A Vertical Partitioning Algorithm for Distributed Object Oriented Databases

Nishant Gaurav
Manipal University, Manipal
Udupi, Karnataka

ABSTRACT

Object Oriented Databases (OODB) is becoming popular day by day and being used in a large number of application domains. In order to support homogeneous distributed OODBs a clear understanding of partitioning of class and how to do it by using different partitioning algorithms is needed. In this paper an algorithm for vertical fragmentation in a model consisting of class and comprising of complex attributes and complex methods is presented. The approach for fragmentation is top-down and entity of fragmentation is class. The algorithm presented here is an enhancement to the previous work of vertical partitioning algorithms in OODB management systems. The algorithm takes input as the class to be partitioned into fragments or groups, generates Method Usage Matrix as its first step from the methods and queries provided. It then generates Method Affinity matrix which is constructed using above matrix and based on method affinity values of two methods. Two new factors are introduced Method Linking Factor and Group Linking Factor which provides more control on deciding groups and increasing the flexibility of the algorithm.

General Terms

Object Oriented Databases, Class, Methods, and Attributes.

Keywords

Vertical Partitioning, Distributed Object Oriented Database Management Systems, Method Affinity, Method Usage, Complex Methods, Complex Attributes.

1. INTRODUCTION

Distributed Database systems (DDBS) technology is union of what appear to be diametrically opposed approaches to data processing: database system and computer networks technologies. DDBs are a collection of multiple logically related databases distributed over a computer network. A distributed database management systems is then defined as the software system that permits the management of the distributed databases and makes the distribution transparent to the users [1].

Two main strategies that have been identified for designing distributed databases are the top-down approach and bottom up approach [2]. The top-down design approach takes a global conceptual schema (GCS) describing the global databases entities and their relationships, and combines it with access pattern information to produce a set of local conceptual schemas (LCS) describing database entities at each local site. Top-down approach for designing OODB is more suitable for tightly integrated homogeneous distributed database management systems while bottom-up approach is more suited to multiple databases.

A fragmentation is a process which breaks a class into a set of smaller classes called fragments. A class can be fragmented vertically or horizontally depending upon the requirements and nature of schema. The vertical class fragment is defined as the non-empty proper subsets of the attributes; while the horizontal class fragments are non-empty proper subsets of objects. The issues involved in distributed design are presented in [3]. Two types of methods are identified: simple and complex methods. A method that does not call/invoke any other methods is called a simple method otherwise it is a complex method. O(M) is the set of objects accessed by the method M and A(M) is the set of attributes of these objects. These sets are further group into sets of objects and attributes based on classes to which they belong. This generates the set pairs of objects and attributes as (O_i, A_j) accessed from a class C_i by a method.

In this paper, top-down approach for fragmentation is followed and OODBs has been designed by distributing the entities over various sites. The presented work aims at dividing a class into fragments or groups which are later distributed over sites for better performance. The paper presents an algorithm for vertically partitioning a class consisting of complex attributes and complex methods. Rest of the paper is as follows: Section 2 describes the work that has been in distributed OODBs and Relational Database Management Systems (RDBMS). Section 3 provides the algorithm for Vertical Partitioning along with its objectives and correctness criteria. Section 4 provides an example of the algorithm. A class has been partitioned using the algorithm and different steps of the algorithm are performed to provide a better illustration. Section 5 gives the conclusion and future work.

2. RELATED WORK

Most of the work has been concentrated on partitioning of Relational Database Systems (RDBS). A little work has been done on partitioning OODBs.

2.1 Relational Databases

The problem of vertical partitioning is to determine a relation into fragments/partitions in order to increase the performance of the database systems. Selecting an optimal partition is a difficult problem: a relation with m attributes can be parted in B(m) different ways, where B(m) is the Bell number (for large m, B(m) approaches m^m). Thus, heuristic approaches are necessary to determine the near optimal partition.

Hoffer and Severance [4] measures the affinity between pair of attributes and try to cluster the attributes according to pair wise affinity by using the Bond Energy Algorithm developed in [5]. Navathe extended the results of Hoffer and Severance

and proposed a heuristic approach for vertical partitioning. They use the given input parameters in the form of an attribute usage matrix and transactions, to construct the attribute affinity matrix on which clustering is performed [6]. After clustering, iterative binary partitioning is attempted, first with an empirical objective function. The process is continued until no further partitioning results. During the second phase, the fragments can be further refined by incorporating estimated cost factors weighted on the basis of the type of problem being solved.

