A Formal Approach to the Vertical Partitioning Problem in Distributed Database Design

J. Muthuraj[1]
S. Chakravarthy[2]
R. Varadarajan

Computer and Information Sciences Department
University of Florida, Gainesville, FL 32611, USA
Email: sharma@snapper.cis.ufl.edu

S. B. Navathe[1]

College of Computing
Georgia Institute of Technology
Atlanta, GA 30332

Abstract

The design of distributed databases is an optimization problem requiring solutions to several interrelated problems: data fragmentation, allocation, and local optimization. Each problem can be solved with several different approaches thereby making the distributed database design a very difficult task.

Although there is a large body of work on the design of data fragmentation, most of them are either *ad hoc* solutions or formal solutions for special cases (e. g., binary vertical partitioning). In this paper, we address the problem of n-ary vertical partitioning problem and derive an objective function that generalizes and subsumes earlier work. The objective function derived in this paper is being used for developing heuristic algorithms that can be shown to satisfy the objective function. The objective function is also being used for comparing previously proposed algorithms for vertical partitioning. We first derive an objective function that is suited to distributed transaction processing and then show how it can be extended to include additional information, such as transaction types, different local and remote accessing costs and replication. Finally, we indicate the current status of implementation.

1 Introduction

The design of distributed databases is an optimization problem requiring solutions to several interrelated problems: data fragmentation, allocation, and local optimization. Each problem phase can be solved with several different approaches thereby making the distributed database design a very difficult task. Traditionally database design has been heuristic in nature. Although the metric being optimized is not stated quantitatively, it is implicitly assumed to be the *processing cost* for a given set of *important* transactions that constitute the bulk of transaction load for the given database.

[1] This work is supported by the National Science Foundation Research Initiation Grant IRI-8716798
[2] This work is supported by the National Science Foundation Research Initiation Grant IRI-9011216

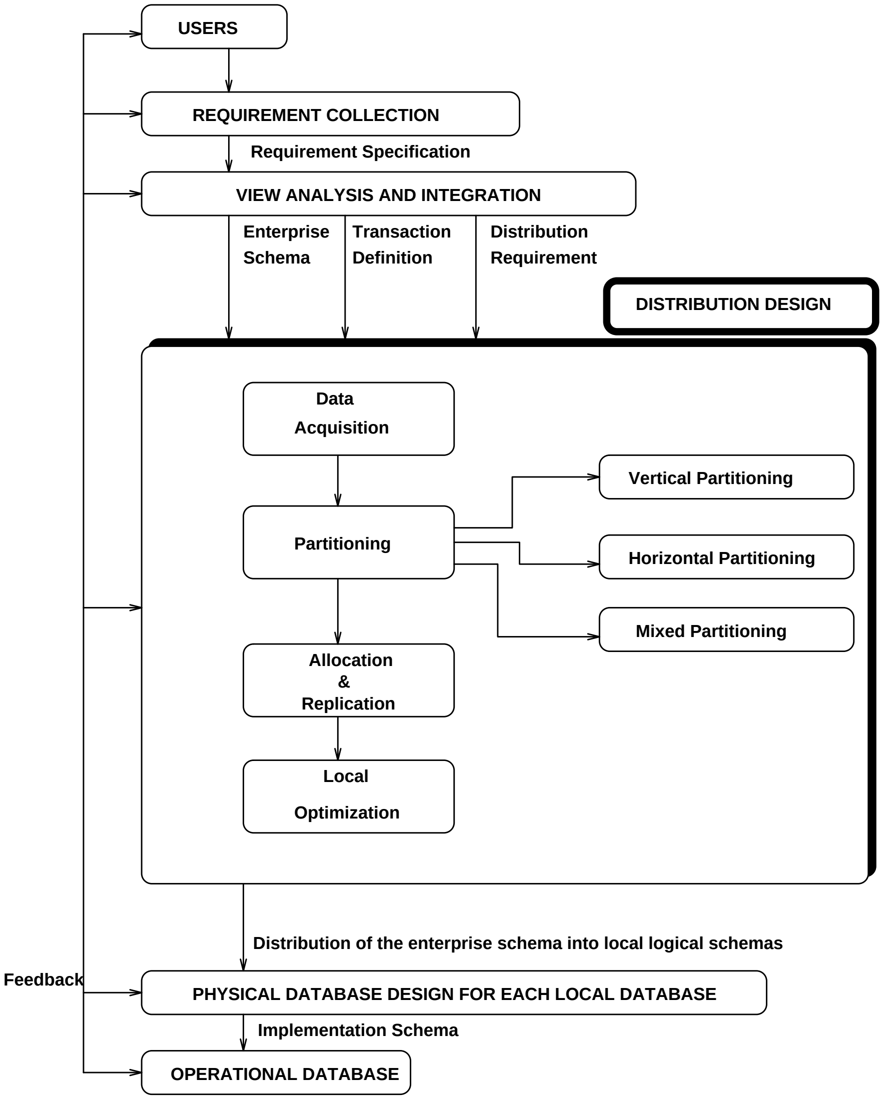

Figure 1: Distributed Database Design Methodology

Figure 1 gives an outline of the overall distributed database design methodology [2]. Distributed database design deviates from conventional non-distributed database design only in the distribution aspect which is highlighted by the box titled distribution design in figure 1. The distribution design involves data acquisition, partitioning of the database, allocation and replication of the partitions and local optimization. Partitioning of the database is done in several ways: vertical, horizontal, and hybrid (also called mixed). Our long-term objective is to develop a distributed database design testbed in which different algorithms for various components of distribution design can be mixed and matched. This work is a first step in that direction and addresses the partitioning (or fragmentation) problem.

In this paper, we delimit our discussion to one of the data fragmentation problems, namely the vertical partitioning problem. More information on this vertical partitioning problem, can be found in Muthuraj's thesis [14]. Vertical Partitioning (also called attribute partitioning) is a technique that is used during the design of a database to improve the performance of transactions [15]. In vertical partitioning, attributes of a relation R[3] are clustered into non-overlapping[4] groups and the relation R is projected into fragment relations according to these attribute groups. In distributed database systems, these fragments are allocated among the different sites. Thus the objective of vertical partitioning is to create vertical fragments of a relation so as to minimize the cost of accessing data items during transaction processing. If the fragments closely match the requirements of the set of transactions provided, then the transaction processing cost could be minimized. Vertical partitioning also has its use in partitioning individual files in centralized databases, and dividing data among different levels of memory hierarchies etc. [15, 18]. In the case of distributed database design, transaction processing cost is minimized by increasing the local processing of transactions (at a site) as well as by reducing the amount of accesses to data items that are not local. The aim of vertical partitioning technique (and in general data partitioning techniques) is to find a partitioning scheme which would satisfy the above objective.

It should be noted that the problem of partitioning can be addressed at various levels of detail by taking additional information into consideration. Figure 1 clearly distinguishes various levels and a feedback path is provided to refine the outcome of the earlier levels if it does not suit the objectives of the next level. In this paper, we are taking only transaction information as input to keep the problem manageable. In essence, the global optimization problem (which includes a large number of parameters and a very complex metric) is partitioned into several smaller optimization problems to reduce the search space and the complexity of each problem. Other detailed information need to be considered on the outcome of this stage in order to obtain a design at the physical level.

