QUERY PROCESSING IN DISTRIBUTED DATABASE SYSTEMS

A Dissertation

Submitted To

The Central Department of Computer Science and Information Technology, Faculty of Science and Technology

Tribhuvan University

In Partial Fulfillment of the Requirements for the degree of

Master of Science in Computer Science and Information Technology

***By*:**

Bardan S.J.B. Rana

April 2009

Tribhuvan University

Institute of Science and Technology

Central Department of Computer Science and Information Technology

Date:___________

Recommendation

I hereby recommend that the dissertation prepared under my supervision by **Mr. Bardan S.J.B. Rana** entitled **“Query Processing in Distributed Database Systems”** be accepted as a partial fulfillment of the requirement for the degree of Master of Computer Science, from Tribhuvan University, Nepal. In my best knowledge this is an original work in computer science.

Prof. Dr. Shashidhar Ram Joshi
Department of Electronics and Computer Engineering,
Institute of Engineering, Pulchowk, Nepal
(Supervisor)

Tribhuvan University

Institute of Science and Technology

Central Department of Computer Science and Information Technology

We certify that we have read this dissertation work and in our opinion it is satisfactory in the scope and quality as a dissertation as the partial fulfillment of the requirement of Master in Computer Science and Information Technology from Tribhuvan University, Nepal.

Evaluation Committee

______________________	______________________
Dr. Tanka Nath Dhamala Head of the Department, Central Department of Computer Science and Information Technology Tribhuvan University, Nepal	**Prof. Dr. Shashidhar Ram Joshi** Department of Electronics and Computer Engineering, Institute of Engineering, Pulchowk, Nepal (Supervisor)
______________________	______________________
(External Examiner)	**(Internal Examiner)**

Date: ______________

ABSTRACT

Query processing on a distributed database system requires transmission of data between computers on a communication network. Minimizing the amount of data transmission is one of the fundamental principles to reduce the query processing cost and to prevent network congestion.

The semijoin operation is important in formulating query processing strategies. Semijoin preprocessing strategy provides enough opportunities to significantly reduce the amount of data required to be transmitted on the network by first reducing the cardinalities of a distributed relations using semijoins and then transmitting the resultant relations to the result node.

Algorithm LIGHT which is based on new heuristic for generating semijoin preprocessing strategies for queries has been developed in this work. Heuristic used in algorithm LIGHT can be thought as modification of heuristic used in algorithm SDD-1.

Simulation model has been constructed to evaluate the performance of existing query preprocessing algorithm SDD-1 and algorithm LIGHT. This model test random queries and results are presented and discussed. It is shown that algorithm LIGHT performs better than algorithm SDD-1 in general.

ACKNOWLEDGEMENT

First of all, I am extremely grateful to my respected teacher and dissertation supervisor Professor Dr. Shashidhar Ram Joshi, Department of Electronics and Computer Engineering, Institute of Engineering, Pulchowk, for his invaluable guidance throughout this work. In spite of his hectic schedule, he has been readily available for advice, reading, or simply a word of encouragement. I learned a lot from him about good research practice and what it takes to achieve this goal.

I would like to acknowledge my deep appreciation to respected teacher Dr. Tanka Nath Dhamala, Head of Computer Science and Information Technology, for paving the first stone in this study and providing pleasant working atmosphere. Likewise, my sincere appreciation goes also to my respected teacher and co-supervisor Mr. Samujjwal Bhandari for his support, critical feedback, suggestions and fruitful discussion which was instrumental.

At present, I have no words to fully express my gratitude to the respected teachers, Dr. Onkar P. Sharma (Marist College, USA), Dr. Subarna Shakya, Mr. Sudarshan Karanjit, Mr. Min Bahadur Khati and all other for the knowledge they poured in me and inspirations they gave to me in the time period of two years.