Navathe and Ra have developed a new algorithm based on a graphical technique [7]. 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. The algorithm generates all meaningful fragments in one iteration by considering a cycle as a fragment. "Affinity cycles" are formed in the tree by including the edges of the 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 cycle apart along cut-edges.

2.2 Object Oriented Databases

Three major partitioning schemes for object oriented databases, namely vertical class partitioning, path partitioning and horizontal class partitioning along with their completeness and reconstruction properties have been presented [8]. Their approach in developing this partitioning scheme is to assure that all the resultant class fragments can be represented and implemented as classes in an OODBMS. This resultant class fragments can be either locally accessed or globally accessed.

Bellatreche and Simonet proposed an algorithm which is adaptation of the graphical technique presented by Navathe and Ra for the relational model [9]. The algorithm is an extension of the graphical algorithm suggested by [7] to a model consisting of complex attributes and complex methods, and the domain of an attribute being an arbitrary class the definition of a class results in an directed graph (S, E) where S represents the graph and E represents the edges which corresponds to the relationship between two classes. These edges are nothing but method affinity of two methods m_i and m_j which is calculated as the total number of accesses of the queries referencing both methods m_i and $m_{j..}$ These method affinities of all the combinations of m_i and m_j are populated in Method Affinity Matrix (MAM) similar as attribute affinity matrix in previous algorithms. Method affinity Matrix is in turn dependent on Method Usage Matrix. It is populated using method usage values which are is the count a method is accessed by a particular query in a class. The algorithm that is presented here is an extension of this algorithm. MUM and MAM is constructed using method usage values and method affinity values respectively. The factors introduced in the algorithm provide more flexibility and control in constructing the groups or the fragments. Also, the attributes are not openly accessed, they are accessible through methods. Basically, all attributes are encapsulated under methods and through these methods only one can access attributes.

3. ALGORITHM

3.1 Objective

The objective of vertical fragmentation (VF) is to break a class model into a set of smaller classes (fragments) that permit user applications to be executed using only one fragment. This means that optimal vertical fragmentation will minimize the execution time of user applications. VF aims at splitting a class so that all the attributes and methods most frequently accessed together by user queries are grouped together.

3.2 Enhancement Factors

The two factors we added in the enhanced version of our algorithm are [10]:

• *Method Linking Factor (MLF):* This factor is added to avoid having poor grouping between two (or more) methods. The factor is used in the formula: $aff(i,j) \geq P(m_i) * ALF/100$ which should be true for linking two methods.

• *Groups Linking Factor (GLF)*: This factor is added to avoid having poor grouping between two groups. Here we have two scenarios: First: If we want to connect method m_i in group k to an independent method m_j, then the condition $aff(i,j) \geq P(g_k) * GLF/100$ must be true. Second: If we want to connect a method m_i in group k to method m_j in group l, then the condition $P(g_i) \geq P(g_k) * GLF/100$ must be satisfied.

3.3 Algorithm: Vertical Fragmentation

Input of the Algorithm: The class to be fragmented, Access Frequency, MLF, GLF.

Step 1: Construct the Method Usage Matrix (MUM) of the owner class C. Given a set of queries $Q = \{q_1, q_2, \ldots q_l\}$ that will run on the class C (A,M), the following parameter is defined for each query $q_j (1 \leq j \leq l)$:

In MUM, the value will be 1 if a method say m_i is referenced by query q_j else it is 0.

This generates the method usage matrix whose rows are queries and columns are methods. Access frequency of a query is the number of access of the query to the instances of objects per unit tine period.

Step 2: Construct the Method Affinity Matrix n × n (n methods) whose (i,j) element equals the two methods m_i and m_j ($1 \leq i,j \leq n$). The affinity is the total number of accesses of the queries referencing both methods m_i and m_j in class C plus the total number of accesses of both methods m_i and m_j of class C. The affinity matrix is symmetric. The rows and columns of the matrix are both methods.

Step 3: Iterate starting from the first method of MAM (from first row) trying to generate initial groups by joining it to other method(s) with the highest affinity value (Max (aff(i,j))), forming the initial groups and $i \neq j$ (run i for rows and j for columns). The resulted group will have a power factor P(g) that takes the affinity value aff(i,j). Here we will have three possible scenarios:

First: The two methods are independent (do not belong to any group), in this case perform a direct grouping is performed if the condition $aff(i,j) \geq P(m_i) * ALF/100$ is satisfied where aff(i,j) is affinity of methods i and j in the method affinity matrix.

