그리디 알고리즘에 의한 클래스 인덱싱 기법
(A Class Indexing Technique using Greedy Algorithm)

안 정 호 † 송 하 주 †† 김 형 주 †††
(Jung-Ho Ahn) (Ha-Joo Song) (Hyoung-Joo Kim)

요 약 객체 지향 데이타베이스 시스템은 기존 관계형 데이타베이스 시스템과는 달리 클래스 및 클래스 계층구조 개념을 제공한다. 따라서 질의 또한 특정 클래스 뿐만 아니라 해당 클래스의 하위 클래스를 포함하는 클래스 계층구조 상에서 이루어진다. 이를 위해 클래스 계층 인덱스 기법을 비롯하여 여러 클래스 인덱싱 기법이 제안되었으나 성능과 실용성에 문제가 있으며, 다차원 인덱스 기법을 사용하는 것 또한 효율성이 떨어지는 단점이 있다

본 논문은 클래스 계층구조 상에서의 질의를 효과적으로 지원하기 위한 인덱스 기법으로서 인덱스 집합 기법을 제안한다. 인덱스 집합 기법은 그리디 알고리즘을 통해 클래스 계층구조상의 여러 클래스들을 주어진 제한 조건내에서 최소의 검색 비용으로서 질의를 처리할 수 있도록 클래스들의 그룹으로 구분한 뒤, 각 그룹에 대해 B^+-트리 인덱스를 각각 할당하는 방법이다. 또한 본 논문에서는 클래스 내의 인스턴스의 수, 키 값의 분포 형태, 질의 형태에 따라 인덱스를 구성할 저장 공간 및 인덱스 검색에 따르는 비용을 예측하기 위한 비용 모델을 제시한다. 본 인덱스 집합 기법은 B^+-트리의 구조에 큰 변형을 가하지 않고 그대로 사용할 수 있기 때문에 실용적인 장점도 함께 가지고 있다.

Abstract Object-oriented database systems (OODBMS) are based on object and class concept, and thus, OODBMS should provide class hierarchy query which can probe every instance of a class and its sub-classes. Several class indexing schemes including class-hierarchy index have been proposed to support this kind of queries. They, however, suffer from poor performance and lack of practicability. In addition, multi-key indexing schemes are not suitable since they are too general.

In this paper, we propose a class indexing scheme, Index Set which allocates B^+-tree indices to groups of classes by the greedy algorithm. We also introduce a cost model which can predict the space and retrieval cost of index based on the number of instances, the distribution of key values in each class, and query pattern on the class hierarchy. Our scheme has a good practical advantage that it can be easily applied to a system since it uses the well-proven B^+-tree structure.

1. 서 론

1.1 OODB 인덱싱의 문제점

데이타 모델의 성공은 이를 얼마나 효율적으로 지원할 수 있느냐에달려 있다. 또한 효과적인 인덱싱이 데이타베이스 시스템의 성능에 매우 결정적인 역할을 한다는 것도 널리 알려져 있는 사실이다. 예를 들면 관계형 데이타 모델은 B^+-트리 구조를 사용해서 선언적 질의의 수행을 효율적으로 지원할 수 있었기에 지금의 성공을 얻을 수 있었다.

그러나 기존의 B^+-트리 구조는 일차원 공간에서의 질의에는 최적의 성능을 보장하나 객체 지향 (또는 객체 관계형) 데이타 모델에서는 그 한계를 가지고 있다. 즉, 클래스 계층 구조상에서의 질의는 검색 조건으로 객체 속성의 값 이외에 객체가 속해야 하는 클래스 범위가 함께 주어지는 이차원 공간에서의 질의이기 때문에 기존의 B^+-트리 구조로는 효과적으로 대응할 수 없다.

· 본 연구는 과학기술부의 특정연구개발과제 사업의 일환인 "객체지향기술을 이용한 인터넷상의 트랜잭션 처리 기술개발"의 지원에 의한 것임
† 정 회 원 (주)삼성전자 연구원
jhahn@oopsla.snu.ac.kr
†† 학생회원 : 서울대학교 컴퓨터공학과
hjsong@oopsla.snu ac.kr
††† 종신회원 : 서울대학교 컴퓨터공학과
hjk@oopsla snu.ac.kr
논문접수 : 1998년 3월 17일
심사완료 : 1998년 11월 17일

이러한 클래스 계층 구조상에서의 질의 성능을 향상시키기 위해서 지금까지 H-트리[1]나 HcC-트리[2] 등 여러 새로운 구조가 제안되었다. 그러나 이러한 새로운 인덱스 구조는 동시성 제어 등의 문제로 실제 사용하는데 어려움이 있어 거의 사용되지 못하고 있다. 한편 클래스 분할(class division)[3] 방법은 B^+-트리 인덱스를 여러 개 중복 사용함으로써 검색 성능을 향상시키는 방법으로 쉽게 적용할 수 있다는 장점을 가지고 있다. 그러나 이 방법은 각 클래스에 속한 인스턴스들의 수나 키 값의 분포를 전혀 반영하지 못하고 있어 질의 성능의 향상을 보장하지 않는다. 이는 H-트리나 HcC-트리에서도 마찬가지이다.

본 논문에서는 이러한 문제점들을 해결하고 다양한 질의에 대해서 주어진 공간 및 갱신 오버헤드의 제약내에서 최대한의 질의 성능 향상을 얻기 위한 방법을 제안하고자 한다.

1.2 관련 연구

지금까지 객체지향 데이타베이스를 위한 인덱스 기법에 관한 많은 연구가 진행되어 왔다[1, 2, 3, 4, 5, 6, 7, 8]. 먼저 클래스 계층 구조 전체를 대상으로 하나의 인덱스를 유지하는 클래스 계층 인덱스(class-hierarchy index)[4]는 클래스 계층 구조의 많은 부분에 대해서 질의가 이루어지는 경우 각 클래스에 대해 인덱스를 각각 유지하고 이를 검색하는 것보다 효율적이라는 점에 기반하고 있다. 또한 클래스 계층 인덱스는 기존의 B^+-트리 구조를 그대로 사용할 수 있기 때문에 구현이 쉽다는 장점을 가지고 있다. 그러나 하위 클래스에 대한 검색의 경우 불필요한 검색 결과가 성능에 상당한 영향을 미친다.

한편 H-트리[1]나 HcC-트리[2], CG-트리[5]는 트리 구조상에서 링크나 체인을 사용함으로써 클래스 계층 인덱스의 단점을 보완하고자 하였다. 그러나 H-트리의 경우, 외부 인덱스와 내부 인덱스의 키 값의 범위가 맞지 않을 때는 내부 인덱스에 대해서 모두 검색을 해야 하며 HcC-트리 역시 클래스 계층 구조 중 일부에 대한 검색은 각각의 클래스 체인을 따라서 이루어지기 때문에 검색 비용이 증가한다. 또한 CG-트리의 경우 상속의 특성을 전혀 고려하지 않았다. 이렇듯 이들 방안은 전반적인 성능의 향상을 보장하지 못하며, 더욱이 복잡한 구조로 인하여 동시성 제어의 비용이 증가한다. 때문에 지금까지는 기존의 B^+-트리 구조를 그대로 사용할 수 있는 클래스 계층 인덱스만이 실제로 이용되어 왔다[3]. 그러므로 현실적으로 가장 타당한 대안은 일차원 검색에서 최적의 성능을 보장하며, 쉽게 적용이 가능한 B^+-트리를 중복 사용함으로써 질의 검색의 성능을 향상시키는 것이다.

