

Vertical partitioning impact on performance and manageability of distributed database systems (A Comparative study of some vertical partitioning algorithms)

Vertical partitioning impact

1

Hassan I. Abdalla

Computer Science Dept., Prince Sultan University, Riyadh, Saudi Arabia

F. Marir

Dept of Computing, Communications Technology and Mathematics, London Metropolitan University, UK

Keywords Distribution, Fragmentation, Query, Affinity, Distributed Database Design

Abstract Users of distributed database systems often observe performance problems such as unexpectedly low throughput or high latency. Determining the cause of the performance problems can be very hard task. Bottlenecks can occur in any of the components through which the data flows: the applications, the operating systems, the network interfaces and hardware. Horizontal and vertical partitioning are important aspects of physical design in relational database system that has a significant impact on performance. The distribution design involves making decisions on the fragmentation and the allocation of data across the sites of a computer network. In this paper we address the fragmentation phase of distributed database systems. In this paper, vertical partitioning problem during the design of distributed databases is discussed by conducting a comparative study for different vertical partitioning algorithms to reach the most efficient vertical fragmentation scheme that leads to a proper data allocation and replication.

Introduction

The advent of telecommunication era and the constant development of hardware and network structures have encouraged the decentralization of data while increasing the needs to access information from different sites leading to great advances in distributed database systems. A distributed database system is a collection of sites connected on a common high-bandwidth network. Logically, data belongs to the same system but physically it is spread over the sites of the network, making the distribution invisible to the user (Ceri and Weiderhold. 1989). Each site is autonomous database with its processing capability and data storage capacity. The advantage of this distribution resides in achieving availability, modularity, performance, and reliability.

Distributed and parallel processing on database management systems (DBMS) is an efficient way of improving performance of applications that manipulate large volumes of data. This may be accomplished by removing irrelevant data accessed during the execution of queries and by reducing the data exchange among sites, which are the two main goals of the design of distributed databases (M. Özsu and P. Valduriez, 1999).

The primary concern of distributed database systems is to design the fragmentation and allocation of the underlying database. The distribution design involves making decisions on the fragmentation and placement of data across the sites of a computer network. The first phase of the distribution design in a top-down approach is the fragmentation phase, which is the process of clustering into fragments the information accessed simultaneously by applications. The fragmentation phase is then followed by the allocation phase, which handles the physical storage of the generated fragments among the nodes of a computer network, and the replication of fragments.

Related Work

Vertical partitioning is the process that divides a relation into sub-relations called vertical and horizontal fragments (but the focus in this paper is on vertical fragmentation process), containing subsets of the original attributes (Ceri and Weiderhold. 1989), (Navathe, Ceri, Weiderhold, and Dou 1984) and (Muthuraj, 1992). Most of the vertical fragmentation algorithms have started from constructing an attribute affinity matrix from the attribute usage matrix: the Attribute affinity matrix is an n x n matrix for the n-attribute problem whose (i, j) element equals the "between-attributes" affinity which is the total number of accesses of transactions referencing both attributes i and j. An iterative binary partitioning method has been used in (Navathe, Ceri, Weiderhold, 1984) and [Cornell, and Yu., 1987] based on first clustering the attributes and then applying empirical objective functions or mathematical cost functions to perform the fragmentation.

18th National Computer Conference 2006 © Saudi Computer Society NCC18

2

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. Attribute partitioning and attribute clustering have been studied earlier by (Baião, 2001), (Babad, 1977), (Eisner and Severance, 1976), (Hoffer, 1976), (Hammer and Niamir, 1987), (Navathe et al., 1984), and (Navathe and Ra, 1989). (Stocker and Dearnley, 1973) have discussed the implementation of a self-reorganizing database management system that carries out attribute clustering. They also show that in a database management system where storage cost is low compared to the cost of accessing the subfiles, it is beneficial to cluster the attributes, since the increase in storage cost will be more than offset by the saving in access cost. (Ceri, and Pernici, 1989) considers a mathematical model of attribute partitioning where each attribute ai is of known length, and has probability pi of being requested by a query.

