Horizontal Fragmentation Technique in Distributed Database

Ms. P. R. Bhuyar
M.E. I st Year (CSE)
Sipna College of Engineering & Technology, Amravati, India

Dr.A.D.Gawande
HOD (CMPS)
Department of computer science & Engineering
Sipna College of Engineering & Technology, Amravati, India

Prof. A.B.Deshmukh
Professor (CMPS)
Department of computer science & Engineering
Sipna College of Engineering & Technology, Amravati, India

Abstract- Distributed database technology is expected to have a significant impact on data processing in the upcoming years. Today's business environment has an increasing need for distributed database and Client/server applications as the desire for consistent, scalable, reliable and accessible information is steadily growing. Distributed processing is an effective way to improve reliability and performance of a database system. Distribution of data is a collection of fragmentation, allocation and replication processes. Previous research works provided fragmentation solution based on empirical data about the type and frequency of the queries submitted to a centralized system. These solutions are not suitable at the initial stage of a database design for a distributed system. The purpose of this work is to present an introduction to Distributed Databases which are becoming very popular now days with the description of distributed database environment, fragmentation and horizontal fragmentation technique. Horizontal fragmentation has an important impact in improving the applications performance that is strongly affected by distributed databases design phase. In this report, we have presented a fragmentation technique that can be applied at the initial stage as well as in later stages of a distributed database system for partitioning the relations. Allocation of fragments is done simultaneously in the algorithm. Result shows that proposed technique can solve initial fragmentation problem of relational databases for distributed systems properly.

Index Terms- Distributed database, Fragmentation, Horizontal Fragmentation, Allocation.

I. INTRODUCTION

1.1 Distributed Database System

A distributed database (DDB) is a collection of data that logically belongs to the same system but is spread over the sites of a computer network. It is not necessary that database system have to be geographically distributed. The sites of the distributed database can have the same network address and may be in the same room but the communication between them is done over a network instead of shared memory. As communication technology, hardware, software protocols advances rapidly and prices of network equipments falls every day, developing distributed database systems become more and more feasible. Design of efficient distributed database is one of the major research problems in database & information technology areas.

A distributed database management system (DDBMS) is then defined as the software system that permits the management of the DDB and makes the distribution transparent to the users. Distributed database system (DDBS) is the integration of DDB and DDBMS. This integration is achieved through the merging the database and networking technologies together. Or it can be described as, a system that runs on a collection of machines that do not have shared memory, yet looks to the user like a single machine. Assumptions regarding the system that underlie these definitions are:

1. Data is stored at a number of sites. Each site is assumed to *logically* consist of a single processor. Even if some sites are multiprocessor machines, the distributed DBMS is not concerned with the storage and management of data on this parallel machine.

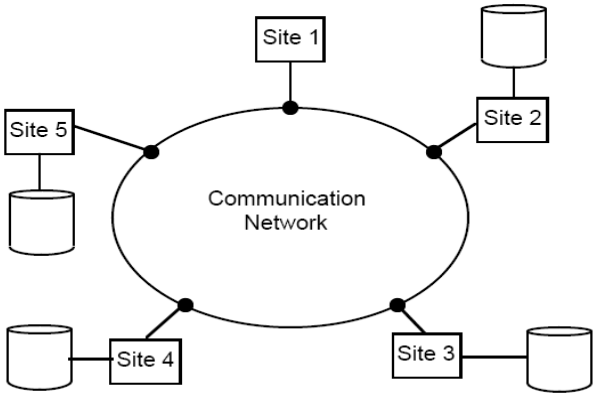

Figure 1: A Distributed Database Environment

2. The processors at these sites are interconnected by a computer network rather than a multiprocessor configuration.
3. To form a DDB, distributed data should be logically related, where the relationship is defined according to some structural formalism, and access to data should be at a high level via a common interface. The typical formalism that is used for establishing the logical relationship is the relational model.
4. The system has the full functionality of a DBMS.

