A Major Project Report on

A Fragmentation Technique in Distributed Databases & Its Implementation

Submitted in partial fulfillment of the

Requirements for the award of the degree

Of

MASTER OF ENGINEERING

In

COMPUTER TECHNOLOGY & APPLICATIONS

By

SHASHI KANT SHARMA

(14/CTA/09)

(University Roll no - 8553)

Under the Guidance of

Dr. Rajni Jindal

DEPARTMENT OF COMPUTER ENGINEERING
DELHI COLLEGE OF ENGINEERING
BAWANA ROAD, DELHI-110042
DELHI UNIVERSITY

Certificate

This is to certify that the major project entitled **"A Fragmentation Technique in Distributed Databases & Its Implementation"** is the work of **Shashi Kant Sharma** (university roll no - 8553), a student of Delhi College of Engineering. This work was completed under my direct supervision and guidance and forms a part of Master of Engineering (Computer Technology & Applications) course and curriculum for the academic year 2009-2011. The matter embodied in this project has not been submitted earlier for the award of any degree or diploma to the best of my knowledge and belief. He has completed his work with utmost sincerity and diligence.

(Dr. Rajni Jindal)
Associate Professor and Project Guide
Department of Computer Engineering
Delhi College of Engineering

Acknowledgement

It gives me a great pleasure to express my profound gratitude to my project guide **Dr. Rajni Jindal,** Associate professor**,** Department of Computer Engineering, Delhi College of Engineering, for her valuable and inspiring guidance throughout the progress of this project. At the same time, I would like to extend my heartfelt thanks to **Dr. Daya Gupta,** Head of the department, Department of Computer Engineering, Delhi College of Engineering, for keeping the spirits high and clearing the visions to work on the project.

Also I would like to thank **Mrs. Narendra Bandarupalli,** Team lead**,** Global Site Selector, Cisco, for his constant support and encouragement. I would like to thank **Mrs. Amit Gurkha**, Software Development Manager, Global Site Selector, Cisco, for making all the resources available and providing healthy environment for the successful completion of the project.

Shashi Kant Sharma
(14/CTA/09)

Table of Contents

***Certificate*..(ii)**
***Acknowlegement*...(iii)**
***Table of Contents*..1**
***List of Figures*...4**
***List of Tables*..5**
***Abstract*...6**
CHAPTER 1 7
Problem Definition 7
1.1 Overview 7
1.2 Related Work & Motivation 8
1.3 Solution Flow 9
1.4 Organization of Dissertation 11
CHAPTER 2 13
Distributed Databases 13
2.1 Introduction 13
2.2 Fragmentation 14
2.3 Allocation 16
2.4 Advantages of Distributed Databases 17
2.5 Disadvantages of Distributed Databases 18
CHAPTER 3 19
Global Site Selector 19
3.1 General Idea 19
3.2 DNS Routing 20
3.2.1 DNS Name Servers 21
3.2.2 DNS Structure 21
3.2.3 Request Resolution 22
3.3 Globally Load Balancing with the GSS 23
3.4 GSS Architecture 26
3.5 GSS Mesh 27
3.5.1 Primary GSSM 27
3.5.2 GSS 28
3.5.3 Standby GSS 28
3.6 Traffic Management Load Balancing 28
3.6.1 DNS Sticky GSLB 29
3.6.2 Network Proximity GSLB 30

Chapter 4 32
Horizontal Fragmentation 32
4.1 Basic Approach to Horizontal Fragmentation 32
4.1.1 Input & Output Definitions 33
4.1.2 Algorithm 33
4.1.3 Summing up 36
4.2 Graphical Approach For HF 38
4.2.1 Definitions 38
4.2.2 Example 38
4.2.3 Creating affinity graph 40
4.2.4 Rules for Selecting the Edges 40
4.2.5 Algorithm 41
4.2.6 Step by Step Solution 42
4.2.7 Optimization of Results 47
4.2.8 Advantages 48
4.2.9 Disadvantages 48
4.3 Advanced Horizontal Fragmentation Technique 49
4.3.1 Attribute Locality Precedence 49
4.3.2 MCURD Matrix 49
4.3.3 Algorithm 49
4.3.4 Example 52
4.3.5 Construction of MCRUD Matrix 53
4.3.6 Calculation of ALP 53
4.3.7 Construction of ALP Table 54
4.3.8 Generation of Predicate Set 54
4.3.9 Fragmentation of Relation 54
4.3.10 Addition of a New Site to DDBMS 55
CHAPTER 5 57
Vertical Fragmentation 57
5.1 Bond Energy Algorithm & Binary Tree Partitioning 58
5.1.1 Some Definitions 58
5.1.2 Example 59
5.1.3 Bond Energy Algorithm 59
5.1.4 Binary Vertical Partitioning 61
5.1.5 Drawback 65
5.2 Graph-Partitioning Algorithm 67
5.2.1 Example 67
5.2.2 Step by Step Solution 68
5.2.3 Advantages 73
5.2.4 Disadvantages 73
5.3 Heuristic Approach for VF 74
5.3.1 Notations and Definitions 74
5.3.2 Algorithm 76
5.3.3 Example 77

5.3.4 Advantages of this Approach 79

CHAPTER 6 80

Grid Fragmentation 80

6.1 Representation Scheme for Fragments 80
6.2 Methodology 82
6.3 Advantages of Our proposal 83
6.4 Allocation of Grid Cells 83

CHAPTER 7 85

Implementation & Results 85

7.1 Configuring GSS 86
7.2 Results 86
7.2.1 Observations 88

Chapter 8 90

Conclusion & Future Work 90

8.1 Conclusion 90
8.2 Future Scope 91

References 90

List of Figures

FIGURE 1 : SOLUTION DESIGN FLOW I 10
FIGURE 2 : SOLUTION DESIGN FLOW II........10
FIGURE 3 : DISTRIBUTED DATABASES........14
FIGURE 4 : (A) HV FRAGMENTATION & (B) VH FRAGMENTATION 15
FIGURE 5 : FRAGMENTATION AND ALLOCATION........16
FIGURE 6 : DOMAIN NAME SPACE 20
FIGURE 7 : DNS REQUEST RESOLUTION 23
FIGURE 8 : GSLB USING THE CISCO GLOBAL SITE SELECTOR........25
FIGURE 9 : GSS ARCHITECTURE 27
FIGURE 10 : PREDICATES & FRAGMENTS........36
FIGURE 11 : ENCLOSED AND ENCLOSING CYCLE 38
FIGURE 12 : STEP BY STEP SOLUTION OF GRAPHICAL APPROACH FOR HF........42
FIGURE 13 : FLOW CHART FOR FINDING PLP........51
FIGURE 14 : SITE MAP FOR BANKING SYSTEM. 52
FIGURE 15 : ADDITION OF NEW SITE TO MAP 55
FIGURE 16 : ORDERING ATTRIBUTES USING BEA 60
FIGURE 17 : GRAPHICAL METHOD FOR VF 68
FIGURE 18 : REPRESENTATION OF GRID CELLS........81
FIGURE 19 : DIFFERENT TECHNIQUES OF FRAGMENTATION STUDIED........82
FIGURE 20 : GRID ALLOCATION SCHEME........84
FIGURE 21 : GRAPH SHOWING THE COMPARISON OF FRAGMENTATION TECHNIQUES.88

List of Tables

TABLE 1 : PREDICATE USAGE MATRIX 39
TABLE 2 : PREDICATE AFFINITY MATRIX 40
TABLE 3 : ACCOUNT RELATION 52
TABLE 4 : MCRUD MATRIX 53
TABLE 5 : ALP 54
TABLE 6 : RELATION FOR SITE IN DELHI 54
TABLE 7 : RELATION FOR SITE IN BANGALORE 54
TABLE 8 : RELATION FOR SITE IN MUMBAI 55
TABLE 9 : RELATION FOR SITE IN DELHI 1 56
TABLE 10 : RELATION FOR SITE IN DELHI 2 56
TABLE 11 : RELATION FOR SITE IN BANGALORE 56
TABLE 12 : RELATION FOR SITE IN MUMBAI 56
TABLE 13 : ATTRIBUTE USAGE MATRIX 59
TABLE 14 : ATTRIBUTE AFFINITY MATRIX 59
TABLE 15 : CLUSTERED AFFINITY MATRIX 61
TABLE 16 : CLUSTERED AFFINITY MATRIX 62
TABLE 17 : SPLITTING THE RELATION BY BVP 63
TABLE 18 : EXAMPLE CLUSTERED AFFINITY MATRIX FOR SPLIT 1 65
TABLE 19 : ATTRIBUTE USAGE MATRIX 67
TABLE 20 : ATTRIBUTE AFFINITY MATRIX 68
TABLE 21 : ATTRIBUTE USAGE FREQUENCY MATRIX 77
TABLE 22 : ATTRIBUTE *REQUEST* MATRIX 78
TABLE 23 : TRANSPORTATION COST FACTORS 78
TABLE 24 : ATTRIBUTE *PAY* MATRIX 78
TABLE 25 : ATTRIBUTE ALLOCATION 78
TABLE 26 : RESPONSES RECEIVED PER SECOND 87

Abstract

The distributed database design is an optimization problem that includes various sub-problems: data fragmentation, allocation and local optimization. Each of these can be solved in various ways. In context of this research we address fragmentation and allocation problems simultaneously for distributed databases management systems (DDBMS) for enhancing the process of server load balancing using Cisco Global Site Selector (GSS).

Fragmentation is mainly of three types namely: horizontal, vertical and hybrid (grid) fragmentation. Although there is large work carried out on the design of data fragmentation but most of them are either horizontal or vertical. The core of this thesis defines a new type of grid fragmentation technique in the context of relational databases. Our proposal combines horizontal fragmentation (HF) on the basis of attribute locality precedence and cost model of vertical fragmentation (VF) to generate grid fragments. Both of these two techniques also address the problem of fragment allocation in distributed databases. As part of whole process we discuss and compare various other algorithms for generating candidate vertical and horizontal partition schemes.

In the end, we compare our approach with one of the graphical grid fragmentation methods by implementing these techniques as part of Cisco GSS, a product for enhancing the DNS resolution process. Results shows that proposed grid fragmentation is a superior approach and it can solve fragmentation and allocation problem of relational database systems properly.

CHAPTER 1

Problem Definition

1.1 Overview

Now a day's size of databases is so huge that it's almost impossible to keep all the data at one place. Even the organizations are spreading their businesses all around the world, which will create bottleneck problem with that single database server. Many large organizations of different types have a history of separate business units developing and maintaining independent customer databases. Typically these legacy systems have been developed autonomously and use a variety of data structures and identifiers to record personal information. In addition, these databases are often 'owned and operated' by separate functional units within the organization. Consequently, the personal information an organization holds about individuals will be fragmented across a number of databases using a variety of different data structures. This makes accessing and collating personal information difficult and time-consuming. Rarely in these cases is there a unified and consolidated view of the information an organization holds about an individual.

This work has been carried out as part of Cisco Global Site Selector (GSS), which is a server load balancing device. GSS offloads the traditional DNS servers by taking control of the DNS resolution process for parts of organizations domain name space. Basically

GSS performs DNS request resolutions and selects one of the least loaded and nearest server in terms of round trip time (or some other network parameters) and returns its address to the requesting client. GSS works in a mesh[1] of 16 GSS that can cover all the data-centers all around world. Whole mesh works like a single cohesive unit, where each GSS coordinates with the PGSSM[2]. In the backend, GSS has various hugely populated and ever-growing databases. Each request resolution in the whole mesh involves various database operations. To improve the request resolution time we need to optimize these database operations. So we create distributed data bases where different GSS can access it without interfering with one another. It may be stored in multiple computers located in the same physical location, or may be dispersed over a network of interconnected computers. One problem associated with data quality that erodes GSS's ability is – the fragmentation of these databases across multiple databases.

Now, the design of distributed database is an optimization problem and the resolution of several sub problems such as data fragmentation (horizontal, vertical, and hybrid), data allocation (with or without redundancy), optimization and allocation of operations (request transformation, selection of the best execution strategy, and allocation of operations to sites). There are some different approaches to solve each problem, so this means that the design of the distributed databases becomes cumbersome. There are many researches connected to the data fragmentation and they are presented both in the case of relational database and in the case of object-oriented database. Here we present a new technique for grid fragmentation in distributed databases, especially to solve the problem of fragmentation and allocation in distributed databases.

1.2 Related Work & Motivation

The concept of using fragmentation and allocation of data as means of improving the performance of database management systems has often emerged in the literature. Most of the past research considers either horizontal fragmentation [9] [10] [11] schemes or vertical fragmentation schemes [1] [4] [7] [12].

Horizontal partitioning using min-term predicate is first proposed by Certi et. Al [10]. Ra presented a graphical algorithm for HF in which predicates are clustered on the basis of predicate affinity [11]. Similar graphical approach was proposed for VF by S.B.Navathe et al given in [1]. Applying Bond Energy algorithm and Binary Partitioning to perform vertical fragmentation is studied in [12]. Marwa et al. (2008) uses the instance request matrix to horizontally fragment object oriented database [13]. Abuelyaman (2008)

[1] Mesh is the interconnected network of GSS.
[2] PGSSM – Primary GSS which controls and coordinates the complete mesh.

proposed a static algorithm StatPart for VF [14]. Mahboubi H. and Darmont J. (2009) used predicate affinity for HF in data warehouse [15].

Shamkant B. Navathe, K Karlapalemand M. Ra proposed a mixed fragmentation methodology by means of graphical approach to both HF and VF in [5]. This uses the graphical approach using partitioning algorithm given in [1] for both horizontal and vertical partitioning. This is not a cost model hence it does not deal with the problem of allocation. Allocation of fragments to sites would normally involve a cost model. To the best of our knowledge no such hybrid fragmentation approach exists that performs fragmentation and allocation simultaneously. Our emphasis in this paper is a grid fragmentation which is a type of mixed or hybrid fragmentation. We have developed a cost model that addresses the problem of allocation in distributed databases. Our methodology uses heuristic approach [7] for vertical fragmentation by H. Ma, KD Schewe and M. Kirchberg and horizontal fragmentation on the basis of query frequency and cost of operation [9] by S. I. Khan & A.S.M. Latiful Hoque. Both of these techniques do allocation of fragments simultaneously with fragmentation. This provides us the flexibility of allocating fragments either by using one of them or both of them. We will discuss this in detail in later chapters.

1.3 Solution Flow

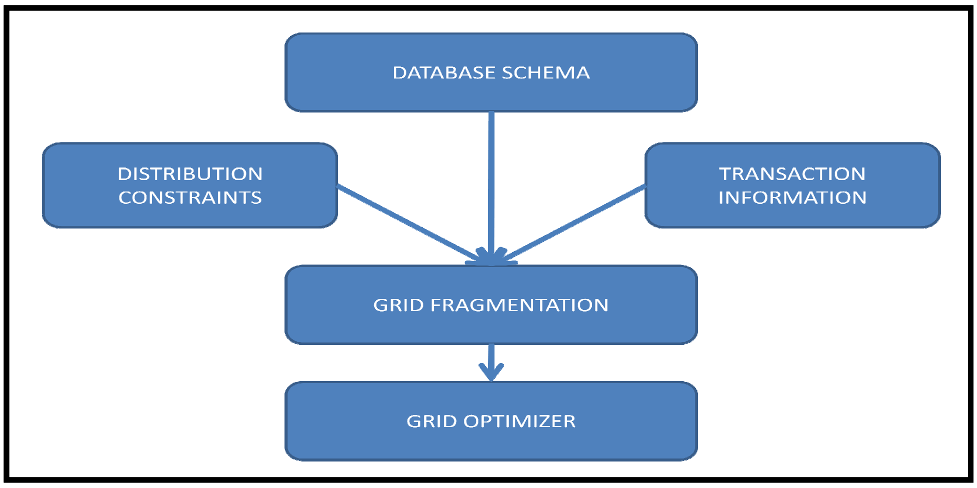
The complete distributed database design is a three step process, namely: initial design, application of design and redesign. Initial design consists of selecting fragmentation and allocation algorithms. Second and third step are iterative processes that depends upon logical and physical changes in the distributed database environment.

In this thesis we use a similar methodology as given in [5], for generating a hybrid fragmentation scheme for the DDBMS to optimize the GSS operations. Figure 1 shows outline of our proposal where allocation is handled along with Fragmentation. Input consists of a set of relations, together with information about the important transactions on the proposed database. It is not necessary to collect information about 100% of transactions. According to 80-20 rule [4], 20% of heavily used transactions account for the 80% of database activity. Hence we provide information about 20% of heavily used transaction. Input to the fragmentation techniques is defines as follows:

- Schema information includes relations, attributes, cardinalities, attribute sizes, predicates used by the database operations.
- Information on transactions includes type (read or write), frequency, attribute usage and predicates usage.

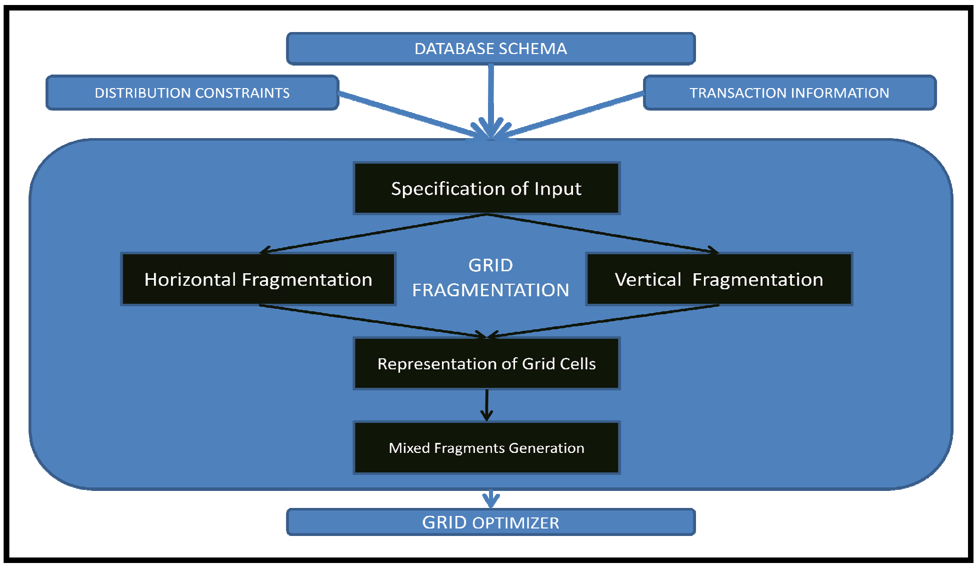
- Other input consists of preferences or special considerations that would influence the fragmentation and allocation, like predetermined partitions or fixed allocation.

Figure 1 : Solution Design Flow I



After gathering input, next is the grid fragmentation which is composed of two modules, namely: vertical fragmentation and horizontal fragmentation, as shown in the figure 2. Horizontal and vertical fragmentation can be done concurrently. We will explain various methods for HF and VF that will result into various ways to perform grid fragmentation. A scheme for representation of grid cells and the binary operations is developed in [5]. Grid optimizer merges the grid cells by applying various binary operations to the grid cells. This is anti-fragmentation but this applies to those fragments or grid cells which belong to the same site. Grid optimizer is not discussed as part of this thesis. It will be considered in future work.

Figure 2 : Solution Design Flow II



1.4 Organization of Dissertation

The rest of this thesis is organized as follows. In chapter 2, we have presented literature review for distributed databases. It describes the various distributed databases design techniques.

Chapter 3 introduces global site selector (GSS), a Cisco product that improves the DNS request resolution with many added features. It is a Server Load Balancer (SLB) device that can balance content requests among two or more servers containing the same content. Server load-balancing devices ensure that the content consumer is directed to the host that is best suited to handle that consumer's request. This work has been carried out as part of GSS to improves the working of GSS by fragmenting the underlying various GSS's databases. We will discuss the traditional DNS request resolution and same using GSS. We clarify the need for fragmentation with a brief overview about server load balancing features.

We start with our core work in chapter 4. Mainly we will explain three horizontal fragmentation schemes. First one is a basic technique that uses min-term predicate and optimizes these min-terms by removing the redundant and non-satisfactory predicates. Second approach for HF is a graphical approach for forming clusters using predicate affinity. We have modified and simplified the partitioning algorithm given in [1], which is to be used for this graphical approach. For reference we call this algorithm as Graph Partitioning Algorithm. Thirdly we explain a new approach for HF [9] using MCURD matrix[1] to generate attribute locality precedence. We presented an example to explain each of the above techniques.

Moving to the chapter 5 we discuss three approaches for vertical fragmentation. First one uses Bond Energy Algorithm [2] to group the attributes of a relation based on Attribute Affinity Matrix[2] and then using Binary Partitioning [3] algorithm to partition the relation. Second method is the same used for HF. Graphical approach using graph partitioning algorithm that partition the relation using attribute affinity. Thirdly we discussed a heuristic approach for VF that uses a attribute usage matrix and cost model. We presented an example to explain each of the above techniques.

[1] A data-to-location MCURD matrix is a table of which rows indicate predicates of the entries of a relation and columns indicate different locations.
[2] Attribute Affinity Matrix specifies the use of attributes by the transaction and there access frequency.

In chapter 6 we introduce how to perform grid fragmentation. We present representation of the grid cells and the various binary operations over them. In the end we discuss when to perform allocation of fragments.

In chapter 7, we start with brief review about implementation. We have shown implementation results of six fragmentation techniques as part of Cisco Global Site Selector. Finally we discuss various observations about the result.

Finally we conclude with a short summary and the future modifications of the proposed methodology in chapter 8.

CHAPTER 2

Distributed Databases

In the previous chapter we discussed about our approach and motivation behind our work. This chapter explains the various aspects of a distributed database design problem, advantages and disadvantages of distributed databases.

2.1 Introduction

A distributed database is a database that is under the control of a central database management system (DBMS) in which storage devices are not all attached to a common CPU. It may be stored in multiple computers located in the same physical location, or may be dispersed over a network of interconnected computers. Collections of data in a distributed database can be distributed across multiple physical locations. A distributed database can reside on network servers on the Internet, on corporate intranets or extranets, or on other company networks. Primary concern of distributed database system design is to making fragmentation of the relations in case of relational database or classes in case of object oriented databases, allocation and replication of the fragments in different sites of the distributed system, and local optimization in each site.

The fragmentation and allocation of databases improves database performance at end-user worksites. We can influence communication costs, load balancing and availability by fragmenting a relation and allocating it accordingly using an optimized approach. Fragmentation decomposes a relation into smaller, disjunctive fragments. These fragments are distributed across nodes or may be replicated. Replication involves using specialized software that looks for changes in the distributive database. Once the changes have been identified, the replication process makes all the databases look the same. The replication process can be very complex and time consuming depending on the size and number of the distributive databases. This process can also require a lot of time and computer resources. On the other hand allocation deals with keeping the fragments at the sites where they are required most. Assigning fragments to sites in the computer network depends upon communication cost and transaction information. Using a cost model we can efficiently minimize the cost of remote access and avoid bottleneck problem. In this thesis we are concentrating on fragmentation and allocation schemes.

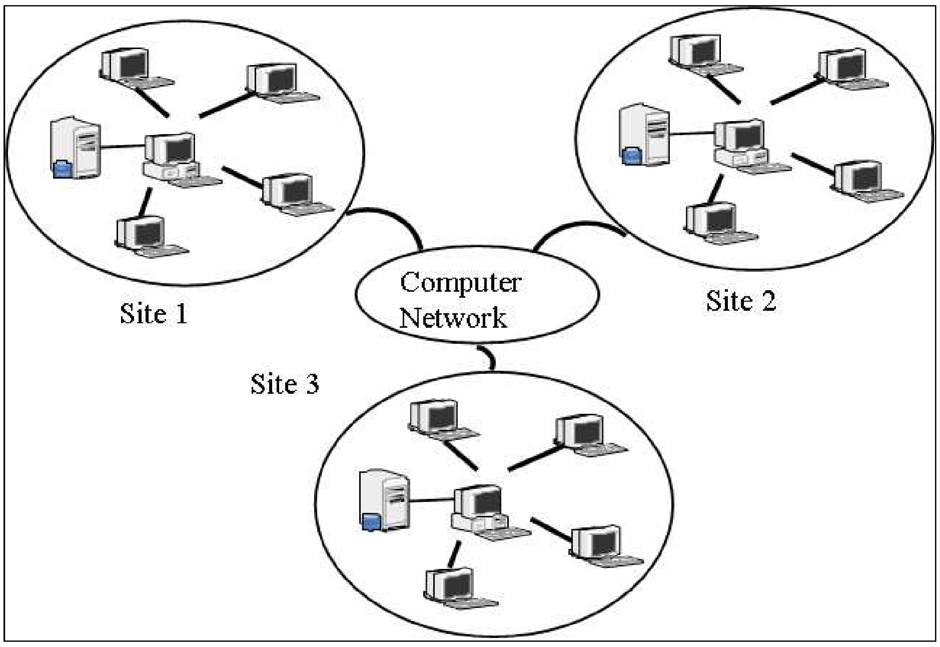


Figure 3 : Distributed Databases

The design of distributed databases is an optimization problem requiring solutions to following two problems:

- Designing the fragmentation of global relations
- Designing the allocation of fragments to the sites of communication network

2.2 Fragmentation

Distributed processing on database management systems (DBMS) is an efficient way of improving performance of applications that manipulate large volumes of data. This may be accomplished by removing irrelevant data accessed during the execution of queries and by reducing the data exchange among sites, which are the two main goals of the design of distributed databases. The main reasons of fragmentation of the relations are to

- increase locality of reference of the queries submitted to database
- improve reliability and availability of data and performance of the system, balance storage capacities and minimize communication costs among sites

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) allows a relation or class to be partitioned into disjoint sets of columns or attributes except the primary key.
- Combination of horizontal and vertical fragmentations to **grid** or **mixed** or **hybrid fragmentations** (MF) are also proposed, which have properties of both. To perform hybrid fragmentation we can perform HF first and then VF (HV fragmentation) or vice versa (VH fragmentation) as shown in the figure 4. Final result is independent of the order of HF and VF.

