□ 研 究 論 文 □

韓國軍事運營分析學會誌
第 22 卷, 第 1 號, 1996. 6. 30

트랜잭션 중심의 발견적 파일 수직 분할 방법 (A transaction-based vertical partitioning algorithm)

박 기 택*, 김 재 련**

Abstract

In a relational database environment, partitioning of data is directly concerned with the amount of data that needs to be required in a query or transaction. In this paper, we consider non-overlapping, vertical partitioning. Vertical partitioning algorithm in this paper is composed of two phases. In phase 1, we cluster the attributes with zero-one integer program that maximize affinity among attributes. The result of phase 1 is called 'Initial Fragments'. In phase 2, we modify Initial Fragments that is not directly considered by cost factors, making use of a transaction-based partitioning method. A transaction-based partitioning method is partitioning attributes according to a set of transactions. In this phase we select logical accesses which needs to be required in a transaction as comparison criteria. In phase 2, proposed algorithm consider only small number of modification of Initial Fragments in phase 1. This algorithm is so insensible to number of transactions and of attributes that it can be applied to relatively large problems easily.

*대우자동차
**한양대학교

1. 서론

트랜잭션(transaction)/질의(query) 처리에 있어, 관심의 대상이 되는 비교적 적은 양의 정보를 검색(retrieve) 또는 갱신(update)하기 위해 많은 양의 데이터를 액세스할 필요가 종종 발생한다. 액세스의 지역성(locality of access)은 전체 릴레이션(relation)에 해당하기 보다는 그 부분집합에만 해당하기 때문에, 트랜잭션에 의해 자주 요구되는 속성들은 디스크에서 메인 메모리로 옮겨지는 페이지 수를 줄이기 위해 함께 그룹핑될 수 있다. 즉, 트랜잭션이 자주 함께 액세스하는 속성들을 물리적으로 같은 곳에 저장함으로써 검색되는 데이터량을 줄이는 것이다. 더욱이 릴레이션을 단편(fragment)으로 분할하므로써, 각각의 단편만을 액세스하는 트랜잭션들을 동시에 수행할 수도 있다.

본 연구는 데이터베이스의 릴레이션을 사용자의 액세스 요구에 따라 여러 개의 단편으로 분할함으로써, 트랜잭션/질의 처리 시 디스크로부터 검색되는 데이터의 양을 줄여 시스템의 성능을 향상시키는 것을 목적으로 한다.

데이터베이스 설계에서 분할(partitioning)이란 데이터베이스의 논리적 스키마로부터 논리적 객체(relation)를 몇 개의 물리적 객체(files)에 할당하는 일련의 과정이다[1]. 이것은 수직, 수평 분할 두 가지로 나눌 수 있다. 수직분할(vertical partitioning)은 속성들을 그룹으로 분할하는데, 분할된 객체는 원래 객체와 같은 속성들을 가진다.

수직 분할은 데이터베이스 설계 시 트랜잭션 처리 시간 향상을 위해 사용된다. 단편들은 비교적 적은 크기의 레코드로 구성되며, 따라서 트랜잭션 처리를 위해 디스크의 적은 페이지가 액세스되는 것이다. 자료를 메모리 계층(memory hierarchy)에 할당할 때도 수직 분할은 가장 많이 액세스되는 속성들을 가장 빠른 메모리에 저장하기 위해 사용된다. 분산 데이터베이스 환경에서는 단편들이 여러 사이트에 중복이 가능하게 할당된다. 수직 단편은 궁극적으로 어떤 물리적 파일 구조를 사용해서 데이터베이스에 저장된다. 그러나 단편들에 대한 실제적인 저장 구조로의 이행은 본 연구에서는 거론되지 않는다.

시스템의 성능 향상을 위하여 단편들은 트랜잭션의 요구와 매우 친밀하게 대응되어야 한다. 가장 이상적인 경우는 각각의 트랜잭션이 하나의 단편에 액세스하는 경우이다. 만약 어떤 속성들의 집합이 트랜잭션에 의해 항상 같이 사용된다면 설계 과정은 매우 단순하다. 그러나 현실 세계에서는 그러한 경우는 거의 발생하지 않으며, 열 개 정도의 속성을 가진 릴레이션을 분할하는 경우라도 수직 분할에 대한 시스템적 접근이 필요하다. [13]에서 지적하였듯이 m개의 속성을 가진 릴레이션은 B(m)가지로 분할될 수 있다. 여기서 B(m)은 m번째 Bell number로 큰 수 m에 대해 B(m)은 m^m에 접근한다. 비교적 적은 수 m에 대해서도 완전 열거법으로 이 문제를 푸는 것은 가능하지 않다.

수직 분할에 대한 기존 연구는 크게 두 가지 범주로 나눌 수 있다. 첫째로, 전형적이고 제한된 가정하에서 최적해를 구하는 것이다. 둘째로, 휴리스틱하게 접근하는 것이다. Hoffer[7]는 수직 분할을 결정하기 위해 비선형 0-1 정수 계획 모형을 개발하였다. 각각의 서브 파일에 주어진 용량제약하에서 저장, 검색, 갱신 비용을 최소화시키는 클러스터 분석

을 사용하여 해를 찾는다. Eisner and Severance[8]는 primary memory에 저장되어 있고 모든 사용자에 의해 액세스 가능한 집합과, secondary memory에 저장되어 각 사용자에 의해 추가 비용으로 검색되는 집합들 중에서 레크드를 세그먼팅하는 수리적 기법을 제시하였다. Hoffer and Severance[12]는 속성들을 조합해서 높은 친밀도를 갖는 것들을 그룹핑하는 앨고리듬을 개발하였다. 속성들간의 친밀도는 트랜잭션에 의해 참조되는 정도로 세 가지의 친밀도를 제시하였다. Navathe 등[1]은 위의 연구를 확장하여 2 단계 수직 분할 앨고리듬을 제안하였다. 즉, 첫째 단계인 직관적인 설계 단계와 , 둘째 단계인 비용에 근거한 설계 단계이다. 직관적인 설계 단계에서는 비용 요인에 대한 자세한 정보 없이 설계자가 설계에 대한 결정을 하도록 한다. 분할에 대한 입력 파라미터는 속성 사용 정보와 각 트랜잭션의 논리적 액세스 횟수(logical accesses frequency)이다. 먼저 속성 친밀도 행렬을 만드는 데 두 속성간의 친밀도는 다음과 같이 표현되다. $aff_{ij} = \sum_{k} acc_{kij}$. acc_{kij}는 속성 i와 속성 j를 동시에 참조하는 트랜잭션 k의 빈도수 이다. 그 다음 행렬을 가능한 한 block diagonal형태로 만들기 위한 행과 열을 조합하는 클러스터링 앨고리듬이 제안되었다. 행렬은 비관련 속성들의 액세스를 최소화시키기 위해 두 개의 집합으로 분리된다. 나누어진 세그먼트의 속성들은 중복될 수도 있다. 속성 친밀도 행렬로부터 얻은 두 속성 사이의 친밀도를 edge 값으로 나타낸 친밀도 그래프 방법도 제안되었다[2]. 속성 친밀도 행렬을 linearly connected spanning tree로 만든 후 하나의 사이클을 하나의 단편으로 보는 이 앨고리듬은 한 번에 모든 의미 있는 단편들을 생성해 낸다. Cornell and Yu[4]는 관계형 데이터베이스의 물리적 설계에 기반한 수직 분할법을 개발하였다. 속성들을 물리적인 세그먼트에 할당함으로써 디스크 액세스 수를 최소화시키는 최적 이진 분할 앨고리듬을 선형 정수 계획법을 사용하여 개발하였다. Chu[3]는 관계형 데이터베이스에서 트랜잭션에 근거한 새로운 수직 분할법을 제안하였다. 여기서는 트랜잭션별로 속성들을 분석하여 파일을 분할한다. 분지한계법에 기초한 최적 이진 분할 앨고리듬(OBP)과 트랜잭션의 수가 많을 때 적용하는 발견적 기법이 개발 되었다.

