Efficient Database Distribution using Local Search Algorithm

D. Sivakumar

A Thesis Submitted to

Indian Institute of Technology Hyderabad

In Partial Fulfillment of the Requirements for

The Degree of Master of Technology

Department of Computer Science and Engineering

June 2011

Declaration

I declare that this written submission represents my ideas in my own words, and where ideas or words of others have been included, I have adequately cited and referenced the original sources. I also declare that I have adhered to all principles of academic honesty and integrity and have not misrepresented or fabricated or falsified any idea/data/fact/source in my submission. I understand that any violation of the above will be a cause for disciplinary action by the Institute and can also evoke penal action from the sources that have thus not been properly cited, or from whom proper permission has not been taken when needed.

D. Sivakumar

(Signature)

D. SIVAKUMAR

(D. Sivakumar)

CS09G004

(Roll No.)

Approval Sheet

This Thesis entitled Efficient Database Distribution using Local Search Algorithm by D. Sivakumar is approved for the degree of Master of Technology from IIT Hyderabad

(Dr. Soumya Jana) Examiner
Dept. of Electrical Engineering
IITH

(Dr. Naveen Sivadasan) Examiner
Computer Science and Engineerng
IITH

(Dr. Ch. Sobhan Babu) Adviser
Dept. of Computer Science and Engineerng
IITH

(Dr. Ravindra Guravannavar) Co-Adviser
Dept. of Computer Science and Engineerng
IITH

(Dr. Sri Rama Murty Kodukula) Chairman
Dept. of Electrical Engineering
IITH

Acknowledgements

I am deeply indebted to my adviser Dr. Ch.Sobhan Babu for his constant guidance and support right from my problem selection to thesis documentation. He patiently listened to all the problems I faced during the course of this work and appropriately guided me with his full support.

This work would have been come to halt without the expertise guidance of my co-adviser Dr. Ravindra Guruvannavar in desining and implementation. I thank him for patiently clearing all my novice doubts.

I thank all the faculty members of the Department of Computer Science & Engineering for sharing their views and giving valuable suggestions during the discussion of my work in department seminars.

I thank all my classmates for their friendly support who made the stay at this institute enjoyable, we shared joy and knowledge. I thank all my friends at IIT Hyderabad for the same.

I thank our director Prof. U.B.Desai for his friendly administrative support in getting all our requirements done as quickly as possible.

Finally, I thank all my family members for their constant love and support.

Dedication

To Bhagawan Sri Sathya Sai Baba

Abstract

A problem in railway database is identified. Focus of the problem is to reduce the average response time for all the read and write queries to the railway database. One way of doing this is by opening more than one database servers and distributing the database across these servers to improve the performance. In this work we are proposing an efficient distribution of the database across these servers considering read and write request frequencies at all locations.

The problem of database distribution across different locations is mapped to the well studied problem called Uncapacitated Facility Location(UFL) problem. Various techniques such as greedy approach, LP rounding technique, primal-dual technique and local search have been proposed to tackle this problem. Of those, we are using local search technique in this work. In particular, polynomial version of the local search approximation algorithm is used to solve the railway database problem. Distributed database is implemented using **postgresql** database server and **jboss** application server is used to manage the global transaction. On this architecture, database is distributed using the local optimal solution obtained by local search algorithm and it is compared with other solutions in terms of the average response time for the read and write requests.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Optimization is the area which solves many practical problems. We want to optimize on any task which consumes our resources which may include time, money, etc. Improving the performance of the database is one of the current problems which needs to be addressed. Distributed processing is one approach which improves reliability and performance of the database. In a distributed database system, the database is stored on several computers. Primary concern of distributed database system design is to make 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[2]. Some of the reasons for building distributed database systems are sharing of data, autonomy, and availability.

Data Fragmentation: 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[5]. There are two different schemes for fragmenting a relation: *horizontal* fragmentation and *vertical* fragmentation. Horizontal fragmentation splits the relation by assigning each tuple of relation to one or more fragments. Vertical fragmentation splits the relation by decomposing the scheme of the relation.

In this work, we are considering a problem which is particular to the railway database where we want to optimize on the response time an user gets for his requests. To improve this response time we want to efficiently distribute the database at different locations where each location gets a particular set of rows in the database. Here a particular record may be placed at more than one location(i.e replications are allowed). We will describe in detail this problem in the next chapter.

Here we are horizontally fragmenting the railway database relation based on the empirical data such as frequecy of read and write requests at each site . Above mentioned problem is solved in an indirect way where we map this problem to an already well defined problem in the literature. This well defined problem is facility location problem. There are several variants to this problem. Identified database problem is mapped to a particular variant of the facility location problem called 'Uncapacitated facility location problem'. This uncapacitated facility locaiton problem is widely studied and so many approacahes are proposed to solve this problem. This problem is described in detail in chapter 4.

The advantage we get by mapping the given problem to this particular standard problem called 'Uncapacitated facility location problem' is that we can directly borrow different techniques which

are there to solve this problem along with their analysis. The method we chose to solve the problem is called local search heuristic. Local search is an approximation algorithm to find the solution for the uncapacitated facility location problem in polynomial time. This technique is described in detail in chapter 6.

1.1 Organisation of the thesis

In chapter 2 we clearly and formally stated the railway database problem for which a solution is proposed in this thesis. Chapter 3 descirbes in detail the facility location problem, using which we solve our railway database problem. Chapter 3 elaborates on uncapacitated facility location problem a variant of facility location problem to which our railway database problem is mapped. In chapter 4 we describe how exactly this railway database problem is mapped to the standard uncapacitated facility location problem. Chapter 5 describes the local search technique, by using which we solve the uncapacitated facility location problem. Polynomial local search approximation algorithm is given in chapter 5 to find an local optimal solution for the uncapacitated facility location problem in polynomial time. Finally in chapter 6 we present the results obtained after distributing the table using different solutions. And, locality gap of the local search approximation algorithm is verified practically.

Chapter 2

Railway Database Problem

2.1 Problem definition

The following problem arises for Indian Railways if they plan to use geographically distributed databases to improve the response time of reservation requests. Suppose that Indian Railways has got main table in which they store the information of seat availability on each train. This table is queried and updated very frequently from different locations in which are randomly spread. We want to partition and replicate the table across different sites so that we may improve the response time of requests. Suppose that the number of sites across which we want to distribute our table is fixed.