Distributed 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. Primary concern of distributed database system design is to making fragmentation of the relations in case of relational database or classes in case of object oriented databases, allocation and replication of the fragments in different sites of the distributed system, and local optimization in each site.

1.2 Fragmentation

Primary concern of distributed database system design is to making fragmentation of the relations in case of relational database or classes in case of object oriented databases, allocation and replication of the fragments in different sites of the distributed system, and local optimization in each site. Fragmentation is a design technique to divide a single relation or class of a database into two or more partitions such that the combination of the partitions provides the original database without any loss of information. This reduces the amount of irrelevant data accessed by the applications of the database, thus reducing the number of disk accesses. Fragmentation can be horizontal, vertical or mixed/hybrid.

1.2.1. Horizontal fragmentation

Horizontal fragmentation (HF) allows a relation or class to be partitioned into disjoint tuples or instances. Intuition behind horizontal fragmentation is that Every site should hold all information that is used to query at the site and the information at the site should be fragmented so the queries of the site run faster.

Horizontal fragmentation is defined as selection operation, σ _p(R).

For example, the following relation

EMPLOYEE (eid, fname, lname, site ,pos, salary)

Eid	Fname	Lname	Site	Pos	Salary
Fragment1					
Fragment2					
Fragment3					

Figure 2: Horizontal fragmentation

1.2.2. Vertical Fragmentation

Vertical fragmentation (VF) allows a relation or class to be partitioned into disjoint sets of columns or attributes except the primary key. Each partition must include the primary key attribute(s) of the table. This arrangement can make sense when different sites are responsible for processing different functions involving an entity.

Objective of vertical fragmentation is to partition a relation into a set of smaller relations so that many of the applications will run on only one fragment.

a. Vertical fragmentation of a relation R produces fragments R1,R2, . . . , each of which contains a subset of R's attributes.
b. Vertical fragmentation is defined using the projection operation of the relational algebra:

Π _A1,A2,...,An(R)

Eid	Fname	Lname	Site	Eid	Pos	Salary
Fragment1				Fragment2		

Figure 3: Vertical fragmentation

1.2.3. Hybrid fragmentation

Combination of horizontal and vertical fragmentations is mixed or hybrid fragmentations (MF). In this type of fragmentation scheme, the table is divided into arbitrary blocks, based on the needed requirements. Each fragmentation can be allocated on to a specific site. This type of fragmentation is the most complex one, which needs more management, in most cases simple horizontal or vertical fragmentation of DB applications.

Mixed fragmentation (hybrid fragmentation) Consists of a horizontal fragment followed schema will not be sufficient to satisfy the requirements of the by a vertical fragmentation, or a vertical fragmentation followed by a horizontal fragmentation. Mixed Fragmentation is defined using the selection and projection operations of relational algebra:

Π_p(_A1,...,An(R))

Π _A1,...,An(_p(R))

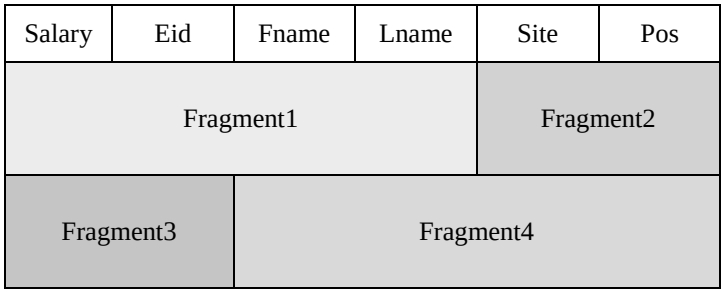

Figure 4: Hybrid fragmentation

The main reasons of fragmentation of the relations are to: increase locality of reference of the queries submitted to database, improve reliability and availability of data and performance of the system, balance storage capacities and minimize communication costs among sites.
Previous techniques of HF, VF or MF have the following problems in common:

- They use frequency of queries, midterm predicates' affinity or attribute affinity matrix (AAM) as a basis of fragmentation. These require sufficient empirical data that are not available in most cases at the initial stage.
- Most of them concentrate only fragmentation problem and overlooked allocation problem to reduce complexity

Allocation is the process of assigning the fragments of a database on the sites of a distributed network. When data are allocated, it may either be replicated or maintained as a single copy. The replication of fragments improves reliability and efficiency of read-only queries but increase update cost

In this report, we have presented a new technique for horizontal fragmentation of the relations of a distributed database. This technique is capable of taking proper fragmentation decision at the initial stage by using the knowledge gathered during requirement analysis phase without the help of empirical data about query execution. It can also allocate the fragments properly among the sites of DDBMS.

II. Literature Review

Distributed databases are not new, nor are they a consideration unique to client/server architectures or relational databases. Data distribution needs, no doubt, arose immediately after the first database management systems appeared 30 years ago, and various solutions to the distribution problem have been implemented over the years on mainframe and minicomputer platforms using a wide variety of database management software.

HF using min-term predicate is first proposed by Ceri et al.[5]. Ozsu and Valduriez proposed an iterative algorithm COMMIN to generate a complete and minimal set of predicates from a given set of simple predicates [1]. Navathe et al. proposed a MF technique. The input of the procedure comprises a predicate affinity table and an attribute affinity table [3]. Bai˜oo et al. inputted predicate affinity matrix to build a predicate affinity graph thus defines horizontal class fragments [4]. Navathe et al. used attribute usage matrix (AUM) and Bond energy algorithm to produce vertical fragments [6]. Shin and Irani proposed knowledge based approach in which user reference clusters are derived from the user queries to the database and the knowledge about the data [7]. Ra presented a graph based algorithm for HF in which predicates are clustered based on the predicate affinities [8]. Cheng et al. presented a genetic algorithm based fragmentation approach that treats horizontal fragmentation as a travelling salesman problem [9]. Ma et al. Used an attribute uses frequency matrix (AUFM) and a cost model for VF [10]. Alfares et al. used AAM to generate groups based on affinity values [11]. Marwa et al. uses the instance request matrix to horizontally fragment object oriented database [12]. Abuelyaman proposed a static algorithm StatPart for VF [13]. Mahboubi H. and Darmont J. used predicate affinity for HF in data warehouse [14].

To the best of our knowledge, only Abuelyaman [13] provided a solution for initial fragmentation of relations of a distribution database. A randomly generated reflexivity matrix, a symmetry matrix and a transitivity module has been used to produce vertical fragments of the relations and no algorithm for horizontal fragmentation. But he could not justify his hypothesis that why it will produce good fragments.

III. Related Work

To solve the problem of taking proper fragmentation decision at the initial stage of a distributed database, we have provided a new technique of fragmentation. That is to fragment a relation horizontally according to locality of precedence of its attributes. Attribute locality precedence (ALP) can be defined as the value of importance of an attribute with respect to sites of distributed database. ALP table will be constructed by database designer for each relation of a DDBMS at the time of designing the database with the help of modified CRUD (Create, Read, Update, and Delete) matrix and cost functions. A block diagram of our system is depicted in Figure 5.

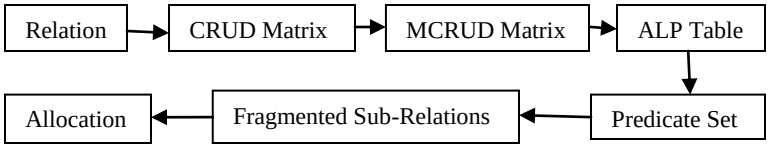

Figure 5: Block diagram of the system

A relation in a database contains different types of attributes those describe properties of the relation. But the important thing is that the attributes of a relation do not have same importance with respect to data distribution in different sites. According to above importance we can calculate locality precedence of each attribute for each relation and construct ALP table for the relations.