기존 연구에서는 주로 속성들의 친밀도를 크게 하는데 초점을 맞추거나, 인데스 등 액세스 방법을 고려한 이진 분할 앨고리듬을 사용하였다. 친밀도를 중심으로 다룬 연구에서는 비용을 제대로 고려하지 못하였고, 물리적 액세스량을 비용 함수로 고려한 이진 분할 앨고리듬에서는 릴레이션을 두 개의 단편으로만 분할하였다. 모든 경우에 있어 속성 수나 트랜잭션의 수에 따라 그 계산량이 급격히 증가하는 문제점이 있다. 본 연구는 속성간 친밀도를 최대화시키는 0-1 정수 모형으로 릴레이션을 초기 분할한 후, 트랜잭션에 근거하여 합리적인 방법으로 초기 분할된 단편들을 개선함으로써 속성 수, 트랜잭션 수에 큰 제약 없이 적용 가능 할 뿐 아니라, 여러 개의 단편으로도 분할 가능하게끔 한다.

본 연구의 구성은 2장에서 수직분할을 초기 설계와 트랜잭션에 기반한 설계로 나누어 설명한다. 3장에서는 다양한 액세스 패턴에 대한 앨고리듬 적용 예와 그 해를 보인다. 4장에서는 결론을 내리고 추후 연구 과제를 제시한다.

2. 수직 분할

데이터베이스는 릴레이션들로 구성되어 있고, 각 릴레이션은 길이가 알려진 속성들로 구성되어 있다. 또한 시스템의 주요 트랜잭션들도 미리 알려져 있으며, 이 트랜잭션들은 80-20 법칙을 따른다는 것을 가정한다. 80-20 법칙이란 20%의 주요 트랜잭션이 80%의 업무를 수행한다는 것으로 대부분의 시스템에 적용되고 있다. 수직 분할에 관련된 트랜잭션 상세는 아래와 같다.

① 트랜잭션의 발생 횟수(단위 시간당).

② 트랜잭션의 검색, 갱신이 필요한 속성들.

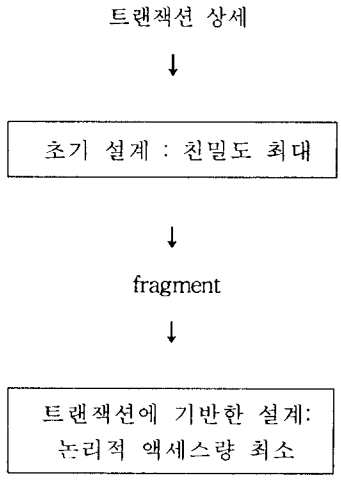
③ 트랜잭션이 한 번 수행될 때, 선택되는 릴레이션의 평균 인스턴스 수.

본 연구는 logical decision(단편의 구조와 구성에 관한 결정)에 중점을 두고 있으며 physical decision (각 단편에 대한 저장 구조나 액세스 방법에 관한 결정)은 이후에 고려될 수 있음을 가정한다. 그러나 설계자가 물리적인 비용 요소에 대한 약간의 사전 지식을 이용하게끔 허용한다. <그림 1>은 본 연구에서 제시한 전반적인 해법을 나타낸 것이다. 초기 설계는 트랜잭션의 논리적 액세스 횟수에 기반하여, 각 속성들을 클러스터링하는 0-1 정수 모형을 수립한다. 이 결과로 속성간 친밀도를 최대로 하는 초기 단편들이 생성된다. 속성간 친밀도가 높은 것끼리 같은 단편에 속하고, 낮은 것끼리는 서로 다른 단편에 속하게 된다. 특히 이 방법은 이전까지의 반복적인 분할 방법과는 달리 한 번에 최적해에 근사하게 도달할 수 있다.

트랜잭션에 기반한 설계에서는 비용 요소를 비교척도로 하여, 초기 단편을 트랜잭션 위주로 분석하여 개선시켜 나간다.

초기 설계 단계에서 "논리적 액세스"란 개념은 트랜잭션에 의해 검색되는 record occurrence를 의미한다. 이 개념은 일반적으로 개개의 속성을 메인 메모리로부터 불러오는 것은 불가능하고, 따라서 하나의 레코드가 불려질 때 그 속에 포함된 모든 속성들이 액세스 된다는 것을 모형화 한 것이다.

트랜잭션에 기반한 설계 단계에서는 구체적인 파일의 특성이나 데이터베이스 시스템의 구조에 의존하지 않은 비용 요소로 논리적인 액세스량을 고려하였다. 제시할 모델은 저장 구조(storage structure), 파일 구조(file organization), 액세스



<그림 1> 전반적인 해법

방법(access method), 트랜잭션 처리 전략(transaction processing strategy) 등의 구체적인 정보를 사용하여 확장될 수 있다. 그러나, 그러한 구체적인 모델은 특별한 환경을 위해서만 사용될 수 있

다. 여기서는 구체적인 물리적 환경에 대한 상세가 없는 일반적인 방법으로 분할 문제에 접근한다.

