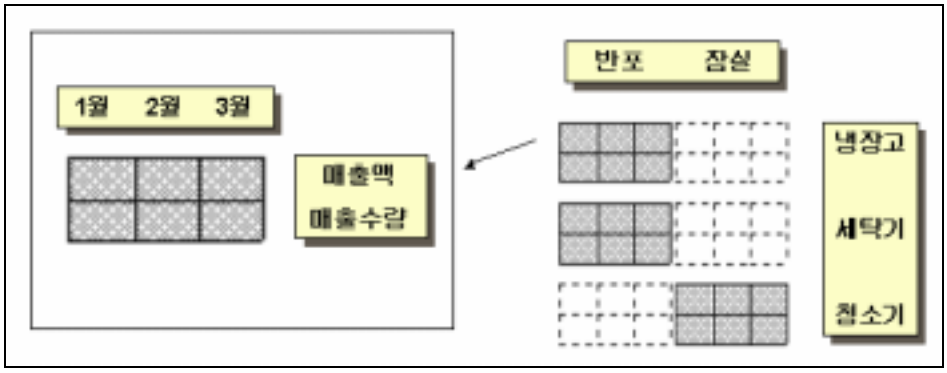

그림 2. 9 기간차원과 변수차원을 밀집차원으로 한 경우

2.4.3 Oracle Express에서 사용하고 있는 희박성 처리기법

Oracle Express의 경우 EssBase에서 사용하고 있는 기법과는 조금은 상이한 복합차원(Composite Dimension Method)이 사용된다[17]. EssBase의 희박성 처리 기법은 차원과 차원 사이의 희박성 관계를 가정한다면 Oracle Express의 경우는 하나의 차원내의 특정 항목과 다른 차원의 특정 항목사이의 희박성 관계를 가정하고 있다. 복합 차원의 방법은 EssBase의 방식과 마찬가지로 다차원 모델의 차원을 희박 차원과 밀집 차원으로 구분한 후 희박 차원들을 이용하여 복합 차원을 만든다[그림 2.10]. 하지만 이러한 복합 차원은 EssBase의 블록 포인터 배열과 같이 분리된 객체로 취급되는 것이 아니라 기존의 차원들과 마찬가지로 취급되어 진다.

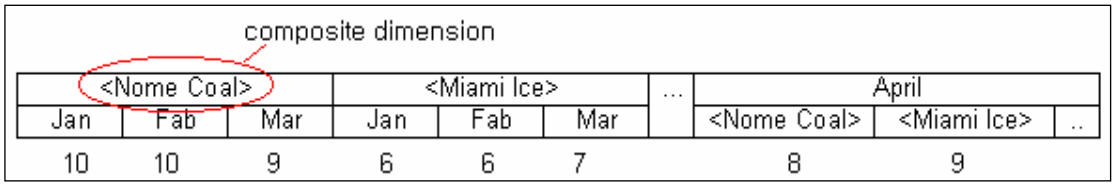

그림 2. 10 복합차원을 이용한 데이터 저장 예

Ⅲ. 웹 로그 데이터의 OLAP적용 시 희박성 원인 및 희박성 형태 분석

본 장에서는 실제 로그 데이터들에 대한 설명과 OLAP시스템에 적용하기 위해 적합한 형태로 변환하는 데이터 정제 및 변환과정으로 이루어진 사전처리 모듈에 대해 설명한다. 그리고 이러한 정제된 데이터들을 사용하여 희박성이 발생되는 원인과 그 형태에 대해 분석해 본다.

3.1 웹 로그데이터의 OLAP 적용 시 희박성 원인

웹 로그 데이터를 사용하여 사용자 질의에 온라인으로 다차원 분석을 가능하게 하기 위하여 OLAP을 적용할 시 희박성을 일으키는 원인들을 몇 가지로 분석해 수 있다. 첫번째로는 각 차원이 가지고 있는 항목들이 밀집 항목과 희박 항목으로 나눠지고, 희박 항목의 수가 밀집 항목의 수보다 많을 경우 이러한 차원들의 결합으로 인해 희박성이 발생 되게 된다. 예를 들어 스키장 사이트의 시간차원의 경우 늦가을이나 겨울에 성수기를 이루어 방문하는 사람의 수가 많아지고 나머지 시간대에는 방문자의 수가 희박하게 나타난다. 둘째로, 데이터 베이스 설계에 있어서 실제 로그 데이터 형식의 변화로 기존에 존재하지 않던 차원과 측정 값의 추가가 만들어 짐으로써 발생하게 된다. 마지막으로는 특정 기간동안만 존재하고 없어져 버리는 데이터들에 의해 발생된다. 예를 들면 이벤트가 수행되는 페이지의 경우 행사 기간이 지나

고 나면, 그 페이지는 없어지게 되고 해당 페이지의 측정값이 존재하지 않아 희박성이 발생한다. 이 중 희박성을 일으키는 가장 주 원인이 되는 첫번째 경우를 실제 세 가지 웹 로그 데이터 사용하여 살펴본다. 다음 절에서는 분석을 위해 사용된 실제 웹 로그 데이터에 대한 대략적인 설명과 데이터 모델 구조 그리고 OLAP에 적용되기 이전에 거치게 되는 사전 처리 모듈에 대해 설명한다.

3.1.1 실제 웹 로그 데이터와 OLAP 적용을 위한 데이터 모델

분석에 사용된 웹 로그 데이터의 형식은 기본적으로 W3C 확장 로그 형식을 기본으로 하며, 분량은 일주일 분량의 웹 로그 데이터를 사용하여 사전처리모듈을 거친 후 OLAP 서비스에 적용한다. 그리고 OLAP 서비스에 적용하기 위한 데이터 모델은 스타 스키마 구조를 따른다. 스타 스키마 구조는 4개의 차원 – 방문자, 페이지, 검색엔진, 시간 – 을 가지며, 방문횟수를 측정값으로 한다.

첫번째 사례에 사용된 사이트는 세계 각국의 스포츠 소식을 전하는 사이트로서 회원가입을 통해서 주요 정보 서비스를 받을 수 있다. 분석에서 사용한 이 사이트의 웹 로그 데이터는 표 3.1에 간단히 도식화 하였다.

표 3. 1 사례 1의 웹 로그 데이터의 설명

기 간	2000년 10월 28일 ~ 11월 4일
데이터용량	실제 로그 데이터 -> 2513975 records (163M) 사전 처리된 로그 데이터 -> 35635 records(1.83M)

사전처리가 끝난 로그 데이터와 고객 데이터 정보와 결합하여 데이터 큐브를 만들기 위한 데이터 모델은 아래 그림 3.1과 같다

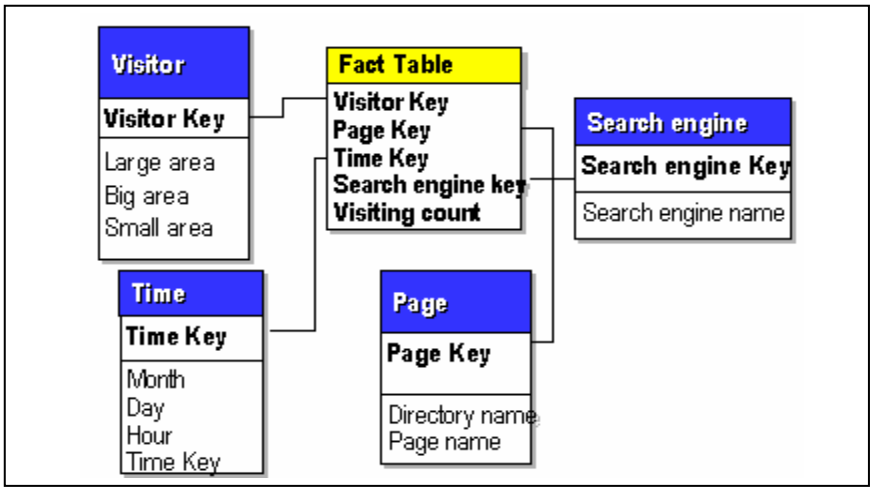

그림 3. 1 사례 1의 다차원 데이터 모델 항목 및 레벨 구조

두 번째 웹 사이트는 이화여자 대학교 데이터 베이스 연구실의 웹 서버의 로그 데이터를 사용하였다. 이 사이트에서는 회원 가입을 요구하지 않으며, 데이터의 용량이 적은 관계로 15일 간의 웹 로그 데이터를 사용하였다. 분석에 사용된 로그 데이터의 설명은 아래 표 3.2에 나타내었다.

표 3. 2 사례 2의 웹 로그 데이터의 설명

기 간	2001년 02월 01일 ~ 02월 15일
데이터용량	실제 로그 데이터 -> 198591 records (120M) 사전 처리된 로그 데이터 -> 72309 records(25.1M)

본 사이트에서는 고객 데이터가 존재 하지 않으므로 순수 웹 로그 데이터를 사용하였다. 데이터 모델은 그림 3.2와 같은 구조를 가진다.

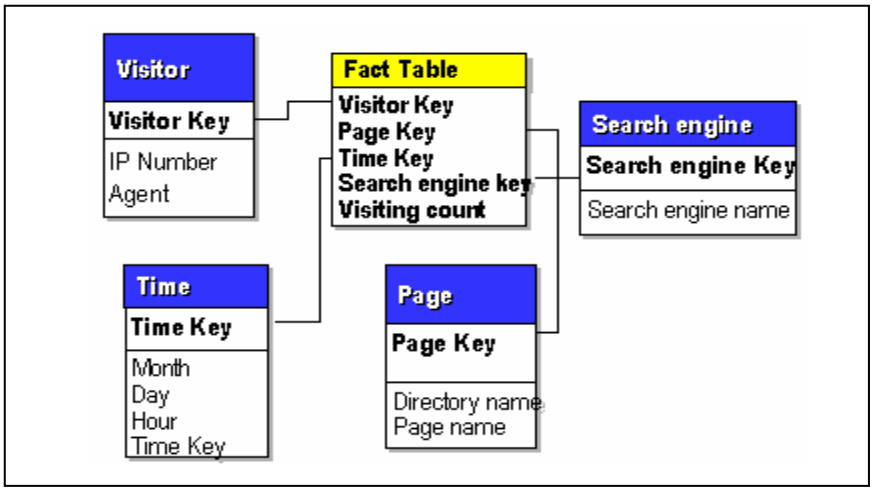

그림 3. 2 사례 2,3의 다차원 데이터 모델 항목 및 레벨 구조

세 번째 사례에 사용된 웹사이트는 자연 다큐멘터리 인터넷 방송국 사이트로서 첫 번째 사례에 나온 스포츠 정보회사 사이트와 같이 회원 가입을 통해 사이트의 중요 정보들을 볼 수 있다. 사이트의 로그 데이터에 대한 설명은 아래 표 3.3에 나타낸다.

표 3. 3 사례 3의 웹 로그 데이터의 설명

기 간	2001년 03월 28일 ~ 04월 03일
데이터용량	실제 로그 데이터 -> 21575 records (13.1M) 사전 처리된 로그 데이터 -> 19739 records(4.72M)

사례 3의 데이터 모델은 순수 웹 로그 데이터만을 사용하여 그림 3.2의 것과 같은 구조를 갖는다.

3.1.2 웹 로그 데이터의 사전 처리 모듈 설계

이번 절에서는 웹 로그 데이터를 OLAP서비스에 적용하기 이전에 사전 준비 과정인 데이터 사전처리 모듈에 대해 알아본다. 웹 로그 데이터 사전처리 모듈은 원시 로그 파일을 입력으로 받아 불필요한 데이터 포맷을 제거하는 데이터 정제 모듈과,

사용자 확인 모듈 그리고 세션 확인 모듈로 구성된다. 사전 처리 모듈의 전체 흐름도는 다음과 같다.

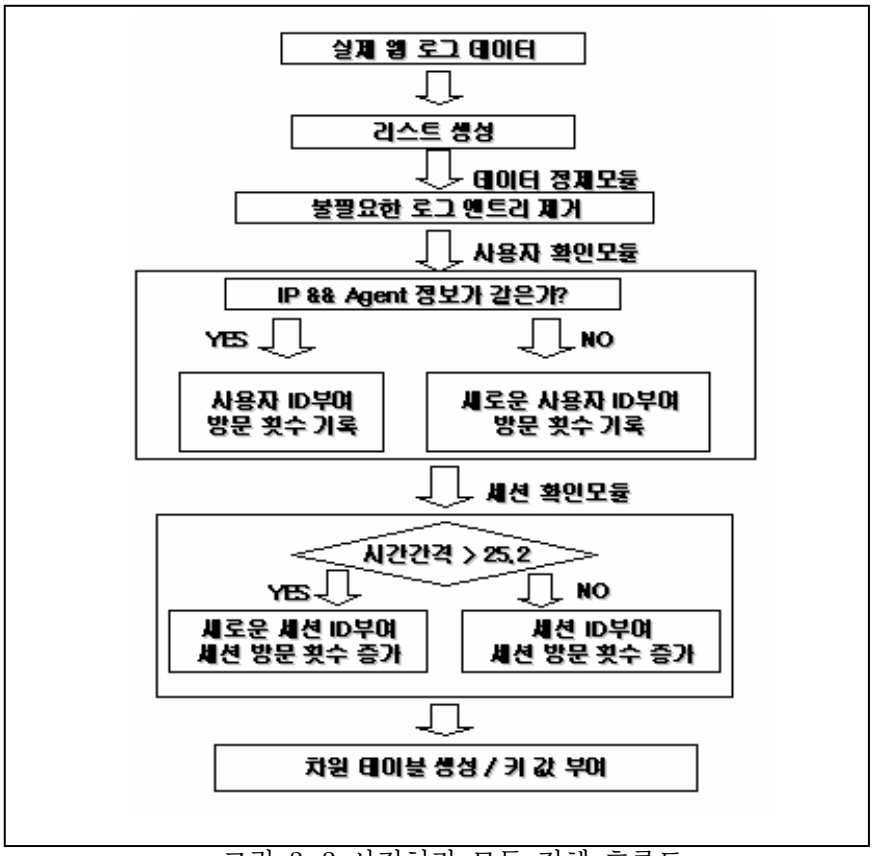

그림 3. 3 사전처리 모듈 전체 흐름도

3.1.2.1 데이터 정제 모듈

HTTP 프로토콜은 웹 서버로부터 요구 되는 모든 파일 들에 대해 각각의 트랜잭션으로 기록 되는데 이러한 특징은 사용자가 하나의 HTML문서를 보고자 할 때 그 문서에 부가된 그림 파일과 스크립트 파일이 동시에 다운로드 되어 로그 파일 내에 기록되게 된다. 이러한 불필요한 로그 데이터항목을 제거하는 것이 정제 모듈 내에서 행해지는 주요 작업이다. 즉 그림 3.4처럼 요청된 페이지의 확장자가 gif, jpeg,

jpg, map 나 avi, cgi와 같은 문서 파일 이외의 파일을 포함하는 로그 데이터 엔트리를 제거하는 과정이다.

```
Char file_extension[6][] = {“.jpeg”,”.jpg”,”.bmp”,”.gif”,”.avi”,”.map”}
BOOL Data_Cleaning()
{
    while( i < 6){
    if(cs_uri_stem의 확장자가 file_extension[i]와 같은경우){
            MainList내에서 해당 엔트리 제거
            break;
            }
            i++
     }
}
```

