From Graph Partitioning Problem to Vertical Fragmentation in Distributed Database Design: A Genetic Approach

Anciano, Juan Luis
janciano@ldc.usb.ve

Arráiz, Emely
arraiz@ldc.usb.ve

Di Serio, Angela
adiserio@ldc.usb.ve

Savino, Nunzio
nunzio@ldc.usb.ve

Departamento de Computación y Tecnología de la Información
Universidad Simón Bolívar
A.P. 89.000, Caracas, Venezuela

ABSTRACT

The design of a distributed system involves making decisions on the placement of data and programs across the sites of a computer network. In the case of distributed databases, the distribution is done physically using techniques such as fragmentation and allocation. Given a relational database schema, fragmentation subdivides each relation into horizontal or vertical partitions. Navathe et al. [1] proposes an heuristic based on the grouping of attributes that have high affinity or relation between them. Some inconveniences are found with this approach, and we present an alternative approach using genetic algorithm and graph partitioning to solve the vertical fragmentation problem.

Keywords: Database, Distributed and Parallel Systems, Optimization and Simulation

1. INTRODUCTION

The design of a distributed system involves making decisions on the placement of data and programs across the sites of a computer network. In the case of distributed databases, the distribution is done physically using techniques such as fragmentation and allocation.

Fragmentation subdivides each relation, on a relational database schema, into horizontal or vertical partitions. Horizontal fragmentation of a relation is accomplished by a selection operation which places each tuple of the relation in a different partition based on a fragmentation predicate. Vertical fragmentation divides a relation into a number of fragments by projecting over its attributes.

An important issue is the appropriate unit of distribution. The decomposition of a relation into fragments, each being treated as a unit, permits a number of transactions to be executed concurrently. The fragmentation of relations results in distributed execution of a single query by dividing it into a set of subqueries that operate on fragments. Therefore, fragmentation typically increases the level of concurrence of the queries and enables the placement of data in close proximity to its place of use, thus improving the performance of the system. Once the database is fragmented, one has to decide on the allocation of the fragments to various sites on the network. The present work will focus on the fragmentation problem, specifically on vertical fragmentation.

The major information required for vertical fragmentation is related to applications that will run on the distributed databases. Since vertical fragmentation places in one fragment those attributes usually accessed together, there is a need for some measure of "togetherness". This measure is the affinity of attributes, which indicates how closely related the attributes are. It depends on the application and query access frequencies. The heuristic proposed by Navathe et al.[1] is based on the grouping of attributes that have high affinity. Therefore, the problem of vertical fragmentation can be expressed as a maximization problem of the affinity between attributes belonging to the same fragment. The general vertical fragmentation algorithm proposed by Navathe et al. is computationally expensive and its complexity is $O(2^n)$.

Since, genetic algorithms are often viewed as function optimizers, it could be used to solve the fragmentation problem. Genetic algorithms are a family of computational models inspired by evolution. These algorithms encode a potential solution to an specific problem on a simple chromosome-like data structure and apply recombination operators to these structures so as to preserve critical information.

The main aim of this paper is to continue our research [9] on the development of an alternative approach to the problem of vertical fragmentation, starting from the "togetherness" approach proposed by Navathe et al. but including genetic techniques that guarantee a fast moving in the solution space reducing the time to obtain a solution for the vertical fragmentation of a relation. We also want to focus on finding an appropriate unit of fragmentation to obtain balanced fragments.

The rest of this paper is organized as follows. Section 2 introduces some background information. Section 3 describes our alternative approach to the vertical fragmentation problem. Section 4 explains how genetic algorithms are applied to the problem. Implementation and experimental results are presented in Section 5. Finally, conclusions are given in Section 6.