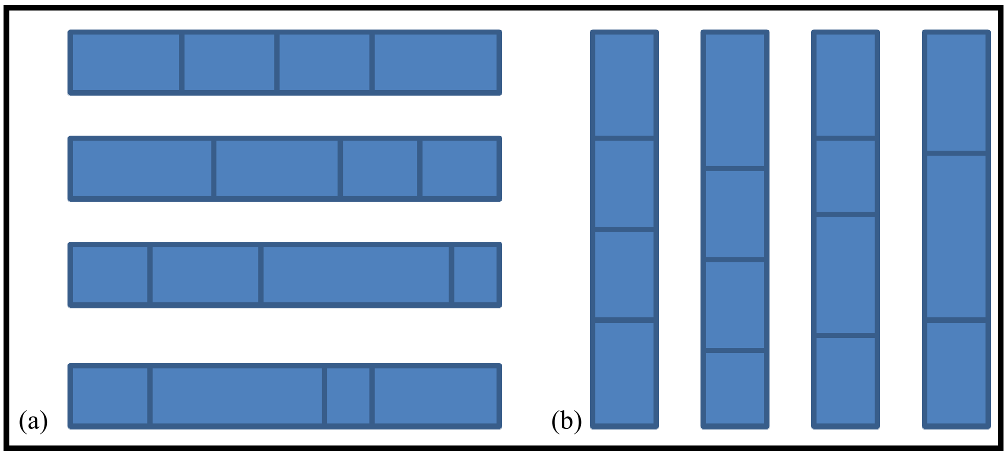


Figure 4 : (a) HV fragmentation & (b) VH fragmentation

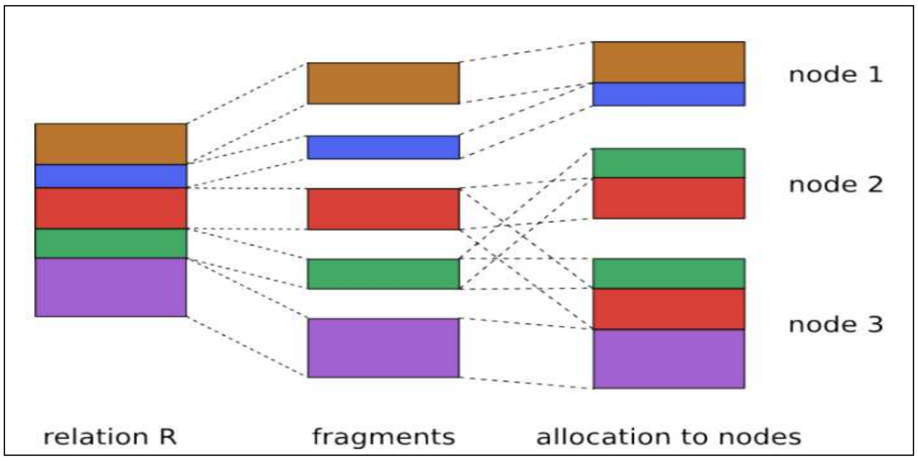
The problem of fragmenting the database is difficult one in itself and variety of approaches exist for fragmenting the database. Many approaches for vertical and horizontal have been researched before. Some of them are graphical approaches that consider predicate affinity [5] or attribute affinity [1] to form clusters. A mixed fragmentation technique [5] uses graphical approach for both HF and VF for creating grid cells. . In this thesis we propose a new type of Grid Fragmentation method which uses cost model for both HF and VF and performs allocation simultaneously. It uses query frequency and cost of a query to calculate attribute locality precedence for HF [9]. For VF it uses cost of allocating a particular attribute to particular site on the basis of transaction information [7].

2.3 Allocation

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

The problem of allocating data in a distributed database system has an important impact upon the performance and reliability of the system as a whole. The aim is to store the fragments closer to where they are more frequently used in order to achieve best performance.

Figure 5 : Fragmentation and Allocation



So, one key principle in distribution design is to achieve maximum locality of data and applications. Since, distributed databases enable more sophisticated communication between sites; the major motivation for developing a distributed database is to reduce communication by allocating data as close as possible to the applications which use them. Thus in a well-designed distributed database 90 percent of the data should be found at the local site, and only 10 percent of the data should be accessed on a remote site. A poorly designed data allocation can lead to inefficient computation, high access cost and high network loads. Various approaches have already evolved for allocation of data in distributed database. In most of these approaches, data allocation has been proposed prior to the design of a database depending on some static data access patterns and/or static query patterns. In a static environment, where the access probabilities of nodes to fragments never change, a static allocation of fragments provides the best solution. However, in a dynamic environment where these probabilities change over time, the static allocation solution would degrade the database performance.

2.4 Advantages of Distributed Databases

There are multiple advantages of DDB over a CDB. Few of them are listed below.

- Management of distributed data with different levels of transparency.
- Increase reliability and availability.
- Easier expansion.
- Reflects a proper organizational structure — database fragments are located in the departments which they relate to.
- Local autonomy — a department can control the data about them (as they are the ones familiar with it.)
- Protection of valuable data — if there were ever a catastrophic event such as a fire, all of the data would not be in one place, but distributed in multiple locations.
- Improved performance — data is located near the site of greatest demand, and the database systems themselves are parallelized, allowing load on the databases to be balanced among servers. (A high load on one module of the database won't affect other modules of the database in a distributed database.)
- Economics — it costs less to create a network of smaller computers with the power of a single large computer.
- Modularity — systems can be modified, added and removed from the distributed database without affecting other modules (systems).
- Reliable transactions - Due to replication of database.
- Hardware, OS, N/w, Fragmentation, DBMS, Replication and Location Independence.
- Continuous operation…
- Distributed Query processing.
- Distributed Transaction management.
- Single site failure does not affect performance of system. All transactions follow A.C.I.D. property: a-atomicity, the transaction takes place as whole or not at all; c-consistency, maps one consistent DB state to another; i-isolation, each transaction sees a consistent DB; d-durability, the results of a transaction must

survive system failures. The Merge Replication Method used to consolidate the data between databases.

2.5 Disadvantages of Distributed Databases

There some disadvantages of a DDB. Few of them are listed below.

- Complexity — extra work must be done by the DBAs to ensure that the distributed nature of the system is transparent. Extra work must also be done to maintain multiple disparate systems, instead of one big one. Extra database design work must also be done to account for the disconnected nature of the database — for example, joins become prohibitively expensive when performed across multiple systems.
- Economics — increased complexity and a more expansive infrastructure means extra costs.
- Security — remote database fragments must be secured, and they are not centralized so the remote sites must be secured as well. The infrastructure must also be secured (e.g., by encrypting the network links between remote sites).
- Difficult to maintain integrity — in a distributed database, enforcing integrity over a network may require too much of the network's resources to be feasible.
- Inexperience — distributed databases are difficult to work with, and as a young field there is not much readily available experience on proper practice.
- Lack of standards — there are no tools or methodologies yet to help users convert a centralized DBMS into a distributed DBMS.
- Database design more complex — besides of the normal difficulties, the design of a distributed database has to consider fragmentation of data, allocation of fragments to specific sites and data replication.
- Additional software is required.
- Operating System should support distributed environment.
- Concurrency control: it is a major issue. It is solved by locking and time stamping.

CHAPTER 3

Global Site Selector

In this chapter we will discuss about Cisco Global Site Selector and its key features. At the end of this chapter we explain the demand and application of our fragmentation techniques.

3.1 General Idea

Organizations with a global reach or businesses that provide web and application hosting services require network devices that can perform complex request routing to two or more redundant, geographically dispersed data centers. These network devices need to provide fast response time and disaster recovery and failover protection through global server load balancing (GSLB).

Server load-balancing devices, such as the Cisco Content Services Switch (CSS), Cisco Content Switching Module (CSM), and Cisco Application Control Engine (ACE) that are connected to a corporate LAN or the Internet, can balance content requests among two or more servers containing the same content. Server load-balancing devices ensure that the content consumer is directed to the host that is best suited to handle that consumer's request.

The Cisco Global Site Selector (GSS) platform allows us to leverage global content deployment across multiple distributed and mirrored data locations, optimizing site selection, improving Domain Name System (DNS) responsiveness, and ensuring data center availability.

The GSS is inserted into the traditional DNS routing hierarchy and is closely integrated with the Cisco CSS, Cisco CSM, Cisco ACE, or third-party server load balancers (SLBs) to monitor the health and load of the SLBs in your data centers. The GSS uses this information and user-specified routing algorithms to select the best-suited and least-loaded data center in real time.

The GSS can detect site outages, ensuring that web-based applications are always online and that customer requests to data centers that suddenly go offline are quickly rerouted to available resources. The GSS offloads tasks from traditional DNS servers by taking control of the domain resolution process for parts of your domain name space, responding to requests at a rate of thousands of requests per second.

3.2 DNS Routing

This section explains some of the key DNS routing concepts behind the GSS. Since the early 1980s, content routing on the Internet has been handled using the Domain Name System (DNS), a distributed database of host information that maps domain names to IP addresses. Almost all transactions that occur across the Internet rely on DNS, including electronic mail, remote terminal access such as Telnet, file transfers using the File Transfer Protocol (FTP), and web surfing. DNS uses alphanumeric hostnames instead of numeric IP addresses that bear no relationship to the content on the host. With DNS, you can manage a nearly infinite number of hostnames referred to as the domain name space (see figure 6). DNS allows local administration of segments (individual domains) of the overall database, but allows for data in any segment to be available across the entire network. This process is referred to as *delegation*.

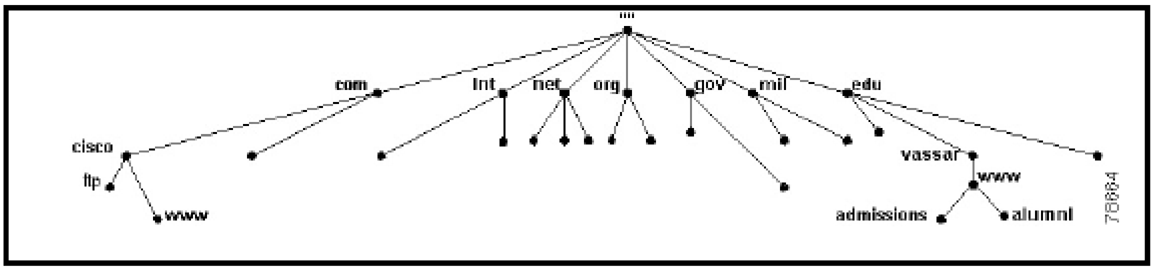


Figure 6 : Domain Name Space

3.2.1 DNS Name Servers

Information about the domain name space is stored on name servers that are distributed throughout the Internet. Each server stores the complete information about its small part of the total domain name space. This space is referred to as a DNS *zone*. A zone file contains DNS information for one domain (“mycompany.com”) or sub-domain (“gslb.mycompany.com”).

The DNS information is organized into lines of information called resource records. Resource records describe the global properties of a zone and the hosts or services that are part of the zone. They are stored in binary format internally for use by the DNS software. However, resource records are sent across the network in a text format while they perform zone transfers.

Resource records are composed of various types of records including:

- Start of Authority (SOA)
- Name Service (NS)
- Address (A)
- Host Information (HINFO)
- Mail Exchange (MX)
- Canonical Name (CNAME)
- Pointer (PTR)

3.2.2 DNS Structure

End users who require data from a particular domain or machine generate a recursive DNS request on their client that is sent first to the local name service (NS), also referred to as the *D-proxy*. The D-proxy returns the IP address of the requested domain to the end user.

The DNS structure is based on a hierarchical tree structure that is similar to common file systems. The key components in this infrastructure are as follows:

- **DNS Resolvers**—Clients that access client name servers.
- **Client Name Server**—Server that runs DNS software that has the responsibility of finding the requested web site. The client name server is also referred to as the client DNS proxy (D-proxy).
- **Root Name Servers**—Server that resides at the top of the DNS hierarchy. The root name server knows how to locate every extension after the period (.) in the hostname. There are many top-level domains. The most common top-level domains include .org, .edu, .net, .gov, and .mil. Approximately 13 root servers worldwide handle all Internet requests.

- **Intermediate Name Server**—Server that is used for scaling purposes. When the root name server does not have the IP address of the authoritative name server, it sends the requesting client name server to an intermediate name server. The intermediate name server then refers the client name server to the authoritative name server.
- **Authoritative Name Server**—Server that is run by an enterprise or outsourced to a service provider and is authoritative for the domain requested. The authoritative name server responds directly to the client name server (not to the client) with the requested IP address.

3.2.3 Request Resolution

If the local D-proxy does not have the information requested by the end user, it sends out iterative requests to the name servers that it knows are authoritative for the domains close to the requested domain. For example, a request for www.cisco.com causes the local D-proxy to check first for another name server that is authoritative for www.cisco.com. Figure 7 summarizes the sequence performed by the DNS infrastructure to return an IP address when a client tries to access the www.cisco.com website.

1. The resolver (client) sends a query for www.cisco.com to the local client name server (D-proxy).
2. The local D-proxy does not have the IP address for www.cisco.com so it sends a query to a root name server (".") asking for the IP address. The root name server responds to the request by doing one of the following:
    a. Referring the D-proxy to the specific name server that supports the .com domain.
    b. Sending the D-proxy to an intermediate name server that knows the address of the authoritative name server for www.cisco.com. This method is referred to as an iterative query.
3. The local D-proxy sends a query to the intermediate name server that responds by referring the D-proxy to the authoritative name server for cisco.com and all the associated sub-domains.
4. The local D-proxy sends a query to the cisco.com authoritative name server that is the top-level domain. In this example, www.cisco.com is a sub-domain of cisco.com, so this name server is authoritative for the requested domain and sends the IP address to the name server (D-proxy).

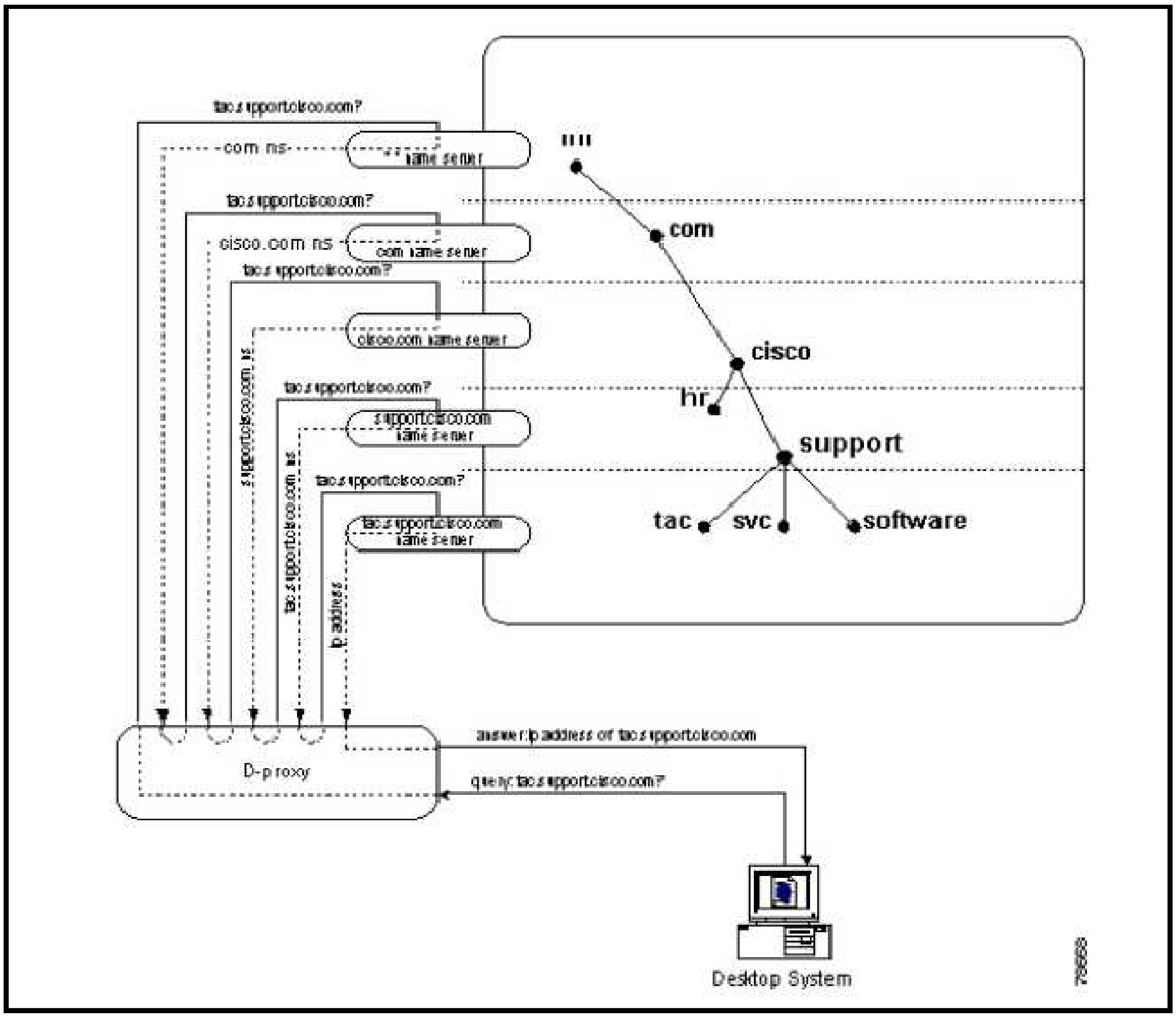


Figure 7 : DNS Request Resolution

5. The name server (D-proxy) sends the IP address (172.16.56.76) to the client browser. The browser uses this IP address and initiates a connection to the www.cisco.com website.

3.3 Globally Load Balancing with the GSS

The GSS addresses critical disaster recovery requirements by globally load balancing distributed data centers. The GSS coordinates the efforts of geographically dispersed SLBs in a global network deployment for the various other Routing and Switching Cisco products.

The GSS supports over 4000 separate virtual IP (VIP) addresses. It coordinates the activities of SLBs by acting as the authoritative DNS server for those devices under its control.

Once the GSS becomes responsible for GSLB services, the DNS process migrates to the GSS. The DNS configuration is the same process as described in the "Request Resolution" section. The only exception is that the NS-records point to the GSSs located at each data center. The GSS determines which data center site should receive the client traffic.

As the authoritative name server for a domain or sub-domain, the GSS considers the following additional factors when responding to a DNS request:

- **Availability** — Servers that are online and available to respond to the query
- **Proximity** — Server that responded to a query most quickly
- **Load** — Type of traffic load handled by each server in the domain
- **Source of the Request** — Name server (D-proxy) that requests the content
- **Preference** — First, second or third choice of the load-balancing algorithm to use when responding to a query

This type of global server load balancing ensures that the end users are always directed to resources that are online, and that requests are forwarded to the most suitable device resulting in faster response time for users.

When resolving DNS requests, the GSS performs a series of distinct operations that take into account the resources under its control and return the best possible answer to the requesting client's D-proxy.

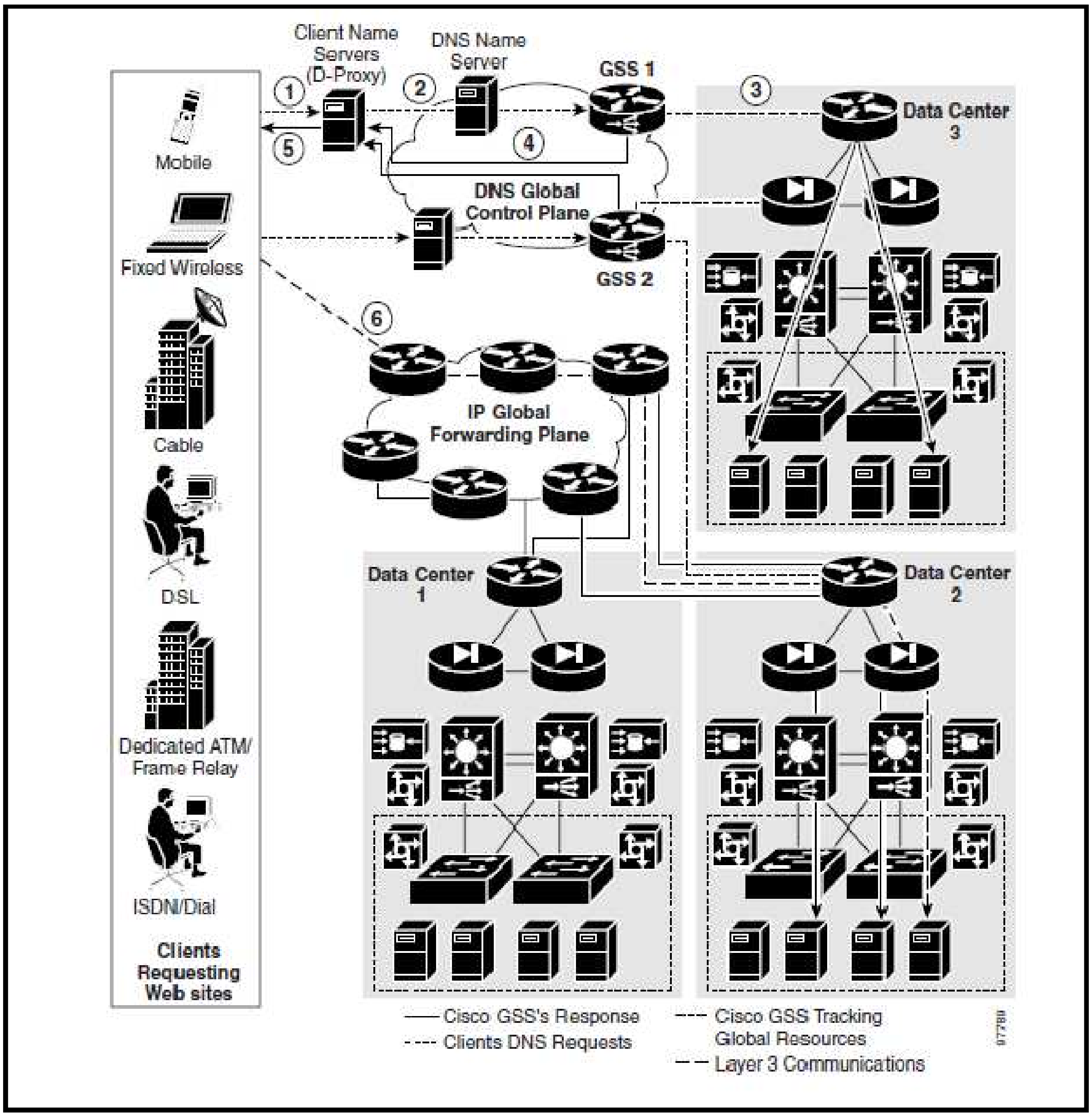


Figure 8 : GSLB Using the Cisco Global Site Selector

Figure 8 outlines how the GSS interacts with various clients as part of the website selection process to return the IP address of the requested content site.

1. A client starts to download an updated version of software from www.cisco.com and types **www.cisco.com** in the location or address field of the browser. This application is supported at three different data centers.
2. The DNS global control plane infrastructure processes the request and the request arrives at a GSS device.
3. The GSS sends the IP address of the "best" server load balancer to the client, in this case the SLB at Data Center 2.
4. The web browser processes the transmitted IP address.

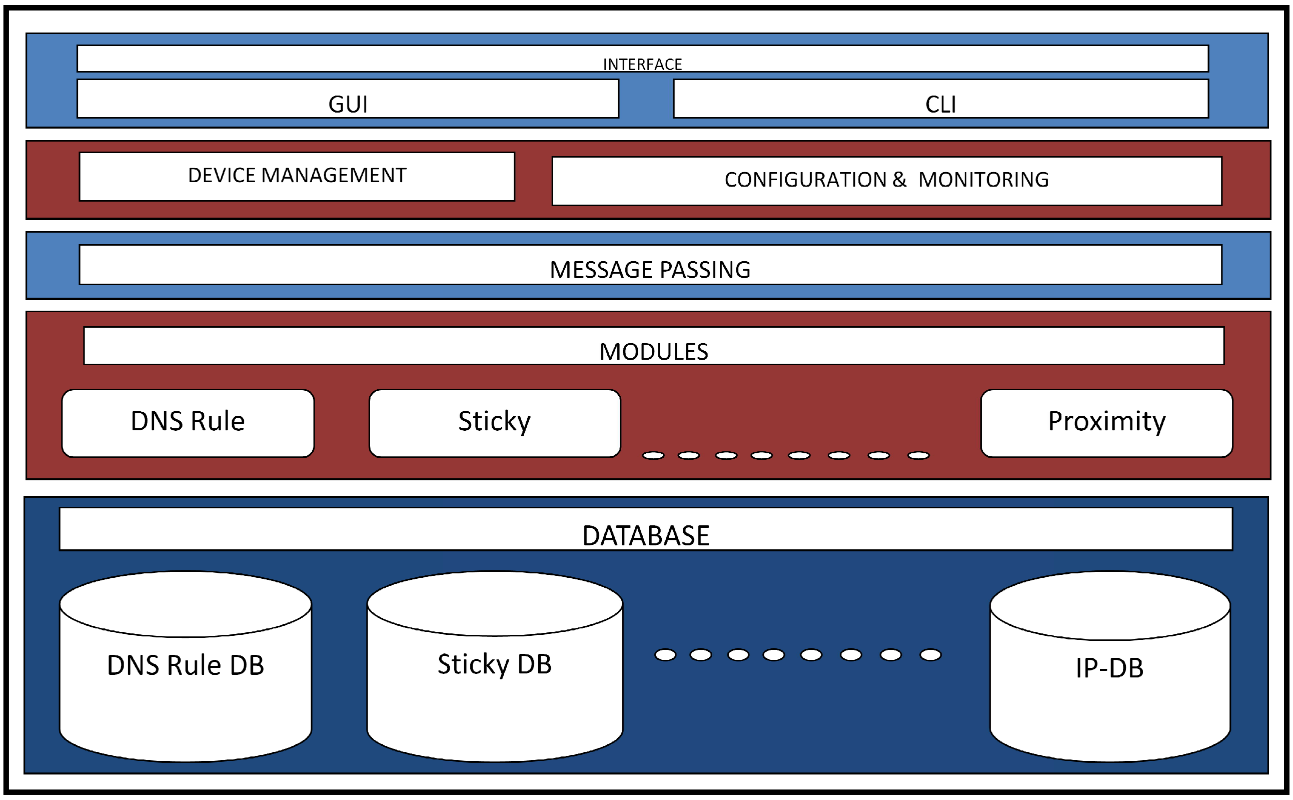
5. The client is directed to the SLB at Data Center 2 by the IP control and forwarding plane.
6. The GSS offloads the site selection process from the DNS global control plane. The request and site selection are based on the load and health information with user-controlled load-balancing algorithms. The GSS selects in real time a data center that is available and not overloaded.