I would like to thank my friends, in alphabetical order, Mr. Amar Man Maharjan, Mr. Bishnu Subedi, Mr. Jagdish Bhatta, Mr. Shishir Paudyal and all other for their co-operation, active participation in sharing ideas and keeping faith in me.

Finally, I would not have reached this point in my academic life without the support and unconditional love of my parents and brothers. I am forever indebt to my parents and brothers for their love, understanding, patience and encouragement when it was required most. Once again I would like to thank each and every individual who are involved in this work because I believe any accomplishment requires effort of many individuals and this work is not different.

Bardan Rana

04/02/2009

ABBREVIATIONS

AP	Application Processor
CC	Current Cardinality
CPU	Central Processing Unit
CSJ	Candidate Semijoin
DDB	Distributed Database
DDBMS	Distributed Database Management System
DDBS	Distributed Database System
DM	Data Manager
DP	Data Processor
I/O	Input/Output
IFS	Initial Feasible Solution
PERF	Partially Encoded Record Filter
SDD	System for Distributed Database
SJP	Semijoin Program
SPJ	Select Project Join
SQL	Structured Query Language
TM	Transaction Manager
TP	Transaction Processor
WAN	Wide Area Network
WWW	World Wide Web
XML	Extended Markup Language

FIGURES

TABLES

TABLE OF CONTENTS

CHAPTER 1
INTRODUCTION

1.1 Introduction

The amount of information available to us is exploding, and the value of data as organization assets is universally recognized. To get the most out of their large and complex datasets, users require tools that simplify the task of managing the data and extracting useful information in a timely fashion. Otherwise, data can become a liability, with the cost of acquiring it and managing it far exceeding the value derived from it.

A database is the collection of related data that corresponds to some aspect of real world. In other words, a database is a collection of relevant persistent data to which concerned users interact to satisfy their need. A database can be of any size and varying complexity. In present world, database and database systems have become the life blood for every organization to remain competitive and more importantly to survive in this competition. In addition, recent progress in communication and database technologies have brought it to an edge; moving from centralization to its counter part de-centralization concept.

Typically in a 'computerized information system' we have a database and methods to access data from the database in response to queries. These database systems usually exist at an enterprise or in a government agency or department. There is also a need for geographically distributed components of these organizations to access data from central databases.

One method to satisfy this need is to establish a communication link between the central computer and the site where data are required. This method has its drawbacks for several reasons. If telecommunication lines are utilized, either a dedicated communication line or the public communication lines may be utilized. If a dedicated communication link exists, the cost of this link is often too expensive. If the public communication lines are used, one must contend with problems. Lines are required to establish a link with the central computer which may be unavailable if lines are busy. In any case there is always the dial up time required to establish the link with the central computer. The central computer may also be serving the maximum number of users and service may not be available. Another drawback is system reliability, if the central computer malfunctions, data access is denied. As the information needs of the organization grow it becomes more difficult and more expensive to increase the speed and capacity of the central computer. It is not too hard to imagine that a single central computer operating with the latest technology may not be able to operate efficiently on a large database.

Another method of satisfying this need for distributed data is to provide each geographically dispersed site with a complete copy of the central database. Major problem with this approach is the simultaneous updating of data at each site so that a consistent version of the database is available to all users which take away all the advantages of this approach. This approach would also involve expensive hardware costs.

A popular intermediate approach, which is gaining much attention in the literature [2, 7, 21, 22, 33, 36] is to have data distributed such that the data most necessary to and most often accessed by, a site is presented at that site. It is called data localization. In this manner the large central database is divided into smaller databases and most data would be distributed non-redundantly among the sites, although some data may be replicated. The site computer and database system need not be as large as the single centralized version. System reliability is enhanced since the failure of a site computer generally only affects that site. Hardware and software updates are much simpler and less costly as local demands increase. Communication between sites is only required when the data required by a query is not resident at the site of origin. This attractive intermediate approach solves information processing needs of geographically dispersed organizations. Such a system is called distributed database system (DDBS) and databases used by such system a distributed database. In short, distributed database is a collection of multiple, logically interrelated databases distributed over a computer network.