Second: One of the methods i or j belongs to a group k and the other method is independent, in this case join the independent method to group k if the condition aff(i,j) ≥ $P(g_k)$ is true. *Third:* Having method m_i in group k and method m_j in group l, then we will join the two groups if $P(g_k)=P(g_l)$. By the end of this step, we will have all possible groups.

Step 5: Trying to search for the best extension, two possible scenarios exist.

First: The best extension connects method m_i in group k and method m_j that has not been joined to any initial group in previous step, in this case the independent method m_j will be joined to group k if the condition aff(i,j) ≥ $P(g_k)$ * GLF/100 is true, then the extended group's power will be equal to aff(i,j) value. Calculate the MinMerge value = $P(g_k)$ – aff(i,j).

Second: The best extension connects method m_i in group k and m_j in group l in this case, ensure that the two conditions aff(i,j) ≥ $P(g_k)$ * GLF/100 and $P(g_l)$ ≥ $P(g_k)$ * GLF/100 are true. The new group's power will be equal to the power of group l. Calculate the MinMerge value = $P(g_k)$ – $P(g_l)$. Keep repeating this last step until there is no possible best extension found and then obtain final groupings of our algorithm. MinMerge should be the minimum value. It will decide the best extension. Methods corresponding to MinMerge value should be joined or the group having MinMerge value should be joined.

3.4 Correctness of the Vertical Fragmentation Algorithm

For an algorithm to be correct it must satisfy the correctness rules - completeness, reconstruction and disjointness.

- *Completeness* : A class *C(A,M)* is fragmented into a set of class fragments F_1, F_2, ..., F_k which is complete if and only if each attribute or method in C(A, *M)* can also be found in some F_i ($1 \leq i \leq k$).
- Reconstruction : The join of all class-fragments should reproduce the original class.
- Disjointness: Each attribute or method in F_i ($1 \leq i \leq k$) should not be in any other fragment F_j ($j \neq i$).

4. EXAMPLE

In figure given below, Schema of a class Dept. and shown how the Faculty class is fragmented according to the set of queries.

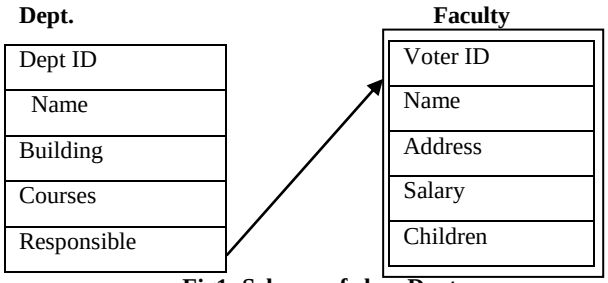

Fig1: Schema of class Dept

The schema of this Object Oriented Databases is:

Class: Dept

Attributes

ID : Integer,
Name : String,
Building : String,
courses : String,
Responsible : String

Method

Modify courses (course) :String

Class: Faculty

Attributes

Voter ID : Integer,
Name : String,
Address : String,
Salary : Integer,
Children : Integer

Methods

Age (Voter ID) : Integer,
Income Tax (Salary and Children) : Real,
Modify courses (course) :String

Faculty class is part of Dept. class.
Queries:

Queries running on Dept class are:
QDept 1: Find the courses of a Dept. given its ID.
QDept 2: Find the name and courses of all Dept.
QDept 3: Find the names of Dept located at given building.
QDept 4:Find the names and salary of HOD of Dept, given its Faculty ID.

Queries running on Faculty class are:
QFact 1: Give the voter ID of all Faculty with age < 35.
QFact 2: Give the voter ID, names and address of all Faculties with salary < 20000.
QFact 3: List the voter ID and children of all the Faculty with Income tax < 15000.
QFact 4:Increase by 10% the salary of all the faculties with children < 3 and Age <45.

QDept 4 which runs on the Dept class uses the name and salary attributes of the Faculty class. Considering this as one more query running in the Faculty class, rename it Q5.

Before the fragmentation of Faculty class, construct the Method usage Matrix of class *Dept*. There are five methods of the Dept class to access attributes.
m_1 = r_DeptID ; m_2 = r_Name ; m_3 = r_Building ; m_4 = r_courses ; m_5 = r_Responsible ; m_6 = r_ModifyCourses.

