A New Fragmentation and Allocation Technique for Distributed Database System

by

Shahidul Islam Khan

A thesis submitted to the Department of Computer Science and Engineering in partial fulfillment of the requirements for the degree of MASTER OF SCIENCE IN COMPUTER SCIENCE AND ENGINEERING

Department of Computer Science and Engineering
BANGLADESH UNIVERSITY OF ENGINEERING AND TECHNOLOGY, DHAKA

March, 2011

The thesis **“A New Fragmentation and Allocation Technique for Distributed Database System”,** submitted by Shahidul Islam Khan, Roll No. 040505016P, Session: April 2005, to the Department of Computer Science and Engineering, Bangladesh University of Engineering and Technology, has been accepted as satisfactory for the partial fulfillment of the requirements for the degree of Master of Science in Engineering (Computer Science and Engineering) and approved as to its style and contents for the examination held on March 13 , 2011.

<u>Board of Examiners</u>

Abstract

An efficient way of improving performance of a database management system is distributed processing. Distribution of data involves fragmentation, replication and allocation process. Previous research works provided fragmentation solution based on empirical data about the type and frequency of the queries. These solutions are not suitable at the initial stage of a distributed database.

In this thesis we have presented a fragmentation technique namely MCRUD Matrix based Fragmentation (MMF) that can be applied at the initial stage as well as in later stages of a distributed database system for partitioning the relations. Instead of using empirical data, we have developed the matrix namely Modified Create, Read, Update and Delete to make fragmentation decisions properly. The main concept of MMF is finding the precedence of attributes to increase data locality. We have named it Attribute Locality Precedence (ALP). The relations have been fragmented considering the highest ALP value among the attributes

Allocation of fragments is done simultaneously in our technique. So using MMF, no additional complexity is added for allocating the fragments to the sites of a distributed database as fragmentation is synchronized with allocation. Performance of a DDBMS can be improved significantly by avoiding frequent remote access and high data transfer among the sites. Result shows that the proposed technique can solve initial fragmentation problem of distributed system properly.

Chapter 1
Introduction

A distributed database is a collection of data that logically belongs to the same system but spreads over the sites of a computer network. A distributed database management system (DDBMS) is defined as the software system that provides the management of the distributed database and makes the distribution transparent to the users [1], [2]. 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 and software advance rapidly and prices of network equipments fall every day, developing distributed database systems have become more and more feasible. Design of efficient distributed databases is one of the major research problems in database and information technology areas. DDBMS is an efficient way of improving the performance of applications that manipulate large volumes of data. Primary concerns of distributed database design are fragmentation of the relations in case of relational database or classes in case of object oriented databases, allocation of the fragments to different sites of the distributed system, and local optimization in each site [1], [2].

1.1 Background

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. Horizontal fragmentation (HF) allows a relation or class to be partitioned into disjoint tuples or instances. Vertical fragmentation (VF) partitioned a relation or class into disjoint sets of columns or attributes except the primary key. Combination of horizontal and vertical fragmentations forms the mixed or hybrid fragmentations (MF). Allocation is the process of assigning the fragments of a database on the sites of a distributed network. The replication of fragments improves reliability and efficiency of read-only queries but increase update cost.

Thus the main reasons of fragmentation of the relations are to: increase locality of reference of the queries, improve reliability and availability of data and performance of the system, balance storage capacities and minimize communication costs among sites [1]-[4].

1.2 Problem Definition

In distributed database design, the foundations of fragmentation (horizontal, vertical or mixed) of relations are:

- Frequency of different queries executed in a system in a specified time,
- Affinity matrix of minterm predicates constructed from combination of predicates
- Attribute affinity matrix constructed based on the relationship between different attributes of a table and run time transactions those access the attributes

To know actual query frequencies or to construct above matrices sufficient experiential data are required. These data are not available in most cases at initial stage of a distributed database.

If proper distribution is not done during the initial stage of the DDBMS, data distribution technique based on empirical data requires huge data transfer cost in most cases. So to reduce the data transfer cost during the evolution of DDBMS, fragmentation and allocation at initial stage is very important, very few work have been found in the literature in this regard.

Almost all the previous techniques concentrated only fragmentation problem and overlooked allocation problem to reduce the complexity of the problem. But overall performance of a distributed system fragmented by a very good fragmentation technique can be very low if proper allocation of the fragments to the sites cannot be ensured.

1.3 Objectives of the Thesis

The main objectives of the thesis are to:

a) design of a fragmentation technique for distributed database management system (DDBMS) that can be applied at the initial stage of a DDBMS when no empirical data are available as well as in any stage of execution of DDBMS for partitioning the relations,
b) design of an allocation technique to allocate the fragments of the relations in the sites of DDBMS properly,
c) design algorithms to implement our fragmentation and allocation technique,
d) analyze the performance of the technique by applying it in designing a DDBMS and
e) compare the performance of our technique with the existing techniques to find the effectiveness and efficiency of the technique.

1.4 Overview of the Thesis

In this thesis we have presented a fragmentation technique namely MCRUD Matrix based Fragmentation (MMF) to partition relations of a distributed database properly at the initial stage. Instead of using empirical data, we have developed a matrix namely Modified Create, Read, Update and Delete (MCRUD) to make fragmentation decisions. 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 improves the bandwidth of the system as well.

1.5 Organization of the Thesis

In chapter 2, a survey of the research in horizontal, vertical and mixed fragmentation techniques of distributed database is presented. Limitations of the available fragmentation techniques are also discussed in brief. The motivation for the research work performed by the author and reported in this thesis is to overcome some of these limitations.

Chapter 3 describes the details of MCRUD Matrix based Fragmentation (MMF) technique that has been used for fragmentation of the relations of distributed database and allocation of the fragments in the sites of the system. Our fragmentation and allocation technique is explained in detail.

Chapter 4 describes the experimental work that has been carried out to investigate the performance of our proposed technique. Results obtained from the experimental works are thoroughly discussed and compared with the experimental results of other existing techniques.

Finally, Chapter 5 presents conclusions of this thesis work and provides suggestions for future research.

Chapter 2
Literature Survey

In this chapter, we have reviewed literature of various fragmentation techniques. We have used separate sections and sub sections for presenting the review of horizontal, vertical and mixed fragmentation. Finally a summary of the features and limitations of the present works is presented at the end.

2.1 Design Techniques: Fragmentation and Allocation

Design of distributed database is one of the major research problems whose solution is supposed to improve performance of a distributed system. It involves database design, data population, fragmentation of databases, allocation and replication of the fragments, and local optimization. Fragmentation and allocation are the most important elements of a distributed database design phase. They play important roles in the development of a cost efficient system [1].

2.1.1 Fragmentation

Fragmentation is a design technique to divide a single database into two or more partitions such that by the combination of the partitions original database can be found without any loss or addition of information. This reduces the amount of irrelevant data accessed by the application, thus reducing the number of disk accesses. The result of the fragmentation process is a set of fragments defined by a fragmentation schema. Fragmentation in relational database can be horizontal, vertical or mixed.

Horizontal fragmentation (HF) partitions a relation or a class into disjoint parts (fragments), which will have exactly the same structure but different contents. Thus a horizontal fragment of a relation or class contains a subset of the whole relation or class instance.

Vertical fragmentation (VF) results in attributes and methods being partitioned into different fragments and therefore reduces irrelevant data accessed by local applications [1] - [4].

Mixed or Hybrid fragmentation (MF) can be achieve by performed VF followed by HF or vice versa. Thus benefits of both horizontal and vertical fragmentation can be attained [3].

2.1.2 Allocation

Allocation is the process of assigning a site to each fragment on the distributed network after the database has been properly fragmented [1]. When data are allocated, it may either be replicated or maintained as a single copy in only one site. The replication of fragments improves the reliability and efficiency of read-only queries and increases cost of update queries. The intention of allocation is to minimize the data transfer cost and the number of messages needed to process a given set of applications, so that the system functions effectively and efficiently [1], [5].

The individual tuple or attribute of a relation cannot be considered as the unit of allocation because the allocation problem would become unmanageable. The fragments are constituted by grouping tuples or attributes that have the same "properties" from the viewpoint of their application. This is based on the idea that two elements in the same fragment that have the same "properties" will be accessed by the applications together. Therefore, the fragments obtained in this way are the appropriate units of allocation [2].

2.1.3 Benefits of Database Fragmentation

The reasons for fragmenting databases are discussed in [1], [2]:

- Applications are usually based on the views of subsets of relations. Thus the applications often access any subset of an entire relation locally. Therefore, fragmentation can reduce irrelevant data accesses and increase data local availability.
- If there is a relation on which many application views are defined at different sites, storing a given relation at one site will result in an unnecessarily high volume of remote data accesses. Storing a given relation at different sites will cause problems in executing updates and may not be desirable if storage is limited.

- The decomposition of a relation into fragments permits many transactions to be executed concurrently and results in the parallel execution of a single query by dividing it into a set of sub-queries that operate on fragments.

2.1.4 Drawback of Database Fragmentation

Fragmentation may introduce the following problems [1]:

- Applications whose views are defined on more than one fragment may suffer performance degradation when the relations are not partitioned into mutually exclusive fragments.
- When the attributes participating in a dependency of a relation are decomposed into different fragments and stored at different sites, the task of checking for dependencies would result in chasing after data in a number of sites.

2.1.5 Complexity of the Problem

The combined problem of fragmentation and allocation is proven NP-hard [6]. In the case of Horizontal fragmentation, if *n* simple predicates are considered then the number of horizontal fragments using minterm predicates is 2^n. If there are k nodes, the complexity of allocating horizontal fragments is O (k^{2^n}).

For example, using 6 simple predicates to perform horizontal fragmentation results in $2^6 = 64$ fragments. To find the optimal allocation of the fragments in 4 sites one needs to compare all the $4^{64} \approx 10^{39}$ possible allocations.

For Vertical fragmentation if a relation has m non-primary key attributes, number of possible fragments: Bell number B (m) $\approx m^m$. The fragment allocation is of complexity O (k^{m^m}).

Due to the complexity of both fragmentation and allocation, allocations of the fragments are often treated independently than fragmentation of the database.

2.2 Horizontal Fragmentation

There are two types of horizontal fragmentation, primary and derived. Primary horizontal fragmentation of a relation or a class is performed using predicates of queries accessing this relation or class, while derived horizontal fragmentation of a relation or a class is performed based on horizontal fragmentation of another relation or class.

2.2.1 Primary Horizontal Fragmentation for Relational Databases

In the context of the relational data model, existing approaches for horizontal fragmentation mainly fall into following three categories [7], [1]:

- minterm-predicate-based approaches: which perform primary horizontal fragmentation using a set of minterm predicates, e.g., [1], [2], [8].
- affinity-based approaches: which first group predicates according to predicate affinities and then perform primary horizontal fragmentation using conjunctions of the grouped predicates, e.g., [9] - [12].
- other approaches: approaches other than minterm predicate or predicate affinity based approach, e.g., [13] – [16].

2.2.1.1 Minterm Predicate Based Approaches

Using minterm predicates to perform horizontal fragmentation was first proposed in [8] to fragment file horizontally to optimize the number of accesses performed by the application programs to different portions of data. They state that the minterm fragments contain records that are homogeneously accessed by all transactions and therefore are the proper units of allocation.

Ceri and Pelagatti [2] proposed to use minterm predicates to fragment relations of a database. To perform primary horizontal fragmentation, a set of disjoint and complete selection predicates have to be determined. Firstly, based on application information simple predicates $P = \{p_1, . . . , p_n\}$ have to be derived, which should satisfy complete and minimal properties. Then a set of minterm predicates are constructed from P. Often the size of the set of simple predicates is big, and the cost of computation might be too expensive. If resulting minterm fragments of a predicate are relevant and accessed differently by queries at the same site, they may still be allocated at the same site. That is, the fragmentation is not necessary and the predicate is not needed for fragmentation.

Ozsu and Valduriez [1] presented COM_MIN, an iterative algorithm to generate a complete and minimal set of predicates P_r' from a given set of simple predicates P_r. The algorithm checks each predicate p_i in the given set of simple predicates P_r to see if it can be used to partition the relation R into at least two parts which are accessed differently by at least one application. If p_i satisfies the fundamental rule of completeness and minimality then it should be included in P_r'. If p_i is non-relevant then it should be removed from P_r'. But this algorithm is not practical because checking p_i cannot be defined with machine readable language. An algorithm named PHORIZONTAL is introduced to describe primary horizontal fragmentation. It uses the algorithm COM_MIN and a set of implications I as inputs to produce a set of satisfiable miniterm predicates M. If a minterm predicate m_i is contradictory to an implication rule in I, then it is removed from M. Minterm fragments are defined according to the set of satisfiable minterm predicates M. But the set I of implications is hard to define. In fact, the algorithm is not very practical, as it will always result in a subset P_r' of Pr, the set of minterm predicates M′ determined by P_r' and the corresponding set of fragments. Simple predicates are omitted from Pr if they do not contribute to the fragmentation that is if they violate the minimality principle. This results in considering just the simple predicates in the most important queries and to take all satisfiable minterm predicates. This obviously leads to fragments that are accessed differently by at least two queries. The algorithm further does not give executable rules for eliminating the unsatisfiable minterm predicates. The major problem, however, is that the number of fragments resulting from the algorithm is exponential in the size of Pr. In practice, it would be important to reduce this number significantly, which would mean to re-combine some of the fragments. In fact, this implies giving up the completeness principle and replacing it by optimization criteria based on a cost model.

2.2.1.2 Affinity Based Approaches

To avoid the complexity of checking completeness of the set of simple predicates, Zhang [9] adopted an affinity-based vertical fragmentation approach to horizontal fragmentation. This approach takes predicate usage and predicate affinity matrix as input and employs the bond energy algorithm to cluster predicates. However, the fragments in the resulting fragmentation schema may overlap each other and therefore cannot satisfy the correctness criteria of fragmentation. Ra [10] presented a graph-based algorithm for horizontal fragmentation, with which predicates are clustered based on the predicate affinities. To remove overlapping, an adjust function is presented to merge two overlapped fragments if merging can reduce transaction costs using cost functions. However, the cost function does not show how costs are computed.

Using predicate matrix as input, Cheng et al. [11] proposed a genetic algorithm-based clustering approach, which treats horizontal fragmentation as a traveling salesman problem (TSP). Horizontal fragmentation is achieved by performing selection operation using the set of the grouped predicates, which are grouped according to the distances. The distance of each pair of attributes actually measure the access frequencies of transactions that do not access the pair attributes together. Additional analysis is needed to simplify the clusters of predicates. The objective of this approach is to group attributes such that the difference between the average distance within groups and the average distance between groups is minimized. However, there is no proof that this approach can indeed minimize the total query costs.

Mahboubi H. and Darmont J. [12] used predicate affinity for horizontal fragmentation in data warehouse. They showed that affinity-based fragmentation out-performed predicate construction based fragmentation in their experiments. They adopted primary horizontal fragmentation in XML context. Obviously, none of the affinity-based horizontal fragmentation approaches takes into consideration of data locality while clustering predicates.

2.2.1.3 Other Approaches

Approaches other than affinity-based and minterm-based approaches are also found in the literature. Chang and Cheng [13] proposed a methodology of decomposition based on mapping user views onto a global schema. However, there is neither clear procedure for processing decomposition nor evaluation of the resultant decomposition.

Shin and Irani [14] proposed a knowledge-based approach in which user reference clusters are derived from the user queries to the database and the knowledge about the data. Their paper mainly emphasizes the extension of first order logic without any procedure or algorithm on how to perform horizontal fragmentation procedurally. Also, the completeness of the knowledge base is a critical issue for the correctness of the final fragmentation. Shin and Irani [15] extended their work of [14] by presenting an example to illustrate how fragmentation can provide enhanced control over data replication and reduce costs on selection operations. However, the discussion is not supported by any cost model.

To minimize the total number of disk accesses, Khalil et al. [16] introduced a horizontal transaction-based partitioning algorithm, which takes a predicate usage matrix as input.

2.2.2 Primary Horizontal Fragmentation for Object Oriented Databases

Fragmentation of object-oriented databases (OODBs) using horizontal fragmentation technique have been proposed since 1990s. Algorithms proposed in the literature are mainly affinity-based and cost driven [7].

Karlapalem et al. [17] proposed to use predicate affinities to perform horizontal fragmentation. However, there is no detailed method on how to perform fragmentation. Ma et al. [18] provided a design procedure of horizontal fragmentation, including primary horizontal fragmentation and derived horizontal fragmentation, for complex data model. The paper presents an approach to minimizing the query processing costs by performing horizontal fragmentation and fragment allocation simultaneously. The technique uses a cost model and tried to globally minimizing costs.

Bellatreche et al. [19] stated that the effect of horizontal fragmentation should be measured by evaluating the performance of the applications in a distributed database system. Cost-Driven Algorithm is presented to find a scheme that lead to the lowest total query cost based on a cost model. However, in the cost model CPU costs and network communication costs are disregarded because only centralized databases are considered. Therefore it cannot be applied to distributed databases, where network communication cost is predominant in calculating total costs. Bellatreche et al. [20] have studied horizontal class partitioning with input as queries which contain either simple and component predicates, the primary algorithm (PA) is based on a graph theoretic algorithm which clusters a set of predicates into a set of HCFs.

A taxonomy of the fragmentation problem in a distributed object based system is presented in [21] to include four realizable class models, simple attributes and methods, complex attributes and simple methods, simple attributes and complex methods and complex attributes and methods. For one of these class models, simple attributes and methods, a set of detailed horizontal fragmentation algorithms are proposed. Continuing the work in [21], Ezeife and Barker [22] presented a comprehensive set of algorithms for horizontally fragmenting over all the four realizable class models following [1]. Ezeife and Zheng [23] have proposed an Object Horizontal Partition Evaluator (OHPE), which contains two components, the local irrelevant access cost and the remote relevant access cost. However, both components only measure the number of instances of a fragment without taking into consideration of size of the object and network information. A class is fragmented using all predicates from the queries accessing the class directly, predicates of all queries of all the descendants of the class that access the class, and predicates of all its containing classes accessing the class, and predicates of all its complex method classes. An example is presented to show how to compute the performance of the object horizontal fragmentation schemata with proposed OHPE. However, it is not shown how the horizontal fragmentation schemata are achieved and how fragments are allocated. An algorithm is proposed to re-fragment the class once input information is changed, including the user query access pattern and frequencies, changes in class hierarchy, change in class composition hierarchy, and change in the instance objects of classes.

Bai˜oo et al. [24], [4] adopted the algorithm proposed in [3] and take predicate affinity matrix as input to build a predicate affinity graph that is used to define horizontal class fragments. Again, the resulting horizontal fragmentation schema only reflects the togetherness of data accessed by transactions or queries but cannot reflect the affinities between data and network sites, that is, data locality.

Marwa et al. [25] uses the instance request matrix to horizontally fragment DOODB. The proposed algorithm is based on the idea that addresses vertical fragmentation and allocation simultaneously for relational system but in the context of horizontal fragmentation of an object model. The investigated approach uses a cost model and claimed to globally minimizing the fragmentation and allocation costs though they did not provide any comparison of performance with recent algorithms.

