VALLEY BASED VERTICAL PARTITIONING IN DISTRIBUTED DATABASE

Sumti Medhavi and Akhilesh Kumar

Department of Computer Science and Engineering, Kanpur Institute of Technology, Kanpur-208001, India.
Email ID: sumitmedhavi_k@rediffmail.com, sumitmedhavi@gmail.com, Ph. +919473883031

Abstract

A Vertical Partitioning approach is defined as the method of dividing the attributes of a relation. An efficient Vertical Partitioning method always puts frequently accessed attributes of a relation together in a fragment. Various Partitioning algorithms have been proposed by several researchers. Still there is a scope of further improvement in previously developed algorithms. In this paper a new algorithm is proposed for Vertical Partitioning in Distributed Database. The proposed algorithm is named as Valley Based Vertical Partitioning Algorithm (VBVPA).This algorithm makes use of Clustered Affinity Matrix (CAM), which is derived using Attribute Usage Matrix (AUM) and Frequency Matrix (FM).

Keywords: Vertical Partitioning or Fragmentation, Attribute Usage Matrix, Frequency Matrix, Attribute Affinity Matrix, Bond Energy Algorithm, Bond Matrix, Clustered Affinity Matrix.

1. INTRODUCTION

In Distributed Database, the fragments of a relation are distributed over the collection of independent sites. Further, it may be possible that the queries may not be able to access the attributes locally. Hence, there is a requirement of communication to other sites to access the required attributes or result. Frequent number of communication to other sites in a distributed system results in increase in Query Response Time (QRT). A Vertical Partitioning approach of relation plays a vital role in enhancing the Query Response Time (QRT). An efficient Vertical Partitioning Approach divides the attributes of a large relation into smaller fragments, thus improving Query Response Time (QRT). Frequently accessed fragments of a relation are stored in main memory resulting in reduction of page accessing from secondary memory. Further in a distributed database system a query can also be divided into sub-queries resulting in concurrent execution on different fragments.

There are basically two approaches for partitioning of a relation namely Horizontal Partitioning and Vertical Partitioning. Horizontal Partitioning divides a relation into smaller fragments on the basis of rows. Each fragment contains equal number of columns or attributes, but the number of rows is reduced. Vertical Partitioning is an approach of dividing a relation into smaller fragments on the basis of columns. Each fragment contains less number of columns.

Since a query does not necessarily requires all the attributes of a relation at the same time. So the Vertical Partitioning approach is more effective in enhancing Query

Response Time (QRT) than Horizontal Partitioning approach. In this paper a new Vertical Partitioning algorithm named as VBVPA is proposed for Vertical Fragmentation.
The input to this algorithm is Clustered Affinity Matrix (CAM) which is calculated using the values of Attribute Usage Matrix (AUM) and Frequency Matrix (FM). After the calculation of Clustered Affinity Matrix (CAM), the fragments of a relation is created using VBVPA taking Clustered Affinity Matrix (CAM) as input. This algorithm partitions the attributes of a relation where a valley will be formed.
Rest of the paper is organized as follows. In section 2 previous works on partitioning has been reviewed. In section 3 technique used in VBVPA for vertical fragmentation has been discussed. In section 4 and 5 experimental set and result has been described. Finally section 6 contains the conclusion and future scope.