Distributed database system (DDBS) technology is the union of what appear to be two diametrically opposed approaches to data processing: database system and computer network technologies. Distributed database leads to the distributed data processing or data computing in which a number of autonomous processing elements (not necessarily homogeneous) that are interconnected by a computer network; co-operates in performing their assigned task.

There are many unsolved problems with distributed database systems. Research is currently being done in the areas of distributed concurrency control, system reliability, failure and recovery, distributed database design, distributed system control, query processing and security. However, this work is confined only with the query processing aspect of distributed database systems.

1.2 Motivation

The need for communication between sites distinguishes a distributed database system from a centralized database system. A low level view of this communication is digitized data being transmitted from site to site. The purpose of this communication is to access data at other sites in the network and for transmission of data in response to queries.

In the past, queries in database systems were typically expressed in terms of application programs written in high level procedural languages for batch processing of queries. Interactive query processing was especially popular in distributed systems. However, in recent years the trend has move to high level non procedural query language like SQL (Structured Query Language) which is also utilized to this end. An example of SQL query on a supplier database is shown below.

```
SELECT    S.PART_NUM
FROM      SUPPLIER AS S AND ORDER AS O
WHERE     S.PART_NUM=O.PART_NUM AND O.PRICE>1000
```

This query requests all part numbers in the SUPPLIER relation such that the part numbers are on order (in the ORDER relation) and these part numbers have their parts price greater than 1000 (taken from [27]). High level query languages permit complex data extraction without the necessity of procedurally specifying how it is to be extracted. The relation data model is an ideal data model for this approach. Codd [Codd 70] defined the relational data model and showed how data may be accessed in terms of relational algebra and relational calculus, both of which form the basis for high level query languages. A high level query may be interpreted by the local distributed database management system as commands (operations) that will extract the necessary distributed data to satisfy queries. Relational algebra is a procedural language in which user has to specify the each step to obtain the result of the query using the certain high level operator. On the other hand relational calculus is nonprocedural language in which the user only specifies the relationships that should hold in the result (like SQL).

An aspect of query processing on distributed database systems that has received much attention is what operations (relational ones) will be utilized in processing queries. The semi-join operations [2, 4, 6, 7, 19, 21, 36] (half a join in relational algebra) has been found to be a good strategy for query processing on distributed systems. Semi-join may be utilized to reduce relation size of distributed relations. Since semi-join may be defined in mathematical terms which form the basis for relational set theory and since its physical operations on data are clearly defined, semi-join is a very useful tactic for processing queries on distributed database systems.

To answer queries, one strategy is to move all relevant relations to the node where the result is required. This is a costly data communication process. Semi-join is used to reduce the size of distributed relations before they are transmitted on the network to the node where the result of the query is desired. Considerable research has been done by [2, 4, 6, 7, 18, 19, 21, 33, 36] concerning semi-join tactics and their use in distributed database systems. Three algorithms [7, 21, 33] have been published that produce semi-join strategies for processing queries. In order to formulate a query processing strategy one must assume a particular model of the distributed database system. This model should correctly estimate the effect of semi-join on the database state. One motivation of this work is to review the existing models of distributed database systems. How to overcome the physical limitation of the wire (i.e. the amount of data that can be transferred per unit time) logically, so that network utilization is as low as possible is the second motivation of this work.

1.3 Objective of Study

The objective of this work is to review an existing semijoin based query processing algorithms in distributed database systems. The specific objective of this work is to formulate a heuristic and use this heuristic to develop an algorithm. Comparing the total communication cost required to answer the imposed queries by an existing algorithm SDD-1, newly developed algorithm LIGHT and initial feasible solution (i.e., simple join) is the another objective of this study. In short, objective of this study can be summarized as:

1. Show how query processing in distributed database system is different from centralized one.
2. Review an existing query processing algorithms based on semijoin strategy.
3. Show how the semi-join can be used to minimize the network resource consumption.
4. Formulate a heuristic and use it to develop an algorithm.
5. Compare the results obtained by various algorithms.