2.2.3 Derived Horizontal Fragmentation for Relational Databases

Derived fragmentation refers to horizontal fragmentation defined on a member relation r of a link according to fragmentation of one of its owner relations s [26], [1]. Derived horizontal fragmentation can be performed by applying semi-join operations.

In [26], a link among relations is introduced to depict the binary relationship between relations. A direct link is drawn between relations that are related to each other by an equijoin operation. The direction of a link shows a one-to-many relationship. It is assumed that the join attributes for a link include the primary key of the owner of the link. Note that, in our complex value data model an owner type is actually a component of a member type.

In [1] it is emphasized that care should be taken with the relations that have more than one link to the owner relations. Two criteria are suggested in such cases: choosing the fragmentation with better join characteristics or choosing the fragmentation used in more applications.

2.2.4 Derived Horizontal Fragmentation for Object Oriented Databases

Unlike the relational model situation the definition of derived horizontal fragmentation is not straightforward in the object-oriented data model.

In [27] owner and member relationships are defined based on paths that an operation navigates through, where a member class is always defined at the “1” side of the relationship link. Owner and member relationship is not defined for many to many relationship.

In [17], derived horizontal fragmentation of a class is performed using component predicates that are defined with path expressions. This may result a set of overlapped fragments. The last step is then to remove overlap between fragments according to the sum of the frequency of accesses of the fragments. The overlapped objects are removed from the fragments that are accessed less frequently. However, it is not necessary to distinguish between simple attributes or complicated attributes. Similarly, it is not necessary to distinguish simple predicates and component predicates. The derived fragmentation is defined as using component predicates, the sink of which is an attribute of another class. The proposed algorithm uses logical connectives but does not mention when each connective should be used. Also, a predicate defined on a path does not always mean that the predicate has a sink as an attribute of another class.

In [21], derived horizontal fragments of a class are generated according to primary fragments of its subclasses, its complex attributes (contained classes), and/or its complex methods. Heuristics are proposed to choose the most appropriate primary fragment to merge with each derived fragment of the member class. At last, derived fragments are merged with a primary fragment that has the highest affinity with it. However, this approach leads to overlaps between resulting derived fragments. Inheritance links are considered in the process of horizontal fragmentation. It is assumed that a pointer is contained in an instance of a storage structure for a class in the class hierarchy. There is no evaluation of the proposed algorithms regarding how it will improve the system performance.

In [4], derived horizontal fragmentation of each member class is performed according to its owner class in frag (owner, member) list, which is based on the owner-member classification. Derived horizontal fragmentation is implemented with a semi-join on the attribute used by the most frequent navigation operations from the member class to the owner class. However, it is not clear how to decide the owner classes to be used for fragmentation. The resulting distributed database schema is analyzed to show improvements in system performance. However, the analysis neither considers queries as distributed queries nor uses any cost models.

2.3 Vertical Fragmentation

Vertical fragmentation can be applied to different areas: file partitioning in centralized environment, data distribution among different levels of memory hierarchies of a database, and data distribution in distributed databases. For applications accessing fragments on different memory levels, the costs are dominated by the cost of retrieving data from secondary storage to main memory while for distributed databases, query costs are dominated by remote data transportation costs. The following reviews the work done regarding vertical fragmentation for relational databases.

2.3.1 Vertical Fragmentation for Relational Databases

Vertical fragmentation of file system has been studied since 1970s. There are two main approaches [7]:

- The pure affinity-based approach takes attribute affinities as the measure of togetherness of attributes to fragment attributes of a relation schema. Research work includes [28]-[35].
- The cost-driven approach uses a cost model while partitioning attributes of a relation schema. Research work includes [36] - [41].

2.3.1.1 Affinity Based Approaches

Affinity-based vertical fragmentation was first proposed by Hoffer and Severance [28], who used Bond Energy Algorithm (BEA) to cluster attributes according to the affinities between attributes. Since then the affinity measure has been widely used for solving the fragmentation problems. Navathe et al.[29] extended the BEA approach in [28] by proposing algorithms that produce non-overlapping fragments and overlapping fragments. This approach minimizes the number of fragments accessed by transactions while considering storage cost factors involved in storing the fragments. This approach consists of two steps:

- In the first step the given input parameters are used in the form of an attribute usage matrix (AUM) to construct an attribute affinity matrix (AAM) on which clustering is performed.
- In the second step estimated cost factors, which reflect the physical environment of fragment storage, are considered to further refine the partitioning schema.

The paper in [29] discusses vertical partitioning problem in three contexts: a database stored on devices of a single type, a database stored in different memory levels, and distributed database. Allocation of fragments in distributed databases targets at maximizing the amount of local transaction processing. At the first stage, the same objective function is used for single site one memory level, and single site with multiple memory levels. The objective function for distributed databases is designed with the consideration of the ratio of the transaction volume satisfied by the upper block to the total transaction volume and the ratio of the size of the fragment defined by upper block to the size of the object. At the second step, an objective function is presented to include cost factors, each of which is of different weight in different contexts. However, there is no justification of the values of the factors. Also, the transportation cost factor is fixed for all transactions between any pair of network nodes.

Navathe and Ra [30] improved the previous work [29] by proposing a vertical partitioning algorithm using a graphical technique. The major feature of this graphical approach is that all fragments are generated by one iteration in a time of $O(n^2)$, which is better than $O(n^2 \log n)$, the complexity of vertical partitioning algorithm in [29]. In the meantime, there is no need of an arbitrary empirical objective function for the algorithm of partitioning. This graphical approach starts with an attribute affinity matrix, based on which, an affinity graph is constructed, and then a linearly connected spanning tree is formed. Affinity cycles, which are the candidate partitions, are constructed simultaneously with the growing of the spanning tree. Partitions are made according to the weight of the edges comparing with the weight of each edge of candidate cycles. The output of the algorithm is a set of vertical partitions of a given relation. However, the resulted number of fragments is not related to the number of Sites of a distributed system. If the resulted number of fragments is bigger than the number of network nodes, fragment recombination needs to be performed. In addition, there is no evaluation of goodness of the resulting vertical fragmentation schema as to how it will improve the distributed database system performance.

Lin and Zhang [31] pointed out that the restriction of an affinity cycle results in formalization is an NP-hard problem and therefore the claimed properties in [30] cannot be guaranteed. A new graphic algorithm is proposed by using 2-connectivity instead of affinity cycle to construct non-overlapping fragments, which is later allocated to distributed network nodes.

Ma H. et al. (2006) used an attribute uses frequency matrix (AUFM) and a cost model for vertical fragmentation [32]. This paper addresses vertical fragmentation and allocation simultaneously in the context of the relational data model. The core of the paper is a heuristic approach to vertical fragmentation, which uses a cost model and is targeted at globally minimizing access costs. M. Alfares et al. used AAM to generate groups based on affinity values [33].

Ngo T. H. used AUM & partitioned a relation into two vertical fragments in the cache memory [34]. In this paper they derived an objective function for vertical partitioning with a new estimated criterion: *cache hit probability*.

Runceanu A. presented a partition evaluator that used AUM to select attributes for vertical fragmentation [35]. In this paper implementation of a heuristic algorithm is presented that uses an objective function who takes over information about the administrated dates in a distributed database and evaluates all the scheme of the database vertical fragmentation.

2.3.1.2 Cost Based Approaches

Cost based vertical fragmentation approaches use cost functions to make proper fragmentation decision. Cornell and Yu [36] discussed vertical fragmentation for relational databases and considered that the response time of transactions is impacted by the number of disk accesses by the transaction. Considering the utility of vertical fragmentation is to minimize the number of disk accesses, Cornell and Yu [36], [37] proposed a two step methodology that consists of a query analysis step to estimate the parameters and a binary partitioning step that can be applied recursively. Chu [38] presented two procedures to solve the attribute partitioning problem to improve system performance by transferring small segments instead of big non-partitioned relations between the primary and the secondary storage. He first defined two concepts, sufficient and support, on which two procedures, MAX and FORWARD SELECTION, are proposed which are targeted at maximizing the value of v, the total reduction of costs which are expressed in terms of the number of disk accesses. The important characteristic of these two procedures is that they treat the transactions instead of the attributes as the decision variables. Therefore, the run time of these procedures does not depend on the number of attributes and can be efficiently executed when the number of attributes is very big. However, this approach may not be suitable to the situation when there are a large number of transactions but a small number of attributes over a relation. Also, this approach only discusses the problem of attribute partitioning for two memory levels on one disk. The objective function only counts the number of disk accesses. Approaches in both [36] and [38] are not suited for distributed databases.

Chakravarthy et al. [39] argued that there should be a way to measure the goodness of a vertical fragmentation schema. For this purpose they set up an objective function, Partition Evaluator (PE), for evaluating different vertical fragmentation algorithms using the same criteria. The PE consists of two components, irrelevant local attribute access cost and relevant remote attribute access cost. However, relevant remote attribute access cost reflects the number of relevant attributes in a fragment accessed remotely with respect to all other fragments by all transactions. Therefore, the PE cannot be used in distributed databases because neither size nor network transaction cost factors have been considered.

Son and Kim [40] argued that vertical partitioning problem should consider the number of fragments finally generated. They discussed n-ary vertical partitioning problem which are more flexible than the optimal partitioning. Their novel contribution is an objective function which aims at minimizing not only the frequency of query accesses to different fragments but also the frequency of interfered accesses between queries. In the objective function, data localization is not considered because queries are not distinguished between sites.

2.3.1.3 Initial Vertical Fragmentation

Abuelyaman [41] provided a solution of initial fragmentation of database using vertical fragmentation technique namely StatPart. To fragment a relation, it starts with a randomly generated matrix of attribute vs. queries called reflexivity matrix. It then construct symmetry matrix from reflexivity matrix using two equations. Symmetry matrix is inputted to transitivity module which uses an algorithm to produce two set of attributes those will be used to break the relation into two binary vertical fragments.

Main two drawbacks of StatPart [41] are:

- It can suggest only two binary vertical fragments independent of number of sites of the distributed system. So this technique is not suitable for a distributed system with more than two allocation sites.
- As it starts with a randomly generated matrix that represents the relationship among attributes and queries, optimum fragmentation decision cannot be provided using this algorithm. So it continuously shift attributes from one fragment to another fragment trial and error basis to improve hit ratio.

2.3.2 Vertical Fragmentation for Object Oriented Databases

In the context of object-oriented data model, fragmentation algorithms are mainly affinity based, with different affinities used as parameters, e.g., attribute affinities, or instance variable affinities [42], [16]. An increasing demand on the performance of object oriented database systems has resulted in the adoption of vertical fragmentation techniques from the relational databases. The features of the object-oriented data model (such as inheritance, encapsulation, ISA relationship and the presence of method) add to the complexity of the partitioning problem [43]. Based on the existing work for vertical fragmentation for object-oriented databases, taxonomy is proposed in [43] has two categories, method based and attributed based.

Karlapalem and Li [44] discussed the foundations of vertical fragmentation by giving a formal representation of vertical fragmentation. Issues regarding internal representation and reconstruction of fragments are discussed. In addition, approaches for supporting ISA relationships and methods are briefly mentioned. There are neither algorithms for horizontal, path and vertical fragmentation nor discussion on when vertical fragmentation should be applied to schema or to methods. Karlapalem et al. [45] presented guidelines for method induced partitioning in object-oriented databases. Karlapalem and Li [46] extended the work done in [44] and [45] through detailed discussions of the method induced partitioning on different types of methods in terms of instance variables accessed, and the complexity of the methods, with the focus on single method partitioning. There is no algorithm proposed in the paper.

Treating relational database as a special case of object-oriented databases, Malinowski and Chakravarthy [47] generalized the work for relational databases in [39] to object-oriented databases. Vertical fragmentation is performed using Transaction-Method Usage Matrix, Method-Method Usage Matrix and Method-Attribute Usage Matrix. A partition evaluator function for object-oriented databases, PEOO, adopted from the relational databases, is used to evaluate all possible combinations of attributes with the number of fragments varying from one to the total number of attributes in the class. However, without considering the size of data transferred among network nodes, the PEOO actually measures the number of irrelevant local accesses and relevant remote accesses rather than real total query costs.

Ezeife and Barker [42] discussed vertical partition for the most complex object model, consisting of complex attributes with complex methods. Ezeife and Barker [48] emphasized that the network communication costs dominate query processing costs. Vertical fragmentation is discussed with reference to four different class models, consisting of simple or complex attributes combined with simple or complex methods. Fragmentation of a class is processed to group all attributes and methods of the class that are frequently accessed together by applications accessing either the class itself, its subclasses, its containing classes, or its complex method classes. For different models affinity matrixes are computed by incorporating all the object-oriented features, e.g., inheritance links and subclasses. For each of the class model a formal vertical fragmentation is presented. Method affinities of a class are calculated by summing up the frequencies of queries that access both the methods simultaneously, either directly or through this subclasses or containing classes. Actually, site information are lost while building affinity matrixes, which means that data localization is not considered. The evaluation of proposed algorithm is based on the Partition Evaluator proposed in [39], which does not really measure the total query costs.

Treating attributes and methods in a uniform and undistinguished way, Bai˜ao [4] adopted the graphical approach in [30] and [3] to object-oriented databases. The process of vertical fragmentation contains two steps, building an element affinity matrix and building an element affinity graph. However, during the process of building the element affinity matrix, data local requirement information is lost. Therefore, there is no link showing that vertical fragmentation using element affinity can improve data locality which in turn can reduce irrelevant remote data transportation costs.

2.4 Mixed Fragmentation

Navathe et al. [3] proposed a mixed fragmentation methodology for initial distributed database design. The process proposed simultaneously applies horizontal and vertical fragmentation on a relation. The input of the procedure comprises a predicate affinity table and an attribute affinity table. A set of grid cells are created first which may overlap each other. Then some grid cells are merged such that total disk accesses for all transactions can be reduced. Finally, overlap between each pair of fragments is removed using two algorithms for the cases of contained and overlapping fragments. However, the merging algorithms are based on a model which measures times of disk access (I/O). Network factors are not considered. For distributed databases, it is important to not only reduce disk access but also reduce the data transportation between sites.

Adopting some developed heuristics and algorithms in [3] to fragmentation in object oriented databases, Bai˜ao and Mattoso [27] proposed a design procedure which includes a sequence of steps: analysis phase, vertical and horizontal fragmentation. In the first step, a set of classes that are needed for horizontal fragmentation, vertical fragmentation, or non-fragmentation, are identified. In the second and third steps, vertical and horizontal fragmentations are performed on the classes identified in the first step, using algorithms extended from the one in [3]. All fragmentation algorithms are affinity based. The evaluations of the resulting fragmentation are not based on any cost model. Bai˜ao et al. [4] considered mixed fragmentation as a process of performing vertical fragmentation on classes first and then performing horizontal fragmentation on the set of vertical fragments.

2.5 Allocation

In the literature, allocation problems are first addressed for file allocation. Chu [49] presented a simple model for a non-redundant allocation of files. Casey [50] proposed a model which allows the allocation of multiple copies. Queries and updates are distinguished in the model. Mahmoud and Riodon [51] proposed a model for studying file allocation and the capacity of communication capacities to obtain optimized solution which minimize storage and communication cost. Since the early 1980s data allocation has been studied in the context of relational databases. Due to the complexity of the problem of data allocation, different researchers make different assumptions to reduce the size of the problem. Some works do not consider replication while making decision of allocation [26, 52, 53], while some others do not consider storage capabilities of network nodes [6, 54].

2.6 Summary

As shown in the above sections, most of the literature about database distribution design considers fragmentation and allocation as two different steps even though they are strongly interrelated problems which take the same input information to achieve the same objectives of improving system performance, reliability and availability. Existing approaches for primary horizontal fragmentation can be characterized into three streams, one using minterm predicates, one using predicate affinity, and a cost-driven approach using a cost model. Even though each of the approaches claims to be able to improve system performance, there is no evaluation to prove that resulting fragmentation schemata can indeed improve the system performance. Horizontal fragmentation with minterm predicates often results in a large number of fragments which will later be allocated to a limited number of network nodes. It can be expected that the number of network nodes gives the upper bound of fragments because fragments allocated at the same network node can be recombined for the benefits of most queries. Affinity-based horizontal fragmentation approaches cannot guarantee to achieve optimal system performance because the information of data local requirement is lost while computing predicate affinities. Cost-driven approaches use cost models to measure the number of disk accesses without considering transportation cost. None of the three approaches takes data local availability as the objective of fragmentation.

For vertical fragmentation there are two main approaches existing in the literature: affinity based and cost-based. The affinity-based vertical fragmentation approach originated for centralized databases with hierarchical memory levels, for which the number of disk accesses is the main factor that affects the system performance. Later, this approach was adapted to distributed databases for which transportation cost is the main cost that affects the system performance. Attribute affinities only reflect the togetherness of attributes accessed by applications. Vertical fragmentation based on affinities may reduce the number of disk accesses. However, there is no clear proof that affinity-based vertical fragmentation can indeed improve data local availability and thus improve system performance. The cost-driven approach performs vertical fragmentation based on a cost model that measures the number of disk accesses. The optimal solution chosen by this approach is the vertical fragmentation schema that have the fewest number of disk accesses.

As to allocation, due to the complexity of the allocation problem that is closely related to query optimization problem, it is infeasible to find optimal solutions. One has to seek heuristic solutions. To do this, many assumptions have been made to reduce the complexity of the problem. The assumption that fragmentation is completed properly is not reasonable. Nor it is possible to solve the fragmentation problem independently from the allocation problem because the optimal fragmentation can only be achieved with respect to the optimal allocation of fragments [8]. However, there is no fragmentation approach, for both horizontal and vertical fragmentation, taking data locality into consideration [7].

In summary, due to the deficiencies of fragmentation and allocation techniques existing in the literature, this research will study fragmentation and allocation in an integrated manner. Based on locality of data, fragmentation and fragment allocation are performed with the objective of minimizing data transmission costs and maximizing locality of data.

Chapter 3
MCRUD Matrix based Fragmentation Technique (MMF)

This chapter describes the details of our proposed fragmentation technique that can be used for fragmentation of the relations of distributed relational database and allocation of the fragments.

3.1 Initial Fragmentation

To achieve the benefits of distributed database, database designers are moving towards fragmentation of database relations or classes for allocating to the sites of distributed systems. Available techniques developed by the researchers so far to support fragmentation cannot provide solution at the initial level of a distributed system. They use frequency of queries executed in a system at runtime, affinity matrix of minterm predicates constructed from combination of predicates or attribute affinity matrix constructed based on the relationship between different attributes of a table and runtime transactions those access the attributes as a basis of fragmentation of the relations. To construct these matrices, sufficient experiential data are required those are not available in most cases at initial stage of a distributed system. So using currently available techniques for fragmentation, the database administrator has to put whole database in a single site of the system and perform fragmentation and allocation after a long period when sufficient empirical data will be available to him.

During this period facilities of distributed database cannot be enjoyed. After the period the database can be fragmented correctly to some extent and allocated to the sites with a high communication cost of transferring huge amount of data from central node to all other nodes of the system. To solve the problem of taking proper fragmentation decision at the initial stage of a distributed database, we have developed a new fragmentation technique based on precedence of attributes to increase data locality. Instead of using empirical data, we have developed Modified

Create, Read, Update and Delete (MCRUD) matrix to obtain fragmentation decisions. The details of the technique are discussed in the following sections.