3.4 GSS Architecture

The architecture of GSS is a layered structure where each layer has a specific function as shown in the figure 9. For configuration and control, administrators have a Cisco IOS Software-like command-line interface (CLI) and an intuitive, embedded GUI. The CLI and the GUI is used for the configuration of all global load-balancing parameters. All the DNS request that come, go to monitoring module which filter the incoming packet and send them to different modules depending upon the kind of request. One request may be processed by more than one module. One particular answer is selected and returned to D-Proxy/Client as per the requirement. Most of the modules have an underlying database. For example Sticky module stores all the entries in a particular database. It returns an answer on the basis of the entries stored in the sticky database. Similarly we have one database for DNS rules and each answer is selected by matching the request parameters with the DNS rule parameter. Each DNS rule in itself is very big with more than 20 parameters in it. Hence most of the DNS request resolution time is spent in searching a particular DNS rule in the database.

These databases are the main area of concern for us in this thesis. We propose various techniques to fragment and allocate the database. As discussed size of databases are huge and searching for one entry will make the whole request resolution process a time consuming one. Database searching time is nearly 70% of the DNS request resolution total time. Hence we need to fragment these databases so that we can fasten the DNS resolution process significantly.

Figure 9 : GSS Architecture



3.5 GSS Mesh

Interconnected network of various GSS is called GSS mesh, where each of the GSS works in coordination with the primary GSS. Maximum 16 GSS can be included in a mesh which is enough to cover the data centers all around the world. GSS mesh consists of following three types of GSS.

- Primary GSSM
- GSS
- Standby GSSM

3.5.1 Primary GSSM

The primary GSSM is a GSS that runs the GSS software. It performs content routing in addition to centralized management and shared global server load-balancing functions for the GSS network.

The primary GSSM hosts the embedded GSS database that contains configuration information for all your GSS resources, such as individual GSSs and DNS rules. All connected GSS devices report their status to the primary GSSM.

On the primary GSSM, one monitors and administers GSS devices using either of the following interfaces:

- CLI commands
- GUI (graphical user interface) functions

All configuration changes are communicated automatically to each device managed by the primary GSSM. Any GSS device can serve as the single, primary GSSM on a configured system.

3.5.2 GSS

The GSS runs the GSS software and routes DNS queries based on DNS rules and conditions configured using the primary GSSM. Each GSS is known to and synchronized with the primary GSSM. We manage each GSS individually through its command-line interface (CLI). Support for the graphical-user interface (GUI) is not available on a GSS or on a standby GSSM.

3.5.3 Standby GSS

The standby GSSM is a GSS that runs the GSS software and routes DNS queries based on DNS rules and conditions configured using the primary GSSM. Additionally, the standby GSSM is configured to function as the primary GSSM if the designated primary GSSM goes offline or becomes unavailable to communicate with other GSS devices.

When the standby GSSM operates as the interim primary GSSM, it contains a duplicate copy of the embedded GSS database currently installed on the primary GSSM. Both CLI and GUI support are also available on the standby GSSM once you configure it as the interim primary GSSM. While operating as the primary GSSM, you can monitor GSS behavior and make configuration changes, as necessary.

Any configuration or network changes that affect the GSS network are synchronized between the primary and the standby GSSM so the two devices are never out of sequence.

3.6 Traffic Management Load Balancing

The GSS includes DNS sticky and network proximity traffic management functions to provide advanced global server load-balancing capabilities in a GSS network. DNS sticky ensures that e-commerce sites provide undisrupted services and remain open for business by supporting persistent sticky network connections between customers and e-commerce servers. Persistent network connections ensure that active connections are not interrupted and shopping carts are not lost before purchase transactions are completed.

Network proximity selects the closest or most proximate server based on measurements of round-trip time to the requesting client's D-proxy location, improving the efficiency within a GSS network. The proximity calculation is typically identical for all requests from a given location (D-proxy) if the network topology remains constant. This approach selects the best server based on a combination of site health (availability and load) and the network distance between a client and a server zone.
Note: In context of GSS, Answer or A-record refers to the IPv4 address of the resource located in the datacenter.

3.6.1 DNS Sticky GSLB

Stickiness, also known as persistent answers or answer caching, enables a GSS to remember the DNS response returned for a client D-proxy and to later return that same answer when the client D-proxy makes the same request. When you enable stickiness in a DNS rule, the GSS makes a best effort to always provide identical A-record responses to the requesting client D-proxy, assuming that the original VIP continues to be available.

DNS sticky on a GSS ensures that e-commerce clients remain connected to a particular server for the duration of a transaction even when the client's browser refreshes the DNS mapping. While some browsers allow client connections to remain for the lifetime of the browser instance or for several hours, other browsers impose a connection limit of 30 minutes before requiring a DNS re-resolution. This time may not be long enough for a client to complete an e-commerce transaction.

With local DNS sticky, each GSS device attempts to ensure that subsequent client D-proxy requests to the same domain name to the same GSS device will be stuck to the same location as the first request. DNS sticky guarantees that all requests from a client D-proxy to a particular hosted domain or domain list are given the same answer by the GSS for the duration of a user-configurable sticky inactivity time interval, assuming the answer is still valid.

With global DNS sticky enabled each GSS device in the network shares answers with the other GSS devices in the network, operating as a fully connected peer-to-peer mesh. Each GSS device in the mesh stores the requests and responses from client D-proxies in its own local database and shares this information with the other GSS devices in the network. As a result, subsequent client D-proxy requests to the same domain name to any GSS in the network causes the client to be stuck.

3.6.2 Network Proximity GSLB

The GSS responds to DNS requests with the most proximate answers (resources) relative to the requesting D-proxy. In this context, proximity refers to the distance or delay in terms of network topology (not geographical distance) between the requesting client's D-proxy and its answer.

To determine the most proximate answer, the GSS communicates with a proximity probing agent, a Cisco IOS-based router or another GSS configured as a DRP agent, located in each proximity zone to gather round-trip time (RTT) metric information measured between the requesting client's D-proxy and the zone. Each GSS directs client requests to an available server with the lowest RTT value

The proximity selection process is initiated as part of the DNS rule balance method clause. When a request matches the DNS rule and balance clause with proximity enabled, the GSS responds with the most proximate answer.

The GSS responds to DNS requests with the most proximate answers relative to the requesting D-proxy. In earlier releases, proximity refers to the distance or delay in terms of network topology, not geographical distance, between the requesting client's D-proxy and its answer.

To determine the most proximate answer, the GSS communicates with a probing device, a Cisco IOS-based router, located in each data center (proximity zone) to gather round-trip time (RTT) metric information measured between the requesting client D-proxy and the zone. Each GSS directs client requests to an available server with the lowest RTT value.

While it may often provide the best answer for the most relevant content server, it has the following limitations:

1. It has to be computed inline i.e. the DNS resolution has to wait while proximity to D-proxy is being computed unless the result has been cached.
2. If RTT computation is infrequent, then the RTT values may be stale and incorrect.
3. RTT values are transient, and hence cannot be assumed to be accurate at every point of time in the future.

In the recent times, various applications like fraud prevention, web analytics have evolved based on physical location (latitude and longitude) of a client machine. Various vendors are providing IP databases for these applications. We shall use one such database to decide proximity based on geographical distance (from client D-proxy) instead of RTT value. This will open up a new avenue for GSS to provide various value-added features.

We introduced location-based load balancing using an IP database. This database contains the range of IP address and their corresponding location. As the number of IP addresses can be huge, so is the size of this database. With the depletion of IPv4 addresses, IPv6 address will come into market. That will again lead to the increase in the size of database.

In the last section we mentioned about sticky database which is shared between all the GSS in the mesh through replication. Similarly GSS contains various other ever growing databases.

As part of this project our job is to propose an optimized fragmentation and allocation method. The size of all these databases is large due to that the cost of even a simple query will be high. Frequency of queries is very fast because for every DNS request that come has to be resolved on the basis of these databases. So an effective fragmentation technique is needed to fragment and allocates these databases to different GSSs.

In addition we discuss various fragmentation techniques to partition the database horizontally & vertically both. At the end we also show that better than partitioning either horizontally or vertically, we can do both one after another which is called grid fragmentation. We also show various combinations of HF and VF to perform different types of grid fragmentations and there benefits in different scenarios. Some of these are already proposed.

CHAPTER 4

Horizontal Fragmentation

In the previous chapters we have studied about fragmentation, its types, need for fragmentation and overview of Cisco GSS. From this chapter onwards we will explain the core work on our research. We explain here following Horizontal Fragmentation techniques:

1. Horizontal fragmentation using min-term predicates [10].
2. We have modified the graph-partitioning algorithm [11] and used the modified one to execute fragmentation.
3. Third approach calculates the attribute locality precedence [9] and fragments the databases accordingly.

4.1 Basic Approach to Horizontal Fragmentation

First of all we will discuss a simple algorithm to fragment a relation horizontally. This approach was first studied by S. Ceri, M. Negri, and G. Pelagatti in [10]. At the outset it creates all the possible min-terms using all the predicates. Then we eliminate some non-satisfactory min-terms & dependent min-terms. Finally we optimize the predicates that define our fragments. This approach is fairly simple enough to understand, but the amount of min-terms that can be created in the intermediate stages will be too high. This makes it difficult to use this in the real time scenarios. It does not use the query frequency for fragmentation. Further it can only be used in centralized database as it does not propose any allocation scheme.

4.1.1 Input & Output Definitions

Input to the algorithm is:

- Relation R ($A_1, \ldots, A_n$), where A_i is an attribute defined over domain D_i = Dom(A_i)
- Set of queries.

A predicate is defined as follows:

- p_j : A_i <operator> Value, with operator $\in$ { <, <=, >, >=, =, !=} & Value $\in$ Dom(A_i)
- p_j : defines potential binary fragmentation of R.

It gives Set of selection expressions M for fragmentation as output.

4.1.2 Algorithm

Now we will explain this algorithm in a stepwise fashion with help of an example.

1. **Obtain user queries and statstics**

   Using following set of queries as an example we will explain the complete algorithm.

   Given Queries:

   - q_1: SELECT DName FROM Department WHERE DCode = 'DCAS'
   - q_2: SELECT Location FROM Department WHERE Budget BETWEEN 50,000 AND 200,000

2. **Identify simple predicates in the queries**

   Predicates identified out of above given queries:

   - p_1: DCode = 'DCAS'
   - p_2: Budget >= 50,000
   - p_3: Budget <= 200,000

3. **Definition of all possible min-terms**

   Set $M_n(P)$ of all n-ary min-terms for a set of predicates P. A min-term m is a conjunction of simple predicates or negation of predicates. One min-term cannot contain both versions of a predicate i.e. predicate and its negation. Initially each min-term contains every predicate either in normal form or in negated form. All the min-terms define a complete and disjoint fragmentation of R.

   We denote $\sigma_m(R)$ as fragment defined by min-term *m*. Union of all the fragments give the original relation without any loss of information.

   - R = U $\sigma_m(R)$, where m ϵ $M_n(P)$

No two fragments have any tuple in common i.e. intersection of any two fragments is null.

- $\forall m_i, m_j, \in M_n(P), m_i \neq m_j : \sigma_{mi}(R) \cap \sigma_{mj}(R) = \emptyset$

We can drive following set of min-terms $M_3(P)$:

- $m_1 : p_1^+ \wedge p_2^+ \wedge p_3^+$
- $m_2 : p_1^+ \wedge p_2^+ \wedge p_3^-$
- $m_3 : p_1^+ \wedge p_2^- \wedge p_3^+$
- $m_4 : p_1^+ \wedge p_2^- \wedge p_3^-$
- $m_5 : p_1^- \wedge p_2^+ \wedge p_3^+$
- $m_6 : p_1^- \wedge p_2^+ \wedge p_3^-$
- $m_7 : p_1^- \wedge p_2^- \wedge p_3^+$
- $m_8 : p_1^- \wedge p_2^- \wedge p_3^-$

Here p^+ denotes he normal form of the predicate without any change & p^- denotes the negation of the predicate.

After putting the respective predicates, we get the following min-terms $M_n(P)$:

- m_1 : (DCode = 'DCAS`) Λ (Budget >= 50,000) Λ (Budget <= 200,000)
- m_2 : (DCode = 'DCAS`) Λ (Budget >= 50,000) Λ ¬(Budget <= 200,000)
- m_3 : (DCode = 'DCAS`) Λ ¬ (Budget >= 50,000) Λ (Budget <= 200,000)
- m_4 : (DCode = 'DCAS`) Λ ¬ (Budget >= 50,000) Λ ¬ (Budget <= 200,000)
- m_5 : ¬ (DCode = 'DCAS`) Λ (Budget >= 50,000) Λ (Budget <= 200,000)
- m_6 : ¬ (DCode = 'DCAS`) Λ (Budget >= 50,000) Λ ¬ (Budget <= 200,000)
- m_7 : ¬ (DCode = 'DCAS`) Λ ¬ (Budget >= 50,000) Λ (Budget <= 200,000)
- m_8 : ¬ (DCode = 'DCAS`) Λ ¬ (Budget >= 50,000) Λ ¬ (Budget <= 200,000)

4. Elimination of non-satisfactory min-terms

As we can see 4th min-term defines as fragment where budget is less than 50000 and budget is more than 200,000 as well, which is not possible. Similarly, we can eliminate 8th predicate.

- m_1 : (DCode = 'DCAS`) Λ (Budget >= 50,000) Λ (Budget <= 200,000)
- m_2 : (DCode = 'DCAS`) Λ (Budget >= 50,000) Λ ¬(Budget <= 200,000)
- m_3 : (DCode = 'DCAS`) Λ ¬ (Budget >= 50,000) Λ (Budget <= 200,000)
- ~~m_4 : (DCode = 'DCAS`) Λ ¬ (Budget >= 50,000) Λ ¬ (Budget <= 200,000)~~

- m_5 : ¬ (DCode = 'DCAS`) Λ (Budget >= 50,000) Λ (Budget <= 200,000)
- m_6 : ¬ (DCode = 'DCAS`) Λ (Budget >= 50,000) Λ ¬ (Budget <= 200,000)
- m_7 : ¬ (DCode = 'DCAS`) Λ ¬ (Budget >= 50,000) Λ (Budget <= 200,000)
- ~~m_8 : ¬ (DCode = 'DCAS`) Λ ¬ (Budget >= 50,000) Λ ¬ (Budget <= 200,000)~~

5. **Elimination of dependent predicates**

   Eliminate predicates in a min-term that are dependent (implications & functional dependencies)

- m_1 : (DCode = 'DCAS`) Λ (Budget >= 50,000) Λ (Budget <= 200,000)
- ~~m_2 : (DCode = 'DCAS`) Λ (Budget >= 50,000) Λ ¬(Budget <= 200,000)~~
- m_2 : (DCode = 'DCAS`) Λ (Budget > 200,000)
- ~~m_3 : (DCode = 'DCAS`) Λ ¬ (Budget >= 50,000) Λ (Budget <= 200,000)~~
- m_3 : (DCode = 'DCAS`) Λ (Budget < 50,000)
- m_5 : ¬ (DCode = 'DCAS`) Λ (Budget >= 50,000) Λ (Budget <= 200,000)
- ~~m_6 : ¬ (DCode = 'DCAS`) Λ (Budget >= 50,000) Λ ¬ (Budget <= 200,000)~~
- m_6 : ¬ (DCode = 'DCAS`) Λ (Budget > 200,000)
- ~~m_7 : ¬ (DCode = 'DCAS`) Λ ¬ (Budget >= 50,000) Λ (Budget <= 200,000)~~
- m_7 : ¬ (DCode = 'DCAS`) Λ (Budget < 50,000)

6. **Estimate selectivity of each min-term**
7. **Find minimal & complete sets of min-terms for defining fragments**
   - ➢ **Minimal**: at least one query accesses fragment
   - ➢ **Complete**: each and every tuple is the part of some fragment.

Given predicates:

- p_1: DCode = ‘DCAS’
- p_2: Budget >= 50,000
- p_3: Budget <= 200,000

As shown in the figure 10 we have divided our database into five fragments. Each fragment is shown belonging to different predicates. This is just the initial fragmentation defined on the basis of given predicates. It has nothing to do with the actual fragmentation.

Given Min-term:

- m_1 : (DCode = 'DCAS`) Λ (Budget >= 50,000) Λ (Budget <= 200,000)

Fragmentation on m_1

- $R_1 = \{ F_2 \}$

Similarly for other min-terms we have,

- m_2 : (DCode = 'DCAS`) Λ (Budget > 200,000), $R_2 = \{ F_3 \}$
- m_3 : (DCode = 'DCAS`) Λ (Budget < 50,000) , $R_3 = \{ F_1 \}$
- m_5 : ¬ (DCode = 'DCAS`) Λ (Budget >= 50,000) Λ (Budget <= 200,000), $R_5 = \{ \}$
- m_6 : ¬ (DCode = 'DCAS`) Λ (Budget > 200,000), $R_6 = \{ F_4, F_5 \}$
- m_7 : ¬ (DCode = 'DCAS`) Λ (Budget < 50,000), $R_7 = \{ \}$

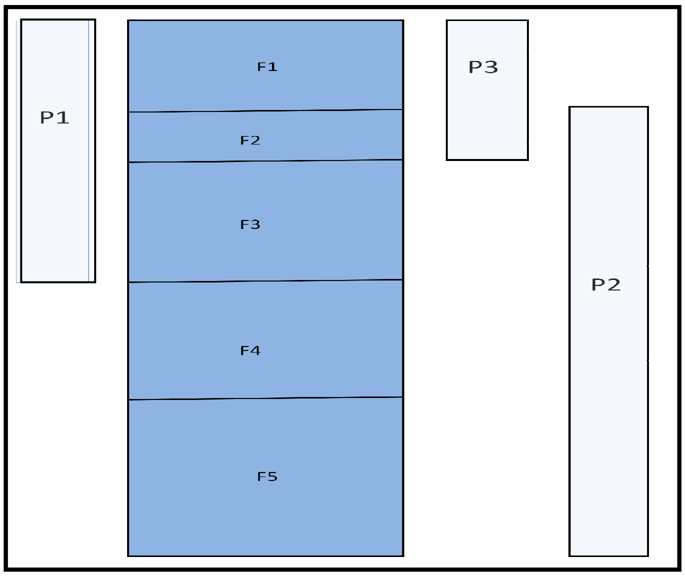


Figure 10 : Predicates & Fragments

Obtained Fragmentation

- $R_1 = \{ F_2 \}$
- $R_2 = \{ F_3 \}$
- $R_3 = \{ F_1 \}$
- $R_6 = \{ F_4, F_5 \}$

4.1.3 Summing up

Over all the process is very much simple and straight forward. Number of min-terms generated in the intermediate steps is exponential to number of predicates. Applying this process to databases would be very mind-numbing. Process includes all the predicates where some may not be involved in the final fragmentation. As in the previous example if we had taken p_2 and p_3, it would have led to the same fragmentation as using all three predicates. In the upcoming sections we will not consider this approach as candidate partitioning technique as implementation of this is not feasible.

4.2 Graphical Approach For HF

Previously what we studied is a naïve approach to horizontal fragmentation with limited application. In this technique we have modified graph partitioning procedure in [1]. This algorithm was first proposed by S. B. Navathe and M. Ra in [1] for vertical partitioning and later on it was proposed for horizontal partitioning by M. Ra in [11]. We have changed the method but the approach is quite similar i.e. creating the partition by forming cycles in the graph.

4.2.1 Definitions

- **Cycle completing edge** denotes edges that would complete a cycle.
- **Cycle connecting edge** denotes edges that connect two cycles or partitions.
- Every node in a cycle is called **Cycle node.**
- **End Nodes** are the end points of linearly connected tree. Only two end nodes exist in a linearly connected tree.
- When a cycle encloses any smaller cycle, then the smaller cycle is called **Enclosed Cycle** and bigger cycle is called **Enclosing cycle.** For Example, as shown in the figure 11 cycle ABCDA is enclosing cycles and the cycle BCDB is enclosed cycle.

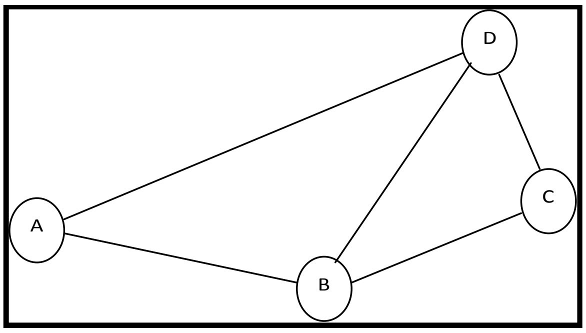


Figure 11 : Enclosed and Enclosing Cycle

4.2.2 Example

We explain our horizontal partitioning methodology by using a simple example below. We use following set of predicates & transactions that uses these predicates.

- p1: ENo < 10
- p2: ENo < 20
- p3: ENo > 20
- p4: 30 < ENo < 50
- p5: ENo < 15
- p6: ENo > 40
- p7: Sal > 80k

- p8: Sal < 80k

Transactions and the predicates used by them:

- T1: p1 & p7
- T2: p2 & p7
- T3: p3 & p7
- T4: p4 & p8
- T5: p5 & p8
- T6: p6 & p8
- T7: p5 & p8
- T8: p6 & p8

Input to our algorithm is predicate usage matrix (PUM) which represents the use of predicates in important transactions. The predicate usage matrix for the example is shown in table 1. Each row refers to one transaction & column refers to the predicates. The '1' entry indicates the transaction uses the corresponding predicate. Depending upon whether transaction performs a write or read operation we put R or W in the type column. Last column is where frequency of the transaction is stored.

Table 1 : Predicate Usage Matrix

Transactions	p_1	p_2	p_3	p_4	p_5	p_6	p_7	p_8	Type	Access Freq
T_1	1	0	0	0	0	0	1	0	R	25
T_2	0	1	0	0	0	0	1	0	R	50
T_3	0	0	1	0	0	0	1	0	R	25
T_4	0	0	0	1	0	0	0	1	R	35
T_5	0	0	0	0	1	0	0	1	W	25
T_6	0	0	0	0	0	1	0	1	W	25
T_7	0	0	0	0	1	0	0	1	W	25
T_8	0	0	0	0	0	1	0	1	W	15

Now from this Predicate usage matrix (PUM) we generate predicate affinity matrix. The numerical value of the (i, j) element in this matrix gives the combined frequency of all transactions accessing both predicates i and j. The value "=>" of the (i, j) element indicates that predicate i implies predicate j, the value "<=" of the (i, j) element indicates that predicate j implies predicate i & the value "*" means that two predicates i and j are close [5]. Two predicate i and j are 'close' when the following conditions are satisfied:

(i) i and j must be defined on the same attribute,

(ii) i and j must be jointly used with some common predicate c, and

(iii) c must be defined on an attribute other than the attribute used in i and j.

In the given example, p_1 and p_2 are “close” because of their usage with common predicate p_7 in the transaction T_1 and T_2. Predicate affinity matrix for the given PUM is shown in table 2.

Table 2 : Predicate Affinity Matrix

Predicates	p_1	p_2	p_3	p_4	p_5	p_6	p_7	p_8
p_1		=>, “*”	“*”	0	0	0	25	0
p_2	<=, “*”		“*”	0	0	0	50	0
p_3	“*”	“*”		<=	0	0	25	0
p_4	0	0	=>		“*”	“*”	0	35
p_5	0	0	0	“*”		“*”	0	50
p_6	0	0	0	“*”	“*”		0	40
p_7	25	50	25	0	0	0		0
p_8	0	0	0	35	50	40	0	

4.2.3 Creating affinity graph

First of all we create affinity graph from predicate affinity matrix (PAM). In this graph, nodes represent a predicate and edge value represents the affinity between the two predicates. Forming linearly connected spanning tree, linearly connected tree is constructed by including one edge at a time such that only edges at the first and the last node of the tree would be considered for inclusion. We include edges with high affinity value & growing the tree as much as possible. In the end we have to form cycles. Different cycles are separated by cycle connecting edges. Let’s go through some rules that would be followed while selecting the edges.

4.2.4 Rules for Selecting the Edges

1. A numerical value (except zero) has higher priority than the values “=>”, “<=” and “*” when selecting a next edge during the progression of the algorithm. This is because more importance is placed on affinity values which are obtained from transaction usage rather than on logical connectivity among the predicates.
2. “<=” and “=>” are considered to have higher affinity than “*” since the latter only represents logical connectivity between the two predicates through their usage with common predicate.
3. If there are two “=>” relationships in a column corresponding to predicate p_k, one implied by predicate p_i and other implied by predicate p_j, then

(i) the entry (i, k) has higher priority than the entry (j, k), if the entry (i, j) is equal to “<=” or

(ii) the entry (j, k) has higher priority than entry (i, k) if the entry (i, j) is equal to “=>”.