클래스 분할에 의한 인덱싱 기법[3]은 이러한 문제점을 해결하기 위해서 제안된 클래스 계층 인덱스의 변형된 형태이다. 클래스 분할 기법은 클래스 계층 구조를 해당 알고리즘에 따라 여러 개의 클래스 집합으로 나누고, 각 클래스 집합에 대해서 인덱스를 각각 생성한다. 이때 각 클래스는 여러 클래스 집합에 중복되어 포함된다. 즉, 클래스 분할 기법은 이러한 인덱싱의 중복을 통해 질의 성능의 향상을 얻음과 동시에 클래스 계층 인덱스 구조를 바로 사용할 수 있다는 장점을 가진다. 그러나 클래스 분할 알고리즘은 다중 상속을 지원하는 데이타 모델에는 적용할 수 없으며 중복도에 대한 제어를 할 수 없다. 또한 앞서 언급한 바와 같이 단순히 클래스 계층 구조만을 가지고 정해진 규칙에 따라 클래스들을 분할하기 때문에 각 클래스에 속한 인스턴스들의 수나 키 값의 분포는 전혀 고려하지 못한다.

이에 본 논문에서 제안하고 있는 클래스 인덱싱 기법은 클래스 분할 알고리즘의 문제점을 보완하여 좀더 작은 중복 비용을 통해서 더 나은 검색 성능을 얻기 위한 방법이다.

1.3 논문의 구성

본 논문의 구성은 다음과 같다. 먼저 제 2절에서는 클래스 계층 구조상에서의 질의를 위한 새로운 인덱싱 기법을 제시하고, 제 3절에서는 본 연구에서 사용한 비용 모델을 기술하였다. 그리고 제 4절에서는 새로운 인덱싱 기법에 대한 성능 평가를 실시하였으며, 제 5절에서는 앞으로의 연구 방안을 제시하였다.

2. 인덱스 집합 기법

클래스 계층 구조상에서의 질의는 질의의 대상이 되는 클래스에 속한 객체 중에서 주어진 조건을 만족하는 객체를 찾는 이차원적 검색이다.

예제 1 그림 1과 같은 클래스 계층 구조상에서 "나이가 21살인 학생을 검색하시오"라는 질의는 '나이가 21살'이라는 조건 외에 **Student** 클래스의 멤버라는 조건을 함께 포함하고 있다. 이때 클래스 계층 구조상의 *IS-A* 관계에 의해 **TA** 클래스의 멤버 역시 **Student** 클래스의 멤버라 할 수 있기 때문에 **Student** 클래스를 포함하여 **Student** 클래스의 모든 하위 클래스 역시 질의의 대상이 된다.

이러한 클래스 계층 구조상에서의 질의 성능을 향상시키기 위해서는 클래스 계층 구조를 고려한 인덱싱 기법이 필요하다. 그러나 지금까지 제안된 클래스 인덱싱을

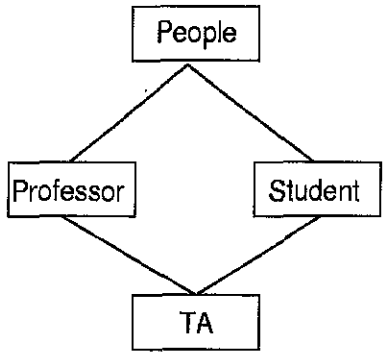


그림 1 People 클래스 계층 구조

위한 새로운 구조들은 전반적인 성능의 향상을 보장하지 못하며, R-트리 구조[9]와 같은 일반적인 다중 키 인덱스 구조 역시 정적인 클래스 도메인을 한 축으로 하는 클래스 인덱싱에서는 좋은 성능을 보이지 못한다[3]. 그러므로 현실적으로 가장 타당한 대안은 일차원 검색에서 최적의 성능을 보장하며, 쉽게 적용이 가능한 B^+-트리를 중복 사용함으로써 질의 검색의 성능을 향상시키는 것이다.

우선 검색 성능만을 고려하였을 때 가장 최적의 인덱스 구성 방법은 각 클래스의 전 익스텐트(full extent)에 대해 각각 인덱스를 구성하는 것이다. 여기서 전 익스텐트란 해당 클래스를 비롯하여 모든 하위 클래스의 인스턴스들의 집합을 말한다. 따라서 예제 1에서 최적의 인덱스 구성은 클래스 People, Professor, Student, TA에 대해서 각각의 전 익스텐트에 대한 인덱스를 구성하는 것이다.

그러나 이러한 방법에는 저장 비용 및 갱신 비용의 오버헤드가 따른다. 즉, 예제 1에서 TA의 인스턴스들에 대해서는 TA 클래스에 대한 인덱스 뿐만 아니라 People, Professor, Student 등 모든 인덱스에 중복하여 인덱싱을 해야 한다.

그러므로 결국 인덱스의 구성 문제는 어떻게 인덱스의 중복을 최소화하면서 검색 성능을 향상시킬 수 있는 가이다. 즉, 공간 비용과 시간 비용의 균형(trade-off) 문제이다. 그러나 최적의 균형점은 동작 환경에 매우 의존적이기 때문에 특정 값을 일률적으로 적용하는 것은 무리가 있다.

이에 우리는 인덱스의 중복도를 변수로 보고, 주어진 중복도 내에서 최적의 인덱스 구성을 찾는 방안인 인덱스 집합 기법을 제시하겠다.

예제 2 클래스 C_1, C_2, C_3에 대한 인덱스 구성에서 생성할 수 있는 모든 인덱스는

$$M=\{\{C_1C_2C_3\}\{C_1C_2\}\{C_1C_3\}\{C_2C_3\}\{C_1\}\{C_2\}\{C_3\}\}$$

과 같다. 여기서 $\{C_i, C_j, \ldots\}$은 클래스 $C_i, C_j, \ldots$의 인스턴스들에 대한 인덱스를 의미한다. 이때 우리의 문제는 결국 집합 M의 부분 집합 중 중복도 R을 만족시키면서 질의 비용을 최소화하는 인덱스 집합 O를 찾는 것이다.

그러나 n개 클래스에 대한 모든 가능한 인덱스 구성 방법의 수는 $2^{2^n-1}-1$로, 이 조합을 모두 다 조사하는 것은 불가능하다. 따라서 우리는 최적의 인덱스 집합을 찾기 위한 그리디 알고리즘(greedy algorithm)을 고안하였다.

2.1 그리디 알고리즘

이번 절에서는 먼저 중복을 허용하지 않았을 때의 그리디 알고리즘을 제시하고, 이를 확장하여 중복을 고려한 알고리즘을 보이겠다.

