論文 97-22-2-2

분산 객체 지향 데이타베이스에서 클래스의 기법

正會員 이 순 미*, 임 해 철*

Vertical Class Fragmentation in Distributed Object-Oriented Databases

Soon Mi Lee*, Hae Chull Lim* *Regular Members*

요 약

본 논문에서는 분산 객체 지향 데이타베이스에서 클래스를 수직 분할 기법에 관하여 연구하였다. 제안된 클래스 수직 분할 기법에서는 우선 애트리뷰트를 분할하여 애트리뷰트 플래그먼트를 생성한 후에 각각의 플래그먼트 내에 속한 애트리뷰트들을 참조하는 메소드를 모아서 메소드 플래그먼트를 생성하였다. 애트리뷰트를 분할할 때에 메소드가 참조하는 애트리뷰트를 반영하기 위하여 질의 접근 행렬과 메소드 접근 행렬을 정의하였으며 클래스 계층이나 클래스 구성 계층을 통한 다른 클래스와의 관계를 나타내기 위하여 질의 접근 행렬, 메소드 접근 행렬 및 애트리뷰트 사용 행렬을 전체 클래스 대상으로 확장하였다.

ABSTRACT

This paper addresses the vertical class fragmentation in distributed object-oriented databases. In the proposed vertical fragmentation, after producing the attribute fragment by partitioning attributes, then the method fragment is produced by gathering methods referring the attribute in each fragment. For partitioning attributes, we define query access matrix(QAM) and method access matrix(MAM) to express attributes that method refers, and extend QAM, MAM and attribute usage matrix(AUM) to universal class environment for representing relationship among other classes through class hierarchy and class composite hierarchy.

I. 서 론

최근에 하드웨어의 발달로 인해 고성능 워크스테이션 상에서 객체 지향 데이타베이스 기법을 요구하는 응용이 자주 발생함에 따라 분산 객체 지향 데이타베이스가 등장하게 되었다[10].

분산 객체 지향 데이타베이스에서는 효과적으로

*홍익대학교 컴퓨터공학과
論文番號:97009-0113
接受日字:1997年 1月 13日

클래스를 분할하여 여러 사이트에 할당하는 분할 기법이 필요하다. 클래스를 분할하는 목적은 가용성과 성능을 향상시키기 위해서이다. 객체를 분산시킴으로써 동시성을 높이고 데이타 전송 및 중복을 줄이며 불필요한 데이타 접근을 줄일 수 있다.

기존의 관계형 데이타베이스에서 릴레이션의 분할 기법은 세 가지 유형, 즉, 수평 분할, 수직 분할 그리고 혼합 분할로 구분된다[1].

본 논문에서는 분산 객체 지향 데이타베이스에서 클래스를 수직으로 분할하는 기법을 연구하였다. 캡슐화, 계승, 복합 객체, 객체 식별자 및 메소드 등의 특징으로 인해 클래스를 수직으로 분할하는 기법은 기존의 관계형 데이타베이스의 분할 기법과 차이가 있다. 본 논문에서는 응용 질의와 메소드 및 다른 클래스와의 관계가 반영된 클래스 분할을 설계하였다.

본 논문의 구성은 다음과 같다. 2장에서는 클래스 분할의 배경을 소개하고, 3장에서는 클래스 분할을 애트리뷰트 분할과 메소드 분할로 나누어 설계하였고, 4장에서는 설계한 수직 분할 기법을 구현하여 타 기법과 비교하였으며 5장에서 결론을 맺었다.

Ⅱ. 클래스 분할의 배경

2.1 관련 연구

관련 연구는 관계형 데이타베이스에서의 수직 분할과 객체 지향 데이타베이스에서의 수직 분할로 나눌 수 있다. 관계형 데이타베이스에서의 수직 분할은 릴레이션을 애트리뷰트의 부분 집합으로 나누는 과정이며 Hoffer와 Severance[3], Navathe et al[8]와 Navathe와 Ra[9] 등에 의하여 수행되었다.

객체 지향 데이타베이스에서의 수직 분할 기법에 관한 연구는 다음과 같다. Karlapalem et al.[6]은 객체 지향 데이타베이스에서 분산을 설계할 때에 고려해야 하는 문제점을 다루었으나 확실한 알고리즘을 제공하지 못하였다. Karlapalem과 Li[4]는 세 가지 분할 기법, 즉, 수직 클래스 분할, 패스 분할 및 수평 클래스 분할을 제안하였다. Karlapalem, Li 및 Vieweg[5]는 메소드의 동작이 분할에 미치는 영향에 관해 초점을 맞추어 메소드로부터 유도된 분할 기법을 연구하였으나, 한 메소드에 의해 분할이 이루어지기 때문에 여러 메소드에 관한 정보나 사용자 질의에 관한 정보가 분할에 포함되지 않는 문제점이 발생한다. Ezeife과 Barker[2]는 응용 질의를 기초로 하여 메소드를 분할한 후에 애트리뷰트를 분할하는방법을 제안하였다. 중복을 피하기 위하여, 중복되는 애트리뷰트를 한 플래그먼트에만 할당하고 있는데, 이것은 객체 지향의 캡슐화 개념에 위배된다.

본 논문의 클래스 수직 분할 기법은 Navathe et al [8]의 분할 기법을 객체 지향 환경으로 확장한 것으로서 메소드와 다른 클래스와의 관계가 반영된 비중복 다중 분할을 설계하였다.