초기 설계는 속성간 친밀도를 이용하여 친밀도가 최대로 되는 수리 모형으로 최적해에 근사한 첫번째 단편을 만드는 것이다. 초기 설계를 위하여 Navathe 등[1]에 제시된 데이터를 입력값으로 사용한다. 먼저, 각 트랜잭션과 그에 필요한 속성, 그리고 그 빈도수를 나타낸 속성 사용 행렬(attribute usage matrix)로부터 속성 친밀도 행렬(attribute affinity matrix)를 구한다. 속성 친밀도 행렬은 그 (i, j)원소가 속성 i와 속성 j간의 친밀도를 표시하는 데, 친밀도란 속성 i와 j를 동시에 참조하는 트랜잭션의 총 접근 횟수로 나타내어진다. 친밀도를 이용한 이전 연구에서는 주로 속성들을 먼저 클러스터링한 후 경험적인 목적 함수를 적용하거나 수리적인 비용 함수에 기반한 이진 분할 방법이 사용되었다. 본 연구는 정수 모형을 제시함으로써 친밀도를 최대로하여 속성들을 분할한다. 이전 연구와 비교하여 이 모형의 주요 장점은 다음과 같다.

1) 반복적인 이진 분할이 필요 없다. Navathe 등[1]에서 사용된 반복적인 이진 분할의 주요 단점은 각 단계마다 계산량을 증가시키는 두 가지 새로운 문제가 발생한다는 것이다. 더구나, 앨고리듬의 종료가 목적 함수의 분별력에 의존한다.

2) Navathe 등[1]에서 처럼 경험적인 목적 함수를 사용할 필요가 없다. Cornell과 Yu[4]에서 지적하였듯이 Navathe 등에서 사용한 직관적인 목적 함수는 구체적인 시스템에서 실제 자세한 비용식이 사용되면 잘 적용되지 않는다.

3) Navathe 등의 SHIFT 앨고리듬과 같은 보충 앨고리듬이 필요하지 않다.

4) Navathe 와 Ra[2]의 앨고리듬은 친밀도가 높은 속성들끼리 같은 단편에 속하게 하는 반면, 본 연구에서 제시한 앨고리듬은 친밀도를 최대화시킴으로써 초기 단편이 최대한 최적해에 가깝도록 유도하였다.

<표 1>과 <표 2>는 각각 속성 사용 행렬과 속성 친밀도 행렬이다.

<표 1> 속성 사용 행렬

속성	1	2	3	4	5	6	7	8	9	10	액세스
길이	10	8	4	6	15	14	4	5	9	12	횟수
T1	1	0	0	0	1	0	1	0	0	0	25
T2	0	1	1	0	0	0	0	1	1	0	50
T3	0	0	0	1	0	1	0	0	0	1	25
T4	0	1	0	0	0	0	1	1	0	0	35
T5	1	1	1	0	1	0	1	1	1	0	25
T6	1	0	0	0	1	0	0	0	0	0	25
T7	0	0	1	0	0	0	0	0	1	0	25
T8	0	0	1	1	0	1	0	0	1	1	15

<표 2> 속성 친밀도 행렬

속성	1	2	3	4	5	6	7	8	9	10
1	75	25	25	0	75	0	50	25	25	0
2	25	110	75	0	25	0	60	110	75	0
3	25	75	115	15	25	15	25	75	115	15
4	0	0	15	40	0	40	0	0	15	40
5	75	25	25	0	75	0	50	25	25	0
6	0	0	15	40	0	40	0	0	15	40
7	50	60	25	0	50	0	85	60	25	0
8	25	110	75	0	25	0	60	110	75	0
9	25	75	115	15	25	15	25	75	115	15
10	0	0	15	40	0	40	0	0	15	40

초기 단편을 구하기 위한 정수 모형에 사용되는 기호는 다음과 같다.

n : 속성의 개수

F : 예상되는 최대 단편의 수

aff_{ij} : 속성 i와 속성 j간의 친밀도

aff_{ij}는 속성 친밀도 행렬의 i행, j열의 원소이다. 단, i=j 인 경우 속성 i와 속성 j 사이의 친밀도는 아무 의미가 없으므로 aff_{ij}의 값을 0으로 둔다. 다음은 결정 변수이다.

$$x_{ij} = \begin{cases} 1 & (\text{속성 } i\text{와 } j\text{가 같은 단편에 속하면}) \\ 0 & (\text{그렇지 않으면}) \end{cases}$$

<표 2>의 속성 친밀도 행렬의 i행, j열 원소를 aff_{ij}의 계수로하여 다음의 수리 모형을 수립한다. 이 모형으로 속성간 친밀도를 최대로 하는 초기 단편들이 생성된다.

$$\max \sum_{i=1}^{n} \sum_{j=1}^{n} aff_{ij} \times x_{ij} \qquad (1)$$

subject to

$$\sum_{j=1}^{n} x_{ij} = 1 \qquad \text{for } all\ i = 1, 2, \cdots\cdots, n \qquad (2)$$

$$\sum_{j=1}^{n} x_{ij} \leq F \qquad (3)$$

$$x_{ij} \leq x_{jj} \qquad \text{for } all\ i = 1, 2, \cdots\cdots, n,\ j = 1, 2, \cdots\cdots, n \qquad (4)$$

$$x_{ij} = 0,\ 1 \qquad (5)$$

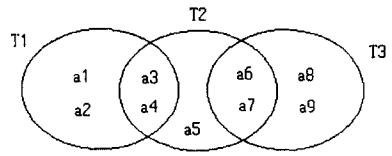
목적식 (1)은 속성 i와 속성 j간의 친밀도를 최대로 한다. 제약식 (2)는 속성 i는 오직 하나의 단편에만 속한다는 제약이고, (3)은 단편의 수는 F개 이하라는 제약이고, (4)는 단편이 생성되지 않으면 속성이 단편에 속할 수 없다는 제약이며, (5)는 0,1 정수 제약이다.

트랜잭션에 기반한 접근법은 Chu[3]가 제안한 방법이다. 개개의 속성을 조작 단위로 고려한 이전까지의 연구와는 달리, 하나의 트랜잭션에 의해 액세스되는 속성들을 하나의 조작 단위로 고려하였다. Chu는 분지한계법을 이용하여 최적 이진 분할 앨고리듬을 제안하였으나, 트랜잭션의 수가 많아지면 계산량이 급격히 증가하고 이진 분할만을 허용하므로 최적해에 도달하는 데에 한계가 있었다. 본 앨고리듬은 친밀도를 최대로 하는 초기 단편으로부터 트랜잭션에 기반한 분할 방법을 도입하여 적은 계산량으로 위의 단점을 해결하고자 한다.