4.2.5 Algorithm

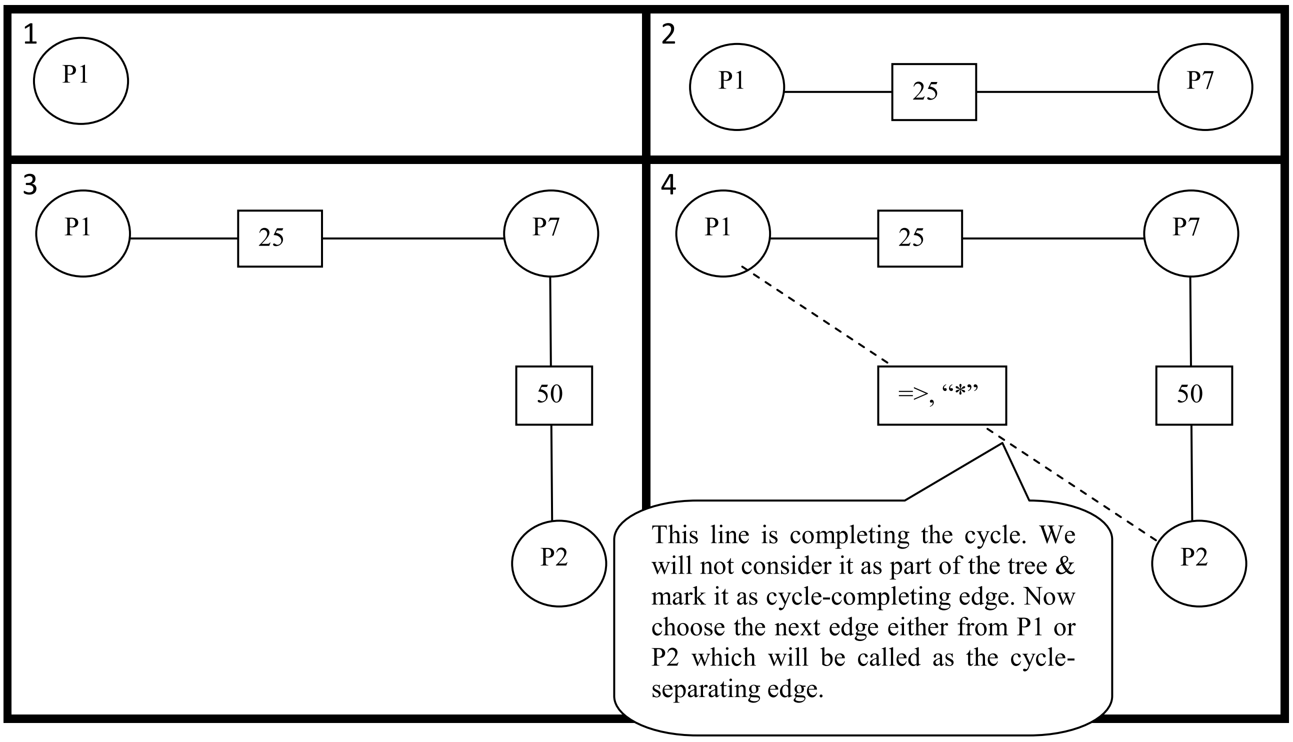
First of all create affinity graph of the predicates. Note: that PAM is itself an adequate data structure to represent this graph. Algorithm for partitioning the graph by forming linearly connected tree and cycles later on is as follows:

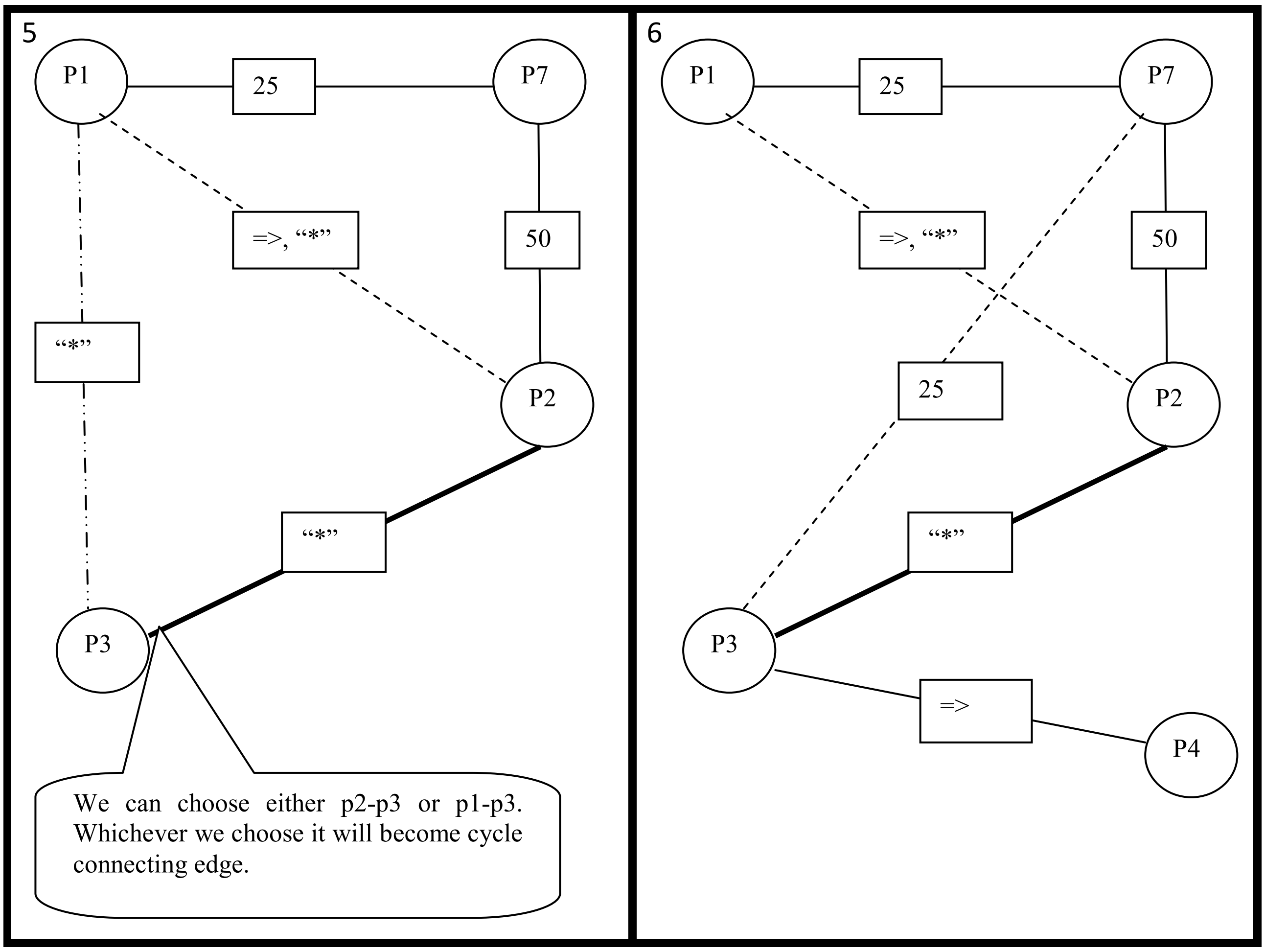
1. Start from any node
2. Select an edge which satisfies the following conditions:
   a) Go to step 5 if all nodes are used for tree construction
   b) It should be linearly connected to the tree already constructed.
   c) It should have the largest value among the possible choices of edges at each end of the tree. Note that if there are several largest values, anyone can be selected.
3. When the next selected edge form a cycle and it does not connect to the cycle node of any cycle except the enclosed cycle.
   a) Mark this edge as cycle-completing edge but this edge is not considered as part of the tree and end nodes remain the same.
   b) If this cycle encloses any cycle whose cycle-completing edge have low or equal affinity than this edge then
      (i) remove the cycle-completing tag from the edge of enclosed cycle and
      (ii) remove the cycle separating tag from the edge that have it inside the enclosing cycle so formed.
   c) Go to step 2.
4. When the next edge selected does not form a cycle
   a) Change the end node to the next node to which this edge is connected.
   b) If the first node of this edge is cycle node then mark this edge as cycle-connecting edge.
   c) Go to step 2.
5. When all the nodes get traversed & become a part of the tree.
   a) Choose the non-selected edge from the any non-cycle node with the highest affinity & go to step 3.
   b) Then mark the selected-edge connecting this cycle with the rest of the graph as cycle-separating edge.
6. Separating out cycles.
   a) Choose the cycle completing-edge with highest affinity and complete the cycle. If the cycle so formed contains a cycle-separating edge then ignore this cycle. Keep doing this until we get all the nodes as part of some cycle.

4.2.6 Step by Step Solution

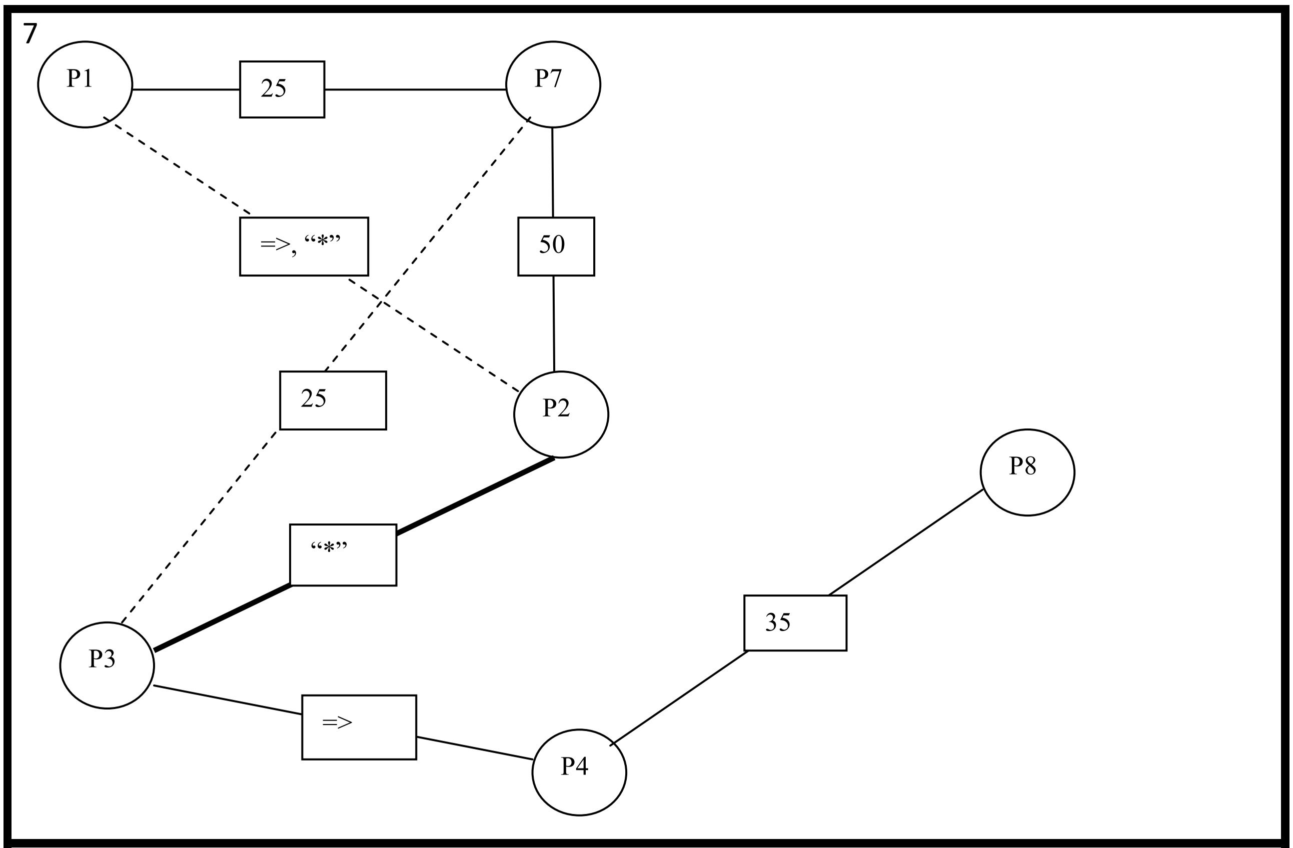
Now we will show the complete solution of the example we had given in the section 4.2.1 graphically by applying the above given algorithm. Step by step solution of the complete approach is shown in the figure 12.

Figure 12 : Step by Step Solution of Graphical Approach for HF

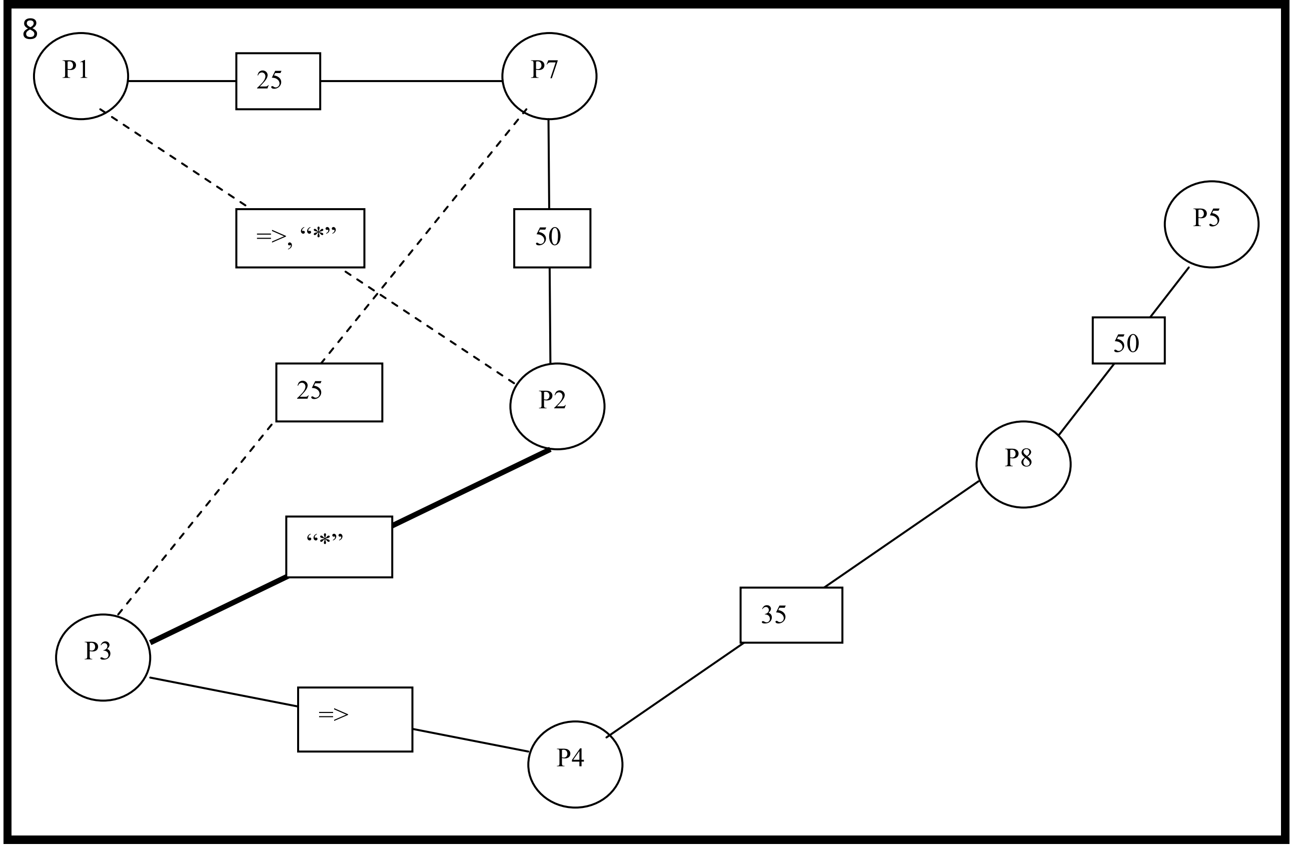

1
P1
2
P1
25
P7
3
P1
25
P7
50
P2
4
P1
25
P7
=>, "*"
50
P2
This line is completing the cycle. We will not consider it as part of the tree & mark it as cycle-completing edge. Now choose the next edge either from P1 or P2 which will be called as the cycle-separating edge.


5
P1
25
P7
=>, "*"
50
"*"
P2
"*"
P3
We can choose either p2-p3 or p1-p3. Whichever we choose it will become cycle connecting edge.
6
P1
25
P7
=>, "*"
50
25
P2
"*"
P3
=>
P4

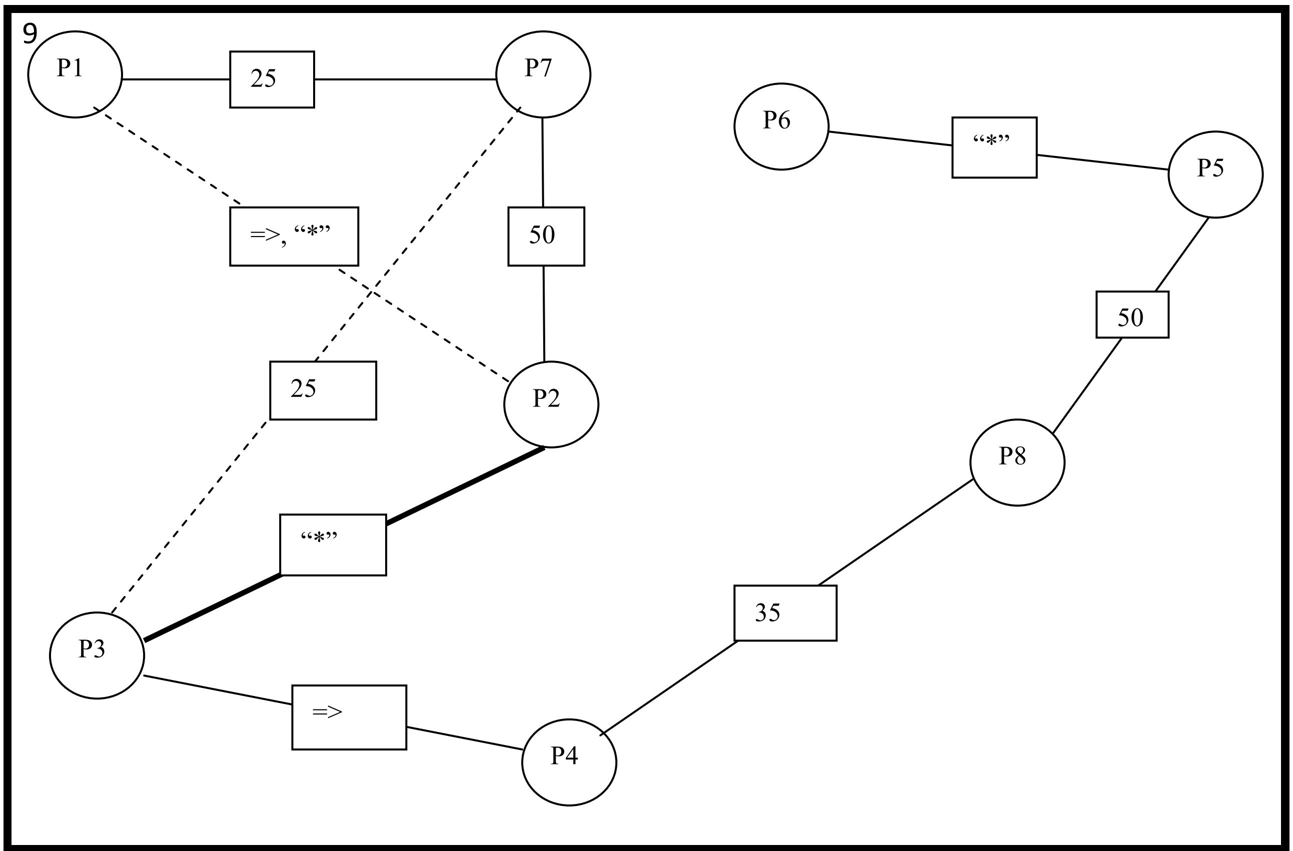
7



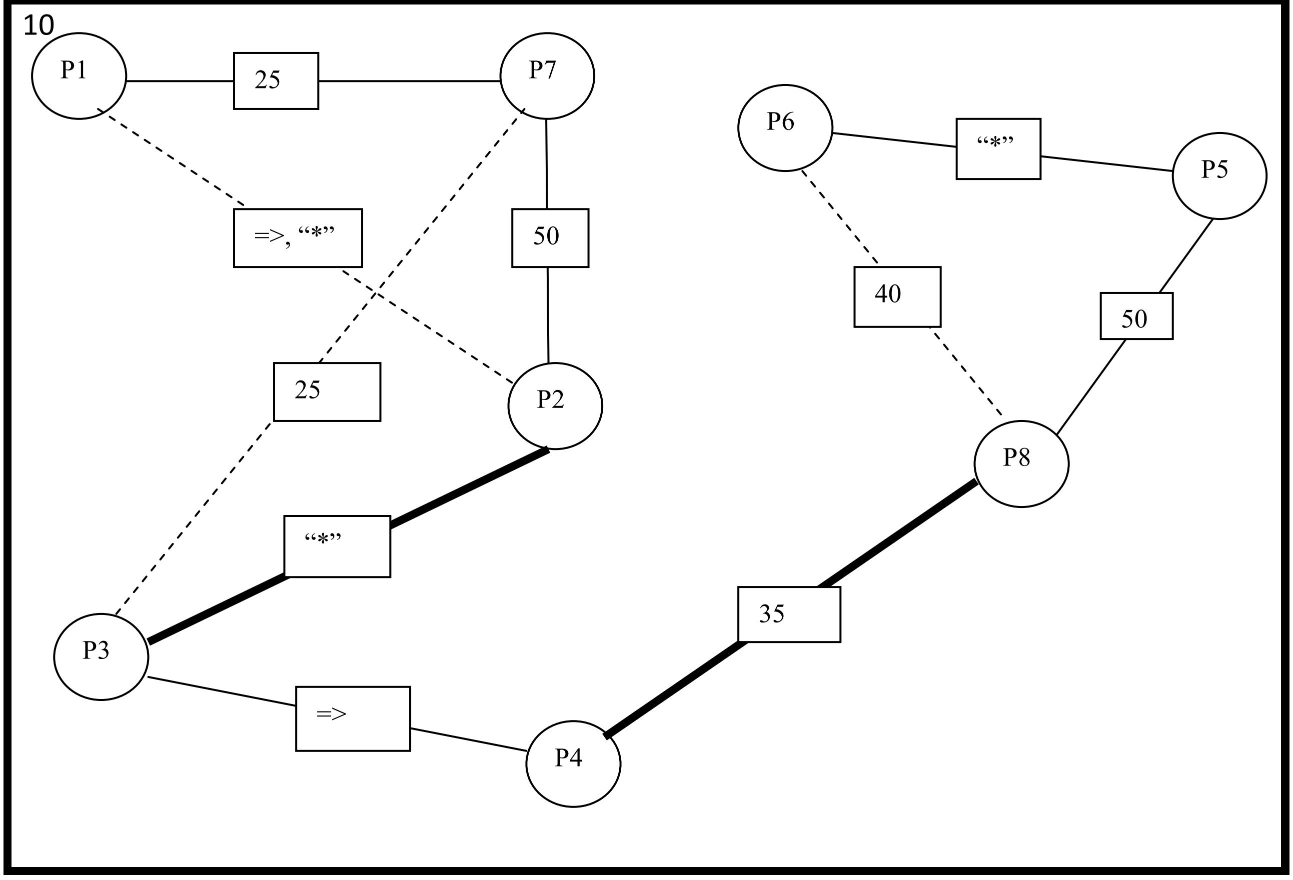
8

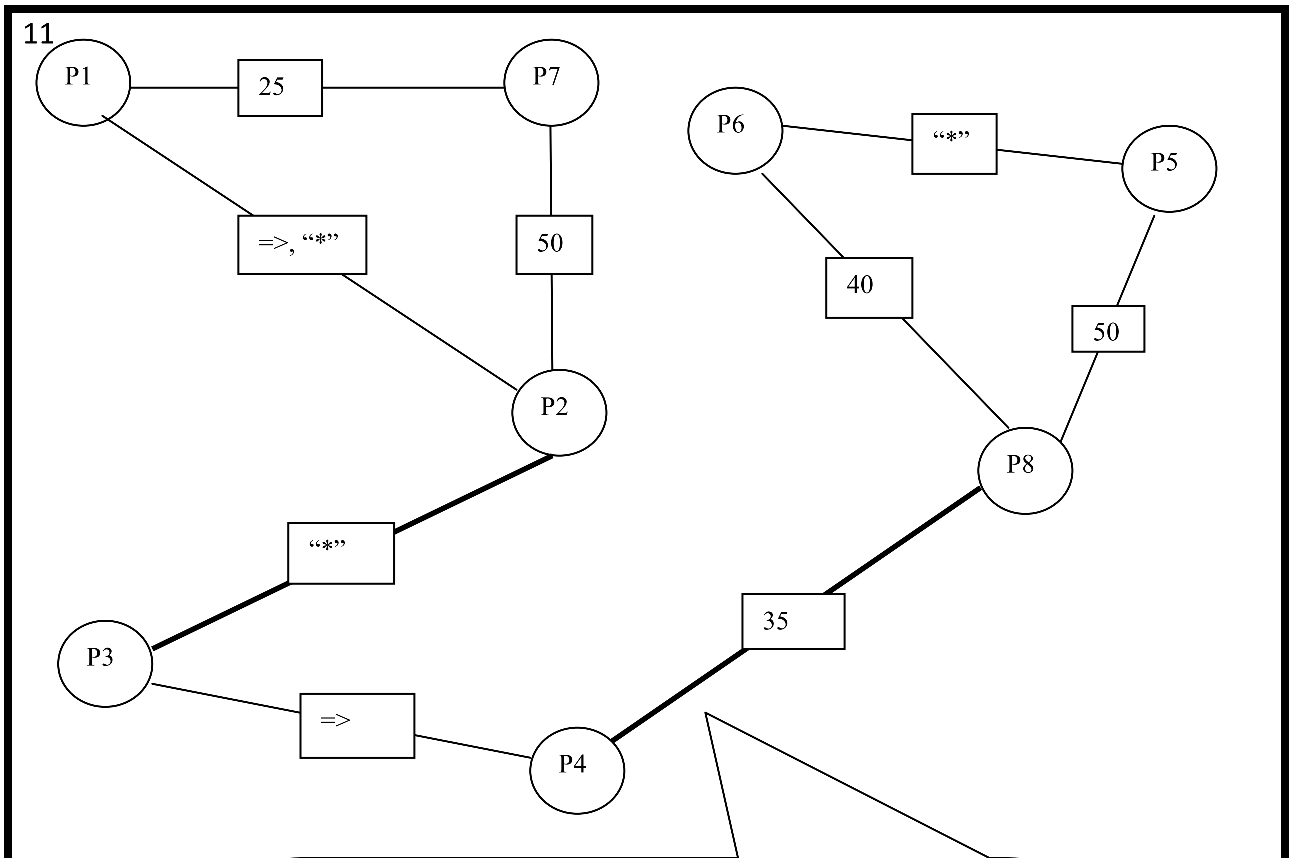


9


P1
25
P7
P6
"*"
P5
=>, "*"
50
50
25
P2
P8
"*"
35
P3
=>
P4

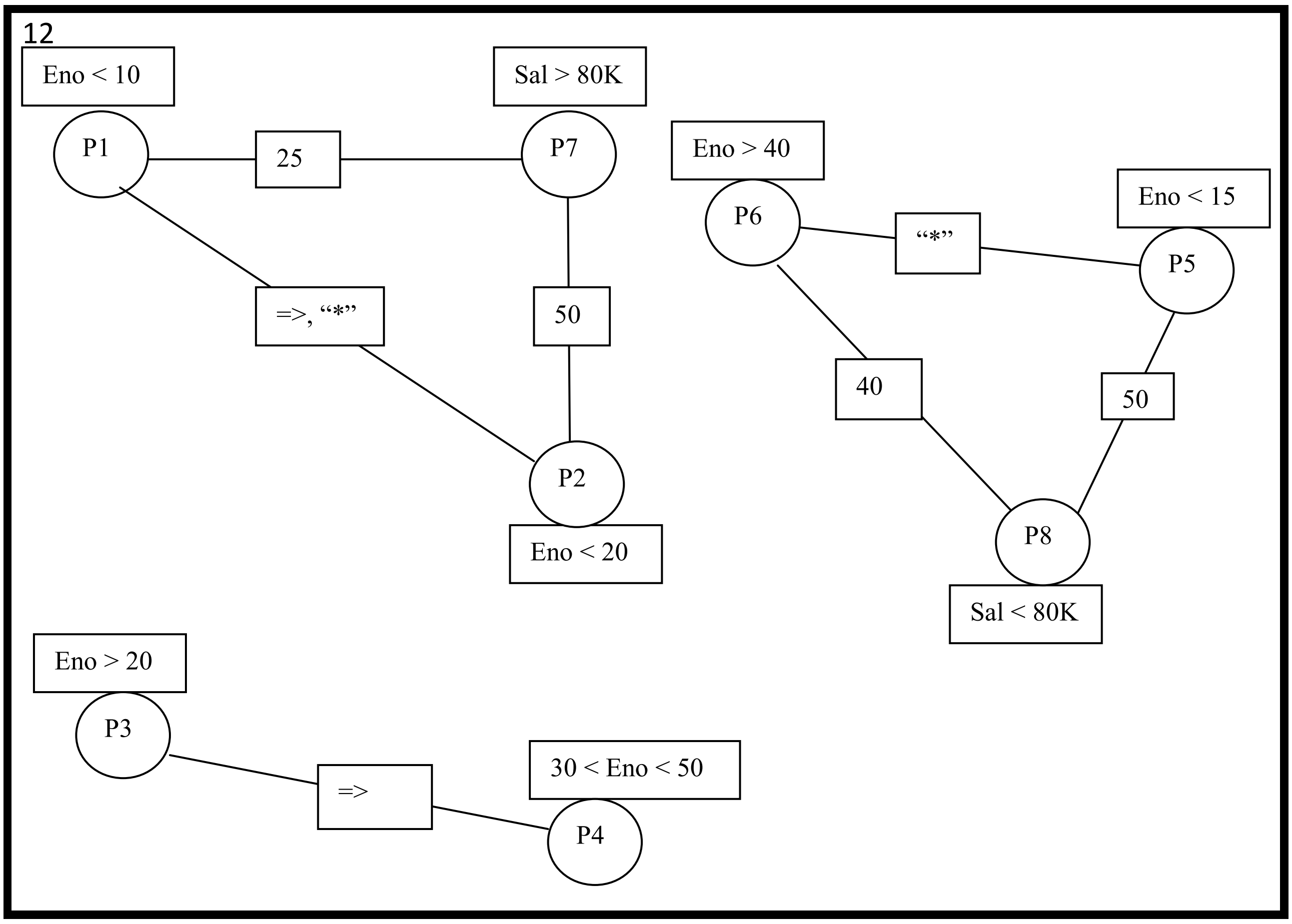
10


P1
25
P7
P6
"*"
P5
=>, "*"
50
40
50
25
P2
P8
"*"
35
P3
=>
P4



Now we choose cycle completing edges.