2.2 기본 개념

수직 분할이란 질의에 의해 함께 참조되는 애트리뷰트들을 모은 애트리뷰트의 부분 집합이다. 클래스를 분할하여 얻은 애트리뷰트의 부분 집합을 애트리뷰트 플래그먼트라고 정의하며, 메소드의 부분 집합을 메소드 플래그먼트라고 정의한다. 애트리뷰트 플래그먼트와 관련된 메소드 플래그먼트를 합쳐서 클래스 플래그먼트라고 정의한다.

본 논문에서 제안된 수직 분할 기법은 다음과 같은 전제 조건 하에서 수행된다.

【전제 조건】

첫째, 상위 클래스에서 하위 클래스로 애트리뷰트가 계승될 때에, 애트리뷰트의 실제 값은 정의된 상위 클래스에만 존재하며 하위 클래스로는 포인터만 계승된다.

둘째, 분할이 사용자 질의를 기초로 하여 수행되며, 데이타베이스에 관한 정보, 질의의 내용과 발생 빈도 등에 관한 정보는 분할되기 전에 이미 알려져 있다.

객체의 분산을 설계하는 과정은 다음의 두 단계로 이루어진다. 첫째, 논리적 단계로서 클래스에 발생된 질의의 접근 빈도를 기초로 하여 자주 사용되는 애트리뷰트들을 모아서 클래스를 분할한다. 둘째, 물리적 단계로서 플래그먼트를 여러 사이트에 할당하는 단계로서, 이 때에는 애트리뷰트의 크기, 불필요한 애트리뷰트 접근 비용, CPU 비용, 기억장치 비용 및 전송 비용 등을 고려하여야 하며, 특히 객체 지향 데이타베이스이기 때문에 계승이나 복합 관계를 고려해서

할당하여야 한다. 본 논문은 첫 번째 단계인 논리적 클래스 분할에 관하여 다룬다.

Ⅲ. 클래스 분할

클래스는 애트리뷰트와 메소드로 구성되기 때문에 클래스의 분할도 애트리뷰트의 분할과 메소드의 분할로 구성된다. 우선 애트리뷰트를 분할한 후에 애트리뷰트를 참조하는 메소드들을 묶어서 메소드를 분할한다.

3.1 애트리뷰트 분할

애트리뷰트의 분할은 질의에 의해 함께 참조되는 애트리뷰트들을 모아 애트리뷰트 플래그먼트를 형성함으로써 이루어진다. 분할 방법은 질의가 접근하는 애트리뷰트를 나타내주는 애트리뷰트 사용 행렬을 생성한 후에, 이것을 기초로 하여 애트리뷰트들 사이의 결합의 정도를 수치로 나타내 주는 애트리뷰트 결합 행렬을 생성한다는 점은 기존의 관계형 데이타베이스에서와 유사하지만 객체지향의 특성으로 인해 객체 지향 데이타베이스에서의 분할은 기존의 분할 방법과 많은 차이가 있다. 차이점은 다음과 같다. 첫째, 메소드를 통해 접근되는 애트리뷰트도 분할에 반영되어야 한다. ORION과 같은 객체 지향 질의어를 사용한 사용자 질의는 애트리뷰트뿐만 아니라 메소드에도 접근할 수 있기 때문에 접근된 메소드가 참조하는 애트리뷰트도 분할에 반영이 되어야 한다. 둘째, 클래스는 계승관계, 복합 객체 관계 등에 의해 다른 클래스와 연결되어 있다. 다른 클래스와의 관계에 의하여 한 클래스의 질의가 다른 클래스의 애트리뷰트에 접근할 수 있는데 이러한 점도 분할에 반영되어야 한다.

본 논문에서는 메소드에 의해 접근되는 애트리뷰트를 반영하기 위하여 질의 접근 행렬과 메소드 접근 행렬을 제안하였으며 다른 클래스와의 관계를 반영하기 위하여 분할 규칙을 생성하였고 질의 접근 행렬과 메소드 접근 행렬을 전체 클래스를 대상으로 확장하였다.

3.1.1 메소드에 의해 접근된 애트리뷰트

질의 접근 행렬은 사용자 질의에 의해 참조되는 애트리뷰트와 메소드를 나타내 준다. 그림 1은 질의 접근 행렬의 예로서 행렬의 각 행은 질의를 나타내며 열은 애트리뷰트 또는 메소드를 의미한다. 원소 "1"은 질의가 애트리뷰트나 메소드를 참조했음을 나타낸다. 질의 빈도는 여러 사이트에서 발생한 질의 빈도의 합을 뜻한다.

메소드는 애트리뷰트를 사용하여 정의되고 구현된다. 즉, 질의 접근 행렬에 존재하는 메소드 또한 해당 클래스의 애트리뷰트를 참조하게 된다. 메소드 접근 행렬은 메소드에 의해 접근되는 애트리뷰트를 나타낸다. 그림 2는 메소드 접근 행렬의 예이다.