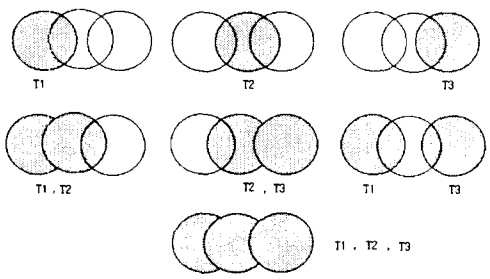
아홉 개의 속성으로 이루어진 릴레이션에 T1, T2, T3 세 개의 트랜잭션이 있다고 가정하자. 각 트랜잭션이 액세스하는 속성을 나타내는 표가 [표 3]에 있다. <그림 2>는 <표 3>의 속성들을 트랜잭션별로 나타낸 속성 교집합 그래프(attribute intersection graph)이다.

<표 3> 액세스 패턴

	a1	a2	a3	a4	a5	a6	a7	a8	a9
T1	1	1	1	1	0	0	0	0	0
T2	0	0	1	1	1	1	1	0	0
T3	0	0	0	0	0	1	1	1	1



<그림 2> 속성 교집합 그래프



<그림 3> Reasonable cuts

정의) self-contained fragment, T_i, 는 트랜잭션 i가 액세스하는 속성들의 집합이다. 그러한 self-contained fragment의 합집합 contained fragment라 한다. 속성을 두 개의 집합으로 나누는 이진 분할을 고려할 때, 두 집합 중 적어도 한 집합이 contained fragment 이면 이러한 이진 분할을 reasonable cut라 한다.

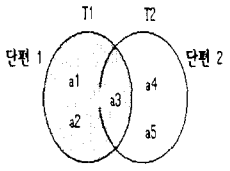
정의) resonable cut이 아닌 모든 이진 분할을 unreasonable cut이라 한다.

<그림 2>에서 트랜잭션 1에 대한 self-contained fragment T_1은 속성 a_1, a_2, a_3, a_4로 구성되어 있다. Self-contained fragment T_1과 T_2의 합집합이 contained fragment이다. <그림 3>은 <그림 2>에 대한 이진 분할 중 reasonable cut을 보여 주고 있다. Chu는 아래의 정리를 증명하여 reasonable cut에 의한 분할 앨고리듬을 개발하였다.

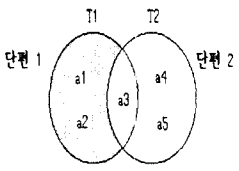
정리) X를 튜플 길이라 하고, $f_i(x)$를 트랜잭션 i에 대한 액세스 비용 함수라 하자. 만약, 모든 i에 대해서 $f_i(x)$의 이계도함수 $f''(x)<0$(concave downward)이면, 주어진 unreasonable cut에 대해 이 resonable cut의 비용보다 적거나 같은 비용이 드는 reasonable cut이 적어도 하나 존재한다.

Chu[3]는 이진 분할 시의 reasonable cut과 unreasonable cut을 정의 하였으나, 본 연구는 이것을 n분할로 확장하여 다음의 이론을 전개한다.

같은 트랜잭션에 사용된 속성들끼리 상호 친밀도가 높고, 초기 분할은 속성간 친밀도를 최대화시킨 것이므로, 초기 분할은 속성들이 트랜잭션별로 나누어 클러스터링 된다. 즉, 초기 분할은 항상 reasonable cut의 형태이다. 그러나 앞에서도 언급하였듯이 이러한 초기 분할은 속성간 친밀도만 최대로 할 뿐 디스크 액세스량을 직접적으로 고려하지 못하였기에 개선의 여지가 크다. 여기서는 논리적인 액세스량을 비교 척도로 하여 초기에 이루어진 단편들을 변화시킨다.



<그림 4(a)>



<그림 4(b)>

<그림 4(a)>의 속성 교집합 그래프를 고려해 보자. 트랜잭션 T1은 속성 a1, a2, a3를 액세스하고, 트랜잭션 T2는 속성 a3, a4, a5를 액세스 한다. 초기 설계 후의 분할은 아래와 같다고 가정하자.

단편 1 = { a_1, a_2, a_3 }

단편 2 = { a_4, a_5 }

두 단편을 모두 액세스하는 트랜잭션은 T2로 공통트랜잭션이라 한다. 초기 분할시엔 T1, T2 두 트랜잭션의 공통 속성인 a3는 속성 a1, a2와 함께 단편을 이루어 트랜잭션 T1을 수행하는 데 효율적이지만, 속성 a4, a5와 단편을 이룰 경우 트랜잭션 T2를 수행하는 데 효율적이다. 후자의 경우 단편은 다음과 같이 구성된다.

단편 1 = { a_1, a_2 }

단편 2 = { a_4, a_5, a_5 }

이것은 초기 단편을 합리적으로 적게 변화시키는 것으로 해가 개선될 가능성이 높으며 그 결과로 reasonable cut의 위치는 변화된다[<그림 4(b)>].

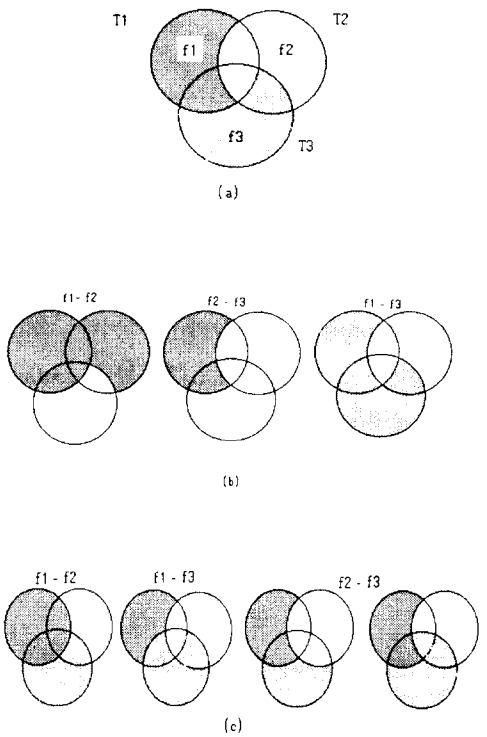
정의) 두 단편 사이에 공통트랜잭션이 존재하지 않으면 두 단편은 서로 독립이다.

정의) 두 단편 사이에 공통트랜잭션이 존재할 경우(독립이 아닐 경우), 단편의 구성을 새롭게 함으로써 reasonable cut의 위치를 바꾸는 것을 reasonable cut의 변화라 한다.

본 연구는 두 가지 경우의 reasonable cut의 변화를 고려하였다.

① 삭제 : 두 단편 조합 (i, j)가 서로 독립이 아닐 경우, 두 단편을 통합시킴으로써 reasonable cut을 변화시킨다 [그림 5(b)].