2. BACKGROUND

Let $R(A_1,A_2,...,A_n)$ a relation where $K=\{A_1,...,A_j\}$ is the primary key for R. The vertical fragmentation problem can be stated as finding a set of relations

$$F_R = \{R_1(A_{11},...,A_{1x_1}),...,R_m(A_{m1},...,A_{mx_m})\}$$

such that the following conditions are satisfied:

- Completeness: All the attributes of R must be in at least one fragment

$$\cup_{i=1}^{m}\{A_{i1},...,A_{ix_i}\} = \{A_1,...,A_n\}$$

- Reconstruction: The join over the primary key attributes of the fragments in FR produces the original relation R.

$$R = \triangleright\triangleleft_K R_i, \forall R_i \in F_R$$

- Disjointness: Only the attributes in K can be repeated in all fragments. This will assure the integrity of the relation when reconstructed.

$$\forall i\ \forall j\ \left(1 \le i,j \le m, i \ne j \longrightarrow \{A_{i1},...,A_{ix_i}\} \cap \{A_{j1},...,A_{jx_j}\} = K\right)$$

In a distributed database system, the objective of the vertical fragmentation process is to partition each relation into a set of smaller relations, so that many of the user applications (at least the most frequents) will run on the same fragments reducing the number of page accesses. Nevertheless, for a relation of n non-primary key attributes, the number of possible vertical fragments rounds to the Bell number B(n), which tends to n^n for large values of n. Such a large domain of solutions leads to consider only heuristics to find a solution (not necessarily the optimal).

For the vertical fragmentation, the information required is related to the applications that will run on the distributed databases. Therefore, there is a need for some measure of "togetherness". The measure used is the affinity of attributes which indicates how closely related the attributes are. It depends on the applications and the query access frequencies.

The traditional heuristic used to find a solution to this problem was proposed by [1] for distributed databases as an extension to the centralized database approach proposed on [3]. It starts with the relation to be fragmented and decides on beneficial partitioning based on the access behavior of applications to the attributes. The "togetherness" measure operator defined in these approaches is called Affinity, and it is defined as:

$$aff(A_i, A_j) = \sum_{k \mid use(q_k, A_i)=1 \wedge use(q_k, A_j)=1} \sum_{\forall S_l} ref_l(q_k) acc_l(q_k) \quad (1)$$

where $ref_l(q_k)$ is the number of accesses to attributes (A_i, A_j) for each execution of application q_k at site S_l of the distributed system, $acc_l(q_k)$ is the access frequency on site S_l of the distributed system, and $use(q_k, A_i)$ indicates the use of attribute A_i by the application q_k.

[1] presents an optimization problem (heuristic) using this information to solve the vertical fragmentation problem. The process involves first clustering together the attributes with high affinity for each other, and then splitting the relation accordingly. The problem found with this approach is that the splitting procedure divides the set of attributes two-way. For larger sets of attributes, it is quite likely that m-way partitioning may be necessary. The alternative solution proposed is to recursively applied the binary partitioning algorithm to each of the fragments obtained during the previous iteration. Therefore, designing a m-way partitioning is possible but is computationally expensive. The complexity of such an algorithm is $O(2^m)$ because the algorithm searches the better fragmentation scheme testing each of the possible solutions over the entire space of alternatives. This alternative is a computationally expensive solution that would make unfeasible the inclusion of the heuristic in a tools for automating the designing of distributed databases. [9] presents a m-way partition using genetic algorithms. The fragmentation problem is modeled as a graph partitioning problem before it is solved using genetic algorithms.

3. PROPOSAL

The approach proposed, in this paper, follows the idea presented in [9]. We established a correspondence between vertical fragmentation problem and the undirected k-partitioning graph problem. The correspondence could be stated in the following way:

Consider a graph $G=(V,E)$, where:

- V represents a set of vertices where each one represents an attribute of the relation R to be fragmented
- E represents a set of undirected edges, such that $(Vi, Vj) \in E$ iff the affinity between Vi and Vj is not equal zero.
- The weight of each edge is represented by the affinity measure between the vertices (attributes) that the edge connects.