Method Usage Matrix of this class is shown. Access column is added to show the access number to a method for a specified period for each query.

	M_1	M_2	M_3	M_4	M_5	M_6	Acc
Q1	1	0	0	1	0	0	40
Q2	0	1	0	1	0	0	15
Q3	0	1	1	0	0	0	25
Q4	1	0	0	0	1	0	15

Fig. 2: Method Usage Matrix of class Dept

Now, partition the class *faculty* which contains five methods to access attributes:
m_1 = r_VoterID ; m_2 = r_Name ; m_3 = r_Address ; m_4 = r_Salary ; m_5 = r_Children.
The other methods are:
m_6 = Age ; m_7 = Income Tax ; m_8 = Modify Salary

The Method Usage Matrix of this class, shown in Fig.3. It is constructed by analyzing the queries on this class and the queries running and queries running on their owner class.

	M_1	M_2	M_3	M_4	M_5	M_6	M_7	M_8	acc
Q1	1	0	0	0	0	1	0	0	30
Q2	1	1	1	1	0	0	0	0	25
Q3	1	0	0	0	1	0	1	0	35
Q4	0	0	0	0	1	1	0	1	15
Q5	0	1	0	1	0	0	0	0	15

Fig. 3: Method Usage Matrix of class Faculty

Now, construct the Method Affinity Matrix (MAM) for faculty class using the algorithm. It is a symmetric square matrix which is populated by the affinity of m_i and m_j based on accessibility on queries. The MAM is presented in fig. 4.

	M_1	M_2	M_3	M_4	M_5	M_6	M_7	M_8
M_1	90	25	25	25	35	30	35	0
M_2	25	40	25	40	0	0	0	0
M_3	25	25	25	25	0	0	0	0
M_4	25	40	25	40	0	0	0	0
M_5	35	0	0	0	50	15	35	15
M_6	30	0	0	0	15	45	0	15
M_7	35	0	0	0	35	0	35	0
M_8	0	0	0	0	15	15	0	15

Fig. 4: Method Affinity Matrix of class Faculty

Taking MLF = 55% and GLF = 60% proceed to step 3 of our algorithm. Running i for rows and j for columns.

1. Start from the first row of MAM (i = 1) and searching for the Max($Aff_{i,j}$) =35 for j = 5 ($i \neq j$). Checking MLF cond. aff(i,j) $\geq$ $P(m_i)$ * ALF/100 i.e. 35 $\geq$ 35 * 0.55,which is true. So, first initial group is formed with 1 and 5 as its elements. The power of the group is 35. j = 7 has also the same affinity as j = 5.So it will also be included in the same group as shown in fig. 6(a).

2. Moving to the next row i = 2, ($i \neq j$) the Max($Aff_{i,j}$) = 40 for j = 4. Checking MLF cond. aff(i,j) $\geq$ $P(m_i)$ * ALF/100 i.e. 40 $\geq$ 40 * 0.55,which is true. So, second initial group is formed with 2 and 4 as its elements. The power of the group is 40 shown in fig. 6(b).

3. Moving to i =3, Max($Aff_{i,j}$) = 25 for j = 1. But 1 belongs to first initial group and since the power of the initial group (35) is greater than $Aff_{i,j}$ i.e. affinity of 1 and 3 which is 25. So, 3 is not included in the group. Same reason goes for next max. affinity for j = 2 and j = 4.

4. For i = 4, Max($Aff_{i,j}$) = 40 for j = 2. But 4 and 2 is already placed in second group. So, skipping it and moving ahead getting Max($Aff_{i,j}$) = 25 for j = 1. Skip j = 1 since power of the group holding 1 is greater than the current max. affinity. All the other affinity are also not included due to the same above reason.

5. For i = 5, Max ($Aff_{i,j}$) = 35 for j = 7 and 1.But 5 is already included in the same group so skipped. Moving ahead, j = 6 Max ($Aff_{i,j}$) = 15,same value as j = 8. But both are not included to the group since the current max. affinity is less than the power of the group holding 5.

6. For i = 6, j = 1 Max ($Aff_{i,j}$) = 30 but not included, same reason as above. Same goes for next max. affinity j = 5. For j = 8 Max ($Aff_{i,j}$) = 15 and 8 is not included in any group so both are independent methods therefore checking MLF condition. 15 $\geq$ 15 * 0.55, which is true. So a new group third initial group is formed containing 6 and 8. Power of this group is 15 shown in fig. 6(c).