The joint probability that attributes ai and aj are requested by the same query is assumed to be pipj. A cost function based on this assumption is derived, which reflects the expected amount of data that must be transmitted in order to answer the query. The objective here is to choose a partition such that this cost function is minimized. (Hoffer, 1976) developed a non-linear, zero-one program, which minimizes a linear combination of storage, retrieval and update costs, with capacity constraints for each file.

(Hammer and Niamir, 1979) 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, 1984) used a two-step approach for vertical partitioning. In the first step, they used 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, 1987) propose an algorithm, as an extension of (Navathe et al, 1984) 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.

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 distributed database design, the number of clusters is an important factor that influences the trade-off between local and remote transaction processing costs.

Al Critical Assessment to Some Vertical Partitioning Algorithms:

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 a matrix, which has attributes as columns, and queries as rows and the accesses frequency of the queries as values in the matrix. Most of earlier data fragmentations 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 queries 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 a mechanism to evaluate the "goodness" of partitions that they produce.

The input to the vertical partitioning algorithm is an attribute usage matrix. An example of AUM is given in table-1 below:



partition.						es	ribute	Att				
Imp	Access	10	9	8	7	6	5	4	3	2	1	Transactions
	Freq											
	$Acc_1=25$	0	0	0	1	0	1	0	0	0	1	T1
	$Acc_2=50$	0	1	1	0	0	0	0	1	1	0	T2
Table 1.	$Acc_3=25$	1	0	0	0	1	0	1	0	0	0	Т3
Attribute	$Acc_4=35$	0	0	1	1	0	0	0	0	1	0	T4
Matrix	$Acc_5=25$	0	1	1	1	0	1	0	1	1	1	T5
1/10/11/1	$Acc_6=25$	0	0	0	0	0	1	0	0	0	1	T6
	Acc ₇ =25	0	1	0	0	0	0	0	1	0	0	Τ7
	$Acc_8=15$	1	1	0	0	1	0	1	1	0	0	T8

Algorithms such as Bond Energy, Binary Vertical Partitioning algorithm use the Attribute Affinity Matrix (AAM) formed from the Attribute Usage Matrix (AUM). Attribute affinity measures the bond between two attributes of a relation according to how they are accessed by applications.

Attribute affinity between attributes *i* and *j* is defined as: $Affij = \sum_{t=1}^{T} {}^{q}t, ij \qquad (1)$

where *qt,ij* is the number of accesses of transaction *t* referencing both attributes *i* and *j*.

The attribute affinity matrix is given in table 2 below.

Attributes	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	0	40
5	75	25	25	0	75	0	50	25	25	0
6	0	0	15	40	0	40	0	0	0	40
7	50	60	25	0	50	0	85	60	60	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

Bond Energy Algorithm

The Bond Energy Algorithm (BEA) (Navathe, Ceri, Weiderhold, 1984) is used to group the attributes of a relation based on the attribute affinity values in AAM. It is considered appropriate for the following reasons (Muthuraj, 1992):

- It is designed specially to determine groups of similar items as opposed to a linear ordering of the items. (ie. It clusters the attributes with larger affinity values together, and the ones with smaller values together).
- The final groupings are insensitive to the order in which items are presented to the algorithm.
- The AAM is symmetric, and hence allows a pairwise permutation of rows and columns, which reduces complexity.
- Because of the definition of *Affij*, the initial AAM is already semiblock diagonal, in that each diagonal element has a greater value of any element along the same row or column.
- The computation time of the algorithm is reasonable. $O(n^2)$, where *n* is the number of attributes.