애트리뷰트 사용 행렬이란 메소드가 참조한 애트리뷰트를 포함한, 질의에서 사용된 모든 애트리뷰트를 나타내주는 행렬이다. 즉, 질의 접근 행렬에서 사용된 메소드가 다시 그 클래스의 애트리뷰트를 참조할 수 있기 때문에 메소드 접근 행렬을 질의 접근 행렬에 대입시켜 애트리뷰트 사용 행렬을 생성할 수 있다. 그림 1과 그림 2로부터 유도된 애트리뷰트 사용 행렬이 그림 3이다. 애트리뷰트 사용 행렬을 기초로 하여 애트리뷰트들 간의 결합력를 나타내 주는 애트리뷰트 결합 행렬이 생성되며 이 결합 행렬을 클러스

	A1	A2	A3	A4	M1	M2	M3	질의 빈도
q1	1	0	1	0	0	0	0	50
q2	0	1	1	0	1	0	1	30
q3	1	0	0	1	0	1	0	90

그림 1. 질의 접근 행렬

메소드 \ 애트리뷰트	A1	A2	A3	A4
M1	0	1	1	0
M2	1	1	0	1
M3	0	1	0	1

그림 2. 메소드 접근 행렬

질의 \ 애트리뷰트	A1	A2	A3	A4	질의빈도
q1	1	0	1	0	50
q2	0	1	1	1	30
q3	1	1	0	1	90

그림 3. 애트리뷰트 사용 해열

터링하여 애트리뷰트를 분할하게 되는데 상세한 내용은 다음 절에서 다룬다.

3.1.2 다른 클래스와의 관계 반영

객체 지향 데이타베이스에서 한 클래스는 다른 클래스와 서로 연관되어 있으며, 이러한 연관 관계가 클래스의 분할에 반영되어야 한다.

클래스 분할에 영향을 미치는 관계는 계승 관계, 복합 관계 및 메소드 링크 관계로 분류된다. 계승 관계는 상위 클래스의 애트리뷰트와 메소드가 하위 클래스로 계승되는 것을 뜻하며, 복합 관계는 한 클래스의 애트리뷰트의 도메인이 다른 클래스가 될 수 있음을 의미한다. 또한 메소드 링크 관계는 메소드 내에서 다른 메소드를 호출하는 것을 의미한다.

(1)분할 규칙

각 관계에 관한 분할 규칙은 다음과 같다.

▶ 계승 관계에 대한 분할 규칙

클래스 P의 하위 클래스를 Q라고 가정할 때,

【규칙1】 클래스 P의 분할 시에 P의 애트리뷰트를 상속받아 사용하는 클래스 Q에 대한 질의를 참여시켜야 한다.

【규칙2】 클래스 P의 분할 시에 P의 메소드를 상속받아 사용하는 클래스 Q에 대한 질의를 참여시켜야 한다.

【규칙3】 Q에서 정의된 메소드 m이 P의 애트리뷰트를 참조하는 경우, 클래스 P의 분할시에 m을 사용하는 Q에 대한 질의를 참여시켜야 한다.

▶ 복합관계에 대한 분할 규칙

클래스 A의 한 애트리뷰트 k의 도메인이 다른 클래스 B라고 가정할 때에,

【규칙4】 클래스 B의 분할시에 애트리뷰트 k를 사용하는 클래스 A에 관한 질의를 참여시켜야 한다.

【규칙5】 클래스 A에서 정의된 메소드 n이 애트리뷰트 k를 참조하는 경우, 클래스 B의 분할시에 n을 사용하는 A에 대한 질의를 참여시켜야 한다.

▶ 메소드 링크 관계에 대한 분할 규칙

【규칙6】 메소드 m은 클래스 C에서 정의되고 메소드 n은 클래스 D에서 정의었으며 m 내에서 n이 호출된다고 가정을 할 때에, 클래스 D의 분할 시에 m을 사용하는 질의를 참여시켜야 한다.

(2)분할 행렬의 확장

다른 클래스와의 관계를 고려한 분할 규칙을 애트리뷰트 분할에 반영시키기 위해서 질의 접근 행렬, 메소드 접근 행렬과 애트리뷰트 사용 행렬을 전체 클래스 환경으로 확장해야 한다. 애트리뷰트 분할에 참여하는 질의는 해당 클래스에 관한 질의 뿐 만 아니라 다른 클래스의 질의로서 계승, 복합 및 메소드 링크 관계를 통해 분할하고자 하는 클래스의 애트리뷰트에 접근하는 질의이다. 따라서, 질의 접근 행렬은 다른 클래스를 포함시킨 전체 질의 접근(universal query access) 행렬로, 메소드 접근 행렬은 전체 메소드 접근 행렬로, 애트리뷰트 사용 행렬은 전체 애트리뷰트 사용 행렬로 확장되어야 한다. 그림 5, 6, 7은 그림 4의 클래스 다이어그램에 대한 전체 질의 접근 행렬, 전체 메소드 접근 행렬 및 전체 애트리뷰트 사용 행렬의 예이다.

그림 5의 전체 질의 접근 행렬에서 1Q1-1Q5는 PERSON에 대한 질의이고 2Q1-2Q5는 EMPLOYEE, 3Q1-3Q6은 DEPARTMENT, 4Q1-4Q3은 COMPANY 클래스에 대한 질의이다.

그림 6은 전체 메소드 접근 행렬의 예로서 복합 메소드를 표현하기 위하여 그림 2를 변형하여 행렬 의 열에 애트리뷰트와 메소드를 배치시켰다. 그림 6에서 행은 각 클래스에 속한 메소드를 나타낸다.