그림 3. 4 데이터 정제 모듈

3.1.2.2 사용자 확인 모듈

다음은 유일한 사용자를 확인하는 과정이다. 이 과정은 웹 서버내의 캐쉬나 방화벽, 프락시 서버들로 인해 장해를 받는다. 따라서 여기서 사용하는 사용자 확인 모듈은 휴리스틱한 방식을 기반으로 동일한 사용자의 경우 동일한 IP와 동일한 에이전트 정보를 가지게 된다는 가정하에 유일한 사용자를 확인한다[5]. 이는 로그 데이터 항목 중에서 c_ip항목과 c_agent항목을 통하여 구분할 수 있다. 이 두 항목이 같은 경우에만 동일한 사용자로 간주한다. 그림 3.5에 사용자 확인모듈과정을 보여준다.

```
Void User_Idnetification(void)
{
      if (c_ip내용과 c_agent내용이 다를 경우)
            새로운 사용자 ID를 부여한다.
      else
            기존의 사용자 ID부여 한다.
}
```

그림 3. 5 사용자 확인모듈

3.1.2.3 사용자 세션 확인 모듈

이 모듈은 각 사용자의 페이지 접근을 각 세션 단위로 나누는데 그 목표가 있다. 여기서는 가장 흔히 사용하는 방법 중 하나인 타임아웃 방식을 사용하여 페이지 접근 시간이 일정 시간을 초과하게 되면 다른 세션으로 간주하여 하나의 사용자에 의해 발생된 연속된 행동단위를 구분한다. [6]에서 제시한 경험적 통계적 데이터를 바탕으로 하여, Time항목과 Date항목을 고려하여 같은 날짜에 접근된 페이지 시간 간

```
void Session_Identification(void)
{
          while(전체 로그 리스트 검색){
                if(전 노드와 현 노드의 사용자 ID가 같으면)
                   if(date 필드 내용이 같으면)
                        if((현재시간 – 이전 방문시간)이 25.5분 이상이면)
                             session 증가
                   else       session 증가
                else       session 초기화
         세션 삽입    }}
```

그림 3. 6 사용자 세션 확인 모듈

격이 25.5분 이상이 되면 새로운 세션 ID를 부여한다. 그림 3.6에 사용자 세션확인 모듈과정을 제시한다.

3.1.3 각 차원별 데이터 항목을 이용한 희박성 원인 분석

앞에서 설명한 것처럼 웹 로그 데이터에서 희박성이 발생하는 주 원인은 각 차원의 데이터 항목 중 밀집 항목과 희박 항목이 뚜렷이 나타나고, 희박 항목이 밀집 항목에 비해 넓은 분포로 존재할 경우에 나타나게 된다. 본 절에서는 위와 같은 사실을 실제 로그 데이터를 사용하여 확인 해 보도록 하겠다. 우선 첫번째로 방문자 차원을 기준으로 보면 고객 정보와 결합하여 방문자 정보를 나타냈던 스포츠 정보 회사의 경우 방문자 차원의 계층 구조 중 대권역을 기준으로 하여 방문 횟수에 대한 분포를 그림 3.7로 나타내었다.

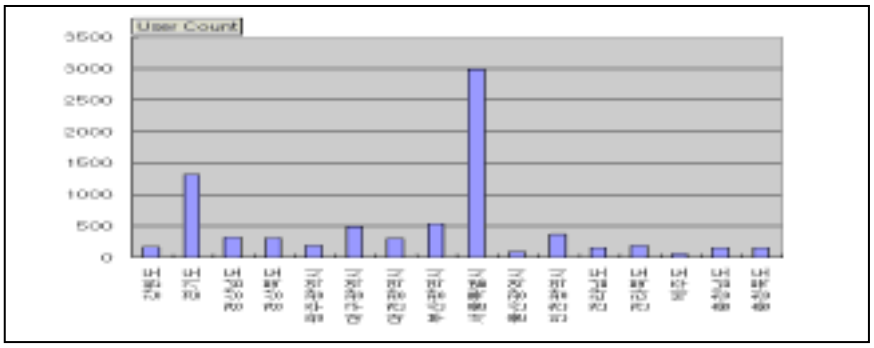

그림 3. 7 사례1 방문자 차원

나머지 사이트의 경우 고객 데이터의 부재로 계층구조가 형성되지 못하므로, 각 방문자를 차원 항목으로 나타낼 경우 방문자의 수가 수십 수백만 명에 이르러 이를 가시화 하기 힘들다. 따라서 방문 횟수별로 사용자의 수를 표 3.4의 형태로 나타낸

표 3. 4 사례2,3의 방문자 차원

사례 2 방문자 차원		사례 3의 방문자 차원	
방문 횟수	방문자 수	방문 횟수	방문자 수
1	1033	1	592
2	482	2	89
3	202	3	6
4	46	4	1
5	14		
6	1		

다. 방문자 차원의 경우 사이트의 방문 정도에 따라 위에서 보여 주는 것처럼 밀집 항목의 방문자와 희박 항목의 방문자로 나눠 볼 수 있고, 밀집 항목의 방문자가 희박 항목의 방문자보다 그 수가 매우 적음을 알 수 있다. 이러한 두 가지 분류가 발생되는 주 원인들은 그림 3.7에서 보여주는 것처럼 지역적 특징에 의해 발생 될 수 있고, 또한 자신이 사이트의 정보에 관심이 많아 회원 가입을 하여 사용하는 회원과 그렇지 않은 비 회원의 여부, 각 사용자들의 북 마킹의 유무, 그 사이트에서 주관하는 이벤트나 그룹 활동 등의 참여 여부, 그리고 개인의 취향이나 관심 분야, 직업 , 성별 등의 요소 들을 들 수 있다.

두 번째는 페이지 차원을 기준으로 각 페이지에 방문한 방문자의 횟수를 알아봄으로써 페이지 차원에서도 방문자 차원처럼 자주 방문 되는 인기 페이지 항목과 비인기 항목을 구별해 볼 수 있다. 그림 3.8 3.9 3.10은 세 사이트의 페이지 차원별 방문 횟수를 그래프로 보여준 그림이다.

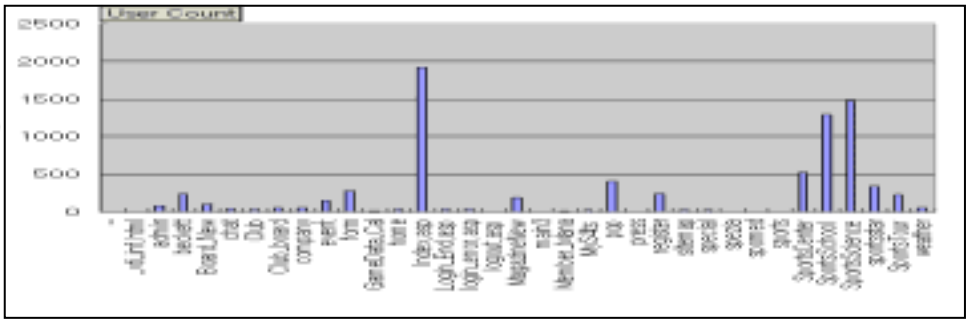

그림 3. 8 사례1 페이지 차원

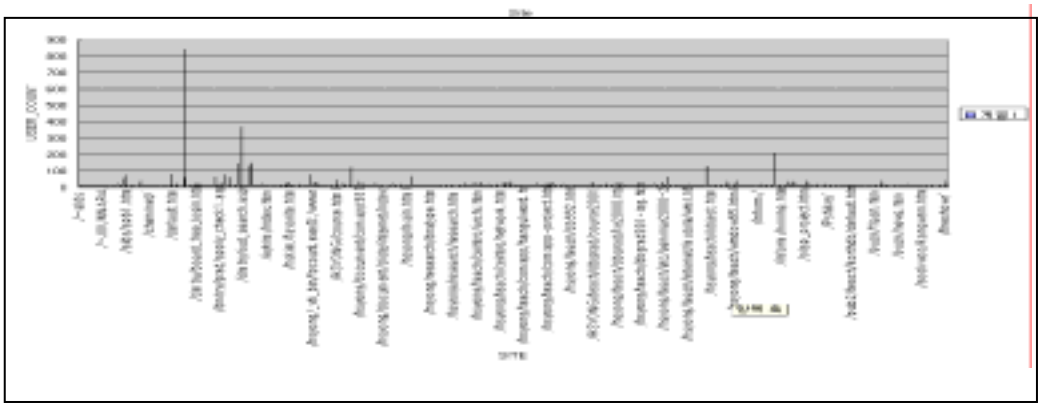

그림 3. 9 사례2 페이지 차원

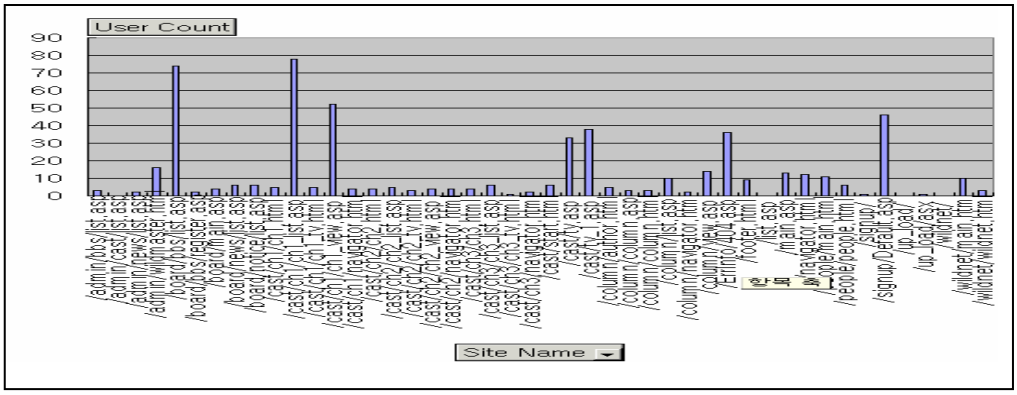

그림 3. 10 사례3 페이지 차원

웹 사이트의 페이지를 방문할 수 있는 방식은 크게 브라우저내의 북 마크를 이용한 히스토리컬 기법과 검색 엔진 이나 다른 참조 사이트들을 사용하는 방식 그리고

직접 방문으로 구분해 볼 수 있다. 히스토리컬 기법을 사용하는 경우 대부분 자신의 흥미 페이지나 그 사이트의 주된 페이지에 직접적으로 방문하게 됨으로 주제가 되는 페이지들의 방문횟수가 많아지고, 직접 방문의 경우 웹 사이트를 시작할 때의 초기 페이지와 이 페이지에 접속하기 위해 필요한 회원 가입페이지, 로그인 페이지등과 그 사이트의 주제가 되는 페이지에 방문 횟수가 많아진다. 따라서 위의 그림에서 볼 수 있듯이 나머지 페이지에서는 접속 분포가 넓게 분산되어 희박한 항목이 발생됨을 알 수 있다. 하지만 그 사이트 내에 주제 페이지의 수가 그 페이지를 찾기 위한 보조 페이지수보다 많이 존재할 경우 주제 페이지보다 주제 페이지까지 가기 위해 거쳐야만 하는 보조 페이지의 접속 횟수가 많이 질 수도 있다.

세 번째로는 시간 차원의 각 시간대별 계층구조를 기준으로 하여 각 시간대에 사용자들의 방문횟수를 가지고 이 차원 내에서의 접속 횟수가 많은 바쁜 시간대와 한가한 시간대를 알아본다. 그림 3.11 3.12 3.13은 세 사이트의 시간대별 방문횟수를 그래프화 한 것이다.

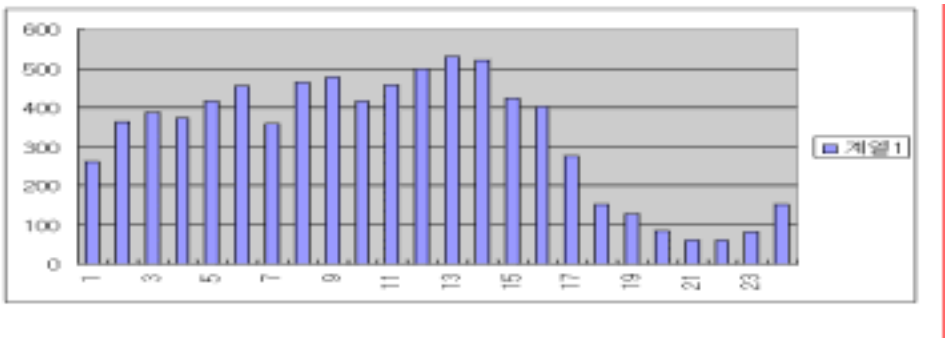

그림 3. 11 사례1 시간 차원

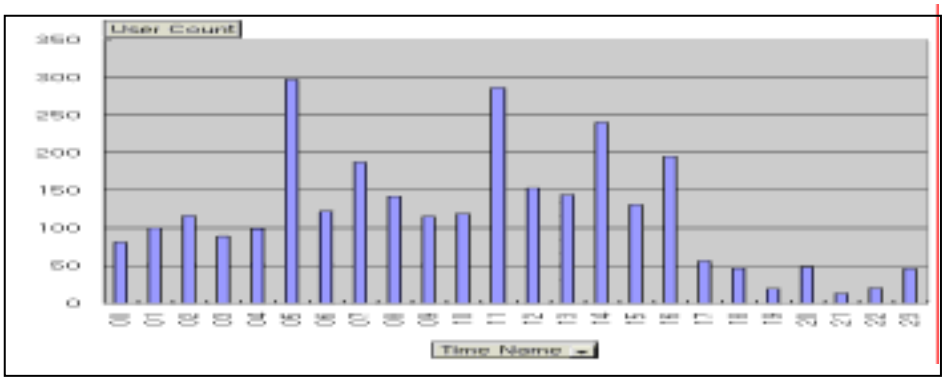

그림 3. 12 사례2 시간 차원

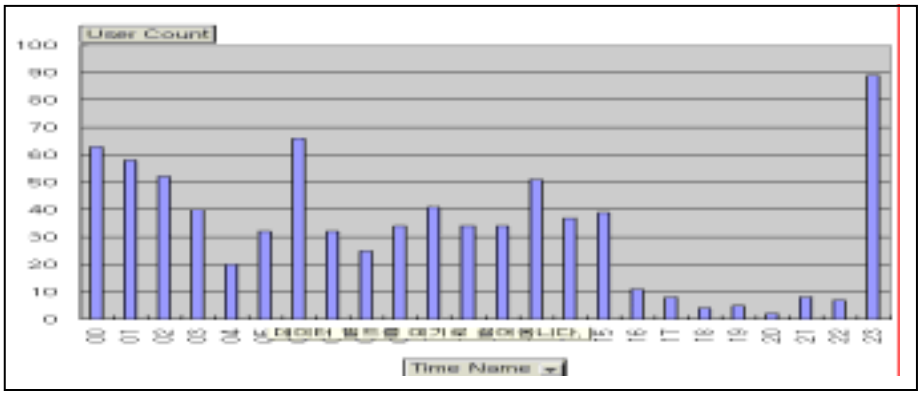

그림 3. 13 사례3 시간 차원

위의 예에서 각 시간대별 접속자수의 접속현황을 보면 저녁시간을 제외한 나머지 시간대에 접속자 수가 많은 바쁜 시간대 임을 볼 수 있다. 시간 차원의 경우 나머지 차원들과는 달리 밀집 항목의 수가 희박 항목에 비해 많은 비율을 차지 하고 있다. 하지만 만약 사이트의 성격이 시간 차원에 매우 민감한 스키장이나, 수영장 사이트와 같은 경우 접속자 수가 확연하게 시즌별 또는 월별 요일별 희박항목이 현저하게 나타나게 된다.