Table 2. Attribute Affinity Matrix



4

This algorithm takes as input the attribute affinity matrix, permutes its rows and columns and generates a clustered affinity matrix (CAM). The permutation is done in such a way to maximize the following global affinity measure (AM).

$$AM = \sum_{i=1}^{n} \sum_{j=1}^{n} Aff_{i,j} \left[Aff_{i,j-l} + Aff_{i,j+l} + Aff_{i-l,j} + Aff_{i+l,j} \right]$$
where $Aff_{0,j} = Aff_{i,0} = Aff_{n+l,j} = Aff_{i,n+l} = 0$
(2)

And since the attribute affinity matrix is symmetric, then the objective function of the formulation above could be reduced to:

$$AM = \sum i=1 \quad \sum j=1 \quad Aff_{i,j} \left[Aff_{i,j-l} + Aff_{i,j+l}\right]$$
(3)

The last set of conditions takes care of the cases where an attribute is being placed in CAM to the left of the leftmost attribute or to the right of the rightmost attribute during column permutations, and prior to the topmost row and following the last row during row permutations.

Before explaining the algorithm we have to define some more quantities. Let us define the bonc between two attributes A_i and A_j as:

$$bond_{i,j} = \sum_{z=1}^{n} Aff_{z,i} Aff_{z,j}$$
(4)

The net contribution to the global affinity measure of placing the attribute k between A*i* and A*j* is:

 $Cont_{ikj} = 2bond_{ik} + 2bond_{kj} - 2bond_{ij}$ ⁽⁵⁾

Generation of the Clustered Affinity Matrix is done in three steps:

n

Initialization: Place and fix one of the columns of AAM arbitrarily into CAM.

Iteration: Pick each of the remaining *n*-*i* columns (where i is the number of columns already placed in CAM) and try to place them in the remaining i+1 positions in the CAM matrix. Choose the placement that makes the greatest contribution to the global affinity measure described above. Continue this until no more columns remain to be placed.

Row Ordering: Once the column ordering is determined, the placement of the rows should also be changed so that their relative positions match the relative positions of the columns.

When the CAM is big, usually more than two clusters are formed and there are more than one candidate partitions. After applying the BEA on the above AAM, the resulting CAM will be:

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

Table 3. Clustered Affinity Matrix



Binary Vertical Partitioning Algorithm

(Navathe et al., 1989) extended the results of (Hoffer and Severance, 1975) by giving algorithms p to quantitatively cluster the attributes together and taking into account blocks of attributes with similar properties. The binary vertical partitioning algorithm uses the clustered affinity matrix to partition an object into two non-overlapping fragments. The approach of this algorithm is splitting rather than grouping with the objective of finding sets of attributes that are accessed mostly by distinct set of applications.

The binary vertical partitioning algorithm uses the clustered affinity matrix to partition an object into two non-overlapping fragments. Assume that point x is fixed along the main diagonal of the clustered affinity matrix, as shown in table 4. The point x defines two blocks: upper (U) and lower (L). Each block defines a vertical fragment given by the set of attributes in that block.



Attributes	5	1	7	2	8	3	9	10	4	6
5	75	75	50	25	25	25	25	0	0	0
1	75	75	50	25	25	25	25	0	0	0
7	50	50	85	60	60	25	60	0	0	0
2	25	25	60	110	110	75	75	0	0	0
8	25	25	60	110	110	75	75	0	0	0
3	25	25	25	75	75	115	115	15	15	15
9	25	_25	25_	_75	_75	_115	<u> 115 </u>	15_	_15	15
10	0	0	0	0	0	15	15	x40	40	40
4	0	0	0	0	0	15	15 L	40	40	40
6	0	0	0	0	0	15	15	40	40	40

Table 4. Partitioned Attribute Affinity Matrix

If A_t is the set of attributes used by transaction t, then it is possible to compute the following sets:

T = (t|t is a transaction) $LT = (t|At \in L)$ $UT = (t|At \in U)$ $IT = T - (LT \cup UT)$

T represents the set of all transactions. LT and UT represent the set of transactions that match the partitioning, as they can be entirely processed using attributes in the lower or upper block, respectively; IT represents the set of transactions that needs to access both fragments.

