International Journal of Electrical Electronics & Computer Science Engineering
Volume 5, Issue 1 (February, 2018) | E-ISSN : 2348-2273 | P-ISSN : 2454-1222
Available Online at www.ijeecse.com

Comparative Analysis of Vertical Fragmentation Techniques in Distributed Environment

Mukta Goel[1], Shalini Bhaskar Bajaj[2]
[1]Assistant Professor, IT, TIT&S, Bhiwani, Haryana, India, [2]Professor, CSE, AUH, Gurugram, India
[1]rishu.muk@gmail.com, [2]shalinivimal@gmail.com

Abstract: Distributed database management system is a software system that manages the distributed database system and makes distribution transparent to the user. Efficient distributed databases can be designed using database fragmentation, allocation and replication. Fragmentation can be applied horizontally or vertically. In this paper, various algorithms used in vertical fragmentation techniques are reviewed and compared. Algorithms reviewed are Apriori algorithm, Enhanced Minimum Spanning Tree algorithm, Modified Bond energy algorithm and Knowledge Oriented clustering technique. These algorithms result in fragmented database and further allocation and replication can be applied to these fragments.

Keywords: EMST, KO, MBEA, ECRUD Matrix, AUM.

I. INTRODUCTION

In distributed database system, database may be stored at multiple sites. Logically it belongs to the same system but is distributed geographically at multiple locations. These locations or sites are interconnected with each other through a network. Accessing the remote site provides some delay that is not desired, so this delay can be overcome by using database fragmentation technique. Database is divided into small fragments. Fragmentation can be done horizontally or vertically [1]. Horizontal fragmentation provides all the attributes to the users. Dividing database horizontally can reduce communication cost as fragments are allocated to those sites where they are most frequently accessed. But sometimes the user is not interested in all the attributes. In that case the horizontal fragmentation does not provide useful results as attributes are duplicated in different fragments. When only some attributes of a database are required by the user, then instead of horizontal fragmentation, vertical fragmentation is done. Vertical fragmentation divides the database according to columns i.e. now fragments are formed in terms of attributes instead of complete row. The attributes that are used frequently can now be placed in same fragment. For vertical fragmentation various algorithms can be applied: Apriori, Enhanced Minimum Spanning Tree, Modified bond energy and Knowledge based clustering.

After fragmentation, allocation techniques are required to distribute the fragments to their respective sites in the network. These techniques are most important in distributed databases and treated separately [2]. In allocation, the fragment F is assigned to site S if it is most frequently accessed by site S. If a fragment is not accessed frequently at the site, then a copy of fragment can be allocated to that site. Assigning a copy of a fragment to a site is called replication. Allocation of fragments can be done at all sites, i.e. full replication or at selective sites, i.e. partial replication. The rest of the paper is organized as follows. Section (II) describes the formation of attribute usage matrix which will be used as the first step in all the algorithms. Section (III) describes the steps followed by all the vertical fragmentation techniques mentioned above. In Section (IV), allocation and replication of fragments is discussed. Section (V) concludes the various techniques described in section (III).

II. ATTRIBUTE USAGE MATRIX (AUM)

For fragmenting database vertically, usage of attribute at particular site is considered. Vertical fragmentation techniques use the Attribute Usage Matrix (AUM) [3]. AUM is 2-D matrix showing whether the attribute is accessed by query or not. If an attribute A is accessed by a query Q executing at site S then usage (A) at site S is represented by 1, otherwise 0. Usage of all attributes is represented as Attribute Usage Matrix. The following example shows the formation of Attribute Usage Matrix.

Example of Attribute Usage Matrix:

Consider a relation Emp with attributes Eid, Ename, Salary and City. Queries Q1, Q2, Q3 and Q4 are applied on the relation as shown below.