2. LITERATURE REVIEW

Since early 1970s, minimization of disk I/O has been an important concern. From that time various partitioning algorithm have been devised to reduce I/O accessing through clustering of attributes of a large relation. This results in the reduction of page accessing from secondary memory.
In 1972, McCormick et al in [4] developed the first algorithm for clustering named as Bond Energy Algorithm (BEA). The purpose of this algorithm was to identify the clusters in a complex relation. The limitation of this algorithm was that it required human interpretation to implement. Sometimes blocks overlapping might be possible and also some blocks may not contain the required elements. So this method of clustering was not considered efficient.
In 1984, after the introduction of BEA, a new algorithm was developed by Navathe et al. in [5]. This algorithm uses frequency of queries first time and reflects the values of Frequency Matrix in Attribute Affinity Matrix (AAM) on the basis of which clustering was performed. The complexity of this algorithm was O (n^2) where n denotes the number of times the partitioning was repeated. The complexity can also increase if the overlapping was allowed.
The Optimal Binary Vertical Partitioning Algorithm was proposed by Wesley W. Chu et.al in [7]. It used the branch and bound technique [3] to make binary tree whose nodes represented the query. This algorithm reduced time complexity as compared to Navathe et.al. in [6]. The drawback of this algorithm was that it didn't consider the impact of frequency of query and also its running time were increasing with the number of queries.
The Graph Traversal Vertical Partitioning was proposed by Navathe et.al. in [6] in 1989. This algorithm traversed the graph and divided the graph into several sub graphs, each of which represented a cluster. The problem in this algorithm was that the frequent and infrequent queries were given the same priority, which might result in inefficient partitioning. This was due to fact that the attributes that were usually accessed together in infrequent queries but were not accessed in frequent queries might be put into same fragment.
The Eltayeb's Optimized Vertical Partitioning Scheme [1] was also based on Attribute Affinity Matrix [5]. This algorithm started with a vertex V that satisfied the minimum degree of Reflexivity and then searched a vertex with the maximum degree of symmetry among V's neighbors of the most recent V recursively until a cycle was formed or no vertex was left. The next step was to calculate the hit ratio of partition. If the partition hit ratio was less than predefined Threshold then identify the attribute with the

minimum hit to miss ratio and moved it to a different subset. The limitation of this algorithm was similar to the Graph Traversal Vertical Partitioning algorithm that the infrequent queries were considered same as frequent queries.

3. DESCRIPTION OF VALLEY BASED PROCEDURE

In this section Valley Based Vertical Partitioning Algorithm (VBVPA), used for fragmentation of a relation is discussed in detail. First of all by using the values of Attribute Usage Matrix (AUM) and Frequency Matrix (FM), Clustered Affinity Matrix (CAM) is calculated. Then VBVPA is used for partitioning of a relation.

3.1 Attribute Usage Matrix (AUM)

The Attribute Usage Matrix denotes which attributes of a relation are used by a query. For each combination of row and column of this matrix has only one of the two values either 0 or 1. As shown in the Table 1, the value 1 in the matrix denotes that the attribute A_I is queried by query Q_J otherwise 0 is associated.

$$USE(Q_I, A_J) = \begin{cases} 1, & \text{if Attribute } A_I \text{ is used by Query } Q_I \\ 0, & \text{otherwise} \end{cases} \quad (1)$$

Table 1. Attribute Usage Matrix

Query	Attribute			
	A_1	A_2	A_3	A_4
Q_1	1	0	1	1
Q_2	0	1	0	1
Q_3	1	0	1	0
Q_4	0	1	1	1

3.2 Frequency Matrix (FM)

This matrix denotes the number of times a query is fired from a particular site as shown in Table 2 below.

Table 2. Frequency Matrix

Query	Site		
	S_1	S_2	S_3
Q_1	10	15	5
Q_2	7	4	0
Q_3	12	15	20
Q_4	4	3	2

3.3 Attribute Affinity Matrix (AAM)

This matrix denotes the strength of an imaginary bond between two attributes of a relation. It is based on the fact that two attributes are used together by a query. The value of this matrix represents the number of times two attributes are accessed together at all the sites.

Attribute Affinity Value between A_I and A_J is represented as Aff (A_I, A_J).

$$\text{Aff}(A_I, A_J) = \sum_{\text{all queries that access } A_I \text{ and } A_J} \text{Query access} \quad (2)$$

Where

Query access (Q_1) = 30
Query access (Q_2) = 11
Query access (Q_3) = 47
Query access (Q_4) = 9

$$\text{Aff}(A_I, A_J) = \sum_{Q_1} \text{query access} = 30$$

In the same way all the other Attribute Affinity values are calculated as shown in Table 3.