Vertical

5

partitioning impact 6

CT counts the total number of transaction accesses to the considered object. CL and CU count the total number of accesses of transactions that need only one fragment; CI counts the total number of accesses of transactions that need both fragments. Totally n-1 possible locations of point x along the diagonal is considered, where n is the size of the input matrix (i.e. the number of attributes). A non-overlapping partition is obtained by selecting the point x along the diagonal such that the following objective function z is maximized:

 $\max z = CL^*CU - CI^2 \tag{6}$

The partition that corresponds to the maximal value of the z function is accepted if z is positive and rejected otherwise. The above objective function comes from an empirical judgment of what should be considered a "good" partitioning. The function is increasing in CL and CU and decreasing in CI. For a given value of CI, it selects CL and CU in such away that the product CL * CU is maximized.

This results in selecting values for CL and CU that are as nearly equal as possible. Thus the above function z will produce fragments that are "balanced" with respect to the transaction load. This algorithm has the disadvantage of not being able to partition an object by selecting out an embedded "inner" block.

Limitations of the Bond Energy and Binary Vertical Portioning Algorithms

• All the aforementioned Algorithms use affinity matrix as input and because the attribute affinity is a measure of an imaginary bond between a pair of attributes, this measure does not reflect the closeness or affinity when more than two attributes are involved.

• In the BEA the creation of partitions is left to the subjective evaluation of the designer.

• There is no common criterion or objective function to compare and evaluate the results of these vertical partitioning algorithms.

• The above algorithms assumes that there will always be a possibility of an (n-1) partitioning for a relation R, ignoring the fact that there could be a situation where considering the entire relation R as one fragment could be the optimum solution, i.e. having an (n-0) partition possibilities.

Graph-based vertical partitioning

A new algorithm has been developed by Navathe and Ra based on a graphical technique (Navathe and Ra, 1989). This algorithm starts from the attribute affinity matrix by considering it as a complete graph called the "affinity graph" in which an edge value represents the affinity between the two attributes, and then forms a linearly connected spanning tree. By a "linearly connected tree" we imply a tree that is constructed by including one edge at a time such that only edges at the "first" and the "last" node of the tree would be considered for inclusion. We then form "affinity cycles" in this spanning tree by including the edges of high affinity value around the nodes and "growing" these cycles as large as possible. After the cycles are formed, partitions are easily generated by cutting the cycles apart along "cut-edges".

The major feature of this algorithm is that all fragments are generated by one iteration in a time of $O(n^2)$ that is more efficient than the previous approaches.

Recalling table 1 of page 3, the attribute usage matrix for a relation containing 10 attributes with respect to 8 transactions, namely, T1 through T8 that are initiated by the applications. Table 2 shows an example of an attribute affinity matrix. Figure 1 shows the result of applying the algorithm to the attribute affinity matrix. In Figure 1 the nodes refer to attributes of the relation.

The resulting vertical fragments are: 1. (a1, a5, a7) 2. (a2, a3, a8, a9) 3. (a4, a6, a10)





We can summarize the major advantages of this method over the previous approaches in the following:

1. There is no need for iterative binary partitioning. The major weakness of iterative binary partitioning used in (Navathe, Ceri, Weiderhold, 1984) is that at each step two new problems are generated increasing the complexity; furthermore, termination of the algorithm is dependent on the discriminating power of the objective function.

2. The method obviates the need for using any empirical objective functions as in (Navathe, Ceri, Weiderhold, 1984). As shown by (Cornell and Yu, 1987) the "intuitive" objective functions used in (Navathe, Ceri, Weiderhold, 1984) do not necessarily work well when an actual detailed cost formulation for a specific system is utilized.

3. The method requires no complementary algorithms such as the SHIFT algorithm of (Navathe, Ceri, Weiderhold, 1984) that shifts the rows and columns of the affinity matrix.

4. The complexity of this approach is $O(n^2)$ as opposed to $O(n^2\log(n))$ in (Navathe, Ceri, Weiderhold, 1984).

Contributions and Enhancements

This paper has shown how Graph-based vertical partitioning algorithm has contributed towards the optimization of data fragmentation problem by providing an efficient way of improving performance of applications.

Several contributions and suggestions to the enhancements of data fragmentation problem in general and the design of vertical partitioning in particular are considered in this paper, among those are:

1. Examining whether the criteria used in the data-clustering domain could be adapted, with some changes, to the data fragmentation problem.