마지막으로 검색엔진 차원의 경우 각 검색엔진에 의해 참조되어 본 사이트에 오게

된 방문자의 수를 알아봄으로써 인기 검색 엔진과 비 인기 검색 엔진을 알 수 있다. 그림 3.14 3.15 3.16은 세 사이트의 검색 엔진에 의해 참조된 횟수를 그래프화 한 것 이다.

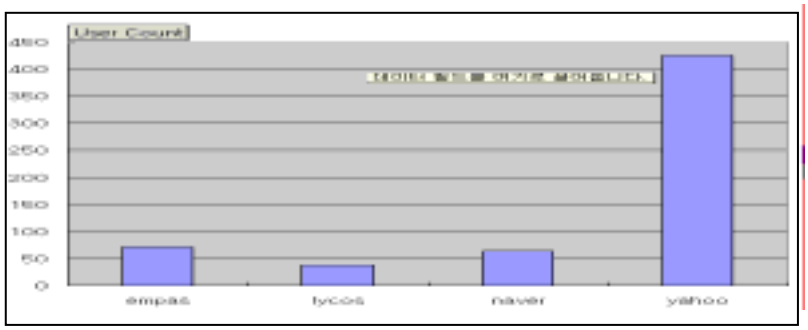

그림 3. 14 사례1 검색 엔진차원

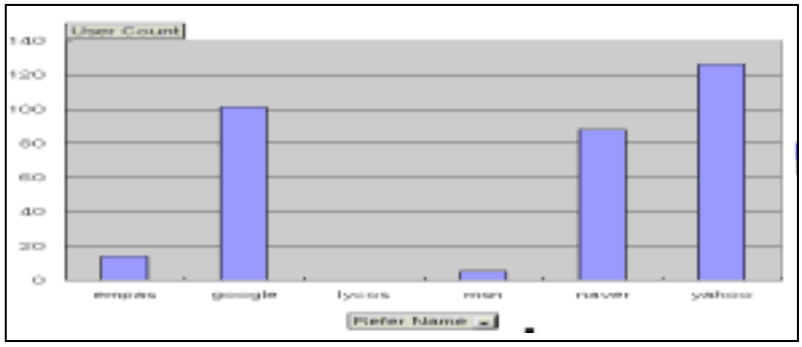

그림 3. 15 사례2 검색 엔진 차원

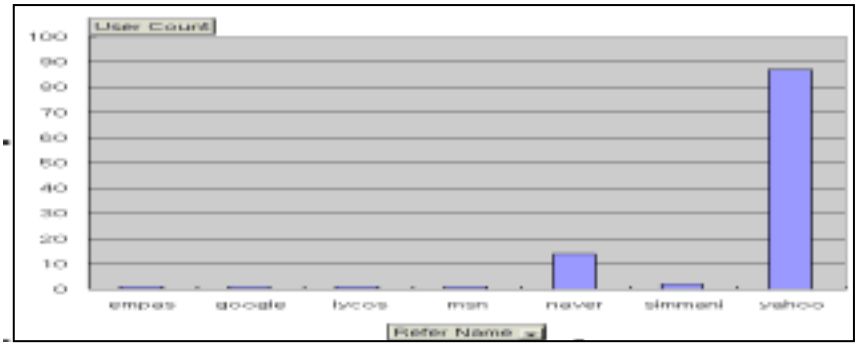

그림 3. 16 사례3 검색 엔진 차원

검색 엔진 차원의 경우 위와 같은 밀집 항목과 희박 항목의 뚜렷한 구분이 일어나는 원인은 각 검색엔진에 검색어를 다양하고 적절하게 제시 함으로써 자신의 사이트를 찾는 사람들이 여러 루트를 통하여 본 사이트를 발견할 수 있게 도와줌으로써 발생하게 된다.

세가지 실제 로그 데이터를 사용하여 위 4가지 차원들에 대해 우리는 각 차원에 희박 항목과 밀집 항목이 존재 하고, 희박 항목의 수가 밀집 항목의 수보다 많음을 확인할 수 있었다. 다음 장에서는 이러한 밀집 항목과 희박 항목으로 구성된 각 차원들의 결합에 의해 만들어 지는 희박성 형태에 대해 알아보도록 하겠다.

3.2 웹 로그데이터의 OLAP 적용 시 희박성 형태

본 장에서는 실제 웹 로그데이터를 OLAP적용 했을 시 만들어 질 수 있는 희박성 형태에 대해 알아보도록 하겠다. 다차원 모델에서 희박성 형태는 다른 차원들과의 상호 연계에 의해 만들어 진다. 다음 절에서는 웹 로그 데이터를 분석할 시 발생되는 2, 3차원의 희박성 형태에 대해 MS SQL 2000 Analysis Service와 DBMiner 3D Cube Explore를 사용하여 분석해 본다.

3.2.1 2차원 데이터 모델에서의 희박성 형태

위에서 제시한 3가지 실제 웹 사이트의 로그 데이터를 사용하여 OLAP적용을 위한 다차원 모델 생성 시, 2차원 모델을 기본으로 하여 만들어 지는 희박성 형태에 대해 알아 보도록 하겠다. 그림 3.17은 사례 3의 페이지 차원과 시간 차원을 이용하여 큐브를 생성했을 때 만들어 지는 희박성 형태 이다.

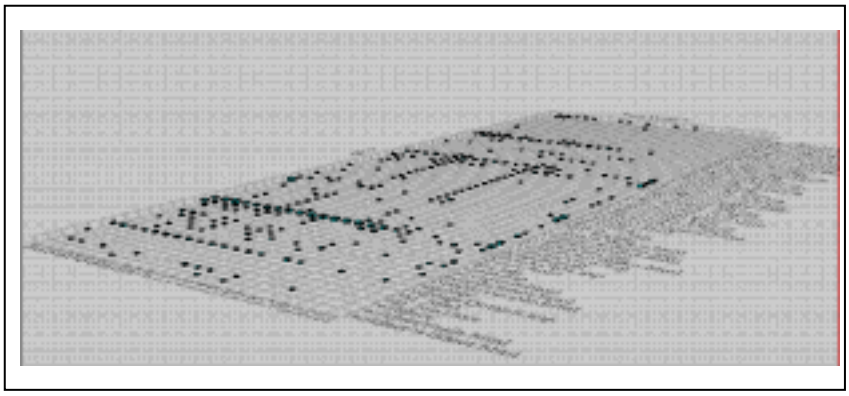

그림 3.17 사례 3의 2차원 희박성 형태 : 페이지-시간차원

위 그림에서 볼 수 있듯이 줄무늬(Stripe)형태의 희박성 형태가 시간 차원과 페이지 차원 내에서 교차가 이루어지면서 그리드(Grid)형태 와 클러스터(Cluster) 형태를 나타내게 된다. 이러한 희박성 형태는 시간차원에 있어서 바쁜 시간대에 해당되는 항목들에 있어서는 페이지 차원의 밀집 항목과 희박 항목에 관계없이 꾸준히 방문한 사람이 많고, 페이지 차원에 있어서는 인기 페이지에 해당되는 항목들에 대해서는 시간대에 제한받지 않고 방문횟수가 많은 경우 발생된다. 그림 3.18은 사례 1의 검색 엔진 차원과 시간 차원에 의해 만들어진 희박성 형태를 보여준다. 여기에서도 위의 경우와 마찬가지로 그리드(Grid)형태와 클러스터(Cluster) 형태의 희박성 형태를 발견할 수 있다.

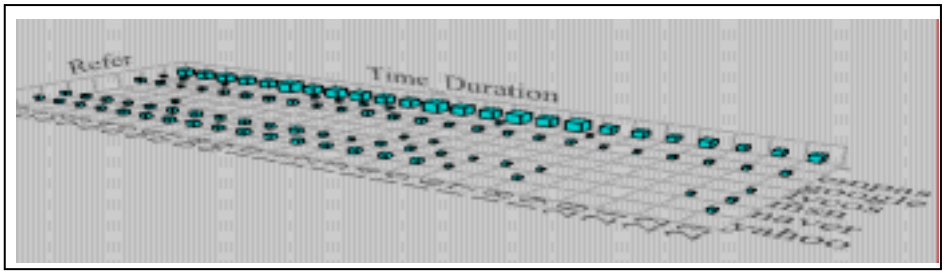

그림 3.18 사례 1의 2차원 희박성 형태 : 검색엔진-시간차원

이러한 희박성의 형태는 페이지차원과 시간차원 뿐만 아니라 사용자와 페이지 차원, 사용자와 시간차원 등 대부분의 2차원 데이터 모델에서 로그 데이터 종류와 관계없이 빈번하게 발생된다.

3.2.2 3차원 데이터 모델에서의 희박성 형태

이번 절에서는 3차원 내에서 발생되는 희박성의 형태에 대해 알아본다. 그림3.19와 3.20은 사례 3과 2의 페이지 차원과 시간차원 그리고 검색엔진차원 사이의 희박성 패턴을 보여 주는 그림이다.

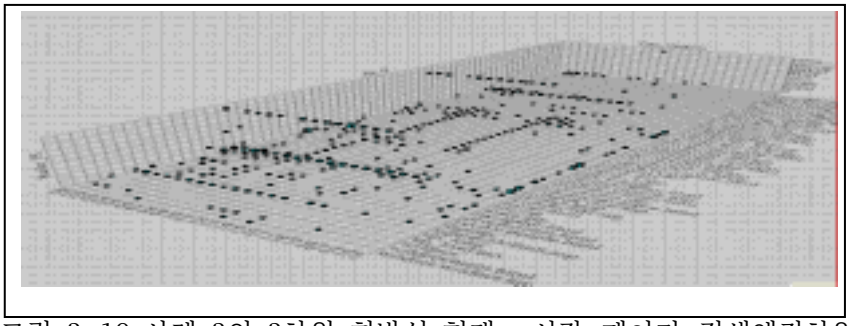

그림 3. 19 사례 3의 3차원 희박성 형태 : 시간-페이지-검색엔진차원

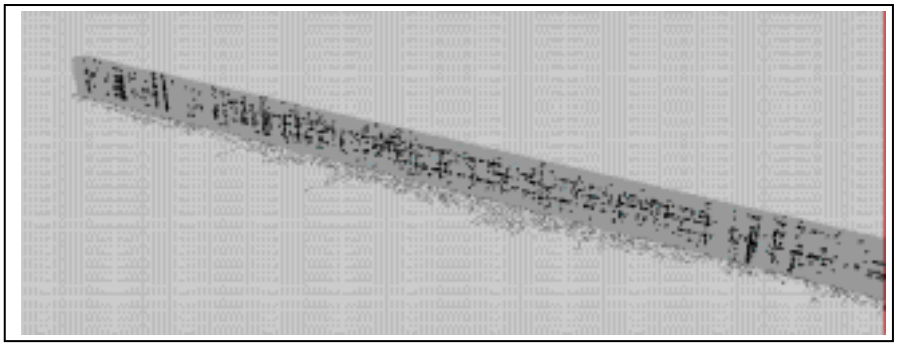

그림 3. 20 사례 2의 3차원 희박성 형태 : 시간-페이지-검색엔진차원

2차원 데이터 모델과 같이 3차원 데이터 모델에서도 그리드(Grid)형태와 클러스터(Cluster)형태의 희박성 형태가 발견됨을 볼 수 있다.

Ⅳ. C-MOLAP시스템의 특징 및 구현

OLAP에서 사용자의 질의에 대해 빠르게 응답하기 위해서는 사전 집계 연산은 반드시 필요한 일이라 할 수 있다. 따라서 여러 개의 관련 group-bys와 집계 연산을 계산하는 것은 OLAP 어플리케이션의 주요 연산자가 된다. 하지만 이러한 과정에서 중요하게 다루어 져야 할 점은 과연 이러한 사전 집계연산을 수행하는 데 있어서 무효 셀들을 어떻게 처리 할 것이며 집계 연산의 속도는 얼마나 빠르게, 적은 메모리 용량을 사용하여 수행할 수 있느냐 하는 것이 중요하다 할 수 있다. 본 장에서는 [9]를 바탕으로 하여 사전 집계 연산을 효율적으로 할 수 있고, 희박한 데이터들에 대해 압축 방식을 사용하는 MOLAP시스템을 C-MOLAP이라 부르기로 한다. 그리고, 이 시스템의 특징을 저장 방식 측면과 집계 연산 수행 측면에서 제시하고 주요 알고리즘을 제시하도록 하겠다.

4.1 C-MOLAP시스템의 저장 방식

첫 번째 저장 방식 측면에서 봤을 때 이 시스템은 기본적으로 MOLAP시스템을 기반으로 하기 때문에 모든 데이터들을 배열 형태로 저장 하게 된다. 배열에 의해 데이터들이 저장 될 경우 각 데이터들의 위치가 처음부터 결정됨으로 인덱스에 영향을 미치지 않고 갱신 될 수 있다는 장점을 가지게 된다. 하지만 이는 큐브 내의 희박성을 일으키는 주 원인이 되기도 한다. 현재 C-MOLAP 시스템에서는 이러한 희박한 데이터 셀들을 처리 하기 위해 n차원의 데이터 배열을 더 작은 단위의 n차원 배열 형

태인 청크 단위로 나누고, 각 청크에 대해 희박 청크와 밀집 청크로 나누어 희박 청크에 대해 Chunk-Offset Compression을 적용한다.