3.2 CRUD Matrix

A data-to-location Create, Read, Update and Delete (CRUD) matrix is a table in which rows indicate attributes of the entities of a relation and columns indicate locations of the applications [55]. It 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 [55], [56]. Example of a traditional CRUD Matrix is shown in Table 3.1.

Table 3.1: Traditional CRUD Matrix

Entity / **Use Case**	**Order**	**Chemicals**	**Requestor**	**Vendor Catalog**
Place Order	C	R	R	R
Change Order	U, D		R	R
Manage Chemical Inventory		C, U, D		
Report on Orders	R	R	R	
Edit Requesters			C, U	

3.3 MCRUD Matrix

We have modified the existing CRUD matrix according to our requirement of horizontal fragmentation and named it Modified Create, Read, Update, and Delete (MCRUD) matrix. It is a table constructed by placing predicates of attributes of a relation in the row side and applications of the sites of a DDBMS in the column side. We have used MCRUD matrix to generate attribute locality precedence (ALP) table for each relation. Example of a MCRUD Matrix is shown in Table 3.2. In this example the distributed system has three sites and one application is running in each site. Entity set, attribute and predicate are denoted by E, *a* and *p* respectedly. If an application of a site has chances to perform create or read or update or delete operation to an attribute’s certain predicate then C or R or U or D will be written in the intersecting cell of the matrix.

Table 3.2: An MCRUD Matrix for E Relation

Site.Application / Entityset . Attribute . Predicates	$Site_1$/ Ap	$Site_2$/ Ap	$Site_3$ /Ap
$E.a_1.p_1$	CRUD	R	R
$E.a_1.p_2$	RU	CRUD	CRU
$E.a_2.p_1$	R	R	CRUD
$E.a_2.p_2$	R	RU	R
$E.a_3.p_1$	CRUD		R
$E.a_3.p_2$	R	R	CRUD

3.4 Attribute Locality Precedence (ALP)

In our technique we fragment a relation according to precedence of attributes to increase data locality. We have named it Attribute Locality Precedence (ALP). We define ALP as the value of importance of an attribute with respect to the sites of a distributed database. 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. For example in Table 3.2, there are three attributes a_1, a_2 and a_3. Among them one may be more significant than others to increase data locality and to reduce remote access in the case of fragmentation. According to the above importance we can calculate locality precedence of each attribute for each relation and construct ALP table for the relations.

3.5 ALP Table

ALP values of different attributes of a relation are placed in a table called ALP table. ALP table is constructed by database designer for each relation of a DDBMS at the time of designing the database with the help of MCRUD matrix and cost functions given in the following section 3.6. The algorithm that is used to calculate ALP and to construct ALP table is given in Algorithm I. An example of ALP table for the MCRUD matrix of Table 3.2 is shown in Table 3.3. Details of how the precedence values of attributes of Table 3.3 are calculated can be found in section 3.6.

Algorithm I: ALP calculation

Input: MCRUD of a relation

Output: ALP table of the 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] = $f_c$*C + $f_r$*R + $f_u$*U + $f_d$*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
                        MAX[i][j] = S[i][j][k];     predicate j is maximum*/
                        POS[i][j] = k;
            SumOther = 0;
            Count =0;
            for ( r =1; r <= A[i][j][k]; r++){
                If (r!=k){
                        SumOther + = S[i][j][r];
                        If (S[i][j][r]>MAX[i][j]/2)      /* selecting the sites where
                            Replicate[Count]=r;            replication of a fragment
                            Count++;                        will be performed */
        ALPsingle[i][j] = MAX[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
        ALP[i] + = ALPsingle[i][j];                       (locality precedence)*/
```

Table 3.3: An ALP Table

EntitySet. Attribute Name	**Precedence**
E . a_1	4
E . a_2	8
E . a_3	13

3.6 ALP Cost Functions

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:

MCRUD cell cost:

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

Here C, R, U, D denotes cost incurred for performing create, read, update and delete operation in the system and f_C, f_R, f_U, f_D are the frequencies of create, read, update and delete operation performed by an application of a site.

$C_{i,j,k,r}$ = cost of predicate j on attribute i accessed by application r at site k

Site cost:

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

$S_{i,j,k}$ = sum of all applications' cost of predicate j of attribute i at site k.

For all sites in the distributed system, cost incurred by the applications of a site is summed.

Maximum-Site cost:

$S_{i,j,m} = \text{Max}(S_{i,j,k})$ ---------- (3)

$S_{i,j,m}$ = maximum cost among the sites for predicate j of attribute i.

For a particular predicate of an attribute, maximum cost among all sites is calculated in $S_{i,j,m}$

Predicate ALP Cost:

$ALP_{ij} = S_{i,j,m} - \sum_{k \neq m}^{A_{ijk}} S_{i,j,k}$ --------- (4)

ALP_{ij} = ALP cost for predicate j of attribute i.

Predicate ALP cost can be found by deducting all costs incurred in the sites from the site where cost of accessing the predicate is maximum.

Attribute ALP Cost:

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

ALP_i = Total ALP cost of attribute i (locality precedence)

An attribute's ALP cost is the summation of ALP costs of its predicates.

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. So initially we have assumed that f_C, f_R, f_U and f_D=1. Typical cost of update operation is more than create and delete operation in a database system and reading from database always incurs least cost. So for simplicity we treated C=2, R=1, U=3 and D=2; Cost incurred for performing create, read, update and delete operations. Justifications of these assumptions can be found in [58].

Frequency Estimation for Cost Functions:

Using the information gathered as in Table 3.4, more accurate estimation of frequency of create, read, update and delete operation by an application can be possible. This form can be used at the requirement analysis phase of a DDBMS design.

Table 3.4: Information Need Analysis Form

Access Statistics / **Users**	**Site k**			
	Application r			
	attribute$_i$. predicate$_j$			
	Create	**Read**	**Update**	**Delete**
U_1		x		
U_2		x	x	
U_3	x	x	x	x
U_4		x		
. . .				
U_n	x	x	x	

3.7 Fragmentation based on MCRUD Matrix

In this section we have described MMF technique in details. The main functionalities of the technique are shown in Fig. 3.1. There are *n* number of relations in the database named R_1, R_2,…, R_n. First n number of MCRUD matrices is constructed by the system designer at design time. These *n* matrices will be the input of our technique. Then using the cost functions of Section 3.6, *n* number of ALP tables ALP (R_1), ALP (R_2), …, ALP (R_n) will be constructed. Then in the next step, *n* number of predicate sets named P_1, P_2, …, P_n will be generated for attributes with highest ALP value for each ALP table. Each predicate set P_i will contain *m* number of predicates. According to the predicate sets, each of the *n* relations R_i is fragmented into *m* fragments and allocate to *m* sites.

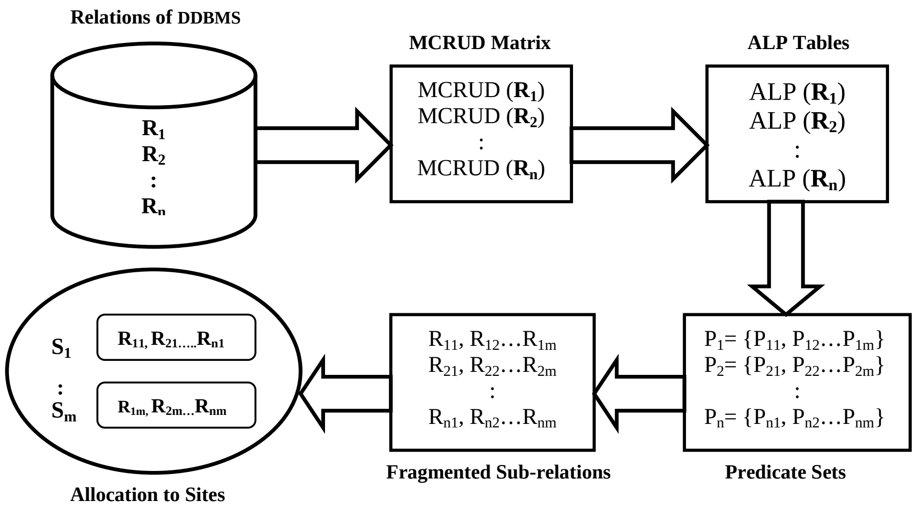

Fig. 3.1: Block Diagram of MMF Technique

Following algorithm, Algorithm II has been used to implement MMF technique.

Algorithm II: FragmentationAllocation

```
Input: Total number of sites: $S = \{S_1, S_2,\dots ,S_n\}$
        Relation to be fragmented: R
        Modified CRUD matrix: MCRUD[R]
Output: Fragments $F = \{F_1, F_2, F_3,\dots, Fn\}$
Step 1: Construct ALP[R] from MCRUD[R] based on Cost functions
Step 2: For the significant highest valued attribute of ALP table
        a. Generate predicate set $P=\{ P_1, P_2, \dots ,P_m \}$
        b. Fragment R using P as selection predicate $\forall_p \mid \sigma_p(R)$
        c. ALLOCATE F to S
Step 3: For non-significant-highest-value (Max-Highest<1.5*2nd-Highest) in ALP[R]
        a. REPLICATE R to $\sum_{j=1}^{n} S_j$ if R is an entity set
        b. Derive Horizontally Fragment R using its owner relation if R is a relationship set
```

Algorithm II takes a relation to be fragmented, MCRUD matrix of the relation and number of allocation sites as input. It finally produces the fragments and allocates them in the sites of DDBMS. Working procedure of the algorithm will be clear in the following subsections.

Now we are presenting a real life example to explain the steps shown in Fig. 3.1. Let we are designing a distributed banking database system (DBDS). One of the relations of this database is Accounts shown in Table 3.5. Initially number of allocation sites of the distributed system is three namely Dhaka, Chittagong and Khulna as shown in Fig. 3.2.

Table 3.5: Accounts Relation

AccountNo	Type	CustId	OpenDate	Balance	BrName
01	Ind	001	20/1/09	12500	Dhk
02	Cor	002	23/1/09	35000	Dhk
03	Cor	003	28/2/09	5200	Ctg
04	Ind	004	25/3/09	15000	Khl
05	Cor	005	17/4/09	50000	Dhk

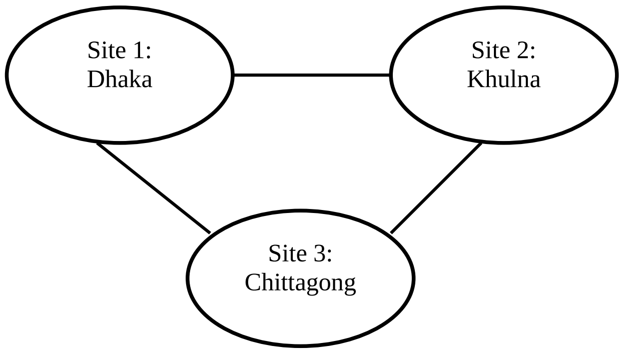

Fig. 3.2: Distributed Banking Database System

3.7.1 MCRUD Construction

The designer will construct the MCRUD matrix for the Accounts Database relation in the requirement analysis phase. An example of MCRUD matrix is shown in Table 3.6. It can be seen from the figure that predicates of the attributes of Accounts relation are shown in row sides and applications of different sites are placed in column sides. Here Ap_1: Application deals with Customer information, Ap_2: Application deals with Accounts information, Ap_3: Application deals with Loan information. Relationship between predicates and applications are represented in their intersecting cells. For example attribute *Type* has two predicates: Ind (Individual) and Cor (Corporate). Ap_2 (Application 2) running in $Site_1$ can performs read and update operation over *Ind* predicate. So *R U* is placed in the intersecting cell. In this way the whole matrix is filled up at requirement analysis phase.

Table 3.6: MCRUD Matrix of Accounts Relation

Site.Application / Entity.Attribute.Predicates	Site1			Site2			Site3		
	Ap1	Ap2	Ap3	Ap1	Ap2	Ap3	Ap1	Ap2	Ap3
Accounts .AccountNo<10000	C		RU						R
Accounts .AccountNo>=10000		R							
Accounts.Type=Ind	CRD	RU	RUD		R				
Accounts.Type=Cor		RU	R				CRUD	RU	R
. .									
Accounts.Balance<50000	R		R			CRUD			R
Accounts.Balance>=50000		CR							
Accounts.BrName=Dhk	CRUD	RU	CRUD			R	R		
Accounts.BrName=Ctg		R		CRUD	CRUD	R		R	
Accounts.BrName=Khl							CRUD	RD	CRU

3.7.2 ALP Calculation

The designer will calculate precedence of locality of each attribute from the MCRUD matrix of Accounts relation according to the cost functions given in section 3.6. Algorithm II represents the algorithm for ALP calculation. If a predicate of an attribute is accessed by more than one site, its precedence is decreased and also overall precedence of the attribute where the predicate belongs t, decreases in our cost function because it reduces data locality. To calculate precedence of a predicate, predicate access values of other sites are deducted from the site where value of accessing that predicate is maximum.

Finding the ALP of the attribute BrName is shown in Table 3.7- 3.9.

Table 3.7: MCRUD Matrix for ALP Calculation (predicate: BrName=Dhk)

Site.Application / Entity.Attribute.Predicates	Site1			Site2			Site3		
	Ap1	Ap2	Ap3	Ap1	Ap2	Ap3	Ap1	Ap2	Ap3
Accounts .AccountNo<10000	C		RU						R
Accounts .AccountNo>=10000		R							
Accounts.Type=Ind	CRD	RU	RUD		R				
Accounts.Type=Cor		RU	R				CRUD	RU	R
.									
Accounts.Balance<50000	R		R			CRUD			R
Accounts.Balance>=50000		CR							
Accounts.BrName=Dhk	CRUD	RU	CRUD			R	R		
ccounts.BrName=Ctg		R		CRUD	CRUD	R		R	
Accounts.BrName=Khl							CRUD	RD	CRU

Table 3.8: MCRUD Matrix for ALP Calculation (predicate: BrName=Dhk)

Site.Application / Entity.Attribute.Predicates	Site1			Site2			Site3		
	Ap1	**Ap2**	**Ap3**	**Ap1**	**Ap2**	**Ap3**	**Ap1**	**Ap2**	**Ap3**
Accounts.AccountNo<10000	**C**		**RU**						**R**
Accounts.AccountNo>=10000		**R**							
Accounts.Type=Ind	**CRD**	**RU**	**RUD**		**R**				
Accounts.Type=Cor		**RU**	**R**				**CRUD**	**RU**	**R**
.									
Accounts.Balance<50000	**R**		**R**			**CRUD**			**R**
Accounts.Balance>=50000		**CR**							
Accounts.BrName=Dhk	**CRUD**	**RU**	**CRUD**			**R**	**R**		
Accounts.BrName=Ctg		**R**		**CRUD**	**CRUD**	**R**		**R**	
Accounts.BrName=Khl							**CRUD**	**RD**	**CRU**

Table 3.9: MCRUD Matrix for ALP Calculation (predicate: BrName=Dhk)

Site.Application / Entity.Attribute.Predicates	Site1			Site2			Site3		
	Ap1	**Ap2**	**Ap3**	**Ap1**	**Ap2**	**Ap3**	**Ap1**	**Ap2**	**Ap3**
Accounts.AccountNo<10000	**C**		**RU**						**R**
Accounts.AccountNo>=10000		**R**							
Accounts.Type=Ind	**CRD**	**RU**	**RUD**		**R**				
Accounts.Type=Cor		**RU**	**R**				**CRUD**	**RU**	**R**
.									
Accounts.Balance<50000	**R**		**R**			**CRUD**			**R**
Accounts.Balance>=50000		**CR**							
Accounts.BrName=Dhk	**CRUD**	**RU**	**CRUD**			**R**	**R**		
Accounts.BrName=Ctg		**R**		**CRUD**	**CRUD**	**R**		**R**	
Accounts.BrName=Khl							**CRUD**	**RD**	**CRU**

According to the cost functions, values of the predicates are as follows:

BrName=Dhk: [{(2+1+3+2) + (1+3) + (2+1+3+2)} - {1+1}] = 18, (Table 3.7)

BrName=Ctg is [{(2+1+3+2) + (2+1+3+2) + (1)} - {1+1}] = 15 (Table 3.8) and

BrName=Khl is [{(2+1+3+2) + (1+2) + (2+1+3)} - {0+0}] = 17. (Table 3.9)

So ALP of BrName = (18+15+1) = 50.

3.7.3 ALP Table Construction

ALP values of all the attributes of the Accounts relation are computed from its MCRUD matrix according to the previous sections. Table 3.10 shows the ALP table for Accounts relation. The attribute with highest precedence value is treated as most important attribute for fragmentation.

Table 3.10: ALP Table of Accounts Relation

Attribute Name	**Precedence**
AccountNo	6
Type	22
CustId	6
OpenDate	7
Balance	10
BrName	50

3.7.4 Predicate Set Generation

Given a relation R (A_1, A_2,…, A_n) where attribute A_i has domain D_i, a predicate p_j defined on R has the form

p_j: A_i θ *Value* Where $\theta \in \{=,<,\neq,\leq,>,\geq\}$ and value $\in D_i$.

Predicate set of an attribute is the set of predicates used by the applications to access that attribute. In MMF technique, only the predicate set for the attribute with highest ALP value of a relation will be generated. We can see from ALP table of Accounts relation (Table 3.10) that BrName attribute has highest ALP value. So predicate set of BrName attribute is generated which is found in MCRUD matrix of Accounts relation.

P= {p_1: BrName=Dhk, p_2: BrName=Ctg, p_3: BrName= Khl}

3.7.5 Fragmentation of Relation

A horizontal fragmentation consists of a subset of the tuples of a relation. It is defined using Selection operation of relational algebra: σ_p(R). Here *p* is the predicate using which records are selected from relation R those satisfied *p*. So for each *p* of predicate set P, a set of records can be selected and treated as a fragment.

According to the predicates of predicate set P which was generated in the previous subsection, the Account relation will be fragmented horizontally. As P consists of three predicates so three horizontal fragments were created.

Allocation is the process of copying a fragment to a site of a system according to certain cost function. Here three fragment generated earlier is allocated to three sites shown in Table 3.11 – 3.13 according to predicate access in respected sites.

Table 3.11: Part of Accounts Relation Allocated to Site 1

AccountNo	Type	CustId	OpenDate	Balance	BrName
01	Ind	001	20/1/09	12500	Dhk
02	Cor	002	23/1/09	35000	Dhk
05	Cor	005	17/4/09	50000	Dhk

Table 3.12: Part of Accounts Relation Allocated to Site 2

AccountNo	Type	CustId	OpenDate	Balance	BrName
04	Ind	004	25/3/09	15000	Khl

Table 3.13: Part of Accounts Relation Allocated to Site 3

AccountNo	Type	CustId	OpenDate	Balance	BrName
03	Cor	003	28/2/09	5200	Ctg

3.8 Analysis of MMF technique

The space complexity and time complexity of MCRUD Matrix based fragmentation technique have been discussed in the following sections.

3.8.1 Memory Cost Analysis

To store the MCRUD matrices of different relations in the system for further processing by Algorithm II, we have used four dimensional arrays. As discussed in Section 3.5, for calculating ALP values we have to store four things: site number, application number, attribute number and predicate number. These values will be stored in 4D arrays and ALP tables are constructed using these arrays. Cost of each cell of MCRUD matrix is computed by the cost function of equation 1.

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

Where $C_{i,\,j,\,k,\,r}$ = cost of predicate j on attribute i accessed by application r at site k.

This is represented in the Algorithm I as follows:

$C[i][j][k][r] = f_c{*}C + f_r{*}R + f_u{*}U + f_d{*}D$,

Where i: attribute number, j: predicate number, k: site number and r: application number. In this thesis number of sites is denoted by S, number of applications is denoted by N, number of Predicates is denoted by P and number of attributes is denoted A. So space requirement to store an MCRUD matrix will be O(S*N*A*P) or $O(n^4)$ if $S \approx N \approx A \approx P \approx n$.

For example, if for a MCRUD matrix there are 12 attributes, 5 predicates for each attribute, 10 sites and 3 applications running at each site that is A=12, P=5, S=10 and N=3 and also to store a real number if it requires 4 bytes of memory then memory requirement of this MCRUD matrix is: 12*5*10*3*4= 7200 Bytes or 7.2KB. If there are 200 tables in the database then total memory requirement is 7.2*200=1440KB or 1.44MB.

ALP table for a relation constructed from 4D array is stored in 2D array where attribute name and its ALP value is placed. For this example, A=12 and if maximum length of an attribute name is 50 characters then ALP table consumes 12*50*4=2400Byte or 2.4 KB. So for 200 database tables, it takes 200*2.4=480 Kilo Bytes or 0.48MB total memory to store all the ALP tables. So overall memory requirement is 1.44+0.48=1.92MB, approximately 2MB. Un-doubtfully this is quite less in comparison with huge memory requirement of several gigabytes in distributed database.

3.8.2 Computational Cost Analysis

Creation of MCRUD matrix for every relation and calculation of ALP from each matrix adds some additional cost in our system.

For calculating ALP, Algorithm I is used. We can find from Algorithm I that, its maximum computational cost is dominated by the computation within four nested loops. The code is as follows:

```
for ( i =1; I <= TotalAttributes; i++){
    for ( j =1; j <= TotalPredicates[i]; j++){
        for ( k =1; k <= TotalSites; k++){
            for ( r =1; r <= TotalApplications[k]; r++)
                C[i][j][k][r] = $f_c$*C + $f_r$*R + $f_u$*U + $f_d$*D
                S[i][j][k] + = C[i][j][k][r]
```

So computational order of this algorithm is O (i*j*k*r). As i_{max} =A, j_{max}=P, k_{max}=S and r_{max}=N, so we can rewrite the order as O(A*P*S*N) or O (n^4) if we treat S≈ N≈ A ≈P ≈ n.

Actual problem of horizontal fragmentation and allocation is O (k^{2^n}) where there are *n* simple predicates and *k* sites because all the combinations have to be generated to find a correct solution. This is impossible in practical large database systems [7].

We have reduced it to O (n^4) by providing solution based on heuristic that use MCRUD matrix.

If we consider the case of Section 3.8.1 that is A=12, P=5, S=10 and N=3 where i: attribute number, j: predicate number, k: site number and r: application number then to compute ALP cost of the cells 12*5*10*3 =1800 * 4 multiplication =7200 multiplication operation and 1800*3=5400 addition operation is required.

But this computational cost can be ignored because ALP calculation from MCRUD matrix will be performed *offline* during the requirement analysis phase of distributed database development. So this computation will not affect negatively on the performance of the system.

3.9 Scalability of MMF Technique

We have investigated some cases to check whether our technique is restricted to some particular predicate numbers, attribute number and site numbers or it is a scalable enough that is not restricted to certain number of predicates, attributes or sites. In the following subsections it can be seen that MMF technique is a generalized technique which can be applied in any distributed system.

3.9.1 Relation between Number of Sites with Number of Predicates

In practice, there may be three cases: number of sites (S) less than number of predicates (P), number of sites (S) equals to number of predicates(P) , number of sites (S) greater than number of predicates(P). It can be recalled that according to the number of predicates of the attribute with highest ALP value, a relation can be fragmented into same number of sub-relations. From the following sub-sections it is be clear that MMF is neither restricted to certain number of predicates nor fixed number of sites.

Case 1: S < P

If S is less than P, in this case total fragments will be more than total sites. So fragments will be assigned to corresponding sites where locality precedence of predicates is highest respectively. It can be clearly understood that in this case some sites will get more than one fragments. For example, in Table 3.14, P=3 namely LnType=SME, LnType=HOME and LnType=CAR of the attribute LnType with highest locality precedence. So three horizontal fragments are created taking above predicates as selection predicates. But S=2 namely Site1 and Site2. So a fragment is allocated to a site where precedence of the predicate use to make the fragment is maximum. Here $Site_1$ got two fragments and $Site_2$ got one fragment.

Table 3.14: Decision Table when S<P

Attribute Name	Predicates	Precedence in Site 1	Precedence in Site 2	Decision
LnType	LnType = SME	5	**13**	Fragment in Site 2
	LnType = HOME	**16**	5	Fragment in Site 1
	LnType = CAR	**9**	5	Fragment in Site 1

Case 2: S = P

This is a straight forward case. Here fragments are assigned to corresponding sites where locality precedence of the site is maximum respectively. For example, in Table 3.15, S=3 and P=3, we can see there are three predicates namely LnType=SME, LnType=HOME and LnType=CAR of the attribute LnType with highest locality precedence. So three horizontal fragments will be created taking above predicates as selection predicates. Number of allocation sites are also three namely $Site_1$, $Site_2$ and $Site_3$. So a fragment will be allocated to a site where precedence of the predicate use to make the fragment is maximum.

Table 3.15: Decision Table when S=P

Attribute Name	Predicates	Precedence in Site 1	Precedence in Site 2	Precedence in Site 3	Decision
LnType	LnType = SME	5	**13**	5	Fragment in Site 2
	LnType = HOME	**16**	5	5	Fragment in Site 1
	LnType = CAR	5	5	**12**	Fragment in Site 3

Case 3: S > P

In this case, fragments are assigned to corresponding sites where locality precedence of the predicates by which fragments are created is maximum respectively and the sites where no fragment is allocated initially, having replica of a fragment whose predicate precedence value is maximum in the sites. Replication is for reducing remote access cost of the queries. For example when S=3 and P=2, in Table 3.16 below we can see there are two predicates namely LnType=SME, and LnType=CAR of the attribute LnType with highest locality precedence. So two horizontal fragments are created taking above predicates as selection predicates. But number of allocation sites are three namely $Site_1$, $Site_2$ and $Site_3$. So two fragments will be allocated to $Site_2$ and $Site_3$ where precedence of the predicates use to make the fragments are maximum respectively and the $Site_1$ will have a replica of that fragment whose predicate precedence is highest in $Site_1$.

Table 3.16: Decision Table when S>P

Attribute Name	Predicates	Precedence in Site 1	Precedence in Site 2	Precedence in Site 3	Decision
LnType	LnType=SME	5	**13**	5	Fragment in Site 2
	LnType=CAR	**10**	5	**12**	Fragment in Site 3 Replica in Site 1

3.9.2 Impact of Schema Change

In MMF, relations of a distributed system are fragmented based on their respected MCRUD matrices. MCRUD matrix of a relation is constructed at system design time. If schema of a relation changes during the design phase or later on e.g. by addition of attributes then only the MCRUD matrix corresponding to the relation have to be reconstructed and then the relation can be fragmented from its ALP table generated from its reconstructed MCRUD matrix. So it can be understood that our technique is not restricted to a particular DDBMS or relations with particular schema.

3.9.2.1 Normalization

If a relation is split into two relations by the database designer after fragmentation process for normalization issue then the ALP table of the whole relation has to split into two ALP tables. If an attribute is present in one relation, its ALP value will be placed in respected ALP table. So no new MCRUD matrix has to be constructed.

Algorithm III: Splitting of ALP Tables

Input: Set of attributes for each of the normalized relations, Previous ALP Table
Output: Fragments for each of the normalized relations
Steps:

```
1. Input set of attributes each of the normalized relations
2. Divide ALP table into two ALP tables according to set of attributes
3. For each new ALP table:
      a. Generate Predicate Set for Highest precedence attribute
      b. Fragment according to generated Predicate Set
      c. Allocate fragments to Previous and new sites
```

3.9.2.2 De-Normalization

If two relations are merged into one relation by the database designer after fragmentation process for de-normalization issue then both the ALP tables of the relations can be merged into one ALP table. So no new MCRUD matrix has to be constructed.

Algorithm IV: Combining ALP Tables

Input: ALP tables of two relations selected for De-Normalization
Output: Fragments for De-Normalized relations
Steps:

```
1. Input two ALP tables
2. Merge the rows of two tables into one table
3. For highest precedence attribute of merged ALP table:
      a. Generate Predicate Set
      b. Fragment according to generated Predicate Set
      c. Allocate fragments to Previous and new sites
```

3.9.3 Impact of Number of Site Increase

As in MMF there is no restriction of total number of sites in a DDBMS, so our technique can be applicable for a distributed system with any number of sites. Sites of a DDBMS are placed in column side of a MCRUD matrix. If number of sites increased, corresponding columns will be added to each MCRUD matrix for every relation. This is analyzed in section 4.7.The steps of site increment is shown in Algorithm V below:

Algorithm V: Site Increment

```
Input: Output: MCRUD Matrix with additional site information
Output: Fragments for updated site information
Steps:
    4. Input Modified MCRUD matrix with new sites
    5. Calculate ALP table for Modified MCRUD matrix
    6. Generate Predicate Set for Highest precedence attribute
    7. Fragment according to generated Predicate Set
    8. Allocate fragments to Previous and new sites
```

3.9.4 Rearranging based on Empirical Data

After certain duration of database execution in the sites of a distributed system when enough empirical data of query execution, attribute access by transaction etc. are available, the MCRUD matrix of the relations can be modified based on those data. It will improve the hit rate (locality of access) of the system to certain extent at the price of data transfer cost among the sites of the distributed system. The process is shown in the following algorithm, Algorithm VI:

Algorithm VI: Re-Fragmentation