상위 클래스에 대한 질의 비율이 높거나 이 클래스에 속한 인스턴스의 수가 하위 클래스에 속한 인스턴스의 개수에 비해 상대적으로 적은 경우에는 이 두 클래스의 인스턴스들에 대해서 하나의 인덱스를 구성하는 것이 각각 인덱스를 유지하는 것보다 전반적인 검색 성능 향상에 도움이 된다. 그러나 반대의 경우에는 인덱스를 각각 유지하는 것이 오히려 성능 향상에 이득이 된다. 이렇듯 클래스들에 대해서 인덱스를 각각 구성하는 경우와 하나의 인덱스로써 질의를 처리하는 경우에 질의 성능은 질의 비율이나 인스턴스의 수, 키 값의 분포 등에 따라서 달라진다. 본 그리디 알고리즘은 각 구성에 대한 질의 비용을 계산하고 질의 성능에 대한 이득을 최대한 얻을 수 있도록 단계적으로 인덱스 구성을 변경해나가는 것을 그 기본 골격으로 하고 있다

즉, 먼저 각 클래스의 인스턴스들에 대한 단일 인덱스들로 - 전 익스텐트에 대한 인덱스가 아니라 - 이루어진 초기 인덱스 집합을 구성한 후, 이 인덱스 집합에 속한 멤버 인덱스들을 합병했을 때 성능 향상을 계산한다. 여기서 합병이란 각 인덱스가 유지하던 인스턴스들을 하나의 인덱스로 구성하는 것을 말한다. 이렇게 계산한 성능 향상 값 중 가장 좋은 성능 향상을 보이는 인덱스 합병을 선택하고 이를 초기 인덱스 집합에 반영하여 새로운 인덱스 집합을 구성한다. 같은 방법으로 더 이상 성능 향상이 없을 때까지 반복해서 인덱스를 합병해 나간다. 그 결과가 중복을 허용하지 않았을 때의 그리디 방법에 의한 최적의 인덱스 집합(greedy optimal index set)이다.

자세한 그리디 알고리즘은 알고리즘 1에 기술하였다.

알고리즘 1 그리디 알고리즘

$O_{new} := \{\{C_1\}\{C_2\}...\{C_n\}\}$
repeat
  $O_{old} := O_{new}$
  for all I,J such that $I,J \in O_{old}$ and $I \neq J$ **do**
    $O := (O_{old} \cup \{I \cup J\}) - \{I\ J\}$
    if 질의비용$_O$ < 질의비용$_{O_{new}}$ **then**
      $O_{new} := O$
    end if
  end for
until $O_{old} = O_{new}$

O_{new}가 중복을 허용하지 않았을 때의 그리디 방법에 의한 최적의 인덱스 집합이다.

예제 3 앞서 설명한 그리디 알고리즘에 따라서 그림 2의 클래스 계층 구조에 대한 최적의 인덱스 집합을 찾는 과정은 다음과 같다. 그림에서 괄호 안의 수는 각 클래스에 속한 인스턴스의 수이다.

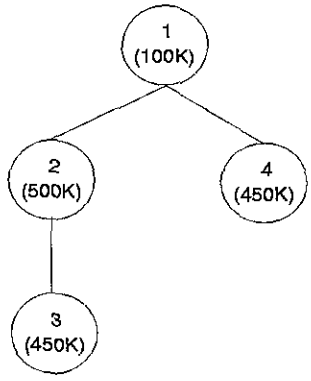


그림 2 예제 클래스 계층 구조

제 1 루프: 먼저 초기 인덱스 집합 $O = \{\{1\}\{2\}\{3\}\{4\}\}$에서 시작하여 다음과 같이 두 멤버 인덱스를 합병했을 때의 질의 비용을 구한다. 인덱스 집합의 질의 비용에 대해서는 3절에서 자세히 설명하겠다.

$O = \{\{1\}\{2\}\{3\}\{4\}\}$: 1.2944
$O_1 = \{\{1\ 2\}\{3\}\{4\}\}$: 1.2782
$O_2 = \{\{1\ 3\}\{2\}\{4\}\}$: 1.3291
$O_3 = \{\{1\ 4\}\{2\}\{3\}\}$: 1.3043
$O_4 = \{\{1\}\{2\ 3\}\{4\}\}$: 1.2549
$O_5 = \{\{1\}\{2\ 4\}\{3\}\}$: 1.4204
$O_6 = \{\{1\}\{2\}\{3\ 4\}\}$: 1.5375

여기서 가장 좋은 질의 성능을 보인 인덱스 집합 O_4를 새로운 인덱스 집합으로 선택한다.

제 2 루프: 앞 단계에서 선택한 O_4에 대해서 같은 방식으로 다음 인덱스 집합을 찾는다.

$O = \{\{1\}\{2\ 3\}\{4\}\}$: 1.2549
$O_1 = \{\{1\ 2\ 3\}\{4\}\}$: 1.2527
$O_2 = \{\{1\ 4\}\{2\ 3\}\}$: 1.2647
$O_3 = \{\{1\}\{2\ 3\ 4\}\}$: 1.5786

여기서는 O_1을 선택한다.

제 3 루프: 같은 방식으로 가능한 합병에 대해서 조사하면,

$O = \{\{1\ 2\ 3\}\{4\}\}$: 1.2527
$O_1 = \{\{1\ 2\ 3\ 4\}\}$: 1.5985

와 같다. 그러나 더 이상의 질의 성능을 향상시킬 수 없기 때문에 $O = \{\{1\ 2\ 3\}\{4\}\}$를 최종 인덱스 집합으로 한다.

그러나 이미 언급한 바와 같이 중복을 고려하지 않고서는 질의 검색성능의 향상에는 한계가 있다. 인덱스의 중복을 허용하기 위해서 앞서 제시한 그리디 알고리즘을 확장하면 알고리즘 2와 같다. 먼저 각 루프에서 두 인덱스의 합병에 따른 성능 향상을 계산할 때 중복을 고려해야 한다. 즉, 두 인덱스가 합병된 후에도 이전의 인덱스를 유지하는 경우를 함께 고려해야 하는데, 알고리즘 2에서 가장 내부의 **for all** 루프가 여기에 해당한다. 이 루프에서 인덱스 합병시 이전의 인덱스를 모두 유지하는 경우, 하나의 인덱스만을 유지하는 경우, 모두 유지하지 않는 경우 등 세 가지 경우를 모두 고려한다.

또한 주어진 중복도를 만족하는 범위 내에서 각 클래스에 대한 단일 인덱스를 추가하는데, 이는 두 인덱스의 합병시 두 인덱스의 교집합에 해당하는 클래스들에 대해서 다시 단일 인덱스를 구성함으로써 성능 향상을 얻을 가능성과 다음 루프에서 합병에 기여할 수 있는 가능성을 위해서이다.

한편 중복을 허용한 그리디 알고리즘의 결과에는 불필요한 인덱스가 있을 수 있다. 이는 합병시 이전의 인덱스를 유지하기 때문이다. 즉, 같은 성능 향상을 얻을 수 있다면 다음 단계에서 새로운 합병을 유도하기 위해서 가능한 한 이전의 인덱스를 유지하기 때문이다. 그러

므로 그리디 알고리즘의 최종 결과에서 불필요한 인덱스를 제거해야 한다.

불필요한 인덱스의 제거는 그리디 알고리즘의 결과 인덱스 집합에서 삭제하여도 질의 성능에 변화가 없는 멤버 인덱스들을 삭제함으로써 이루어진다. 즉, 인덱스 집합의 각 멤버 인덱스에 대해서 삭제를 한 경우 질의 성능을 계산하여 동일하면 해당 멤버 인덱스를 제거한다.