3.1 CRUD Matrix

A CRUD (data-to-location) matrix is a table of which rows indicate attributes of the entities of a relation and columns indicate different locations of the applications (processes that affect those attributes). If a particular process uses a particular

entity attribute, the appropriate cell is filled in with the letters C, R, U, or D.A "C" in the cell of a CRUD matrix indicates that the process sometimes creates new instances of the corresponding entity type. An "R" in the cell indicates that the process sometimes reads existing instances of the entity type. A "U" in the cell indicates that the process sometimes updates instances of the corresponding entity type. A "D" in the cell indicates that the process sometimes deletes instances of the corresponding entity type.

A process does not necessarily use an entity every time it occurs. This does not mean that the interaction should not be shown on the CRUD matrix. If the process ever uses the entity, the interaction must be documented in the CRUD matrix. A CRUD matrix is used by the system analysts and designers in the requirement analysis phase of system development life cycle for making decision of data mapping to different locations.

MCRUD Matrix - We have modified the existing CRUD matrix according to our requirement of HF and name it Modified Create, Read, Update, and Delete (MCRUD) matrix. It is a table constructed by placing predicates of attributes of a relation as the rows and applications of the sites of a DDBMS as the columns. We have used MCRUD to generate ALP table for each relation.

We treated cost as the effort of access and modification of a particular attribute of a relation by an application from a particular site. For calculating precedence of an attribute of a relation we take the MCRUD matrix of the relation as an input and use the following cost functions:

$$C_{i,j,k,r} = f_C C + f_R R + f_U U + f_D D \quad (1)$$

$$S_{i,j,k} = \sum_{r=1}^{A_{i,j,k}} C_{i,j,k,r} \quad (2)$$

$$S_{i,j,m} = \max_{A_{i,j,k}}(S_{i,j,k}) \quad (3)$$

$$ALP_{i,j} = S_{i,j,m} - \sum_{k \neq m} S_{i,j,k} \quad (4)$$

$$ALP_i = \sum_{j=1}^{l} ALP_{i,j} \quad (5)$$

where f_C = frequency of create operation
f_R = frequency of read operation
f_U= frequency of update operation
f_D = frequency of delete operation
C = weight of create operation
R = weight of read operation
U = weight of update operation
D = weight of delete operation
$C_{i,j,k,r}$= cost of predicate j of attribute i accessed by application r at site k
$S_{i,j,k}$ = sum of all applications' cost of predicate j of attribute i at site k
$S_{i,j,m}$ = maximum cost among the sites for predicate j of attribute i
ALP_{ij} = actual cost for predicate j of attribute i
ALP_i = total cost of attribute i (locality precedence)

For simplicity we have assumed that f_C, f_R, f_U and f_D=1 and C=2, R=1, U=3 and D=2. The justification of the assumption is that at the design time of a distributed database, the designer will not know the actual frequencies of read, delete, create and update of a particular attribute from different applications of a site and generally update incurs more cost than create and delete, and reading from database always incurs least cost.

After construction of ALP table for a relation, predicate set P will be generated for the attribute with highest precedence value in the ALP table. Finally each relation will be fragmented horizontally using the predicates of P as selection predicate. The procedures can be clearly understood from the following algorithm and pseudo code of Fig 6 and 7.

Input: Total number of sites: S = {S_1, S_2,… ,Sn}
Relation to be fragmented: R
Modified CRUD matrix: MCRUD[R]
Output: Fragments F = {F_1, F_2, F_3,…, Fn}
Step 1: Construct ALP[R] from MCRUD[R] based on Cost functions
Step 2: For the highest valued attribute of ALP table
a. Generate predicate set P={ P1, P2, … ,Pm }
b. Rearrange P so that #P = #S
c. Fragment R using P as selection predicate