그림 7은 전체 애트리뷰트 사용 행렬의 예로서, 각 질의에 의해 참조되는 모든 애트리뷰트들을 나타내

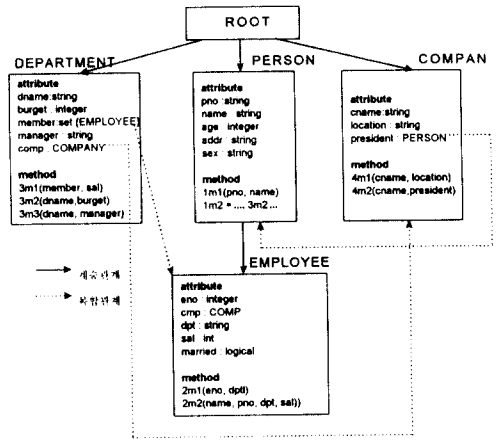


그림 4. 클래스 관계

	PERSON							EMPLOYEE0							DEPARTMENT									COMPANY					접근
	pno	name	age	addr	sex	1m1	1m2	eno	cmp	dpt	sal	marr	2m1	2m2	dname	member	burget	memNo	manager	comp	3m1	3m2	3m3	cname	location	president	4m1	4m2	빈도
1Q1	1	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	40
1Q2	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	60
1Q3	0	1	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	45
1Q4	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	45
1Q5	0	1	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	40
2Q1	0	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	25
2Q2	0	1	0	1	0	0	0	1	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	40
2Q3	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	25
2Q4	0	0	0	0	0	0	0	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	30
2Q5	0	1	0	0	0	1	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	35
3Q1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	40
3Q2	0	0	0	0	0	0	0	0	1	0	0	1	0	0	1	1	0	0	1	0	0	0	0	0	0	0	0	0	35
3Q3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	0	1	1	1	0	0	50
3Q4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0	1	0	1	1	0	0	0	25
3Q5	0	0	0	0	0	0	0	1	0	0	1	0	0	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	40
3Q6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	25
4Q1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	30
4Q2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0	35
4Q3	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	1	40

그림 5. 전체 질의 접근 행렬

	PERSON							EMPLOYEE							DEPARTMENT									COMPANY					접근
	pno	name	age	addr	sex	1m1	1m2	eno	cmp	dpt	sal	marr	2m1	2m2	dname	member	burget	memNo	manag	comp	3m1	3m2	3m3	cname	location	presid	4m1	4m2	빈도
1m1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	80
1m2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	0	0	1	0	0	0	0	0	0	40
2m1	0	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	40
2m2	1	1	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	30
3m1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	25
3m2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	65
3m3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0
4m1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	35
4m2	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	40

그림 6. 전체 메소드 접근 행렬

	PERSON					EMPLOYEE					DEPARTMENT						COMPANY			접근
	pno	name	age	addr	sex	eno	cmp	dpt	sal	marr	dname	member	burget	memNo	manager	comp	cname	location	president	빈도
1Q1	1	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	40
1Q2	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	60
1Q3	0	1	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	45
1Q4	1	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	45
1Q5	0	1	1	0	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	40
2Q1	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	25
2Q2	0	1	0	1	0	1	0	1	1	0	0	0	0	0	0	0	0	0	0	40
2Q3	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	25
2Q4	1	1	0	0	0	1	0	1	1	0	0	0	0	0	0	0	0	0	0	30
2Q5	1	1	0	0	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	35
3Q1	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	40
3Q2	0	0	0	0	0	0	1	0	0	1	1	1	0	0	1	0	0	0	0	35
3Q3	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	1	1	1	50
3Q4	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	1	1	1	0	25
3Q5	0	0	0	0	0	1	0	0	1	0	0	1	0	0	1	0	0	0	0	40
3Q6	0	0	0	0	0	0	0	0	1	0	0	1	0	0	1	0	0	0	0	25
4Q1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	30
4Q2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	35
4Q3	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	1	0	1	40

그림 7. 전체 애트리뷰트 사용 행렬

고 있는데, 계승 관계, 복합 관계 및 메소드 링크 관계를 반영한 모든 애트리뷰트들이 표시된다.

예를 들어, 그림 5에서 EMPLOYEE 클래스에 대한 질의 2Q4는 eno와 메소드 2m2에 접근하는데 그림 6에 의하면 2m2는 애트리뷰트 dpt와 sal 및PERSON으로부터 상속받은 애트리뷰트 pno와 name을 참조한다. 따라서, 그림 7의 전체 애트리뷰트 사용 행렬에서 질의 2Q4가 사용하는 애트리뷰트는 EMPLOYEE 클래스의 eno, dpt, sal과 PERSON의 pno, name이다. 그림 5에서, 3Q2는 복합 관계를 나타내주는 DEPARTMENT에 대한 질의로서, 애트리뷰트 member를 통해 EMPLOYEE의 애트리뷰트 cmp와 marr에 접근한다(그림4 참조). 1Q5는 메소드 링크 관계를 나타내주는 질의로서 그림6에 의하여 사용된 메소드 1m2 내에서 또 다른 메소드인 3m2가 호출된다.