Several vertical partitioning algorithms have been proposed in the literature. Hoffer and Severance [7] measure the affinity between pairs of attributes and try to cluster attributes according to their pairwise affinity by using the *bond energy algorithm* (BEA) [12]. Hammer and Niamir [6] use a file design cost estimator and a heuristic to arrive at a "bottom up" partitioning scheme. Navathe, et al [15] extend the BEA approach and propose a two phase approach for vertical partitioning. Cornell and Yu [4] apply the work of Navathe [15] to physical design of relational databases. Ceri, Pernici and Wiederhold [3] extend the work of Navathe [15] by considering it as a 'divide' tool and by adding a 'conquer' tool. Navathe and Ra [16] construct a graph-based algorithm to the vertical partitioning problem where the heuristics used includes an intuitive objective function which is not explicitly quantified. In addition to these vertical partitioning algorithms, there are many data clustering techniques [9], traditionally used in pattern recognition and statistics, some of which can be adapted to partitioning of a database. These data clustering algorithms include Square-error clustering [9], Zahn's clustering [22], Nearest-neighbor clustering [11] and Fuzzy [9]clustering.

The partitioning algorithms mentioned above use some heuristics to create fragments of a relation. The input to most of these algorithms is an Attribute Usage Matrix *(AUM).* AUM is

[3] For most of our discussion it does not matter whether R is a Universal relation or a relation that is in some normal form as long as key attributes are identified. In the section on extensions, we discuss how we can take that information into account.

[4] Overlapping partitions (of non-primary key attributes) may also be considered when availability is an important criterion. This is discussed in the section on extensions.

a matrix which has attributes as columns, and transactions as rows and the access frequency of the transactions as values in the matrix. Most of the earlier data fragmentation algorithms use an Attribute Affinity Matrix (*AAM*) derived from the AUM provided as input. An AAM is a matrix in which for each pair of attributes, the sum total of frequencies of transactions accessing that pair of attributes *together* is stored. The results of the different algorithms are sometimes different even for the same attribute affinity matrix indicating that the objective functions used by these algorithms are different. Most of the proposed vertical partitioning algorithms do not have an objective function to evaluate the "goodness" of partitions that they produce. Also, there is no common criterion or objective function to compare and evaluate the results of these vertical partitioning algorithms.

1.1 Contributions

This paper makes several contributions to the problem of data fragmentation in general and the design of vertical partitioning in particular [14]. Specifically:

1. We have, perhaps for the first time, studied the applicability of some data clustering algorithms for distributed database design[5] proposed in areas such as pattern classification, statistics etc., [9], [22], [11], to data fragmentation problem. In fact, we start from one such objective function proposed for data clustering and modify and extend it to the specific problem at hand.
2. We have formulated an objective function for n-ary partitions, with two components that provide the desirable behavior for minimizing transaction processing cost.
3. Finally, we are using the approach of formulating an objective function (termed Partition Evaluator in this paper) before developing (heuristic) algorithms for the partitioning problem. This approach enables us to study the properties of algorithms with respect to an agreed upon objective function, and also to compare different algorithms for "goodness" using the same criteria. The objective function formulated in this paper is a step in this direction. Moreover, the objective function derived in this paper can be easily extended to include additional information (e. g., query types – retrieval/update, allocation information about the partitions, remote processing cost, and the transaction usage pattern at any particular site). Some of these extensions are discussed at end of the paper.

Our long-term objective is to either extend this objective function or to develop new objective functions to take into account additional information pertaining to replication, storage, and transmission costs that are critical to a distributed environment. However, we view this work as filling a void that currently exists even at the conceptual level.

The organization of the paper is as follows. Section 2 discusses previous related work on data clustering. Section 3 describes prior work in the area of vertical partitioning and summarizes the need for the development of our Partition Evaluator. In section 4, we derive the Partition

[5] Schkolnik [18] uses data clustering techniques for partitioning a hierarchical structure for an IMS database using detailed cost information which is different from the problem addressed in this paper.

Evaluator (PE) appropriate for distributed database design at the conceptual level. In section 5, we illustrate the use of our Partition Evaluator with an example. We show the actual behavior of the PE and compare it with the expected behavior. Section 6 describes how the PE is amenable to extensions with illustrations and includes the current status of implementation. We show the actual behavior of the PE and compare it with the expected behavior. Section includes summary and future work.

2 Previous Work

In this section we briefly summarize the work done in the area of data clustering and the work done in data fragmentation and bring out the similarities and differences between them.

A number of data clustering algorithms have been developed in application areas such as statistics and pattern classification and analysis which address the problem of grouping or clustering data using various criteria. The most commonly used partitioning clustering strategy is based on the square-error criterion [9]. The general objective is to obtain that partition which, for a fixed number of clusters, minimizes the square-error. Minimizing square-error, or within-cluster variation, has been shown to be equivalent to maximizing the between-cluster variation. Clusters can also be viewed as regions of the attribute pattern space in which the patterns are dense, separated by regions of low attribute pattern density. In the mode-seeking partitioning algorithm due to Torn [21], clusters are identified by searching for regions of high density, called modes, in the pattern space. Each mode is associated with a cluster center and each pattern is assigned to the cluster with the closest center.

Zahn [22] has demonstrated how the minimum spanning tree (MST) can be used to detect clusters. His choice of MST was influenced by the Gestalt principle, which favors the grouping of attribute patterns based on Euclidean distance measure. Shaffer et al [19] demonstrate the similarity of the mode-seeking partitioning algorithm [10] to the graph algorithm of Zahn [22] based on minimum spanning trees. Lu and Fu [11] used another graph-based approach called "Nearest-Neighbor clustering algorithm" to cluster patterns during character recognition [9].

The concept of using fragmentation of data as a means of improving the performance of a database management system has often appeared in the literature on file design and optimization. Hoffer [8] developed a non-linear, zero-one program which minimizes a linear combination of storage, retrieval and update costs, with capacity constraints for each file. Babad [1] formulated a less restrictive vertical partitioning problem for variable length attributes as a non-linear zero-one program. In the work of Eisner and Severance [5], a file can be partitioned into two subfiles: a primary and secondary subfile. Two forms of cost function are used in this approach. The first function is the sum of storage charges for subtuples in the primary subfile, and the cost of accessing all the subtuples residing in the secondary subfile. The second function is nonlinear, and measures the total costs of access, transfer, and storage for subtuples in both primary and secondary subfiles. The limitation of this approach is that at most two subfiles are allowed [17]. March and Severance [13] extended this model to incorporate block factors for both primary and secondary memories. Hoffer and Severance [7] grouped the attributes of a relation based on the extent to which they were used together (measured the "affinity between pairs of attributes"). This clustering of attributes based on their pairwise affinity was done using the bond energy algorithm (BEA). The BEA produced matrix in which an cost function was minimized for the entire matrix using the affinity attribute matrix. They left the creation of partitions to the subjective evaluation of the designer. Schklonik [18] has examined the problem of partitioning a hierarchical structure (for a hierarchical database) in order to minimize the access time to it for a given access pattern. Segment sizes and scan information is used to minimize the page faults.