This process of distribution can be done in many ways. One naive way of doing this is by replicating the whole table across all the sites. One major problem with this type of allocation is that whenever we want to update any particular row in the table we need to update it at all the locations, which is time consuming. Moreover during this updating process we won't be able to read that particular row which is being updated. These issues may cause poor response time if we follow this procedure to distribute our table.

There are some more elements we need to consider while solving this problem. Every region has got different set of demands i.e at each site the number of requests will vary for every row in the table. Even at a given site the number of reads will be different from the number of writes for a particular row in the table.

Therefore, considering all these factors we need to come up with a solution which allocates a subset of rows of main table at all the sites so that the average response time for requests is optimized. Here the problem is defined in a more formal way:

- **T** is a table consisting of n records(rows) $(r_1, r_2, ..., r_n)$.
- $s_1, s_2, ..., s_m$ represents m sites across which we want to distribute our main table T.
- $R(s_i, r_j)$, $W(s_i, r_j)$ represents number of read requests and write requests respectively originating at site s_i for record r_j per unit time(we consider per minute).
- To each site s_i we must assign a subset of records T_i of T so as to minimize the average response time of requests.

- **Response time of a read request** for record r_j at site s_i is **0** if r_j is assigned to site s_i(stored locally) otherwise it is **min** $C(s_i, s_k)$ where s_k is any site to which r_j is assigned and $C(s_i, s_k)$ represents the cost of communication between sites s_i and s_k, generally it is taken proportional to the distance between the sites.

- **Response time of a write request** for record r_j at site s_i is
$\sum 2 * C(s_i, s_k)$ where s_k is any site to which r_j is assigned. To write a record the site must lock all the copies of the records and then update all the copies. Factor 2 is used here in the response time because two-phase commit protocol is used in writing at all the sites where the record is located. This requires twice the communication compared to read request. In next section we give a brief description of two-phase commit protocol.

- Every record must be stored at atleast one site i.e
$T_1 \cup T_2$ ... $\cup T_m$ = T.

Extended problem: This problem can be extended where there exists a maximum load a site can handle. Each site s_i can handle a maximum of c_i requests per unit time.

2.2 Two-phase commit protocol

To ensure atomocity, at all the sites which are involed in a transaction T, T must either commit at all sites or must abort at all sites. This is taken care by two-phase commit protocol [6, 7]. Two-phase commit protocol is a distributed algorithm that coordinates all the processes that participate in a distributed atomic transaction on whether to commit or abort the transaction.

The protocol comprises of two phases viz. commit-request phase and commit phase. One site is designated as the coordinator and the rest of the sites are designated as cohorts. The protocol is initiated by the coordinator after the last step of the transaction has been reached.

2.2.1 Commit-request phase

- The coordinator sends a **query to commit message** to all sites(cohorts) and waits until it has received a reply from all cohorts.
- Each site executes the transaction upto the point where they will be asked to commit. They write an entry to the log.
- Each site replies with an **aggrement messsage** if the transaction at that site is succeeded or an **abort message** if it experiences a failure.

2.2.2 Commit phase

Success

If the coordinator received an agreement message from all cohorts during the commit-request phase:

1. The coordinator sends a commit message to all the cohorts.
2. Each cohort completes the operation, and releases all the locks and resources held during the transaction.
3. Each cohort sends an acknowledgment to the coordinator.
4. The coordinator completes the transaction when all acknowledgments have been received.

Failure

If any cohort votes No during the commit-request phase (or the coordinator's timeout expires):

1. The coordinator sends a rollback message to all the cohorts.
2. Each cohort undoes the transaction using the undo log, and releases the resources and locks held during the transaction.
3. Each cohort sends an acknowledgement to the coordinator.
4. The coordinator undoes the transaction when all acknowledgements have been received.

2.3 Assumptions

After setting up of more than one database servers for railway database, if a user wants to access the railway database(through read or write requests), he is allowed to access only the nearest database server which is currently in condition. For read request, the particular database server fetches the information from its local database if that data exists, otherwise it fetches from the nearest database server where that particular data is located. For write request, the contacted server ensures that the data is updated at all the required locations. The same procedure is followed in the simulation of the distributed database.

Chapter 3

Related Work

3.1 Database Partitioning

The problem of partioning(fragmentation) of database optimally is faced in several scenarions in information system design [18]-[24] . A partition is a division of a logical database or its constituting elements into distinct independent parts. Database partitioning is generally done for improving manageability, performance, availability and reliability reasons.

The decomposition of a relation into fragments permits a number of transactions to execute concurrently. In addition, the fragmentation of relations typically results in the parallel execution of a single query by dividing it into a set of subqueries that operate on fragments. Thus fragmentation can increase the level of concurrency and therefore the system throughput.

There are also difficulties raised by the fragmentation. If the applications have conflicting requirements that prevent decomposition of the relation into mutually exclusive fragments, those applications whose views are defined on more than one fragment may suffer performance degradation. It might be the case where we need to retrieve data from two fragments and then take their join, which is costly. Therefore, minimizing the number of distributed joins is a fundamental fragmentation issue.

3.1.1 Partitioning types

The partitioning can be done by either building separate smaller databases (each with its own tables, indices, and transaction logs), or by splitting selected elements, for example just one table.

Horizontal partitioning: Horizontal partitioning splits the relation by assigning each tuple of relation to one or more partitions. For example, customers with ZIP codes less than 50000 are stored in CustomersEast, while customers with ZIP codes greater than or equal to 50000 are stored in CustomersWest. The two partition tables are then CustomersEast and CustomersWest, while a view with a union might be created over both of them to provide a complete view of all customers.

Vertical partitioning: Vertical partitioning splits the relation by decomposing the scheme of the relation. Normalization also involves this splitting of columns across tables, but vertical partitioning goes beyond that and partitions columns even when already normalized.

3.1.2 Traditional criteria for partitioning

Relational database management systems provide for different criteria to split the database. They take a partitioning key and assign a partition based on certain criteria.

Range partitioning : Selects a partition by determining if the partitioning key is inside a certain range. An example could be a partition for all rows where the column zipcode has a value between 70000 and 79999.

List partitioning A partition is assigned a list of values. If the partitioning key has one of these values, the partition is chosen.

Hash partitioning : The value of a hash function determines membership in a partition. Assuming there are four partitions, the hash function could return a value from 0 to 3. Composite partitioning allows for certain combinations of the above partitioning schemes, by for example first applying a range partitioning and then a hash partitioning. Consistent hashing could be considered a composite of hash and list partitioning where the hash reduces the key space to a size that can be listed.

3.2 Complexity of Partitioning Problem

File allocation problem : The problem of database partitioing is related to file allocation problem if we add some constraints.