$$\forall_{p} l\ \sigma_p$$

d. Allocate F to S

Figure 6: Fragmentation Allocation algorithm

```
Input: MCRUD of a relation that to be fragmented
Output: ALP table for that relation

for ( i =1; i <= TotalAttributes; i++)
{
    for ( j =1; j <= TotalPredicates[i]; j++)
    {
      MAX[i][j] = 0;
      for ( k =1; k <= TotalSites; k++)
      {
         for ( r =1; r <= TotalApplications[k]; r++) /* Calculating
sum of   all applications" cost of predicate j of attribute i at site k */
         {
           C[i][j][k][r] = fc*C + fr*R + fu*U + fd*D
           S[i][j][k] + = C[i][j][k][r]
         }
         If S[i][j][k] > MAX[i][j] /*Find out at which site cost of
                                    predicate j is maximum*/
        {
           MAX[i][j] = S[i][j][k]
           POS[i][j] = k
         }
         SumOther = 0
         for ( r =1; r <= A[i][j][k][k]; r++)
         {
            If (r!=k)
            SumOther + = S[i][j][r]
          }
     }
     ALPsingle[i][j] = S[i][j][POS[i][j]] – SumOther /* actual
                                    cost for predicate j of attribute i */
  }
```

```
    ALP[i] = 0
    for ( j =1; j <= TotalPredicates[i]; j++) /*calculating total cost
                                for attribute i (locality precedence)*/
    {
       ALP[i] + = ALPsingle[i][j]
    }
}
```

Figure 7: ALP-table-construction Pseudo-code

To justify our technique, we have implemented a distributed banking database system. One of the relations of the database is Accounts shown in Table 1. Initially number of sites of the distributed system is three.

Table 1: Accounts Relation

AccNo	Type	CustId	OpenDate	Balance	Branch
01	Saving	101	05/01/12	30000	Pune
02	Current	102	18/01/12	48000	Pune
03	Current	103	10/02/12	15900	Nagpur
04	Saving	104	06/03/12	37750	Mumbai
05	Current	105	12/03/12	50000	Pune
06	Saving	106	25/03/12	25000	Nagpur
07	Current	107	28/03/12	45000	Mumbai

3.2. Construction of MCRUD Matrix

We have constructed the MCRUD matrix for the Accounts relation in the requirement analysis phase. Part of MCRUD matrix is shown in Figure 8.

Site / Application / Entity,Attribute.Predicate	Site1			Site2			Site3		
	Ap 1	Ap 2	Ap 3	Ap 1	Ap 2	Ap 3	Ap 1	Ap 2	Ap 3
Accounts.AccNo<10000	C		RU						R
Accounts.AccNo>=10000		R							
Accounts.Type=Saving	CRD	RU	RUD		R				
Accounts.Type=Current		RU	R				CRUD	RU	R
. . .									
Accounts.Balance<50000	R		R			CRUD			R
Accounts.Balance>=50000		CR							
Account.Branch=Pune	CRUD	RU	CRUD			R	R		
Account.Branch=Nagpur		R		CRUD	CRUD	R		R	
Account.Branch=Mumbai							CRUD	RD	CRU

Figure 8: MCRUD matrix of Accounts

3.3 Calculation of ALP

We have calculated locality precedence of each attribute from the MCRUD matrix of Accounts relation according to the cost functions of equation (1)-(5). Calculating the locality precedence of the attribute Branch is shown in Figure 9.

According to the cost functions, value of the predicate Branch=Pune is (8+4+8) - (1+1) = 18, Branch=Nagpur is (8+8+1) – (1+1) = 15 and Branch=Mumbai is (8+3+6) – 0 = 17. So ALP of Branch = 18+15+17 = 50.