전체 애트리뷰트 사용 행렬에서, 질의 q_i에 대한 애트리뷰트 A_j의 사용 값 use(q_i, A_j)는 다음과 같이 정의된다.

$$use(q_i, A_j) = \begin{cases} 1 & q_i\text{가 } A_j\text{를 참조하는 경우} \\ 0 & \text{그렇지 않은 경우} \end{cases}$$

전체 애트리뷰트 사용 행렬이 생성된 후에, 각각의 클래스에 대하여 애트리뷰트들간의 결합력을 나타내주는 애트리뷰트 결합(attribute affinity) 행렬이 생성된다. 한 클래스 C_m에 속한 두 애트리뷰트 A_i와 A_j 사이의 결합력은 다음과 같이 정의된다.

$$aff_m(A_i, A_j) = \sum_{k \mid use(q_k, A_i)=1 \wedge use(q_k, A_j)=1} acc(q_k)$$

여기서, $aff_m(A_i, A_j)$는 C_m 클래스의 애트리뷰트인 A_i와 A_j 사이의 결합 값이며, $acc(q_k)$는 질의 q_k의 접근 수인데 이 때에 q_k는 클래스 C_m에 대한 질의 뿐만 아니라 다른 클래스에 대한 질의도 될 수 있다. $aff_m(A_i, A_j)$는 애트리뷰트 A_i와 A_j를 함께 접근하는 모든 질의에 대한 acc의 합이다.

그림 8(a)는 그림 7의 DEPARTMENT 클래스에 대한 애트리뷰트 결합 행렬이다.

3.1.3 클러스터링과 분할

(1) 클러스터 행렬의 생성

생성된 애트리뷰트 결합 행렬을 클러스터링한 후에 이것을 여러 플래그먼트로 나누게 되는데 애트리뷰트를 클러스터링하기 위하여 Bond Energy Algorithm(BEA)[7]이 사용된다. BEA를 사용하는 목적은 애트리뷰트 결합 행렬의 큰 값은 큰 값들끼리 모으고 작은 값은 작은 값들끼리 모으기 위함이다. 애트리뷰트 결합 행렬을 입력받아 각 행과 열을 순열로 배치하여 클러스터 결합 행렬이 생성되는데, 다음의 수식을 최대화하는 방법으로 수행된다.

$$\sum_{i=1}^{n} \sum_{j=1}^{n} aff(A_i, A_j) \{(aff(A_i, A_{j-1}) + aff(A_i, A_{j+1})\} \quad (식1)$$

이때에,

$aff(A_i, A_0) = aff(A_0, A_j) = aff(A_i, A_{n+1}) = aff(A_{n+1}, A_j) = 0$

그림 8(a)의 애트리뷰트 결합 행렬을 클러스터링한 결과가 그림 8(b)의 클러스터 결합 행렬이다. 클러스터 결합 행렬의 대각선을 따라 애트리뷰트의 분할이 이루어진다.

(2) 애트리뷰트 플래스먼트의 생성

분할 방법은 애트리뷰트의 중복 여부에 따라 중복 분할과 비중복 분할이 있을 수 있으며, 플래그먼트수에 따라 이분(binary) 분할과 다중(n-ary) 분할로 나눌 수 있다. 본 논문에서는 키 값이 중복되지 않는 완전

애트리뷰트	dname	member	burget	memNo	manager	comp
dname	140	35	105	0	35	25
member	35	100	0	0	100	0
burget	105	0	105	0	0	25
memNo	0	0	0	50	0	50
manager	35	100	0	0	100	0
comp	25	0	25	50	0	75

(a) 애트리뷰트 결합 행렬

애트리뷰트	memNo	comp	burget	dname	manager	member
memNo	50	50	0	0	0	0
comp	50	75	25	25	0	0
burget	0	25	105	105	0	0
dname	0	25	105	140	35	35
manager	0	0	0	35	100	100
member	0	0	0	35	100	100

(b) 클러스터 결합 행렬

그림 8. 클러스터 결합 행렬

한 비중복 다중 분할을 지원하도록 설계하였다. 다중 분할이 알고리즘에 의해 수행될 경우에 알고리즘의 난이도는 $O(2^n)$이다[8]. 따라서, 본 논문에서의 다중 분할은 복잡성을 줄이기 위하여 우선 비중복 이중 분할을 한 후에 분할된 각각의 플래그먼트에 대해서 다시 비중복 이중 분할을 반복하는 방법을 채택하였다.

그림 8(b)의 대각선 위에 한 점 X가 있다고 가정을 하자. 점 X에 의해 애트리뷰트는 두 부분, 즉 좌측 상단과 우측 하단 부분으로 나누어지며 이것을 각각 T와 B로 표기한다. 각 영역 T와 B에 속한 애트리뷰트들이 각각 애트리뷰트 플래그먼트에 해당한다. 임의의 클래스 C_k에 속한 애트리뷰트들을 분할하기 위하여 다음과 같은 정의가 필요하다.