3.2.1 File Allocation Problem

Assume that there are a set of fragments F = $\{f_1, f_2, ..., f_n\}$ and a distributed system consisting of sites S = $\{s_1, s_2, ..., s_m\}$ on which a set of applications Q = $\{q_1, q_2, ..., q_q\}$ is running. The **file allocation problem** consists of finding the optimal distribution of F to S.

The optimality can be defined with respect to two measures :

1. **Minimal cost**: The cost function consists of the cost of storing each f_i at a site s_j, the cost of querying f_i at site s_j, the cost of updating f_i at all sites where it is stored, and the cost of data communication. The allocation problem, then, attempts to find an allocation scheme that minimizes a combined cost function. In our problem defined in chapter 2 we are not considering this cost.

2. **Performance**: The other very important aspect we consider in getting optimal solution to this problem is to increase the performace. Two well-known ones are to minimize the response time and to maximize the system throughput at each site.

The optimality measure should include both the performance and the cost factors. One should be looking for an allocation scheme that, for example, answers user queries in minimal time while keeping the cost of processing minimal. A similar statement can be made for throughput maximization.

Let us consider a very simple formulation of the problem. Let F and S be defined as before. For instance, we consider only a single fragment, f_k . We make a number of assumptions and definitions that will enable us to model the allocation problem.

1. Assume that Q can be modified so that it is possible to identify the update and the retrieval-only queries, and to define the following for a single fragment f_k : T = $\{t_1, t_2, ..., t_m\}$ where t_i is the read-only traffic generated at site s_i for f_k, and U = $\{u1, u2, ..., um\}$ where u_i is the update traffic generated at site s_i for f_k .

2. Assume that the communication cost between any two pair of sites s_i and s_j is fixed for a unit of transmission. Furthermore, assume that it is different for updates and retrievals in order that the following can be defined:
   C(T) = { $c_{12}, c_{13}, ..., c_{1m}, ..., c_{m-1,m}$ }
   C(U) = { $c'_{12}, c'_{13}, ..., c'_{1m}, ..., c'_{m-1,m}$}
   where c_{ij} is the unit communication cost for retrieval requests between sites s_i and s_j, and c'_{ij} is the unit communication cost for update requests between sites s_i and s_j.

3. Let the cost of storing the fragment at site s_i be d_i . Thus we can define D = $\{d_1, d_2, ..., d_m\}$ for the storage cost of fragment f_k at all the sites.

4. Assume that there are no capacity constraints for either the sites or the communication links.

Then the allocation problem can be specified as a cost-minimization problem where we are trying to find the set I ⊆S that specifies where the copies of the fragment will be stored. In the following, x_j denotes the decision variable for the placement such that

$$x_j = \begin{cases} \exp x & \text{if fragment } f_k \text{ is assigned to sites} s_j \\ 0 & \text{otherwise} \end{cases}$$

So, the specific problem formulation is :

$$\min \left[\sum_{i=1}^{m} \left(\sum_{j|s_j \in I} x_j u_j c'_{ij} + \min_{j|s_j \in I} c_{ij} \right) + \sum_{i=1}^{m} x_j d_j \right], \tag{3.1}$$

subject to $x_j = 0$ or 1.

There are a number of reasons why simplistic formulations such as the one we have discussed are not suitable for distributed database design. These are inherent in all the early file allocation models for computer networks.

This simplistic formulation of facility location problme is not suitable for distributed database design, since there we need to consider so many other factors like access pattern of the application to the data, cost of integrity enforcement, relation ship with other fragments, cost of enforcing concurrency control mechanism,...etc.

But this formulation matches to the problem we defined in chapter 2 because of the assumptions we made. Only difference between above formulated file allocation problem to the problem defined in chapter 2 is that we are not considering the storage costs in our problem.

The first term in the objective function 3.1 corresponds to the cost of transmitting the updates(write cost) to all the sites where the replicas are situated, and the cost of retrieving the requests at the site. The second term of the objective function calculates the total cost of storing all the duplicate copies of the fragment.

The above formulated problem was proven to be NP-complete by Eswaran[19]. So, the database partition problem which is more complex than file allocation problem is also NP-complete and can be shown.

3.3 Solution Methods

Since our database partitioning problem introduced in chapter 2 is almost similar to the file allocation problem, here we give an overview of the techniques that had been introduced to solve this problem and some related database partitioning problems, rather than the solution methods for general database partionining problem which involves more complex issues which are not required to solve the problem we are considering.

The previous work in the file allocation is mostly based on static distribution, which means allocation does not change with time. Grapa and Belford have shown that a particular solution to this file allocation problem solved a thirty node problem in one hour on an IBM 360/91 computer [33]. The difficulty in optmization is discussed in [34]. The algorithms for static allocation can be divided into two types:

- Mathematical programming
- Exhaustive searches, and heuristics.

3.3.1 Mathematical Programming

Mathematical programming approach has been followed by Chu[18], Casey [25], Levin and Morgan [26], [21] and Chen[22]. Using Casey's formulation, we got the formulation of file allocation problem as

$$\min \left[\sum_{i=1}^{m} \left(\sum_{j|s_j \in I} x_j u_j c_{ij}^{'} + \min_{j|s_j \in I} c_{ij} \right) + \sum_{i=1}^{m} x_j d_j \right], \tag{3.2}$$

subject to $x_j = 0$ or 1.

Optimization problem can be solved by using integer programming techniques [35]. In [13] it is shown that file allocation problem can be formulated into a non linear zero-one programming problem. It is shown that by adding additional constraint equations, these non-linear terms in the objective and constraint equations can be reduced to linear equations. But here updations are not considered. Casey[25] and Levin and Morgan [26], [21] have used the hypercube technique to enumerate over a reduced set of possible solutions in order to find the optimum. However, the approach of using integer programming or exhaustive enumeration is only feasible when the problem size is small. Due to this difficulty, Grapa and Belford have developed some simple conditions to determine whether a copy of a file should be placed at a node [33]. This helps in reducing the complexity of the problem.

3.3.2 Heuristic methods

Heuristics refers to experience-based techniques for problem solving, learning and discovery. Heuristics are reasonable or polynomial time search strategies which do not guarantee optimality.

Firstly, a feasible solution is generated. Then the decision algorithm decides whether to improve the solution or not and how to improve it. An example of a decision algorithm is the add-drop algorithm which perturbs on an existing solution to see if a better solution can be obtained [20]. Other heuristics include the greedy algorithm [27, 28], steepest ascent and subgradient method [28] and clustering algorithm [29].