Q1: Find the Salary of Employee, if Eid is given

Select Salary

from Emp

where Eid = value;

Q2: Find the Eid, Ename and Salary of all Employees

Select Eid, Ename, Salary From Emp;

Q3: Find the name of Employee living in particular City

Select Ename

From Emp

Where City = value;

International Journal of Electrical Electronics & Computer Science Engineering
Volume 5, Issue 1 (February, 2018) | E-ISSN : 2348-2273 | P-ISSN : 2454-1222
Available Online at www.ijeecse.com

Q4: Find total Salary of Employees living in same City

Select sum (Salary)

From Emp

Group by City

Consider A1—Eid, A2—Ename, A3—Salary, A4—City. In AUM, rows represent queries and columns represent attributes as shown in Table 1. Value of each cell in the matrix will be either 0 or 1. Condition for inserting value in AUM is:

If A_i is used by Q_i then 1

Otherwise 0

Table I. Attribute Usage Matrix (AUM)

	A1	**A2**	**A3**	**A4**
Q1	1	0	1	0
Q2	1	1	1	0
Q3	0	1	0	1
Q4	0	0	1	1

After AUM is formed a similarity matrix is generated, called Attribute Similarity Matrix (ASM) as shown in Table 2. This table is formed between the attributes of a relation. Number of queries accessing both attributes A_i and A_j together is shown in this matrix and this is called similarity between attributes. For example, A1 and A3 are accessed together 2 times by queries Q1 and Q2. This value is stored in ASM under column A3 and row A1 of attributes. Every cell in ASM shows access frequency of two attributes A_i and A_j together.

Table II. Attribute Similarity Matrix (ASM)

	A1	**A2**	**A3**	**A4**
A1	0	1	2	0
A2	1	0	1	1
A3	2	1	0	1
A4	0	1	1	0

Highly accessed attributes are placed together in same fragment. Now these AUM and ASM matrices will be used by different vertical fragmentation approaches discussed in section III to achieve fragmentation.

III. DISTRIBUTED DATABASE VERTICAL FRAGMENTATION TECHNIQUES

1. Apriori Approach:

In this approach [4] frequent item sets of attributes are determined. Frequent item sets are sets of those attributes which are called together very often by different sites. Frequent item sets are formed for a prescribed min (support), i.e. for min (support) = 3, an attribute must appear at least in 3 queries. The value 3 is support measure called min (support). For generating frequent item sets, Attribute Usage Matrix of Table 1 is used. First of all support for each attribute is calculated separately. The attributes having value equal to or greater than min (support) are called frequent items. Then list of all pairs of frequent items is generated so that they all meet or exceed the min (support) value. If any pair is not frequent then its larger set cannot be frequent. So these item sets are pruned at this step. Then frequent sets of triples will be found excluding the pruned pairs of item sets. Frequent item sets of length k can be generated and all must follow the min (support) value. For generating different combinations of attributes min (support) can be varied. In this way using Apriori algorithm on Attribute Usage Matrix, frequent item sets of attributes are obtained. These frequent item sets are considered as fragments of the database. Each Fragment consists of different combination of attributes. Now fragments can be allocated and replicated to various sites.

Algorithm 1:

C_k: candidate item set of size k

$L_{k:}$ frequent item set of size k

1. L_1 = find frequent 1-item set
2. For (k=1;L_k != null; k++)
   {
   C_{k+1} = candidates generated from L_k
   }
3. For each transaction t in database
   {
   Increment the count of all candidates in C_{k+1} that are contained in t
   }
4. L_{k+1} = Candidate in C_{k+1} with min (support)

In Algorithm 1, fragmentation is done at initial stage. Different combination of attributes can be retrieved on the basis of min (support). If min support is higher, some important combinations may be lost. If min (support) is lower, irrelevant combinations may be there. Main advantage of this approach is that it doesn't involve much calculation. The disadvantage is that it may not cover all attributes.