1) P6-P8, as it has highest affinity.
2) P3-P7, will not be chosen because the cycle formed by this {P7-P2-P3} involves the cycle-connecting edge P2-P3 whose affinity is lowest of all the edges in the cycle {P1-P7-P2}. Hence we choose P1-P2.



4.2.7 Optimization of Results

As result of algorithm applied on the given example we obtained following three set of predicates:

- (p_1, p_2, p_7)
- (p_3, p_4)
- (p_5, p_6, p_8)

Putting the values of respective predicates we get:

- (Eno < 10, Eno < 20, Sal > 80K)
- (Eno > 20, 30 < Eno < 50)
- (Eno > 40, Eno < 15, Sal < 80K)

We can refine these predicates sets further by checking for dependencies and inclsion. In the first subset, Eno < 10 ➔ Eno < 20 hence it is refined into (Eno < 20, Sal > 80K). Similarly, in the second 30 < Eno < 50 ➔ Eno > 20 hence it is refined into (Eno > 20). Third one cannot be refined further. Now we have:

- (Eno < 20, Sal > 80K)
- (Eno > 20)

- (Eno > 40, Eno < 15, Sal < 80K)

If the predicates belonging to the same set refer to the same attributes then they are OR-ed, otherwise they are AND-ed.

- (Eno < 20 AND Sal > 80K)
- (Eno > 20)
- ((Eno > 40 OR Eno < 15) AND Sal < 80K)

Applying the law of distributivity:

- (Eno < 20 AND Sal > 80K)
- (Eno > 20)
- (Eno > 40 AND Sal < 80K)
- (Eno < 15 AND Sal < 80K)

Further we can merge the overlapping sets. As 2nd set will be superset of the 3rd set. So we can write:

- (Eno < 20 AND Sal > 80K)
- (Eno > 20)
- (Eno < 15 AND Sal < 80K)

These will form the final set of fragments. But these will not include all the tuples of the relation. There may be some tuples which will not fall into any of the fragment. Hence number of fragments is at most equal to the number of predicate terms plus one.

4.2.8 Advantages

Following are some of the advantages of the approach we discussed:

1. Fragments are bases on actual predicates. By applying implication the number of fragments is reduced.
2. Amount of data involved is much less than the previous approach.

4.2.9 Disadvantages

Besides having some advantages this approach is still not the one that suites distributed databases. Here we discuss some of the disadvantages of this approach:

1. Need of optimization. Fragments obtained are needed to be refined. There is no particular method to refine the fragments.
2. Problem of allocation remain unsolved.
3. Only simple predicates are used. And how to select the predicates for fragmentation is another major task.

4.3 Advanced Horizontal Fragmentation Technique

To solve the problem of allocation along with fragmentation in distributed database, we have provided this technique of fragmentation. It forms the part of our final proposal of grid fragmentation. This technique was proposed by S. I. Khan & Dr. A.S.M. Latiful Hoque in [9]. It also solves the problem of fragmentation decision at the initial stage of a distributed database. It fragments the relation horizontally according to locality of precedence of its attributes.

4.3.1 Attribute Locality Precedence

Attribute locality precedence (ALP) can be defined as the value of importance of an attribute with respect to sites of distributed database. ALP table will be constructed by database designer for each relation of a DDBMS at the time of designing the database with the help of modified CRUD (Create, Read, Update, and Delete) matrix and cost functions.

"A data-to-location CRUD matrix is a table of which rows indicate attributes of the entities of a relation and columns indicate different locations of the applications."

4.3.2 MCURD Matrix

A relation in a database contains different types of attributes those describe properties of the relation. But the important thing is that the attributes of a relation do not have same importance with respect to data distribution in different sites. According to above importance we can calculate locality precedence of each attribute for each relation and construct ALP table for the relations. CRUD matrix is used by the system analysts and designers in the requirement analysis phase of system development life cycle for making decision of data mapping to different locations [16]. We have modified the existing CRUD matrix according to our requirement of Horizontal Fragmentation and name it Modified Create, Read, Update, and Delete (MCRUD) matrix.

"MCRUD matrix is a table constructed by placing predicates of attributes of a relation as the rows and sites of a DDBMS as the columns."

4.3.3 Algorithm

We have used MCRUD matrix to generate ALP table for each relation. We treated cost as the effort of access and modification of a particular attribute of a relation by an application from a particular site. For calculating precedence of an attribute of a relation we take the MCRUD matrix of the relation as an input and use the cost functions given in equation 4.1 for calculating cost of predicate j of attribute i accessed by an application at site k. This gives us the cost of a predicate at different sites. Predicate locality precedence

(PLP) is equal to the maximum cost minus sum of all other costs of that predicate. Attribute locality precedence (ALP) is equal to the sum of PLP of the predicates of that attribute.

$$\mathbf{C_{i,j,k} = f_c * C + f_r * R + f_u * U + f_d * D} \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots (4.1)$$

$$\mathbf{PLP_{i,j} = C_{i,j,m} - \sum_{k=0\ \&\ k \neq m}^{n} C_{i,j,k}} \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots (4.2)$$

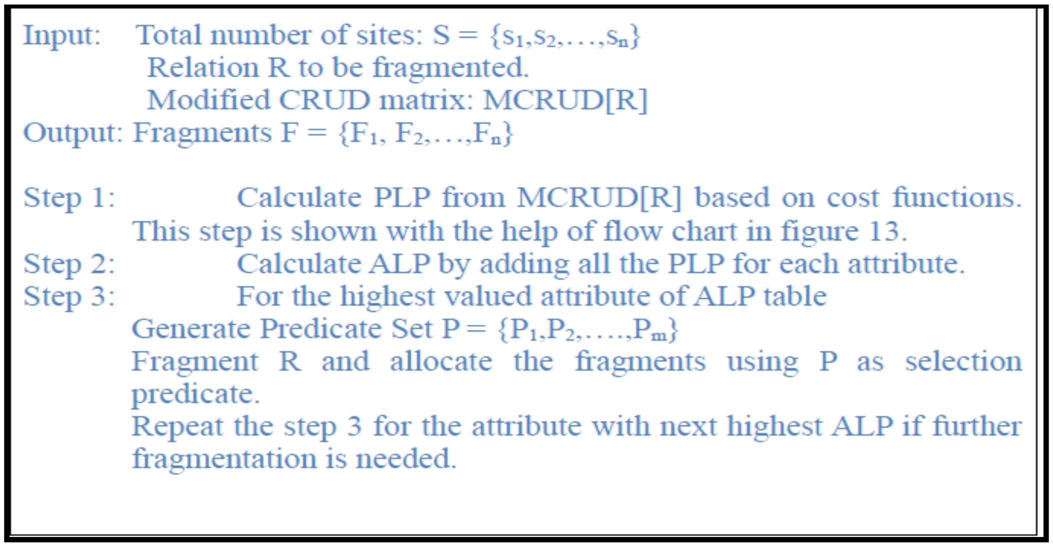
$$\mathbf{ALP_i = \sum PLP_{i,j}} \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots (4.3)$$

Where,

f_c - frequency of create operation
f_r - frequency of read operation
f_u - frequency of update operation
f_d - frequency of delete operation
C - weight of create operation
R - weight of read operation
U - weight of update operation
D - weight of delete operation
$C_{i,j,k}$ = cost of predicate j of attribute i accessed by an application at site k
$C_{i,j,m}$ = maximum cost among the sites for predicate j of attribute i
$PLP_{i,j}$ = actual cost for predicate j of the attribute i
ALP_i = total cost of attribute i(locality precedence)

For simplicity we have assumed that f_c, f_r, f_r and f_d=1 and C=2, R=1, U=3 and D=2. The justification of the assumption is that at the design time of a distributed database, the designer will not know the actual frequencies of read, delete, create and update of a particular attribute from different applications of a site. Later when database is actually used we update these frequencies in the MCURD matrix and recalculate the ALP. We can do this recalculation periodically.

Generally update incurs more cost than create and delete, and reading from database always incurs least cost. After construction of ALP table for a relation, predicate set P will be generated for the attribute with highest precedence value in the ALP table. Finally each relation will be fragmented horizontally using the predicates of P as selection predicate. For further fragmentation of the fragments select the attribute with next highest ALP and use its predicates for fragmenting. The procedures can be clearly understood from the following pseudo code.



```
Input:   Total number of sites: $S = \{s_1, s_2, \ldots, s_n\}$
         Relation R to be fragmented.
         Modified CRUD matrix: MCRUD[R]
Output:  Fragments $F = \{F_1, F_2, \ldots, F_n\}$

Step 1:  Calculate PLP from MCRUD[R] based on cost functions.
         This step is shown with the help of flow chart in figure 13.
Step 2:  Calculate ALP by adding all the PLP for each attribute.
Step 3:  For the highest valued attribute of ALP table
         Generate Predicate Set $P = \{P_1, P_2, \ldots, P_m\}$
         Fragment R and allocate the fragments using P as selection
         predicate.
         Repeat the step 3 for the attribute with next highest ALP if further
         fragmentation is needed.
```

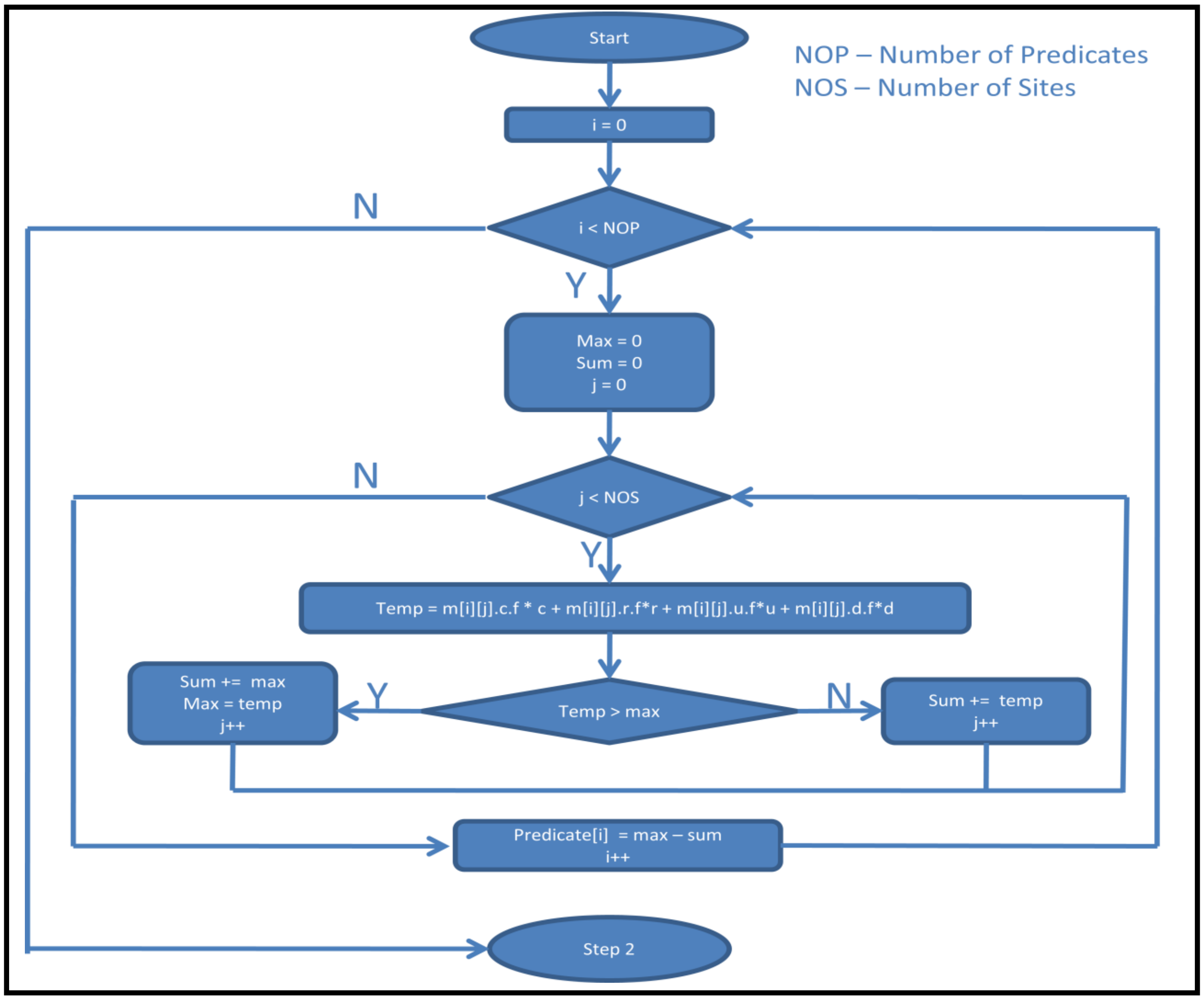


Figure 1 : Flow Chart for Finding PLP

Now we elaborate all these steps with the help of an example.

4.3.4 Example

To explain this method in more detail we have taken a distributed banking database system. One of the relations of the bank database is Accounts as shown in table 3. Initially number of sites of the distributed system is three as shown in figure 14.

Table 3 : Account Relation

Acc_num	Type	Id	Branch	City	Balance
01	Sav	101	Rohini	Delhi	230000
02	Cur	102	K.B.Hali	Bangalore	120000
03	Sav	103	Kormangla	Bangalore	6500
04	Sav	104	R.K.Puram	Mumbai	32100
05	Cur	105	NSP	Delhi	400000

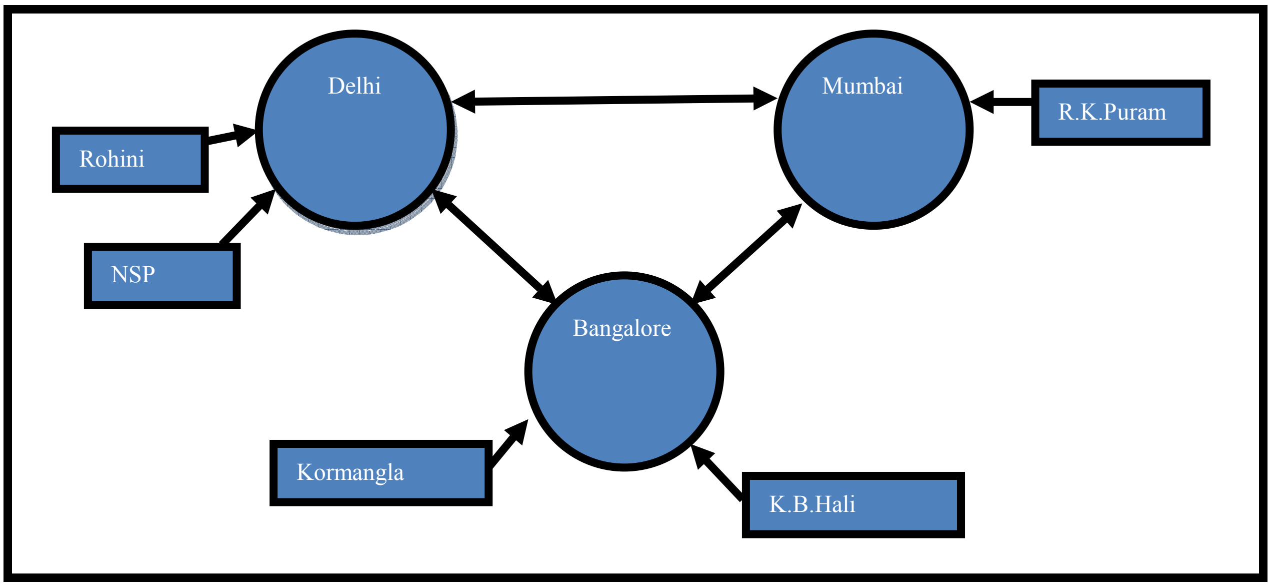


Figure 14 : Site Map for Banking System.

4.3.5 Construction of MCRUD Matrix

Part of MCRUD matrix for accounts relation is shown in table 4.

Table 4 : MCRUD Matrix

Sites / Predicates	Site 1(Delhi)				Site 2 (Bangalore)				Site 3(Mumbai)			
	C	R	U	D	C	R	U	D	C	R	U	D
Acc.acc_num < 100	1	1	1							1		
Acc.acc_num >=100	1											
Acc.type = 'Sav'	1	3	2	2		1						
Acc.type = 'Cur'		2	1						1	3	2	1
………. ………. ………. ……….												
Acc.balance < 10000		2			1	1	1	1		1		
Acc.balance >= 10000	1	1										
Acc.city = 'Delhi'	2	3	3	2		1				1		
Acc.city = 'Bangalore'		1			2	3	2	2		1		
Acc.city = 'Mumbai'									2	3	2	2

4.3.6 Calculation of ALP

We have calculated locality precedence of each attribute from the MCRUD matrix of account relation according to the cost functions given in the equations 4.1, 4.2 and 4.3. Calculating the locality precedence of the attribute City is as follows:

$$\mathbf{PLP_{acc.city = 'Delhi'} = (2*2 + 3*1 + 3*3 + 2*2) - \{ (1*1) + (1*1) \} = 18}$$

$$\mathbf{PLP_{acc.city = 'Banngalore'} = (2*2 + 3*1 + 2*3 + 2*2) - \{ (1*1) + (1*1) \} = 15}$$

$$\mathbf{PLP_{acc.city = 'Mumbai'} = (2*2 + 3*1 + 2*3 + 2*2) = 17}$$

So ALP of City is

$$\mathbf{ALP_{city} = 18 + 15 + 17 = 50}$$

4.3.7 Construction of ALP Table

ALP values of all the attributes of the Accounts relation computed from its MCRUD matrix is shown in the table 5. The attribute with highest precedence value will be treated as most important attribute for fragmentation.

Table 5 : ALP

Attribute	**ALP**
Acc_num	6
Type	22
ID	6
Branch	7
Balance	8
City	50

4.3.8 Generation of Predicate Set

Predicate set generated for City, the attribute with highest locality precedence of Account relation.

P= { p1: city = 'Delhi', p2: city = 'Bangalore', p3: city = 'Mumbai' }

4.3.9 Fragmentation of Relation

According to the predicate set P, Account relation is fragmented and allocated to 3 sites. The relation allocated to site in Delhi is shown in table 6. Similarly, for site in Bangalore and Mumbai is shown in table 7 and 8 respectively.

Table 6 : Relation for site in Delhi

Acc_num	Type	Id	Branch	City	Balance
01	Sav	101	Rohini	Delhi	230000
05	Cur	105	NSP	Delhi	400000

Table 7 : Relation for site in Bangalore

Acc_num	Type	Id	Branch	City	Balance
02	Cur	102	K.B.Hali	Bangalore	120000
03	Sav	103	Kormangla	Bangalore	6500

Table 8 : Relation for site in Mumbai

Acc_num	Type	Id	Branch	City	Balance
04	Sav	104	R.K.Puram	Mumbai	32100

4.3.10 Addition of a New Site to DDBMS

We have added another site in Delhi to the current DDBMS (see figure 15). In this case the fragment in Delhi is re-fragmented horizontally based on the attribute with next highest locality precedence which is Type.

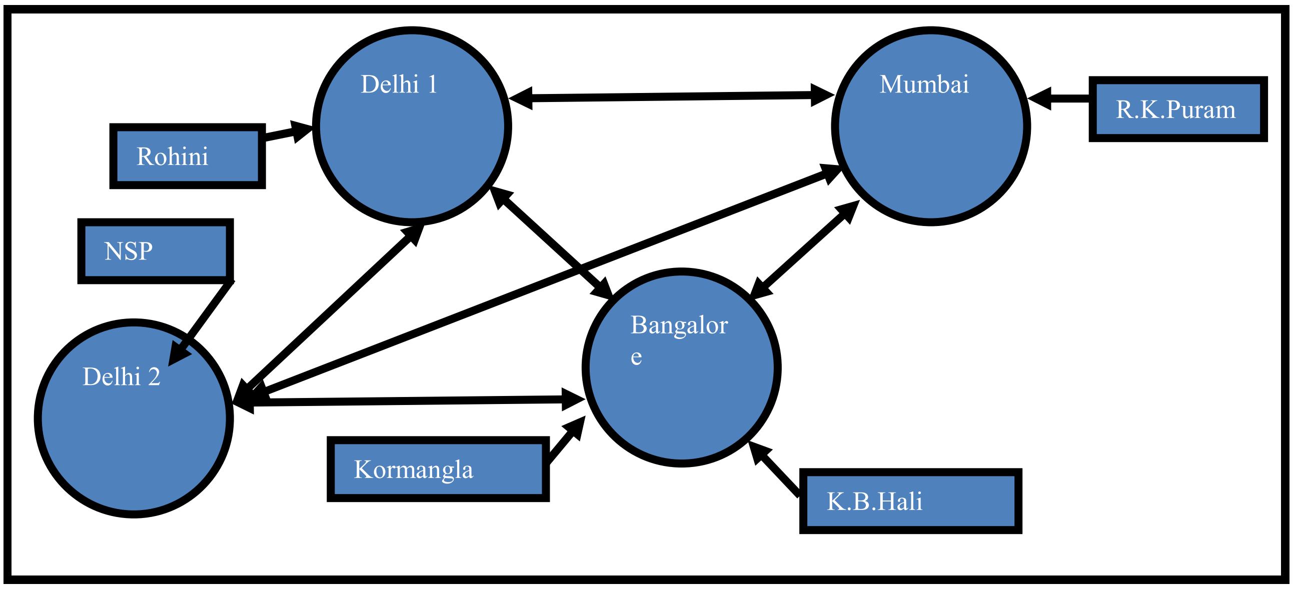


Figure 15 : Addition of New Site to Map

Here attribute Type have the ALP equal to 22. Predicate set generated for Type

$P = \{ p_1\text{: Type = Sav}, p_2\text{: Type = Cur} \}$

These two predicates produce min-term with the former predicate of site in Delhi as follows:

p_{11} : branch=Delhi Λ Type= Sav ,

p_{12} : branch=Delhi Λ Type= Cur.

Hence the final predicate set is

$P=\{ p_{11}, p_{12}, p_2, p_3 \}$

Account relation was then fragmented according to P and allocated to 4 sites as shown in table 9, 10, 11 and 12.

Table 9 : Relation for site in Delhi 1

Acc_num	Type	Id	Branch	City	Balance
01	Sav	101	Rohini	Delhi	230000

Table 10 : Relation for site in Delhi 2

Acc_num	Type	Id	Branch	City	Balance
05	Cur	105	NSP	Delhi	400000

Table 11 : Relation for site in Bangalore

Acc_num	Type	Id	Branch	City	Balance
02	Cur	102	K.B.Hali	Bangalore	120000
03	Sav	103	Kormangla	Bangalore	6500

Table 12 : Relation for site in Mumbai

Acc_num	Type	Id	Branch	City	Balance
04	Sav	104	R.K.Puram	Mumbai	32100

From the above result we can see that this technique has successfully fragmented the account relation and allocated the fragments among the sites of the distributed system. As we have only taken highest valued attribute from ALP table, no unwanted fragments are created. For simplicity we have considered only four sites of the system for allocation. It is worth mentioning that this fragmentation technique will work in the same way for large number of sites of any distributed system.

Using this 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 DBMS can be improved significantly by avoiding frequent remote access and high data transfer among the sites. We use this technique as part of our grid fragmentation proposal later in the chapter 6. We also implement this technique as part of Cisco GSS and compare the results with other techniques in chapter 7.