Since heuristics do not always generate optimal solutions and it is difficult to solve analytically for the average and worst-case behavior, evaluations are generally done by simulations on example cases. Therefore, heuristics may perform unpredictably on unanticipated cases. Even though the file allocation problem is NP- completeness, some special cases of this problem can be solved in polynomial time. Ghosh proposed polynomial time algorithms to distribute a database for parallal searching [36], [30]. Kikuno et al. have found that the placement of multiple files on a tree network with no storage capacity constraint and no update cost is polynomially solvable [31]. It is shown [37] that the number of copies of a file can be optimized very easily to achieve the maximum read throughput when it is assumed that there are infinitely many reads and updates arrive stochastically.

The isomorphism of the file allocation problem and the single commodity warehouse location problem has been shown by Ramamoorthy[41]. Therefore heuristics developed by operations researchers have commonly been adopted to solve the file allocation problem. Examples of these heuristics include knapsack problem solution [38], branch-and-bound techniques [28], and network flow algorithms [39].

Chapter 4

Facility Location Problem

Optimization is very important activity which is done on all the operations any organization is performing. One of these operations includes setting up of resources or facilities in a cost effective manner and which allow efficient access of these facilities from the demand points. Examples of these include setting up of a supply chain of a business, locating essential services such as health care and education, and construction of transportation networks.

One simple faciltiy problem is, suppose a company is considering opening as many as four warehouses in order to serve 12 different regions. Main objective of the company is to minimize the sum of fixed costs associated with opening warehouses as well as the various transportation costs incurred to ship goods from the warehouses to the regions.

Here we give the definition of a faciltiy location problem in general. The goal of facility location problem is to open a set of facilities from the given set of facilites and assign each client to one of the open facilities to satisfy the demands of the clients in a cost effective and efficient manner. There is a cost associated with opening a facility at a given location. We generally assume that there exists a metric among facilities and clients(metric version of the problem) that is distances are defined among the facilities and clients.

4.1 Terms related with the problem

Let **F** denotes set of facilities and **C** denotes set of clients. The distances between a pair of points i,j $\in$ F $\cup$ C is denoted by c_{ij}. These distances define a metric.

Service cost of the solution: Suppose in the solution a client j is served by a facility i then the service cost of the client j is nothing but the distacne c_{ij}. Service cost of the solution is sum of the service costs of all the clients in C.

Facility cost of the solution: Let f_i denotes cost associated with opening a facility at i $\in$ F. The cost of opening all the facilities in the solution is called as facility cost of the solution.

Capacities: Depending on the variant of the facility location problem we are considering there may be a limit on number of clients a facility can serve, this is called as the *capacity* of the facility.

The facility location problem is NP-hard. The proof for the NP-hardness is based on the reduction from the minimal set cover problem which is NP-hard[40].

4.2 Variants of the problem

By varying facility costs, service costs and capacity constraints differently, we will get differnet versions of the facility locatioin problem. Here are the major variants of the facility location problem:

1. k-median problem.
2. Uncapacitated Facility Location(UFL).
3. k-Uncapacitated Facility Location (k-UFL).
4. Facility Location with Soft Capacities (∞-CFL).
5. Universal Facility Locaiotn Problem (UniFL).
6. Budget Constrained k-median problem

Since we will be using the Uncapacitated Facility Location(UFL) version of the problem, we will be elaborating upon it in the next section.

4.3 The uncapacitated facility locatioin problem

Uncapacitated facility location problem[1, 4] is one of the most widely studied variants of the facility location problem. This problem arises mostly when both the facility cost and service cost are incurred only once.

- **Input to the problem:** The set of facilities F, the set of clients C, and the distance metric (c_{ij}). For each i $\in$ F , the cost of opening a facility at i, denoted by f_i, is also given.
- **Output:** Find a set S $\subseteq$ F such that ($\sum_{i \in S} f_i + \sum_{i \in C} dist(i, S)$) is minimized, where dist(i,S) denotes the minimum distance between i and a facility in S(solution). We want to open facilities such that sum of facility cost and service cost is minimized.

Approximation algorithm for UFL was first given by Hochbaum[9] based on the greedy heuristic for the set cover problem and it was proved to have an approximation factor of O(log n). Later so many techniques were provided and got constant factor approximation algorithms. The best result for UFL problem was given by Guha and Kuller that the problem cannot be approximated with a factor of (1.436 - ϵ), for $\epsilon > 0$, unless NP $\subseteq DTIME(n^{O(log\, log\, n)})$. We will be using local search procedure to solve this problem.

Chapter 5

Mapping Railway Database Problem to UFL

In this chapter we are going to map the railway database problem to the Uncapacitated facility location problem(UFL). When we are not considering the capacity limits at each location,the presence of a record at a site won't effect the service or facility cost of other records at the same site. Here we are assuming that queries are only read and write and they involve only one row at a time. So, we will be considering each row separately and find out the best sites at which we can place the record. Combining all these solutions we will get the solution for the main problem.

5.1 The Mapping

Here we are going to map the given problem (finding the best locations for a given row in the table, given demands and costs associated with the locations) which is defined in chapter 2 to UFL which is described in the previous chapter. For immediate reference, we once more state the input to the UFL problem here, i.e:

Input to the UFL problem: The set of facilities F, the set of clients C, and the distance metric (c_{ij}). For each i ∈ F , the cost of opening a facility at i, denoted by f_i, is also given.

In mapping the railway database problem to UFL, first thing to decide upon is: what are the set of facilities and the set of clients? Since any site is capable of holding the given record, each site $(s_1, s_2, ..., s_m)$ becomes a facility and since every site is having some amount of demand(here we are assuming that every location has got demand for any given record, if that is not the case then we can put number of read requests and write requests for that site to be zero), every site $(s_1, s_2, ..., s_m)$ becomes a client. Therefore $\mathbf{F} = \{f_1, f_2, ..., f_m\}$ where each $f_i = s_i$ and $\mathbf{C} = \{c_1, c_2, ..., c_m\}$ where each $c_i = s_i$

The distance metric between the facilities and clients was formed using the communication cost between any two pair of sites(here we are assuming that communication costs satisfy metric properties). Therefore $c_{ij} = c(s_i, s_j)$ where $c(s_i, s_j)$ dentotes the cost of communication between sites s_i and s_j.

Now, we need to define the costs associated with the solution i.e service cost and facility cost. Note that these costs are depend upon the the solution, if we don't have soution in our hand then

we won't be able to find the values for these quantities. Suppose **S** is any solution to the problem.