7. For i = 7 Max ($Aff_{i,j}$) = 35 for j = 5 but they are in same group, so skipped.

8. For i = 8 Max ($Aff_{i,j}$) = 15 for j = 5 and 6. Method 5 is excluded since the power of the group holding 5 is greater than the max. affinity. Method 6 is also not included because it belongs to the same group.

9. Now all the initial groups have been formed. Three initial groups are there with their respective powers. Method 3 is an independent method.

10. Searching for the best extension. Iterating from the first row i = 1 of MAM and searching for Max ($Aff_{i,j}$) for method j where $i \neq j$ and i and j are not in the same group. Now for i = 1 , j = 5 and 7 are skipped (same group), for j = 6 Max ($Aff_{i,j}$) = 30. Here, we are joining method 1 of first initial group and method 6 of third initial group i.e. we are trying to join two groups, so checking for GLF condition, $P(g_l)$ $\geq$ $P(g_k)$ * GLF/100 (15 $\geq$ 35 * .6) which is false, so these two groups cannot be joined. For j = 3, Max ($Aff_{i,j}$) = 25. Since 3 is an independent method so we have to check GLF condition for independent method for linking it with other group i.e. aff(i,j) $\geq$ $P(g_k)$ * GLF/100 (25 $\geq$ 35 * 0.6) is true.
MinMerge = $P(g_k)$ – aff(i,j) = 35 – 25 =10. Hence, Method 3 can be linked to first initial group.

11. For i = 2, j = 4 cannot be included (same group) choosing j = 3, Max ($Aff_{i,j}$) = 25. Checking the GLF condition for linking independent method with another group, aff(i,j) $\geq$ $P(g_k)$ * GLF/100 (25 $\geq$ 35 * 0.6) which is true. MinMerge = $P(g_k)$ – aff(i,j) = 40 - 25 =15 but current MinMerge is not less than the previous MinMerge which is 10. So cannot be linked.

12. For i = 3 same result occurs as in 11 and 12. So no change occurs.

13. Iterating for the remaining methods, we found the same earlier results occurs for different methods.

14. Searching for the best extension we could find only one linking, i.e. method 3 to first initial group. So, placing in that group also the power of the group will be 25 shown in fig. 7(a).

15. Continuing the search for next best extension we couldn't find any so the algorithm stops.

Final groups as shown in fig. 7:

G_1 = {r_voterID, r_Address, r_IncomeTax, r_Salary}
G_2 = {r_name, r_Salary}
G_3 = {r_Age, r_ModifySalary}

Because we are interested in the attributes used by the methods, we are going to construct an Attribute Usage Matrix (AUM) shown in Fig 5.
In AUM, the value will be 1 if attribute A_j is referenced by m_i else it is 0.

A_1 = VoterID, A_2 = Name, A_3 = Address, A_4 = Salary, A_5 = Children

	A_1	A_2	A_3	A_4	A_5
M_1	1	0	0	0	0
M_2	0	1	0	0	0
M_3	0	0	1	0	0
M_4	0	0	0	1	0
M_5	0	0	0	0	1
M_6	1	0	0	0	0
M_7	0	0	0	1	1
M_8	0	0	0	1	0