본 시스템에서는 OLAP연산 수행 시 청크 단위로 디스크 내에서 메모리로 읽어오고, 쓰기 때문에 디스크 I/O단위인 디스크 블록 사이즈로 청크 크기를 결정 하게 된다. 따라서 각 청크의 사이즈는 하나의 디스크 블록 사이즈와 가장 가까운 크기로 결정한다.

$$|Ci| = |Di| * (1/2)^n$$

|Ci| = i번째 차원의 청크 사이즈,|Di| = i번째 차원의 청크 사이즈, n:반복 횟수

그림 4. 1 청크 사이즈 결정 식

그림 4.1은 청크 사이즈를 결정하는 수식으로 [9]에서 가정하는 것 처럼 각 차원의 사이즈와 관계없이 일정 크기로 청크 사이즈를 결정하는 것과 달리 차원의 사이즈를 고려하여 청크 사이즈를 결정할 수 있다. 위의 수식을 통해 제시된 각 청크 사이즈들의 조합이 디스크 블록 사이즈보다 적거나 같아졌을 때, 각 차원의 항목의 수를 $(1/2)^n$으로 나눠가며 청크 사이즈를 결정 할 수 있다. 그림 4.2는 그림 4.1을 구현한 알고리즘이다.

```
void Chunk_Size(int dim_cnt, int disk_page) {
// dim_cnt: 전체 차원의수, disk_page:디스크 블럭 사이즈
        int chunk_size = 1; // 전체 청크 사이즈 결정
        int *temp, temp1; //
        for(int j = 0; j < 100; j++) {
          for(int i = 0; i < dim_cnt; i++){
                temp1 = (int)((Dimension_I[i].dim_size ) / (pow(2,j)));
        if(temp1   > 1 ) //각차원의 항목은 적어도 둘이상의 항목을 가져야 한다.
```

```
temp[i] = temp1;
chunk_size *= temp[i];
}
if(chunk_size < disk_page)
        break;
```

그림 4. 2 청크 사이즈 자동 결정 모듈

현재 시스템 내에서는 하나의 디스크 블록 사이즈를 50K로 가정하고, 차원의 사이즈가 50*2160*10000이라 할 경우 그림 4.2의 알고리즘에 따라 7번의 순환을 지난 후에 3*16*78사이즈의 청크 사이즈를 결정 할 수 있다. 청크 사이즈를 결정하기 위한 자동 결정 알고리즘은 위에서 제시한 간단한 방식 이외에 좀더 정교한 방식이 연구되어 져야 한다. 최적의 청크 사이즈는 저장 공간의 낭비를 줄일 수 있고, 디스크 I/O타임을 줄일 수 있어 성능에 영향을 미칠 수 있기 때문이다.

이러한 각각의 청크들에 대해서는 청크 내 셀들이 40%이상 데이터 값을 가지는 지의 유무에 따라 밀집 청크와 희박 청크를 정의 한다. 밀집 청크에 대해서는 모든 셀 내의 값만을 저장한다. 반면, 희박 청크에 대해서는 “chunk-offset compression” 방식을 사용한다.[그림 4.3]

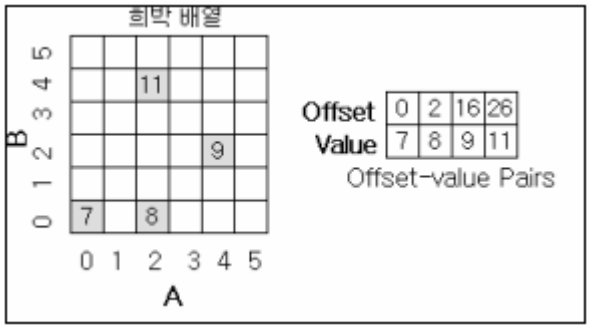

그림 4. 3Chunk-Offset compression

이 방식에서는 청크 내의 데이터 값들의 배열의 인덱스 값을 계산하는 행 우선 방식이나 열 우선 방식을 사용하여 offset 값을 정수 값으로 계산하고, offset 과 데이터 값들의 쌍을 저장한다. 희박 청크의 경우 청크의 크기가 항상 변하게 됨으로 데이터 파일의 시작에 메타 데이터로서 각 청크의 길이를 저장하게 된다. 아래 그림 4.4은 각 청크들의 각 차원별 시작항목과 끝 항목을 계산해서 청크의 범위를 저장하는 메타 데이터와 실제 데이터 값들의 저장 구조 이다.

```
typedef struct Chunk_Numbering{
        int start_x;   int end_x;
}Chunk_N; // 각 청크의 차원들의 시작항목과 마지막 항목을 계산한다.
typedef struct A_Mem_Save{
    int offset;                    // 각 chunk영역 내에서의 data offset
    int data_val;
    int chunk_num; }A_Mem;
```

그림 4. 4 각 청크 내의 메타 데이터와 실제 데이터 저장 구조

4.2 C-MOLAP시스템의 집계연산 수행방식

사전 집계 연산을 수행하는 측면에서 살펴 보도록 하겠다. N개의 차원으로 이루어진 데이터 큐브에 대해 group_bys계산은 2^n개가 성립되게 된다. 2^n개의 모든 group_bys들을 메모리 내에 모두 유지 시키기 위해서는 충분한 메모리 용량이 있을 경우에만 가능하다. 따라서 [9]에서 제시하고 있는 사전 집계 연산 알고리즘은 최소한의 메모리를 사용하여 여러 개의 group_bys를 동시에 계산할 수 있도록 하는데 그 목표를 두고 있다. 현재 C-MOLAP시스템에서 사용하고 있는 사전 집계 연산 알고리즘은 “A Single-Pass Multi-Way Array Cubing Algorithm”으로 전체 배열을 메모리 내에 로드 하는 방식을 좀더 개선해서 사전 집계 연산을 수행하는데 있어서 최소의

메모리를 사용하여 매 단계 마다 관련된 부분만을 로드해서 한번의 디스크 스캔으로 모든 사전 집계 연산을 수행한다. 다음은 이 알고리즘 내에서 주요하게 다루어 지는 개념들과 프로그램 모듈들 이다.

- Minimum Memory Spanning Tree(MMST)

MMST는 큐브 내의 group_bys를 동시에 계산할 수 있도록 하기 위해 필요한 최소한의 메모리를 구하는 자료 구조를 가르킨다. 그림 4.5에서 보여 주는 것처럼 3개의 차원을 가지며, 각 차원의 항목의 수는 16*16*16이고, 청크 사이즈는 4*4*4로 가정하는 큐브가 있고, 이 큐브는 A,B,C차원 순서대로 읽어 나간다고 가정하자.

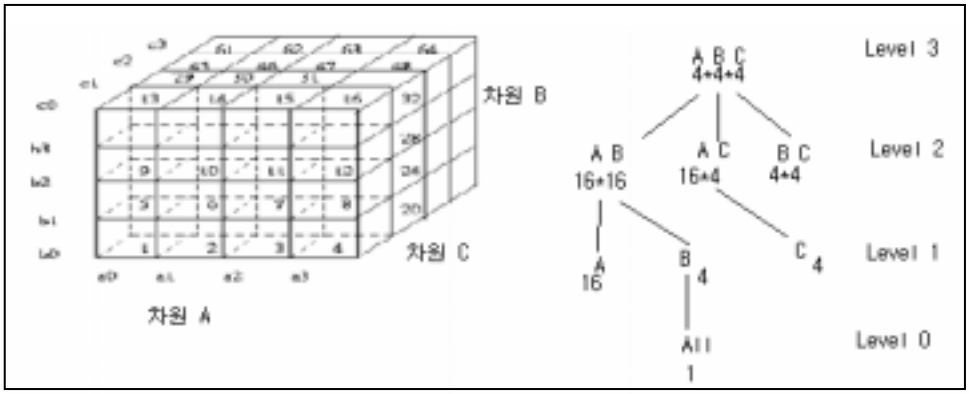

그림 4. 5 3차원 큐브와 MMST

위에 그림에서 우리가 BC라는 서브 group_by를 계산한다고 할 때, 1~4번까지의 청크들에 있어서 1~4번까지의 청크를 읽어서 b_0c_0의 group_by값을 계산할 수 있다. 따라서 BC를 위해서는 하나의 청크 크기만큼의 메모리를 가지고서 모든 BC관련 group_bys를 계산할 수 있게 된다. 이와 같이 AC의 경우는 16개의 청크를 할당해야 한다. 따라서 전체 서브 group_bys에 대한 메모리 할당은 다음과 같은 식으로 나타

낼 수 있다.

$\prod_{i=1}^{p}|D_i| \times \prod_{i=p+1}^{n-1}|C_i|$ $|D_i|$: 차원 i의 사이즈 $|C_i|$: 차원 i의 청크 사이즈

그림 4. 6 서브 group_bys의 메모리 할당량

p는 부모 노드와 현재 자신의 노드의 접두사가 같은 부분의 길이를 의미한다. 각 차원의 항목의 수는 각 청크의 항목의 수에 비해 매우 크다는 것을 우리는 알고 있다. 따라서 이 수식에서 볼 수 있듯이 전체 메모리를 할당하는데 있어서 중요한 영향을 미치는 요소는 각 차원의 순서를 어떻게 정하느냐에 따라 좌우 되게 된다. 아래 그림 4.7은 이를 위한 최적의 차원 순서를 정하는 수식이다.

최적의 차원의 순서 : |D1| <= |D2| <= .... <= |Dn|

그림 4. 7 최적의 차원 순서

그림 4.7에서 볼 수 있듯이 항목 수가 적은 차원에서 큰 차원 순서대로 읽는 것이 가장 작은 메모리 양을 요구함을 알 수 있다. 그림 4.6와 4.7의 수식을 통해 우리는 전체 group_by를 계산하는데 필요한 최소한의 메모리 양을 계산할 수 있고, 각각의 전 group_by를 계산하기 위해 필요한 메모리 양을 계산해 낼 수 있다. 그림 4.8에서는 전체 MMST사이즈와 각 MMST의 서브 group_bys을 계산하는데 필요한 메모리 양 을 계산하는 모듈을 보여 준다.

```
typedef struct MMST_Tree{
        int size_info[MAX_DI];
        int start_p; // 각각의 서브 group_by의 계산 량을 저장 할 일차원 배열
                     의 시작점을 기억한다.
        int level; //자신이 해당된 레벨을 기억한다.
```

```
int chunk_su; // 해당 서브group_bys를 계산하는데 필요한 청크 수
} MMST_T;
Total_MMST(int dim_cnt) // dim_cnt : 총 차원의 수
{
        iter = 전체 서브 group_bys을 계산해 낼 수 있다.;
        bool flag = false; // 처음 노드가 1로 세팅되어 있는지 확인하기 위
                                해 사용된다.

        cnt = 0;// 각 서브 group_bys의 레벨 수를 결정
        first_I = 0;   //fisrt_I 비트 중 가장 처음으로 1로 세팅된 큐브 찾기
        MMST_Size = 0; // 전체 MMST Size저장
        temp_data = 1; // 서브 group_bys의 메모리 할당량 저장

        MMST_Size = chunk_size;  // 첫번째 루트 큐브의 청크 사이즈
        MMST[0].start_p = 0;
        temp_data = chunk_size;
                for(int i = 1; i < (iter-1); i++){
// 1번째 서브 group_by에서 부터 마지막 서브 group_by들까지 계산한다.
                cnt = 0;
                MMST[i].start_p  = MMST[i-1].start_p + temp_data;
// 일차원 배열내에 현재 group_by가 계산되서 저장되어야 할 위치를 계산한다.
                temp_data = 1;

                if(MMST_Ini[i*dim_cnt + 0] == 1){
    // 첫번째 노드가 1인지 체크한다. 즉A,B,C중 A가 1로 세팅되어 있는지 확인
                first_I = 0;
                flag = true; // 처음 index가 1로 setting 되었는지 확인
        }
                for(int j = 0; j < dim_cnt; j++){
                    if(flag == true){ // 처음 노드가 1로 세팅되어 있는 경우
                        temp_data *= (Dimension_I[j].dim_size);
```

```
        MMST[i].size_info[j] = (Dimension_I[j].dim_size);
        if(first_I > j)
                first_I = j;
        if(MMST_Ini[i * dim_cnt + j+1] == 1)
// 현재 자신의 노드와 연속적으로 다음 노드가 존재하는지 확인
                flag = true;
        else
                flag = false;
        cnt++;
        }
   else { // 처음 노드가 1로 세팅되어 있지 않은 경우
                        flag = false;

          if(MMST_Ini[i*dim_cnt + j] == 1){
                        temp_data *= (Dimension_I[j].dim_chunk_s);
                        MMST[i].size_info[j] = (Dimension_I[j].dim_chunk_s);
                        if(first_I > j)
                                first_I = j;
                        cnt++;
                }}
          }
             MMST[i].level = cnt;
             MMST_Size += temp_data; // 총 메모리 사이즈를 계산한다


        }
        // All에 해당하는 MMST구조를 완성한다.
        MMST[iter-1].start_p = MMST[iter -2].start_p + temp_data;
        MMST[iter-1].level = 0;
        MMST_Size++;
```

그림 4. 8 MMST내 서브 group_bys의 메모리 할당량 구하는 알고리즘

Ⅴ. 테스트 데이터 생성기와 성능 평가 시스템 구현 및 결과 분석

본 장에서는 Ⅲ 장에서 제시한 웹 로그 데이터의 희박성 형태를 기반으로 성능 평가를 위해 데이터 모델 및 질의 모델을 제시하고, 2,3 차원의 데이터 모델에 따라 희박성 생성 알고리즘을 제시한다. 그리고 생성된 테스트 데이터를 이용하여 MS SQL 2000 Analysis Service, Oracle Express Server, C-MOLAP 시스템들에 대해 실제 OLAP모델을 구성한다. 그리고 각각의 제품에 대해 설정한 질의 모델들을 수행하여 웹 로그 데이터 희박성에 따른 성능을 측정할 수 있는 프로그램을 구현한다.

5.1 웹 로그 데이터의 희박성 형태에 따른 데이터 생성기 설계 및 구현

본 절에서는 앞에서 분석한 2,3차원에서의 희박성 형태에 따라 테스트 데이터 생성기에서 사용하게 될 다차원 데이터 모델과 질의 모델을 설계 하고, 이를 바탕으로 희박성 형태에 따른 테스트 데이터 생성기를 설계한다.

5.1.1 데이터 모델 및 질의 모델 설계

성능 평가를 위해 사용된 데이터 모델들은 차원 테이블은 웹 로그데이터를 분석하는데 가장 핵심이 되는 방문자, 페이지, 시간을 기반으로 하고, 측정 값은 방문 횟수로 한다. 각각의 데이터 모델들의 계층 구조와 자세한 정보는 아래 표 5.1와 같다.