Hammer and Niamir [6] developed two heuristics, grouping and regrouping, and used them to perform the partitioning. The grouping heuristic starts by initially assigning each attribute to a different partition. On each iteration, all possible grouping of these partitions is considered and the one with maximum improvement is chosen as the candidate grouping for the next iteration. During regrouping, attributes are moved between partitions to achieve any additional improvements possible. Navathe et al [15] use a two step approach for vertical partitioning. In the first step, they use the given input parameters in the form of an attribute usage matrix to construct the attribute affinity matrix on which clustering is performed. After clustering, an empirical objective function is used to perform iterative binary partitioning. In the second step, estimated cost factors reflecting the physical environment of fragment storage are considered for further refinement of the partitioning scheme. Cornell and Yu [4] propose an algorithm, as an extension of Navathe et al [15] approach, which decreases the number of disk accesses to obtain an optimal binary partitioning. This algorithm uses specific physical factors such as number of attributes, their length and selectivity, cardinality of the relation etc. Navathe and Ra [Nava 89] present a graph-based approach to the vertical partitioning problem. This approach is based on the observation that all pairs of attributes in a fragment must have high "within fragment affinity" but low "between fragment affinity". Reduction in complexity is claimed as the main advantage of their approach.

There are important differences in the criteria that are used in traditional clustering problems and data fragmentation problem. In data clustering algorithms, the number of clusters is usually fixed. Otherwise, the extreme case of only a single cluster in the partition will minimize the inter-cluster variation. However in the database design application, there is a need to determine the *number* of clusters as well and hence the objective function used in data clustering algorithms cannot be borrowed without any changes to vertical partitioning in databases. Most importantly, in data base design problem, the number of clusters is an important factor that influences the trade-off between local and remote transaction processing costs.

Our interest in the above was to see whether the criteria used in the data clustering domain could be adapted, with some changes, to the data fragmentation problem.

3 Need for an objective function

Algorithms such as Bond Energy, Binary Vertical Partitioning, Ra's algorithm and Zahn's algorithm etc. use affinity matrix as the input. The attribute affinity is a measure of an imaginary bond between *a pair* of attributes. Because only a pair of attributes is involved, this measure does not reflect the closeness or affinity when *more than two* attributes are involved. Hence the algorithms which use attribute affinity matrix are using a measure (that is an ad hoc extrapolation of pairwise affinity to cluster affinity) that has no bearing on the affinity as measured with respect to the entire cluster. As a consequence, we believe, it was difficult to show or even characterize affinity values for the resulting clusters having more than two attributes.

As we wanted to obtain a general objective function and a criterion for describing affinity value for clusters of different sizes, our approach does not assume an attribute affinity matrix. The input model that we consider is a matrix which consists of attributes (columns) and the transactions (rows) with the frequency of access to the attributes for each transaction, as the values in the matrix. With this input model we overcome the limitations that are inherent to approaches based on attribute affinity matrix.

As is evident from the discussion in the previous section, there are a number of partitioning algorithms available both in the database design area and in other application areas. Many of these algorithms use different criteria to arrive at a partitioning scheme. The objective function used by one algorithm is not suitable for evaluating the "goodness" of other algorithms. Thus we do not have a common objective function to compare and evaluate the results of these partitioning algorithms, or in general evaluate the "goodness" of a particular partitioning scheme. Hence we need a partition Evaluator to compare and evaluate different algorithms, that use the same input in the database design process. Since attribute usage matrix is the most commonly used input available during the initial design stage, we first design an Evaluator which can be used to evaluate the "goodness" of partitions arrived at using this input. This Partition Evaluator can be used as a basis for developing algorithms to create fragments of a relation. With this approach, there is hope that admissibility aspects of algorithms can be shown. In addition, this Partition Evaluator has the flexibility to incorporate other information, such as type of queries (retrieval/updates), allocation information about the partitions, remote processing cost (transmission cost) and the transaction usage pattern at any particular site. In the next section we will discuss the development of the Partition Evaluator in detail.

4 Development of the Partition Evaluator

In any practical database application, a transaction does not usually require all the attributes of the tuples of a relation being retrieved during the processing of the transaction. When a relation is *vertically* divided into data fragments, the attributes stored in a data fragment that are irrelevant (i.e., not accessed by the transaction) with respect to a transaction, add to the retrieval and processing cost, especially when the number of tuples involved in the relation is very large. In a centralized database system with memory hierarchy, this will lead to too many accesses to the secondary storage. In a distributed database management system, when the relevant attributes (i.e., attributes accessed by a transaction) are in different data fragments and allocated to different sites, there is an additional cost due to remote access of data. Thus one of the desirable characteristics of a distributed database management systems that we wish to achieve through partitioning is the *local accessibility* at any site. In other words, each site must be able to process the transactions *locally* with *minimal access* to data located at remote sites.

Ideally, we would like any transaction to access only the attributes in a single data fragment with no or minimal access of irrelevant attributes in that fragment. But this is impossible to achieve in the general cased since transactions access different and overlapping subsets of attributes of a relation. Moreover, transactions are run at different sites and hence some of the data fragments that contain relevant attributes of a transaction may reside in remote sites. The overall transaction processing cost in a distributed environment thus consists of local transaction processing cost and the remote transaction processing cost. Though it is possible to replicate the data to avoid remote processing cost, for the first step we assume no data redundancy to avoid modeling overhead to ensure data integrity and consistency and also additional storage costs. In this paper, we assume that during the database design process, "partitioning" phase is followed by the "allocation" phase during which the non-overlapping data fragments obtained during the partitioning phase are allocated to different sites possibly with some replication. Hence the partition evaluator we propose will evaluate vertical partitioning schemes wherein the data fragments are *non-overlapping* in the attributes. Here non-overlapping refers only to non-primary key attributes. The primary key is preserved in each partition. This is necessary to obtain the original relation without any loss or addition of spurious tuples. Discussion of our approach to relations that are not in Boyce-Codd normal form is in a later section.

The goal of partitioning the attributes and allocating to different sites, is to arrive at a minimum processing cost for any transaction originating at any site. Since the transaction processing cost has two components, one due to local processing and another due to remote processing, our Partition Evaluator that measures the "goodness" of a vertical partitioning scheme also has two corresponding component terms; we refer to these terms as "irrelevant local attribute access cost" and "relevant remote attribute access cost" terms respectively. For simplicity, we assume that a single access of a data fragment corresponds to an unit cost; this assumption can easily be relaxed if more information is available regarding the access methods, network parameters etc. The irrelevant local attribute cost term measures the local processing cost of transactions that is due to irrelevant attributes of data fragments assuming that all the data fragments required by a transaction are available locally. The relevant remote attribute access term measures the remote processing cost due to relevant attributes of data fragments that are accessed remotely by transactions; note that the contribution to remote access cost due to irrelevant attributes is already included in the first term. Since we do not know during partitioning how the data fragments are allocated, we compute the second term assuming that data fragments needed by a transaction are located at different sites. In the absence of any information regarding the transaction execution strategies, we can compute the second term either by determining the average of all the remote access costs obtained by running the transaction at every site containing a fragment needed by the transaction or by assuming the transaction to be run at one of the fragment sites. If more information is available regarding the transaction strategies, we can incorporate it in the second term.