알고리즘 2 확장된 그리디 알고리즘

// r_{C_i}는 클래스 C_i에 대한 중복도이며, r은 주어진 최대 허용 중복도이다.
$O_{new} := \{\{C_1\}\{C_2\}..\{C_n\}\}$
repeat
  $O_{old} := O_{new}$
  for all I,J such that $I,J \in O_{old}$ and $I \neq J$ **do**
    for all K such that $K \in \{(I\ J)\}\{(I)\}\{(J)\varnothing\}$ **do**
      $O := (O_{old} \cup \{(I \cup J)\}) - K$
      각 클래스 C_i에 대해서 인덱스 집합 O의 중복도 r_{C_i}를 계산한다.
      $O := (O - \{\{C_k\} \mid r_{C_k} > r\}) \cup \{\{C_i\} \mid r_{C_i} < r\}$
      if 질의비용$_O$ < 질의비용$_{O_{new}}$ and O still satisfies the constraint r **then**
        $O_{new} = O$
      end if
    end for
  end for
until $O_{old} = O_{new}$
O_{new}에서 불필요한 인덱스를 제거한다.
O_{new}가 중복도를 r로 하였을 때의 그리디 방법에 의한 최적의 인덱스 집합이다.

예제 4 다음은 예제 3의 클래스 계층 구조에 대해서 최대 허용 중복도를 2로 하였을 때의 수행 과정을 보인 것이다. 인덱스 합병 결과 중 동일한 인덱스 집합 구성은 표시하지 않았다.

제 1 루프: 초기 인덱스 집합에 대해서 가능한 합병은 다음과 같다.

$O = \{\{1\}\{2\}\{3\}\{4\}\} : 1.2944$
$O_1 = \{\{1\}\{2\}\{3\}\{4\}\{1\ 2\}\} : 1.2599$
$O_2 = \{\{1\}\{2\}\{3\}\{4\}\{1\ 3\}\} : 1.2646$
$O_3 = \{\{1\}\{2\}\{3\}\{4\}\{1\ 4\}\} : 1.2646$
$O_4 = \{\{1\}\{2\}\{3\}\{4\}\{2\ 3\}\} : 1.1196$
$O_5 = \{\{1\}\{2\}\{3\}\{4\}\{2\ 4\}\} : 1.2162$

제 2 루프: 제 1 루프에서 선택된 O_4에 대해서 다시 합병을 시도하면 다음과 같다. 여기서 $O_{4,1}$과 $O_{4,2}$는 {1}과 {2 3}을 합병하였을 때의 구성인데, $O_{4,1}$은 {2 3}을 유지하는 경우이고 $O_{4,2}$는 {1}을 유지하는 경우이다. 모두 유지하는 경우와 모두 유지하지 않는 경우는 각각 $O_{4.1}$, $O_{4,2}$와 같으므로 생략하였다.

$O = \{\{1\}\{2\}\{3\}\{4\}\{2\ 3\}\} : 1.1196$
$O_1 = \{\{1\}\{3\}\{4\}\{2\ 3\}\{1\ 2\}\} : 1.1196$
$O_2 = \{\{1\}\{2\}\{4\}\{2\ 3\}\{1\ 3\}\} : 1.1593$
$O_3 = \{\{1\}\{2\}\{3\}\{4\}\{2\ 3\}\{1\ 4\}\} : 1.0898$
$O_{4,1} = \{\{1\}\{4\}\{2\ 3\}\{1\ 2\ 3\}\} : 1.2169$
$O_{4,2} = \{\{1\}\{2\}\{3\}\{4\}\{1\ 2\ 3\}\} : 1.0954$
$O_5 = \{\{1\}\{3\}\{4\}\{2\ 3\}\{2\ 4\}\} : 1.1196$
$O_6 = \{\{1\}\{2\}\{4\}\{2\ 3\}\{3\ 4\}\} : 1.2402$
$O_{7,1} = \{\{1\}\{4\}\{2\ 3\}\{2\ 3\ 4\}\} : 1.1733$
$O_{7,2} = \{\{1\}\{2\}\{3\}\{4\}\{2\ 3\ 4\}\} : 1.1023$

같은 방식으로 그리디 알고리즘을 반복 수행하면 마지막 루프에서 다음과 같은 결과를 얻는다.

$O = \{\{1\}\{2\}\{3\}\{4\}\{1\ 2\ 3\ 4\}\} : 1.0780$
$O_1 = \{\{3\}\{4\}\{1\ 2\ 3\ 4\}\{1\ 2\}\} : 1.0780$
$O_2 = \{\{2\}\{4\}\{1\ 2\ 3\ 4\}\{1\ 3\}\} : 1.1178$
$O_3 = \{\{2\}\{3\}\{1\ 2\ 3\ 4\}\{1\ 4\}\} : 1.1113$
$O_4 = \{\{1\}\{4\}\{1\ 2\ 3\ 4\}\{2\ 3\}\} : 1.1353$
$O_5 = \{\{1\}\{4\}\{1\ 2\ 3\ 4\}\{2\ 4\}\} : 1.2133$
$O_6 = \{\{1\}\{2\}\{1\ 2\ 3\ 4\}\{3\ 4\}\} : 1.3192$

이때 더 이상 성능 향상을 얻을 수 없기 때문에 O를 최종 결과로 하고, 여기서 멤버 인덱스 {1}과 {2}는 불필요하기 때문에 삭제한다.

그러므로 최종 결과는 $O = \{\{3\}\{4\}\{1\ 2\ 3\ 4\}\}$가 된다.

2.2 질의의 수행

질의는 구성된 인덱스 집합에서 최소한의 비용으로 처리할 수 있는 방안을 찾아 수행한다. 질의 수행의 최적 방안은 먼저 주어진 인덱스 집합에서 검색을 위해서 반드시 필요한 멤버 인덱스들을 찾고, 이들 인덱스로서 처리할 수 있는 클래스들을 구한다. 그리고 질의 대상 클래스 중 필수 멤버 인덱스로서 처리할 수 없는 클래스에 대해서는 나머지 멤버 인덱스들을 사용한다. 이때

검색 비용을 최소화하기 위해서는 이 나머지 멤버 인덱스들의 모든 조합을 검사하여 그 비용이 최소가 되는 조합을 구한다. 비록 이 조합을 찾는 것 또한 시간 복잡도가 클래스 수에 기하급수적으로 증가하나, 일반적인 경우 - 클래스의 수가 20개 이내 - 인덱스 집합내의 인덱스의 수가 적으며, 또한 한 번 찾고 나면 질의 수행시 이를 계속해서 사용할 수 있기 때문에 성능에는 별다른 영향을 미치지 않는다. 그리고 클래스의 수가 많은 경우에는 클래스 계층 구조를 나누어 각각에 대해서 인덱스 집합 방법을 적용하면 된다.

또는 나머지 멤버 인덱스들의 모든 조합을 검사하는 대신에 최적의 질의 수행 방법을 찾는 것 역시 그리디 알고리즘을 적용할 수 있다. 예를 들면 나머지 멤버 인덱스 중 가장 많은 질의 대상 클래스를 포함하는 인덱스부터 차례로 선택해 나가는 방법을 생각해 볼 수 있다.

본 논문의 성능 평가에서는 첫번째 방법을 사용하여 질의 수행 계획을 찾았다.