Site / Application / Entity,Attribute.Predicate	Site1			Site2			Site3		
	Ap 1	Ap 2	Ap 3	Ap 1	Ap 2	Ap 3	Ap 1	Ap 2	Ap3
Accounts.AccNo<10000	C		RU						R
Accounts.AccNo>=10000		R							
Accounts.Type=Saving	CRD	RU	RUD		R				
Accounts.Type=Current		RU	R				CRUD	RU	R
. . .									
Accounts.Balance<50000	R		R			CRUD			R
Accounts.Balance>=50000		CR							
Account.Branch=Pune	CRUD	RU	CRUD			R	R		
Account.Branch=Nagpur		R		CRUD	CRUD	R		R	
Account.Branch=Mumbai							CRUD	RD	CRU

Figure 9: ALP cost for Branch=Pune

3.4. Construction of ALP Table

ALP values of all the attributes of the Accounts relation was computed from its MCRUD matrix. The attribute with highest precedence value will be treated as most important attribute for fragmentation. Table 2 shows the ALP table for Accounts relation.

Table 2: ALP table for Accounts relation

Attribute Name	Precedence
AccNo	6
Type	22
CustId	6
OpenDate	7
Balance	10
Branch	50

3.5. Generation of Predicate Set

Predicate set was generated for Branch, the attribute with highest locality precedence of Accounts relation.

P= {p1: Branch=Pune, p2: Branch=Nagpur, p3: Branch= Mumbai}

3.6. Fragmentation of Relation

According to the predicate set P, Account relation was fragmented and allocated to 3 sites (figure 10) shown in table 3-5.

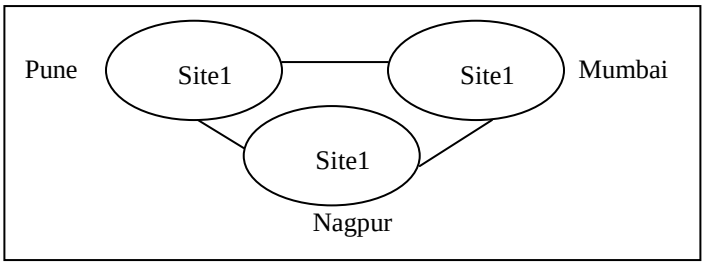

Figure 10: Distributed banking database system

Table 3: Part of Accounts relation allocated to site 1

Acc No	Type	CustId	OpenDate	Balance	Branch
01	Saving	101	05/01/12	30000	Pune
02	Current	102	18/01/12	48000	Pune
05	Current	105	12/03/12	50000	Pune

Table 4: Part of Accounts relation allocated to site 2

Acc No	Type	CustId	OpenDate	Balance	Branch
04	Saving	104	06/03/12	37750	Mumbai
07	Current	107	28/03/12	45000	Mumbai

Table 5: Part of Accounts relation allocated to site 3

Acc No	Type	CustId	OpenDate	Balance	Branch
03	Current	103	10/02/12	15900	Nagpur
06	Saving	106	25/03/12	25000	Nagpur

From the above result, we can see that our technique has successfully fragmented the Accounts relation and allocated the fragments among the sites of the distributed system. As we have only taken highest valued attribute from ALP table, no unwanted fragments were created. Other relations of the distributed banking database can be fragmented in the same way like Accounts.

For simplicity we have considered only four sites of the system for allocation. It is worth mentioning that our fragmentation technique will work in the same way for large number of sites of any distributed system.

IV. CONCLUSION

In this report, we presented an introduction to distributed database system through a study that targeted two main parts: in the first part we presented an exploration of distributed database environment and types of fragmentation. In the second part, we explore the horizontal fragmentation technique of a relation according to locality of precedence of its attributes.

Making proper fragmentation of the relations and allocation of the fragments is a major research area in distributed databases. Many techniques have been proposed by the researchers using empirical knowledge of data access and query frequencies. But proper fragmentation and allocation at the initial stage of a distributed database has not yet been addressed. In this report, we have presented a fragmentation technique to partition relations of a distributed database properly at the initial stage when no data access statistics and query execution frequencies are available. Using our technique, no additional complexity is added for allocating the fragments to the sites of a distributed database as fragmentation is synchronized with allocation. So performance of a DDBMS can be improved significantly by avoiding frequent remote access and high data transfer among the sites. This work can be extended to support fragmentation in distributed object oriented databases as well.