The *ideal* characteristic of a Partition Evaluator is summarized in Figure 2 where the behavior of the two components as a function of partition size (number of data fragments) is illustrated. The first component should be maximum for a partition size of 1 and should be zero for a partition size equal to number of attributes in the relation. The second component on the other hand should be zero and maximum respectively for these two extremes. In between the two extremes, the first component should be more responsive to smaller partition sizes while the second component should be more responsive to larger partition sizes. This is necessary to avoid unnecessary bias toward one component or another since we know that in a distributed database design, it is undesirable to have the two extreme cases of the partition size. In addition to being responsive to partition sizes, the evaluator function should discriminate between different distributions of the attributes among the fragments for a fixed partition size. Later, we will present experimental evidence to demonstrate that our Partition Evaluator meets these criteria reasonably well.

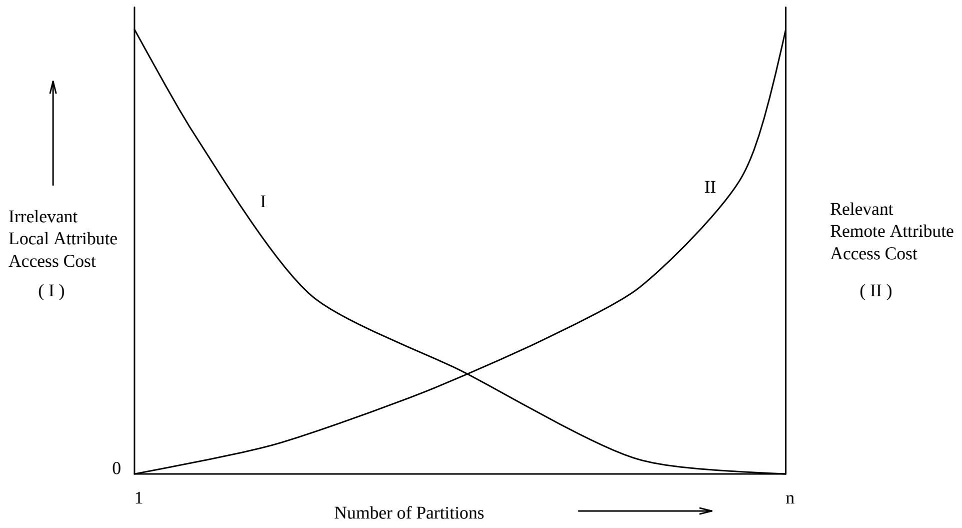

Figure 2: Ideal characteristics of PE components

4.1 Derivation of the Partition Evaluator

4.1.1 Database and Transaction Specification

We derive a partition evaluator in this section without making any assumptions on the input. The input assumed is a relation (consisting of a set of attributes) and an attribute usage matrix. We really do not have to make any assumptions as to whether the input relation is a Universal relation or whether it is in a particular normal form. The partitioning obtained is non-overlapping except for the key attributes in the input relation. The key attributes are part of each partition. In a later section we indicate how normalization and functional dependency information can be easily added to our approach.

The input model that we use is an attribute usage matrix [AUM(t,j)] which consists of the attributes(j) in a relation as columns and the transactions(t) as rows with the frequency of access to the attributes for each transaction as the values in the matrix. Note that we are avoiding the construction of any attribute affinity matrix which has its own limitation pointed out earlier.

A representative attribute usage matrix[6] is shown below:

[6] In this section, no distinction is made between update and retrieval transactions; it is discussed as one of the extensions in a later section

$Trans\backslash Attrs$	$A1$	$A2$	$A3$	$A4$	$A5$
$T1$	0	$q1$	0	$q1$	$q1$
$T2$	$q2$	$q2$	$q2$	0	$q2$
$T3$	$q3$	0	0	$q3$	$q3$
$T4$	0	$q4$	$q4$	0	0
$T5$	$q5$	$q5$	$q5$	0	0

4.1.2 Definitions and Notations

A *partition* (scheme) is a division of attributes of a relation into vertical fragments in which for any two fragments, the set of attributes of one is non-overlapping with the set of attributes of another. For example, the partition $\{(1,3)(2,4)(5)\}$ defines a collection of fragments in which attributes 1 and 3 are in one fragment, 2 and 4 are in another and 5 is in a separate fragment. The following are used in the derivation of the Partition Evaluator.

n : Total number of attributes in a relation that is being partitioned.
T : Total number of transactions that are under consideration.
q_t : frequency of transaction t for $t = 1, 2, \ldots, T$.
M : Total number of fragments of a partition.
n_i : Number of attributes in fragment i.
n_{ikt}^r : Total number of attributes that are in fragment k
accessed remotely with respect to fragment i by transaction t.
f_{tj}^i : frequency of transaction t accessing attribute j in fragment i
note that f_{tj}^i is either 0 or q_t
A_{ij} : Attribute Vector for attribute j in fragment i.
t-th component of this vector is f_{tj}^i
S_{it} : Set of attributes contained in fragment i that the transaction t
accesses; It is empty if t does not need fragment i.
$|S_{it}|$: number of attributes in fragment i that the transaction t accesses.
R_{itk} : Set of relevant attributes in fragment k accessed remotely
with respect to fragment i by transaction t;
these are attributes not in fragment i but needed by t
$|R_{itk}|$: number of relevant attributes in fragment k accessed remotely
with respect to fragment i by transaction t

The attribute vector A_{11} (assuming A_1 is in partition 1) for the example given above is as follows.

$$A_{11} = \begin{bmatrix} 0 \\ q2 \\ q3 \\ 0 \\ q5 \end{bmatrix}$$

Consider the partitioning scheme (1,3),(2,4),(5), for the attribute usage matrix given above. Fragment 1 is (1,3), fragment 2 is (2,4) and fragment 3 is (5). Assume that transaction T2 (i.e., $t = 2$) is run at the site where fragment 1 is located. Then S_{it} is (1,3) and $|S_{it}|$ is 2. n^r_{i2t} is 2 and n^r_{i3t} is 1. $|R_{it2}|$ is 1 and $|R_{it3}|$ is 1.

Next we explain how each of the two components of our Partition Evaluator is derived.

4.1.3 Irrelevant local attribute access cost

For the first component, we use the square-error criterion as given in [Jain 88] for data clustering. The objective here is to obtain a partition which will minimize the square-error for a fixed number of fragments. This criterion assigns a penalty factor whenever irrelevant attributes are accessed in a particular fragment.

Let us assume that n attributes have been partitioned into M fragments $(P_1, P_2, ...P_M)$ with n_i attributes in each fragment. Thus $\sum_{i=1}^{M} n_i = n$. The mean vector V_i for fragment i is defined as follows.

$$V_i = \frac{1}{n_i} \sum_{j=1}^{n_i} A_{ij} \qquad\qquad 0 < i \leq M \qquad\qquad (1)$$

This mean vector represents an average access pattern of the transactions over all attributes of fragment i. For an attribute vector A_{ij}, $(A_{ij} - V_i)$ is called the "difference vector" for attribute j in fragment i. The square-error for the fragment P_i is the sum of the squares of the lengths of the difference vectors of all the attributes in fragment i. It is given by

$$e_i^2 = \sum_{j=1}^{n_i} (A_{ij} - V_i)^T (A_{ij} - V_i) \qquad\qquad 0 < i \leq M \qquad\qquad (2)$$

If $A_{ij} = V_i$ then e_i^2 will be zero. This will occur for the trivial case when there is a single attribute in each fragment or for the case when all the attribute in each fragment are relevant to all the transactions that access that fragment. It is the latter case that we are interested in and to avoid the former case, we will use the second component.