1.4 Significance and Limitations

Physical limitation imposed by the wire on the amount of data transferred within a certain distance and certain time period naturally gives us the idea that amount of data transfer directly affects the performance of query processing in distributed database. Moreover, the amount of data that can be transferred through the wire per unit time is directly proportional to the distance (characteristic of any signal carrying media like wire). For example, the minimum round trip message propagation delay in satellite based system is about 1 second. Semi-join tactics which is more likely to reduce this amount of data transfer is therefore a main focus of this study. Reducing the amount of data transfer is even more pronounced in wireless distributed database systems.

Amount of data transfer is considered as dominant factor in this study. Even though other factors like I/O (Input/Output) cost, CPU (Central Processing Unit) time cost also affects the performance of the query processing; they are not taken into consideration. Only retrieval based queries are considered and no update queries is being entertained as the retrieval query is the most often queries that are being submitted and update queries whose frequency are less often can be done is off hours. A WAN (Wide Area Network) is considered as the structure of the distributed database and it is assumed that sites are physically connected by wire. This work is only concerned with the query processing in distributed database and no aim is taken on design and concurrency control aspect of distributed database.

1.5 Report Structure

Coming eight chapters consist the remaining part of thesis. The issues discussed above are investigated in later chapters. In Chapter 2, background and problem definition of this work is presented. Reasons for the evolution of distributed database systems, promises it brings with it, fundamental principle of distributed database system and like are presented in it.

The review of the literature relevant in the context of a distributed query processing is the topic of discussion in Chapter 3. In this chapter, classical approach of semi-join to the most recent trend of negotiation based query processing strategy of distributed database has been discussed.

The relational data model is described in Chapter 4 in the context of a distributed database system. The utilized relational operations are introduced and the applicability of the relational data model to this area is discussed.

Chapter 5 is about query processing on distributed database systems. The basic assumptions of our work in this area are given. Our views on communication network, data transmission and incurred cost are presented. Use of semi-join as a query processing strategy is outlined. The complexity of determining optimal query processing strategies is described. This chapter is concluded with the concept of database state updates (i.e. how the database statistics are affected by an inclusion of semi-join operation in a semijoin program).

In Chapter 6, an estimation function for calculating the hypothetical update of database state when a possible candidate semijoin (CSJ) is added to the semijoin program (SJP) is presented. In this chapter, how the joining and non joining attributes of the reduced relation is affected when it is chosen to include in the semijoin program is illustrated.

In Chapter 7, discussion of using semijoin as a preprocessing strategy for queries has been extended. Two published algorithm [2, 7] which produce semijoin preprocessing strategies have been introduced. An algorithm LIGHT, a query preprocessing algorithm that represents a result from our work has also been introduced. Finally, an example is illustrated to demonstrate the total cost incurred by each of these algorithms.

Chapter 8 is about the implementation of algorithm SDD-1 and algorithm LIGHT and the analysis of the result obtained via simulation.

Chapter 9 concludes our work.

CHAPTER 2
BACKGROUND AND PROBLEM DEFINITION

2.1 Background

Researchers and practitioners have been interested in distributed database systems since the 1970s. At that time, the main focus was on supporting distributed database management of large co-operations or organizations that keep their data at different offices or subsidiaries. Although there was a clear need and many good ideas and prototypes (e.g. System R* [William et al. 1981], SDD-1 [Bernstein et al. 1981], and Distributed Ingres [Stone Baker 1985]) the early effort in building distributed database system were never commercially successful [24]. One of the main reasons is that the early distributed database systems were ahead of their time. First, communication technology was not stable enough to ship megabytes of data as required for these systems. Second, large business somehow manage to survive without sophisticated distributed database technology by sending tapes, diskettes, or just paper to exchange data between their offices.