2. Enhanced Minimum Spanning Tree (EMST) Approach:

International Journal of Electrical Electronics & Computer Science Engineering
Volume 5, Issue 1 (February, 2018) | E-ISSN : 2348-2273 | P-ISSN : 2454-1222
Available Online at www.ijeecse.com

In this approach [5] an enhanced minimum spanning tree is formed using Attribute Usage Matrix (AUM) and Attribute Similarity Matrix (ASM), shown in Table 1 and 2 respectively. ASM matrix is used to represent the similarity measure between attributes, i.e. how many times the attributes of the relation are accessed together. From ASM, Enhanced Minimum Spanning Tree is generated. In the tree each node is represented as an attribute and the weight of edge between nodes specify the similarity measure. Minimum spanning tree is used to find out the shortest path between two nodes, but here it is used to find out the similarity between two attributes. So, rather than picking minimum weight edge, it will pick up the edge with highest weight because weights on the tree represent the frequency that how many times these two attributes are called together and it should be high. Following algorithm is used to generate enhanced minimum spanning tree:

Algorithm 2:

Input: ASM matrix, fragment K

Output: List of fragments

1. Transform the ASM into a graph G.
2. Generate minimum spanning tree.
3. Create the graph G by selecting any one vertex V.
4. Initialize Empty graph EG with the selected vertex V.
5. Select a list of edges E from G such that at most one vertex of edge E is in EG.
6. From E select the edge M with Maximum weight.
7. Add M to EG.
8. Repeat from step 5 until all vertices are added to EG.
9. Partition the relation removing K-1 edges from EG.

If the value of K = 2, then the edge with minimum weight is removed and tree is divided into two fragments. Now these fragments can be allocated to the sites. If more fragments are required then value of K can be increased. For example, If K = 3 then, K-1 = 2, edges will be removed and total fragments formed will be 3. Every time the edge removed will be of minimum weight so that the fragments will consist of attributes with highest similarity measure. This approach results in fragments that cover all attributes.

3. Modified Bond Energy (MBEA) Algorithm:

In bond energy algorithm, similarity between two attributes is calculated using affinity measure [6]. It is based on simultaneous access of attributes A_i and A_j by query Q_k as shown in Table 1. Along with AUM, frequency of query at particular site and access per execution is also considered. Affinity is calculated by

Aff (A_i, A_j) =

$$\sum_{use(Q_k,A_i)=1 \wedge use(Q_k,A_j)=1} \sum_{\forall S_l} freq_l(Q_k) * acc_l(Q_k)$$

$use(Q_k, A_i) = 1, if\ A_i\ is\ used\ by\ Q_k$

$freq_l(Q_k)$ = access frequency of query *k* on site *l*

$acc_l(Q_k)$ = access per execution of query *k* on site *l*

After calculating Affinity, clusters of attributes are generated using the split function. Split permutes the rows and columns of Affinity matrix and cluster matrix is generated. Permutation maximizes the global affinity measure.

In modified approach, when the two attributes have concurrent occurrence in a query then its similarity is considered and for the same query concurrent absence of the attributes could be considered as a weighted measure of similarity. Attribute Affinity Measure is modified and is denoted as S_{ij}.

$$S_{ij} = n_{11} + n_{00}/n_{11} + n_{00} + w_1(n_{01} + n_{10})$$

This formula gives the match between two attributes directly. Here n_{11} and n_{00} show number of simultaneous presence and absence of attributes. In denominator, n_{10} and n_{01} represent that only one attribute is accessed by the query at a time. Weights are assigned to dissimilar pairs on the basis of their contribution to the similarity. In this approach less similar attributes are separated correctly.

4. Vertical Fragmentation Based on Knowledge Oriented (KO) Clustering Technique:

In this approach, similarity between two attributes is calculated and is denoted as affinity [7]. Attributes which are accessed maximum number of times together by various queries have highest affinity. Attribute Usage Matrix and Attribute Similarity Matrix are used to calculate similarity. After constructing these matrices, reference measure for each attribute is calculated. Reference measure determines that how many times an attribute is accessed by a particular query and is denoted as M (q_i, A_j). The following algorithm is used to construct fragments:

Algorithm 4:

1. Construct Attribute Usage Matrix
2. Calculate the reference measure of transaction q_i on attribute A_j
3. Calculate reference feature vector of each pair of attributes, such as VA_p and VA_r
4. Obtain similar matrix using S (A_p, A_r)

International Journal of Electrical Electronics & Computer Science Engineering
Volume 5, Issue 1 (February, 2018) | E-ISSN : 2348-2273 | P-ISSN : 2454-1222
Available Online at www.ijeecse.com

5. Place the attributes with highest similarity in same fragment.

In Algorithm 4, Reference measure of transaction q_i on attribute A_j is denoted by M (q_i, A_j).