CHAPTER 5

Vertical Fragmentation

In the previous chapter we studied the 1st part of our proposal. Now we will study various techniques for vertical fragmentation which forms the 2nd part of it.

Vertical Partitioning is the process of subdividing the attributes of a relation into groups, creating fragments. Vertical partitioning is used to store the most closely accessed attributes in the primary memory. During distributed database design, fragments are allocated and replicated at different sites that improve the performance of transaction processing. For better performance, fragments must be closely matched with the transaction requirements.

First of all, in this chapter we explain the Bond Energy Algorithm [2] to order the attribute in a proper manner on the basis of attribute affinity and then we use Binary partitioning algorithm [4] to make partition from that order [12]. Second approach is similar to the second approach used for horizontal partitioning, where we form attribute affinity graph and then partition it using modified graph partitioning algorithm. Finally we explain a cost model that will address the problem of allocation along with fragmentation [7]. Basis of this approach lies in the need or request of a particular attribute from a particular site.

5.1 Bond Energy Algorithm & Binary Tree Partitioning

Algorithms such as Bond Energy Algorithm (BEA) and Binary Vertical Partitioning Algorithm use the attribute affinity matrix (AAM) formed from the attribute usage matrix (AUM). This technique was proposed by J. Muthuraj, S.Chakravarthy, R. Varadarajan and S.B.Navathe in [12]. Complete method has been divided into two major steps. First step is BEA which groups the attributes and form clustered affinity matrix. Second step selects the best partition out of all the possible by calculating the quality of each split.

5.1.1 Some Definitions

Attribute usage matrix (AUM) gives the use of each attribute by the all the queries. Attribute affinity measures the bond between two attributes of a relation according to how they are accessed by applications. AAM is obtained from Attribute Usage Matrix (AUM). Attribute affinity between attributes i and j is defined as

$$Aff_{ij} = \sum_{t=1}^{T} q_{t,ij} \quad \text{...}(5.1)$$

Where $q_{t,ij}$ is the number of accesses of transaction t referencing both attributes i and j.

The bond between two attributes i and j is defined as:

$$bond_{ij} = \sum_{z=1}^{n} Aff_{zi} * Aff_{zj} \quad \text{...}(5.2)$$

Contribution is defined as the value of placing an attribute between other two attributes. Our aim is to maximize the contribution at each step. The contribution of placing the attribute k between i and j is

$$cont_{ikj} = bond_{ik} + bond_{kj} - bond_{ij} \quad \text{.....................................}(5.3)$$

where $bond_{0i} = 0$ and $bond_{j0} = 0$.

The last set of conditions takes care of the cases where an attribute is being placed in CAM to the left of the leftmost attribute or to the right of the rightmost attribute during column permutations, and prior to the topmost row and following the last row during row permutations.

5.1.2 Example

For explaining this approach we use AUM given in the table 13.

Table 13 : Attribute Usage Matrix

Attributes / Queries	Name (A1)	Family (A2)	Age (A3)	Position (A4)	Location (A5)
T1	21	21	21	0	0
T2	0	0	24	24	0
T3	0	90	0	90	90
T4	0	0	11	0	11

The AAM matrix obtained from the given AUM using equation (5.1) is shown in the table 14.

Table 14 : Attribute Affinity Matrix

Attributes	A1	A2	A3	A4	A5
A1	21	21	21	0	0
A2	21	111	21	90	90
A3	21	21	56	24	11
A4	0	90	24	114	90
A5	0	90	11	90	101

5.1.3 Bond Energy Algorithm

This algorithm takes as input the attribute affinity matrix, permutes its rows and columns, and generates a clustered affinity matrix (CAM). The permutation is done in such a way to maximize the contribution.

Generation of the Clustered Affinity Matrix is done in three steps:

1. **Initialization**: Place and fix one of the columns of AAM arbitrarily into CAM.
2. **Iteration**: Pick each of the remaining n-i columns (where i is the number of columns already placed in CAM) and try to place them in the remaining i+1 positions in the CAM matrix. Choose the placement that makes the greatest contribution to the global affinity measure described above. Continue this until no more columns remain to be placed.
3. **Row Ordering**: Once the column ordering is determined, the placement of the rows should also be changed so that their relative positions match the relative positions of the columns.

Initially place attributes A1 and A2 in the same order. Now we place rest of the attributes using the above equations (5.2) and (5.3). Complete procedure of calculating and ordering attribute is shown in the figure 16.

Figure 16 : Ordering attributes using BEA

Place A3
Contribution as pos 0 = 2058
Contribution as pos 1 = 5943
Contribution as pos 2 = 7098
Attribute A3 is placed at pos 3 : [A1 A2 A3]

Place A4
Contribution as pos 0 = 2394
Contribution as pos 1 = 28035
Contribution as pos 2 = 28716
Contribution as pos 3 = 6960
Attribute A4 is placed at pos 2 : [A1 A2 A4 A3]

Place A5
Contribution as pos 0 = 2121
Contribution as pos 1 = 26319
Contribution as pos 2 = 26271
Contribution as pos 3 = 26531
Contribution as pos 4 = 5777
Attribute A5 is placed at pos 3 : [A1 A2 A4 A5 A3]

Resulting order [A1 A2 A4 A5 A3]

We use the same ordering for the rows as well and form the clustered affinity matrix as shown in the table 15.

Table 15 : Clustered Affinity Matrix

Attributes	A1	A2	A4	A5	A3
A1	21	21	0	0	21
A2	21	111	90	90	21
A4	0	90	114	90	24
A5	0	90	90	101	11
A3	21	21	24	11	56

The Bond Energy Algorithm is used to group the attributes of a relation based on the attribute affinity values in AAM. It is considered appropriate for the following reasons [12]:

- It is designed specifically to determine groups of similar items as opposed to a linear ordering of the items. (i.e. It clusters the attributes with larger affinity values together, and the ones with smaller values together).
- The final groupings are insensitive to the order in which items are presented to the algorithm.
- The AAM is symmetric, and hence allows a pair wise permutation of rows and columns, which reduces complexity.
- Because of the definition of Aff_{ij}, the initial AAM is already semiblock diagonal, in that each diagonal element has a greater value of any element along the same row or column.
- The computation time of the algorithm is reasonable. $O(n^2)$, where n is the number of attributes.

The following algorithm extends bond energy algorithm to identify all the clusters in the CAM matrix

5.1.4 Binary Vertical Partitioning

Now we have determined the order of attributes & next step is fragmenting using this order of attributes. For this we have the algorithm called Binary Vertical Partitioning [4]. The approach taken behind this algorithm is splitting rather than grouping. The rationale behind this approach is that the "optimal" solution is much closer to the group composed of all attributes, assumed to be the starting point, than to groups that are single attribute partitions. The objective of splitting activity is to find sets of attributes that are accessed solely, or for the most part, by distinct sets of applications. The binary vertical partitioning algorithm uses the clustered affinity matrix to partition an object into two non-overlapping fragments. When the CAM is big, usually more than two clusters are formed and there is more than one candidate partition.

Assume that a point x is fixed along the main diagonal of the CAM, as shown in table 16. The point x defines two blocks U (for “upper”) and L (for “lower”). Each block defines a vertical fragment given by the set of attributes in that block.

Table 16 : Clustered Affinity Matrix

Attributes	A1	A2	A4	A5		A3
A1	21	21	0	0		21
A2	21	111	90	90		21
A4	0	90	114	90		24
A5	0	90	90	101		11
					X	
A3	21	21	24	11		56

Let A_t be the set of attributes used by transaction t defined as follows:

$$A_t = (i \mid q_{ti} > 0) \quad \text{……………………………………………………………(5.4)}$$

Using A_t, it is possible to compute the following sets:

$$T = (t \mid t \text{ is a transaction}) \quad \text{…………………………………………………(5.5)}$$

$$LT = (t \mid A_t \subseteq L) \quad \text{……………………………………………………………(5.6)}$$

$$UT = (t \mid A_t \subseteq U) \quad \text{……………………………………………………………(5.7)}$$

$$IT = T - (LT \cup UT) \quad \text{………………………………………………………(5.8)}$$

T represents the set of all transactions. LT and UT represent the set of transactions that “match” the partitioning, as they can be entirely processed using attributes in the lower or upper block, respectively. IT represents the set of transactions that needs to access both fragments.

$$CT = \sum_{t \in T} q_t \quad \text{………………………………………………………………..(5.9)}$$

$$CL = \sum_{t \in LT} q_t \qquad (5.10)$$

$$CU = \sum_{t \in UT} q_t \qquad (5.11)$$

$$CI = \sum_{t \in IT} q_t \qquad (5.12)$$

CT counts the total number of transaction accesses to the considered object. CL and CU count the total number of accesses of transactions that need only one fragment either lower or upper. CI counts the total number of accesses of transactions that need both fragments. Totally n-1 possible locations of point x along the diagonal is considered, where n is the size of the input matrix (i.e. the number of attributes).

A non-overlapping partition is obtained by selecting the point x along the diagonal such that the following objective function z is maximized:

$$max\ z = CL * CU - CL^2 \qquad (5.13)$$

The partitioning that corresponds to the maximal value of the z function is accepted. The objective function shown above comes from an empirical judgment of what should be considered a “good” partitioning. The function is increasing in CL and CU and decreasing in CI. For a given value of CI, it selects CL and CU in such a way that the product CL*CU is maximized. This result in selecting values for CL and CU, that is as nearly equal as possible. Thus the above z function will produce fragments that are “balanced” with respect to the transaction load.

Now we apply binary partitioning algorithm to the order of attribute we obtained in previous section for the given example. In total 10 partitions are possible. Table 17 shows split quality calculation for each of them.

Table 17 : Splitting the Relation by BVP

1	Split at [A1 A2 A4 A5] [A3] Access fragment-1 alone = 90 Access fragment-2 alone = 0 Access fragment-1 & fragment-2 = 56 Split Quality = -3136

2	Split at [A1 A2 A4] [A5 A3] Access fragment-1 alone = 11 Access fragment-2 alone = 0 Access fragment-1 & fragment-2 = 135 Split Quality = -18225
3	Split at [A1 A2] [A4 A5 A3] Access fragment-1 alone = 35 Access fragment-2 alone = 0 Access fragment-1 & fragment-2 = 111 Split Quality = -12321
4	Split at [A1] [A2 A4 A5 A3] Access fragment-1 alone = 0 Access fragment-2 alone = 125 Access fragment-1 & fragment-2 = 21 Split Quality = -441
5	Split at [A1 A2 A4 A3] [A5] Access fragment-1 alone = 45 Access fragment-2 alone = 0 Access fragment-1 & fragment-2 = 101 Split Quality = -10201
6	Split at [A1 A2 A3] [A4 A5] Access fragment-1 alone = 21 Access fragment-2 alone = 0 Access fragment-1 & fragment-2 = 125 Split Quality = -15625
7	Split at [A1 A3] [A2 A4 A5] Access fragment-1 alone = 90 Access fragment-2 alone = 0 Access fragment-1 & fragment-2 = 56 Split Quality = -3136
8	Split at [A1 A5 A3] [A2 A4] Access fragment-1 alone = 11 Access fragment-2 alone = 0 Access fragment-1 & fragment-2 = 135 Split Quality = -18225

9	Split at [A1 A2 A5 A3] [A4]
	Access fragment-1 alone = 32
	Access fragment-2 alone = 0
	Access fragment-1 & fragment-2 = 114
	Split Quality = -12996
10	Split at [A1 A4 A5 A3] [A2]
	Access fragment-1 alone = 35
	Access fragment-2 alone = 0
	Access fragment-1 & fragment-2 = 111
	Split Quality = -12321
Optimal Split : [A1] [A2, A3, A4, A5] with Split Quality = -441	

Table 18 : Example Clustered Affinity Matrix for split 1

Attributes	A1	A2	A4	A5		A3
A1	21	21	0	0		21
A2	21	111	90	90		21
A4	0	90	114	90		24
A5	0	90	90	101		11
					X	
A3	21	21	24	11		56

We obtained two fragments using BVP as shown in the table 17.

5.1.5 Drawback

This algorithm has the disadvantage of not being able to partition an object by selecting out an embedded “inner” block. This disadvantage can be avoided by using the procedure SHIFT, which moves the leftmost column of the AAM to the extreme right, and the topmost row of the matrix to the bottom. SHIFT is called a total of n times, so that every diagonal block gets the opportunity of being brought to the upper left corner in the matrix. When the SHIFT procedure is used, the complexity of the algorithm increases by factor n. Experience has shown that the use of the SHIFT procedure improves the solution of the binary vertical partitioning problem in several cases.

Secondly the overall complexity of the algorithm is very high. Whole process is divided into two major algorithms which make the process tedious.

Thirdly it does not solve the problem of allocation hence it does not go well with the distributed databases. Due to these drawbacks we will not consider this process as part of our final comparison in chapter 7.

5.2 Graph-Partitioning Algorithm

In the previous approach, a clustering algorithm is applied to the AAM. In this approach we consider AAM as a complete graph called affinity graph. This approach is similar to the graphical approach for horizontal partitioning we studied in section 4.2. This method was proposed by S. B. Navathe and M. Ra for VF in [1]. It uses Attribute Affinity Matrix instead of Predicate Affinity Matrix used in horizontal partitioning. Then it forms linearly connected tree & form cycles out of that [12]. Each cycle would be considered as a separate partition. For algorithm refer to section 4.2.

5.2.1 Example

We will not discuss the algorithm details but explain its application with the help of an example. First of all we consider an attribute usage matrix as shown in the table 19, which specifies which attributes are used in which transaction and frequency of transaction.

Table 19 : Attribute Usage Matrix

Trans \ Attr	A_1	A_2	A_3	A_4	A_5	A_6	A_7	A_8	A_9	A_{10}
T_1	25	0	0	0	25	0	25	0	0	0
T_2	0	50	50	0	0	0	0	50	50	0
T_3	0	0	0	25	0	25	0	0	0	25
T_4	0	35	0	0	0	0	35	35	0	0
T_5	25	25	25	0	25	0	25	25	25	0
T_6	25	0	0	0	25	0	0	0	0	0
T_7	0	0	25	0	0	0	0	0	25	0
T_8	0	0	15	15	0	15	0	0	15	15

Table 20 shows the attribute affinity matrix for above attribute usage matrix. Affinity of two attributes is the sum of frequencies of the transactions which access both of these two attributes. Attribute affinity matrix represents the affinity graph. We will apply modified graph partitioning algorithm on this graph and form subsets of attributes. Resultant subset of attributes defines the fragments.

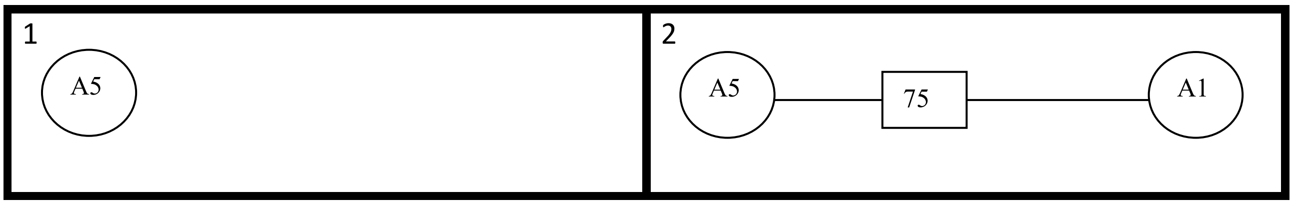
Table 20 : Attribute Affinity Matrix

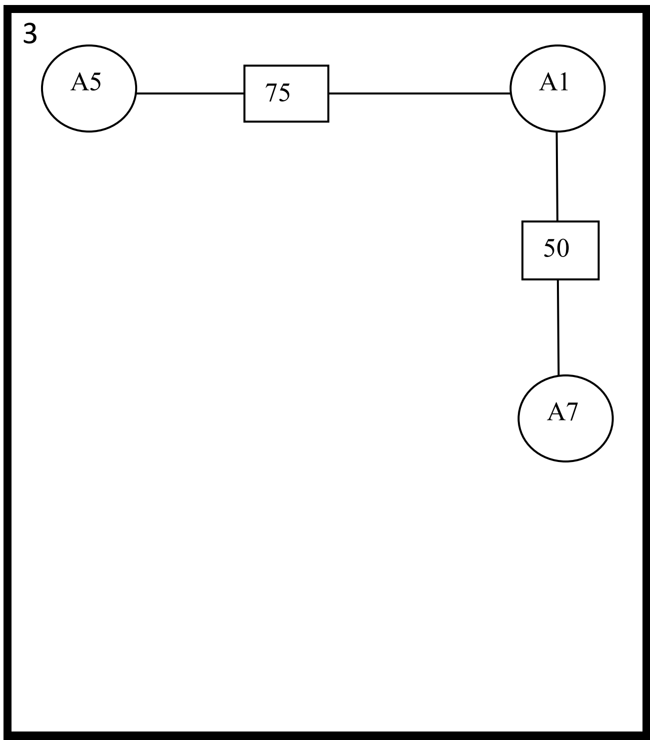
Attr\Attr	A_1	A_2	A_3	A_4	A_5	A_6	A_7	A_8	A_9	A_{10}
A_1	75	25	25	0	75	0	50	25	25	0
A_2	25	110	75	0	25	0	60	110	75	0
A_3	25	75	115	15	25	15	25	75	115	15
A_4	0	0	15	40	0	40	0	0	15	40
A_5	75	25	25	0	75	0	50	25	25	0
A_6	0	0	15	40	0	40	0	0	15	40
A_7	50	60	25	0	50	0	85	60	25	0
A_8	25	110	75	0	25	0	60	110	75	0
A_9	25	75	115	15	25	15	25	75	115	15
A_{10}	0	0	15	40	0	40	0	0	15	40

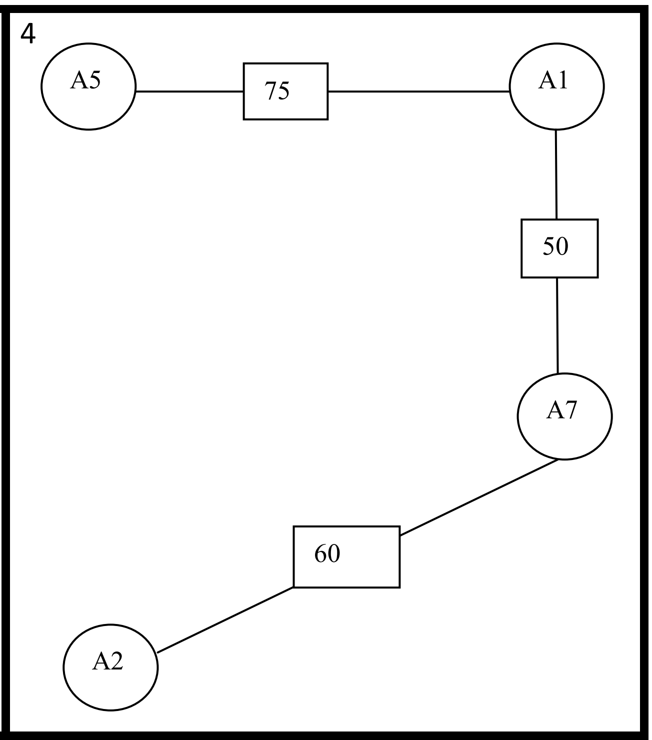
5.2.2 Step by Step Solution

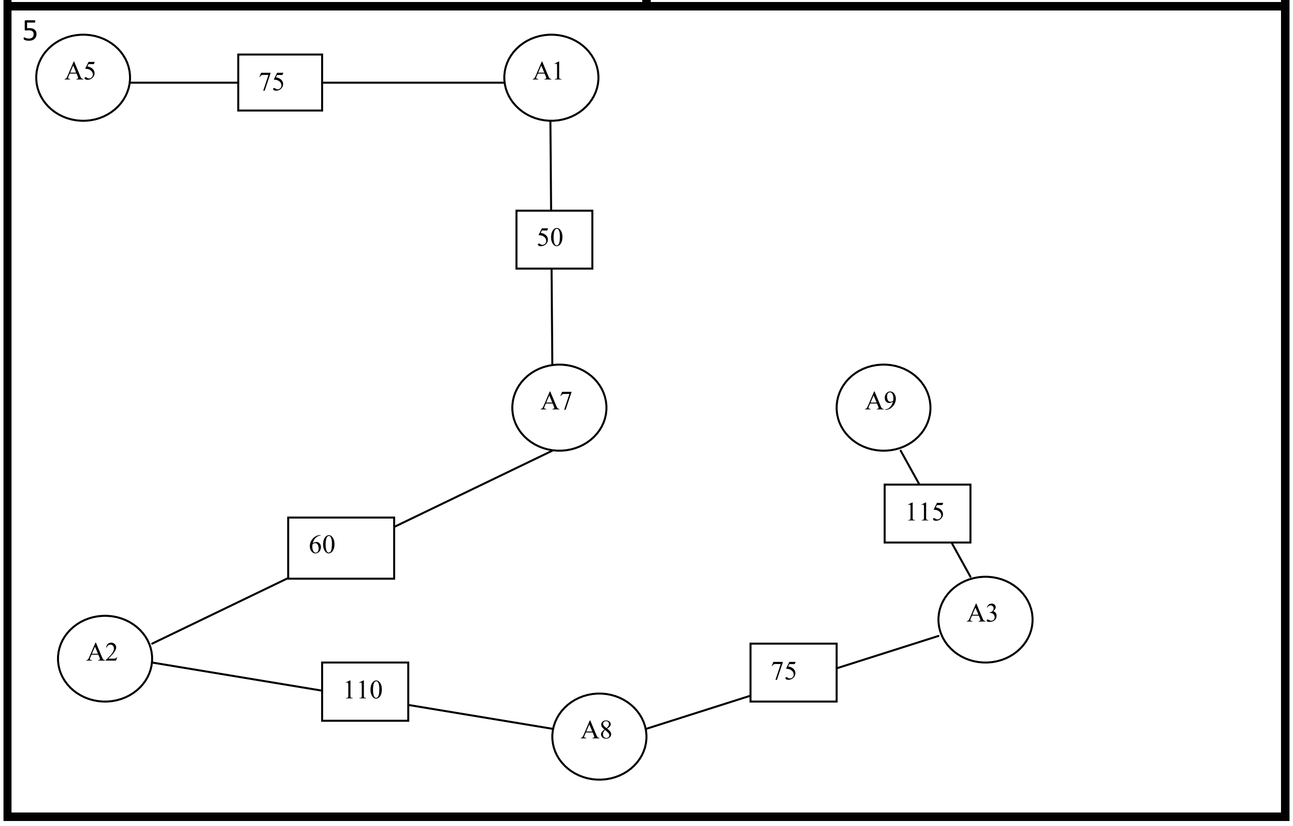
Now we will describe the complete solution of given example graphically using our modified graph-partitioning algorithm. Complete step by step solution is shown in the figure 17.