The square-error for the entire partition scheme containing M fragments is given by

$$E_M^2 = \sum_{i=1}^{M} e_i^2 \tag{3}$$

Smaller the value of E_M^2, smaller is the cost due to access of irrelevant attributes. E_M^2 however does not reflect the cost that might be incurred by accessing attributes remotely when the fragments may be in different sites. Hence, for distributed database applications, we cannot evaluate partitions on the basis of E_M^2 alone. The behavior of E_M^2 is given in figure 3 as curve I for a data set consisting of 10 attributes and eight transactions (more details of this example are discussed in the next section).

From this graph, we can see that the minimum value for E_M^2 is certainly achieved for n partitions in an n attribute system (The minimum value may be reached even for less than n partitions depending on the AUM). We would like to have a number of fragments which is typically much less than n and still having the least E_M^2 value. In some data clustering techniques, the number of data clusters is minimized using an index called Davies-Bouldwin (DB) index [9] which is a measure of the spread between centers of the clusters. For a typical data set with small standard deviation (less than 0.1), this index reaches a global minimum (highest spread) for a partition size that falls between the extremes. From the curve I of figure 3, we can see that the standard deviation of the data set given by the attribute usage matrix is greater than 0.1, which does not meet the condition for using the DB index. For this reason, we seek another quantity which will reflect the remote access cost. In the next section, we will discuss the development of a quantity for remote attribute access cost.

4.1.4 Relevant Remote Attribute Access Cost

Now we will include the second component which would compute a penalty factor that computes the function shown earlier in figure 2. Given a set of partitions, for each transaction running on a partition compute the ratio of the number of remote attributes to be accessed to the total number of attributes in each of the remote partitions. This is summed over all the partitions and over all transactions giving the following equation. The second term is given by

$$E_R^2 = \sum_{t=1}^{T} \Delta_{i=1}^{M} \sum_{k \neq i} \left[q_t^2 * |R_{itk}| \frac{|R_{itk}|}{n_{itk}^r} \right] \tag{4}$$

Here Δ[7] is an operator that is either an average, minimum or maximum over all i. These different choices of the operator give rise to average, optimistic and pessimistic estimates of the remote access cost. If specific information is available regarding transaction execution strategies, then we can determine for each transaction t, the remote fragments accessed by the transaction and the remote access cost can be refined accordingly. In our experimental investigation, we use the optimistic estimate for illustration.

We will show in the next section how we obtained the above form for the second term. Our Partition Evaluator (PE) function is given by

[7] Δ is introduced to keep the formula general and applicable to a wide class of problems; also, a desired new operator can be substituted for Δ without having to change the objective function.

$$PE = E_M^2 + E_R^2 \tag{5}$$

4.1.5 Compatibility of the Two Components in PE

As the second component has frequency in quadratic form, we need to make sure that both the components are compatible in terms of the units they produce. Specifically, we need to see whether the frequency appears in the same way in the first component. In order to do that, it is instructive to look closely at the first term. In particular, we will rewrite E_M^2 differently so as to identify the contributions due to each transaction to the square error term of each fragment in the partition. Thus

$$E_M^2 = \sum_{i=1}^{M} \sum_{j=1}^{n_i} (A_{ij} - V_i)^T (A_{ij} - V_i) \tag{6}$$

Now the mean Vector V_i for fragment i, can be defined as follows.

$$V_i = \begin{bmatrix} \frac{|S_{i1}|*q_1}{n_i} \\ \frac{|S_{i2}|*q_2}{n_i} \\ \cdots \\ \cdots \\ \cdots \\ \cdots \\ \frac{|S_{it}|*q_t}{n_i} \end{bmatrix}$$

The attribute vector A_{ij} is,

$$A_{ij} = \begin{bmatrix} f_{1j}^i \\ f_{2j}^i \\ \cdots \\ \cdots \\ \cdots \\ \cdots \\ f_{tj}^i \end{bmatrix}$$

Now,

$$E_M^2 = \sum_{i=1}^{M}\sum_{j=1}^{n_i}\left[f^i_{1j} - \frac{|S_{i1}| * q_1}{n_i}, \ldots, f^i_{tj} - \frac{|S_{it}| * q_t}{n_i}\right]\begin{bmatrix} f^i_{1j} - \frac{|S_{i1}| * q_1}{n_i} \\ f^i_{2j} - \frac{|S_{i2}| * q_2}{n_i} \\ \cdots \\ \cdots \\ \cdots \\ \cdots \\ f^i_{tj} - \frac{|S_{it}| * q_t}{n_i} \end{bmatrix} \tag{7}$$

The above formula can be rewritten as follows so as to allow us to identify the different components that make up the irrelevant local attribute access term.

$$E_M^2 = \sum_{i=1}^{M}\sum_{j=1}^{n_i}\sum_{t=1}^{T}\left[\delta_{jt} * q_t^2\left(1 - \frac{|S_{it}|}{n_i}\right)^2 + (1 - \delta_{jt})\left(q_t * \frac{|S_{it}|}{n_i}\right)^2\right] \tag{8}$$

where,

δ_{jt} = 1 if the attribute j is accessed by the transaction t
= 0 if the attribute j is not accessed by the transaction t.
The first term $q_t^2\left(1 - \frac{|S_{it}|}{n_i}\right)^2$ represents relevant attribute accesses and the second term represents irrelevant attribute accesses. Even if we have a 0 in A_{ij}, we still have the mean squared quantity $\left(q_t * \frac{|S_{it}|}{n_i}\right)^2$.

Therefore,

$$E_M^2 = \sum_{i=1}^{M}\sum_{t=1}^{T}\left[|S_{it}| * q_t^2\left(1 - \frac{|S_{it}|}{n_i}\right)^2 + (n_i - |S_{it}|)\left(q_t * \frac{|S_{it}|}{n_i}\right)^2\right] \tag{9}$$

where

$\sum_{j=1}^{n_i}\delta_{jt}$ = $|S_{it}|$ and
$\sum_{j=1}^{n_i}(1 - \delta_{jt})$ = $n_i - |S_{it}|$

Hence,

$$E_M^2 = \sum_{i=1}^{M}\sum_{t=1}^{T}\left[q_t^2\,|S_{it}|\left(1 - \frac{|S_{it}|}{n_i}\right)^2 + q_t^2(n_i - |S_{it}|)\left(\frac{|S_{it}|}{n_i}\right)^2\right] \tag{10}$$

If $n_i = |S_{it}|$, then $E_M^2 = 0$. This implies that the transaction t accesses all attributes in fragment i whenever it accesses the fragment i. We can still reduce the above equation as follows.

$$E_M^2 = \sum_{i=1}^{M}\sum_{t=1}^{T}\left[q_t^2 * |S_{it}|\left(1 + \frac{|S_{it}|^2}{n_i^2} - 2 * \frac{|S_{it}|}{n_i}\right) + q_t^2 * |S_{it}|\,(n_i - |S_{it}|)\left(\frac{|S_{it}|}{n_i^2}\right)\right] \tag{11}$$