$CA(C_k)=\{A_n \mid A_n$은 클래스 C_k에서 정의된 애트리뷰트이다.$)$

$IQ(C_k)=\{q_m \mid q_m$은 클래스 C_k에 대한 질의이다.$\}$

$FQ(C_k)=\{q_m \mid q_m$은 C_k가 아닌 다른 클래스에 대한 질의로서, C_k의 애트리뷰트에 접근하는 질의이다.$\}$

$UA(C_k, q_m)=\{A_n \mid use(q_m, A_n)=1 \wedge A_n \in CA(C_k)\}$

$UQ(C_k)=\{q_m \mid use(q_m, A_n)=1 \wedge A_n \in CA(C_k)\}$

$=\{q_m \mid q_j \in IQ(C_k) \cup q_m \in FQ(C_k)\}$

$TQ(C_k)=\{q_m \mid UA(C_k, q_m) \subseteq T\}$

$BQ(C_k)=\{q_m \mid UA(C_k, q_m) \subseteq B\}$

$IQ(C_k)=UQ-\{TQ \cup BQ\}$

$IQ(C_k)$는 내부질의, $FQ(C_k)$는 외부질의라고 부른다. 외부질의는 계승, 복합 및 메소드 링크 관계를 통해 C_k의 애트리뷰트에 접근하는 질의를 의미한다. $UA(C_k, q_m)$는 질의 q_m가 사용한 클래스 C_k의 애트리뷰트의 집합이다. $UQ(C_k)$는 클래스 C_k의 애트리뷰트를 참조하는 모든 질의로서 내부질의와 외부질의의 합집합이다. $TQ(C_k)$는 상위 영역인 T에 속한 애트리뷰트 만을 접근하는 질의이며 $BQ(C_k)$는 하위 영역 B에 속한 애트리뷰트 만을 접근하는 질의이고 $IQ(C_k)$는 T와 B를 함께 접근하는 질의이다.

TQ나 BQ와 같이 하나의 플래그먼트만을 접근하는 질의의 총 접근 수는 최대화하고 IQ와 같이 두 플래그먼트를 모두 접근하는 질의의 총접근 수는 최소화하는 위치가 최적의 위치이다. 최적의 위치를 찾아 분할을 하기 위하여 다음과 같은 수식의 정의가 필요하다.

$$CTQ=\sum_{qm \in TQ} acc(q_m),$$

$$CBQ=\sum_{qm \in BQ} acc(q_m)$$

$$CIQ=\sum_{qm \in IQ} acc(q_m)$$

CTQ와 CBQ는 각각 한 플래그먼트 T나 B만을 접근하는 질의의 접근 수의 합이며 CIQ는 두 플래그먼트를 모두 접근하는 질의의 접근 수의 합이다. 수직 분할을 위한 분할 함수는 다음과 같이 정의된다.

$$Z=CTQ \times CBQ-CIQ^2 \qquad (식2)$$

함수 Z의 값을 최대화시키는 점 X를 찾음으로써 최적의 수직 분할이 이루어진다.

애트리뷰트를 분할하여 두 개의 애트리뷰트 플래그먼트를 생성한 후에 각각의 플래그먼트에 대해 다시 반복적으로 이중 분할을 수행함으로써 다중 분할이 이루어진다. 반복할 때에는 플래그먼트와 관련이 있는 질의만을 고려하며 관련이 없는 질의는 제거된다. 더 이상의 분할이 일어나지 않을 때까지 반복을 계속하는데, 위에서 정의한 분할 함수(식 2)의 최대값이 양수인 경우는 분할이 이루어지며 음수인 경우는 분할 수행을 취소한다.

그림 8(b)의 DEPARTMENT의 클러스터 결합 행렬에 대한 다중 분할의 결과는 그림 9와 같다.

분할	memNo	comp	burget	dname	manager	member	반복회수
1	x	x					2
2			x	x			2
3					x	x	1

그림 9. DEPARTMENT의 다중 분할 결과

3.2 메소드 분할

한 클래스에 속한 애트리뷰트가 분할되어 애트리뷰트 플래그먼트를 형성한 후에 메소드의 분할이 진행된다. 메소드 분할은 애트리뷰트 플래그먼트에 속

한 애트리뷰트들을 참조하는 메소드들을 모음으로써 이루어진다.

【정의 1】

클래스 C의 각 애트리뷰트 플래그먼트 AF_i에 대한 참조 메소드 집합 RM_i는 AF_i내의 모든 애트리뷰트들 각각을 참조하는 메소드들의 합집합이다.

메소드가 플래그먼트에 할당될 때에 같은 메소드가 중복되어 할당될 수도 있다. 이것은 한 메소드에 의해 참조되는 애트리뷰트들 중 일부는 한 애트리뷰트 플래그먼트에 속하고 또 다른 일부는 플래그먼트에 속함을 의미한다. 클래스 플래그먼트는 일종의 클래스로서 객체 지향 개념을 유지하여야 하는데 중복되는 메소드는 객체 지향의 캡슐화 개념에 위배된다. 이러한 문제점을 해결하기 위하여 수직 분할을 하는 클래스 C의 내부적인 구조를 C′로 변경하며, C′와 플래그먼트 사이에 클래스 구성 계층이 형성되도록 한다. 그리고, 중복되는 메소드를 참조 메소드 집합으로부터 제거하여 클래스 구성 계층 상의 참조하는 클래스(부모 클래스)에 배치시킨다.

그림10에서 DEPARTMENT는 DEPARTMENT′로 변형되며, DEPARTMENT′의 a1, a2와 a3의 도메인은 클래스 플래그먼트인 CF1, CF2와 CF3가 되며 중복되는 3m3은 DEPARTMENT′에 할당된다.

【정의 2】

클래스 C의 각 애트리뷰트 플래그먼트 AF_i에 대한 메소드 플래그먼트 MF_i는 C의 모든 참조 메소드 RM_i에서 중복되는 메소드를 제거시킨 것이다.

【정의 3】

클래스 플래그먼트 CF_i는 애트리뷰트 플래그먼트 AF_i와 그에 해당하는 메소드 플래그먼트 MF_i를 결합한 것이다.