3. 비용 모델

본 절에서는 인덱스 구성의 질의 및 저장 비용의 계산을 위한 모델을 제시하겠다. 먼저 본 비용 모델의 기본 가정 및 매개변수들을 소개하고, 이로부터 질의 비용 및 저장 비용의 계산을 유도하겠다. 그리고 제시한 비용 모델에 대한 타당성 검증을 위한 실험 결과를 보이겠다. 우리의 인덱스 집합은 각 멤버 인덱스에 대해서 클래스 계층 인덱스 구조를 사용한다. 즉, 기존의 B^+-트리 구조에 클래스 ID의 저장을 위해 말단 노드의 구성을 변형한 형태를 사용한다. 따라서 본 비용 모델은 [4]에서 제시한 비용 모델을 기반으로 하였다.

먼저 본 비용 모델의 기본 가정은 다음과 같다.

- 모든 키 값의 평균 길이는 같다.
- 모든 키 값은 각 키 값의 범위 내에서 균일한 분포를 갖는다.
- 말단 노드의 레코드는 모두 인덱스 페이지 크기보다 작거나 모두 인덱스 페이지 크기보다 크다.

3.1 매개변수

데이타베이스 매개변수

- D_i – 인덱스 i에서 서로 다른 키 값의 수
- D_{C_i} – 클래스 C_i의 인스턴스 중 서로 다른 키 값의 수
- D – 서로 다른 키 값의 총 수

$$D = \bigcup_{\text{for all indexes}} D_i$$

- N_{C_i} – 클래스 C_i의 총 인스턴스 수
- N – 인덱스 i에 포함된 클래스들의 총 인스턴스 수
- N – 데이타베이스 내의 총 인스턴스 수

$$N = \sum_{\text{for all classes}} N_{C_i}$$

- K_i – 인덱스 i에서 동일한 키 값을 갖는 평균 인덱스 개체 수

$$K_i = N_i / D_i$$

- NC_i – 인덱스 i에서 각 인덱스 레코드에 포함된 평균 클래스 수

$$NC_i = \sum_{\text{for all classes in index } i} \frac{D_{C_i}}{D_i}$$

인덱스 매개변수

- P – 인덱스 페이지의 크기
- f – 내부 노드(internal node)의 평균 진출차수(fanout)
- kl – 키 값의 평균 길이
- XL_i – 인덱스 i의 말단 노드 레코드의 평균 길이

$$\begin{aligned} XL_i = \; & header\_length + kl \\ & + (sizeof(\text{CLASSID}) \\ & + sizeof(\text{offset}) \\ & + sizeof(\text{number\_of\_OIDs})) \times NC_i \\ & + sizeof(\text{OID}) \times K_i \end{aligned}$$

여기서 레코드의 헤더는 record_length, key_length, overflow_page_id, number_of_classes 등의 필드로 구성된다.

- LP_i – 인덱스 i의 말단 노드 페이지의 수 (오버플로우 페이지 제외)
- OP_i – 인덱스 i의 오버플로우 페이지의 수

$$\begin{aligned} &\text{if } XL_i \le P \quad && LP_i = \lceil (D_i \times XL_i)/P \rceil \\ &\text{if } XL_i > P \quad && LP_i = D_i \\ & && LP_i + OP_i = D_i \times \lceil XL_i/P \rceil \end{aligned}$$

- H_i – 인덱스 i의 내부 높이 (말단 노드 단계 제외)

$H_i = (LP_i + \lceil LP_i/f \rceil + \lceil \lceil LP_i/f \rceil /f \rceil + \cdots + 1)$에서의 항의 수

3.2 저장 비용 모델

- 인덱스 i의 저장 비용

$$SC_i = \begin{cases} \text{if } XL_i \le P \\ \quad LP_i + (\lceil LP_i/f \rceil + \lceil \lceil LP_i/f \rceil /f \rceil + \cdots + 1) \\ \text{if } XL_i > P \\ \quad LP_i + OP_i + (\lceil LP_i/f \rceil + \lceil \lceil LP_i/f \rceil /f \rceil + \cdots + 1) \end{cases}$$

- 총 저장 비용

$$SC = \sum_{\text{for all indexes}} SC_i$$

3.3 질의 비용 모델

단일-키 질의(Single-Key Query)의 비용

● 인덱스 i에 대한 질의 비용

$$RC_i^{single} = \begin{cases} H_i + 1 & \text{if } XL_i \le P \\ H_i + \lceil XL_i/P \rceil & \text{if } XL_i > P \end{cases}$$

● 질의 q에 대한 총 질의 비용

$$RC^{single} = \sum_{\text{for all indexes used by } q} RC_i^{single}$$

영역 질의(Range Query)의 비용

● 인덱스 i에 대한 질의 비용

$$RC_i^{range} = \begin{cases} H_i + \lceil query\_range \times LP_i \rceil \\ \text{if } XL_i \le P \\ H_i + \lceil query\_range \times (LP_i + OP_i) \rceil \\ \text{if } XL_i > P \end{cases}$$

● 질의 q에 대한 총 질의 비용

$$RC^{range} = \sum_{\text{for all indexes used by } q} RC_i^{range}$$

평균 질의 비용 비율 인덱스 구성의 질의 성능은 질의의 대상 클래스에 따라 매우 달라진다. 그러므로 두 인덱스 구성의 성능 비교를 위해서는 하나의 단일 평가 방법이 필요하다. 이를 위하여 우리는 평균 질의 비용 비율(Average Retrieval Cost Ratio)을 도입하였다. 평균 질의 비용 비율은 질의 성능만을 고려하였을 때 최적의 구성 방법, 즉 각 클래스마다 전 익스탠트에 대해서 인덱스를 구성하는 방법에 대한 상대적인 평균 질의 비용의 비율을 말한다.

또한 실제 환경에서의 질의 성능은 질의 패턴에 따라서도 영향을 받는다. 즉, 단일-키 질의와 영역 질의의 비율은 어떻게 되는지, 또 각 클래스에 대한 질의의 비율은 어떤가에 따라서 인덱스 구성의 성능은 달라지게 된다.

이러한 실제 환경에서의 질의 패턴을 반영하기 위해서 평균 질의 비용의 계산에 단일-키 질의 비율과 각 클래스에 대한 질의 가중 값을 적용하였다.

먼저 각 클래스 C_i에 대한 질의 비용 비율 R_{C_i}는,

$$R_{C_i} = \frac{\sum_{\text{for all index } I_j \text{ used for a query on } C_i} H_{I_j}}{H_{I_C^0}} \times \text{single point query ratio} + \frac{\sum_{\text{for all index } I_j \text{ used for a query on } C_i} LP_{I_j}}{LP_{I_C^0}} \times (1 - \text{single point query ratio})$$

이다. 여기에 질의 가중 값을 적용하면 평균 질의 비용 비율은 다음과 같다.

$$\text{평균 질의 비용 비율} = \sum_{\forall C_i} R_{C_i} \times \text{query weight on } C_i$$

여기서 $I_{C_i}^0$는 클래스 C_i의 전 익스탠트에 대한 인덱스를 말한다.

3.4 비용 모델의 검증

우리가 제시한 비용 모델의 적합성을 검증하기 위해 실제 구현된 인덱스를 사용해서 키 값의 범위와 인스턴스의 개수, 단일-키 질의의 비율, 질의 키 값의 범위 등을 변화시키면서 각 질의시 접근된 인덱스 페이지 개수를 측정하고 이를 비용 모델을 통한 예상 값과 비교하였다. 그 중 한 결과인 표 1은 각각 10만개의 인스턴스를 가지는 세 개의 클래스에 대해서, 하나의 인덱스로 구성했을 때와 각각 인덱스를 구성했을 때의 측정 결과와 비용 모델에 의해 예측한 결과를 보인 것이다. 표에서 오차 비율은 (실험 결과 값-예상값)/예상값×100으로 계산하였다.