Table 3. Attribute Affinity Matrix

Attribute	Attribute			
	A_1	A_2	A_3	A_4
Q_1	1	0	1	1
Q_2	0	1	0	1
Q_3	1	0	1	0
Q_4	0	1	1	1

3.4 Clustered Affinity Matrix (CAM)

For fragmentation of attributes in a relation, the attributes must be clustered. Clustering problem is widely researched in data mining, databases and statistics communities [8], [9], [10], [11], [12], [13]. Hover and Severance in [2] has proposed that a Bond Energy Algorithm (BEA) should be used for clustering. This algorithm takes Attribute Affinity Matrix as input, changes the order of its rows and columns, and produces a Clustered Affinity Matrix (CAM) as output. The Bond Energy Algorithm clusters the attribute which have high Attribute Affinity value. Bond Energy Algorithm has been implemented in three steps.

Initialization: In this step, first two columns of Attribute Affinity Matrix (AAM) are placed directly to the respective columns in the Clustered Affinity Matrix (CAM).

Iteration: After the initialization step, the remaining attributes (N-1) are picked one by one and try to place them in remaining (I+1) positions in Clustered Affinity Matrix. The placement is done on the basis of Global Affinity Measure. This process is continued until no more columns attribute remains to be placed.

Row Ordering: Once the placement of attribute in column is determined, the placement of row attributes should also be changing so that their relative positions match the relative positions of the columns attributes.

BEA is used to get the position of attributes in Clustered Affinity Matrix (CAM). The attribute is placed to a position where the contribution of attribute placement is highest.

3.4.1 Placement of attributes in CAM

Placement of A_1 and A_2:
In the initialization step first and second columns of Attribute Affinity Matrix (AAM) are placed in the first and second columns of Clustered Affinity Matrix (CAM) respectively.
Attribute A_1 is placed at position 1 in CAM: [A_1]
Attribute A_2 is placed at position 2 in CAM: [A1, A2]

Placement of A_3:
Attribute A_3 is placed at position 1 in CAM =27442
Attribute A_3 is placed at position 2 in CAM =28324
Attribute A_3 is placed at position 3 in CAM =3468
Attribute A_3 is placed at position 2 in CAM: [A_1, A_3, A_2]

Placement of A_4:
Attribute A_4 is placed at position 1 in CAM =13626
Attribute A_4 is placed at position 2 in CAM =1772
Attribute A_4 is placed at position 3 in CAM =15622
Attribute A_4 is placed at position 4 in CAM =3502
Attribute A_3 is placed at position 2 in CAM: [A_1, A_3, A_4, A_2]

Hence in the Clustered Affinity Matrix (CAM), the order of placement of attributes in rows and columns are shown below:

Table 4. Clustered Affinity Matrix

Attribute	Attribute			
	A_1	A_3	A_4	A_2
A_1	77	77	30	0
A_3	77	86	39	9
A_4	30	39	50	20
A_2	0	9	20	20

3.5 Valley Based Vertical Partitioning Algorithm

The objective of this algorithm is to search the set of frequently accessed attributes by a distinct set of queries. Using the Valley Based Vertical Partitioning Algorithm, user fragments a relation based on Clustered Affinity Matrix (CAM), calculated from Attribute Usage Matrix (AUM) and Frequency Matrix (FM). In this algorithm, first row of the Clustered Affinity Matrix (CAM) is taken as input to find the clusters of attributes in a relation. Further, we calculate the difference between neighboring attribute values of first row of Clustered Affinity Matrix (CAM) and the point at which the current differentiated value is less than the previous and the next differentiated value is considered as split point. The pseudo code for the VBVPA is given below:

Algorithm: VBVPA
Input: CAM: Clustered Affinity Matrix
Output: P: set of fragments
Begin
[Initialize Variables]

```
X [N];          // Used to store value from 1 to N of loop in corresponding index.
Y [N];          //Used to store the differentiated values.
splitPoint = 0; //Used to store point of split.
```

[Evaluating the split point]

```
Y [1] = CAM (1, 1);
X [1] = 1;
For i = 2 to n do
                Y [i] = CAM (1, i) – CAM (1, i-1);
                X [i] = i;
End- For
splitPoint = 1;
For i = 2 to n do
                If (Y [i] < Y [i-1] && Y [i] < Y [ i+1]) then
                        splitPoint = X[i];
                End-If
End-For
End-Begin
```

This algorithm mainly involves three steps:
Initialization: In this step the variables and arrays are initialized as required by algorithm.