② 이동 : 두 단편 조합 (i, j)가 서로 독립이 아닐 경우, 공통 트랜잭션의 공통 속성을 상대편 단편에 속하게 함으로써 reasonable cut을 변화시킨다 [그림 5(c)].



<그림 5. (a) 초기단편 (b) reasonable cut 삭제 (c) reasonable cut 이동>

단편 조합 (i, j)가 서로 독립일 때 reasonable cut의 변화를 고려하지 않는 이유는 reasonable cut을 변화시켜도 그 변화로 인하여 하나의 단편만을 액세스하여서 처리될 수 있는 트랜잭션이 추가로 발생하지 않기 때문이다.

reasonable cut은 속성 교집합 그래프 상에서 쉽게 변화시킬 수 있다. 하지만 트랜잭션의 수가 증가하면 속성 교집합 그래프가 복잡해질 뿐 아니라 작성하기도 용이하지 않다. 트랜잭션들은 서로 다른 속성들을 액세스하지만 그들이 액세스하는 속성들 사이에 집합 관계가 존재할 수 있다. 이러한 집합 관계가 존재할 때, 한 트랜잭션이 다른 트랜잭션의 부분집합이거나 다른 트랜잭션의 합집합인 경우, 이 트랜잭션은 속성 교집합 그래프에 나타나지 않는다. 왜냐하면, 본 연구의 관심의 대상이 되는 것은 reasonable cut의 변화이고 다른 트랜잭션의 부분집합이나 합집합은 reasonable cut의 변화에 영향을 미치지 않기 때문이다.

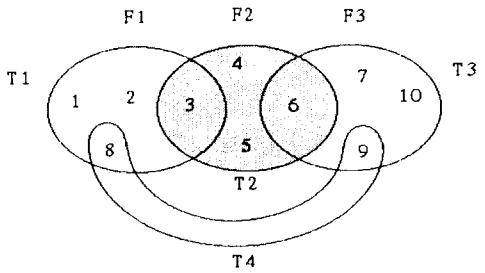
본 연구에서 초기해의 변화로 reasonable cut의 이동과 삭제를 고려하는 것은 함께 사용되는 속성들끼리 묶어 둔다는 파일 분할의 원리에 근거한다. 따라서, <그림 4>에서 a2를 단편 2에 속하게 하는 cut은 고려하지 않는다.

위에서 고려한 두 가지 reasonable cut의 변화는 적은 계산량으로 더 좋은 해를 찾기 위해 선택한 것이므로 더욱 다양한 reasonable cut의 변화(삭제 후 이동 등)를 생각해 볼 수도 있다.

위의 앨고리듬은 트랜잭션이 많거나 속성 교집합 그래프가 복잡한 경우에는 많은 수의 reasonable cut의 삭제와 이동을 고려해야 하므로, 선택적으로 변환 시 성능 향상에 영향을 크게 미칠 트랜잭션들만 고려할 필요가 있다. 이러한 트랜잭션을 '대상트랜잭션'이라 하고 아래의 Tcand 값이 큰 순서로 변환 대상트랜잭션을 선택한다.

$$T_{cand} = freq_i \times length_i$$

여기서 $freq_i$는 트랜잭션 i의 빈도수이고, $length_i$는 트랜잭션 i가 액세스하는 속성들의 길이의 합이다.



<그림 6> 대상트랜잭션

<그림 6>은 속성 교집합 그래프이며 F_i는 분할된 단편 i 이다. 그림에서 T_4의 T_{cana}값이 T_1, T_2, T_3의 T_{cana}값들보다 적을 경우, T_4가 단편 (1,3)간의 공통트랜잭션이지만 T_4에 대한 reasonable cut의 변화를 생략할 수 있다. 만약 T_4를 대상트랜잭션으로 선택한다면, 단편 (1,3)간 reasonable cut은 속성들을 {1,2}, {7,8,9}로 분할하거나, {1,2,8,9}, {7}로 분할하는 것으로 변화될 수 있다.

본 연구가 제시한 파일의 수직 분할 방법론을 정리하면 다음과 같다.

초기 설계
• 속성 친밀도 행렬 작성 • 친밀도를 최대로 하는 0-1 정수 모형 수립

↓

초기 단편 생성

↓

트랜잭션에 기반한 설계
• 속성 교집합 그래프 작성 • reasonable cut 삭제 • reasonable cut 이동

↓

최종 단편 생성

시스템에서 디스크 I/O의 수는 트랜잭션 액세스 패턴, 액세스 빈도, 액세스 방법에 달려 있다. 모든 트랜잭션에 대한 디스크 I/O 수는 각 트랜잭션에 의한 디스크 I/O의 합이다. 수직 분할의 목적은 릴레이션을 여러 개의 단편으로 나눔으로써 총 디스크 액세스 수를 최소화시키는 것이다. 즉,

$$\min \sum_{i=1}^{n}\sum_{j=1}^{d} f_i(l(I+F_j))$$

여기서 d는 분할 후 생성된 단편의 수, I는 원래의 튜플을 확인하기 위해 모든 단편의 각 튜플에 덧붙여지는 primary key 혹은 tuple identifier이다. $l(I+F_j)$는 단편 F_j의 속성과 I의 길이의 합이고, $f_i(l(I+F_j))$는 트랜잭션 i가 단편 F_j에 액세스하는 비용함수이다.

본 연구에서 제시한 초기 단편과 reasonable cut 변화 후의 단편을 비교할 비용 함수는 논리적인 액세스량이며 아래와 같이 표현되다.

$$Cost = \sum_{i=1}^{n}\sum_{j=1}^{d} l(I+F_j)\times freq_i \times SIZE, \quad A_i \cap F_j \neq \emptyset \qquad (6)$$

A_i는 트랜잭션 i에 의해 액세스 되는 속성들의 집합이고, $freq_i$는 트랜잭션 i의 빈도수이며, SIZE는 릴레이션의 튜플수이다.

위의 비용 함수는 논리적인 설계 결정만을 가능하게 하지만, 저장 구조, 파일 구조, 액세스 방법이 구체적으로 명시된 시스템에서는 이들을 고려하여 물리적인 설계 결정을 내릴 수 있다.

3. 수치예제 및 평가

예제 1과 2는 Navathe 등[1]에서 다룬 예제로서 본 연구에서 제시한 앨고리듬을 설명하기 위해 사용하였다. 예제 3은 다양한 형태의 속성 사용 행렬과 그에 대한 앨고리듬 적용 결과를 나타내고 있다.