표 1 비용 모델의 검증

단일-키 질의 비율(%)	5			10			95		
질의 키 값의 범위(%)	1	5	10	1	5	10	1	5	10
비용 모델에 의한 예상 값	1.63	1.63	1.63	1.71	1.71	1.71	2.93	2.93	2.93
실험 결과 값	1.71	1.61	1.61	1.75	1.70	1.68	2.90	2.90	2.90
오차 비율(%)	+4.9	-1.2	-1.2	+2.3	-0.6	-1.8	-1.0	-1.0	-1.0

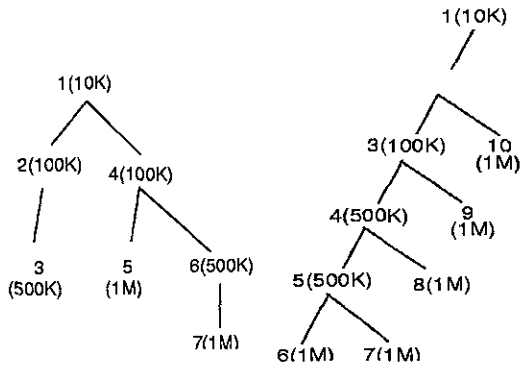
표의 결과에서 보듯이 어느 경우나 예상 값과 실험 결과 값은 5% 이내의 오차를 보이고 있다. 우리의 비용 모델에서는 질의 키 값의 범위를 고려하지 않았기 때문에 질의 영역이 좁은 경우나 영역 질의 비율이 높을 수록 오차가 다소 커진다.

4. 성능 평가

본 성능 평가 실험은 그림 3의 다섯 가지 클래스 계층 구조에 대해서 실시하였다. 클래스 계층 구조는 H1과 같은 단순한 구조, H2와 같은 한쪽으로 치우친 구조, H4와 같은 부시 구조, 치우친 구조와 부시 구조를 함께 갖는 H3, 그리고 H5의 다중 상속 구조 등 대표적인 클래스 계층 구조들로 구성하였다. 성능 평가는 각 중복도에 따른 인덱스 집합 구성 방식과 클래스 분할 방식, 그리고 클래스 계층 인덱스 구조에 대해서 실시하였다.

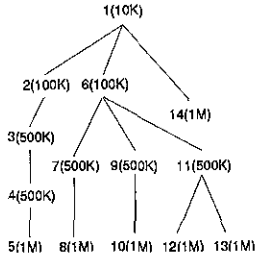
본 실험은 제 3절의 비용 모델에 기반하여 각 클래스에 대한 질의 비용 및 평균 질의 비용 비율, 그리고 총

저장 비용 비율을 산출하여 실시하였다 이때 총 저장 비용 비율은 평균 질의 비용 비율과 마찬가지로 모든 클래스에 대해서 중복하여 인덱스를 구성했을 때를 기준으로 한 저장 비용 비율을 말한다. 실험에서 사용한 매개변수의 값은 표 2와 같다. 또한 단일-키 질의 비율은 10%로 하였으며, 각 클래스의 질의 가중 값은 동일한 것으로 실험하였다. 데이타 분포는 완전 포함 분포로 키 값의 범위의 크기는 500,000이며 각 클래스에 속한 인스턴스들의 수는 그림 3에서 괄호 안에 표시하였다.

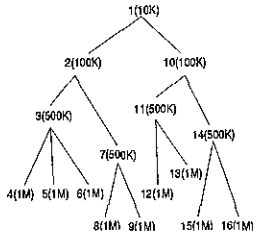


(a) 계층 구조 H1

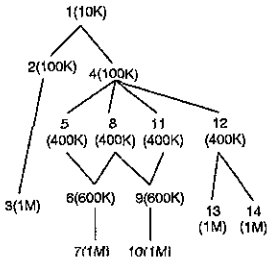
(b) 계층 구조 H2



(c) 계층 구조 H3



(d) 계층 구조 H4



(e) 계층 구조 H5

그림 3 성능 평가 클래스 계층 구조

표 2 실험 매개변수

P	4096
f	255 =(4096/(8+4))×(3/4)
kl	8
sizeof(OID)	8
sizeof(CLASSID)	4
sizeof(Page_id)	4
sizeof(offset) *sizeof*(number_of_OIDs) *sizeof*(record_length) *sizeof*(key_length) *sizeof*(number_of_classes)	2

4.1 성능 비교

먼저 표 3은 각 중복도에 따라서 구성한 인덱스 집합의 결과와, 클래스 분할 알고리즘에 따른 인덱스 구성을 보인 것이다. 여기서 IS-i는 중복도를 i로 하였을 때 인덱스 집합을 말하며, CD와 CH는 각각 클래스 분할에 의한 인덱스 구성과 클래스 계층 인덱스 구조를 말한다.

표 3 인덱스 집합 및 클래스 분할에 의한 인덱스 구성

(a) 계층구조 H1

	인덱스 구성
IS-1	{{1-3}{4-5}{6 7}}
IS-2	{{1-7}{1-3}{6 7}{5}}
IS-4	{{1-7}{2 3}{6 7}{3}{5}{7}}
CD	{{1-7}{2 3}{4 5}{6 7}{3}{5}{7}}

(b) 계층구조 H2

	인덱스 구성
IS-1	{{1 10}{2 3 9}{4 8}{5 6}{7}}
IS-2	{{1 2 9 10}{3-8}{6}{7}{8}{9}{10}}
IS-3	{{1-10}{5-7}{6}{7}{8}{9}{10}}
IS-4	{{1-10}{4-8}{5-7}{6}{7}{8}{9}{10}}
IS-5	{{1-10}{3-9}{4-8}{5-7}{6}{7}{8}{9}{10}}
CD	{{1-10}{3 4 8 9}{5-7}{2 10}{4 8}{6}{7}{8}{9}{10}}

(c) 계층구조 H3

	인덱스 구성
IS-1	{{1-5}{6 12}{7 8}{9 10}{11 13}{14}}
IS-2	{{1-5}{6 11-13}{7 8}{5} {9 10}{8}{10}{12}{13}{14}}
IS-3	{{1-5}{6-14}{4 5}{7 8}{9 10} {11 12}{5}{8}{10}{12}{13}{14}}
IS-4	{{1-5}{6-14}{6-13}{3-5}{4 5}{7 8} {9 10}{11-13}{5}{8}{10}{12}{13}{14}}
CD	{{1-14}{2-5}{6-10}{11-13}{4 5}{7 8} {9 10}{3}{5}{8}{10}{12}{13}{14}}

(d) 계층구조 H4

	인덱스 구성
IS-1	{{1-4}{7 8}{10 13}{11 12}{14 16} {5}{6}{9}{15}}
IS-2	{{1-6}{7-9}{10-13}{14-16}{4}{5}{6}{8}{9} {12}{13}{15}{16}}
IS-3	{{1-9}{10-16}{3-6}{7-9}{11-13}{14-16}{4}{5} {6}{8}{9}{12}{13}{15}{16}}
CD	{{1-16}{5-8}{3 4}{5 6}{7 8}{11 12}{15 16}{2} {4}{5}{6}{8}{9}{10}{12}{13}{14}{15}{16}}