Service Cost: Service cost of client i is dist(i,S) * R(i), where dist(i,S) denotes the minimum distance between i and a facility in S(solution) and R(i) dentotes the number of read requests at location i for the given record. Therefore total service cost of the solution S is:

$$\sum_{i \in C} dist(i, S) * R(i) \tag{5.1}$$

Facility Cost: We pushed the read requests response time into service cost. Similarly we push all the write request response time into facility cost. By opening a record at a site, we are forcing any site which is updating the record to communicate this updating information to this site. We turn this writing cost(cost of communication) into facility cost of the site.

$$f_i = \sum_{j \in C} 2 * c_{ij} * W(j) \tag{5.2}$$

where j iterates through all the clients in C and W(j) denotes the number of write requests at j. Facility cost of the solution is sum of all the facility costs of the facilities opened.

NOTE : While mapping database partitioning problem to the uncapacitated facility location problem, read request response time is pushed into service cost and write request response time into facility cost. If we alter these(i.e read response is taken as facility cost and write response is taken as service cost), it won't make any change in our solution because , what we are trying to optimize is the sum of facility cost and service cost. This alteration may not work for all the variants of the facility location problem. If we are to consider capacity limits(Facility location with soft capacities) then the above alteration produces wrong results.

Chapter 6

Solution through Local Search

Hochbaum was the first one to come up with an approximation algorithm for Facility location problem. Since then many different techniques to solve the problem were proposed. Here is the list of techniques which have been used to solve some variants of the facility location problem.

- Greedy Heuristics.
- LP Rounding techniques.
- Primal-Dual techniques.
- Local search techniques.

We have chosen local search technique to solve the problem in first instance due to its simplicity and even though the algorithm looks simple, its approximation facotr is near to the approximation factors of other good techniques.

6.1 Local Search Technique

Suppose P is an optimization problem which we want to solve and let I be the instance of the problem. Given instance I of the problem P, local search procedure starts from an arbitrary solution and iterates through the space of feasible solutions to I and outputs a local optimum solution. The local step the procedure takes at every iteration depends on the neighbordhood structure we define for the particular local search procedure. The generalized local search procedure is given in figure 6.1 from [1].

The elements we need to have, to describe local search procedure are:

1. **S**: Set of all feasible solutions to the instance I of the given problem P.
2. **Cost Function cost(S)**: Represents the cost associated with the given solution(sum of facility and service cost) cost: $\mathbf{S} \rightarrow \mathbb{R}$.
3. **Neigborhood Structure N(s)**: This function maps every solution to a subset of the feasible solution space.
   N: $\mathbf{S} \rightarrow 2^{\mathbf{S}}$

Algorithm Local Search

1. s $\leftarrow$ an arbitrary feasible solution in **S**.
2. While $\exists$ s$'$ $\in$ N(s) such that cost(s$'$) $<$ cost(s)
3. s $\leftarrow$ s$'$
4. return s

Figure 6.1: General local search algorithm

Algorithm Local Search

1. s $\leftarrow$ an arbitrary feasible solution in **S**.
2. While $\exists$ s$'$ $\in$ N(s) such that cost(s$'$) $<$ (1-ϵ/Q)cost(s)
3. s $\leftarrow$ s$'$
4. return s

Figure 6.2: Polynomial local search algorithm

4. Given any solution s $\in$ **S** we should be able to find out s$'$ $\in$**N(s)** (if exists) such that cost(s$'$) $<$ cost(s).

Cost function and the neighbordhood structure changes from problem to problem. A solution s$\in$**S** is called local optimum solution if it is the best solution in its neighbordhood i.e cost(s) $<$ cost(s$'$) for all s$'$ $\in$ N(s). The algorithm described in figure 6.1 returns a local optimum solution to the given problem. Now we define a property of the local search procedure called *locality gap*. Locality gap of procedure LS(P)(Local search procedure for problem P) is defined as supremum local(I)/global(I) where local(I) denotes the cost of the local optmum solution returned by procedure LS(P) and global(I) denotes the cost of the global optimum solution.

6.2 Polynomial Local Search Algorithm

The algorithm given in figure 6.1 is not guarantied to run in polynomial time. To make the algorithm run in polynomial time we make a small change in the algorithm. In the local step, instead of picking up any solution which has cost lesser than the current one in the neighbordhood, we pick up a solution only if it reduces the cost by a certain factor. Therefore the improved algorithm is given in figure 6.2 where $\epsilon > 0$ is constant and Q is a suitable integer which is polynomial in the size of input. One thing to note here is that, the solution returned by the algorithm given in 6.2 is not local optimum. We need to choose Q such that locality gap is not increases much.

6.3 Local Search Algorithm for UFL

Here we give the local search algorithm for the uncapacitated facility location problem. We already have a general algorithm 6.1, the only thing we need to specify is the neighbordhood structure. Neigborhood is defined by the local operations allowed at each step. The local operations allowed here are adding a facility, dropping a facility and swapping a pair of facilities. Therefore

$$N(s) = \{S + \{s'\} | s' \in F\} \cup \{S - \{s\} | s \in S\} \cup \{S - \{s\} + \{s'\} | s \in S, s' \in F\}.$$

. It is proved that the local search procedure for the metric uncapacitated facility location problem with the neighbordhood structure N given above has a locality gap of atmost 3 [1]. Using scaling technique from [8], we can get an algorithm with approximation factor of $1 + \sqrt{2} + \epsilon$.

Chapter 7

Implementation of Distributed Database

7.1 Database Setup

To simulate the real world scenario of the railway database, few terminals were taken which were connected throug LAN. On each terminal PostgresSql(9.0.3) database server was installed. To manage the global transactions the Jboss Application server(5.0.0) was used. This Application server was also installed on all the machines. PostgresSql driver which supports distributed transactions was placed in the library of the application server.

Schema of the simple relation table we chose to work on is AVAILABILITY(TRAIN_ID, AVAILABLE) where TRAIN_ID is VARCHAR type and AVAILABILE is INTEGER type. This table contains the information about the available unbooked seats for a particular train. This table was created in all the database servers at all terminals(sites). In our work we are considering only one tuple of the relation and finding out the best possible locations(sites) to place the tuple. Since determination of this is independet of other tuples of the table, we can do this for all the tuples independently and distribute the table accordingly.