M_{ij} = M (q_i, A_j) = Use (q_i, A_j)*f_i

M_{ij} is the frequency with which transaction q_i reference attribute A_j.

f_i is frequency of execution of qi on the sites. Use (q_i, A_j) is obtained from AUM as shown in Table 1. So, M_{ij} obtain the frequency with which transaction q_i refers to attribute A_j.

VA_j is reference feature vector of attribute A_j with reference to the transactions (q_1, q_2,… q_m). So VA_j will be:

$VA_j = M_{1j}, M_{2j}, \ldots M_{mj}$ with respect to $q_1, q_2, \ldots q_m$

The similarity measure of two attributes A_p and A_r now has two feature vectors with respect to transactions (q_1, q_2,… q_m):

$VA_p = (M_{1p}, M_{2p}, \ldots M_{mp})$

$VA_r = (M_{1r}, M_{2r}, \ldots M_{mr})$

$$S(A_p, A_r) = \frac{\sum_{i=1}^{m} M_{ip} * M_{ir}}{\sqrt{\sum_{i=1}^{m} M_{ip}^2} * \sqrt{\sum_{i=1}^{m} M_{ir}^2}}$$

The result of this equation generates similarity matrix and give the similarity value for each pair of attributes. Attributes having the highest similarity are placed in same fragment. Least similar attributes are placed in different fragment. This provide multiple fragments and each fragment consist of similar attributes. In this approach there is no need to generate tree as in algorithm 2, but many calculations has to be done.

IV. ALLOCATION AND REPLICATION

Using one of the above techniques fragments are generated. But fragmentation does not work alone. Fragments are of no use unless these are provided to some site. For this, allocation procedures are used. Various allocation procedures can be used. One is allocation on the basis of site usage. For this, Site usage matrix can be formed. This matrix represents the usage of fragments on a particular site, i.e. how frequently a fragment is accessed by a particular site. Suppose fragment F_k has maximum frequency at site S_j then it will be allocated to S_j.The fragments with frequency less than max (freq) are replicated at their respective sites. Only allocation is not sufficient so replication is also applied along with allocation [8, 9]. Another approach is use of CRUD matrix for allocation and replication. This matrix is used to define create, read, update and delete operations performed by queries at site S_j. Mainly two operations are considered for allocation and replication: update and read. The site at which more update operations are performed on fragment F_k, will be allocated F_k and the site at which more read operations for F_k are performed, F_k is replicated there. Each fragment is allocated to at least one site and replicated at more than one site.

V. COMPARISON

The techniques described in section III are compared and Table 3 shows the various parameters considered.

First parameter is usage of CRUD (Create, Read, Update and Delete) operations. If any one of these operations is performed then its entry is recorded in AUM. Use of CRUD matrix reduces the complicated calculations. There is no requirement of empirical data about the frequency of user queries.

Second parameter, similarity measure, shows how many times the attributes are accessed together by same queries. Two ways are there to show the similarity between two attributes: formation of ASM, as shown in Table 2, and forming frequent item sets as used in Apriori algorithm. Both methods require AUM as an input.

Third parameter is Query Accessed Frequency measure. This shows how many times a query is accessed by different sites.

Fourth parameter is Storage Cost. It determines the cost of storing the attributes at particular site. This cost is considered so that it will be easy to find out whether a particular fragment should be allocated to that site or not. If cost is very high then fragment is not allocated to that site.

Fifth parameter is Communication Cost between sites. If a user is on local site $S_i$and accessing a particular fragment from remote site S_j then communication cost between S_i and S_j is considered. If it is high then it is better to replicate the fragment at local site instead of accessing the remote site.

Sixth parameter is Allocation. After fragmentation, allocation of fragments to the sites is done. Fragment is placed on the site where the access frequency of that fragment is highest.

Seventh parameter is Replication. Fragments can be placed at one site or more than one sites. Replicas of fragments are created and copied on the sites where the access frequency is high. Some algorithms cover only fragmentation and some cover all techniques like fragmentation, allocation and replication.