표 5. 1 차원정보

차원	방문자	페이지	시간
항목 수	10,000	50	2,160
계층 수	1	4	3

위에서 제시한 3가지 차원 정보를 이용하여 본 논문에서 사용하게 될 2,3 차원 데이터 모델은 다음과 같다.

- 2 차원

2차원 모델은 방문객과 시간, 방문객과 페이지 차원들로 이루어 진다. 시간 차원의 경우 달, 날짜, 시간 3개의 계층으로 이루어 지고, 페이지 차원의 경우 루트 디렉토리, 2개의 서브 디렉토리, 페이지 4개의 계층으로 이루어 진다. 방문객과 페이지 차원으로 이루어진 2차원 데이터 모델은 사례 A로 방문객과 시간차원으로 이루어진 것에는 사례 B로 가정한다. 사례 A와 B에 대한 데이터 모델과 자세한 정보는 그림 5.1과 표 5.2과 같다.

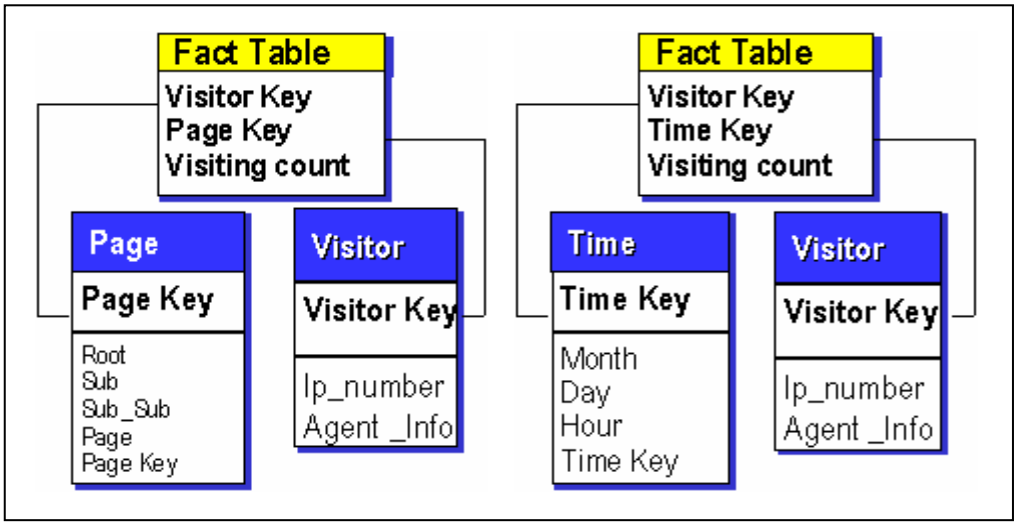

그림 5. 1 2차원 데이터 모델

- 3차원

3 차원 모델은 방문객, 시간, 페이지 3가지 차원들로 이루어 진다. 3가지 차원 테이블의 계층 구조는 2차원에 사용된 정보와 같고, 3차원 데이터 모델은 사례 C로 가정한다. 사례 C에 대한 데이터 모델과 자세한 정보가 그림 5.2와 표 5.2과 같다.

표 5. 2 2,3차원 데이터 모델의 상세정보

사례	차원	전체 레코드 수	희박성(10%)
A	방문객, 페이지	500,000	50,000
B	방문객, 시간	21,600,000	2,160,000
C	방문객, 시간, 페이지	1,080,000,000	108,000,000

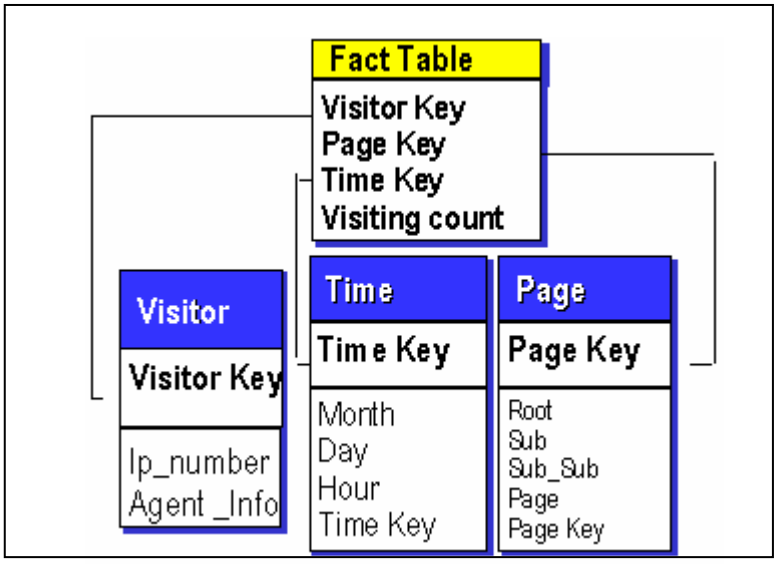

그림 5. 2 3차원 데이터 모델

● 질의 모델

2,3 차원 데이터 모델에 의해 만들어진 테스트 데이터를 사용하여 OLAP시스템들의 성능을 평가하기 위해 사용되는 질의 모델은 다음과 같다. 우선 OLAP시스템에서 일반적으로 자주 사용하고 있는 질의로는 그림 5.3과 표 5.3과 같다[7, 8].

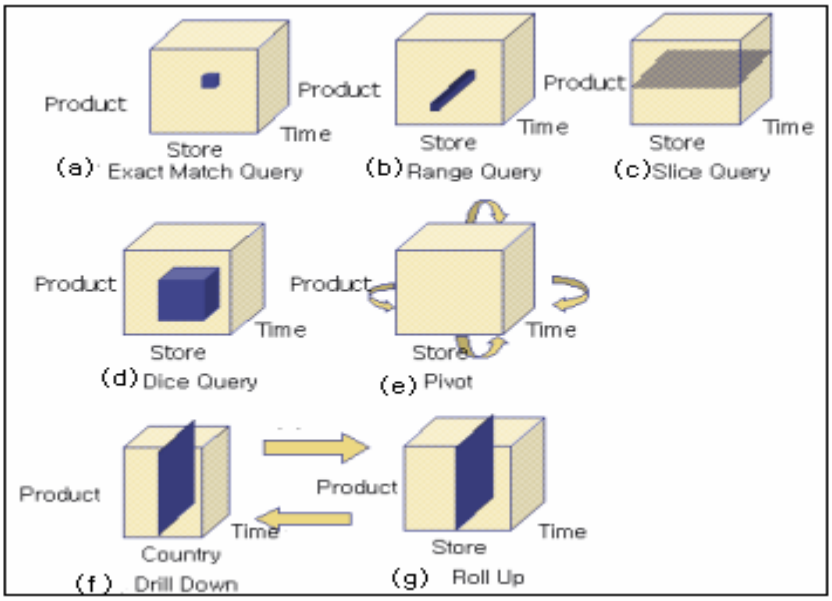

그림 5. 3 OLAP의 일반적인 7가지 질의 종류

표 5. 3 OLAP의 일반적인 7가지 질의에 대한 자세한 설명

Exact Match Query	큐브의 모든 차원의 특정 항목에 해당하는 하나의 셀을 선택하는 질의
Range Query	기간 차원과 같이 연속적인 차원의 연속적인 항목 값에 해당하는 셀들을 선택하는 질의
Slice Query	다차원 모델 중 하나의 차원에 대한 항목 값을 고정시키고 고정된 항목에 대한 큐브 데이터의 서브 집합을 선택하는 질의
Dice Query	다차원 모델 중 하나 이상의 차원에서 하나 이상의 고정된 항목 값에 대한 큐브 데이터의 서브 집합을 선택하는 질의

Pivoting	큐브를 보여주던 차원 기준을 달리하여 큐브를 회전 시켜 새로운 기준에 맞도록 데이터를 보여주는 질의
Drill Down	데이터를 차원의 높은 계층에서 낮은 계층을 따라 상세 데이터로 보여주는 질의
Roll Up	데이터를 차원의 낮은 계층에서 높은 계층을 따라 통합된 데이터로 보여주는 질의

본 논문에서는 위의 7가지 일반적인 OLAP 질의와 더불어 OLAP 질의 중 많은 항목들 사이에 숨어 있는 데이터를 쉽게 파악하고자 할 경우 사용되는 랭킹 질의를 사용하여 웹 로그 데이터 분석을 하는데 있어 주 방문객들을 파악할 수 있게 하는 Top500질의를 설계하였다. 위에서 제시 했던 사례 A,B,C 데이터 모델에 대해 7가지 일반적인 OLAP질의 모델은 아래와 같다.

표 5. 4 사례 A의 일반적인 OLAP질의 모델

질의 종류	질의 내용
Exact Match	이벤트 페이지 (PK00003)에 방문자(VK03018) 가 방문한 횟수
Range	자주 방문하는 방문자 VK03000 ~ VK03600 가 메뉴 S00001에 방문한 횟수
Slice	방문자(VK03018)가 전체 페이지에 방문한 횟수
Dice	전체 방문자에 대해 메뉴 S00001에 방문한 횟수
Pivot	열에 사용자 차원, 행에 페이지 차원으로 보여 주던 것을 열에 페이지, 행에 사용자 차원으로 보여주시오.
Drill Down	방문자 VK0318이 페이지차원의 R0001디렉토리를 기준으로 하여 한 차원 내린 디렉토리의 방문 횟수를 보여주시오.
Roll Up	방문자 VK0318의 페이지 차원의 페이지 레벨을 기준으로 한 차원 위인 상위 디렉토리의 방문 횟수를 보여주시오.

표 5. 5 사례 B의 일반적인 OLAP질의 모델

질의 종류	질의 내용
Exact Match	시간 (TK00042)에 방문자(VK03018) 가 방문한 횟수
Range	자주 방문하는 방문자 VK03000 ~ VK03600가 8월 2일 하루 동안 방문한 횟수
Slice	방문자(VK03018)가 전체 시간대에 방문한 횟수
Dice	전체 방문자에 대해 8월 2일 하루동안의 방문한 횟수
Pivot	열에 시간 차원, 행에 사용자차원으로 보여주던 것을 열에 사용자, 행에 시간 차원으로 보여주시오.
Drill Down	방문자 VK3018의 시간 차원의 8월 2일을 기준으로하여 한 차원 내린 시간대의 방문 횟수를 보여주시오.
Roll Up	방문자 VK3018의 시간 차원의 8월 2일을 기준으로 한 차원 위인 8 월달의 방문 횟수를 보여주시오.

표 5. 6 사례 C의 일반적인 OLAP질의 모델

질의 종류	질의 내용
Exact Match	이벤트페이지(PK00003)에 방문자(VK00012)가 TK02041시간대에 방문한 횟수
Range	8월 1일에 자주 방문하는 방문자그룹 (VK00001 - VK00100)이 새로 단장한 페이지 PK00001 - PK00030에 방문한 횟수
Slice	방문자(VK00001)가 전시간대에 전 페이지에 걸친 방문 횟수
Dice	자주 방문하는 방문자그룹이 (VK00300 ~ VK00330) 8월 1일날 전 페이지에 대한 방문 횟수
Pivot	방문자 VK00012에 대해 행에 페이지 차원와 열에 시간 차원로 보여 줬던 것을 행에 시간 차원 열에 페이지 차원으로 방문 횟수를 보여 주시오.

Drill Down	사용자 'VK000012' 와 'PK00008' 페이지에 대해 시간차원에서 8월 1일을 기준으로 보여 주었던 것을 한 차원 내린 시간을 기준으로 방문 횟수를 보여 주시오.
Roll Up	사용자 'VK000012' 와 'PK00008' 페이지에 대해 시간차원에서 8월 1일을 기준으로 보여 주었던 것을 한 차원 올려 보여 주도록 한다.

표 5.7은 3가지 사례에 대해 각각 우수 고객을 얻기 위해 수행되는 Top500 질의에 대해 보여준다.

표 5. 7 사례A, B, C의 Top500질의 모델

사례 A	현재 이벤트가 수행되는 페이지(PK00003)에 대해 가장 많이 방문한 사람의 Top500을 구하시오.
사례 B	이번 한달 동안(10월) 가장 많이 방문한 사람의 Top500을 구하시오.
사례 C	이번 한달 동안(10월) 새롭게 단장한 메뉴(PK00003)에 대해 가장 많이 방문한 사람의 Top500을 구하시오.

5.1.2 테스트 데이터 생성 알고리즘

본 절에서는 5.1.1에서 정의 했던 데이터 모델 각각에 대해 희박성 형태에 따른 테스트 데이터를 생성 하기 위한 알고리즘을 설계한다.

5.1.2.1 클러스터(Cluster) 형태

- 사례 A의 클러스터 형태 생성 알고리즘

사례 A에서의 클러스터 형태 생성 알고리즘은 페이지 차원의 각 항목을 10개씩 5개의 부분으로 나누고 각각의 구역 내에서 시작 항목과 마지막 항목을 선택하게 한다. 이에 반해 방문자 차원의 경우는 1000번째 이내의 방문자중 한명을 임의로 선택

선택해서 시작 항목으로 정하고 이 항목에서 5000명까지의 방문객들 중 마지막 방문객을 선택하여 클러스터된 희박성 형태를 생성해 낸다. 다음 변수들은 전체 알고리즘에 사용된 주요 변수들이다.