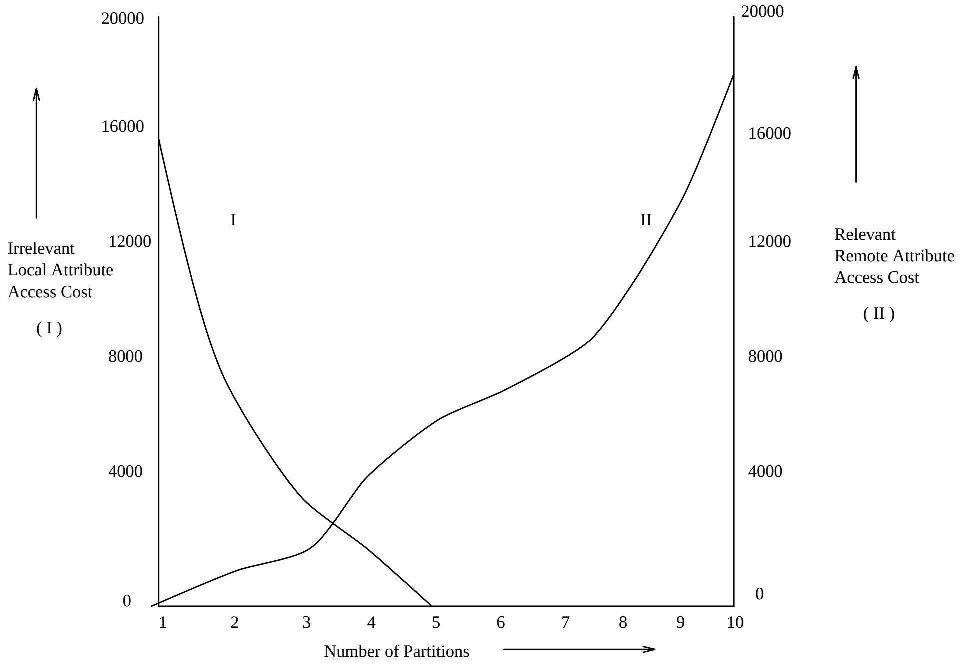

Figure 3: Behavior of PE components for an example

Simplifying the equation above we get,

$$E_M^2 = \sum_{i=1}^{M} \sum_{t=1}^{T} \left[q_t^2 * |S_{it}| \left(1 - \frac{|S_{it}|}{n_i} \right) \right] \tag{12}$$

The equation above is the same as equation as 6, but in a much simpler form. We can clearly see from this equation the contribution to E_M^2 by the irrelevant attributes; $\frac{(1-|S_{it}|)}{n_i}$ is the fraction of irrelevant attributes in fragment i as far as transaction t is concerned. As it was mentioned earlier, E_M^2 is the cost factor only for local transaction processing.

Now we can see that the E_R^2 term shown earlier, is very much similar in form to the E_M^2 term. Also the necessity for having a squared frequency term in E_R^2 is made clear. Hence the PE is given by

$$PE = E_M^2 + E_R^2 \tag{13}$$

Curve II of Figure 3 illustrates the behavior of E_R^2 for the same example of 10 attributes and 8 transactions.

5 Analysis of the Partition Evaluator

The final form of the Partition Evaluator is given in equation 13. In order to analyze and test the behavior of the PE, an exhaustive enumeration algorithm has been implemented. The input to the program is an attribute usage matrix. The following input was used to test the Partition Evaluator. For each fragment (from 1 to 10), the Partition Evaluator was computed.

Input: Attribute Usage matrix.

$Trans.\backslash Attrs.$	1	2	3	4	5	6	7	8	9	10
$T1$	25	0	0	0	25	0	25	0	0	0
$T2$	0	50	50	0	0	0	0	50	50	0
$T3$	0	0	0	25	0	25	0	0	0	25
$T4$	0	35	0	0	0	0	35	35	0	0
$T5$	25	25	25	0	25	0	25	25	25	0
$T6$	25	0	0	0	25	0	0	0	0	0
$T7$	0	0	25	0	0	0	0	0	25	0
$T8$	0	0	15	15	0	15	0	0	15	15

An exhaustive enumeration program was written in C/C++ to produce all the possible combinations of attributes with number of fragments varying from 1 to 10. The Partition Evaluator was applied to each of these combinations. We assume that if a transaction is to be run on a fragment, and that fragment does not contain even a single attribute accessed by that transaction, then that transaction is not run on that fragment. Each of the transactions is run on each fragment and the minimum, maximum and the average value of the Partition Evaluator is calculated. Total number of fragments evaluated was 115975. The optimal values (minimum) along with the partitioning scheme for each partition size is given below. The results are plotted in figure 4.

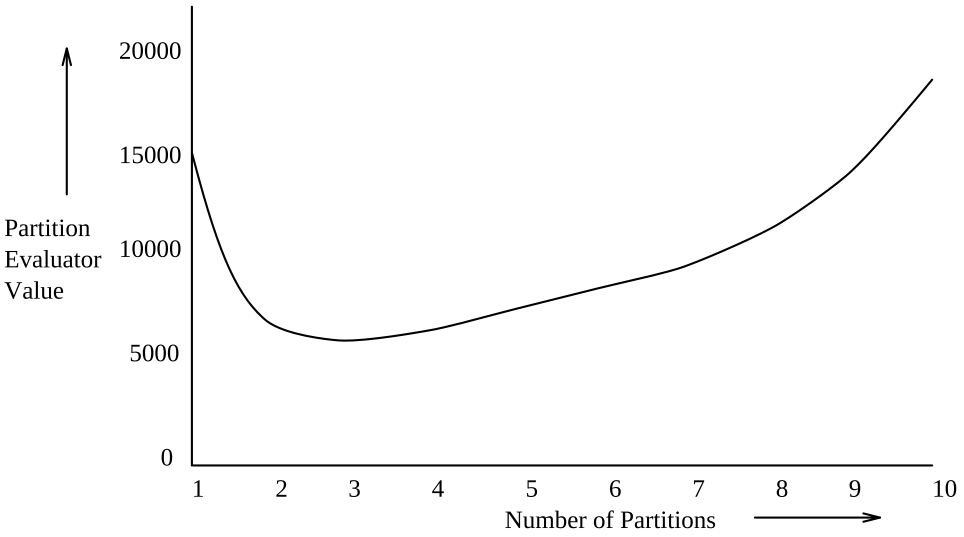

Figure 4: Behavior of partition evaluator for an example

Number Of Fragments	*Min PE Value*	*Partition Scheme*
1	15085	(12345678910)
2	8457	(1456710)(2389)
3	5820	(157)(2389)(4610)
4	6024	(15)(2389)(4610)(7)
5	6874	(15)(2389)(46)(7)(10)
6	7724	(15)(2389)(4)(6)(7)(10)
7	8976	(1)(2389)(4)(5)(6)(7)(10)
8	11692	(1)(289)(3)(4)(5)(6)(7)(10)
9	14000	(1)(28)(3)(4)(5)(6)(7)(9)(10)
10	18350	(1)(2)(3)(4)(5)(6)(7)(8)(9)(10)