1) 예제 1 : 속성 10개, 트랜잭션 8개인 경우

■ 초기 설계

<표 2>의 속성 친밀도 행렬을 계수로 하여 0-1 정수 모형을 수립한 후, LINDO 패키지로 초기 단편을 구한다. 다음은 F=10일 때의 해이다.

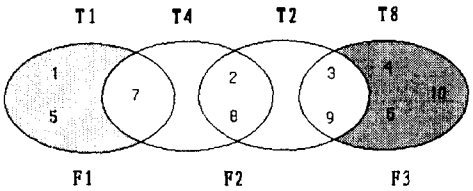
$$X_{3,2} = X_{4,10} = X_{5,1} = X_{6,10} = X_{7,2} = X_{8,2} = X_{9,2} = X_{1,1} = X_{2,2} = X_{10,10} = 1$$

나머지 변수는 모두 0이다.

이것은 다음의 초기 단편을 생성시킨다.

단편 1: { 1, 5 }, 단편 2 : { 2, 3, 7, 8, 9 }, 단편 3 : { 4, 6, 10 }

이 결과를 <그림 2>의 속성 사용 행렬을 이용하여 속성 교집합 그래프를 그리면 <그림 7>과 같다. <그림 7>에서 다른 트랜잭션의 부분집합인 T_3, T_6, T_7과, 합집합인 T_5는 표현하지 않았다.



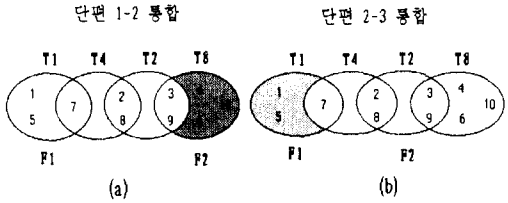
<그림 7> 속성 교집합 그래프

■ 트랜잭션에 기반한 설계

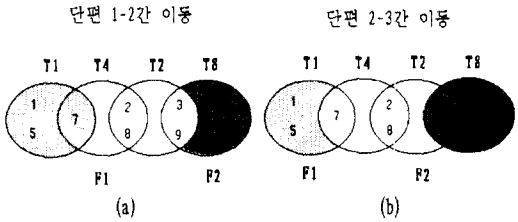
이 단계에서는 초기 단편을 reasonable cut을 변화시킴으로써 해를 발견적으로 개선시킨다. 식 (6)이 비용으로 사용된다. 속성 1을 primary key로 SIZE는 100으로 하였다.

1. 삭제 : 단편 1,2 통합과 단편 2,3 통합
   두 가지[그림 8].
2. 이동 : 단편 1-2간과 단편 2-3간
   reasonable cut 이동 두 가지 <그림 9>

위에서 단편 1-3간 reasonable cut의 변화는 두 단편이 독립이므로 고려하지 않았다.



<그림 8> reasonable cut 통합



<그림 9> reasonable cut 이동

<표 4> 분할 간 비용 분석

초기 단편 비용		1,038,000	
reasonable cut 변화			
삭 제		이 동	
단편1-2간	단편2-3간	단편1-2간	단편2-3간
1,248,000	1,607,500	1,016,000	1,308,500

각각의 대안에 대해 식(6)을 적용하면 <표 4>의 결과를 얻는다. 초기 단편은 비용이 1,038,000 이고, 가장 적은 비용은 단편 1-2간 reasonable cut을 이동시켰을 때인 1,016,000이다. 위 표는 <그림 8(a)> (단편 1-2 간 reasonable cut 이동)로 분할하는 것이 가장 효율적임을 보여 준다.

2) 예제 2 : 속성 20개, 트랜잭션 15개인 경우

■ 초기 설계

사용된 속성 사용 행렬은 <표 5>이다.

LINDO를 이용한 정수 모형의 해는 다음과 같다.

$$X_{2,12}=X_{3,11}=X_{4,1}=X_{6,1}=X_{7,11}=X_{8,1}=X_{9,12}$$
$$=X_{10,11}=X_{13,12}=X_{14,12}=X_{16,15}=X_{17,11}=X_{18,11}$$
$$=X_{19,15}=X_{20,15}=X_{1,1}=X_{11,11}=X_{12,12}=X_{15,15}=1$$

<표 5> 속성 사용 행렬(속성 20개, 트랜잭션 15개)

속성	1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0	액세스 횟수
길이	8 8 8 8 4 8 8 2 0 2 0 2 4 8 6 5 3 0 2 8 6 6	
T1	1 0 0 1 1 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0	50
T2	0 1 0 0 0 0 0 0 0 1 0 0 1 1 1 0 0 0 0 0 0	50
T3	0 0 1 0 0 0 1 0 0 1 1 0 0 0 0 0 1 1 0 0	50
T4	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 1 1	50
T5	1 0 0 0 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0	15
T6	1 0 0 0 0 0 0 1 0 1 1 0 0 0 0 0 0 0 0 0	15
T7	0 0 1 0 0 0 1 0 0 1 1 0 0 0 0 0 1 0 0 0	15
T8	0 1 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 0 1 0 1	15
T9	0 1 0 0 1 0 0 0 0 0 1 0 0 1 0 0 0 0 1 0	10
T10	1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 1 0 0	10
T11	1 1 1 0 1 1 0 0 1 0 0 1 1 0 0 0 0 0 0 0	10
T12	0 0 0 1 0 0 1 0 0 1 0 0 0 1 0 0 0 0 1 1	10
T13	0 0 0 0 0 0 0 1 0 0 1 0 0 0 1 1 1 1 0 0	10
T14	1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0	5
T15	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1	5

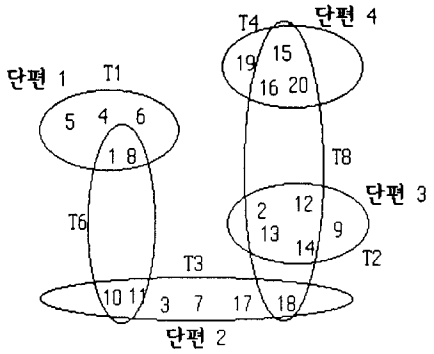
위 결과는 다음의 초기 단편을 형성한다.

단편 1 : { 1, 4, 5, 6, 8 }
단편 2 : { 1, 3, 7, 10, 11, 17, 18 }
단편 3 : { 1, 2, 9, 12, 13, 14 }
단편 4 : { 1, 15, 16, 19, 20 }

<그림 10>은 위에서 얻은 초기 단편을 나타내는 속성 교집합 그래프이다. 다른 트랜잭션의 부분 트랜잭션인 T_5, T_7과 액세스 횟수가 적은 T_9이하의 트랜잭션에 대해서는 reasonable cut 변화를 고려하지 않았다.