(e) 계층구조 H5

	인덱스 구성
IS-1	{{1-3}{5-7}{4 8-11}{12 14}{13}}
IS-2	{{1-3}{4 12-14}{5-7}{8-11}{3}{7}{10}{13}{14}}
IS-3	{{1-4 12-14}{1-3}{4 12-14}{5-8}{9-11}{6 7} {9 10}{3}{7}{10}{13}{14}}
IS-4	{{1 5-11}{2-4 12-14}{4 12-14}{2 3} {5-7}{9-11}{12-14}{6 7}{9 10} {3}{7}{10}{13}{14}}
IS-5	{{1-14}{1-3}{5-11}{12-14}{5-7}{9-11}{6 7} {9 10}{3}{7}{10}{13}{14}}

다음 그림 4는 각 구성 방식에 대한 질의 비용을 그래프로 표시한 것이다. 먼저 왼쪽의 그래프는 각 클래스별로 질의 비용 비율을 보인 것이고, 오른쪽 그래프는 평균 질의 비용 비율과 총 저장 비용 비율을 각 구성 방식에 따라 보인 것이다.

각 인덱스 구성 방식에 따른 성능 평가 결과를 살펴보면 다음과 같다. 먼저 클래스 계층 인덱스 구성의 경우 모든 계층 구조에 대해서 가장 나쁜 성능을 보이고 있다. 클래스 계층 인덱스는 루트 클래스에 대한 질의에 대해서만 최적의 결과를 보이고 있을 뿐, 말단 클래스로 갈수록 성능의 악화가 심해짐을 볼 수 있다.

클래스 분할 방식의 경우 인덱스의 중복 구성을 통해 우수한 검색 성능을 보이고 있다. 그러나 클래스 분할 구성은 중복을 통한 성능의 향상이 인덱스 집합 구성보다 상대적으로 작다. 즉, 그림 4에서 볼 수 있듯이 모든 경우에서 클래스 분할 구성 방식은 같은 저장 비용을 갖는 인덱스 집합 구성보다 질의 성능이 떨어진다. 예를 들면, 그림 4(a)의 경우 IS-3 구성의 평균 질의 비용 비율과 총 저장 비용 비율이 각각 1.022, 0.784인데 반해 클래스 분할 구성의 경우 저장 비용 비율은 0.951로 높은 반면 질의 비용 비율은 1.028로 오히려 IS-3 구성보다 나쁘다.

인덱스 집합 방식의 경우, 중복도가 높아짐에 따라서 검색 성능이 향상됨은 당연하다. 그러나 이때 저장 비용 비율의 증가와 검색 성능의 비율 향상이 항상 선형적인 관계가 있는 것이 아니다.그러므로 실제 환경에서는 최소한의 비용으로 만족할 만한 검색 성능을 얻을 수 있는 중복도를 선택하는 것이 중요하다. 본 실험 결과에서는 대부분의 경우에서 중복도 2 또는 3에서 꽤 만족할 만한 성능을 얻을 수 있었으며, 그 이상의 중복도에서는 중복에 따른 오버헤드만큼 검색 성능의 향상을 얻을 수 없었다.

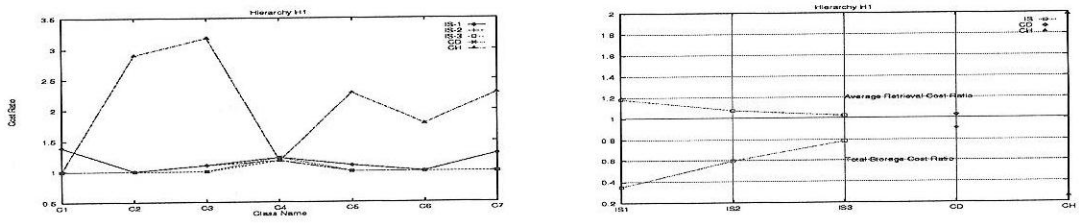
결과에서 몇 가지 특징적인 면을 살펴보면, 먼저 한쪽으로 치우친 구조를 갖는 클래스 계층 구조 H2의 경우에는 중복도를 2로 할 때 1의 경우에 비해서 급격한 질의 성능의 향상을 보였다. 반면 그 이상에서는 완만한 성능의 향상을 보였다. 이는 중복도를 2로 함으로써 각 말단 클래스에 대한 검색 성능을 최적화함과 동시에 상위 클래스들에 대해서도 어느 정도 성능 향상을 얻을 수 있었기 때문이다.

또한 다중 상속을 포함한 클래스 계층 구조 H5에 대해서도 인덱스 집합 구성 방식은 좋은 결과를 보여주고 있다. 그러나 클래스 분할 알고리즘은 다중 상속을 포함한 경우 이를 처리할 수 없기 때문에 실험에서 제외하였다.

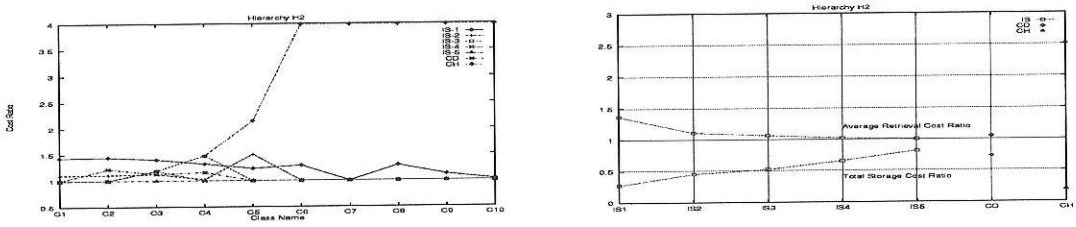
인덱스 집합 구성 방식은 중복도를 1로 하였을 때, 즉 중복이 없이 파티션하여 인덱스를 구성하였을 때에도 클래스 계층 인덱스 구성 방식에 비해서 매우 좋은 질의 성능을 보여주고 있다. 이는 저장 및 갱신 비용의 오버헤드 없이도, 인덱스 구성 방식에 따라 상당한 질의 성능의 향상을 얻을 수 있음을 보여주는 것이다.