As we can see from the above table the optimum value (third row above) is found to be in a partition of three fragments. The Partition Evaluator values above are the minimum of the values where the second component of the PE is calculated using optimistic (minimum value) estimate. For this particular example, other algorithms such as Ra's [16], Zahn's [22] and Binary Vertical Partitioning [15] identify the above mentioned partition set (i.e., 3 fragments) as optimum. In Zahn's approach, once the maximum spanning tree is obtained two different conditions can be used to determine the partitions [9]. For this example, when these two conditions are applied to Zahn's algorithm, they produce two different partitioning schemes. One of them is the same as the optimal partitioning scheme obtained as above and the other one is not. Hence, one may want to use another criterion (such as the PE) to guide the selection of the most appropriate condition when faced with a number of alternatives. Since producing any partitioning scheme using the Bond Energy algorithm is subjective in nature, this Partition Evaluator can be used to guide the partitioning once the Bond Energy Algorithm is applied. Ra's and Binary Vertical Partitioning algorithms were applied to another example with twenty attributes and fifteen transactions [Nava 84]. The results of these two algorithms were different. Ra's algorithm produces five fragments and Binary Vertical partitioning algorithm produced four fragments. The Partition Evaluator was applied to both the results and it was found that indeed the four fragment result is better than the five fragment result. This example highlights the usefulness of the Partition Evaluator to evaluate the results of the different partitioning algorithms and select the appropriate partitioning algorithm.

EXAMPLE: The following discussion explains how one of these figures is arrived at.

Number Of Fragments	*Min PE Value*	*Partition Scheme*
3	5820	(157)(2389)(4610)

Let us call (1 5 7) as fragment I, (2 3 8 9) as fragment II and (4 6 10) as fragment III. The first step is to calculate the square error for the given input.

Mean for each fragment is:

FragmentI	*FragmentII*	*FragmentIII*
25	0	0
0	50	0
0	0	25
12	18	0
25	25	0
17	0	0
0	13	0
0	8	15

1) The square error

$$E_M^2 = \sum_{i=1}^{M}\sum_{j=1}^{n_i}(X_{ij} - V_i)^T(X_{ij} - V_i)$$

$E_M^2 = 1234 \quad + \quad 2078 \quad + \quad 0 = 3312$

2) Cost due to relevant remote attribute accesses of a transaction (T4) is:

	Value	Minimum of the values
T4 run @ Fragment I:	$35^2 * 2 * 2/4 = 1225$	
T4 run @ Fragment II:	$35^2 * 1 * 1/3 = 408$	408
T4 run @ Fragment III:	cannot be run	

Total cost of accessing relevant remote attributes by all transactions is

$$408 + 1875 + 225 = 2508$$

Therefore,

$$\mathrm{PE} = 3312 + 2508 = 5820.$$

The above example shows that utility of the Partition Evaluator developed in this paper to evaluate the results of any partitioning scheme and especially for comparing previously developed algorithms.

6 Extensions and Implementation

So far, we have discussed the formulation of an objective function (Partition Evaluator) assuming that attribute usage matrix is the only information available to the designer during the partitioning phase of the design process. In this section we first discuss briefly how we can use the objective function in the presence of additional information and then discuss the status of the implementation effort.

Dependencies: If a universal relation or a relation which is not in the Boyce-Codd normal form is used as input, several constraints need to be considered when a partition is generated. The cost of maintaining functional and multi-valued dependencies is reduced if all the attributes participating in a dependency are either in the same fragment or at least allocated to the same node. The first case can be easily accommodated without changing the objective function by defining a *dependency checker* which evaluates whether a partition should be considered for cost computation. In other words, during the design process any fragment that violates any of the dependencies would be rejected even though that fragment might give the minimum value for the partition evaluator; such partitioning schemes are deemed not admissible. For the second case, the information is passed to the allocation phase.

Fixed partition size: In many cases, the number of sites are known a priori and the objective function need to minimize the cost for a given number of fragments. This, again, does not require any changes as one can choose the minimum cost of all the alternatives for the number of fragments specified.

Update vs. retrieval transactions: Although no distinction was made between update and retrieval transaction in the earlier discussion, it is quite straightforward to distinguish between them. Typically, retrieve only transaction access data items only once where as there is an additional overhead of writing back for updated data items on to secondary storage (sometimes more if the data item has already been flushed to secondary storage). A weight factor can be used to distinguish update type from retrieval type transactions. We can increase the frequency (for example double it) of update type transactions to reflect their processing cost. In our simulation, we can take into account the type of queries by increasing the frequency for update type queries in the attribute usage matrix.

Local vs. remote processing cost : In our formulation of PE, we have assumed a unit cost for processing local as well as remote data once the remote data reaches the local node. We also assumed a uniform transmission cost between any two sites in the network. However, realistically, there is difference between accessing local data possibly using access methods and remote data for which there are no access methods (unless dynamically created at the local site). Also, the transmission costs between any two nodes is not likely to be the same.

We can include another factor to reflect the ratio of local processing cost to the processing of data coming from remote sites. This factor can be included giving different weights to the two component terms in equation 13. If W_1 is the factor for processing local data and W_2 is the factor for processing remote data, it can be accommodated by modifying equation 13 as follows:

$$PE = E_M^2 + (W_2/W_1)E_R^2 \tag{14}$$

Note, however, that a better way would be to integrate the factors into the formula.

Non-uniform transmission cost between nodes can be taken into account by modifying equation 4 to include a multiplicative factor TR_{ik} (which reflects the actual transmission cost/unit data between sites i and k) inside the inner summation.

Replication: The Partition Evaluator can be extended with minimal changes to accommodate replication of fragments. In this case, we assume that attributes are remotely accessed only if they do not exist locally. Thus in computing the second term, only those attributes that exist in a fragment remotely accessed with respect to the site of another fragment (call it "local") but do not exist in the local fragment are taken into account. However, the form of the Partition Evaluator remains the same. Although replication provides availability, the cost of maintaining consistency of replicated data need to be considered. The above suggestion takes only the cost of retrieval into account and not the cost of update propagation.

6.1 Implementation Status

A distributed database design testbed is being developed at the Database Systems R & D Center, Univ. of Florida, Gainesville. The testbed includes several different vertical partitioning algorithms and modules that compare and evaluate the results of these algorithms. The algorithms that were developed as part of this effort are as follows:

- Bond Energy Algorithm [12].