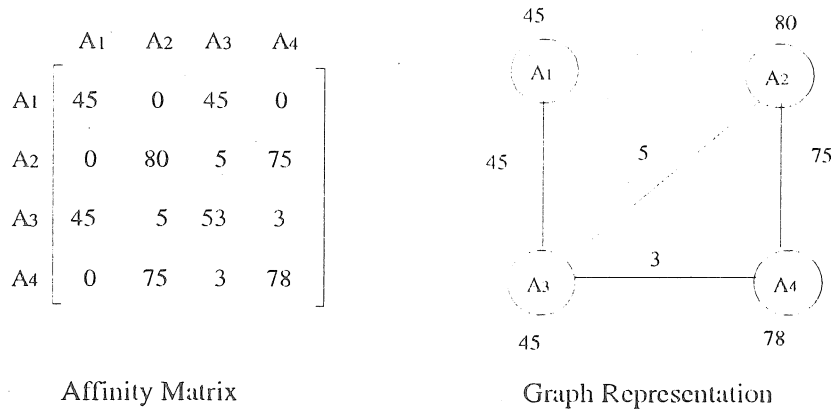

Figure 1: Generated Graph from an affinity matrix

The graph partitioning problem consists of finding an assignment scheme $A: V \rightarrow P$ that maps attributes to partitions. Each partition will correspond to a vertical fragment of the relation.

The desired assignation scheme A should minimize the number of applications that requires to access attributes belonging to different partitions and should try to balance the fragmentation accesses to increase the concurrence level of the applications.

We denote by *Part(t)* the set of vertices assigned to a partition t, i.e.

$$Part(t) = \{v \in V : A(v) = t\} \tag{2}$$

An alternative to find a partitioning of the relation is to try to minimize the cost of the outgoing edges from a partition. We define C as the global affinity measure between attributes belonging to different partitions.

$$C = \sum_{\substack{\forall (V_i, V_j) \in E \\ V_i \in Part(t), V_j \notin Part(t)}} aff(V_i, V_j) \tag{3}$$

where *aff* is the same as defined in (1).

The inconvenient of this approach is that it could assign all the attributes to the same partition, and in this case the cut-size is minimal, or it could assign attributes with lower affinity on the same partition, or one partition could be accessed more than others. We would like to obtain "balanced" partitions in the sense of the number of accesses supported by each partition or fragment.

Therefore, we need to include another term that will help to find a balanced partitioning. Then, for each partition t we define $W_p(t)$ as the global affinity measure between the attributes belonging to the partition t

$$W_p(t) = \sum_{\forall (V_i, V_j) \in E} aff(V_i, V_j) \qquad (4)$$
$$V_i, V_j \in Part(t)$$

It is an approximation of the number of accesses supported by the partition t. Let W be the total affinity measure

$$W = \sum_{\forall (V_i, V_j) \in E} aff(V_i, V_j) \qquad (5)$$

It is an approximated measure of the total accesses supported by the relation to be fragmented. Based on the measures W_p, W and C, we would like to make an assignment such that

$$\sum_t \left(W_p(t) - \frac{W}{N} \right)^2 + C^2 \qquad (6)$$

is minimized where N is the number of partitions or fragments. To avoid that all the attributes are assigned to one partition, we need to include a restriction over N ($N \geq 2$). The first term on (6) finds a balanced partitioning, and it is trying to minimize the difference between the affinity of each partition and the average access. The second term represents the number of accesses that needs more than one partition. It is raised to the power of two because both terms have the same importance and the same weight in the expression.

Contrary to the objective function proposed in [2], our propose does not establish a limitation on the number of fragments to be considered. In that way, to obtain an assignment scheme A such that (6) is minimized over the graph G is equivalent to obtain a vertical fragmentation scheme for the relation R.

Obtaining exact solutions for graph partitioning is computationally intractable, and several suboptimal methods have been suggested [5][6] for finding good solutions to the graph partitioning problem. Genetic algorithms are been used successfully to find suboptimal solutions for a wide variety of problems.

4. GENETIC ALGORITHM

This section describes the representation used to solve the vertical fragmentation problem modeled as a graph partitioning problem, the function to be optimized and the genetic operators defined that exploit domain-specific knowledge to improve the solution and the convergence of the problem.

Representation: we use an integer vector representation for each candidate solution or individual in which the i^{th} element of an individual is j iff the i^{th} attribute of the relation R is allocated to the fragment or partition labeled j. The size of an individual is equal to the number of non key attributes in the relation R. The initial population is randomly initialized.

Fitness Function: the fitness function is a measure, relative to the rest of the population, of how well the individual satisfies a problem-specific metric. In our case, we want to allocate those attributes that are accessed together in the same fragment. We use (6) as fitness function.