7.2 Simulation

EJBs(Enterprise Java Beans) were used in the Jboss application server to do the distributed transactions. EJBs are the Java EE server side components that run inside the ejb container and encapsulates the business logic of an enterprise application. EJB container is a program that runs on the server and implements the EJB specifications. EJB container provides special type of the environment suitable for running the enterprise components. Enterprise beans are used to perform various types of tasks like interacting with the client, maintaining sessions for the clients, retrieving and holding data from the database and communicating with the server. There are two types of enterprise beans viz. Sessinon bean and Message driven bean. Session beans directly interact with the client and contains business logic of the business application. Message beans works like a listener or a customer for a particular messaging service such as JavaMessage API. Session beans were used in our simulation.

All the EJBs which were deployed at all the sites have got the same functionality. They contain functions which can be called by the clients remotely. The major functions these EJBs contain are getAvailability() which returns the number of available seats and setAvailability(int) which takes an input parameter and reduces the number of available seats by those many number. At each site there was a client program running which executes the operations based on the workload file given to it. The workload file contains sequence of read and write operations with delays in between. The client program at a particular site contacts the local ejb and that ejb responds by contacting the local database server if the record exists locally(for read operations) or it fetches the record from the nearby site to which this site was assigned by contacting the corresponding ejb on that machine.

To test the algorithm, we gave some random read and write request values at each site for the particular tuple we had chosen. Distances between the sites were taken randomly such that they obey the triangle inequality. These values were given as input to our algorithm(approximation algorithm: local search) and then the solution was obtained which indicates us where to place the record. To simulate distances, whenever an ejb at a site contacts the ejb at other site, the contacting ejb was delayed for some time proportional to the distance between them. Workload file was created at each site which reflects the read and write request frequencies at that site. This wass achieved by including wait requests for a particular period between each read and write operations such that per minute the number of read and write requests match the frequencies we gave as input to our algorithm. The clients at all the sites were run using this workload files and the time taken for read and write requests were calculated at each site and then the average time taken for the requests(read and write) were calculated.

7.3 Results

Initially we designed the problem by doubling the communication cost between the servers involved in write request as compared to read request. This was because for the write request we require two-phase commit protocol, where as for read request we don't require it. But in Jboss application server by default all the global transactions are following the two-phase commit protocol. Currently Jboss transaciton manager doesn't directly allow us to make a transactio read-only(which prevents it to follow 2 phase commit protocol). So for this purpose we changed communication cost in write request accordingly and generated the local optimal soltuions for the following configurations.

7.3.1 Configuration 1

Table 7.1: Distance metric

	1	**2**	**3**	**4**	**5**	**6**
1	0	5	10	16	20	30
2	5	0	14	10	17	29
3	10	14	0	15	13	15
4	16	10	15	0	10	16
5	20	17	13	10	0	11
6	30	29	15	16	11	0

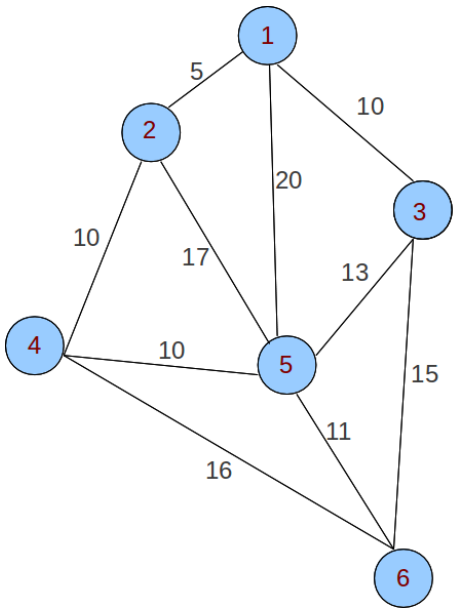

Figure 7.1: Sites configuration_1(fully connected, but all links are not shown)

The configuration of sites were as shown in the figure 7.1. Sites are labelled as 1,2,3,4,5 & 6. Distances between each of the sites are as given in the table 7.1. Number of read and write requests per minute at each site are as given in the table 7.2. Convention we follow to express where the record is placed is as follows. We express this using an array where each postion refers to a site and each postion in array contains 0 or 1, where 0 represents that the record is not placed at that particular site and 1 represents that the record is placed at that site. When we give this configuration to the local search algorithm , the solution array we got is (0 0 0 0 1 0) which means we need to open the record at only one site that is at location **5**.

The terms ϵ and 'Q' mentioned in the algorithm 6.2 were taken as 0.1 and 12 respectively.The algorithm started with a random solution that is (0 0 0 0 0 1) whose cost of solution is *1682* and the cost of the final local optimal solutioin (0 0 0 0 1 0) is *1002*. This solution we got after one local step of swapping. But this solution was turned out to be the optimal solution. Our algorithm is random as it chooses the initial solution randomly every time we run it. After running the algorithm for a number of times we could get other solution (0 0 0 1 0 0) as local optimal solution. The reason that we were interested in other solution rather than the optimal solution is to compare the local optimal solution and the optimal solution. Here the ratio (local optimal soltuion cost/optimal solution cost) turned out to be 1.02 which is less than 3 which is locality gap for the local search apporximation algorithm.

When we ran the clients at all the sites with the work load as given in table 7.2 with the solution(0 0 0 0 1 0), the average response time for read and write requests was 1002 milli seconds. And the average response time for the other solutions are given in table 7.3. When an ejb was contacted from the another server the current ejb was made to wait the distance between them multipled by 100 milli seconds before calling the method in ejb.

Time taken for read and write requests at each particular site for different solutions are given in tables 7.4, 7.5, 7.6 and 7.7.

Table 7.2: Read and write requests frequencies

Sites	**read requests per minute**	**write requests per minute**
1	3	1
2	15	10
3	7	4
4	12	8
5	20	12
6	9	5

Table 7.3: Results of the simulation

Solutions	**cost**	**avg. response time(in millisecs)**
0 0 0 1 0 0 (local optimal solution)	1023	1027
0 0 0 0 1 0 (best solution)	1002	1002
0 0 0 1 1 0	1141	1139
1 1 1 1 1 1	2992	2156

7.3.2 Configuration 2

In the previous configuration write cost dominated the read cost. Therefore the optimal solution consists of opening record only at one place. In the following configuration 2, number of read requests are far more than number of write requests. This yielded an optimal solution where we need to open record at more than one place.

When this configuration 7.2 with read and write frequencies as given in 7.9 was given to the local search algorithm the local optimal solution we got is (1 0 0 0 1 1) whose cost is 7984. This also turned out to be the optimal solution. How many number of times we ran our local search algorithm we end up with the same optimal solution. In this case also values for ϵ ,'Q' were taken as 0.1 and 12 respectively, which are terms in the polynomial local search algorithm given in Figure6.2.