```
Input: Total number of sites: $S = \{S_1, S_2, \dots ,S_n\}$
        Relation to be fragmented: R
        Modified MCRUD matrix based on empirical data: MCRUD[R]
Output: Fragments $F = \{F_1, F_2, F_3, \dots, Fn\}$
Step 1: Construct ALP[R] from MCRUD[R] based on Cost functions
Step 2: For the significant highest valued attribute of ALP table
        a. Generate predicate set $P=\{ P_1, P_2, \dots ,P_m \}$
        b. Fragment R using P as selection predicate $\forall_p \mid \sigma_p(R)$
        c. ALLOCATE F to S
Step 3: For non-significant-highest-value (Max-Highest<1.5*2nd-Highest) in ALP[R]
        a. REPLICATE R to $\sum_{j=1}^{n} S_j$ if R is an entity set
        b. Derive Horizontally Fragment R using its owner relation if R is a relationship set
```

3.9.5 Implementation of other Fragmentation Types

In this thesis we have performed the fragmentation of the relations of distributed database using horizontal fragmentation technique. This is because of improving performance significantly of a distributed database, we have to maximize locality of data or hit rate of the queries. That is query generating in one site access data of that site only. This will reduce remote access cost and cost of data transfer among the sites. Locality of data can be achieved more using horizontal fragmentation than vertical fragmentation.

MMF technique is not limited to horizontal fragmentation only. If we slightly modify the MCRUD matrix that is if we place attributes of a relation in the row side and applications of the sites of a DDBMS in the column side and modifying the cost functions we can produce vertical fragmentation using MMF technique. Modification of MCRUD matrix for vertical fragmentation is shown in Table 3.17:

Table 3.17: MCRUD Matrix for Vertical Fragmentation

Site.Application / Entity.Attribute	Site1			Site2			Site3		
	Ap1	Ap2	Ap3	Ap1	Ap2	Ap3	Ap1	Ap2	Ap3
Accounts.AccountNo	C		RU						R
Accounts.Type	CRD	RU	RUD		R				
. .									
Accounts.Balance	R		R			CRUD			R
Accounts.BrName	CRUD	RU	CRUD			R	R		

Like other Hybrid or Mixed fragmentation techniques, MF can be performed in our MMF technique by applying vertical fragmentation followed by horizontal fragmentation or vice versa. It is worth mentioning that MF is only applied in distributed databases if the relations have too many attributes and huge number of records in the relations.

3.10 Summary

In this chapter we have presented a model for our proposed MMF technique of fragmentation of distributed database relations. Algorithms of ALP table construction and fragmentation are also presented and analyzed in details. Scalability of our technique has also analyzed from different perspectives.

Chapter 4

Results and Discussion

The objective of the experimental work is to verify the applicability and feasibility of MMF, the proposed fragmentation technique based on MCRUD matrix. The experimental evaluation has been performed with synthetic data and reasonable number of queries.

4.1 Experimental Environment

To justify our technique we have implemented a distributed banking database system in the post-graduate lab of BUET using DELL computers with Core-two Duo 2.80 processors and 2GB RAM. We have used Windows XP operating system and Oracle 10g for database creation. Entity Relationship Diagram of our implemented database namely Distributed Banking Database System (DBDS) is shown in Fig. 4.1.

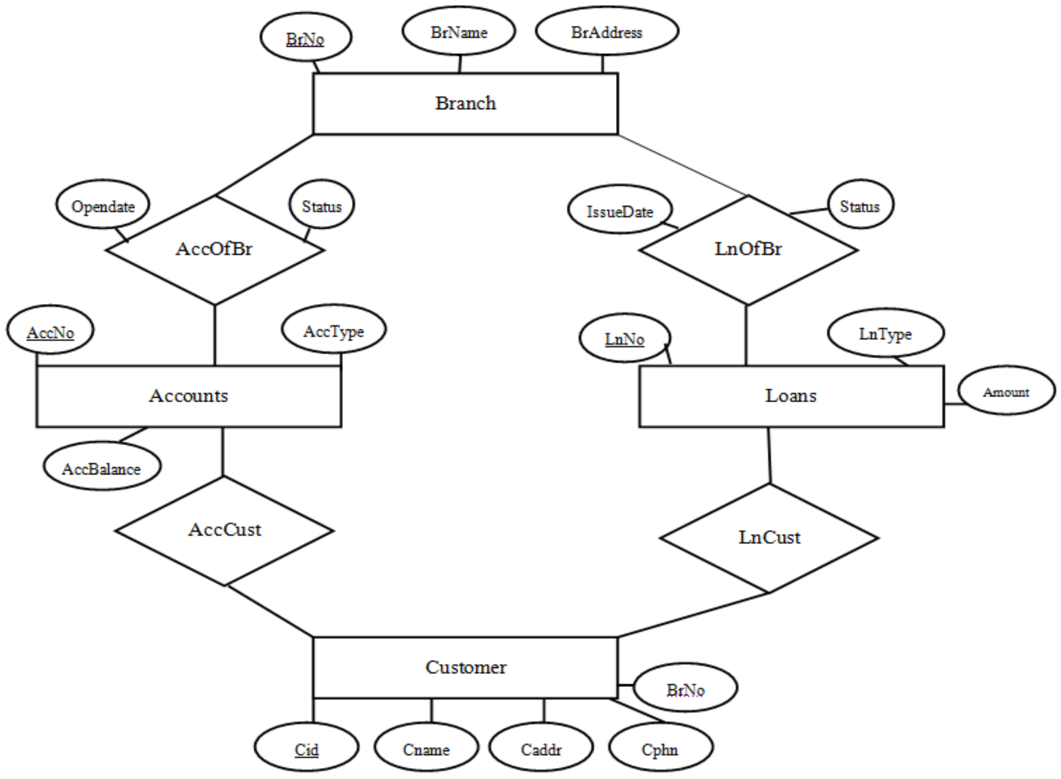

Fig 4.1: ER diagram of DBDS

The transformation of E-R schema of Figure 4.1 into relational schema is as follows:

Customer-Schema = (Cid, Cname, Caddr, Cphn, BrNo)

Loans-Schema = (LnNo, LnType, Amount)

Accounts-Schema = (AccNo, AccType, AccBalance)

Branch-Schema = (BrNo, BrName, BrAddress)

LnCust-Schema = (LnNo, Cid)

AccCust-Schema = (AccNo, Cid)

AccofBranch-Schema = (AccNo, Opendate, Status, BrNo)

LnofBranch-Schema = (LnNo, Issuedate, Status, BrNo)

Fig 4.2: Relation Schema of E-R Diagram of Fig. 4.1

Initially number of sites of the distributed system is three as shown in Fig. 4.2. In each site, three applications were executed.

Application 1 deals with Customer related information.

Application 2 deals with Account related information.

Application 3 deals with Loan related information.

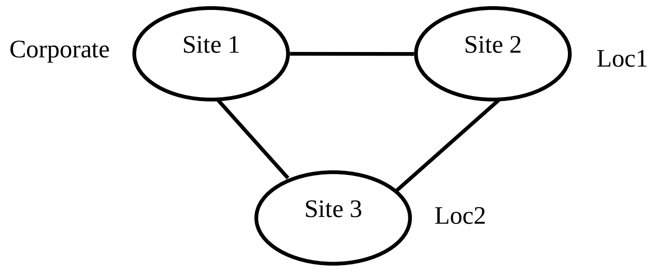

Fig 4.3: Distributed Banking Database System Network

4.2 Construction of MCRUD Matrix

We have constructed the MCRUD matrix for each of the eight relations in the requirement analysis phase. An MCRUD matrix is constructed by placing predicates of attributes of a relation in the row side and applications of the sites of a DDBMS in the column side of a table in the requirement analysis phase of system development. The matrices constructed for all the relations of Fig. 4.2 are shown in Table 4.1 - 4.8.

Table 4.1: MCRUD Matrix of Branch relation

Site.Application	Site1			Site2			Site3		
Entity.Attribute.Predicates	Ap1	Ap2	Ap3	Ap1	Ap2	Ap3	Ap1	Ap2	Ap3
Branch.BrNo=B01	R	R	R					R	
Branch.BrNo=B02			R	R		R			
Branch.BrNo=B03							R		R
Branch.BrName=Corporate	R	R							
Branch.BrName=Loc1				R	R			R	
Branch.BrName=Loc2	R			R			R	R	
Branch.BrAddress=?			R						

Table 4.2: MCRUD Matrix of Loan relation

Site.Application	Site1			Site2			Site3		
Entity.Attribute.Predicates	Ap1	Ap2	Ap3	Ap1	Ap2	Ap3	Ap1	Ap2	Ap3
Loan .LnNo<10000	RU	R	CRUD	RU	R	CRUD	R	R	CRUD
Loan .LnNo>=10000	R	R	CRUD	R	RU	CRUD	R	RU	CRUD
Loan.LnType=SME	R		RU	RU	R	CRUD	R		RU
Loan.LnType=HOME	RU	RU	CRUD	R		RU	R		RU
Loan.LnType=CAR	R		RU	R		RU	RU		CRUD
Loan.Amount<50000	R		CRUD	R		CRUD	R		CRUD
Loan.Amount=50000:100000	R	R	CRUD	R		CRUD	R		CRUD
Loan.Amount>100000	R		CRUD	R		CRUD	R		CRUD

Table 4.3: MCRUD Matrix of Customer relation

Site.Application	Site1			Site2			Site3		
Entity.Attribute.Predicates	Ap1	Ap2	Ap3	Ap1	Ap2	Ap3	Ap1	Ap2	Ap3
Customer.Cid<10000	CRUD	R	R	CRUD	R	R	CRUD	R	R
Customer. Cid >=10000	CRUD	R	R	CRUD	R	R	CRUD	R	R
Customer.Cname=?	CRUD	R	R	CRUD	R	R	CRUD	R	R
Customer.Cphn=?	CRUD	R	R	CRUD	R	R	CRUD	R	R
Customer.Caddr=?	CRUD	R	R	CRUD	R	RU	CRUD	R	R
Customer. BrNo=B01	CRUD	R	R	RU	R	R	RU		
Customer. BrNo=B02	RU			CRUD	R	R	RU		
Customer. BrNo=B03	RU			RU			CRUD	R	R

Table 4.4: MCRUD Matrix of Accounts relation

Site.Application	Site1			Site2			Site3		
Entity.Attribute.Predicates	Ap1	Ap2	Ap3	Ap1	Ap2	Ap3	Ap1	Ap2	Ap3
Accounts .AccNo<10000	RU	CRUD	R	RU	CRUD	RU	R	CRUD	R
Accounts .AccNo>=10000	R	CRUD	R	R	CRUD	RU	R	CRUD	R
Accounts.AccType=Ind	R	RU	R	RU	CRUD	RU	RU	CRUD	R
Accounts.AccType=Cor	RU	CRUD	RU	R	RU		R	RU	
Accounts.AccBalance<50000		CRUD	R		CRUD	R		CRUD	R
Accounts.AccBalance=50000:100000	R	CRUD	R		CRUD	R		CRUD	R
Accounts.AccBalance>100000		CRUD	R		CRUD	R		CRUD	R

Table 4.5: MCRUD Matrix of AccofBranch relation

Site.Application	Site1			Site2			Site3		
Entity.Attribute.Predicates	Ap1	Ap2	Ap3	Ap1	Ap2	Ap3	Ap1	Ap2	Ap3
AccofBranch.AccNo<10000		CRUD			CRUD			CRUD	
AccofBranch.AccNo>=10000		CRUD			CRUD			CRUD	
AccofBranch.OpenDate=?		CRUD			CRUD			CRUD	
AccofBranch.Status=A		CRUD			CRUD			CRUD	
AccofBranch.Status=I		CRUD			CRUD			CRUD	
AccofBranch. BrNo=B01		CRUD							
AccofBranch. BrNo=B02					CRUD				
AccofBranch. BrNo=B03								CRUD	

Table 4.6: MCRUD Matrix of LnofBranch relation

Site.Application	Site1			Site2			Site3		
Entity.Attribute.Predicates	Ap1	Ap2	Ap3	Ap1	Ap2	Ap3	Ap1	Ap2	Ap3
LnofBranch.LnNo<10000			CRUD			CRUD			CRUD
LnofBranch.BrNo>=10000			CRUD			CRUD			CRUD
LnofBranch.IssueDate=?			CRUD			CRUD			CRUD
LnofBranch.Status=R			CRUD			CRUD			CRUD
LnofBranch.Status=D			CRUD			CRUD			CRUD
LnofBranch. BrNo=B01			CRUD						
LnofBranch. BrNo=B02						CRUD			
LnofBranch. BrNo=B03									CRUD

Table 4.7: MCRUD Matrix of AccCust relation

Site.Application / Entity.Attribute.Predicates	Site1			Site2			Site3		
	Ap1	Ap2	Ap3	Ap1	Ap2	Ap3	Ap1	Ap2	Ap3
AccCust.AccNo=?		CRUD			CRUD			CRUD	
AccCust.Cid=?	CRUD			CRUD			CRUD		

Table 4.8: MCRUD Matrix of LnCust relation

Site.Application / Entity.Attribute.Predicates	Site1			Site2			Site3		
	Ap1	Ap2	Ap3	Ap1	Ap2	Ap3	Ap1	Ap2	Ap3
LnCust.AccNo=?	R		CRUD			CRUD			CRUD
LnCust.Cid=?	CRUD			CRUD			CRUD		

4.3 Calculation of ALP Values and Construction of ALP Tables

We have calculated locality precedence of each attribute from the MCRUD matrix of each relation using attribute locality precedence (ALP) calculation algorithm. Using the ALP values we have constructed ALP table for each relation. ALP table is a 2D array where attributes of a relation and its locality precedence is stored. For each attribute, Create, Read, Update and Delete operation over its predicates from different applications of different sites is calculated and sum up to have locality precedence of that attribute. Details ALP calculation and ALP table construction can be found in section 3.4.2 and 3.4.3. Attribute with highest precedence implies that taking predicates of this attribute as selection predicate for horizontal fragmentation will maximize the hit ratio. It is depicted in Table 4.9.

Table 4.9: Precedence Calculation for LnType Attribute of Loan Relation

Attribute Name	Predicates	Precedence in Site 1	Precedence in Site 2	Precedence in Site 3	Precedence of Predicate	ALP	Decision
LnType	LnType=SME	5	**13**	5	13-5-5=3	3+6+2=11	Fragment in Site 2
	LnType= HOME	**16**	5	5	16-5-5=6		Fragment in Site 1
	LnType=CAR	5	5	**12**	12-5-5=2		Fragment in Site 3

ALP tables of for all the relations of Fig. 4.2 are shown in Table 4.10 – 4.17.

Table 4.10: ALP Table of Loan

Attribute Name	Precedence
LnNo	-20
LnType	11
LnAmount	-26

Table 4.11: ALP Table of Branch

Attribute Name	Precedence
BrNo	5
BrName	3
BrAddress	1

Table 4.12: ALP Table of Customer

Attribute Name	Precedence
Cid	-20
Cname	-10
Cphn	-7
Caddr	-10
BrNo	6

Table 4.13: ALP Table of Accounts

Attribute Name	Precedence
AccNo	-14
AccType	3
AccBalance	-26

Table 4.14: ALP Table of AccofBranch

Attribute Name	Precedence
AccNo	-8
Opendate	-8
Status	-8
BrNo	24

Table 4.15: ALP Table of LnofBranch

Attribute Name	Precedence
LnNo	-8
Issuedate	-8
Status	-8
BrNo	24

Table 4.16: ALP Table of AccCust

Attribute Name	Precedence
AccNo	-8
Cid	-8

Table 4.17: ALP Table of LnCust

Attribute Name	Precedence
LnNo	-7
Cid	-8

4.4 Generation of Predicate Set and Fragmentation of the Relations

Predicate set was generated for the attributes with highest locality precedence of the relations respectively. These predicate sets were used to fragment the relations.

P_{Loan} ={LnType=SME, LnType=HOME , LnType=CAR }

$P_{Customer}$ ={BrNo=B01, BrNo=B02, BrNo=B03}

$P_{Accounts}$ ={AccType=Ind, AccType=Cor}

$P_{AccofBranch}$ ={BrNo=B01, BrNo=B02, BrNo=B03}

$P_{LnofBranch}$ ={BrNo=B01, BrNo=B02, BrNo=B03}

As for AccCust and LnCust relations, no attribute has significant higher precedence than other attributes, so predicate set was not generated for the relations. Instead these relations are to be fragmented derived horizontally with the help of their mother relation.

For Horizontal fragmentation of Customer relation, following queries are used:

$Q_{Customer1}$ =Select * from Customer where BrNo=B01;
$Q_{Customer2}$ =Select * from Customer where BrNo=B02;
$Q_{Customer3}$ =Select * from Customer where BrNo=B03;

For Horizontal fragmentation of Loan relation, following queries are used:

Q_{Loan1} =Select * from Loan where LnType=SME;
Q_{Loan2} =Select * from Loan where LnType= HOME;
Q_{Loan3} =Select * from Loan where LnType= CAR;

For Horizontal fragmentation of Accounts relation, following queries are used:

$Q_{Accounts1}$ =Select * from Accounts where AccType=Ind;
$Q_{Accounts2}$ =Select * from Accounts where AccType=Cor;

For Horizontal fragmentation of AccofBranch relation, following queries are used:

$Q_{AccofBranch1}$ =Select * from AccofBranch where BrNo=B01;
$Q_{AccofBranch2}$ =Select * from AccofBranch where BrNo=B02;

$Q_{AccofBranch3}$ =Select * from AccofBranch where BrNo=B03;

For Horizontal fragmentation of LnofBranch relation following queries are used:

$Q_{LnofBranch1}$ =Select * from LnofBranch where BrNo=B01;

$Q_{LnofBranch2}$ =Select * from LnofBranch where BrNo=B02;

$Q_{LnofBranch3}$ =Select * from LnofBranch where BrNo=B03;

For Horizontal fragmentation of AccCust relation, following queries are used:

$Q_{AccCust1}$ =Select AccNo, Cid from AccCust, Customer where AccCust.Cid = Customer.Cid and Customer.BrNo=B01;

$Q_{AccCust2}$ =Select AccNo, Cid from AccCust, Customer where AccCust.Cid = Customer.Cid and Customer.BrNo=B02;

$Q_{AccCust3}$ =Select AccNo, Cid from AccCust, Customer where AccCust.Cid = Customer.Cid and Customer.BrNo=B03;

For Horizontal fragmentation of LnCust relation, following queries are used:

$Q_{LnCust1}$ =Select LnNo, Cid from LnCust, Customer where LnCust.Cid = Customer.Cid and Customer.BrNo=B01;

$Q_{LnCust2}$ = Select LnNo, Cid from LnCust, Customer where LnCust.Cid = Customer.Cid and Customer.BrNo=B02;

$Q_{LnCust3}$ = Select LnNo, Cid from LnCust, Customer where LnCust.Cid = Customer.Cid and Customer.BrNo=B03;

Branch relation was not fragmented as it is a very small relation and most of access to its records is by read operation. Instead Branch relation will be replicated to all the sites of the DBDS.

In this way all the relations of the distributed banking system of Fig. 4.2 ware fragmented using the above queries and allocated to the three computers (sites).

4.5 Queries for Performance Analysis of MMF

We have executed twenty queries in each site with a total of sixty selected queries in the distributed system according to Pareto Principle often referred as 80/20 rule [59], [60] to see the performance of MMF. The queries were selected from the following *query domain* to accomplish enough variation of a real database system:

- **Insertion** e.g. Insert into RRR values (xxx, yyy, zzz);
- **Selection (Point)** e.g. Select A_1, $A_{2\ldots}$ A_n from RRR where xxx= P
- **Selection (Range)** e.g. Select A_1, $A_{2\ldots}$ A_n from RRR where xxx< BBB
- **Selection (Join)** e.g. Select A_1, A_2 … A_n from R_1, R_2 where $R_1.A_i=R_2.A_j$ AND $R_1.A_k$=CCC
- **Selection (Aggregation)** e.g. Select Sum (AA) from RRR where P
- **Update** e.g. Update RRR set A_i = xxx where A_j = yyy
- **Deletion** e.g. Delete * from RRR where P

We define *hit* as a result of a query of any type accessed records of local fragment of the site where the query was initiated and *miss* as a result of a query of any type accessed records of one or more remote fragments of other sites. The results of our experiment are shown in Table 4.18 – 4.25 and Fig. 4.4 – 4.11 below:

Table 4.18: Hit Miss Ratio for Loan

Site	Percentage of Hit	Percentage of Miss
$Site_1$	100%	0%
$Site_2$	75%	25%
$Site_3$	75%	25%
Average	83.33%	16.67%

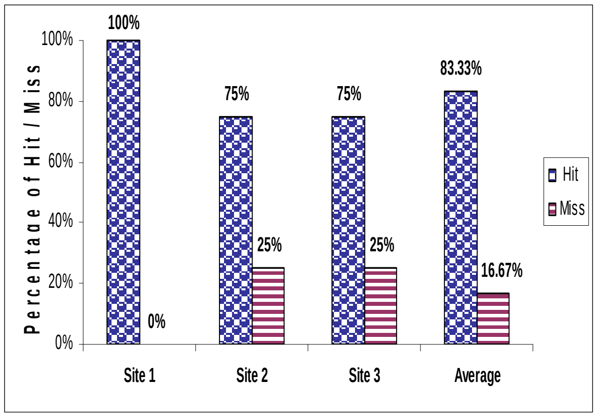

Fig. 4.4: Hit Miss Ratio for Loan Relation

From Table 4.18 we can see that all the queries of $Site_1$ accessed records from local fragment of Loan relation. So hit ratio in $Site_1$ is 100%. We also see that 75% queries executed at $Site_2$ and $Site_3$ accessed records of local fragment and 25% queries accessed records of fragment stored in other (remote) site rather than query generation site. Average hit ratio for Loan relation is 83.33%.

Table 4.19: Hit Miss Ratio for Customer

Site	Percentage of Hit	Percentage of Miss
Site$_1$	100%	0%
Site$_2$	66.67%	33.33%
Site$_3$	100%	0%
Average	88.89%	11.11%

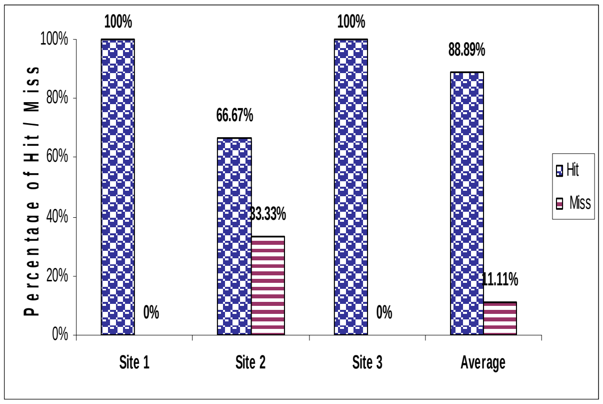

Fig. 4.5: Hit Miss Ratio for Customer

From Table 4.19 we can see that all the queries of Site$_1$ and Site$_3$ accessed records from local fragment of Customer relation. 33.33% queries generated in Site$_2$ accessed data of remote fragments. Average hit ratio is 88.89% and miss ratio is 11.11%.

Table 4.20: Hit Miss Ratio for Accounts

Site	Percentage of Hit	Percentage of Miss
Site$_1$	100%	0%
Site$_2$	66.67%	33.33%
Site$_3$	100%	0%
Average	88.89%	11.11%

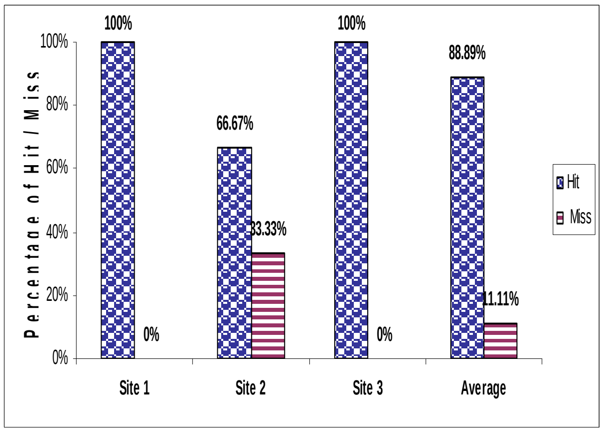

Fig. 4.6: Hit Miss Ratio for Accounts

From Table 4.20 we can see that all the queries of Site$_1$ and Site$_3$ accessed records from local fragments of Customer relation. So hit ratio is 100%. 33.33% queries generated in Site$_2$ accessed data of remote fragments. Average hit ratio is 88.89% and miss ratio is 11.11%.

Table 4.21: Hit Miss Ratio for AccofBranch

Site	Percentage of Hit	Percentage of Miss
$Site_1$	100%	0%
$Site_2$	100%	0%
$Site_3$	50%	50%
Average	83.33%	16.67%

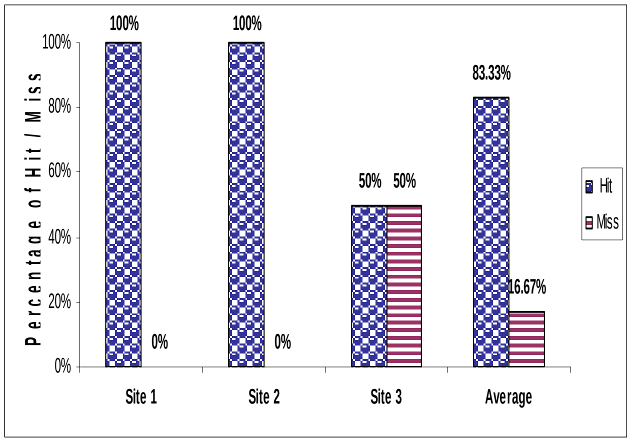

Fig. 4.7: Hit Miss Ratio for AccofBranch

From Table 4.21 we can see that all the queries of $Site_1$ and $Site_2$ accessed local fragment of AccofBranch relation. So hit ratio in $Site_1$ and $Site_2$ are 100%. We also see that 50% queries executed at $Site_3$ accessed records of local fragment and. Average hit ratio for AccofBranch relation is 83.33%.

Table 4.22: Hit Miss Ratio for LnofBranch

Site	Percentage of Hit	Percentage of Miss
$Site_1$	50%	50%
$Site_2$	66.67%	33.33%
$Site_3$	100%	0%
Average	72.22%	27.78%

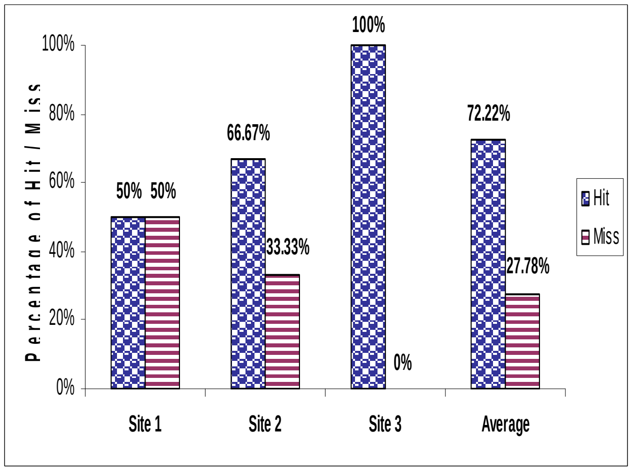

Fig. 4.8: Hit Miss Ratio for LnofBranch

From Table 4.22 we can see that 50%, 66.67%, 100% queries of $Site_1$, $Site_2$ and $Site_3$ accessed local fragment of LnofBranch relation respectively, average hit ratio for the relation is 72.22%.

Table 4.23: Hit Miss Ratio for AccCust

Site	Percentage of Hit	Percentage of Miss
$Site_1$	50	50
$Site_2$	100%	0%
$Site_3$	100%	0%
Average	83.33%	16.67%

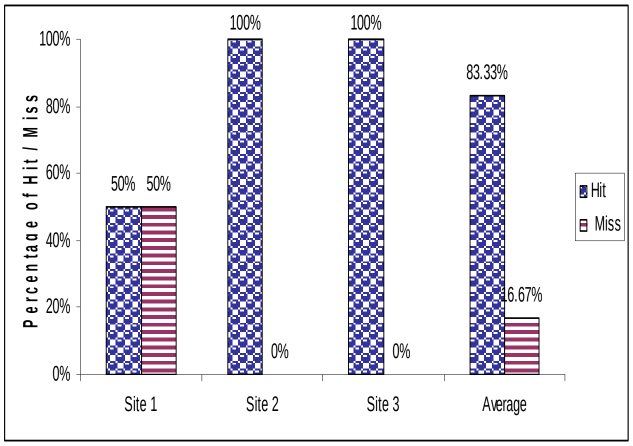

Fig. 4.9: Hit Miss Ratio for AccCust relation

From Table 4.23 we can see that all the queries of $Site_2$ and $Site_3$ accessed records from local fragment of AccCust relation. So hit ratio at these two sites are 100%. 50% queries of $Site_1$ accessed data of remote sites. Average hit ratio is 83.33% and miss ratio is 16.67%.

Table 4.24: Hit Miss Ratio for LnCust

Site	Percentage of Hit	Percentage of Miss
$Site_1$	100%	0%
$Site_2$	100%	0%
$Site_3$	50%	50%
Average	83.33%	16.67%

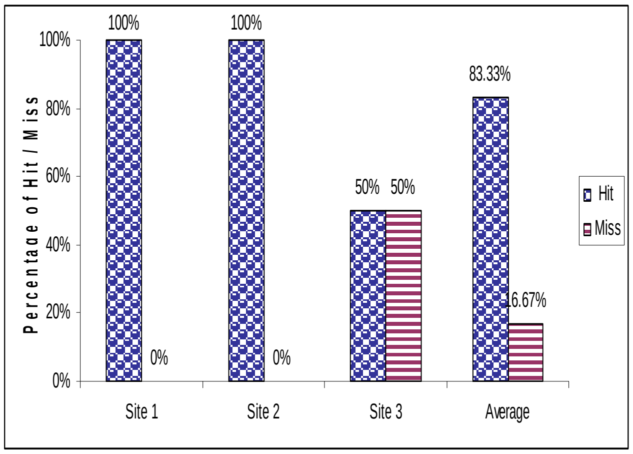

Fig. 4.10: Hit Miss Ratio for LnCust

From Table 4.24 we can see that all the queries of $Site_1$ and $Site_2$ accessed records from local fragment of AccCust relation. So hit ratio at these two sites are 100%. 50% queries of $Site_3$ accessed data of remote sites. Average hit and miss ratio are 83.33% and 16.67% respectively.

Table 4.25: Hit Miss Ratio for Branch

Site	Percentage of Hit	Percentage of Miss
$Site_1$	100%	0%
$Site_2$	100%	0%
$Site_3$	100%	0%
Average	100%	0%

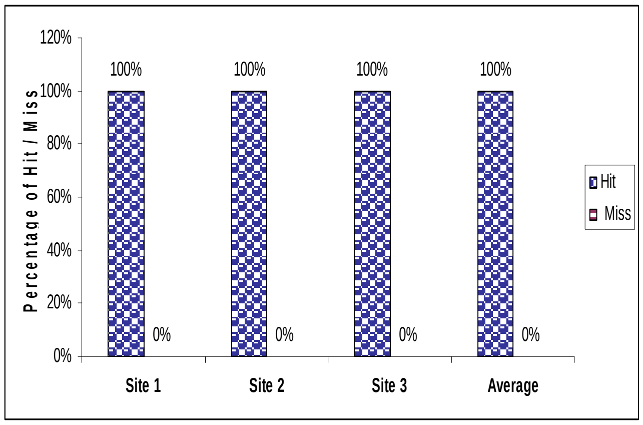

Fig. 4.11: Hit Miss Ratio for Branch

From Table 4.25 we can see that all the queries of $Site_1$, $Site_2$ and $Site_3$ accessed records from local fragment of LnCust relation. So hit ratio at all three sites as well as average hit ratio is 100% and miss ratio is 0%.

Overall Performance:

Table 4.26 and Fig. 4.12 show the overall performance of the distributed system after fragmenting the relations using MMF technique. We can see that after fragmentation and allocation using MMF technique, 85% of the queries generated in any site accessed records of only that site and remote access reduced to 15%. This is definitely a significant achievement.

Table 4.26: Overall System Performances of MMF

Site Name	Queries executed	Accessed fragment stored in local site	Accessed fragment stored in remote site	Percentage of Hit	Percentage of Miss
$Site_1$	20	18	2	90%	10%
$Site_2$	20	16	4	80%	25%
$Site_3$	20	17	3	85%	15%
DDBMS	60	51	9	85%	15%

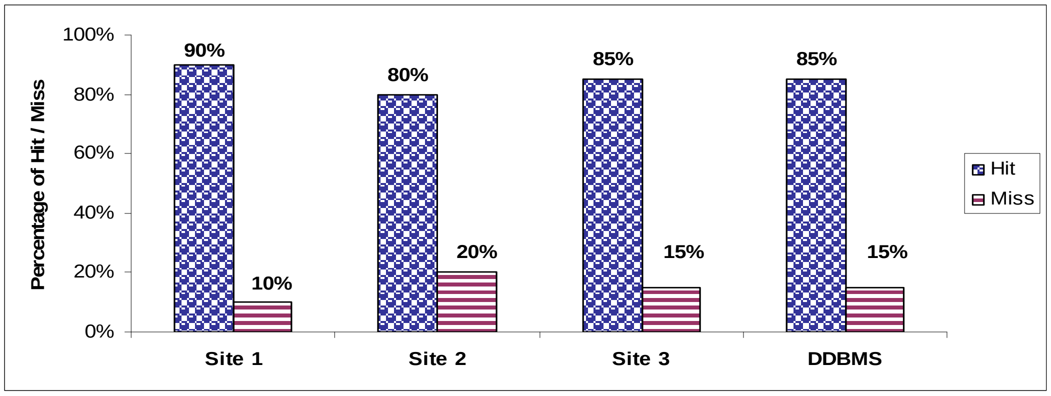

Fig. 4.12: Hit Miss Ratio of MMF Technique for Three Sites

4.6 Comparison with other Techniques

We have named the techniques deals with fragmentation problem of distributed database without addressing the initial stage problem as Techniques without Initial Fragmentation (TWIF) as in [1] – [40], [42] - [49]. TWIF uses the following model in general:

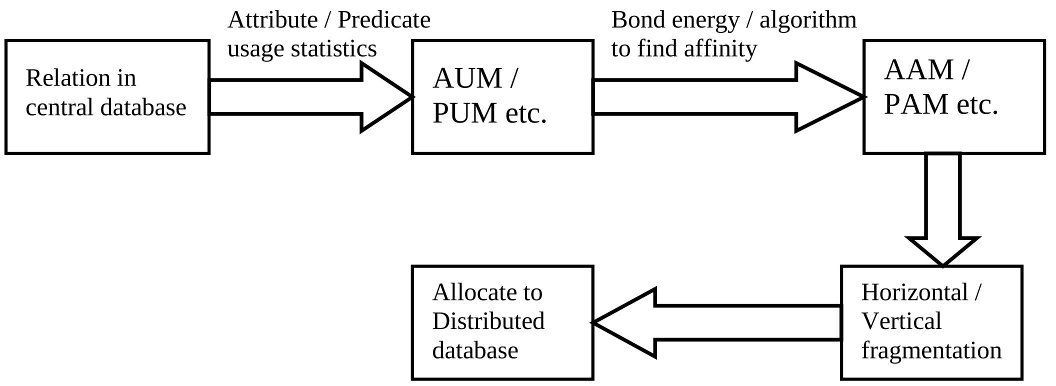
Relation in central database
Attribute / Predicate usage statistics
AUM / PUM etc.
Bond energy / algorithm to find affinity
AAM / PAM etc.
Horizontal / Vertical fragmentation
Allocate to Distributed database

Fig. 4.13: Model of other Non-initial Fragmentation Techniques

TWIF first store the relations of a distributed database in a single site of the distributed system as a centralized database. The other sites where database is not stored, access the database with different type of queries using remote network connection of the system. Information about attribute, predicate access pattern and frequencies of access by different queries from different sites are gathered in tables called Attribute Usage

Matrix (AUM) or Predicate Usage Matrix (PUM) or similar tables. After a certain period when sufficient statistical data are gathered for calculating the relationship (known as affinity) of attribute or predicate with transaction of sites, Attribute Affinity Matrix (AAM) or Predicate Affinity Matrix (PAM) are generated using Bond Energy algorithm or similar algorithm. From AAM and PAM, vertical and horizontal fragmentation decision is made respectively. Then produced fragments are to be stored in the sites of the distributed database though almost all TWIF ignore allocation of the fragments to reduce complexity.

We have implemented the above model in our lab and execute the same sixty queries those were used to test our technique with the assumption that at the initial stage the centralized database is stored at $Site_1$. Table 4.27 shows the overall system performance of TWIF before DDBMS is fragmented and allocated to sites. We can see from Table 4.27 that during a long period before reasonable amount of statistical record access frequencies by transactions are available for constructing attribute affinity matrix or predicate affinity matrix and to fragment and allocate the database among the three sites, percentage of hit of the overall system is only 33.33% which is much less in comparison with our achieved 85.71% hit rate. This is graphically represented in Fig. 4.14 below. The reason of poor performance of TWIF is that, all sites other than central site have no data. So all queries that are generating in those sites requires remote data access thus scores *miss*. Only $site_1$ got 100% hit because the whole database is stored there centrally before fragmentation is performed to the DDBMS.

Table 4.27: Overall System Performance of TWIF

Site Name	Queries executed	Access fragment stored in local site	Access fragment stored in remote site	Percentage of Hit	Percentage of Miss
$Site_1$	20	20	0	100%	0%
$Site_2$	20	0	20	0%	100%
$Site_3$	20	0	20	0%	100%
DDBMS	60	20	40	33.33%	66.66%

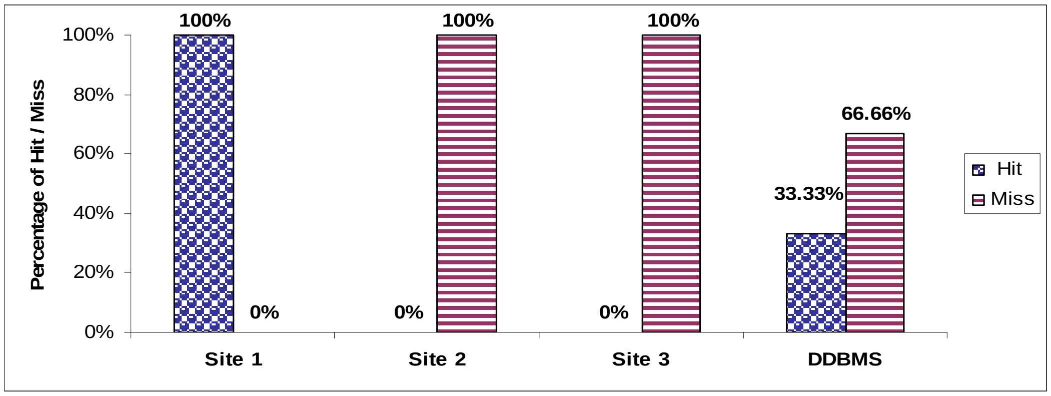

Fig. 4.14: Hit Miss Ratio of TWIF for Three Sites

Comparison of MMF and TWIF by their hit and miss ratio is represented in Fig. 4.15-Fig. 4.16.

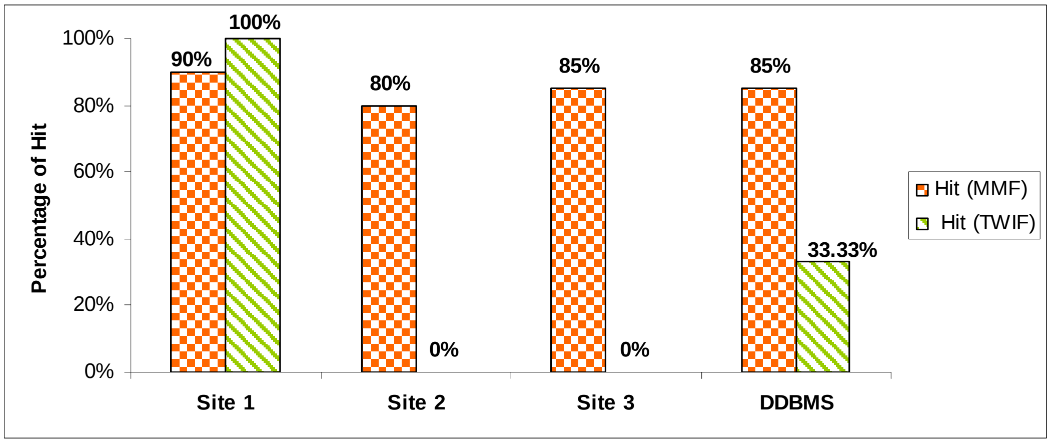

Fig. 4.15: Comparison of Hit between MMF & TWIF for Three Sites

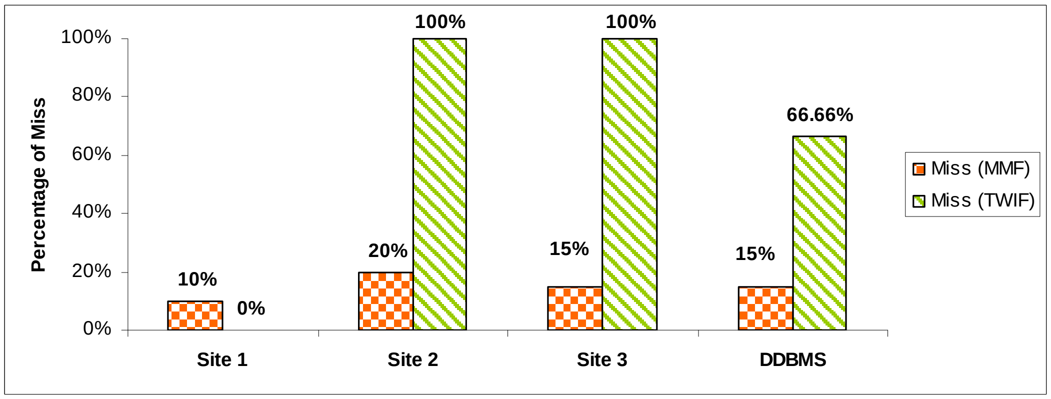

Fig. 4.16: Comparison of Miss between MMF & TWIF for Three Sites

After a long period when sufficient empirical data will be available for construction of AAM or PAM, TWIF will fragment their relations and allocate the fragmented sub-relations to the sites of the distributed system. Then the percentage of hit of the overall system will increase. This is shown in Table 4.28. We have fragmented the Loan relation of the distributed banking database by the techniques of [1], [18] and [25]. We have used the statistics of the same queries executed to find out performance of MMF to construct predicate affinity matrix for TWIF.

Table 4.28: Performance of TWIF for Loan Relation after Allocation

Site Name	Percentage of Hit	Percentage of Miss
Site_1	75%	25%
Site_2	100%	0%
Site_3	100%	0%
DDBMS	91.66%	8.33%

We can see from Table 4.28 that after allocation of fragments of relation Loan into the sites of the distributed system, hit ratio of TWIF increases from 33.33% to 91.66%. As actual query statistics of the system is found and relations are fragmented based on that statistics in TWIF, so hit rate significantly increased. For the same relation, MMF achieves 83.33% hit ratio which is much closer to TWIF. This situation is depicted in Fig. 4.17.

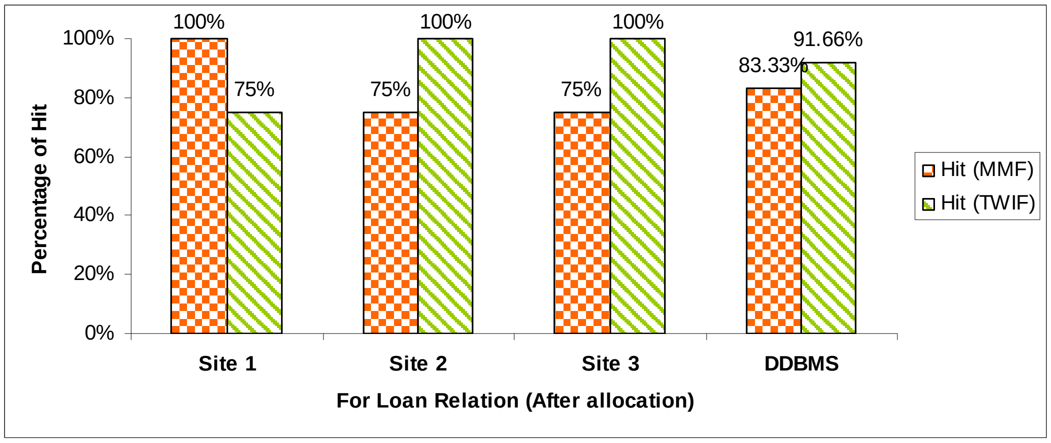

Fig. 4.17: Hit Miss Ratio of TWIF for Loan Relation after Allocation

Cost of allocation of fragments for TWIF:

From Table 4.28 it can be seen that performance of TWIF increases significantly after fragmentation based on empirical data and allocate the fragments to respective sites. An important thing to note that as TWIF stores all the data of the distributed database into a single site (Central node) before allocation, so transferring data to different sites will incur high cost. Following graph of Fig. 4.18 shows amount of data transfer and time required if fragmentation and allocation is performed after 1, 2, 3, 4, 5 and 6 months. A simulation was done in MATLAB with following assumptions:

- Database used: Distributed Banking Database System (DBDS)
- Number of tables: 8
- Number of sites: 10
- Number of application running in each site: 3
- Frequency of data entry: 1 tuple in each table of each site every second
- Data transfer rate among the sites of distributed system: 256 KBPS