Ⅳ. 구현 및 비교 분석

4.1 구현 결과

본 논문에서 설계된 클래스 수직 분할 기법을 IBM-PC 상에서 C 언어로 구현하였다. 구현 과정은 첫째로 예제 클래스 모델과 질의를 설정한다. 클래스 모델의 스키마는 그림 4와 같고 예제 질의 분석도는 그림 11과 같다. 둘째, 전체 질의 접근, 전체 메소드 접근, 전체 애트리뷰트 사용 행렬을 생성하는데 예제 클래스와 질의에 대한 각각의 행렬은 그림 5, 6, 7과 같다. 셋째, 각각의 클래스에 대해 애트리뷰트 결합 행렬과 클러스터 결합 행렬을 생성한다. 클러스터 결합 행렬의 결과는 그림 12와 같다. 넷째, 애트리뷰트 플래그먼트와 메소드 플래그먼트를 생성한다. 분할 방법은 3.1.3절의 분할수식(식 2)을 최대화시키는 위치(점 X)를 클러스터 결합 행렬에서 찾는 것이다.

각각의 클래스에 대한 분할 결과는 다음과 같다.

▶ PERSON 클래스의 분할

애트리뷰트 플래그먼트:

$AF1=\{addr, pno, name\}$, $AF2=\{age, sex\}$

메소드 플래그먼트(그림 6 참조):

$MF1=\{1m1, 1m2\}$, $MF2=\{\ \}$

따라서, 클래스 플래그먼트는 다음과 같다.

$CF1=\{AF1, MF1\}$

$=\{\{addr, pno, name\}, \{1m1, 1m2\}\}$

$CF2=\{AF2, MF2\}=\{\{age, sex\}, \{\ \}\}$

▶ EMPLOYEE 클래스의 분할

$CF1=\{AF1, MF1\}$

$=\{\{sal, eno, dpt\}, \{2m1, 2m2\}\}$

$CF2=\{AF2, MF2\}=\{\{cmp, married\}, \{\ \}\}$

▶ DEPARTMENT 클래스의 분할

$CF1=\{AF1, MF1\}=\{\{memNo, comp\}, \{\ \}\}$

$CF2=\{AF2, MF2\}=\{\{burget, dname\}, \{3m2\}\}$

$CF3=\{AF3, MF3\}=\{\{manager, member\}, \{3m1\}\}$

▶ COMPANY 클래스의 분할

COMPANY 클래스의 경우, 분할 함수의 값이 모두 음수이므로 분할이 이루어지지 않는다.

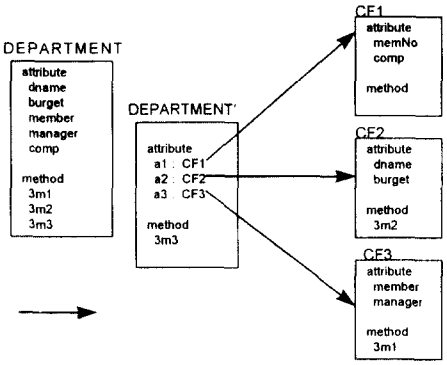


그림 10. DEPARTMENT 클래스의 변형

		애트리뷰트			메소드				
		계승관계		복합 관계	계승관계		복합 관계		메소드링크관계
		단순 애트리뷰트 (A1)	상속받은 애트리뷰트 (A2)	도메인이 다른 클래스인 애트리뷰트(A3)	단순메소드 (M1)	상속받은 메소드 (M2)	A2를 사용하는 메소드(M3)	A3를 사용하는 메소드(M4)	복합 메소드 (M5)
PERSON	1Q1	O							
	1Q2	O							
	1Q3	O							
	1Q4	O			O				
	1Q5	O							O
EMPLOYEE	2Q1	O							
	2Q2		O		O				
	2Q3	O							
	2Q4	O					O		
	2Q5		O			O			
DEPARTMENT	3Q1	O							
	3Q2			O					
	3Q3			O					
	3Q4			O	O				
	3Q5			O					
	3Q6	O						O	
COMPANY	4Q1	O							
	4Q2	O			O				
	4Q3			O				O	

그림 11. 질의 분석

PERSON					
애트리뷰트	addr	pno	name	age	sex
addr	185	145	125	0	0
pno	145	210	150	0	0
name	125	150	235	85	45
age	0	0	85	125	85
sex	0	0	45	85	85

EMPLOYEE					
애트리뷰트	sal	eno	dpt	cmp	married
sal	135	110	70	0	0
eno	110	135	95	0	0
dpt	70	95	120	25	0
cmp	0	0	25	95	70
married	0	0	0	70	70

DEPARTMENT						
애트리뷰트	memNo	comp	burget	dname	manager	member
memNo	50	50	0	0	0	0
comp	50	75	25	25	0	0
burget	0	25	105	105	0	0
dname	0	25	105	140	35	35
manager	0	0	0	35	100	100
member	0	0	0	35	100	100

COMPANY			
애트리뷰트	president	cname	location
president	90	90	50
cname	90	180	140
location	50	140	140

그림 12. 클러스터 결합 행렬

4.2 타 기법과 비교

본 논문에서 제시한 클래스 스키마와 질의를 Ezeife[2]와 Karlalem[5]의 분할 기법에 적용하였으며 그 결과와 본 논문의 분할 기법을 비교하였다.