Mutation Function: Traditional mutation functions select a random group of individuals that will undergo mutation. The mutation operator will replace a gene *i* of the individual with a value selected uniformly random from 1 to the number of non key attributes in the relation *R*. We use a different approach to implement the mutation function. The mutation operator will replace a gene *i* of the individual with a value that is not selected uniformly random, but with the value of the partition where the most affinity attribute was placed.

Crossover: The crossover operator takes genes from each parent and combines them to create new individuals. The type of crossover commonly used is a two-point crossover, in which the parents *abc* and *def* produce offspring *aec* and *dbf*. The i[th] component of an offspring is chosen to be the same as that of one of the two parents with equal probability. This kind of crossover ignores the fact that one parent may have much better genetic material than the other. In that sense, we use a Knowledge-based Non-Uniform Crossover operator as in [4] and we define a probability vector $pr=(pr_1,...,pr_n)$, where pr_i is a real number $\in [0,1]$. The value of this probability vector depends on the relative fitness of the parents, and on the knowledge about the problem. Given *pr* and the two parent, $a=(a_1,...,a_n)$ and $b=(b_1,...,b_n)$, the offspring c is obtained such that if $a_i = b_i$ then $c_i = a_i$, else the probability that $c_i = a_i$ is pr_i. Let B be the best individual of the last generation. For any candidate solution X, let af(i,X) be defined as the affinity of the partition where the attribute i is assigned

$$af(i,X) = \sum_{\forall j | Part(i)=Part(j)} aff(i,j) \tag{7}$$

and let (i,X,B) be defined as the relation between the affinity of the partition where the attribute i is assigned using the candidate solution X and the affinity of the partition where the attribute i is assigned using the best individual of the last generation

$$(i,X,B) = \frac{af(i,X)}{af(i,B)} \tag{8}$$

If a and b are the two parents, we define de probability vector as

$$pr_i = \begin{cases} 0.5 & if\ (i,a,B)+(i,b,B)=0 \\ \dfrac{(i,a,B)}{(i,a,b)+(i,b,B)} & otherwise \end{cases} \tag{9}$$

5. IMPLEMENTATION AND EXPERIMENTAL RESULTS

In this work was implemented the fitness function (6) and the mutation operator explained in the previous section. The crossover function was not implemented and we use a two-point crossover. Our proposal was

implemented using the PGAPack Parallel Genetic Algorithm Library [7]. PGAPACK is a parallel genetic algorithm library that is intended to provide most capabilities desired in a genetic algorithm package. The heuristic proposed by Navathe et al. extended to m-way partitioning was also implemented in order to test our approach. For this implementation we did not use the parallel facilities of PGAPACK library, we only construct a sequential implementation of the genetic algorithm.

The experiments were carried out using a relation with ten non key attributes, and we obtained the same fragmentation schema using the m-way extension to Navathe et al optimization function as well as our proposed optimization function (6). All the experiments were done with a crossover rate of 85%, mutation rate of 10% and replacement rate of 10%. Table 1 reports different results using both optimization functions. Experiments were conducted varying the population size.

Population Size	*Extended Navathe et al. optimization Function*		*Our optimization Function*	
	Traditional Mutation	Our Mutation Operator	Traditional Mutation	Our Mutation Operator
500	*>700*	70	120	60
400	*>700*	110	100	70
300	*>700*	130	180	110
200	170	110	90	80
100	*>700*	*>500*	*>500*	90

Table 1: Experimental results using different population sizes

A simple inspection of Table 1 shows that our optimization function requires less iterations than the Navathe et al. m-way extended optimization function. Additionally, using the mutation operator defined in this work, the number of genetic sequences (iterations) needed to obtain the expected result is decreased without varying the population size.

It is important to remark that with a population size of 10 individuals, our optimization function using the mutation operator here defined, achieve the expected solution in only 230 iterations. This means that instead of using large population size (with the associated cost time to carry out a genetic sequence over the population), it could be reduced the population size and still obtain solutions as good as the achieve by Navathe's et al. optimization function.