Visitor_Key : 사실 테이블에 입력할 방문자 키 값

Page_Key : 사실 테이블에 입력할 페이지 키 값

Time_Key : 사실 테이블에 입력할 시간 키 값

Visit_Cnt : 사실 테이블에 입력할 방문 횟수 값

Record_Cnt : 사실 테이블에 입력된 레코드의 수

Sparsity_Value : 전체 레코드수 * 10%

그림 5.4는 사례 A에서의 클러스터된 희박성 형태를 생성하는 알고리즘 이다.

```
Example_A_Cluster()
{
        while(Record_Cnt < Sparsity_Value){
                start_visitor = 방문자 차원 항목 중 시작 항목 ;
                start_page = 페이지 차원 항목 중 시작 항목 ;
                end_visitor = 방문자 차원 항목 중 마지막 항목 ;
                end_page = 페이지 차원 항목 중 마지막 항목 ;
                visitor_counter = start_visitor;
                page_counter = start_page;

                while(page_counter <= end_page){
                        while(visitor_counter <= end_visitor){

                                Record_Cnt 값 증가;
                                Visitor_Key생성;
```

```
                        Page_Key생성;
                        Visit_Cnt생성;
                        List에 Visitor_Key, Page_Key, Visit_Cnt저장;
                        If ( Record_Cnt >= Sparsity_Value)
                                전체 루프를 빠져 나간다.;

                        If ( Record_Cnt % 5000 == 0)
                                        List의 값들을 파일에 입력;
                        visitor_counter증가;
                        }
                        page_counter 증가;
                }
        }
        List의 값들을 파일에 입력 ;
}
```

그림 5. 4 사례A의 클러스터 형태 데이터 생성 알고리즘

● 사례 B의 클러스터 형태 생성 알고리즘

사례 B에서의 클러스터 형태 생성 알고리즘은 사례 A의 것과 비슷한 방법으로 이루어 진다. 시간 차원에서 날마다 1시부터 3시, 오후 6시부터 10시 까지 바쁜 시간대로 정하게 된다. 방문자 차원의 경우는 시간차원에서 정한 바쁜 시간대에 대해 각각 1000번째 이내의 방문자중 한명을 임의로 선택해서 시작 항목으로 정하고 이 항목에서 5000명 이내의 방문객들 중 한 명을 마지막 방문객으로 선택하여 클러스터된 희박성 형태를 생성해 낸다. 그림 5.5는 사례 B에서의 클러스터된 희박성 형태를 생성하는 알고리즘 이다.

```
Example_B_Cluster()
{
        day = 0; // 날짜가 지나가는 것을 카운트 한다.
        while(Record_Cnt < Sparsity_Value){

                start_visitor = 방문자 차원의 시작 항목;
                end_visitor = 방문자 차원의 마지막 항목 ;
                visitor_counter = start_visitor;

                while(visitor_counter <= end_visitor){
                        while(1~3시까지){

                                Record_Cnt 값 증가;
                                Visitor_Key생성;
                                Time_Key생성; // (day * 24) + 시간
                                Visit_Cnt생성;
                                List에 Visitor_Key, Time_Key, Visit_Cnt저장;
                                If ( Record_Cnt >= Sparsity_Value)
                                        전체 루프를 빠져 나간다.;
                                if ( Record_Cnt % 5000 == 0)
                                        List의 값들을 파일에 입력;
                        }
                        while(18~22시까지){

                                Record_Cnt 값 증가;
                                Visitor_Key생성;
                                Time_Key생성 ;// (day * 24) + 시간
                                Visit_Cnt생성;
```

```
                                List에 Visitor_Key, Time_Key, Visit_Cnt저장;
                                If ( Record_Cnt >= Sparsity_Value)
                                        전체 루프를 빠져 나간다.;
                                if ( Record_Cnt % 5000 == 0)
                                        List의 값들을 파일에 입력;
                        }
                        visitor_counter증가;
                }
                day 증가 ;// 하루가 지남
        }
        List의 값들을 파일에 입력;
}
```

그림 5. 5 사례B의 클러스터 형태 데이터 생성 알고리즘

- 사례 C의 클러스터 형태 생성 알고리즘

사례 C에서는 시간, 페이지, 방문자 세가지 차원 테이블을 모두 사용하여 테스트 데이터를 생성한다. 그림 5.6는 사례 C에서의 클러스터된 희박성 형태를 생성하는 알고리즘이다.

```
Example_C_Cluster()
{
        day = 0; // 매일매일을 카운트 한다.
        while( Record_Cnt < Sparsity_Value   && day < 90 ){
                offset_time = 성황을 이루는 시간 간격 ;
                start_page = 페이지 차원의 시작 항목;
                end_page = 페이지 차원의 마지막 항목;
                start_visitor = 방문자 차원의 시작 항목;
                end_visit = 방문자 차원의 마지막 항목;
```

```
                for(1시 ~ 1시 + offset_time){
                        for(start_page ~ end_page){
                                for(start_visitor ~ end_visitor){
                                        Record_Cnt 값 증가;
                                        Visitor_Key생성;
                                        Time_Key생성;// (day * 24) + 시간
                                        Page_Key생성;
                                        Visit_Cnt생성;
                List에 Visitor_Key, Time_Key, Page_Key, Visit_Cnt저장;
                                        If ( Record_Cnt >= Sparsity_Value)
                                                전체 루프를 빠져 나간다.;
                                        if ( Record_Cnt % 5000 == 0)
                                                 List의 값들을 파일에 입력;
                                        }
                                }
                        }
        day증가;
        }
        List의 값들을 파일에 입력;
}
```

그림 5. 6사례C의 클러스터 형태 데이터 생성 알고리즘

5.1.2.2 그리드(Grid) 형태

- 사례 A의 그리드 형태 생성 알고리즘

사례 A의 그리드 형태 생성 알고리즘은 페이지 차원의 항목 중 인기 있는 페이지를 선택해 이를 busy_page_list에 저장하고 이 페이지에 접속 하는 방문자들의 영역을 구해 하나의 페이지 항목에 대한 줄무늬 형태의 데이터를 생성한다. 이와 마찬가지로 방문자 차원의 경우 방문자 차원의 항목을 100개 단위로 나누고, 그 중 한 명

을 선택해 그 사람에 대한 페이지 항목 영역을 구한 후 방문자에 대한 줄무늬 형태의 데이터를 형성하고 서로 교차되는 부분은 busy_page_list를 체크해 필터링 시킨다. 이에 대한 자세한 알고리즘은 그림 5.7에서 설명한다.

```
Example_A_Grid()
{
        while(Record_Cnt < Sparsity_Value) {
                busy_page_list; // 정수형 포인터
                start_visitor = 방문자 차원의 시작 항목 ;
                end_visitor = 방문자 차원의 마지막 항목;
                busy_page = 인기 있는 페이지 항목 ;
                busy_page_list에 busy_page항목 값 저장;

                for(start_visitor ~ end_visitor){
                        Record_Cnt 값 증가;
                        Visitor_Key생성;
                        busy_page항목을 Page_Key로 확정;
                        Visit_Cnt생성;
                        List에 Visitor_Key, Page_Key, Visit_Cnt저장;

                        if(Record_Cnt >= Sparsity_Value)
                                전체 루프를 빠져 나간다.;
                        if((Record_Cnt) % 50000 == 0)
                                List의 값들을 파일에 입력;
                }}
                visitor_counter = 0;

        while(Record_Cnt < Sparsity_Value && visitor_counter <= 100) {
```

```
        visitor = 방문자 차원 중 단골 고객항목 ;
         // 100개의 항목으로 나누어 그 중 한명을 선택하게 한다

          visitor_counter 증가;
           iter = rand() % 3; // 최대한 하루에 2번까지 들어 옴
           for(0~iter)
           {
                start_page = 페이지 차원 중 시작 항목;
                end_page = 페이지 차원 중 마지막 항목;
                for(start_page ~ end_page){
                  if(해당 페이지가 busy_page_list와 교차하는가? == Yes){
                          break;
                   }
                  else {
                          Record_Cnt 값 증가;
                          visitor의 값을 Visitor_Key로 확정;
                          Page_Key생성;
                          Visit_Cnt생성;
                          List에 Visitor_Key, Page_Key, Visit_Cnt저장;

                          if(Record_Cnt >= sparsity_value)
                                  전체 루프를 빠져 나간다.;
                          if((Record_Cnt) % 50000 == 0)
                                  List의 값들을 파일에 입력;
                }
              }
           }
           List의 값들을 파일에 입력 ;
}
```

그림 5. 7사례 A의 그리드 형태 데이터 생성 알고리즘

● 사례 B의 그리드 형태 생성 알고리즘

사례 B의 그리드 형태 생성 알고리즘은 시간 차원의 항목 중 10,11그리고 18,19시를 바쁜 시간대로 정하고 이 시간대에 방문하는 방문객들의 집단을 정하여 이 시간대에 해당되는 줄 무늬 형태의 희박성 형태를 생성한다. 그리고 방문자 차원의 항목에 대해서도 바쁜 시간대로 설정한 시간대의 값과 중복 되지 않은 상태에서 시간차원의 항목 값과 같이 줄무늬 형태의 희박성 형태를 생성한다. 자세한 설명은 아래 그림 5.8에서 보여 준다.

```
Example_B_Grid()
{
      busy_visitor = 단골 방문객을 방문자 차원에서 하나의 항목으로 확정;
      day = 0으로 초기화;
       while(Record_Cnt < Sparsity_Value){

          first_class = 첫번째 시간대에 방문객의 수(10시);
          second_class = 두번째 시간대에 방문객의 수(18시);

          for(10~11시) // 오후 6~7시까지도 같은 과정을 반복{

                    for(0~first_class){

                         Record_Cnt 값 증가;
                         Visitor_Key생성 ;// busy_visitor를 사용하여 생성
                         Time_Key생성 ;// (day * 24) + 시간
                         Visit_Cnt생성;
                         List에 Visitor_Key, Time_Key, Visit_Cnt저장;
```

```
                    if ( Record_Cnt >= Sparsity_Value)
                            전체 루프를 빠져 나간다.;
                    if ( Record_Cnt % 5000 == 0)
                             List의 값들을 파일에 입력;
                }
            }
        start_time = 10시 이내의 시간항목을 선택;
        end_time = 10시 이후의 시간 항목 선택;
        first_class = 400번 이내의 방문자 항목 선택;
        second_class = 400번 이후의 방문자 항목 선택;

        for(start_time ~ end_time){
                if( (10~11시, 오후 6~7시 와의 교차부분)
                                continue;
                for(busy_visitor+first_class~busy_visitor+first_class+3){

// busy_visitor+second_class~busy_visitor+second_class+3도 같은 과정 반복
                Record_Cnt 값 증가;
                Visitor_Key생성 ;
                Time_Key생성 ;// (day * 24) + 시간
                Visit_Cnt생성;
                List에 Visitor_Key, Time_Key, Visit_Cnt저장;
                If ( Record_Cnt >= Sparsity_Value)
                        전체 루프를 빠져 나간다.;
                if ( Record_Cnt % 5000 == 0)
                         List의 값들을 파일에 입력;
                }}
        day증가;}
    List의 값들을 파일에 입력; }
```

그림 5. 8사례 B의 그리드 형태 데이터 생성 알고리즘

● 사례 C의 그리드 형태 생성 알고리즘

사례 C의 그리드 형태 생성 알고리즘은 자주 방문하는 방문객들을 일정 간격으로 생성하고 각 방문객들에 대한 페이지 차원과 시간 차원에 대한 2차원 그리드 형태의 희박성 형태를 생성 함으로써 3차원 큐브 내의 그리드 희박 형태를 만들어 낼 수 있다. 이에 대한 자세한 설면은 그림 5.9에 제시된 알고리즘을 참조한다.

```
Example_C_Grid()
{
    start_visitor = 방문자 차원 항목 중 시작 항목 선택;
    visitor_s = rand() % 100 ;

    start_time = 시간 항목 중 시작 항목 선택;
    end_time = 시간 항목 중 마지막 항목 선택;
    busy_time1 =  10시;
    busy_time2 =  18시;

    start_page = rand() % 10;
    end_page = 30 + rand() % 10;
    busy_page1 = 10;
    busy_page2 = 30;
    while(Record_Cnt < Sparsity_Value) {
        while(visitor_i < 방문차 차원 항목 전체 수) {
                for(day = 0; day < MONTH*DAY; day += 7) {
                        for(start_time ~ end_time) {
                                if(10시나 18시와 같으면) {
                                        for(start_page~end_page){
                        if(10번 페이지와 30번페이지와 중복되지 않으면){
                                Record_Cnt 값 증가;
                                Visitor_Key생성 ;
                                Time_Key생성 ;// (day * 24) + 시간
```

```
                Page_Key생성; //start_page~end_page값 사용
                Visit_Cnt생성;
        List에 Visitor_Key, Time_Key, Page_Key, Visit_Cnt저장 ;
        }

    if ( Record_Cnt >= Sparsity_Value)
                전체 루프를 빠져 나간다.;
    if ( Record_Cnt % 5000 == 0)
                List의 값들을 파일에 입력;
  }}
  else {
                Record_Cnt 값 증가;
                Visitor_Key생성 ;
                Time_Key생성 ;// (day * 24) + 시간
                Page_Key생성; //busy_page1,busy_page2값 사용
                Visit_Cnt생성;
        List에 Visitor_Key, Time_Key, Page_Key, Visit_Cnt저장;

                if ( Record_Cnt >= Sparsity_Value)
                        전체 루프를 빠져 나간다.;
                if ( Record_Cnt % 5000 == 0)
                        List의 값들을 파일에 입력;
}}} visitor_i+=visitor_s; }}
List의 값들을 파일에 입력; }
```

그림 5. 9 사례 C의 그리드 형태 데이터 생성 알고리즘

5.2 성능 평가 시스템 설계 및 구현

본 장에서는 앞에서 제시한 웹 로그 데이터의 희박성 형태의 데이터 모델에 의해 생성한 테스트 데이터를 사용하여 MS SQL 2000 Analysis Service , Oracle Express Server, C-MOLAP 3가지 시스템에 대해 다차원 질의를 수행하여 데이터 희박성에 따른 성능을 측정 할 수 있는 모듈에 대해 설명한다.

5.2.1 시스템 구현 환경 및 시스템 전체 구조도

본 논문에서 사용하고 있는 OLAP제품들의 성능 평가 프로그램의 구현환경은 아래 표 5.8와 같다.

표 5. 8 시스템 구현환경

운영 체제	Windows NT 4.0 Server
OLAP 서버	Oracle Express, MS SQL 2000 Analysis Service
개발 도구 및 언어	Visual Basic 6.0 MDAC (Microsoft Data Access Components) 2.5 SDK (ADO MD: ActiveX Data Objects Multidimensional), MDX (Multidimensional Expressions) OEO (Oracle Express Object) Express Basic, Express Language Visual C++ 6.0

MS SQL 2000 Analysis Service는 성능 평가 프로그램을 구현하기 위해 개발 언어로는 Visual Basic 6.0를 사용하였다. Visual Basic 프로그램내에서 수동적으로 이미 만들어져 있는 OLAP큐브에 접근 하기 위해 ADO MD API를 사용하고, MDX구문을

통해 질의를 수행한다[10]. Oracle Express Server의 경우는 Oracle Express Administrator에 의해 다차원 모델을 구성하고 Oracle Express Object시스템과 Visual Basic언어와 비슷한 Oracle Basic언어를 사용하여 성능평가 인터페이스를 구현하였다. 그리고 Oracle Language를 사용해 Oracle Express내의 큐브에 대한 접근 및 질의를 수행하였다[11, 12]. 마지막으로 C-MOLAP시스템의 경우 MOLAP시스템 구현에 사용하였던 언어 와 동일한 Visual C++6.0을 사용하여 성능 평가 프로그램을 구현하였다. 생성한 테스트 데이터를 로드해서 성능 평가를 하여 분석하는 전체 시스템 구조는 그림 5.10와 같다.