■ 트랜잭션에 기반한 설계

속성 1을 primary key로 SIZE를 1로하여 각 reasonable cut의 변화에 대한 비용을 <표 6>에 나타내었다. <표 6>을 보면 초기 단편의 비용이 28,405로 가장 좋음을 알 수 있다. 위 두 예제의 결과는 Navathe 등[1]의 것과 동일하다.



<그림 10> 속성 교집합 그래프

<표 6> 분할 간 비용 분석

초기 단편	reasonable cut 변화							
	삭 제				이 동			
	단편 1-2간	단편 2-3간	단편 2-4간	단편 3-4간	단편 1-2(a)	단편 1-2(b)	단편 2-3간	단편 2-4간
28,000	34,320	34,815	34,625	33,350	32,720	30,140	30,315	28,870

3) 예제 3

여기서는 다양한 액세스 패턴을 가진 몇 가지 예제에 대한 앨고리듬의 적용 사례를 보인다. 단, 속성 교집합 그래프에서 트랜잭션이 액세스 횟수가 적은 경우, 다른 트랜잭션의 부분집합이나 합집합인 경우, 여러 단편에 걸쳐 있는 경우는 고려하지 않았다.

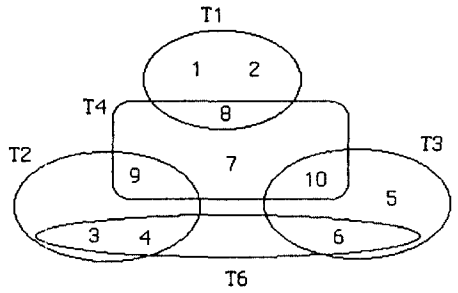
a. 주요 트랜잭션들 간에 공통으로 액세스하는 속성이 없는 경우 - 트랜잭션 T1, T2, T3 [그림 11]

$$X_{1,2}=X_{4,3}=X_{5,10}=X_{6,10}=X_{7,10}=$$
$$X_{8,2}=X_{9,10}=X_{2,2}=X_{3,3}=X_{10,10}=1$$

초기단편 :{ 1, 2, 8 }, { 3, 4 }, { 5, 6, 7, 9, 10 }

<표 7> 속성 사용 행렬 (예제 3-a)

속성	1	2	3	4	5	6	7	8	9	10	액세스 횟수 / 단위 시간
길이	10	8	4	13	3	7	12	5	10	8	
T1	1	1	0	0	1	0	0	1	0	0	50
T2	0	0	1	1	0	0	0	0	1	0	50
T3	0	0	0	0	1	1	0	0	0	1	50
T4	0	0	0	0	0	0	1	1	1	1	50
T5	1	1	0	0	1	0	1	1	0	0	30
T6	0	0	1	1	0	1	0	0	0	0	30
T7	0	1	0	0	0	0	1	1	0	0	20
T8	1	0	0	0	1	1	1	0	1	1	20



<그림 11> 속성 교집합 그래프 (예제 3-a)

⇒개선후: { 1, 2, 7, 8 }. { 3, 4, 9 }, { 5, 6, 10 }

위의 결과로부터 서로 간에 공통으로 액세스하는 속성이 없는 트랜잭션 T1, T2, T3는 각각 하나의 단편을 형성함을 알 수 있다.

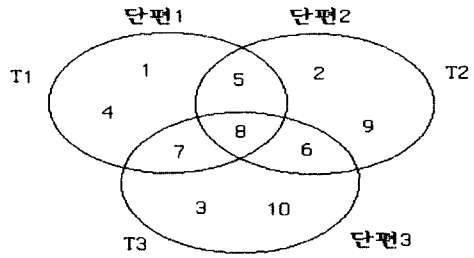
b. 주요 트랜잭션들이 공통으로 액세스하는 속성들이 적고 서로 겹쳐 있는 경우 [그림 12].

<표 8> 속성 사용 행렬 (예제3-b)

속성	1	2	3	4	5	6	7	8	9	10	액세스 횟수 / 단위 시간
길이	10	4	5	12	9	14	8	4	8	5	
T1	1	0	0	1	1	0	1	1	0	0	50
T2	0	1	0	0	1	1	0	1	1	0	40
T3	0	0	1	0	0	1	1	1	0	1	30
T4	1	0	0	1	0	0	1	1	0	0	40
T5	0	1	0	0	1	0	0	0	1	0	40
T6	0	0	0	0	0	0	1	1	0	0	30
T7	0	1	1	0	0	1	0	0	1	1	20
T8	0	0	1	0	0	0	0	0	0	1	50

$$X_{3,10}=X_{4,1}=X_{5,1}=X_{6,1}=X_{7,1}=$$
$$X_{8,1}=X_{9,2}=X_{1,1}=X_{2,2}=X_{10,10}=1$$

초기 단편:{ 1, 4, 5, 6, 7, 8 }, { 2, 9 }, { 3, 10 }



<그림 12> 속성 교집합 그래프 (예제 3-b)

⇒ 개선후 단편: { 1, 4, 5, 7, 8 }, { 2, 6, 9 }, { 3, 10 }

위의 경우에는 액세스 패턴에 따라 단편의 구성이 달라진다.

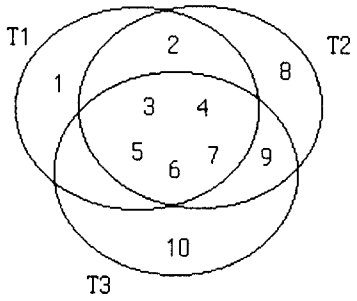
c. 주요 트랜잭션들이 공통으로 액세스하는 속성이 많고 서로 겹쳐 있는 경우 [그림 13].

<표 9> 속성 사용 행렬 (예제3-c)

속성	1	2	3	4	5	6	7	8	9	10	액세스 횟수
길이	10	8	4	13	3	7	12	5	10	8	/ 단위 시간
T1	1	1	1	1	1	1	1	0	0	0	50
T2	0	1	1	1	1	1	1	1	1	0	50
T3	0	0	1	1	1	1	1	0	1	1	50
T4	1	1	0	0	0	0	0	1	1	0	30
T5	0	1	0	0	0	1	1	0	0	1	30
T6	1	0	0	0	1	1	1	0	0	1	30
T7	1	0	1	0	1	0	1	0	1	0	20
T8	0	1	0	1	0	1	0	1	0	1	20

$$X_{1,7}=X_{2,7}=X_{3,7}=X_{4,7}=X_{5,7}=$$
$$X_{6,7}=X_{8,7}=X_{9,7}=X_{10,7}=X_{7,7}=1$$

초기 단편 : { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 }



<그림 13> 속성 교집합 그래프 (예제 3-c)

⇒ 개선 후 단편: { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 }

위의 경우에 트랜잭션 T1, T2, T3가 공통으로 액세스하는 속성들이 많으므로 분리되지 않는다.