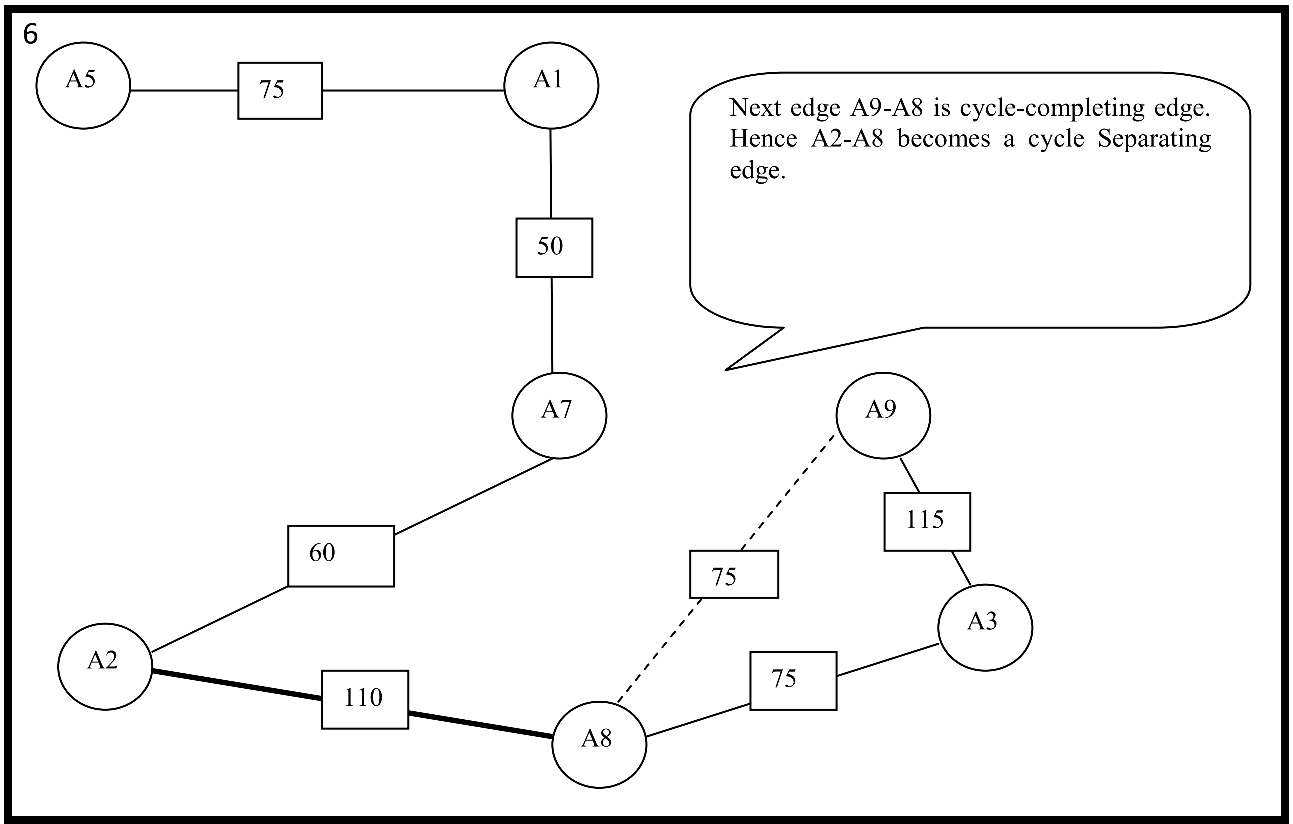
Figure 17 : Graphical Method for VF



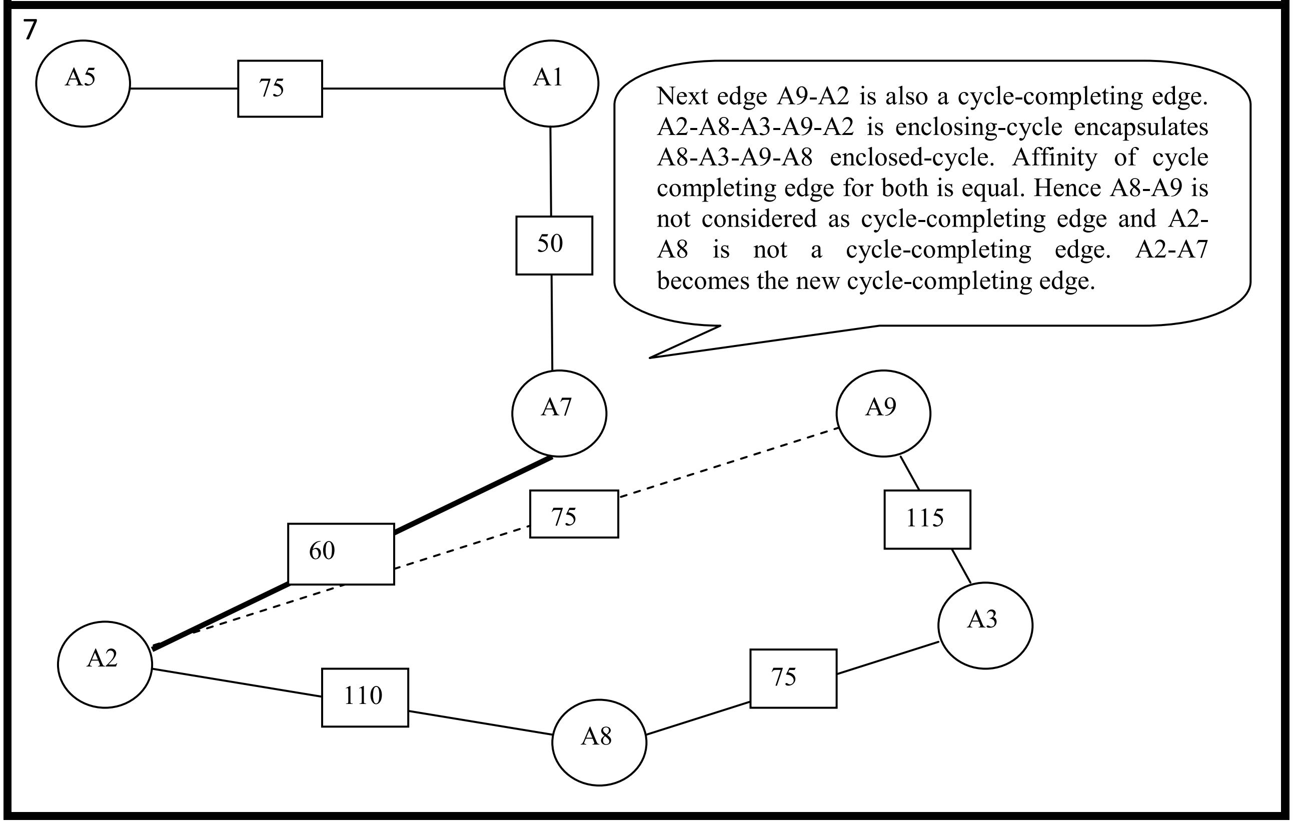

3
A5
75
A1
50
A7


4
A5
75
A1
50
A7
60
A2


5
A5
75
A1
50
A7
A9
115
60
A3
A2
75
110
A8

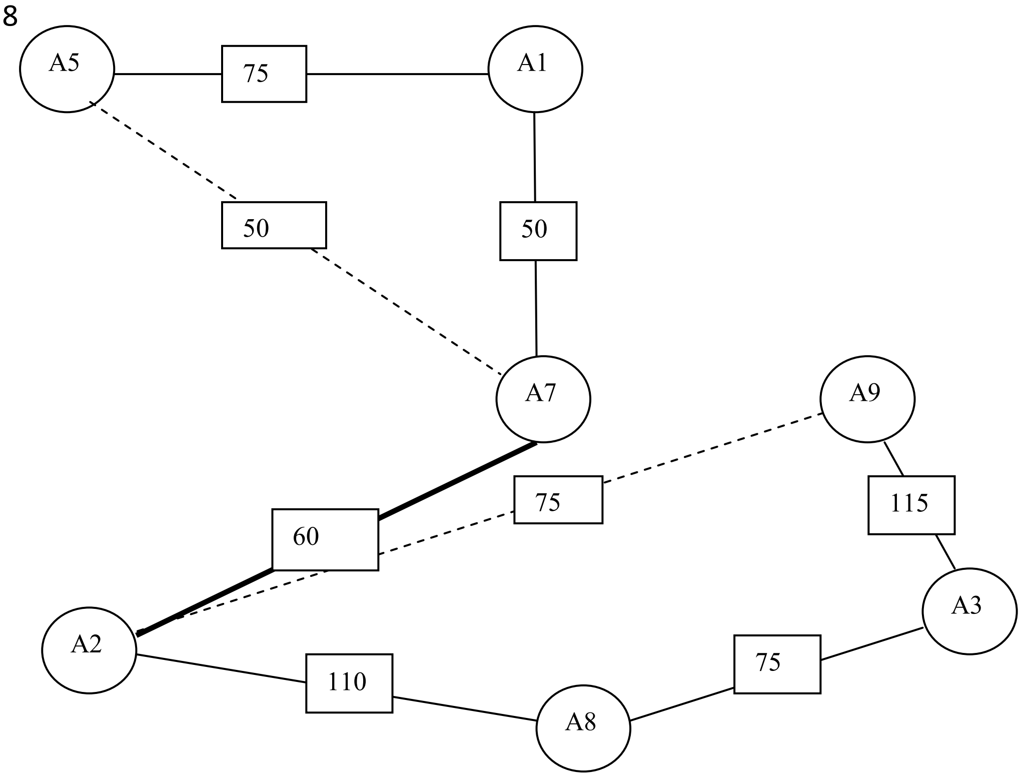


Next edge A9-A8 is cycle-completing edge. Hence A2-A8 becomes a cycle Separating edge.

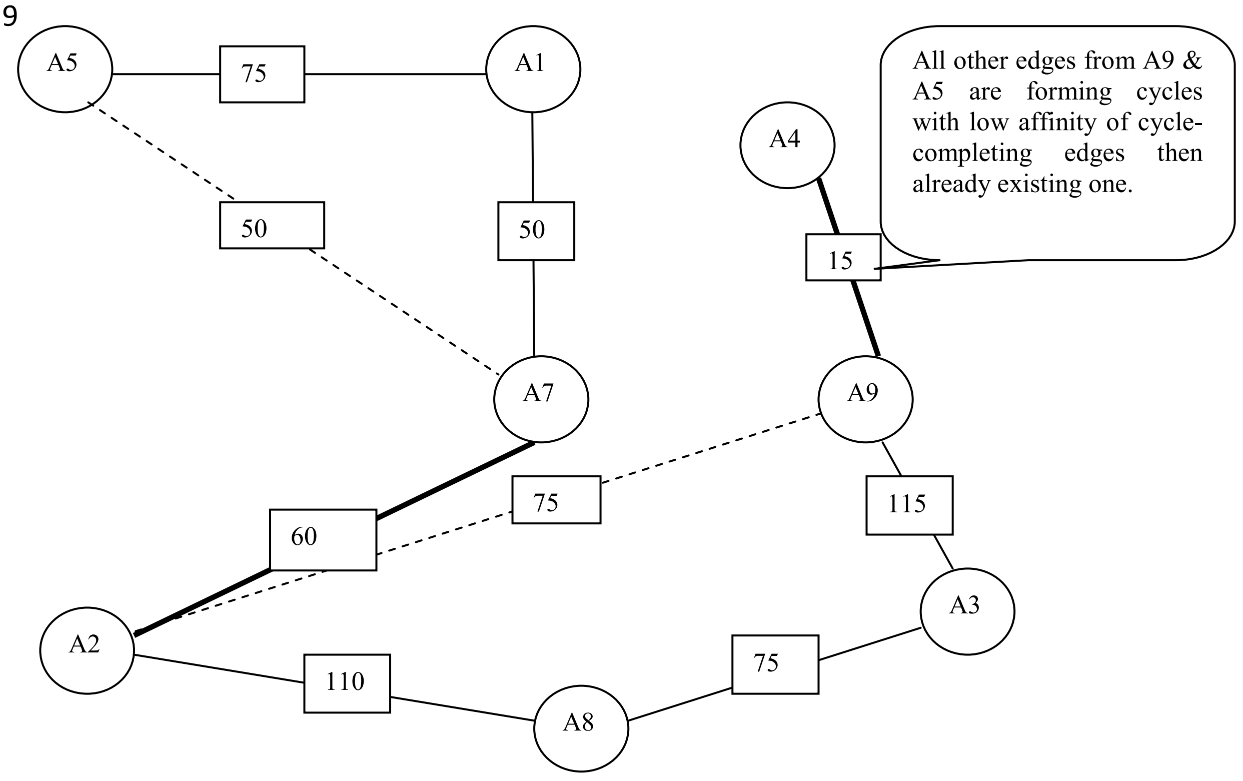


Next edge A9-A2 is also a cycle-completing edge. A2-A8-A3-A9-A2 is enclosing-cycle encapsulates A8-A3-A9-A8 enclosed-cycle. Affinity of cycle completing edge for both is equal. Hence A8-A9 is not considered as cycle-completing edge and A2-A8 is not a cycle-completing edge. A2-A7 becomes the new cycle-completing edge.

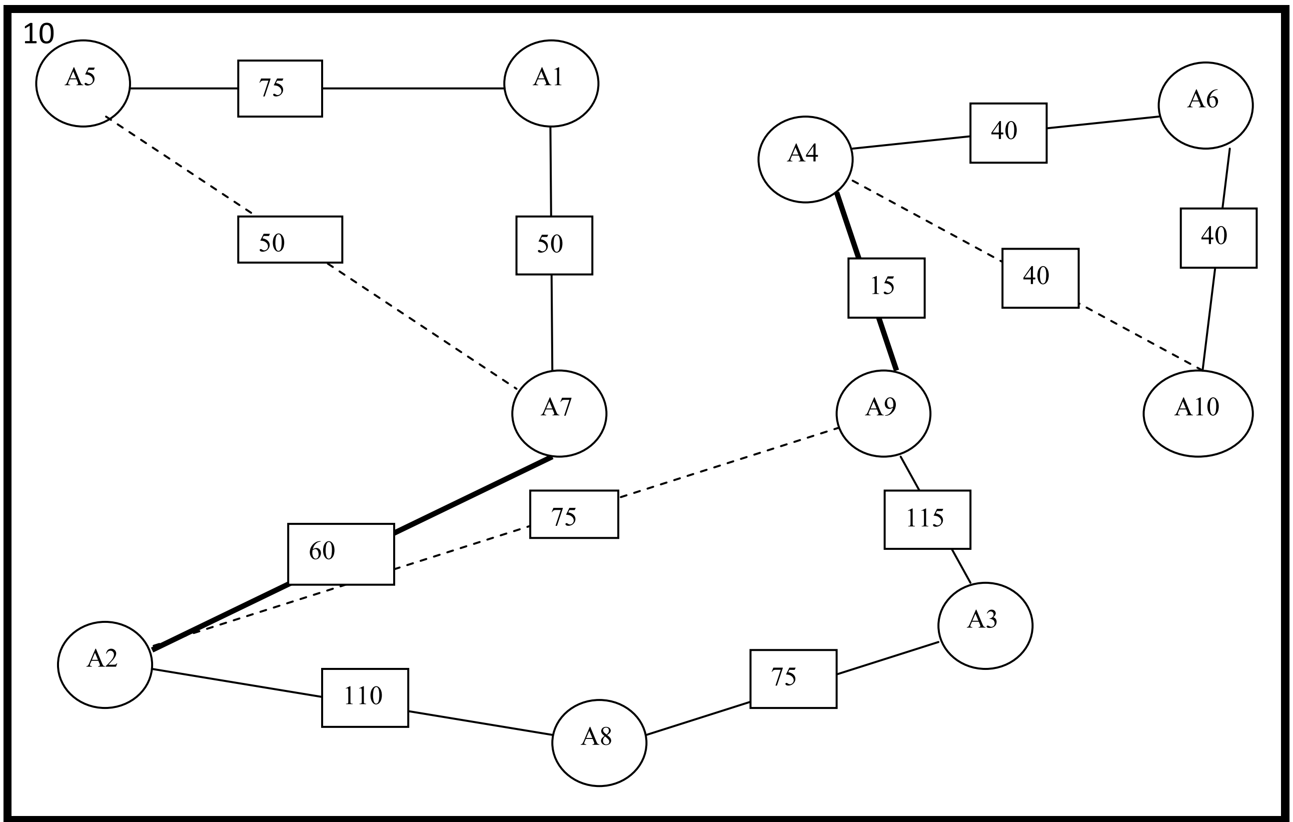
8


A5
75
A1
50
50
A7
A9
75
115
60
A3
A2
75
110
A8

9

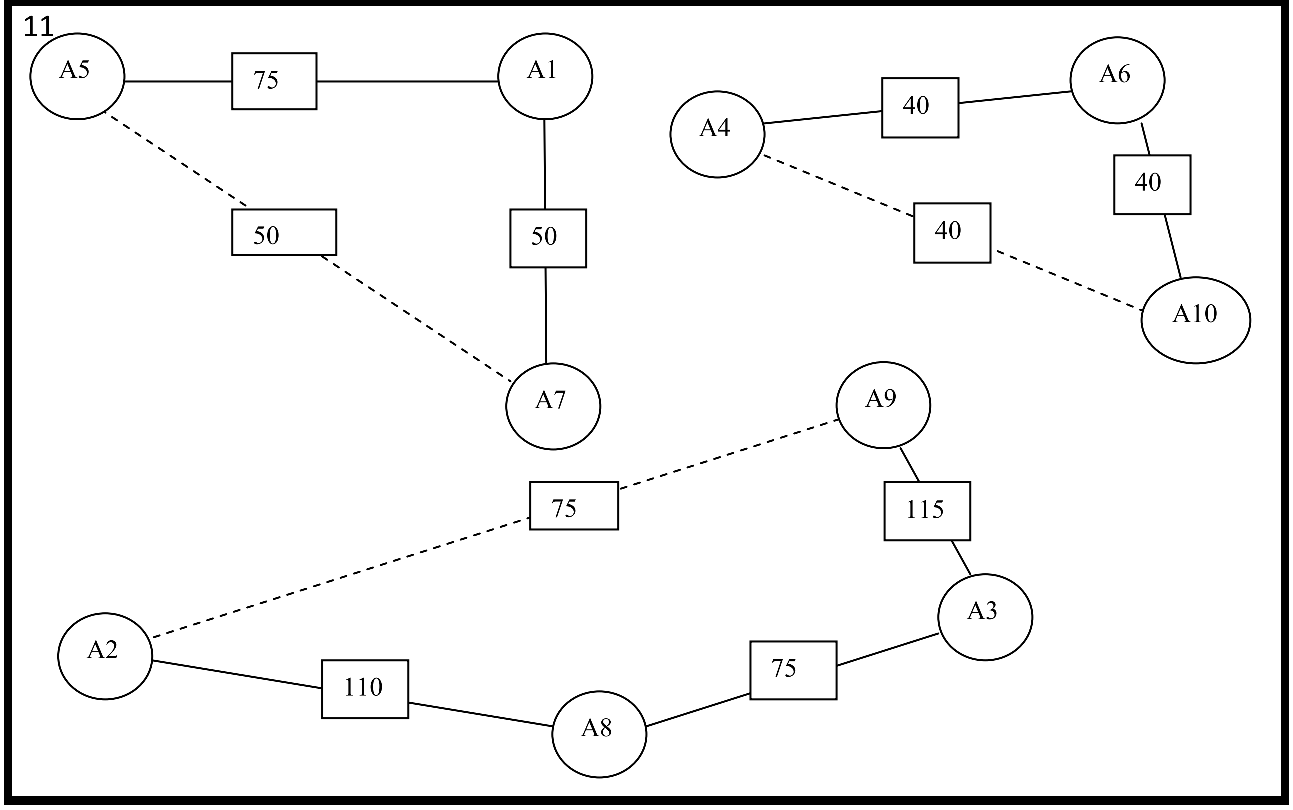

A5
75
A1
All other edges from A9 & A5 are forming cycles with low affinity of cycle-completing edges then already existing one.
A4
50
50
15
A7
A9
75
115
60
A3
A2
75
110
A8

10



A5
75
A1
A6
40
A4
50
50
40
15
40
A7
A9
A10
115
75
60
A3
A2
75
110
A8

11



A5
75
A1
A6
40
A4
40
50
50
40
A10
A7
A9
75
115
A3
75
A2
110
A8

The resulting vertical fragments are:

1) (A_1, A_5, A_7)
2) (A_2, A_3, A_8, A_9)
3) (A_4, A_6, A_{10})

5.2.3 Advantages

We summarize the major advantages of this method over the previous approaches:

1. There is no need for iterative binary partitioning. The major weakness of iterative binary partitioning discussed in the previous section is that at each step two new problems are generated increasing the complexity.
2. The complexity of the approach is $O(n^2)$ as opposed to $O(n^2\log(n))$ of the previous approach.

5.2.4 Disadvantages

Although the overall process is very effective one but it also has some drawbacks. It suits centralized databases, as it does not address the problem of allocation of fragments to different sites.

5.3 Heuristic Approach for VF

In last sections we discussed algorithms for VF using attribute affinities. There are many other algorithms proposed in the literature using attribute affinities. Fragmentation, allocation and replication are database distribution design techniques that aim at improving the system performance. Among the two fragmentation techniques, vertical fragmentation is often considered more complicated than horizontal fragmentation, because the huge number of alternatives makes it nearly impossible to obtain an optimal solution to the vertical fragmentation problem. Therefore, we can only expect to find out a heuristic solution. Often fragmentation and allocation are considered separately, disregarding that they are using the same input information to achieve the same objective, i.e. improve the overall system performance. Here we discuss vertical fragmentation and allocation simultaneously in the context of the relational model. The core of our approach is a heuristic approach to vertical fragmentation, which uses a cost model and is targeted at globally minimizing these costs. This algorithm was proposed by Hui Ma, Klaus-Dieter Schewe and Markus Kirchberg in [7].

We will incorporate all query information, including the site information, by using a simplified cost model for VF. Doing this way, we can obtain vertical fragmentation and fragment allocation simultaneously with low computational complexity and resulting high system performance.

5.3.1 Notations and Definitions

We now define some terms that will be used in our discussion.

- Assume a relation R = $\{a_1, \ldots, a_n\}$ being accessed by a set of queries $Q_m = \{Q_1, \ldots, Q_j, \ldots Q_m\}$ with frequencies $f_1, \ldots, f_m$, respectively.
- To improve the system performance, relation R is vertically fragmented into a set of fragments $\{F_1, \ldots, F_u, \ldots, F_v\}$, each of which is allocated to one of the network nodes $N_1, \ldots, N_h, \ldots, N_k$. Note that the maximum number of fragments is k, i.e.,$v \leq k$.
- We use $\lambda(Q_j)$ to indicate the site that issues query Q_j and use $A_j = \{a_i | f_{ji} = f_j\}$ to indicate the set of attributes that are accessed by Q_j, with f_{ji} as the frequency of the query Q_j accessing attributes a_i. Here, $f_{ji} = f_j$ if the attribute a_i is accessed by Q_j. Otherwise $f_{ji} = 0$.

Input to this cost model for optimal vertical fragmentation is:

- Frequency of queries that access the object. When the same query is issued at different sites it is treated as different queries.
- The subset of attributes used by queries.

- The size of each attribute of the object.
- The site that issue the queries.

To record the above input information we introduce Attribute Usage Frequency Matrix (AUFM) which is similar to AUM. Each row represents one query Q_j; the head of column is the set of attributes of a relation E. In addition, there are two columns with one column indicating the site that issues the queries and the other indicating the frequency of the queries. The values on a column indicate the frequency f_{ji} of the query Q_j that use the corresponding attributes a_i grouped by the site that issues queries. Note that we treat the same query at different sites as different queries. Doing this way we only need one matrix to record all the information rather than two matrices, Attribute Usage Matrix and Access Matrix that are used in our previous approach. Subsequently, the following up calculation is easy to be formulated.

From one site each attribute is requested by multiple queries. The request of an attribute at a site *h* is the sum of frequencies of all queries at the site *h* accessing the attribute. It can be calculated with the formula below:

$$\boldsymbol{request_h(a_i) = \sum_{j=1,\ \lambda(Qj)=h}^{m} f_{ji}} \quad \text{...(5.14)}$$

Let f_{ji} be the frequency of a query accessing an attribute a_i and l_i be the length of this attribute. The *need* of this attribute at a site *h* is calculated with the following formula:

$$\boldsymbol{need_h(a_i) = l_i * \sum_{j=1,\ \lambda(Qj)=h}^{m} f_{ji}} \quad \text{...(5.15)}$$

$$\boldsymbol{need_h(a_i) = l_i * request(a_i)} \quad \text{..(5.16)}$$

Finally, a term pay to measure the costs of allocating a single attribute to a network node. The pay of allocating an attribute a_i to a site h measures the costs of accessing attribute a_i from all queries at the other sites h` which is different from h. It can be calculated using the following formula:

$$\boldsymbol{pay_h(a_i) = \sum_{h`=1, h \neq h`}^{k} request_{h`}(a_i) * c_{hh`}} \quad \text{.......................(5.17)}$$

Note that the cost factor $c_{hh'} = 0$, if h = h’.

In distributed databases, costs of queries are dominated by the cost of data transportation from a remote site to the site that issues the queries. To compare different vertical fragmentation scheme we would like to compare how it affect the transportation costs.

Taking the simplified cost model we now analyze the relationships between *cost*, the *pay* and the *request*. We compute the following formulae:

$$cost_h(a_i) = \sum_{h`=1, h \neq h`}^{k} need_{h`}(a_i) * c_{hh`} \quad \ldots\ldots\ldots\ldots(5.18)$$

$$= l_i * \sum_{h`=1, h \neq h`}^{k} request_{h`}(a_i) * c_{hh`} \quad \ldots\ldots\ldots\ldots(5.19)$$

$$cost_h(a_i) = l_i * pay_h(a_i) \quad \ldots\ldots\ldots\ldots(5.20)$$

The above formula give rise to two alternative heuristics for the allocation of an attribute a_i (i = 1, . . . , n).

- The first heuristic allocates a_i to a network node N_w such that $pay_w(a_i)$ is minimal, i.e., we choose a network node in such a way that the total transport costs for all queries arising from the allocation are minimized.
- The second heuristic allocates a_i to a network node N_w such that $request_w(a_i)$ is maximal. i.e., we choose the network node with the highest request of the attribute a_i. This guarantees that there is no transportation cost associated with data of attribute a_i for those queries that need the data of a_i most frequently. In addition, the availability of data of attribute a_i will be maximized.

5.3.2 Algorithm

Taking the first heuristic we perform vertical fragmentation with the following steps. We do not distinguish read and write queries. Take the most frequently used 20% queries Q_n.

1. Optimize all the queries and construct an AUFM for each relation based on the queries.
2. Calculate the request at each site for each attribute to construct an Attribute request matrix using equation (5.14).
3. Calculate the pay at each site for each attribute to construct an attribute pay matrix using equation (5.17).
4. Cluster all attributes to the site which has the lowest value of the pay.
5. Attach the primary key to each fragment.

5.3.3 Example

We take the example problem [1] to illustrate how our approach works. Firstly, we take the Attribute Usage Matrix and Attribute Access Matrix to construct an AUFM grouped by site that issues the queries. The AUFM is shown in Table 21.

Table 21 : Attribute Usage Frequency Matrix

Site	Query	Frequency	a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8	a_9	a_{10}
1	T_1	10	10	0	0	0	10	0	10	0	0	0
1	T_2	10	0	10	10	0	0	0	0	10	10	0
1	T_4	10	0	10	0	0	0	0	10	10	0	0
1	T_6	10	10	0	0	0	10	0	0	0	0	0
1	T_5	5	5	5	5	0	5	0	5	5	5	0
1	T_7	5	0	0	5	0	0	0	0	0	5	0
1	T_8	5	0	0	5	5	0	5	0	0	5	5
2	T_2	20	0	20	20	0	0	0	0	20	20	0
2	T_1	15	15	0	0	0	15	0	15	0	0	0
2	T_5	10	10	10	10	0	10	0	10	10	10	0
2	T_7	10	0	0	10	0	0	0	0	0	10	0
2	T_6	5	5	0	0	0	5	0	0	0	0	0
2	T_8	5	0	0	5	5	10	5	0	0	5	5
3	T_3	15	0	0	0	15	0	15	0	0	0	15
3	T_4	15	0	15	0	0	0	0	15	15	0	0
3	T_2	10	0	10	10	0	0	0	0	10	10	0
3	T_5	5	5	5	5	0	5	0	5	5	5	0
3	T_6	5	5	0	0	0	5	0	0	0	0	0
3	T_7	5	0	0	5	0	0	0	0	0	5	0
3	T_8	3	0	0	3	3	0	3	0	0	3	3
4	T_2	10	0	10	10	0	0	0	0	10	10	0
4	T_3	10	0	0	0	10	0	10	0	0	0	10
4	T_4	10	0	10	0	0	0	0	10	10	0	0
4	T_5	5	5	5	5	0	5	0	5	5	5	0
4	T_6	5	5	0	0	0	5	0	0	0	0	0
4	T_7	5	0	0	5	0	0	0	0	0	5	0
4	T_8	2	0	0	2	2	0	2	0	0	2	2

Secondly, we compute the request using equation (5.14) for each attribute at each site and get the Attribute request Matrix shown in Table 22.

Table 22 : Attribute *request* Matrix

Site	a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8	a_9	a_{10}
1	25	25	25	5	25	5	25	25	25	5
2	30	30	45	5	30	5	25	30	45	5
3	10	30	23	18	10	18	20	30	23	18
4	10	25	22	12	10	12	15	25	22	12

Thirdly, assuming we have been given the values of transportation cost factors in Table 23. We can now calculate the pay of each attribute at each site by putting the values of the request given in Table 22 and values of cost factors given in Table 23 in equation (5.17).