International Journal of Electrical Electronics & Computer Science Engineering
Volume 5, Issue 1 (February, 2018) | E-ISSN : 2348-2273 | P-ISSN : 2454-1222
Available Online at www.ijeecse.com

Table III. Comparison of Techniques

	EMST	KO	Apriori	MBEA
CRUD	Yes	No	No	No
Similarity Measure	ASM	ASM	Frequent Item Set	ASM
QueryAccessed Frequency measure	No	Yes	Yes	Yes
Storage Cost	No	No	No	Yes
Communication Cost between Sites	No	No	No	Yes
Allocation	Yes	No	No	Yes
Replication	Yes	No	No	No

VI. CONCLUSION

As shown in Table 3, only EMST technique uses enhanced CRUD matrix. This technique does not involve empirical data about query executions and does not take into consideration location of sites. It covers allocation and replication of fragments with multiple transactions at each site. EMST is graph based technique while KO and MBEA use similarity measure for making fragments. Number of fragments formed in KO and MBEA are decided on the basis of similarity, but in EMST and Apriori, user decide the number of fragments to be formed. Only fragmentation is of no use until it is provided to some site. So allocation is also discussed in EMST and MBEA. Allocation at one site does not provide good results every time so replication of the fragment is required at other sites. Sites are chosen for allocating replicas of fragments on the basis of the number of accesses for that particular fragment. This comparison provides a comprehensive study of various vertical fragmentation techniques proposed in literature and will provide a future direction to the researchers.

VII. REFERENCES

[1] S.I. Khan and A.S.M. Latiful Hoque, (2012) "Scalability and Performance Analysis of CRUD Matrix Based Fragmentation Technique for Distributed Database", 15th IEEE International Conference on Computer and Information Technology (ICCIT), 22-24 Dec, Chittagong, Bangladesh.

[2] Hassan I. Abdalla and Ali A. Amer, (2012) "Dynamic Horizontal Fragmentation, Replication and Allocation Model in DDBSs", IEEE International Conference on Information Technology and e-Services, 24-26 March, pp. 1-6.

[3] Shahidul Islam Khan, Dr. A. S. M. Latiful Hoque, (2010) "A New Technique for Database Fragmentation", International Journal of Computer Applications, Volume 5– No.9, pp. 20-24.

[4] Ramesh Dharavath, Vikas Kumar, Chiranjeev Kumar, Amit Kumar, (2013) "An Apriori-Based Vertical Fragmentation Technique for Heterogeneous Distributed Database Transactions", Intelligent Computing, Networking, and Informatics, Advances in Intelligent Systems and Computing, Springer, New Delhi, vol 243, pp. 687-695.

[5] Ahmed E. Abdel Raouf, Nagwa L. Badr, M. F. Tolba, (2015) "An Optimized Scheme for Vertical Fragmentation, Allocation and Replication of a Distributed Database", IEEE Seventh International Conference on Intelligent Computing and Information Systems (ICICIS' 15), 12-14 December, pp. 506-513.

[6] Hossein Rahimi, Fereshteh-Azadi Parand, Davoud Riahi, (2016) "Hierarchical Simultaneous Vertical Fragmentation and Allocation using Modified Bond Energy Algorithm in Distributed Databases", Applied Computing and Informatics, Saudi Computer Society, King Saud University, pp. 1-7.

[7] Van Nghia Luong, Ha Huy CuongNguyen and Van Son Le, (2015) "An improvement on fragmentation in distribution Database Design Based on Knowledge-Oriented Clustering Techniques", IJCSIS, Vol. 13, No. 5.

[8] Ahmed E. Abdel Raouf, Nagwa L. Badr, M. F. Tolba, (2017) "Distributed Database System (DSS) Design over a Cloud Environment", Multimedia and Forensic Security, Springer International Publishing, Chapter 10, pp. 97-115.

[9] Jon Olav Hauglid, Norvald H. Ryeng, Kjetil Norvag, (2010) "DYFRAM: Dynamic Fragmentation and Replica Management in Distributed Database System", Distributed Parallel Databases, Springer Open Access, Vol. 28, pp. 157-185.