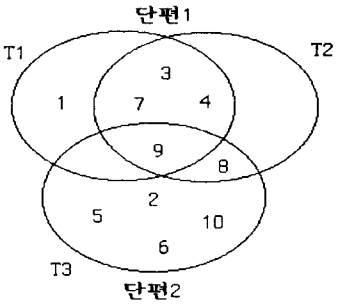
d. 공통으로 액세스하는 속성이 많은 트랜잭션과 공통으로 액세스하는 속성이 적은 트랜잭션이 함께 있는 경우 [그림 14].

<표 10> 속성 사용 행렬 (예제 3-d)

속성	1	2	3	4	5	6	7	8	9	10	액세스 횟수
길이	10	8	4	13	3	7	12	5	10	8	/ 단위 시간
T1	1	0	1	1	0	0	1	0	1	0	50
T2	0	0	1	1	0	0	1	1	1	0	40
T3	0	1	0	0	1	1	0	1	1	1	30
T4	1	0	1	1	0	0	0	1	0	0	40
T5	0	0	1	1	0	0	1	0	0	0	40
T6	0	1	0	0	0	0	0	1	1	0	30
T7	0	0	1	1	0	0	0	1	1	0	20
T8	0	1	0	1	0	1	0	1	0	1	20

$$X_{2,6}=X_{3,1}=X_{4,1}=X_{5,6}=X_{7,1}=$$
$$X_{8,1}=X_{9,1}=X_{10,6}=X_{1,1}=X_{6,6}=1$$

초기 단편 : { 1, 3, 4, 7, 8, 9 }, { 2, 5, 6, 10 }



<그림 14> 속성 교집합 그래프 (예제 3-d)

⇒개선후 단편:{ 1, 3, 4, 7, 8, 9 }, { 2, 5, 6, 10 }

위에서 공통으로 액세스하는 속성이 많은 T1, T2는 합쳐서 하나의 단편을 형성하고, 그렇지 않은 T3는 따로 하나의 단편을 형성한다. 예제 3의 단편 형성의 특징을 정리하면 아래와 같다.

3-a. 공통으로 액세스하는 속성이 없는 트랜잭션은 각각 하나의 단편을 형성한다.

3-b. 트랜잭션 간 공통으로 액세스하는 속성

이 적은 경우, 액세스 패턴에 따라 단편 구성이 달라진다.

3-c. 공통으로 액세스하는 속성이 많으면 하나의 단편을 이룬다.

3-d. 공통으로 액세스하는 속성이 많은 트랜잭션들은 통합되어 하나의 단편을 형성하고, 그렇지 않은 경우는 따로 다른 단편을 형성한다.

4. 결론

이상에서 우리는 트랜잭션에 기반한 파일 수직분할에 대하여 논의 하였다. 본 연구가 제시한 수직분할은 초기 설계와 트랜잭션에 기반한 설계로 구성되어 있다. 초기 설계에서는 속성간 친밀도를 최대로 하는 0-1 정수 모형을 사용해서 반복적인 작업없이 최적해에 근사한 초기 단편을 생성해 낸다. 트랜잭션에 기반한 설계에서는 논리적 액세스량을 비용 함수로하여 초기 단편을 트랜잭션에 근거한 분할 방법으로 개선한다. 물론 저장 구조나 액세스 방법이 명시된 시스템에서는 비용 함수로 사용한 논리적 액세스량을 물리적인 측면을 고려한 함수로 대체시킬 수 있다. 특히, 제시된 앨고리듬은 파일 분할 문제의 계산량에 가장 큰 영향을 미치는 속성 수나 트랜잭션 수에 매우 둔감하므로 큰 문제를 비교적 빠른 시간 내에 해결할 수 있다. 추후 연구 과제로는 여러 개의 릴레이션을 동시에 고려할 때의 분할과 단편간 속성의 중복을 허용하는 중복 분할의 연구, 그리고 최적해에 근접하는 보다 효율적인 휴리스틱 해법 개발이 필요하다.

참 고 문 헌

[1] Navathe, S., Ceri, S., Wiederhold, G. and Dou, J., "Vertical partitioning Algorithms for Database Design," ACM Trans. Database Systems, vol. 9, no. 4, pp. 690-710, 1984.

[2] Navathe S. and Ra, M., "Vertical Partitioning for Database Design: A Graphical Algorithm," *in Proc. ACM SIGMOD Int. Conf. Management Data*, 1989.

[3] Chu, W. and Ieong, I., "A Transaction-Based Approach to Vertical Partitioning for Relational Database Systems," *IEEE Trans. Software Eng.*, vol. 19, no. 8, pp. 804-812, 1993.

[4] Cornell D.W. and Yu, P.S., "An Effective Approach to Vertical Partitioning for Physical Design of Relational Database," *IEEE Trans. Software Eng.*, vol. 16, no. 2, pp. 248-258, 1990.

[5] March, S.T., "Techniques for Structuring Database Records," ACM Compt. Surveys, vol. 15, no. 1, pp. 45-79, 1983

[6] Kusiak, A., "The Generalized Group Technology Concept," Int. J. Prod. Res., vol. 25, no. 4, pp. 561-569, 1986

[7] Hoffer J., "An Integer Programming Formulation of Computer Database Design Problems," *Inform. Sci.* vol. 11,

pp.29-48, 1976.

[8] Eisner M. and Severance, D., "Mathematical techniques for efficient record segmentation in large shared database," *J. ACM,* vol. 23, no. 4, pp. 619-635, 1976.

[9] Ceri, S., Navathe, S.B. and Wiederhold, G., "Distribution Design of Logical Database Schemas," IEEE Trasn. Software Eng., vol. SE-9, no. 4, pp. 487-504, 1983

[10] Navathe, S., Ra, M., Varadarajan, R. Karlapalem, K. and Sreewastav, K., "A Mixed Partitioning Methodology for Distributed Database Design," UF-CIS TR 90-17, Univ. Florida, 1990

[11] Elmasri R. and Navathe, S.B, *Fundamentals of database systems.* Benjamin/Cummings, 1989.

[12] Hoffer, J. and Severance, D., "The use of cluster analysis in physical database design," Proc. First VLDB, 1975

[13] Hammer, M. and Niamir, B., "A heuristic approach to attribute partitioning," Proc. ACM SIGMOD Intl. Conf. Management of Data(Boston, Mass. 1979), ACM, New York.