Table 23 : Transportation Cost Factors

Site	1	2	3	4
1	0	10	25	20
2	10	0	20	15
3	25	20	0	15
4	20	15	15	0

The resultant attribute pay matrix is shown in Table 24.

Table 24 : Attribute *pay* Matrix

Attribute	1	2	3	4	5	6	7	8	9	10
$pay_1(a_i)$	750	1550	1465	740	750	740	1050	1550	1465	740
$pay_2(a_i)$	600	1225	1040	590	600	590	875	1225	1040	590
$pay_3(a_i)$	1375	1600	1855	405	1375	405	1350	1600	1855	405
$pay_4(a_i)$	1100	1400	1520	445	1100	445	1175	1400	1520	445

Finally, for each attribute we compare all the pay at all sites to find the minimal one. We subsequently allocate attribute a_i to the site with minimal pay. The allocation of attributes to sites is shown in Table 25.

Table 25 : Attribute Allocation

Attribute a_i	1	2	3	4	5	6	7	8	9	10
Site N_j	2	2	2	3	2	3	2	2	2	3

Therefore, relation R has been fragmented into two fragments with $F_1 = \{a_1, a_2, a_3, a_5, a_7, a_8, a_9\}$ and $F_2 = \{a_4, a_6, a_{10}\}$, which have been allocated to site 2 and 3 respectively.

5.3.4 Advantages of this Approach

The advantages of this heuristic approach for vertical fragmentation and allocation are:

- Except primary-key attributes, there is no overlap among all the vertical fragments. Therefore, we do not need extra procedure to remove overlaps.
- The change of queries will be reflected by the fragmentation solution. Query information may reflect the needs to retain attributes from some sites more often than some other sites. Even though on the affinity graph the cutting edges will be same.
- The complexity of this approach is low. Let m be the number of queries, n be the number of attributes, k be the number of network nodes. The complexity of our approach, which deals with vertical fragmentation and allocation, is $O(m * n + k^2 * n)$, while the complexity of graphical approach is $O(n^2 * m + k * n)$ for the whole design procedure, including building the affinity matrix, vertical fragmentation and allocation.
- This approach suits the situation that for each relation the number of attributes is small and the number of queries is big. Usually, the number of queries taken into consideration is bigger than the number of attributes of a relation.

We have studied various VF techniques and heuristic approach is the only one which addresses the problem of allocation. Hence the heuristic approach to vertical fragmentation forms 2nd half of our grid fragmentation proposal.

CHAPTER 6

Grid Fragmentation

In the last two chapters the algorithms for generating the vertical and horizontal fragmentation schemes have been described. We discussed three algorithms for each vertical and horizontal fragmentation. In this chapter we discuss grid fragmentation. One of the methodology for grid fragmentation is proposed by Shamkant B. Navathe, K Karlapalem,and M. Ra in [5]. This technique does not address the problem of allocation. Second we proposes a new technique for grid fragmentation using techniques for HF and VF discussed in previous chapters.

The grid cells are generated by either applying the horizontal fragmentation scheme on each of the vertical fragments or by applying the vertical fragmentation scheme on each of the horizontal fragments. If the vertical fragmentation scheme generates *n* vertical fragmentation and the horizontal fragmentation scheme generates *m* horizontal fragments, then *n* x *m* grid cells will be generated.

6.1 Representation Scheme for Fragments

A grid is created by applying both the horizontal and vertical fragmentation schemes on the relation. Let $H = \{1, 2,..., n\}$ be set of horizontal fragments and $V = \{a, b,..., m\}$ be the set of vertical fragments of a relation respectively. Each grid cell belongs to exactly one horizontal and one vertical fragment of the relation. The set of grid cells represented as $(1_a, 1_b, . . . , 1_m ; 2_a, 2_b, . . . , 2_m ; . . . ; n_a, n_b, . . . , n_m)$.

The set of vertical fragments of a horizontal fragment form *horizontal grid cells*. For a horizontal fragment *p* they are represented as (p_a, p_b, . . . , p_m). In figure 18, the set (2_a, 2_b, . . . , 2_m) forms the set of horizontal grid cells for the horizontal fragment 2. The set of horizontal fragments of a vertical fragment are known as vertical grid cells of that fragment. The vertical grid cells of a vertical fragment *i* are represented as (1_i, 2_i, . . . , n_i). In figure 18 the set of grid cells (1_b, 2_b, . . . , n_b) form the vertical grid cells for the vertical fragment b.

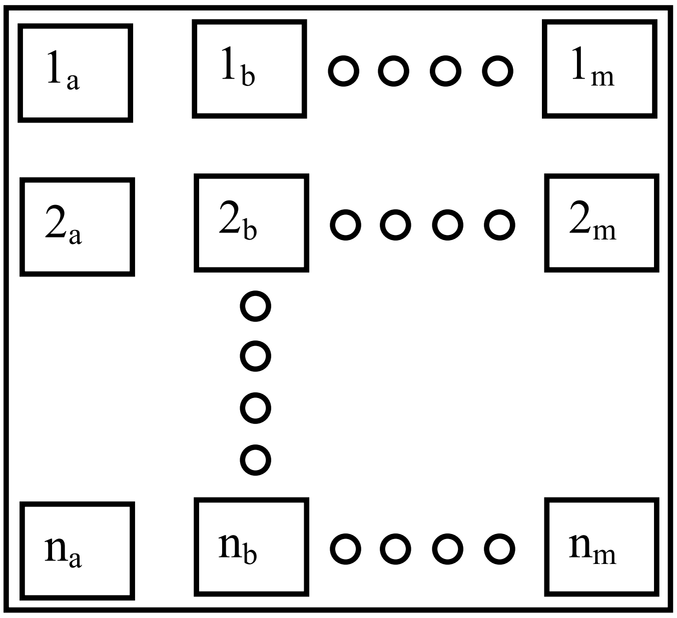


Figure 18 : Representation of Grid cells

Following are the two binary operations defined over these grid cells [5]:

- Concatenate: the horizontal merging operator. The concatenate operator is a special case of join operator where only corresponding tuple id's of the relation are matched. All the relations involved in the concatenate operation have the same number of tuples and same set of tuple identifiers.

  $(\mathbf{i_p}, \mathbf{i_q}) = \mathbf{i_p} \parallel \mathbf{i_q}$

  Two grid cells are concatenated only if they are horizontal grid cells of the same horizontal fragment.
- Union: the vertical merging operator. All the relation involved in the Union operator have the same set of attributes.

  $(\mathbf{i_p}, \mathbf{j_p}) = \mathbf{i_p} \ \mathbf{U} \ \mathbf{j_p}$

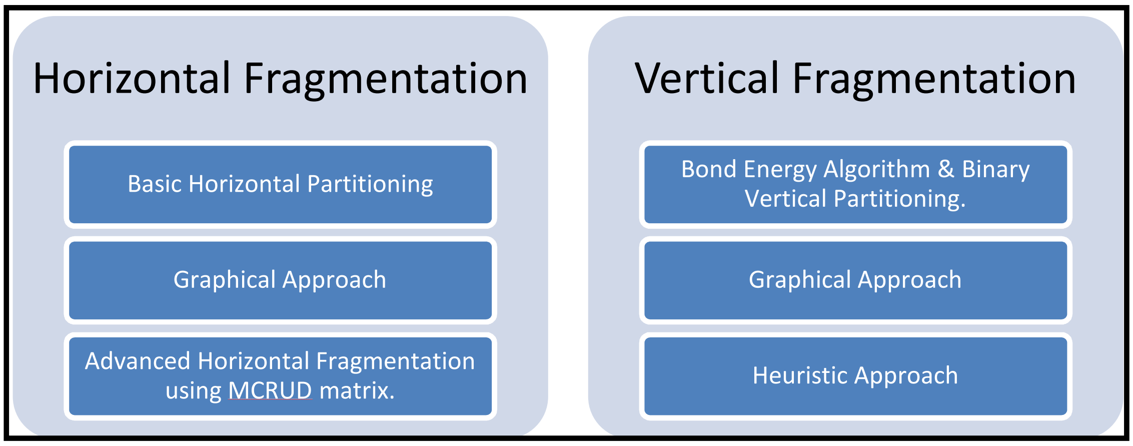
  Union of two grid cells is allowed only if they are vertical grid cells of the same vertical fragment.

Both of these two binary operations are commutative and associative over the set of vertical grid cells and horizontal grid cells.

6.2 Methodology

We have explained three techniques of each Horizontal Fragmentation & Vertical Fragmentation in chapter 4 and 5 respectively. Figure 19 **Error! Reference source not found.**shows these different Horizontal Fragmentation & Vertical Fragmentation techniques that we studied. Next step is to apply HF algorithm and VF algorithm together to form a grid on a relation giving rise to a set of grid cells. We can form various combinations of HF and VF techniques that can result into grid fragmentation of the relation. As we have explained advantages and disadvantages of various techniques in previous chapters, we can select a particular algorithm for each of the type depending upon the requirements.

Figure 19 : Different Techniques of Fragmentation Studied


Horizontal Fragmentation
Basic Horizontal Partitioning
Graphical Approach
Advanced Horizontal Fragmentation using MCRUD matrix.
Vertical Fragmentation
Bond Energy Algorithm & Binary Vertical Partitioning.
Graphical Approach
Heuristic Approach

Secondly we need not to consider the order of fragmentation either HF first then VF (HV partitioning) or VF first and then HF (VH Partitioning). Both the cases will result into similar grid fragmentation.

While performing grid fragmentation we can consider all the combination but some of the techniques mentioned above have few big drawbacks. Basic horizontal partitioning cannot be implemented because it is more of a theoretical approach. Number of min-terms involved in the intermediate steps makes it very difficult to implement. Similarly complexity of vertical fragmentation using bond energy algorithm and binary partitioning makes it inefficient approach. Overall process is divided into two major parts that makes it even more complex.

Graphical approach for horizontal fragmentation needs further optimization hence combining it with other approaches does not give any further advantage. It does not solve the problem of allocation. A mixed fragmentation technique given in [5] combines graphical approach for HF and VF.

Along with fragmentation we are also aiming allocation of fragments to different sites. Hence the techniques we choose for HF and VF are Advanced HF using MCRUD matrix and Heuristic Approach for VF respectively. We propose advanced grid fragmentation using these two techniques. These techniques both take care of fragmentation & allocation of fragments to different sites simultaneously.

6.3 Advantages of Our proposal

Here we discuss some advantages of our grid fragmentation proposal:

1. Our grid fragmentation model addresses both the distributed databases design problem i.e., fragmentation and allocation.
2. It combines the cost model of both HF and VF. It uses the transaction frequency as input. When frequency changes we can re-fragment the relation.
3. Complexity of both the processes is low as discussed in previous chapters.
4. Both the approaches are mathematical, hence easy to implement.

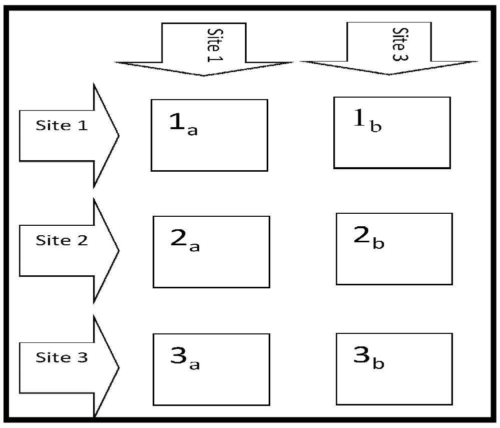
Next we will explain when to allocate fragments in grid fragmentation.

6.4 Allocation of Grid Cells

We are mixing the two techniques to form grid fragments. Each technique has its own allocation scheme. So question is when and how to allocate the fragments to different sites? We can do this in following 3 ways:

1. First of all fragmenting the database using Horizontal Fragmentation and allocating the horizontal fragments to different sites according to horizontal fragmentation schemes only. Then fragmenting vertically the each horizontal fragment at their respective site.
2. Otherwise we can do the reverse of this. Performing vertical fragmentation and then allocating accordingly. Finally fragmenting horizontally each vertical fragment at their respective sites.
3. Third approach that we can take is little bit different where we first perform both the fragmentation over the relation before allocation. Figure 20 shows a relation fragmented with site allocation using arrow heads.

Figure 20 : Grid Allocation Scheme



Fragment 1_a belongs to vertical fragment which is allocated to the site 1 and also belongs to the horizontal fragment which is also allocated to the site 1, hence we can allocate 1_a to site 1. Similarly we can allocate fragment 3_b site 3. For other fragments we can replicate them to both the sites they belong. For example we can keep 1_b at site 1 and 3 both.

We will implement and compare the following techniques as part of our research in Cisco GSS and give the results in the next chapter:

I. Graphical Horizontal Fragmentation using modified graph-partitioning
II. Graphical Vertical Fragmentation using modified graph-partitioning.
III. Horizontal Fragmentation using MCURD matrix.
IV. Vertical Fragmentation Cost Model.
V. Graphical Grid Fragmentation.
VI. Advanced Grid Fragmentation.

CHAPTER 7

Implementation & Results

We have discussed various fragmentation techniques of all types i.e. horizontal, vertical and grid fragmentation in the previous chapters. We also discussed the need for these fragmentation techniques in the global site selector for improving the DNS request resolution process by improving the query processing in database. In this chapter we will explain the results after applying these fragmentation techniques on the distributed databases of Cisco GSS.

As mentioned in the previous chapter we will study the results of applying following fragmentation techniques:

I. Graphical Horizontal Fragmentation using modified graph-partitioning.
II. Graphical Vertical Fragmentation using modified graph-partitioning.
III. Horizontal Fragmentation using MCURD matrix.
IV. Vertical Fragmentation Cost Model.
V. Graphical Grid Fragmentation.
VI. Advanced Grid Fragmentation.

7.1 Configuring GSS

Here we explain how to install a particular fragmentation technique inside GSS. How to configure GSS? Step by step description of the complete process is as follows:

1. **Implementation**: First of all we implement the fragmentation scheme inside the GSS code base. We can implement it for all the databases of GSS.
2. **Compilation**: After successful compilation of the code, it generates an .upg file which is called the image of the GSS.
3. **Installation**: Then we will install the image on the GSS and boot it in run-mode 5. Each GSS runs in various run-modes, where run-mode 5 means that GSS is running perfectly well.
4. **DNS Rule Creation**: After ensuring that GSS is running in run-mode 5 we start creating DNS rules. Each DNS rule specifies many parameters some of them are:
   - source address,
   - domain name,
   - answer IP address to be returned
   - sticky enabled/disabled
   - proximity enabled/disabled
5. **Sending Request**: Once all the DNS rules are created GSS is ready to receive request. We start sending packets from a traffic generating tool outside GSS. The traffic generating tool that we used was Tarantula. This traffic generating tool simulates client sending DNS request. We define the kind of request to be sent and domain name for the request. We can set number of request sent per second. This tool also receives the responses sent by GSS and records the time taken to receive them. For each of the above mentioned techniques we will record the response time by varying the number of requests sent per second. Hence we can measure the performance of the system.

7.2 Results

We followed the above procedure for each fragmentation scheme we mentioned. We configured traffic generator tool to send 100 requests per second in each case for six seconds. This tool also records the number of responses received back each second. Table 26 shows the reading. Last row shows the total time taken to receive all the responses back. Number of requests sent in 6 seconds are 600, so readings are recorded till all the responses are received back.

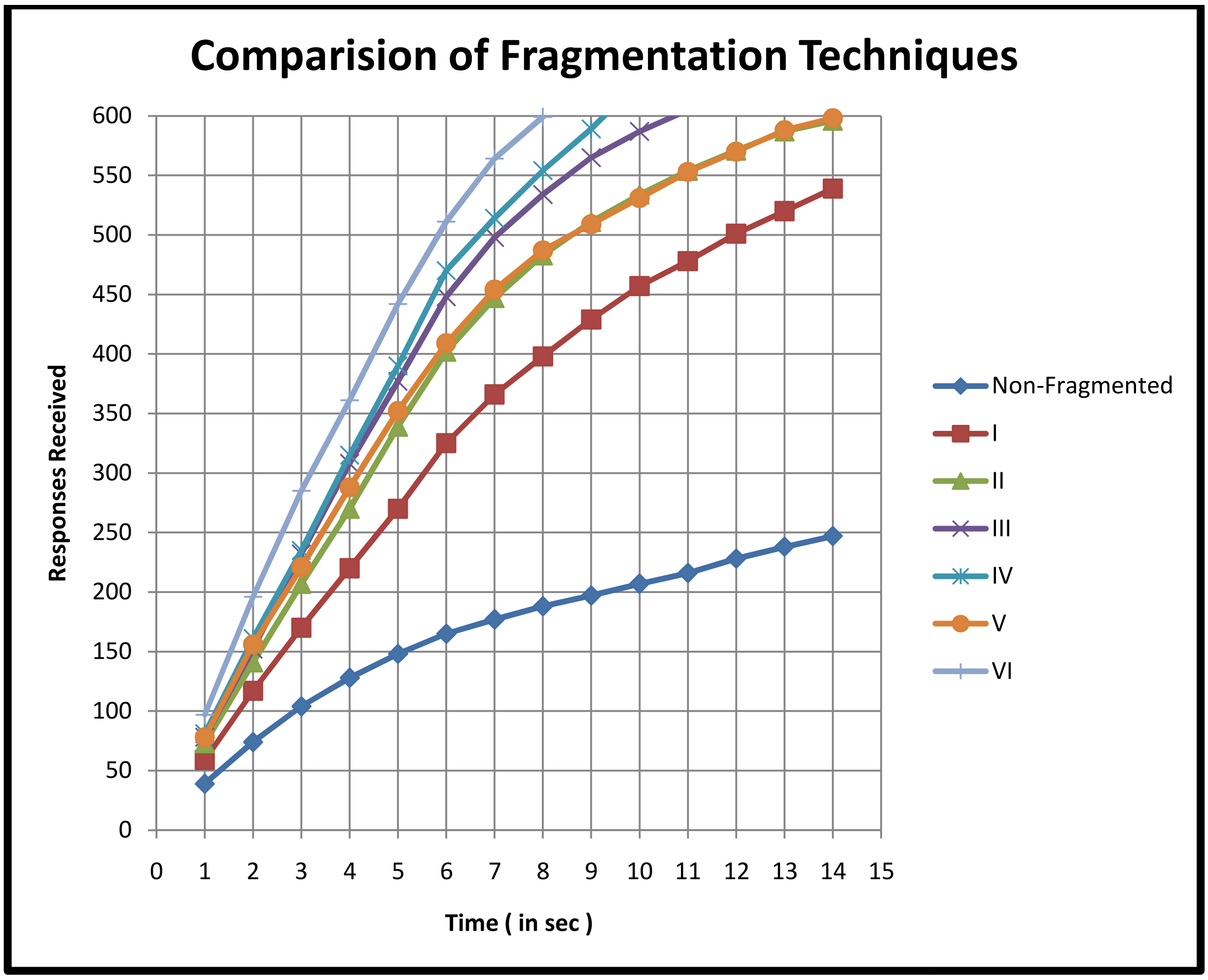
Table 26 : Responses Received Per Second

Fragmentation Schemes / Time (in secs)	Non-Fragmented	I	II	III	IV	V	VI
1	39	59	73	78	81	78	97
2	74	117	141	152	161	156	196
3	104	170	207	232	235	221	285
4	128	220	270	308	315	288	361
5	148	270	339	377	390	352	442
6	165	325	402	448	470	409	511
7	177	366	447	498	514	454	564
8	188	398	483	534	554	487	599
9	197	429	511	565	589	509	
10	207	457	534	587		531	
11	216	478	554			553	
12	228	501	571			570	
13	238	520	587			588	
14	247	539	596			598	
Total Time Taken (in sec)	**72.33**	**16.20**	**14.05**	**10.11**	**9.13**	**14.07**	**8.01**

We plotted the table 26 on a two-dimensional graph as shown in figure 21. Where,

- **x-axis** corresponds to the time-line. Maximum limit for the time-line is set to 15seconds.
- **y-axis** represents number of responses received per second. Maximum limit for this is set to 600 since we sent 600 requests per second.

Figure 21 : Graph showing the Comparison of Fragmentation Techniques



7.2.1 Observations

Following are some of the important observations of the above graph:

1. None of the graphs are perfectly straight line. As the number of requests keeps on increasing, length of the request queues before the databases keep on increasing. This results in slowing down of the overall request resolution. Hence each graph is leaning towards right side.
2. Difference between non-fragmented and fragmented techniques is significant. Blue line corresponds to non-fragmented technique and all other are fragmented. We can observe the gain due to fragmentation.
3. Vertical fragmentation gives the better results than horizontal fragmentation. Reason behind this behavior is that the size of tuple is very low in vertical fragments as compared to horizontal fragments. In horizontal fragment we need to read the complete tuple.

4. Grid fragmentation using graphical method (V- orange line) gives similar results to graphical vertical fragmentation (II – green line). There is no standard procedure for selecting predicate for horizontal fragmentation. As we have many transactions and many attributes are involved in these transactions. So we don't have well defined predicate affinity. Whereas in vertical fragmentation we have well defined attribute affinity. Hence vertical fragmentation dominates over horizontal fragmentation in Grid fragmentation using graphical method.
5. Technique III and IV are better techniques than graphical approach since these techniques address the problem of allocation of fragments simultaneously. Both these techniques are cost models hence give better results.
6. Advanced Grid fragmentation technique (VI – sky blue) which is the combination of III and IV gives the best results. As it is a cost model and considers the actual frequency of the queries for fragmentation. Both techniques involved solve the problem of allocation as well. Hence the database is not centralized that gives a major advantage to this technique. Also size of each fragment and tuple both is decreased.

Coming to the end, we have successfully implemented our grid fragmentation as part of Cisco GSS. Result shows that our technique solves the fragmentation and allocation in distributed databases problem properly.

CHAPTER 8

Conclusion & Future Work

As part of this chapter we conclude our research work with a short summary and also discuss the future scope of this research work.

8.1 Conclusion

In this research work we have proposed a grid fragmentation technique using advance horizontal and vertical fragmentation technique. We also discussed various other fragmentation techniques. The major feature of the proposed approach is that it incorporates fragmentation of relation on the basis of actual cost of fragmentation. It takes frequency of transaction as input. Secondly attribute locality precedence is calculated according to actual frequency of queries.

We also implemented the proposed methodology in distributed database inside Global Site Selector. We compared the results obtained with other techniques we discussed.

Many techniques have been proposed by the researchers using empirical knowledge of data access and allocation. Most of those techniques are either HF or VF. But proper fragmentation at the initial stage has not been addressed yet using mixed fragmentation. We can also use our fragmentation technique at the initial stage of database. To the best of my knowledge no such grid fragmentation technique exists.

We suggested some allocation schemes that can solve the problem of allocation efficiently. Using our technique no additional complexity is added for allocating the fragments to the site of a distributed database as fragmentation is synchronized with allocation. Our technique improves the DDBMS performance significantly by avoiding frequent remote access and high data transfer.

8.2 Future Scope

Further extension of our research is in the direction of grid optimization. We can perform grid optimization using binary operations over grid fragments.

Monitoring the effect of change in transaction and change in data over the fragments and processing of queries. Timing of re-fragmentation is next major decision that depends upon the change in transaction frequency.

Implementing the allocation schemes suggested is another area where this research can be extended. We implemented our fragmentation technique that resolves the allocation problem but the timing of allocation can be varied.

This research can be extended to support fragmentation in distributed object oriented databases as well.

References

[1] S. B. Navathe and M. Ra, “Vertical partitioning for database design: A graphical algorithm” in Proceedings of ACM SIGMOD International Conference on Management of Data, Vol. 14, No. 4, pp. 440-450, 1989.

[2] W. McCormick, P. Schweitzer and T. White, “Problem Decomposition and Data Reorganization by a Clustering technique *Operations Research*,” 20, Sep. 1972.

[3] S. Lu, and K. Fu. “A sentence-to-sentence clustering procedure for pattern analysis” IEEE Transactions on Systems, Man and Cybernetics SMC 8, 381-389.

[4] S. Navathe, S. Ceri, G. Wiederhold, and J. Dou. “Vertical Partitioning Algorithms for Database Design” ACM Transactions on Database Systems, Vol. 9, No. 4, Dec. 1984.

[5] Shamkant B. 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.

[6] Katja Hose, Ralf Schenkel “Distributed Database Systems, Fragmentation and Allocation,” Max-Planck-Institute for Informatik, Cluster of Excellence MMCI. October 28, 2010.

[7] Hui Ma, Klaus-Dieter Schewe and Markus Kirchberg, “A Heuristic Approach to Vertical Fragmentation Incorporating Query Information” in Proc. 7th International Baltic Conference on Databases and Information Systems (DB & IS), pp 69-76, 2006.

[8] Vertical Splitting Bond Energy Algorithm, Exercise by Ali Salehi (BC 143), 2006.

[9] Shahidul Islam Khan & Dr. A.S.M. Latiful Hoque, “A New Technique for Database Fragmentation in Distributed Systems,” International Journal of Computer Application. Volume 5-No. 9, pp 20-24, August 2010.

[10] S. Ceri, M. Negri, and G. Pelagatti, “Horizontal data partitioning in database design,” in Proc. ACM SIGMOD, pp. 128–136, 1982.

[11] M. Ra, “Horizontal partitioning for distributed database design,” In Advances in Database Research, World Scientific Publishing, pp. 101–120, 1993.

[12] J. Muthuraj, S.Chakravarthy, R. Varadarajan, and S.B.Navathe, “A formal approach to the vertical partitioning problem in distributed databases design,” in Proc. of Second International Conference on Parallel and Distributed Information Systems, San Diego, California, 1993.

[13] F. F. Marwa, I. E. Ali, A. A. Hesham, “A heuristic approach for horizontal fragmentation and alllocation in DOODB,” in Proc. INFOS2008, 2008, pp. 9-16.

[14] E. S. Abuelyaman, “An optimized scheme for vertical partitioning of a distributed database,” Int. Journal of Computer Science & Network Security, Vol. 8, No. 1, 2008.

[15] H. Mahboubi and J. Darmont, “Enhancing XML Data Warehouse Query Performance by Fragmentation,” in Proc. ACM SAC09, pp.1555-1562, 2009.

[16] P. Surmsuk, “The integrated strategic information system planning Methodology,” IEEE Computer Society Press, pp. 467-475, 2007.