4.2 그리디 알고리즘의 성능

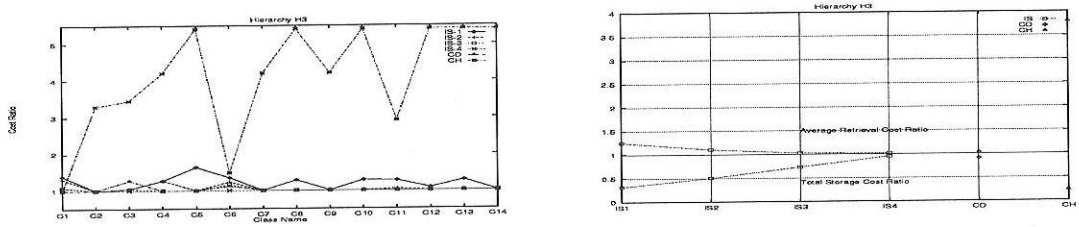
앞서 성능 평가 결과에 나타난 바와 같이 본 논문에서 제시한 그리디 알고리즘은 매우 좋은 결과를 보이고 있다. 그러나 그리디 알고리즘은 근본적으로 지역적 최적화(local optimum)에 빠질 수 있다. 이를 피하기 위해서 예견 기법(look ahead)을 사용할 수 있다. 실제 앞 실험의 대상 클래스에 대해서 1차 예견 기법을 사용하여 인덱스 집합을 구한 결과 다섯 가지 경우에서 성능 향상을 보였다. 그러나 이때 그리디 알고리즘이 지역적 최적화에 빠진 경우에도 전역적 최적 결과에 비해서 그리 나쁘지 않은 결과를 보여 주었는데, 이는 클래스 계층 구조상에서의 질의는 직접 또는 간접적으로 *IS-A* 관계를 가지고 연결되어 있는 클래스들을 대상으로 해서 이루어지므로 그리디 알고리즘에서와 같은 단계적 합병으로도 좋은 결과를 얻을 수 있기 때문이다.



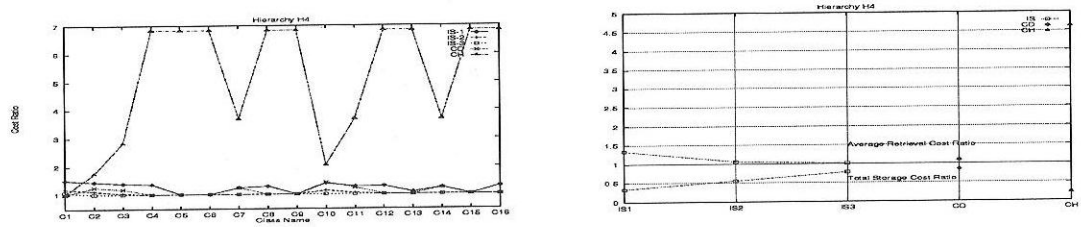
(a) 클래스 계층구조 H1



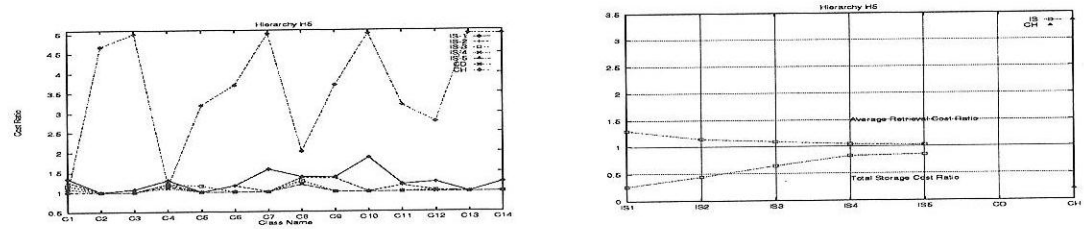
(b) 클래스 계층구조 H2



(c) 클래스 계층구조



(d) Hierarchy H4



(e) 클래스 계층 구조

그림 4 각 인덱스 구성의 성능 평가 결과

본 그리디 알고리즘의 성능에 대한 이론적 분석은 현재 진행 중에 있다.

5. 결 론

인덱싱은 데이타베이스 시스템의 성능 향상에 절대적으로 중요하며, 데이타베이스의 성능은 바로 데이타 모델의 성공과 직결된다.

본 논문에서는 주어진 중복 허용도내에서 최적의 인덱스 구성을 찾는 방법, 인덱스 집합 기법을 제시하였다. 본 인덱스 집합 기법은 기존의 B^+-트리 구조를 그대로 이용할 수 있기 때문에 실제 데이타베이스 시스템에 바로 적용할 수 있다는 장점을 가지고 있다. 또한 각 사용 환경에 따른 데이타의 분포나 질의 패턴을 인덱스 구성에 반영함으로써, 실제 환경에 맞는 최적의 인덱스 구성을 제공할 수 있다.

그리고 다양한 클래스 계층 구조를 대상으로 본 인덱스 집합 기법의 성능을 비교 평가하였다. 실험에서 인덱스 집합 기법은 클래스 분할 방식에 비해서 같은 중복도로도 더 많은 성능 향상을 보여주었으며, 중복이 없는 경우에도 인덱스 집합은 클래스 계층 인덱스에 비해 매우 좋은 질의 성능을 나타내었다. 인덱스 집합 기법은 중복도가 증가할 수록 질의 성능이 향상되었는데, 대부분의 경우 중복도 2나 3에서 매우 큰 질의 성능의 향상을 얻을 수 있었다.

앞으로의 연구 과제는 그리디 알고리즘에 대한 이론적인 분석과 인덱스 집합 기법을 병렬 데이타베이스 시스템으로 확장하는 것이다.

참 고 문 헌

[1] C. C. Low, B. C. Ooi, and H. Lu, "H-trees: A Dynamic Associative Search Index for OODB," in *Proceedings of the ACM SIGMOD International Conference on Management of Data*, (San Diego, California), June 1992.

[2] B. Sreenath and S. Seshadri, "The hcC-tree: An Efficient Index Structure for Object-Oriented Databases," in *Proceedings of the International Conference on Very Large Data Bases*, 1992.

[3] S. Ramaswamy and P. C. Kanellakis, "OODB Indexing by Calss-Division," in *Proceedings of the ACM SIGMOD International Conference on Management of Data*, (San Jose, CA), 1995.

[4] W. Kim, K.-C. Kim, and A. Dale, *Object-oriented Concepts, Databases, and Applications*, ch. Indexing techniqes for object-oriented databases, Addison-Wesley, 1989.

[5] C. Kilger and G. Moerkotte, "Indexing Multiple Sets," in *Proceedings of the International Conference on Very Large Data Bases*, 1994.

[6] Y. Ishikawa, H. Kitagawa, and N. Ohbo, "Evaluation of Signature Files as Set Access Facilities in OODBs," in *Proceedings of the ACM SIGMOD International Conference on Management of Data*, (Washington, DC), May 1993.

[7] E. Bertino and W. Kim, "Indexing Techniques for Queries on Nested Objects," *IEEE Trans. on Knowledge and Database Eng.*, vol. 1, pp. 196-214, June 1989.

[8] E. Bertino and P. Foscoli, "Indexing Organization for Object-Oriented Database Systems," *IEEE Trans. on Knowledge and Database Eng.*, vol. 7, pp. 193-209, April. 1995.

[9] A. Guttman, "R-TREES: A Dynamic Index Structure for Spatial Searching," in *Proceedings of the ACM SIGMOD International Conference on Management of Data*, (Boston, Ma), June 1984.

안 정 호

1991년 2월 서울대학교 컴퓨터공학과 졸업. 1993년 2월 서울대학교 컴퓨터공학과 졸업. (공학석사). 1998년 8월 서울대학교 컴퓨터공학과 박사. 1998년 3월 ~ 현재 삼성전자 선임연구원. 관심분야는 객체지향 시스템, 데이터베이스.

송 하 주

1993년 2월 서울대학교 컴퓨터공학과 졸업. 1995년 2월 서울대학교 컴퓨터공학과 졸업(공학석사). 1995년 3월 ~ 현재 서울대학교 컴퓨터공학과 박사과정. 관심분야는 트랜잭션 시스템, 객체지향 시스템.

김 형 주

제 25 권 제 2 호(B) 참조