We can see from Fig. 4.18 that, for a very small database DBDS, about 110 GB data have to be transferred and approximately 4 days are required for fragmentation and allocation based on TWIF.

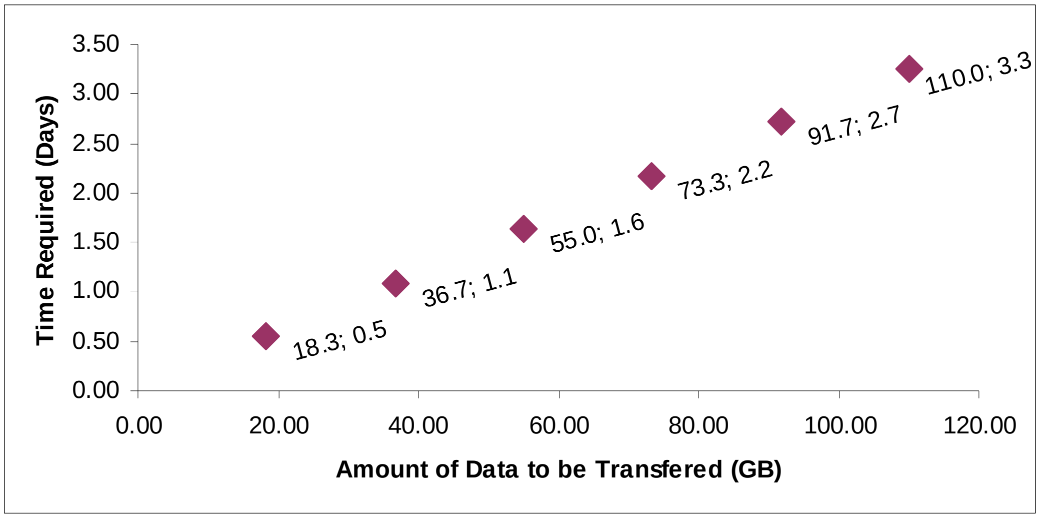

Fig. 4.18: Simulation Results for Data Transfer & Time Requirement (TWIF)

Fig. 4.19 shows a comparison among total data generation, amount of data transfer required using MMF with updated MCRUD matrix based on empirical data and amount of data transfer required using TWIF for fragments allocation. We can see that if we fragment the relations based on MMF technique previously then total data transfer requirement is much less comparing with TWIF.

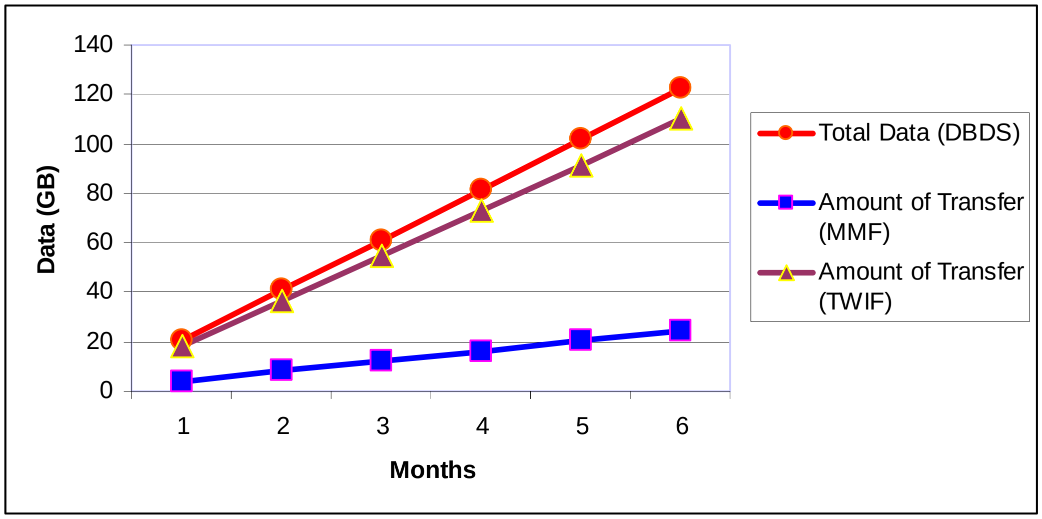

Fig. 4.19: Comparison of Amount of Data

Fig. 4.20 shows a comparison between data transfer time required using MMF with updated MCRUD matrix based on empirical data and transfer time required using TWIF for fragments allocation. We can see that if we fragment the relations based on

MMF technique previously then much less data transfer time required comparing with TWIF.

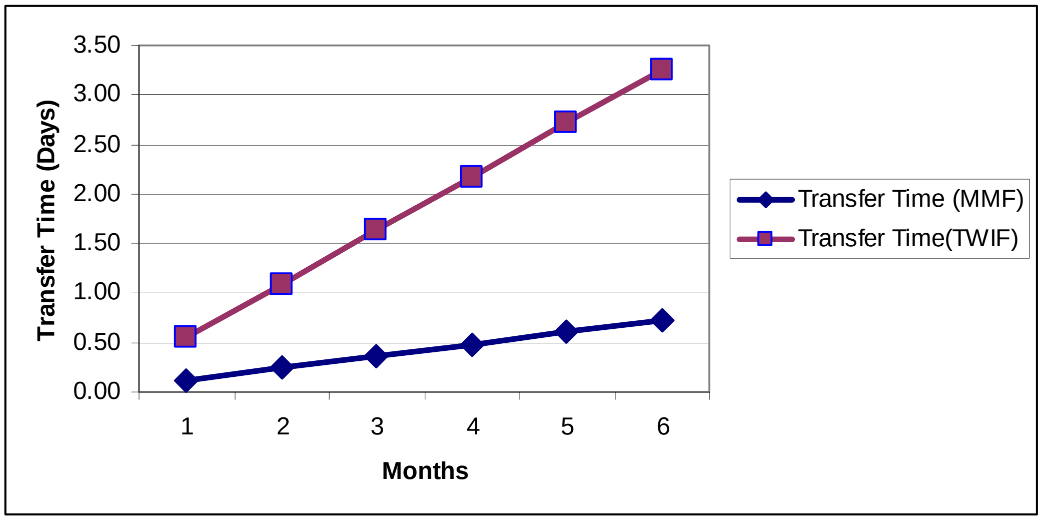

Fig. 4.20: Comparison of Transfer Time for MMF and TWIF

Comparison with StatPart:

Existing technique that provided a solution of initial fragmentation is StatPart described in [41]. To fragment a relation, it starts with a randomly generated matrix of attribute vs. queries called reflexivity matrix. It then construct symmetry matrix from reflexivity matrix using two equations. Symmetry matrix is inputted to transitivity module which uses an algorithm to produce two set of attributes that are used to break the relation into two binary vertical fragments.

Main drawbacks of StatPart [41] are:

- It can suggest only two binary vertical fragments independent of number of sites of the distributed system. So this technique is not suitable for a distributed system with more than two allocation sites.
- As it starts with a randomly generated matrix that represents the relationship among attributes and queries, optimum fragmentation decision cannot be provided using this algorithm. So it continuously shift attributes from one fragment to another fragment trial and error basis to improve hit ratio. But this policy is not feasible on trial because of high cost incurred by transferring large amount of data among sites.

Table 4.29 shows the comparison between MMF and StatPart techniques. Both the techniques address initial fragmentation problem.

Table 4.29: Comparison between StatPart and MMF Techniques

Criteria	StatPart	MMF
Address initial fragmentation problem?	Yes	Yes
Number of Fragments	Always two	Any number
Allocation	Trial and error basis	Where ALP maximum
Replication	Not supported	Supported
Performance	Random	Good and steady
Fragmentation type	Vertical	Horizontal / Vertical

4.7 Impact of the Increase of Number of Sites

Now we want to experiment the generalization of MMF so that we can verify whether our technique is applicable to any number of sites of distributed system.

4.7.1 Number of Allocation Site is Four

We have increased total number of sites to four at design time by adding a local branch of DBDB named Loc$_3$ at Site$_4$. This situation is depicted in Fig. 4.21.

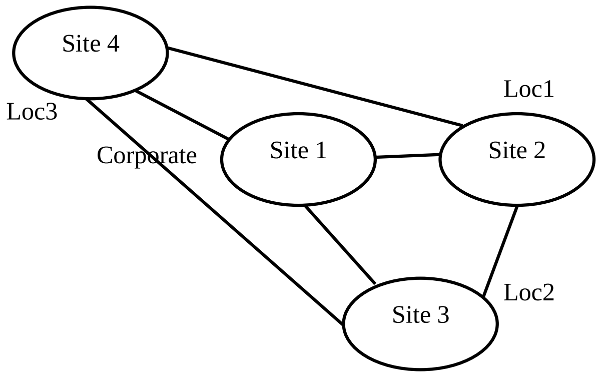

Fig. 4.21: DBDB with Four Sites

4.7.1.1 Implementation of MMF for Four Sites

We have constructed the MCRUD matrix of Loan relation for four sites with three applications running in each site. It is shown in Table 4.30 below:

Table 4.30: MCRUD Matrix of Loan Relation with Four Sites

Site.Application / Entity.Attribute.Predicates	Site1			Site2			Site3			Site4		
	Ap1	Ap2	Ap3	Ap1	Ap2	Ap3	Ap1	Ap2	Ap3	Ap1	Ap2	Ap3
Loan.LnNo<10000	RU	R	CRUD	RU	R	CRUD	R	R	CRUD	RU	R	CRUD
Loan.LnNo>=10000	R	R	CRUD	R	RU	CRUD	R	RU	CRUD	R	RU	CRUD
Loan.LnType=SME	R		RU	RU	R	CRUD	R		RU	RU		CRUD
Loan.LnType=HOME	RU	RU	CRUD	R		RU	R		RU	R		RU
Loan.LnType=CAR	R		RU	R		RU	RU		CRUD	RU	R	CRUD
Loan.Amount<50000	R		CRUD	R		CRUD	R		CRUD	RU	R	CRUD
Loan.Amount=50000:100000	R	R	CRUD	R		CRUD	R		CRUD	R	R	CRUD
Loan.Amount>100000	RU	R	CRUD	R		CRUD	R		CRUD	R		RU

From Table 4.30 we have calculated ALP table for Loan relation shown in Table 4.31. The process of how fragmentation and replication decision is made in four sites can be understood from Table 4.32.

Table 4.31: ALP Table of Loan Relation with Four Sites

Attribute Name	**Precedence**
LnNo	-46
LnType	-17
LnAmount	-42

Table 4.32 Precedence Calculation and Fragmentation Decision for Loan Relation

Attribute Name	**Predicates**	**Precedence in Site 1**	**Precedence in Site 2**	**Precedence in Site 3**	**Precedence in Site 4**	**Decision**
LnType	LnType = SME	**5**	**13**	**5**	**12**	**Fragment in Site 2 Replica in site 4**
	LnType = HOME	**16**	**5**	**5**	**5**	**Fragment in Site 1**
	LnType = CAR	**5**	**5**	**12**	**13**	**Fragment in Site 4 Replica in site 3**

Predicate set is generated for the attribute LnType of Loan relation.

P_{Loan} ={LnType=SME, LnType=HOME , LnType=CAR }

For Horizontal fragmentation of Loan relation, following queries were used:

Q_{Loan1} =Select * from Loan where LnType=HOME;
Q_{Loan2} =Select * from Loan where LnType= SME;
Q_{Loan3} =Select * from Loan where LnType= CAR;

$Q_{Loan4.1}$=Select * from Loan where LnType=SME;
$Q_{Loan4.2}$ =Select * from Loan where LnType= CAR;

4.7.1.2 Performance Analysis of MMF for Four Sites

We have executed same queries as previous in four sites of DBDS to check the impact of site addition on hit miss ratio. Result is shown in Table 4.33 and Fig 4.22. We can see that average hit ratio is 81.25% that is very close to our previous result 83.33% achieved for three sites.

Table 4.33: Performance of MMF for Loan Relation Distributed in Four Sites

Site	Percentage of Hit	Percentage of Miss
$Site_1$	100%	0%
$Site_2$	75%	25%
$Site_3$	75%	25%
$Site_4$	75%	25%
Average	81.25%	19.75%

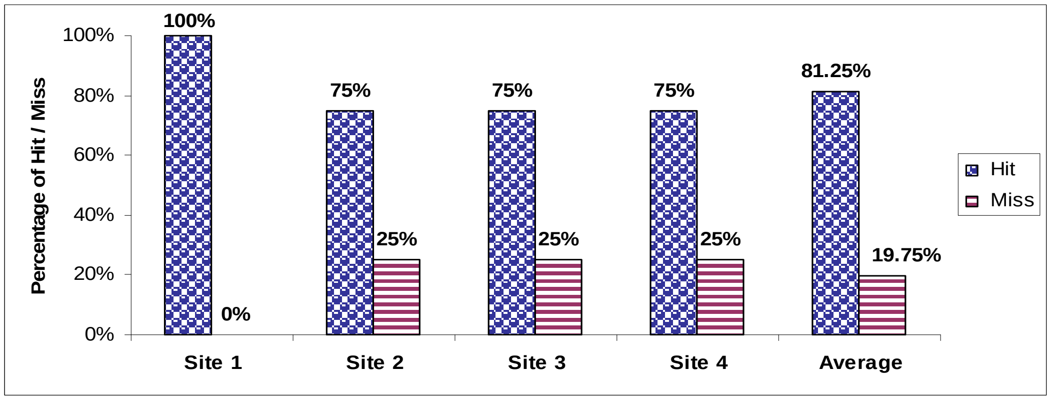

Fig. 4.22: Hit Miss Ratio of MMF for Loan Relation Distributed in Four Sites

4.7.2 Number of Allocation Site is Five

We have increased total number of sites to five at design time by adding a branch deals with industrial matters named Industrial at Site 5. This situation is depicted in Fig. 4.23.

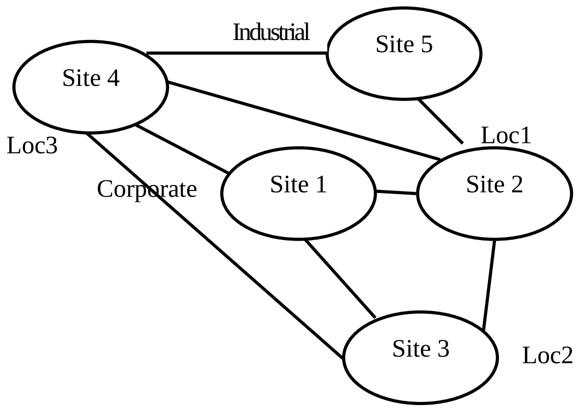

Fig. 4.23: DBDB with Five Sites

4.7.2.1 Implementation of MMF for Five Sites

We have constructed the MCRUD matrix of Loan relation for five sites with three applications running in each site. It is shown in Table 4.34.

Table 4.34: MCRUD Matrix of Loan Relation with Five Sites

Site.Application Entity.Attribute.Predicates	Site1			Site2			Site3			Site4			Site5		
	Ap1	Ap2	Ap3	Ap1	Ap2	Ap3	Ap1	Ap2	Ap3	Ap1	Ap2	Ap3	Ap1	Ap2	Ap3
Loan .LnNo<10000	RU	R	CRUD	RU	R	CRUD	R	R	CRUD	R	R	CRUD	R	R	CRUD
Loan .LnNo>=10000	R	R	CRUD	R	RU	CRUD	R	RU	CRUD	R	RU	CRUD	R	RU	CRUD
Loan.LnType=SME	R		RU	RU	R	CRUD	R		RU	RU		CRUD	R		R
Loan.LnType=HOME	RU	RU	CRUD	R		RU	R		RU	R		RU	R		R
Loan.LnType=CAR	R		RU	R		RU	RU		CRUD	RU	R	CRUD	R	R	R
Loan.LnType=INDSTRY	R		R	R		R	R		RU	R		R	RU	R	CRUD
Loan.Amount<50000	R		CRUD	R		CRUD	R		CRUD	R		CRUD	R		CRUD
Loan.Amount=50000:100000	R	R	CRUD	R		CRUD	R		CRUD	R		CRUD	R		CRUD
Loan.Amount>100000	R		CRUD	R		CRUD	R		CRUD	R		CRUD	R		CRUD

From Table 4.34 we have calculated ALP table for Loan relation shown in Table 4.35. The process of how fragmentation and replication decision is made in five sites can be understood from Table 4.36.

Table 4.35: ALP Table of Loan relation with Five Sites

Attribute Name	**Precedence**
LnNo	-63
LnType	-22
LnAmount	-80

Table 4.36: Precedence Calculation and Fragmentation Decision for Loan Relation

Attribute Name	**Predicates**	**Precedence in Site 1**	**Precedence in Site 2**	**Precedence in Site 3**	**Precedence in Site 4**	**Precedence in Site 5**	**Decision**
LnType	LnType = SME	**5**	**13**	**5**	**12**	**2**	**Fragment in Site 2 Replica in site 4**
	LnType = HOME	**16**	**5**	**5**	**5**	**2**	**Fragment in Site 1**
	LnType = CAR	**5**	**5**	**12**	**13**	**3**	**Fragment in Site 4 Replica in site 3**
	LnType= INDUSTRY	**2**	**2**	**4**	**2**	**13**	**Fragment in Site 5**

Predicate set is generated for the attribute LnType of Loan relation.

P_{Loan} ={LnType=SME, LnType=HOME , LnType=CAR }

For Horizontal fragmentation of Loan relation, following queries were used:

Q_{Loan1} =Select * from Loan where LnType=HOME;
Q_{Loan2} =Select * from Loan where LnType= SME;
Q_{Loan3} =Select * from Loan where LnType= CAR;

$Q_{Loan4.1}$=Select * from Loan where LnType=SME;
$Q_{Loan4.2}$ =Select * from Loan where LnType= CAR;

Q_{Loan5} =Select * from Loan where LnType= INDUSTRY;

4.7.2.2 Performance Analysis of MMF for Five Sites

We have executed same queries as previous in five sites of DBDS to check the impact of site addition on hit miss ratio. Result is shown in Table 4.37 and Fig. 4.20. We can see that average hit ratio is 82% which is quite close to our previous result 83.33% achieved for three sites and 81.25% achieved for four sites. Another thing is to mention that from experimental result it can be concluded that MMF has no inverse relation of performance with increase of number of sites. Rather, in a site if the queries that are generating are identical to the MCRUD matrices, hit rate will better. Otherwise hit rate will decrease.

Table 4.37: Performance of MMF for Loan Relation Distributed in Five Sites

Site	Percentage of Hit	Percentage of Miss
$Site_1$	100%	0%
$Site_2$	75%	25%
$Site_3$	80%	20%
$Site_4$	80%	20%
$Site_5$	75%	25%
Average	82%	18%

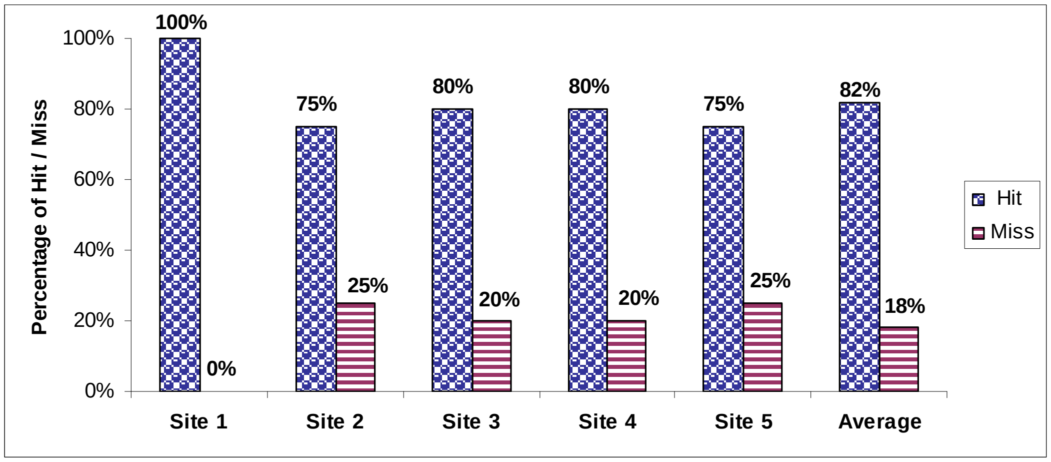

Fig. 4.24 Hit Miss Ratio of MMF for Loan Relation Distributed in Five Sites

Table 4.38 shows the hit and miss ratio of TWIF for Loan relation when number of sites of DBDB are five. It can be seen that average hit ratio of the system is 20% which is very poor in comparison with MMF that achieves 82% hit.

Table 4.38: Performance of TWIF for Loan Relation when Sites are Five

Site	Percentage of	Percentage of
Site_1	100%	100%
Site_2	0%	0%
Site_3	0%	0%
Site_4	0%	0%
Site_5	0%	0%
Average	20%	20%

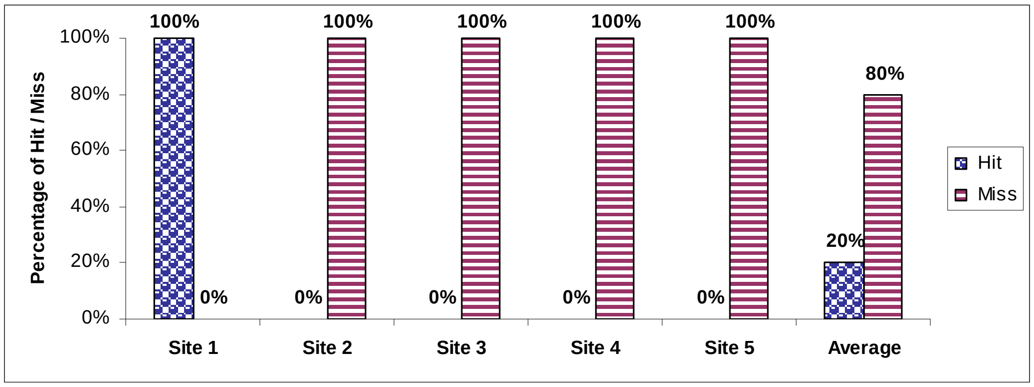

Fig. 4.25: Hit Miss Ratio of TWIF for Loan Relation When Sites is Five

Fig. 4.26 shows the performance of MMF and TWIF with the increase of number of sites in the distributed system. We can see that MMF shows much better and quite steady performance as sites increases from three to ten. In the same time performance of TWIF falls gradually as new sites are added to the system. This is because when new sites increase, they are only generating queries but have no data to answer the queries. So it contributes to increase the miss rate of overall system fragmented based on TWIF. It can be expressed by the equation: Hit rate = $^1/_S$, where S is the total number of sites in the system.

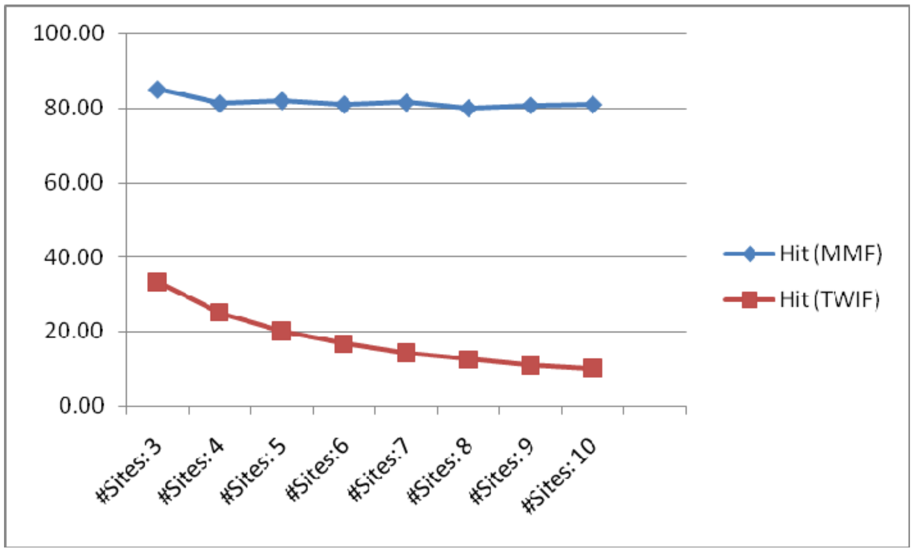

Fig. 4.26: Comparison of Hit Ratio between MMF and TWIF with Increasing Number of Sites

4.8 Summary

From the above result we can see that our technique has clearly outperforms the technique stated in [41]. Our fragmentation technique achieved a very good hit rate which is approximately 84%. As other techniques described in [1] – [40], [42] - [49] could not provide solutions for initial state of the distributed system. Using TWIF initial performance (hit ratio) of the system is only 33.33%. After a long period when sufficient data for fragmenting the centralize database were available, hit rate of TWIF increased significantly as much as 91.66% but in the price of high transfer cost incurred for transferring data among the sites of the distributed system using communication network.

Another thing is to mention that MMF achieves a steady hit rate over 80% and TWIF's performance falls gradually from 33.33% to 10% with the increase of number of sites of DBDS from three to ten.

Chapter 5
Conclusion and Future Research

Making proper fragmentation of the relations and allocation of the fragments is a major research area in distributed systems. Many techniques have been proposed by the researchers using empirical knowledge of data access by different queries and frequencies of queries executed in different sites of a distributed system. But proper fragmentation and allocation at the initial stage of a distributed database has not yet been addressed.

In this thesis 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. Instead of using empirical data, we have developed a matrix namely Modified Create, Read, Update and Delete (MCRUD) to find out precedence of attributes which increase locality of data. We have named this precedence as Attribute Locality Precedence (ALP) which is used for making fragmentation decisions. 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.