Fig. 5: Attribute Usage Matrix of class Faculty

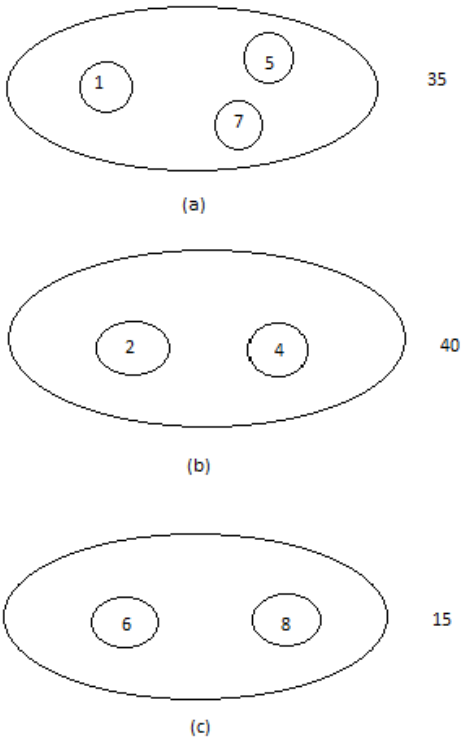

Fig. 6: Initial groups with their powers

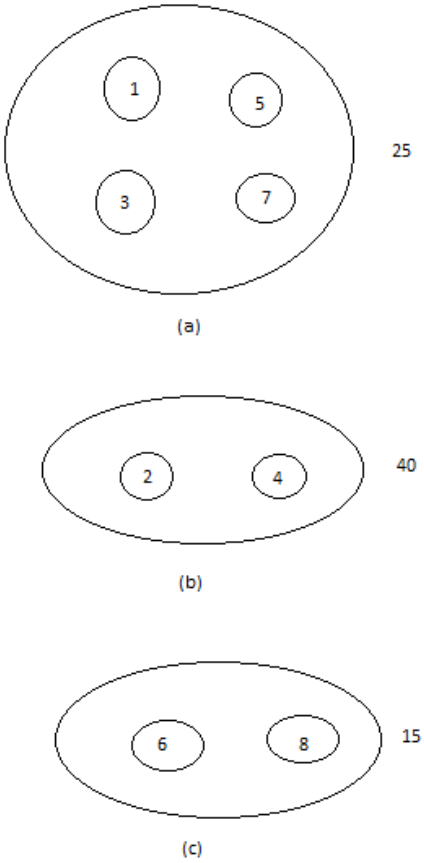

Fig. 7: Final groups with their powers

5. CONCLUSION

Most of the algorithms that have been worked are in the area of Relational Database Management Systems (RDBMS) or Object Oriented Database Systems (OODBS) concentrated on a single site. Little work has been done on homogeneous distributed databases. In this paper, an algorithm which is far more flexible than previous algorithm (Graphical, BEA etc.) and easy to understand has been proposed. The class fragmentation is based on the object model with complex attributes and complex models. All the three conditions of the correctness of the algorithm are satisfied.

The algorithm is more efficient because the added factors (MLF) and (GLF) provide an enhanced grouping of methods based on problem specification. The contributed factors in the algorithm provide more control on the methods and generate no unnecessary calculation. The level of performance is better and the time taken for whole computation is also less.

Also, the values for the enhancement factors are chosen based on several qualitative and quantitative issues such as network bandwidth, number of sites, number of methods and attributes, query/transaction frequencies. In future, extensive work will be on horizontal and hybrid partitioning algorithms, to obtain more solution for fragment allocation problem.

6. REFERENCES

[1] M. T. Ozsu and P. Valduriez, Principles of Distributed Database Systems, Prentice Hall, 1991.

[2] S. Ceri, M. Negri, and G. Pelagatti, Horizontal Data Partitioning in Database Design, In Proceedings of the ACM SIGMOD International Conference on Management of Data. SIGPLAN Notices, 1982.

[3] K. Karlapalem, S. B. Navathe, and M. M. A. Morsi, Issues in Distributed Databases of Object-Oriented Database Systems, in Distributed Object Management Edited by M. T. Ozsu, U. Dayal, P. Valduriez, Morgan Kauffmann Publishers Inc., 1994.

[4] J. A. Hoffer, and D. G. Severance, The Use of Cluster Analysis in Physical Database Design, In Proceedings of the 1st International Conference on Very Large Databases. Morgan Kaufmann Publishers, 1975.

[5] W. T. McCormick, P. J. Schweitzer, and T. W. White, Problem Decomposition and Data Reorganization by a Clustering Technique, Operation Research, Vol. 20, No 5, September 1972.

[6] S. B. Navathe, S. Ceri, G. Winderhold and J .Dou Vertical Partitioning Algorithms for Databases Design ACM transactions on Database Systems, Vol. 9, No. 4, 1984.

[7] S. B. Navathe, M. Ra and J. Dou, Vertical Partitioning Algorithms for Databases Design ACM transactions on Database Systems, Vol. 9, No. 4, 1989.

[8] K. Karlapalem, and Q. Li, Partitioning Schemes for Object Oriented Databases, Proc. of the Fifth International Workshop on Research Issues in Data Engineering-Distributed Object Management, RIDE-DOM'95,1995.

[9] L. Bellatreche, A. Simonet and M.Simonet, Vertical Fragmentation in Distributed Object Database with Complex Attributes and Complex Methods, in International Workshop on Database and Expert Systems Applications (DEXA'96), September, 1996.

[10] F. Marir, Y. Najjar, M. A. AlFaress and H. I. Abdalla, An Enhanced Grouping Algorithm for Vertical Partitioining Problem in DDBs, IEEE Conference, 2007.