Ezeife 방법은 결합 행렬과 클러스터 행렬에 다른 클래스의 메소드가 포함됨으로 인하여 행렬의 크기가 커지며 불필요한 오버헤드와 복잡성이 증가된다 (그림 13 참조). 반면에 본 논문의 방법은 결합 및 클러스터 행렬과 분할 결과에 다른 클래스의 메소드와 애트리뷰트가 포함되지 않는다. 또한, Ezeife 방법은 중복되는 애트리뷰트를 한 플래그먼트에 할당시킴으로써 메소드는 있으나 그 메소드를 수행할 때에 필요한 애트리뷰트가 존재하지 않아 캡슐화에 위배되나 본 논문에서는 중복되는 메소드를 클래스 구성 계층상의 부모 클래스에 배치시킴으로써 캡슐화를 유지시켜 준다. 또한, Ezeife 방법은 키 애트리뷰트와 메소드를 중복시키고 있으나 본 논문에서는 키 값을 중복시키지 않았으며 재구성을 할 때에는 객체 식별자를 이용한다.

Karlapalem 방법은 사용자 질의에 관한 정보는 전혀 고려하지 않고 메소드 만을 가지고 분할하기 때문에 서로 다른 플래그먼트 내에 있는 애트리뷰트들을 함께 접근하는 질의가 자주 발생할 경우에 데이타 전

	PERSON	EMPLOYEE	DEPARTMENT	COMPANY
애트리뷰트 수	5	5	6	3
Ezeife	9×9	10×10	15×15	7×7
본 논문	5×5	5×5	6×6	3×3

그림 13. 클러스터 결합 행렬의 크기

송이 많아지는 문제점이 발생한다. 또한, 하나의 메소드를 선택하여 그 메소드에 의해 분할되기 때문에 여러 메소드의 동작이 함께 분할에 반영되지 않는다. 반면에 본 논문에서는 질의의 접근 빈도와 모든 메소드의 정보가 분할에 반영되므로 Karlapalem 방법에 비하여 정확성이 높다.

V. 결 론

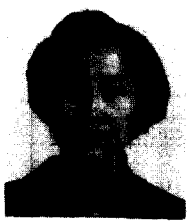
본 논문은 분산 객체 지향 데이타베이스에서 클래스를 수직으로 분할하는 방법을 연구하였다. 설계된 수직 분할 기법은 비중복 다중 분할이 지원하도록 설계되었으며 연구 내용은 다음과 같다. 첫째, 질의에서 사용된 메소드를 분할에 반영하기 위하여 질의 접근 행렬, 메소드 접근 행렬을 정의하였다. 둘째, 다른 클래스와의 관계를 고려한 분할 규칙을 생성하였다. 셋째, 다른 클래스와의 관계를 나타내기 위하여 질의 접근 행렬, 메소드 접근 행렬 및 애트리뷰트 사용 행렬을 전체 클래스 환경으로 확장하였다. 넷째, 메소드 플래그먼트를 생성하였다.

타 분석 기법과의 비교에 의하면, 분할 행렬의 크기가 줄었고 불필요한 애트리뷰트와 메소드가 포함되지 않았으며 캡슐화 개념을 유지시켜 준다.

추후의 연구 과제로서 분할된 플래그먼트를 여러 사이트에 할당하는 연구를 진행 중에 있다.

참 고 문 헌

1. Ceri, S. & Pelagatti, G. Distributd atabases: Principles and Systems. NY, McGraw Hill, 1984.
2. Ezeife, C. I. & Barker, K. Vertical Class Fragmentation in a Distributes Object Based System. TR 94-03, Univ. of Manitoba, 1993.
3. Hoffer, J. A., & Severance, D. G. The Use of Cluster Analysis in Physical Database Design. In 1st VLDB Conference, Framingham, Mass., 1975.
4. Karlapalem, K. & Li, Q. Partitioning Schemes for Object Oriented Database. In Fifth International Workshop on RIDE-DOM, 1995.
5. Karlapalem, K., Li, Q. & Vieweg, S. Method Induced Partitioning Schemes in OODB. In 16th int'l conf. on DCS, Hong Kong, 1996.
6. Karlapalem, K., Navathe, S. B. & Morsi, M. M. A. Issues in Distribution design of OODB. In Distributed Object Management, pages 148-164. Morgan Kaufmann Publishers, 1994.
7. McCormick, W. T. & Schweitzer, P. J., Problem Decomposition and Data Reorganization by a Clustering Technique. Oper. Res. 20, 1977.
8. Navathe, S. B., Ceri, S. Wiederhold, G. & Dou, J, Vertical partitioning algorithms for database design. in ACM TODS 9(4), 1984.
9. Navathe, S. B. and Ra, M. Vertical partitioning for database design: A graphical algorithm. In Proceedings of the ACM SIGMOD, 1989.
10. Ozsu, M. T. and Valduriez, P. Distributed Database System: Where Are We Now? IEEE Computer Vol. 24, No. 8, 1991.



이 순 미(Soon Mi Lee) 정회원

1984년 2월 : 이화여자대학교 수학과(학사)

1986년 2월 : 이화여자대학교 대학원 수학과 전자계산 전공(석사)

1992년 9월~현재 : 홍익대학교 대학원 전자계산학과 박사과정

※주관심분야 : 데이타베이스, 객체 지향 데이타베이스, 분산 시스템

임 해 철(Hae Chull Lim) 정회원

한국통신학회 논문지 제22권 제2호 참조