As before sites are labelled as 1,2,3,4,5 & 6. Distances between each of the sites are as given in the table 7.8. Number of read and write requests per minute at each site are as given in the table 7.9. Time taken for read and write requests at each particular site for different solutions are given tables 7.11, 7.12 and 7.13. Average response times for each solution are given in table 7.10.

Table 7.4: Time taken at different sites for solution (0 0 0 1 0 0)

Server	no.of read requests	time taken for reads(milli secs)	no.of write requests	time taken for writes(milli secs)	total time(milli secs)
1	3	5114	1	1645	6759
2	15	15997	10	10665	26662
3	7	11407	4	6334	17741
4	12	460	8	464	924
5	20	20699	12	12547	33246
6	9	15206	5	8353	23559

Table 7.5: Time taken at different sites for solution (0 0 0 0 1 0)

Server	no.of read requests	time taken for reads(milli secs)	no.of write requests	time taken for writes(milli secs)	total time(milli secs)
1	3	6213	1	2046	8259
2	15	26339	10	17597	43936
3	7	9815	4	5430	15245
4	12	12681	8	8494	21175
5	20	590	12	631	1221
6	9	10579	5	5854	16433

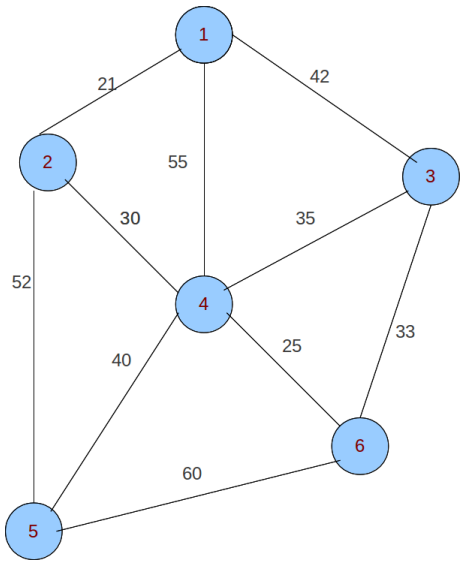

Figure 7.2: Sites configuration_2(fully connected, but all links are not shown)

Table 7.6: Time taken at different sites for solution (0 0 0 1 1 0)

Server	**no.of read requests**	**time taken for reads(milli secs)**	**no.of write requests**	**time taken for writes(milli secs)**	**total time(milli secs)**
1	3	5023	1	3675	8698
2	15	15831	10	27996	43827
3	7	9503	4	11626	21129
4	12	395	8	8725	9120
5	20	356	12	13047	13403
6	9	10552	5	14080	24632

Table 7.7: Time taken at different sites for solution (1 1 1 1 1 1)

Server	**no.of read requests**	**time taken for reads(milli secs)**	**no.of write requests**	**time taken for writes(milli secs)**	**total time(milli secs)**
1	3	5023	1	3675	8698
2	15	15831	10	27996	43827
3	7	9503	4	11626	21129
4	12	395	8	8725	9120
5	20	356	12	13047	13403
6	9	10552	5	14080	24632

Table 7.8: Distance metric

	1	**2**	**3**	**4**	**5**	**6**
1	0	21	42	55	57	72
2	21	0	66	30	52	99
3	42	66	0	35	90	33
4	55	30	35	0	40	25
5	57	52	90	40	0	60
6	72	99	33	25	60	0

Table 7.9: Read and write requests frequencies

Sites	**read requests per minute**	**write requests per minute**
1	75	2
2	35	5
3	40	7
4	20	3
5	60	12
6	10	10

Table 7.10: Results of the simulation

Solutions	**cost**	**avg. response time(in secs)**
1 0 0 0 1 1 (local optimal & optimal)	7984	2.3
1 0 0 0 0 1	9680	3.4
0 0 0 1 0 0	12710	3.8

Table 7.11: Time taken at different sites for solution (0 0 0 1 0 0)

Server	**no.of read requests**	**time taken for reads(milli secs)**	**no.of write requests**	**time taken for writes(milli secs)**	**total time(milli secs)**
1	75	414581	2	11088	425669
2	35	107039	5	15319	122358
3	40	134232	7	53194	187426
4	20	532	3	168	700
5	60	242722	12	48645	291367
6	10	25660	10	25653	51313

Table 7.12: Time taken at different sites for solution (1 0 0 0 0 1)

Server	**no.of read requests**	**time taken for reads(milli secs)**	**no.of write requests**	**time taken for writes(milli secs)**	**total time(milli secs)**
1	75	744	2	14535	15279
2	35	75049	5	60464	135513
3	40	133961	7	53208	187169
4	20	51066	3	24328	75394
5	60	344832	12	141633	486465
6	10	275	10	72794	73069

Table 7.13: Time taken at different sites for solution (1 0 0 0 1 1)

Server	**no.of read requests**	**time taken for reads(milli secs)**	**no.of write requests**	**time taken for writes(milli secs)**	**total time(milli secs)**
1	75	891	2	26096	26987
2	35	75340	5	86803	162143
3	40	134265	7	116531	250796
4	20	51110	3	36483	87593
5	60	1053	12	1311	2364
6	10	378	10	133562	133940

Chapter 8

Summary and Future Work

In summary, we have identified a problem in Indian Railway Database and come up with a solution based on local search algorithm for uncapacitated facility location problem. By using the solution of this algorithm and distributing the database accordingly, we can considerably improve the response time for the requests that users submit to the database. We showed this by simulating the real world scenario using jboss applicaiton server. We assumed that we are properly given all the input data like number of read requests and write requests at a particular site for a particualr row in the table.

Future work includes

1. Finding a solution for the extended database problem which is defined in chapter 2.
2. Implementation of other techniques and finding the best one in terms of time taken and quality of the solution.
3. Comparing this approach with the other data fragmentation techniques in general.

References

[1] V.Pandit *Local search heuristic for facility location problems*, Ph.D. thesis, IIT Delhi 2004.

[2] S. I. Khan and D. A. L. Hoque. A new technique for database frgamentation in distributed Systems *International Journal of Computer Applications* 9,5(2010).

[3] Madhukar R. Korupulu and C. Greg Plaxton and Rajmohan Rajaraman. Analysis of local searh heuristic for facility location problems *Proceedings of the ninth annual ACM-SIAM symposium on Discrete Algorithms*,1998.

[4] Madhukar R. Korupulu and C. Greg Plaxton and Rajmohan Rajaraman. Analysis of local searh heuristic for facility location problems Technical Report DIMACS, (1998) 9830.