REFERENCES

[1] M. T. Ozsu and P. Valduriez, Principles of Distributed Database Systems, 2nd ed., New Jersey: Prentice-Hall, 1999.
[2] S. Ceri and G. Pelagatti, Distributed Databases Principles and System, 1st ed., New York: McGraw-Hill, 1984.
[3] S. Navathe, K. Karlapalem, and M. Ra, "A mixed fragmentation methodology for initial distributed database design," Journal of Computer and Software Engineering Vol. 3, No. 4 pp 395–426, 1995.
[4] F. Bai˜ao, M. Mattoso, and G. Zaverucha, "A distribution design methodology for object DBMS," Distributed and Parallel Databases, Springer, Vol. 16, No. 1, pp. 45–90, 2004.
[5] S. Ceri, M. Negri, and G. Pelagatti, "Horizontal data partitioning in database design," in Proc. ACM SIGMOD, 1982, pp. 128–136.
[6] S. B. Navathe, S. Ceri, G. Wiederhold, and J. Dour, "Vertical partitioning algorithms for database design," ACM Transactions on Database Systems (TODS), Vol. 9, No. 4, pp. 680–710, 1984.
[7] D. G. Shin, and K. B. Irani, "Fragmenting relations horizontally using a knowledge based approach," IEEE Transactions on Software Engineering (TSE), Vol. 17, No. 9, pp. 872–883, 1991.
[8] M. Ra, "Horizontal partitioning for distributed database design," In Advances in Database Research, World Scientific Publishing, pp. 101–120, 1993.
[9] C. H. Cheng, W. K. Lee, and K. F. Wong, "A genetic algorithm-based clustering approach for database partitioning," IEEE Transactions on Systems, Man, and Cybernetics, Vol. 32, No. 3, pp. 215–230, 2002.
[10] H. Ma, K. D. Schewe, and M. Kirchberg, "A heuristic approach to vertical fragmentation incorporating query information," in Proc. 7th International Baltic Conference on Databases and Information Systems (DB&IS), 2006, pp. 69–76.
[11] M. AlFares et al, "Vertical Partitioning for Database Design: A Grouping Algorithm", in Proc. International Conference on Software Engineering and Data Engineering (SEDE), 2007, pp. 218-223.
[12] F. F. Marwa, I. E. Ali, A. A. Hesham, "A heuristic approach for horizontal fragmentation and alllocation in DOODB," in Proc. INFOS2008, 2008, pp. 9-16.
[13] E. S. Abuelyaman, "An optimized scheme for vertical partitioning of a distributed database," Int. Journal of Computer Science & Network Security, Vol. 8, No.1, 2008.
[14] H. Mahboubi and J. Darmont, "Enhancing XML Data Warehouse Query Performance by Fragmentation," in Proc. ACM SAC09, 2009, pp.1555-1562.
[15] Haroun Rababaah, Dr. H. Hakimzadeh ,"Distributed Databases:Fundamentals and research",Advanced Database – B561. Spring 2005.
[16] M. Tamer Özsu, Patrick Valduriez, "Distributed Database Systems: Where Are We Now?" Appeared in IEEE Computer, Vol. 24, No. 8, August 1991.

AUTHORS

First Author – Ms. Priyanka R. Bhuyar, M.E.-I Year (CSE), Sipna college of Engineering & Technology, Amravati, India, priyanka.bhuyar@gmail.com

Second Author – Dr. A.D.Gawande, HOD (CMPS), Department of computer science & Engineering, Sipna College of Engineering & Technology, Amravati, India, adgawande@rediffmail.com

Third Author – Prof.A.B.Deshmukh, Professor(CMPS), Department of computer science & Engineering, Sipna College of Engineering & Technology, Amravati, India.