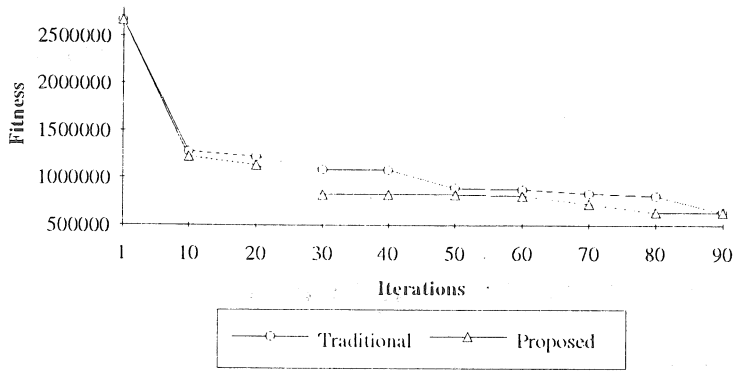

Figure 2: Convergence curves for our fitness function with traditional and our proposed mutation operator.

Figure 2 shows how the novel mutation operator improves the convergence rate compared with the traditional random gene mutation.

6. CONCLUSIONS

Distributed database design has become an important element of effective information management. Design quality is critical to achieve good performance of applications over the database. An important phase of distributed database design is vertical fragmentation process. In this paper, we presented an alternative approach to carry out this process with the following advantages:

- Instead of exhaustively searching over the space of probable solutions giving by the heuristics proposed by Navathe, genetic algorithm restrict the search over the space of possible solutions.

- The strategy proposed in this work is more feasible than the one showed in [2] because it is possible to obtain vertical fragmentation that contains any number of fragments instead of two fragments.

- The strategy proposed is more effective than the one proposed in [9] since it could be used a small population and fewer iterations or generations to obtain the solution of the problem.

Additionally, we have introduced novel operators that exploit the locality information inherent in vertical fragmentation problem. We have shown this enhances the number of genetic sequences (iterations) required to get a good vertical fragmentation schema.

We have presented experimental results showing the feasibility of our approach; unfortunately, fragmenting very large relations does require high amounts of computation by the genetic algorithm and the crossover operator proposed.

We are currently implementing the Knowledge-base Crossover Operator and parallelizing the algorithm to run on distributed memory machines such as the PoweXplorer Parsytec. We are also working on designing a tool that will help the database designer to build and simulate distributed databases. Furthermore, we are studying whether the genetic approach can be used in case of incremental changes (aggregation or elimination of attributes in the original relation) without performing a complete recalculation. The idea we are working on is to begin from the previously achieved fragmentation schema and with a small number of genetic sequences obtain the schema for the new relation.

7. ACKNOWLEDGMENTS

We used a PoweXplorer Parsytec with 8 PowerPC601 modules and T805 each one financed by the project ITDC139 of the European Union.

8. REFERENCES

[1] S.B. Navathe, S. Ceri, G. Wiederhold and J. Dou. *Vertical Fragmentation Algorithms for Database Design.* ACM Transactions on Database Systems. December 1984. Vol 9, No. 4, pages 680-710

[2] M. Özsu, P. Valduriez. *Principles of Distributed Database Systems.*Prentice Hall. 1991.

[3] H.A. Hoffer and D.G. Severance. *The Use of Cluster Analysis in Physical Database Design.* Proceedings of the First International Conference on Very Large Data Bases. Frammingham, Masschusets, September 1975.

[4] H. Maini, K. Mehrotra, C. Mohan, S. Ranka. *Genetic Algorithms for Graph Partitioning and Incremental Graph Partitioning.* IEEE SuperComputing ´94, November 1994.

[5] A. Pothe, H. Simon and K-P Liou. *Partitioning sparse matrices with eigenvectors of graphs.* SIAM J. Matrix Anal. Appl. 11, 3 (July), 1990.

[6] H. Simon. *Partitioning of unstructured mesh problems for parallel processing.* Proc. Conf. Parallel Methods on Large Scale Structural Analysis and Physics Applications, Pergamon Press, 1991.

[7] D. Levine. *Users Guide to the PGAPACK Parallel Genetic Algorithm Library.* January 1996.

[8] D. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning.* Addison-Wesley Publishing Company, 1989.

[9] J. Anciano, E. Arráiz, A. Di Serio, N. Savino. *Alternative Approach for Distributed Database Design.* To be published on World Multiconference on Systemics, Cybernetics and Informatics (SCI´97). July 1997.