[5] M. T. Ozsu and P. Valduriez. Principles of Distributed Database Systems. Prentice-Hall, 1999.

[6] Wikipedia, *http://en.wikipedia.org/wiki/Two-phase_commit_protocol*

[7] Abraham Silberschatz, Henry F.Korth and S. Sudarshan. Database System Concepts, McGraw-Hill, 5, 2006.

[8] M. Charikar and S. Guha, Improved combinatorial algorithms for the facility location and k-median problems. *Proceeding of the 40th Annual Symposium on Foundations of Computer Science*, 77, (1999) 378-388.

[9] D.S. Hochbaum. Heuristics for the fixed cost median problem . *Mathematica Programming* , 22, 1982(148-162).

[10] Eva Tardos Karen Aardal and David B. Shmoys. Approximation algorithms for facility location problems *STOC '97 Proceedings of the twenty-ninth annual ACM symposium on Theory of computing* , (1997).

[11] Sanjay Agrawal, Surajit Chaudhuri, Lubor Kollar, Arun Marathe, Vivek Narasayya and Manoj Syamala. Database tuning advisor for Microsoft SQL Server 2005. *Proceeding of the* 30^th *VLDB Conference*, Toronto, Canada, 2004.

[12] Anirban Mondal, Sanjay Kumar Madria and Masaru Kitsuregawa. CLEAR: An efficient context and location-based dynamic replication scheme for Mobile-P2P Networks. *Proceedings of the International Conference on Database and Expert Systems Applications* , 2006.

[13] Optimal file allocation in a multiple computer system. *IEEE Transactions on Computers*, Vol:C-18 No:10, October 1969.

[14] David L. Black and Daniel D. Sleator. Competitive algorithms for replication and migration Problems. *Technical Report Carnegie Mellon University* , CMU-CS-89-201, 1989.

[15] Jun Rao, Chun Zhang, Guy Lohman and Nimrod Megiddo. Automating physical database design in a parallel database. *Proceeding of the ACM SIGMOD* , June 2002.

[16] Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C. Hsieh, Deborah A. Wallach Mike Burrows, Tushar Chandra, Andrew Fikes and Robert E. Gruber. Bigtable: A distributed storage system for structured Data. *OSDI'06: Seventh Symposium on Operating System Design and Implementationx* , November, 2006.

[17] S. Ceri, M. Negri, G. Pelagatti. Horizontal data partitioning in database design. *Proceedings of the ACM SIGMOD international conference on Management of data*, 1982.

[18] W. W. Chu. Optimal file allocation in a multiple computer system. *IEEE-TC, vol. C-18, no. 10*, 1969.

[19] K. P. Eswaran. Placement of records in a file and file allocation in a computer network. *Proc. IFIP Congress*, North Holland, 1974.

[20] S. Mahmoud, J. S. Riordon. Optimal allocation of resources in distributed information networks. *ACM- TODS*, Vol. 1, no. 1, 1976.

[21] H. L. Morgan, J. D. Levin. Optimal program and data location in computer networks. *CACM*, Vol. 20, no. 5, 1977.

[22] P. P. S. Chen, J. Akoka. Optimal design of distributed information system. *IEEE-TSE*, Vol. SE-6, no. 12, Dec. 1980.

[23] S. Ceri, G. Martella, G. Pelagatti. Optimal file allocation on a network of minicomputers. *Proc. Int. Conf. on Database*, Heyden Pub., Aberdeen, July 1980.

[24] S. Ceri, S. B. Navathe, G. Wiederhold. Optimal design of distributed databases. *Working paper, Stanford University*, 1981.

[25] R. G. Casey. Allocation of copies of a file in an information network. *AFIPS, SJCC 1972, pp. 617-625.*

[26] K. D. Levin. Organizing distributed data bases in computer networks. *Univ. Pennsylvania, Philadelphia*, Ph.D. dissertation, 1974.

[27] K. B. Irani and N. G. Khabbaz, A methodology for the design of communication networks and the distribution of data in distributed supercomputer systems. *IEEE Trans. Comput.*, vol. C-31, pp 419-434, May 1982.

[28] M. L. Fisher, and D. S. Hochbaum. Database locations in computer networks. *J. Ass. Comput. Mach.*, Vol. 27, pp. 718-735, Oct. 1980.

[29] M. E. S. Loomis, Data base design: object distribution and resource constrained task scheduling. *Ph.D. dissertation, Dep. Comput. Sci., Univ. California*, Los Angeles, 1975.

[30] B. Srinivasan and R. Sankar. Algorithms to distribute a database for parallel searching. *IEEE Trans. Software Eng.*, Vol. SE-7, p. I12, Jan. 1981.

[31] T. Kijuno et al. On a file initial-placement problem in distributed database systems. *Hiroshima Univ.*, Japan, CSG Tech. Res. 81-08, Apr. 1981.

[32] A. J. Smith, Optimization of I/O systems by cache disks and file migration. *New York: North Holland*,1981, vol. 1, pp. 249-262.

[33] E.Grapa and G. G. Belford, Some theorems to aid in solving the file allocation problem. *Commun. ACM,* 20(11):878882, 1977.

[34] L. V. Sickle and K. M. Chandy, Computational complexity of network design algorithms, *Information Processing 77, IFPS*, 1977.

[35] A. M. Geoffrion and R. E. Marsten, Integer programming: a frame-work and state-of-the-art survey, *Manage. Sci.,* vol. 18, pp. 465-491, May 1972.

[36] S. P. Ghosh, Distributing a data base with logical associations on a computer network for parallel searching, *IEEE Trans. Software Eng.,* vol. SE-2, pp. 106-113, June 1976.

[37] E. G. Coffman, Jr. et al., Optimization of the number of copies in a distributed data base, *IEEE Trans. Software Eng.*, vol. SE-7, pp. 78-84, Jan. 1981.

[38] Ceri, S., Martella, G., and Pelagatti, G. Optimal file allocation in a computer network: A solution method based on the knapsack problem. *Comp. Netw.*, 6:345357. 1982.

[39] Chang, S. K. and Liu, A. C. , File allocation in a distributed database. *Int. J. Comput. Inf. Sci,* 11(5):325340. 1982.

[40] J. van Louveen. Algorithms modeling and complexity. In lecture notes, university of Utrecht.

[41] Ramamoorthy, C. V. and Wah, B. W., The isomorphism of simple file allocation. *IEEE Trans. Comput.,* C-23(3):221231, 1983.