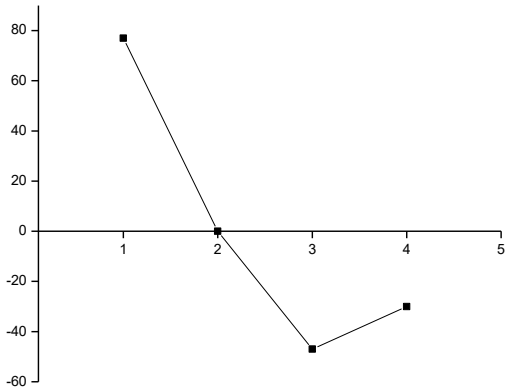
Processing: In this step first row of Clustered Affinity Matrix (CAM) are taken as input for finding the clusters of attributes in a relation. The user keeps the first value of row as it is in Y[1] and then finds the difference of remaining CAM(1,i) values and stores it in Y[i].

Table 5. First Row Of Clustered Affinity Matrix

Attribute	Attribute			
	A_1	A_3	A_4	A_2
A_1	77	77	30	0

Comparison: In this step the user compares each value of array Y[i] with the immediate previous and next values of Y[i], wherever the current value is found less than previous and next values that point is considered as split-point. The following calculation is performed with referenced to CAM.

The graph below shows the Valley value Y [i] at point i.



Y [1] = CAM (1, 1) = 77, X [1] = 1
Y [2] = CAM (1, 2) = 77-77= 0, X [2] = 2
Y [3] = CAM (1, 3) = 30-77= -47, X [3] = 3
Y [4] = CAM (1, 4) = 0-30= -30, X [4] = 4

The graph above shows the valley formed between the values X [2] =2 and X [4] =4 i.e. at point X [3] =3.So the split point is recorded between second and third attribute of CAM. Hence the Clustered Affinity Matrix is divided into two fragments. The first fragment contains the attributes {A_1, A_3} while the second fragment contains the attributes {A_4, A_2}.

4. EXPERIMENTAL SETUP

An experiment performed to test the working of the proposed Valley Based Partitioning Algorithm (VBVPA). It has been carried out on a system with CORE i4 processor, 4GB RAM, NetBeans 7.3(Java 1.6)Tool, Oracle 10g database. A relation naming Employee has been used for fragmentation. This relation has been stored in oracle 10g database as following.

Table 6. Employee

ENO	ENAME	SALARY	CITY
E1	RAM	20000	NEW YORK
E2	SHYAM	15000	LAS VEGAS
E3	MOHAN	25000	NEW YORK
E4	SUMIT	10000	CHICAGO

The following are the queries generated from three sites named S_1, S_2, S_3 .
Q_1: Find the employee number and their sum or salaries given their city name. (SELECT ENO, SUM (SALARY) FROM EMPLOYEE WHERE CITY = value).
Q_2: Find the name of the employees from a given city name. (SELECT ENAME EMPLOYEE WHERE CITY= value)
Q_3: Find the salary of an employee whose name is given. (SELECT SALARY FROM EMPLOYEE WHERE ENAME = value).
Q_4: Find the name of employees and their salaries where city name is given. (SELECT ENAME, SALARY FROM EMPLOYEE WHERE CITY = value).

The Attribute Usage Matrix of the relation named Employee is

Table 7. Attribute Usage Matrix Employee

Query	Attribute			
	ENO	ENAME	SALARY	CITY
Q_1	1	0	1	1
Q_2	0	1	0	1
Q_3	1	0	1	0
Q_4	0	1	1	1

The Frequency of queries Q_1, Q_2, Q_3, Q_4 at three sites S_1, S_2, S_3 is

Table 8. Frequency Matrix Employee

Query	Site		
	S_1	S_2	S_3
Q_1	10	15	5
Q_2	7	4	0
Q_3	12	15	20
Q_4	4	3	2

5. EXPERIMENTAL RESULT

Clustered Affinity Matrix (CAM) is calculated from Attribute Usage Matrix (AUM), Frequency Matrix (FM) and also by using Hoffer and Severance Bond Energy Algorithm (BEA) in [2]. The CAM is shown in Table 9 below.