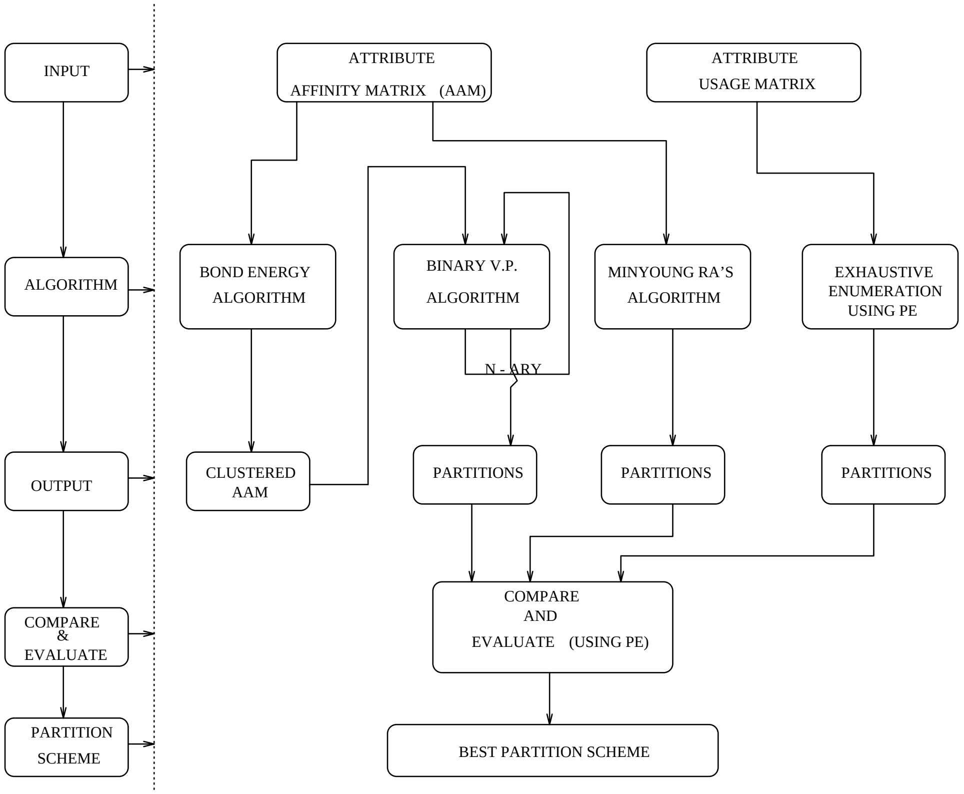

Figure 5: Distributed Database Design testbed prototype

- Binary Vertical Partitioning Algorithm [15].

- Minyoung Ra's Graphical Algorithm [16].

- Exhaustive Enumeration Algorithm.

The figure 5 gives an overall outline of the design testbed.

More algorithms can be easily "hooked" on to this design testbed prototype.

7 Summary and Future Work

In this paper, we have presented a general approach to the vertical partitioning problem. Our study brought out the problems associated with the use of attribute affinity matrix currently used in almost all of the earlier data partitioning algorithms. We started with the objective function used in clustering methods and extended it to suit the requirements of database design. As a result we have brought together work in two isolated areas and established a correspondence between them. Using the objective function derived in this paper and the approach proposed, one can evaluate any vertical partitioning algorithm that uses the same input as our model. We wanted to identify and express an objective function quantitatively before embarking on the development of algorithms. It will now be easier to develop algorithms (heuristics-based or otherwise) exploiting the properties of the function being optimized. Our Partition Evaluator satisfies the need for a common criteria or objective function and can be used to compare and evaluate the extant vertical partitioning algorithms. Finally, the PE developed in this paper has the flexibility to incorporate additional design information such as type of queries (retrieval/updates), transmission cost, and replication as briefly outlined in the previous section.

We are currently developing heuristic algorithms for the objective function derived in this paper. We are in the process of developing integrated objective function to include other inputs to the database design process, such as query types, constraints on allocation, transmission cost, transaction usage pattern at any particular site, into the objective function derived in this paper. We plan on comparing different algorithms in more detail using the partition evaluator. We hope to integrate the the Partition Evaluator into a database design testbed which will help designers to choose the right algorithm for database initial design as well as for redesign.

References

[1] M. Babad. A record and file partitioning model. *Commun. ACM 20*, 1(Jan 1977).

[2] S. Ceri, S.B. Navathe, G. Weiderhold. Distribution Design of Logical Database Schemas *IEEE trans. on SW engg. Vol SE-9 No.4*, p 487 - 503, July 1983.

[3] S. Ceri, S. Pernici, and G. Weiderhold. Optimization Problems and Solution Methods in the Design of Data distribution. *Information Sciences Vol 14, No. 3*, p 261-272, 1989.

[4] D. Cornell, and P. Yu. A Vertical Partitioning Algorithm for Relational Databases. *Proc. Third International Conference on Data Engineering*, Feb. 1987, pp. 30-35.

[5] M. Eisner, and D. Severance. Mathematical techniques for efficient record segmentation in large shared databases. *J. ACM 23*, 4(Oct. 1976).

[6] M. Hammer, and B. Niamir. A heuristic approach to attribute partitioning. *In Proceedings ACM SIGMOD Int. Conf. on Management of Data*, (Boston, Mass., 1979), ACM, New York.

[7] J. Hoffer, and D. Severance. The Uses of Cluster Analysis in Physical Database Design *In Proc. 1st International Conference on VLDB*, Framingham, MA, 1975, pp. 69 - 86.

[8] J. Hoffer. An integer programming formulation of computer database design problems. *Inf. Sci.*, 11(July 1976), 29-48.

[9] A. Jain, and R. Dubes. Algorithms for clustering Data. Prentice Hall Advanced Reference Series, Englewood Cliffs, NJ, 1988.

[10] J. Kittler. A locally sensitive method for cluster analysis. *Pattern Recognition 8*, 22-33.

[11] S. Lu, and K. Fu. A sentence-to-sentence clustering procedure for pattern analysis. *IEEE Transactions on Systems, Man and Cybernetics SMC 8*, 381-389.

[12] W. McCormick, P. Schweitzer, and T. White. Problem Decomposition and Data Reorganization by a Clustering technique *Operations Research*, 20 Sep. 1972.

[13] S. March, and D. Severance. The determination of efficient record segmentation and blocking factors for share data files. *ACM Trans. Database Syst. 2*, 3(Sept. 1977).

[14] R. Muthuraj. A formal approach to the vertical partitioning problem in distributed database design. *M.S. Thesis, Dept. of Computer Science, Univ. of Florida*, Aug. 1992.

[15] S. Navathe, S. Ceri, G. Weiderhold, and J. Dou. Vertical Partitioning Algorithms for Database Design *ACM Transactions on Database Systems*, Vol. 9, No. 4, Dec. 1984.

[16] S. Navathe, and M. Ra. Vertical Partitioning for Database Design: A Graphical Algorithm. ACM SIGMOD, Portland, June 1989.

[17] B. Niamir. Attribute Partitioning in Self-Adaptive Relational Database System. *Ph. D. Dissertation*, M.I.T. Lab. for Computer Science, Jan. 1978.

[18] M. Schkolnic. A Clustering Algorithm for Hierarchical Structures ACM TODS, Vol. 1, No. 2, pp. 27-44. March 1977.

[19] E. Shaffer, R. Dubes, and A. Jain. Single-link characteristics of a mode-seeking algorithm. *Pattern Recognition 11*, 65-73.

[20] N. Tamer, P. Valduriez. Principles of Distributed Database Systems. *Prentice Hall Englewood Cliffs, New Jersey 07362.*

[21] A. Torn. Cluster analysis using seed points and density-determined hyperspheres as an aid to global optimization. *IEEE Transactions on Systems, Man and Cybernetics SMC 7*, 610-616.

[22] C. Zahn. Graph-theoretical methods for detecting and describing Gestalt Clusters. *IEEE Transactions on Computers C 20*, 68-86.