2. Studying the applicability of some data clustering algorithms for distributed database design to data fragmentation problem.

3. Outlining a design strategy for n-ary partitions, with the desirable behavior for minimizing queryprocessing cost.

4. Recommending an algorithm that will bring us closer to achieve the ideal objective of having any query to access only the attributes in a single data fragment with no or minimal access of irrelevant attributes in that fragment.

NCC18

5. Finally, describing how partitioning can provide the functionality that enables each site to process the queries locally with minimal access to data located at remote sites.

Conclusions and future work

In this paper, we have presented different approaches that handle vertical fragmentation problem during the design of distributed databases by evaluating different vertical partitioning algorithms. We have compared BEA algorithm, Binary vertical algorithm and Graph based vertical partitioning algorithm and brought out the problems associated with the use of attribute affinity matrix currently used in almost all of the earlier data partitioning algorithms.

Implementation results have shown that the Graph based vertical partitioning algorithm offers better performance for query processing time. Experiments using Graph-based vertical partitioning resulted in fragmentation schemas with better performance results when compared to other fragmentation schemas proposed in the literature. The main contribution of this paper is in providing a clear guidance to choose the most adequate fragmentation technique to be applied in each relation of the database schema. Our study did not cover horizontal fragmentation algorithms which play an important role in optimizing queries and maintenance overhead and which could be covered in future work.

Data partitioning in general, and vertical partitioning in particular, presents many new opportunities for distributed and parallel computing. Several new challenges for data partitioning are present. First, new algorithms are needed to maintain database consistency since maintenance is performed by a set of distributed computing components that receive data from autonomous sources. Second, the parallelization of maintenance tasks is an important research area, since there are many available choices and the performance implications are significant.

References

M. Özsu and P. Valduriez, Principles of Distributed Database Systems, 2nd edition (1 st edition 1991), New Jersey, Prentice-Hall, 1999.

- M. Babad. A record and file partitioning model. Commun. ACM 20, 1(Jan 1977).
- F. Baião "A Methodology and Algorithms for the Design of Distributed Databases using Theory Revision" D.Sc. Thesis, COPPE/UFRJ, Dec 2001. (http://www.cos.ufrj.br/~baiao/thesis/baiaoDSc.pdf).
- 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.
- Y. Chen and S. Su, "Implementation and Evaluation of Parallel Query Processing Algorithms and Data Partitioning Heuristics in Object Oriented Databases", International Journal of Distributed and Parallel Databases, Kluwer Academic Publishers, vol. 4(2), 1996, pp. 107-142.
- D. Cornell, and P. Yu. A Vertical Partitioning Algorithm for Relational Databases. Proc. Third International Conference on Data Engineering, Feb. 1987.
- M. Eisner, and D. Severance. Mathematical techniques for efficient record segmentation in large shared databases. J. ACM 23, 4(Oct. 1976).
- 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.
- J. Hoffer, and D. Severance. The Uses of Cluster Analysis in Physical Database Design In Proc. 1st International Conference on VLDB, Framingham, MA, 1975.
- J. Hoffer. An integer programming formulation of computer database design problems. Inf. Sci., 11(July 1976), 29-48.
- J. Kittler. A locally sensitive method for cluster analysis. Pattern Recognition 8, 22-33.
- 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.
- S. Navathe, and M. Ra. Vertical Partitioning for Database Design: A Graphical Algorithm. ACM SIGMOD, Portland, June 1989.
- S. Navathe, S. Ceri, G. Weiderhold, and J. Dou. Vertical Partitioning Algorithms for Database Design ACM Transactions on Database Systems, Vol. 9, No. 4, 1984.
- B. Niamir. Attribute Partitioning in Self-Adaptive Relational Database System. Ph. D. Dissertation, M.I.T. Lab. for Computer Science, Jan. 1978.
- M. Stocker and A. Dearnley. Self-organizing Data Management Systems Com-puter Journal. 16, 2(May 1973).