Table 9. Clustered Affinity Matrix Employee

Attribute	Attribute			
	ENO	SALARY	CITY	ENAME
ENO	77	77	30	0
SALARY	77	86	39	9
CITY	30	39	50	20
ENAME	0	9	20	20

Now by taking first row of Clustered Affinity Matrix (CAM) as input in Valley Based Vertical Partitioning Algorithm (VBVPA), two fragments of relation Employee has been found. The first fragment contains the attributes ENO and SALARY whereas second fragment has attributes CITY and ENAME.
Below Table 10 and 11 shows the two resulting fragments.

Table 10.

ENO	SALARY
E1	20000
E2	15000
E3	25000
E4	10000

Table 11.

CITY	ENAME
NEW YORK	RAM
LAS VEGAS	SHYAM
NEW YORK	MOHAN
CHICAGO	SUMIT

6. CONCLUSION AND FUTURE SCOPE

In this paper, Valley Based Vertical Partitioning Algorithm has been proposed and has been successfully implemented in order to improve the Query Response Time (QRT) in Distributed Database. The proposed algorithm VBVPA takes CAM as input and produces the fragmentation of a relation. CAM is calculated by using the values of AUM and FM.

The future scope of this proposal is that it may be possible that there can be a long interval between two valleys formed in a relation. Hence a new Valley based algorithm is needed to be devised in order to minimize the fragments having more number of attributes.

7. REFERENCES

[1] Abuelyaman, E. S., "An Optimized Scheme for Vertical Partitioning of a Distributed Database," in International Journal of Computer Science and Network Security (IJCSNS), Vol. 8, No. 1, January 2008, 310-316.

[2] Hoffer, J.A. and Severance, D.J. 1975.The use of cluster analysis in physical database design. In Proceedings of the 1st International Conference on Very Large Data Bases, New York, USA.

[3] Horowitz, E. and Sahni, S. 1978. Fundamentals of Computer Algorithms. Computer Science Press Rockville, Maryland.

[4] McCormick, W. T. Schweitzer P.J., and White T.W., "Problem Decomposition and Data Reorganization by A Clustering Technique," Operation Research, Vol. 20 No. 5, September 1972, 993-1009.

[5] Navathe, S., Ceri, S., Wierhold, G. and Dou, J., "Vertical Partitioning Algorithms for Database Design," ACM Transactions on Database Systems, Vol. 9 No. 4, December 1984, 680-710.

[6] Navathe, S. and Ra M., "Vertical Partitioning for Database Design: A Graph Algorithm," ACM Special Interest Group on Mamagement of Data (SIGMOD) International Conference on Management of Data, Vol.18 No. 2, June 1989, 440-450.

[7] Chu, W. W. and Ieong, I. "A Transaction-Based Approach to Vertical Partitioning for Relational Database Systems," IEEE Transactions on Software Engineering, Vol. 19 No. 8, August 1993, 408-412.

[8] Bradley, P. S., Fayyad, U. M. and Reina, C., "Scaling Clustering Algorithms to Large Databases", in proceedings of the 4th International Conference on Knowledge Discovery & Data Mining , June 1998, 9-15.

[9] Guha, S., Rastogi, R. and Shim, K., "CURE: an efficient clustering algorithm for large databases", in proceedings of the 1998 ACM SIGMOD international conference on Management of data, Vol. 27, Issue 2, June 1998, 73-84.

[10] Ng, R. T. and. Han, J., "Efficient and Effective Clustering Methods for Spatial Data Mining", Proceedings of the 20th International Conference on Very Large Data Bases, September 1994, 144-155.

[11] Jain, A. and Dubes, R., "Algorithms for Clustering Data", Prentice Hall, New Jersey, 1998.

[12] Kaufman, L., Rousseuw, P., "Finding Groups in Data- An Introduction to Cluster Analysis", Wiley Series in Probability and Math. Sciences, 1990.

[13] Zhang, T., Ramakrishnan, R. and Livny, M., "An Efficient Data Clustering Method for Very Large Databases", in proceedings of the SIGMOD international conference on Management of data, June 1996, 103-114.

[14] Ashish Ranjan Mishra, Neelendra Badal, "Slop based Partitioning for Vertical Fragmentation in Distributed Database System", in International Journal of Computer Applications (0975 – 8887) Volume 99– No.4, August 2014.