5.1 Contributions of the Thesis

- ❖ The main contribution of this research is to develop a fragmentation technique that can fragment relations of distributed database without the help of runtime empirical data.
- ❖ Relations are fragmented initially with the help of ALP tables those are constructed from MCRUD matrices using our developed cost functions. This overcomes initial fragmentation problem of distributed database that is not properly addressed in other fragmentation techniques.
- ❖ A very good hit rate (Approximately 85%) is achieved using out technique for various kinds of insertion, selection, join, deletion and other queries.

- MMF technique can be applicable for any number of sites of the system. Its performance is quite stable with increasing number of sites.
- In our technique large amount of costly data transfer using communicational network can be avoided as fragments are correctly allocated to different sites at the initial stage of the system.
- Creation of MCRUD matrix for every relation and calculation of ALP from each matrix adds some additional cost in our system but this can be ignored because matrix construction and ALP calculation will be perform offline during the requirement analysis phase of distributed database development.

5.2 Future Research

In this research we have focused mainly on horizontal fragmentation of relational database using MCRUD matrix. Our research can be extended to several directions.

Firstly, technique for vertical fragmentation of relational database using MCRUD matrix can be developed. Integrating horizontal and vertical fragmentation, a mixed or hybrid fragmentation technique can also be developed in the next step.

As distributed object oriented databases and data warehouses are gaining popularities now a day so our research can be extended to support fragmentation in distributed object oriented databases and data warehouses as well.

Related Publications

- Shahidul Islam Khan and Dr. A. S. M. Latiful Hoque, “A New Technique for Database Fragmentation in Distributed Systems”, International Journal of Computer Applications (IJCA), Vol. 5 No. 9, August 2010, ISBN: 978-93-80746-60-9 (print), ISSN 0975-8887(online), pp. 20-24.

- Shahidul Islam Khan and Dr. A. S. M. Latiful Hoque, “A Novel Technique for Initial Fragmentation of Distributed Relational Database using CRUD Matrix”, Accepted in Annual International Conference on Advances in Distributed and Parallel Computing, ADPC 2010, 1 – 2 November 2010, Singapore, www.dpcomputing.org.

Bibliography

[1] M. T. Ozsu and P. Valduriez, *Principles of Distributed Database Systems*. Prentice-Hall, New Jersey, 1999.

[2] S. Ceri and G. Pelagatti, *Distributed Databases Principles and System*. McGraw- Hill, New York, 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*, vol. 16, no. 1, pp. 45–90, 2004.

[5] A. M. Tamhankar and S. Ram, "Database Fragmentation and Allocation: An Integrated Methodology and Case Study," *IEEE Trans. Systems Management*, vol. 28, no. 3, pp. 194–207, 1998.

[6] R. Blankinship, A. R. Hevner, and S. B. Yao, "An Iterative Method for Distributed Database Design," *Proc. 17th Int'l Conf. on Very Large Data Bases*, pp. 389–400, 1991.

[7] H. Ma, *Distribution Design for Complex Value Databases*, Doctoral thesis, Department of Information Systems, Massey University, 2007.

[8] S. Ceri, M. Negri, and G. Pelagatti, "Horizontal Data Partitioning in Database Design," *Proc. ACM SIGMOD Int'l Conf. on Management of Data*, pp. 128–136, 1982.

[9] Y. Zhang, "On Horizontal Fragmentation of Distributed Database Design," *Advances in Database Research*, World Scientific Publishing, pp. 121–130, 1993.

[10] M. Ra, "Horizontal Partitioning for Distributed Database Design," *Advances in Database Research*, World Scientific Publishing, pp. 101–120, 1993.

[11] C.-H. Cheng, W.-K. Lee, and K.-F. Wong, "A Genetic Algorithm-Based Clustering Approach for Database Partitioning," *IEEE Trans. Systems, Man, and Cybernetics, Part C*, vol. 32, no. 3, pp. 215–230, 2002.

[12] H. Mahboubi and J. Darmont, "Enhancing XML Data Warehouse Query Performance by Fragmentation," *Proc. ACM Symposium on Applied Computing (SAC09)*, pp.1555-1562, 2009.

[13] S.-K. Chang and W.-H. Cheng, "A Methodology for Structured Database Decomposition," *IEEE Trans. Software Engineering (TSE)*, vol. 6, no. 2, pp. 205–218, 1980.

[14] D. G. Shin and K. B. Irani, "Partitioning a Relational Database Horizontally Using a Knowledge-Based Approach," *ACM SIGMOD Record* vol. 14, no. 4, pp. 95–105, 1985.

[15] D. G. Shin and K. B. Irani, "Fragmenting Relations Horizontally Using a Knowledge Based Approach," *IEEE Trans. Software Engineering (TSE)*, vol. 17, no. 9, pp. 872–883, 1991.

[16] N. Khalil, D. Eid, and M. Khair, "Availability and Reliability Issues in Distributed Databases Using Optimal Horizontal Fragmentation," *Springer Lecture Notes in Computer Science*, vol. 1677, pp. 771–780, 1999.

[17] K. Karlapalem, S. B. Navathe, and M. M. A. Morsi, "Issues in Distribution Design of Object-Oriented Databases," *Proc. Int'l Workshop Distributed Object Management (IWDOM)*, pp. 148–164, 1992.

[18] H. Ma, K.-D. Schewe, and Q. Wang, "A Heuristic Approach to Cost-Efficient Derived Horizontal Fragmentation of Complex Value Databases," *Proc. 18th Australasian Database Conf. (ADC)*, pp. 103 – 111, 2007.

[19] L. Bellatreche, K. Karlapalem, and G. Basak, "Horizontal Class Partitioning for Queries in Object Oriented Databases," *HKUST-CS98-6 Tech. report*, 1998.

[20] L. Bellatreche, K. Karlapalem, and A. Simonet, "Algorithms and Support for Horizontal Class Partitioning in Object-Oriented Databases," *Distributed and Parallel Databases*, vol. 8, no. 2, pp. 155–179, 2000.

[21] C. I. Ezeife and K. Barker, “Horizontal Class Fragmentation in Distributed Object Based Systems,” *Proc. Second Biennial European Joint Conf. on Engineering Systems Design and Analysis*, pp. 225–235, 1994.

[22] C. I. Ezeife and K. A. Barker, “Comprehensive Approach to Horizontal Class Fragmentation in a Distributed Object Based System,” *Distributed and Parallel Databases*, vol. 3, no. 3, pp. 247–272, 1995.

[23] C. I. Ezeife and J. Zheng, “Measuring the Performance of Database Object Horizontal Fragmentation Schemes,” *Proc. Int’l Database Engineering and Applications Symposium (IDEAS)*, pp. 408–414, 1999.

[24] F. Bai˜ ao, M. Mattoso, and G. Zaverucha, “Horizontal Fragmentation in Object DBMS: New Issues and Performance Evaluation,” *Proc. 19th IEEE Int’l Performance, Computing and Communications Conf.*, pp. 108–114, 2000.

[25] F.F. Marwa, I.E. Ali, and A. A. Hesham, “A Heuristic Approach for Horizontal Fragmentation and Alllocation in DOODB,” *Proc. INFOS2008*, pp. 9-16, 2008.

[26] S. Ceri, S. B. Navathe, and G. Wiederhold, “Distribution Design of Logical Database Schemas,” *IEEE Trans. Software Engineering (TSE)* vol. 9, no. 4, pp. 487–504, 1983.

[27] F. Bai˜ ao, and M. Mattoso, “A Mixed Fragmentation Algorithm for Distributed Object Oriented Databases,” *Proc. Int’l Conf. Computing and Information*, pp. 141–148, 1998.

[28] J. A. Hoffer and D. G. Severance, “The Use of Cluster Analysis in Physical Database Design,” *Proc. First Int’l Conf. Very Large Data Bases (VLDB)*, pp. 69–86, 1975.

[29] 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.

[30] S. B. Navathe and M. Ra, “Vertical Partitioning for Database Design: A Graphical Algorithm,” *ACM SIGMOD Record*, vol. 14, no. 4, pp. 440–450, 1989.

[31] X. Lin and Y. Zhang, “A New Graphical Method of Vertical Partitioning in Database Design,” *Proc. 4th Australian Database Conf. (ADC)*, pp. 131–144, 1993.

[32] H. Ma, K.-D. Schewe, and M. Kirchberg, “A Heuristic Approach to Vertical Fragmentation Incorporating Query Information,” *Proc. 7th Int’l Baltic Conf. on Databases and Information Systems (DB&IS)*, pp. 69–76, 2006.

[33] M. AlFares, H. Abdalla, and F. Marir, “Vertical Partitioning for Database Design: A Grouping Algorithm,” *Proc. Int’l Conf. Software Engineering and Data Engineering (SEDE)*, pp. 218-223, 2007.

[34] T. H. Ngo, “New Objective Function for Vertical Partitioning in Database System,” *Proc. Spring Young Researcher's Colloquium on Database and Information Systems*, 2008.

[35] Runceanu A. “Fragmentation in Distributed Databases,” *Innovations and Advanced Techniques in Systems, Computing Sciences and Software Engineering*, Springer, pp. 57–62, 2008.

[36] D. Cornell and P. Yu, “A Vertical Partitioning Algorithm for Relational Databases,” *Proc. Int’l Conf. on Data Engineering*, pp. 30–35, 1987.

[37] D. W. Cornell and P. S. Yu, “An Effective Approach to Vertical Partitioning for Physical Design of Relational Databases,” *IEEE Trans. on Software Engineering*, vol. 16, no. 2, pp. 248–258, 1990.

[38] P.-C. Chu, “A Transaction Oriented Approach to Attribute Partitioning,” *Information Systems* vol. 17, no. 4, pp. 329–342, 1992.

[39] S. Chakravarthy, J. Muthuraj, R. Varadarajan, and S. B. Navathe, “An Objective Function for Vertically Partitioning Relations in Distributed Databases and its Analysis,” *Distributed and Parallel Databases*, vol. 2, no. 2, pp. 183–207, 1994.

[40] J. H. Son and M. H. Kim, “An Adaptable Vertical Partitioning Method in Distributed Systems,” *Journal of Systems and Software*, vol. 73, no. 3, pp. 551–561, 2004.

[41] E. S. Abuelyaman, "An Optimized Scheme for Vertical Partitioning of a Distributed Database," *Int'l Journal of Computer Science and Network Security*, Vol.8, No.1, pp 310-316, 2008.

[42] C. I. Ezeife and K. Barker, "Vertical Fragmentation for Advanced Object Models in a Distributed Object Based System," *Proc. 7th Int'l Conf. Computing and Information*, pp. 613–632, 1995.

[43] G. S. Chinchwadkar and A. Goh, "An Overview of Vertical Partitioning in Object Oriented Databases," *The Computer Journal*, vol. 42, no. 1, pp. 39–50, 1999.

[44] K. Karlapalem and Q. Li, "Partitioning Schemes for Object Oriented Databases," *Proc. 5th IEEE Int'l Workshop on Research Issues in Data Engineering- Distributed Object Management (RIDE-DOM)*, pp. 42–49, 1995.

[45] K. Karlapalem, Q. Li, and S. Vieweg, "Method Induced Partitioning Schemes for Object-Oriented Databases," *Proc. Int'l Conf. Distributed Computing Systems*, pp. 377–384, 1996.

[46] K. Karlapalem and Q. Li, "A Framework for Class Partitioning in Object-Oriented Databases," *Distributed and Parallel Databases*, vol. 8, no. 3, pp. 333–366, 2000.

[47] E. Malinowski and S. Chakravarthy, "Fragmentation Techniques for Distributing Object-Oriented Databases," *Proc. Int'l Conf. Conceptual Modeling*, pp. 347–360, 1997.

[48] C. I. Ezeife and K. Barker, "Distributed Object Based Design: Vertical Fragmentation of Classes," *Distributed and Parallel Databases*, vol. 6, no. 4, pp. 317–350, 1998.

[49] W. W. Chu, "Optimal File Allocation in a Multiple Computer System," *IEEE Trans. Computers*, vol. 18, no. 10, pp. 885–889, 1969.

[50] R. G. Casey, "Allocation of Copies of Files in an Information Network," *Proc. of AFIPS SJCC*, vol. 40, pp. 617–625, 1972.

[51] S. Mahmoud and J. S. Riordon, "Optimal Allocation of Resources in Distributed Information Networks," *ACM Trans.Database Systems (TODS)*, vol. 1, no. 1, pp. 66–78, 1976.

[52] I. Ahmad, K. Karlapalem, Y.-K. Kwok, and S.-K. So, "Evolutionary Algorithms for Allocating Data in Distributed Database Systems," *Distributed and Parallel Databases*, vol. 11, no. 1, pp. 5–32, 2002.

[53] M.-S. Menon, "Allocating Fragments in Distributed Databases," *IEEE Trans. Parallel and Distributed Systems*, vol. 16, no. 7, pp. 577–585, 2005.

[54] Y.-F. Huang and J.-H. Chen, "Fragment Allocation in Distributed Database Design," *Information Science and Engineering*, vol. 17, pp. 491–506, 2001.

[55] P. Surmsuk, "The Integrated Strategic Information System Planning Methodology," *IEEE Computer Society Press*, pp. 467-475, 2007.

[56] J. L. Whitten and L. D. Bentley, *Systems Analysis and Design Methods.* McGraw-Hill, 2004.

[57] K. E. Wiegers, *Software Requirements.* Microsoft Publication, 2003.

[58] A. Silberschatz, H. F. Korth, and S. Sudarshan, *Database Systems Concepts.* McGraw-Hill, 2001.

[59] C. S. Mullins, "Defining Database Performance," http://www.craigsmullins.com/cnr_db.htm, 2010.

[60] G. Fritchey and S. Dam, *SQL Server 2008 Query Performance Tuning Distilled.* Apress, 2009.

Table of Contents

List of Figures

List of Tables