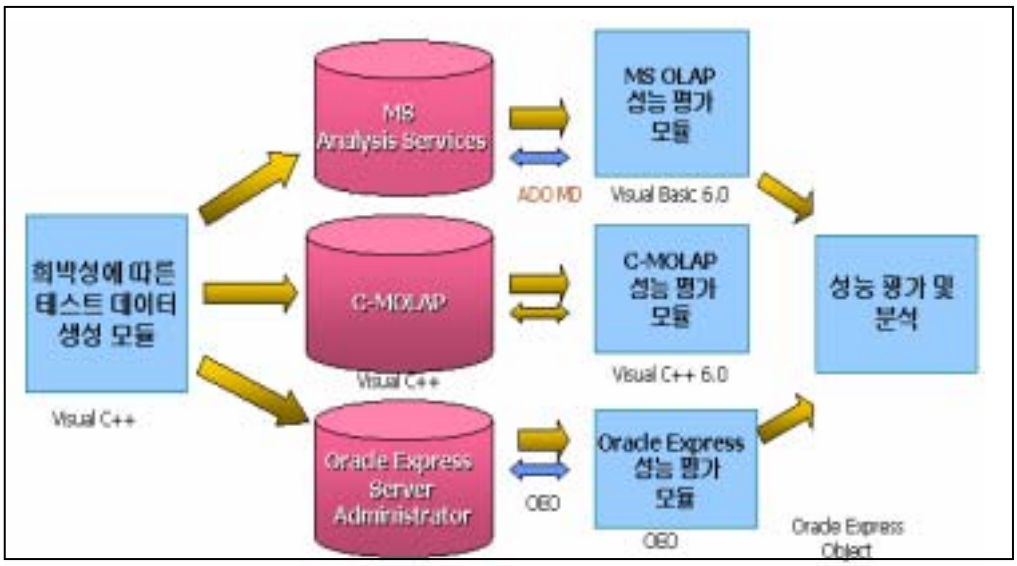

그림 5. 10 시스템 전체 구성도

앞에서 정의한 데이터 모델과 희박성 형태에 따라 생성된 데이터를 파일 형태로 저장하고, 각각 세가지 OLAP서버들 내에 로드 한다. MS SQL 2000 Analysis Service의 경우 MS SQL 2000내부의 DTS기능을 사용하여 각각 RDB형태로 저장하고 저장된 데이터들을 사용하여 Analysis Service내에서 2,3 차원에 대한 큐브를 정

의 한다. Oracle Express의 경우는 Oracle Express Administrator내의 Database 위저드를 사용하여 파일 형태의 데이터들을 큐브로 변환하여 저장한다. 세가지 OLAP시스템 내부에서는 2,3 차원 형태의 큐브들에 대해 사전 연산을 수행하고, 이들 각각의 큐브들에 대해 성능 평가 모듈을 통해 앞에서 정의한 쿼리 모델에 대한 수행시간을 측정한다. 마지막 과정으로 희박성 형태에 따른 쿼리들의 수행시간을 통해 3가지 시스템의 성능을 평가하고 분석한다.

5.2.2 성능 평가 프로그램

MS SQL 2000 Analysis Service의 경우 성능 측정 프로그램은 Visual Basic으로 구현하였다. 본 프로그램에서는 큐브를 생성하는 방법으로 수동적으로 Analysis Service내부의 위즈드를 사용하여 큐브 생성 및 사전 연산을 처리 하였다. 이와 같이 생성된 큐브에 질의를 수행하기 위해서 Microsoft MDAC SDK 2.5를 통해 제공하는 ADO MD API를 사용하였다. 그림 5.11은 MS SQL 2000 Analysis Service의 성능 평가 프로그램의 인터페이스 이다. 이 그림은 사례C의 그리드 형태의 희박성 형태를 지닌 데이터 모델에 대해 일반적인 7가지 OLAP 질의 중 Slice 연산을 수행한 수행 결과와 수행 시간을 보여준다.

두 번째 Oracle Express Server 또한 큐브 생성은 수동적으로 수행하고 이들 큐브에 대해 질의를 수행하기 위해 Express Language를 사용하였고, 성능 평가 인터페이스를 위해서 Oracle Express Object와 Express Basic언어를 사용하였다. 그림 5.12는 사례A의 클러스터 형태에 대해 Top500질의를 수행한 결과 테이블과 수행 시간을 보여준다.

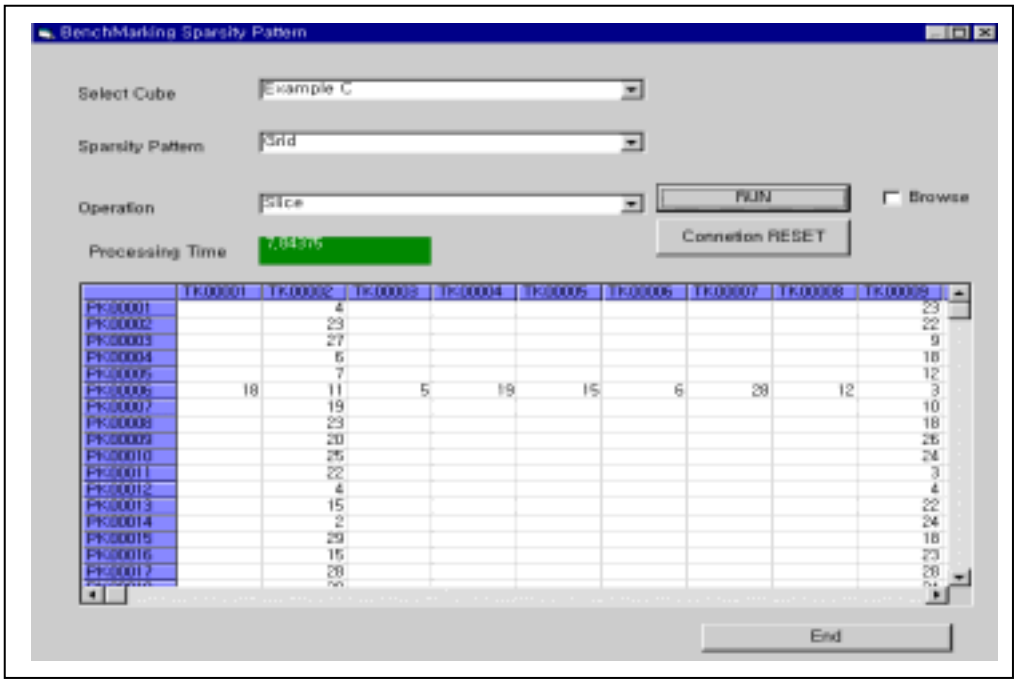

그림 5. 11 MS SQL 2000 Analysis Service 성능 평가 인터페이스

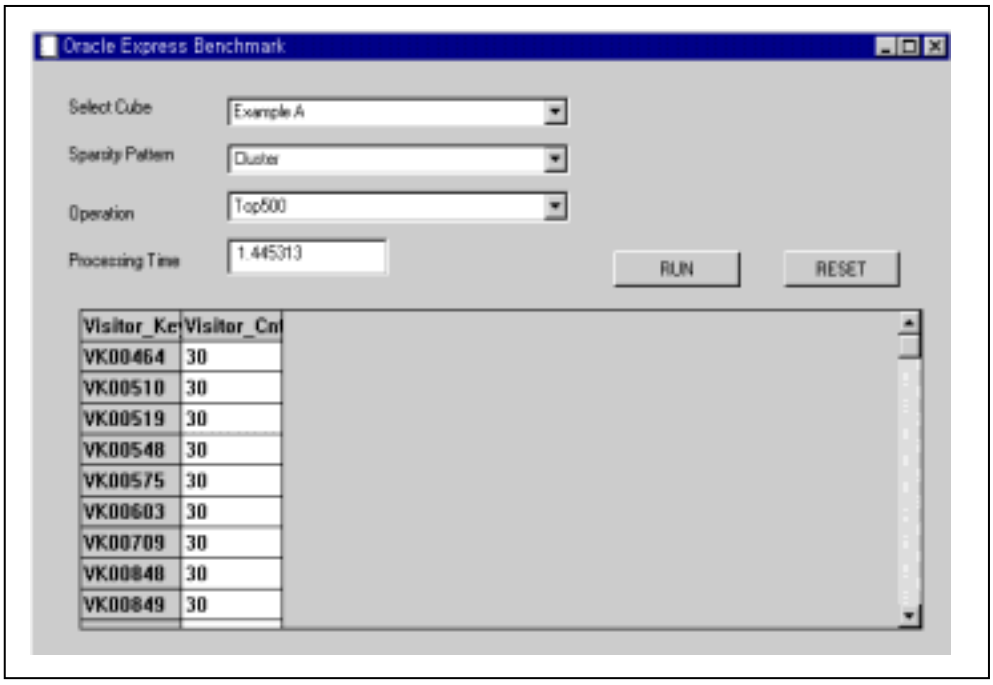

그림 5. 12 Oracle Express 성능 평가 인터페이스

그림 5.13는 C-MOLAP을 사용하여 클러스터 형태를 가진 사례 A에 대하여 Range질의를 수행 했을 때의 화면을 보여준다.

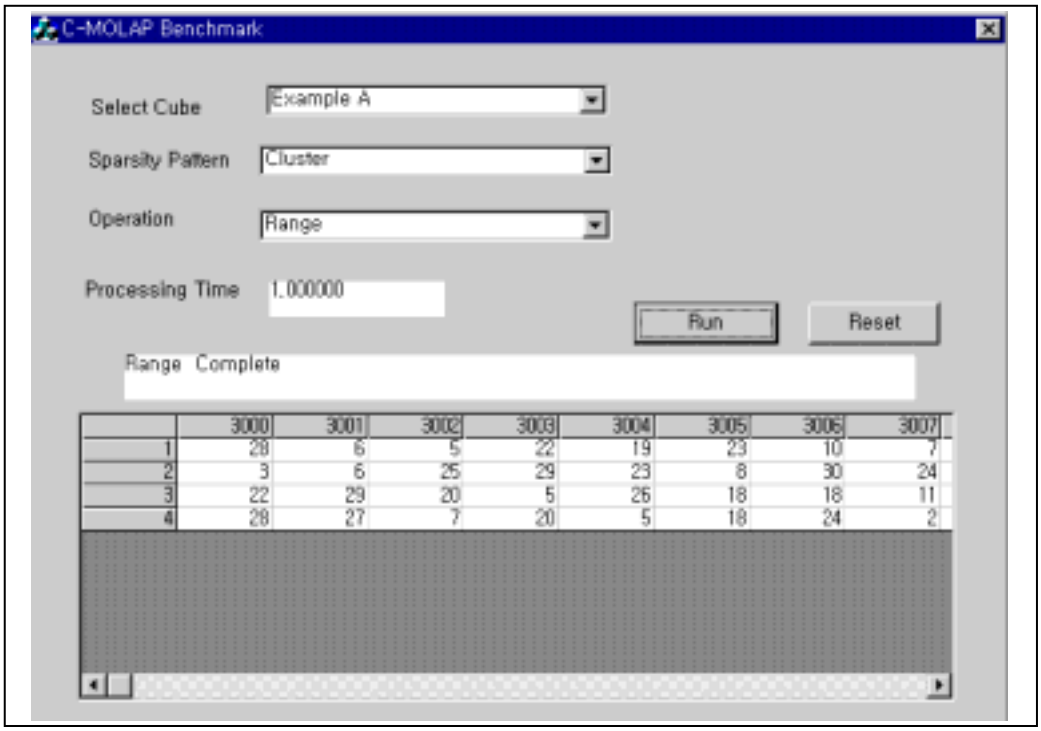

그림 5. 13 C-MOLAP 성능 평가 인터페이스

5.3 성능 평가 결과 및 분석

본 절에서는 5.2에서 구현한 성능 평가 프로그램을 통하여 얻어진 질의에 대한 수행 결과를 분석한다. 3가지 OLAP시스템에 대한 성능 평가를 하기 위해 앞에서 제시했던 질의 모델을 수행할 시 소요되는 시스템 시간을 측정 하였다. 각 시스템 시간은 초(Sec)단위로 측정하였고, 각 제품별 성능 측정치는 캐쉬에 의한 영향을 최소화하기 위해 처음 수행되어 나오는 수치들의 3회 수행을 그것의 평균을 사용하였다. 측정된 결과를 분석하기 위해 각각의 데이터 모델들에 대해 3가지 OLAP시스템과

희박성 형태에 따라 질의 결과를 보여 주었다.

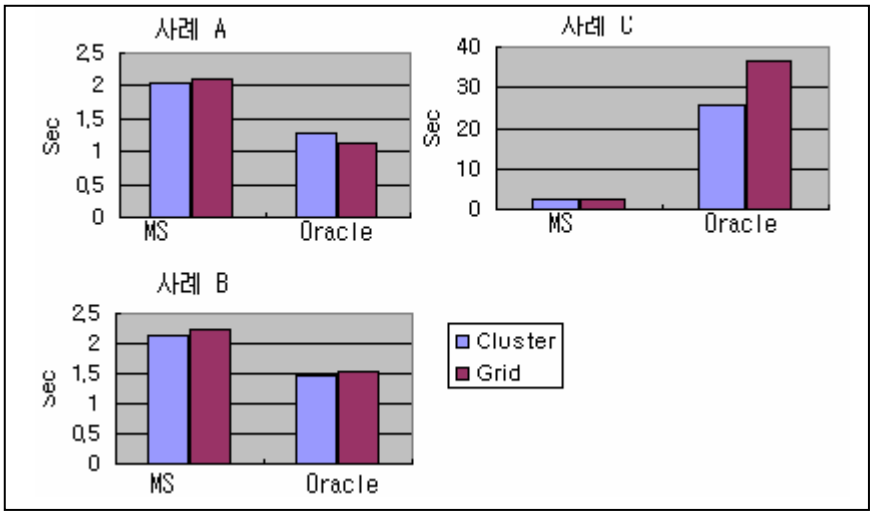

그림 5. 14 Top500 성능 평가 결과

그림 5.14은 Top500 질의를 수행한 결과이다. Top500 질의의 경우 OLAP질의 모델 중 랭킹 질의의 일종으로 정렬 알고리즘이 그 성능에 주요한 영향을 미치게 된다. 따라서 C-MOLAP시스템의 경우 이 질의에 대한 성능 평가에 있어서 제외 시키기로 한다. 위 그림에서 볼 수 있듯이 희박성 형태에 관계없이 사례 A나 B에서 처럼 차원의 수가 적고, 전체 큐브 내에 값들이 적을 경우 Oracle Express가 1초 이내의 빠른 결과를 보여 주지만 사례 C와 같이 차원의 수가 커지고 데이터 양이 많아 짐에 따라 MS SQL 2000 Analysis Service가 10배 이상의 향상된 성능을 보여준다.

다음에서는 OLAP의 일반적인 질의 7가지에 대해 성능 평가한 결과를 분석해 보도록 하겠다. 본 측정 결과에서는 3가지 시스템에서 성능 측정 결과가 2초 이내인 경우 큐브 사이즈와 희박성 형태에 따라 해당 질의에 대해서는 좋은 성능을 나타낸

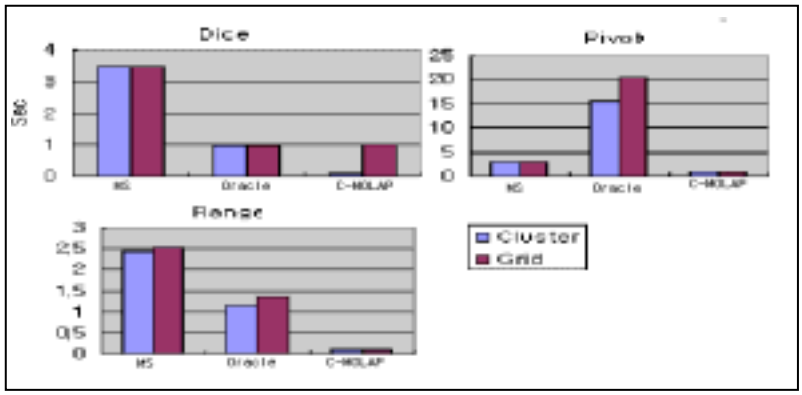

그림 5. 15 사례A의 성능 평가 결과

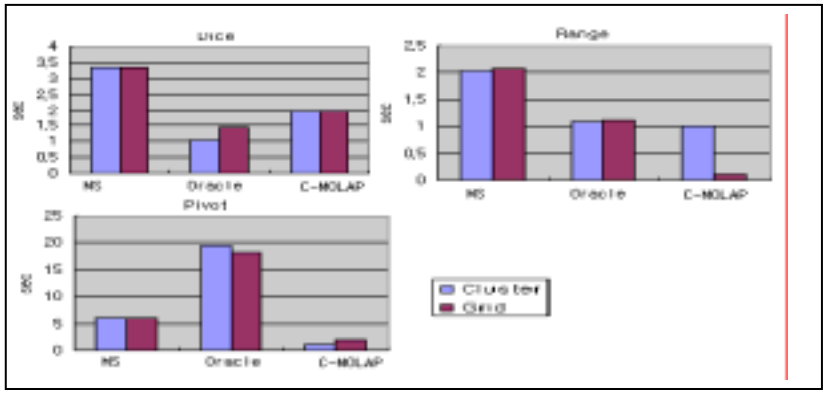

그림 5. 16 사례B의 성능 평가 결과

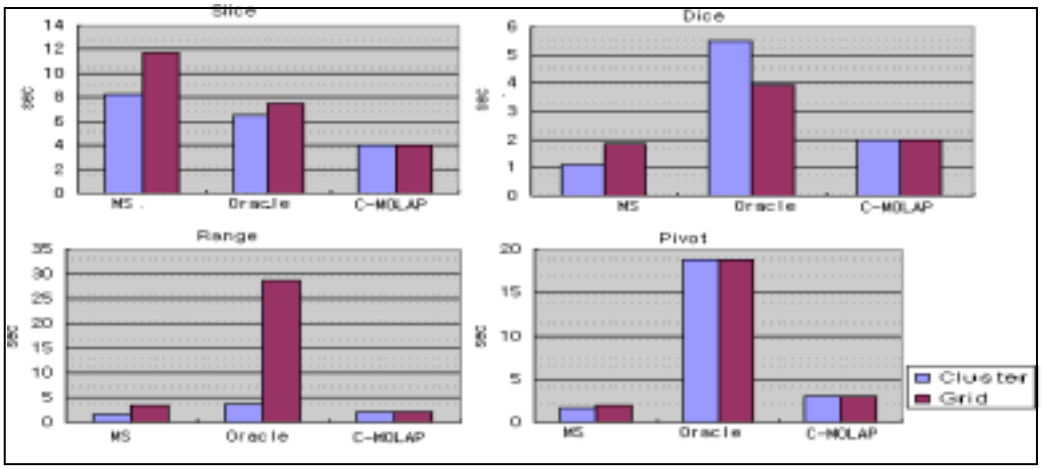

그림 5. 17 사례C의 성능 평가 결과

다고 가정하고, 그래프화 하지 않았다. 그림 5.15, 5.16, 5.17에서는 사례A, 사례B, 사례C의 성능 측정 결과를 보여준다. 위의 그림에서 알 수 있듯이 사례 A와 B에서 각 차원의 부분 집합의 조합에 의해 사용자 질의에 응답하는 Dice와 Range질의의 경우 MS SQL 2000 Analysis Service가 다른 두 가지 시스템에 비해 1~2초의 성능 저하를 보여 준다. 하지만 데이터 항목이 많고, 차원의 수가 하나 더 많아진 사례C의 경우 Oracle Express의 성능이 2~25초의 성능 저하를 보여준다. 특히 그리드 형태의 Range 질의에 있어서 Oracle Express의 성능은 크게 저하 됨을 볼 수 있다. 이에 반해 Pivot 연산에 있어서는 Oracle Express가 큐브 사이즈와 희박성 형태와 관계없이 가장 나쁜 성능을 보여줌을 확인할 수 있다.

결론적으로, 항목의 수와 차원의 수가 많아짐에 따라 희박성의 형태와 관계없이 랭킹 질의에서도 그리고 Dice와 Range Pivot 질의 에서도 Oracle Express의 성능이 다른 두 가지 시스템에 비해 현저하게 저하됨을 볼 수 있다.

Ⅵ. 결론 및 향후 과제

CRM의 목적은 각 기업이나 회사들이 새로운 고객을 유치하고, 기존의 고객들을 유지하면서 장기간의 이익 관계를 이어나가기 위해 고객의 취향과 필요를 이해하는 이다. CRM을 위해 웹 로그 데이터를 대상으로 OLAP을 적용할 시 사용자의 응답 성능을 빠르게 하기 위해 사전 연산은 반드시 이루어 져야 하는 과정이다. 하지만 이러한 과정에서 데이터의 희박성이 커지고, 차원의 수가 많아짐에 따라 OLAP의 성능은 저하되고, 데이터 폭발(Data Explosion) 현상이 발생한다.

본 논문에서는 웹 로그 데이터를 분석하는데 있어서 OLAP을 적용할 시 희박성이 발생되는 원인에 대해 알아 보고, 이들 원인 중 주요하다고 생각되는 요인에 대해 실제 로그 데이터를 사용하여 이를 확인해 보았다. 그리고 2,3차원내의 데이터 모델에서 그리드(Grid)와 클러스터(Cluster)형태의 희박성 형태가 주로 발생됨을 알 수 있었다. 분석된 희박성 형태를 기반으로 테스트 데이터 모델을 설계하고 이러한 데이터 모델에 따라 테스트 데이터 생성 모듈을 구현하여 데이터를 생성하였다. 7가지 OLAP의 일반적인 질의 모형과 Top500질의를 사용하여 질의 모델을 설계하고, 생성된 데이터와 함께 3가지 OLAP시스템(MS SQL 2000 Analysis Service, Oracle Express, C-MOLAP)에 대한 성능을 평가해 보았다. 성능 평가 결과 큐브의 사이즈가 적을 시에는 Top500연산, Dice연산, Range연산에 있어서 MS SQL 2000 Analysis Service의 성능이 다른 시스템들에 비해 약간의 성능 저하를 나타내지만, 차원의 수가 증가하고 항목의 수가 많아짐에 따라 3가지 연산에서 모두 Oracle Express의 성능이 현저 하게 저하됨을 볼 수 있었다. 또한 각 축

을 돌려가며 사용자의 보고싶은 관점에 따라 데이터들을 회전해서 볼 수 있는 Pivot연산에 있어서는 Oracle Express의 성능이 희박성 형태와 관계없이 그리고 큐브의 사이즈와 관계없이 가장 나쁜 성능을 보임을 알 수 있었다.

본 논문의 결과를 기반으로 다음과 같은 향후 연구 과제를 제시 할 수 있다,

- 실제 로그 데이터와 유사한 로그 데이터의 생성 모듈의 개발은 웹 로그 데이터를 사용하여 성능 평가를 하거나 Data Mining알고리즘에 적용하여 CRM을 하기 위한 작업의 기초가 될 수 있다.
- C-MOLAP시스템에 있어서 효율적인 청크의 사이즈를 결정 할 수 있는 알고리즘의 구현을 향후 과제로 제시 할 수 있다. 청크 사이즈가 너무 작게 나누어져 있을 시에는 각 청크를 가져오는데 디스크 I/O 타임이 많아지고, 순차 검색을 하는 것과 별반 다를 것이 없는 결과를 가져 오게 된다. 이에 반해 청크 사이즈가 너무 클 시에는 하나의 디스크 페이지 내에 불필요한 여과 용량을 낭비 할 수 있다.
- 본 논문에서 분석한 웹 로그 데이터의 희박성 형태와 성능 평가 결과를 기반으로 이들을 저장하고, 처리하기 위한 일반화되고 효율적인 웹 로그 데이터 분석기의 개발을 향후 연구 과제로 제시 한다.

참고 문헌

[1] MaxScan Corp, White Paper. “An Accelerator for Click Stream Data Analysis Applications”

[2] *White Paper*, http://www.olapreport.com/DatabaseExplosion.htm

[3] Michael L. Gonzales, “Estimating the Explosion of Derived Cells”, DB2 magazine, pp 14-17, 2000.

[4] Sanjay Goil and Alok Choudhary, “ Sparse Data Storage of Multi-Dimensional Data for OLAP and Data Mining”, Technical Report CPDC-TR-9801-005, Center for Parallel and Distributed Computing, Northwestern University, 1997.

[5] Robert Cooley, Bamshad Mobasher, Jaideep Srivastava, “Data Preparation for mining world wide web browsing patterns”, the Journal of Knowledge and Information System, Vol. 1, No. 1, 1999

[6] L. Catledge and J. Pitkow. “Characterizing browsing behaviors on the World Wide Web”, Computer Networks and ISDN Systems, 1995

[7] 조재희, 박성진, *SIGMA Consulting Group*, “데이터 웨어하우스의 효과적 활용 기법 OLAP 테크놀러지”, 1999.

[8] OLAP Council, “APB-1 OLAP Benchmark Release II”, Dec 1998. http://www.olapcouncil.org/research/bmarkly.htm

[9] Y. Zhao, P. M. Deshpande, and J. F. Naughton, “An Array-Based Algorithm for Simultaneous Multidimensional Aggregates”, In Proc. of ACM SIGMOD, pp 159-170, 1997

[10] MDAC(Microsoft Data Access Component)Documentation, http://www.microsoft.com/data/whatcom.htm

[11] Oracle Corp, “Oracle Express Objects User’s Guide Release 6.3”, Aug 1999

[12] Oracle Corp, “ Oracle Express Programmer’s Guide to the Express Language Release 6.3”, Sep 1999

[13] Microsoft Corp, “Microsoft Corp. SQL Server 2000 Analysis Services”, Nov 2000

[14] Ju-young Kang, “Classification of sparsity patterns and performance evaluation in OLAP system” , the Master’s Thesis of Department of Computer Science and Engineering, Ewha Institute of Science and Technology, 2000

[15] Sanjay Goil and Alok Choudhary, “ Sparse Data Storage of Multi-Dimensional Data for OLAP and Data Mining, *Technical Report CPDC-TR-9801-005*, Center for Parallel and Distributed Computing, Northwestern University, 1997.

[16] Robert J. Earle. Arbor Software corporation, “Method and Apparatus for Storing and Retrieving Multi-dimensional Data in Computer Memory”, *U.S. patent* #5359724, Oct. 1994.

[17] Oracle Corp., “Sparsity Management System for Multi-dimensional Databases”, *U.S. patent* #5943677, Aug, 1999.

ABSTRACT

Web Log Data Sparsity Analysis and Performance Evaluation for OLAP

Department of Computer Science & Engineering

Ewha Institute of Science and Technology

Ji Hyun Kim

Recently in competitive business environments, IT for CRM has been growing and developed rapidly. Typical applications are statistical analysis tools, on-line multidimensional analytical processing (OLAP) tools, and data mining algorithms (such neural networks, decision trees, and association rules). Business data that these applications analyze include customer data, payroll data, inventory data, supplier data and competitive data, etc. Specially, web click stream data among customer data can be easily obtained. But because of a tremendous of the customer click stream data, enterprises are laboring to answer even basic business questions, such as "which page is most popular". To use these data efficiently, they must set up the online analytical processing (OLAP) cube. It has to precalculate multidimensional summary results in order to get fast response. But as the number of dimensions and sparse cells

increases, *data explosion* occurs seriously and the performance of OLAP decreases.

In this paper, we presented why the web log data sparsity occurs and then what kinds of sparsity patterns generate in the two and the three dimensions for OLAP. Based on these research, we set up the multidimensional data models and query model for beanchmark with each sparsity patterns. Finally, we evaluated the performance of three OLAP systems (MS SQL 2000 Analysis Service, Oracle Express and C-MOLAP).

감사의 글

우선 무엇보다도 2년 동안의 대학원 과정을 지내며 이 논문이 나올 때 까지 저를 지켜봐 주시고 힘들 땐 용기를 주셨던 저의 지도 교수님께 감사의 마음을 전하고 싶습니다.

그리고 같은 논문학기로 힘든 시기를 서로 격려하며 마음으로 서로를 위해 주었던 우리 동기 현숙언니, 현정이, 은주언니 비록 짧은 시간이었지만 아마 앞으로도 서로에게 좋은 조언자가 되고, 힘이 되어 줄 것이라 생각합니다. 그리고 연구실 식구들 일본에 있지만 나의 논문에 결정적으로 영향을 끼쳤던 그리고 선배로서 항상 좋은 조언을 해주었던 주영이, 우리 방 왕 언니 이자 열심히 사는 모습을 직접 보여 주시며 모범이 되시는 호숙언니, 지금은 졸업하고 사회인으로서 나의 정신적 지주가 되어주는 인기, 은정언니, 성은언니, 후배지만 후배 같지 않게 성숙한 보영이, 그리고 겉으론 유하면서 안으론 강한 우리 2학기들 항상 윗사람들에게 깍듯하고, 자기들끼리는 우애하며 후배지만 우리에게 모범이 되어주는 귀엽고 이쁜 후배들 모두모두 너무 감사하고 대학원 생활의 많은 추억과 기억을 남겨주어 고맙다는 마음을 전합니다.

나의 추억이자 나의 든든한 보배 내 친구들 은희, 지영*2, 소영, 은미*2, 덕효, 문주, 혜성,미진, 은경, 혜진, 미성, 선아, 상요 뒤에서 말없이 내가 외롭고 힘들 때 힘이 되 주어 항상 감사합니다.

마지막으로 이름만 불러도 눈물이 먼저 나는 우리 부모님. 날 믿어주시고 항상 사랑으로서 날 감싸주셔서 지금 이 자리까지 온전한 정신으로 바르게 자랄 수 있도록 도와주셔서 감사합니다. 귀여운 동생들도..... 그리고 우리 오빠도 ....