Today, the situation has changed dramatically. Distributed data processing is both feasible and needed. Distributed data processing is feasible because of recent technologies advance (e.g. hardware, software protocols, and standards). Distributed data processing is needed because of changing business requirements which have made distributed data processing cost-effective and in certain situation only viable option.

2.2 Reasons for Distribution

The classical answers to this questions indicates that distributed processing better corresponds to the organizational structure of today's widely distributed enterprises and that such a system is more reliable and more responsive. More importantly, many of the current applications of computer technology are inherently distributed. Electronic commerce over the Internet, multimedia applications such as news-on-demand etc. are all examples of such applications.

From a more global perspective, however, it can be stated that the fundamental reason behind distributed processing is to better able to solve the big and complicated problems that we face today, by using a variation of the well-known divide and conquer rule. If the necessary software support for distributed processing can be developed, it might be possible to solve these complicated problems simply by dividing them into smaller pieces and assigning them to different software groups, which works on different computers and produce a system that runs on multiple processing elements but can work efficiently toward the execution of a common task.

This approach has two fundamental advantages from an economics standpoint. First distributed computing provides an economical method of harnessing more computing power by employing multiple processing elements optimally. The second economic reason is that by attacking these problems in smaller groups working more or less autonomously, it might be possible to discipline the cost of software development.

2.3 DDBMS Components

The DDBMS must include (at least) the following components:

1. Computer workstations (sites or nodes) that form the network system. The distributed database system must be independent of the computer system hardware.
2. Network hardware and software components that reside in each workstation. The network components allow all sites to interact and exchange data. Network system independence is a desirable distributed database system attribute.
3. Communications media that carry the data from one workstation to another. The DDBMS must be communications media-independent; that is, it must be able to support several types of communication media.
4. The transaction processor (TP) which is the software component found in each computer that requests data. The transaction processor receives and processes the applications data requests (remote and local). The TP is also known as the application processor (AP) or the transaction manager (TM).

5. The data processor (DP), which is the software component residing on each computer that stores and retrieves data located at the site. The DP is also known as the data manager (DM). A data processor may even be a centralized DBMS.

2.4 DDBSs Promises

Many advantages of DDBSs have been cited in literature, ranging from sociological reasons for decentralization to better economics. All of these can be distilled to four fundamentals which may also be viewed as promises of DDBS technology.

1. Transparent management of distributed and replicated data.
2. Reliability through distributed transaction.
3. Improved Performance.
4. Easier System Expansion.

Consider an engineering firm that has offices in Boston, Edmonton, Paris and San Francisco. They run projects at each of these sites and would like to maintain a database of their employees, the projects and other related data. Assuming that the database is relational, we can store this information in two relations: EMP(<u>ENO</u>, ENAME, TITLE) and PROJ(<u>PNO</u>, PNAME, BUDGET). Let PAY(<u>TITLE</u>, SAL), ASG(<u>ENO, PNO</u>, DUR, RES) respectively store salary information and which employees have been assigned to which projects for what duration with what responsibility (taken from [27]). If all of these data were stored in a centralized DBMS and we want to find out the names and salaries of employees who worked on a project for more than 12 months, we would specify this using the following SQL query:

```
SELECT    ENAME, SAL
FROM      EMP, ASG, PAY
WHERE     ASG.DUR>12
AND       EMP.ENO=ASG.ENO
AND       PAY.TITLE=EMP.TITLE
```

However, given the distributed nature of this firm's business, it is preferable, under these circumstances to localize each data such that data about the employees in Edmonton office are stored in Edmonton, those in Boston office are stored in Boston and so forth. The same applies for the project and salary information. This process of partitioning each of the relations and stored each partitioning at a different site is known as fragmentation.

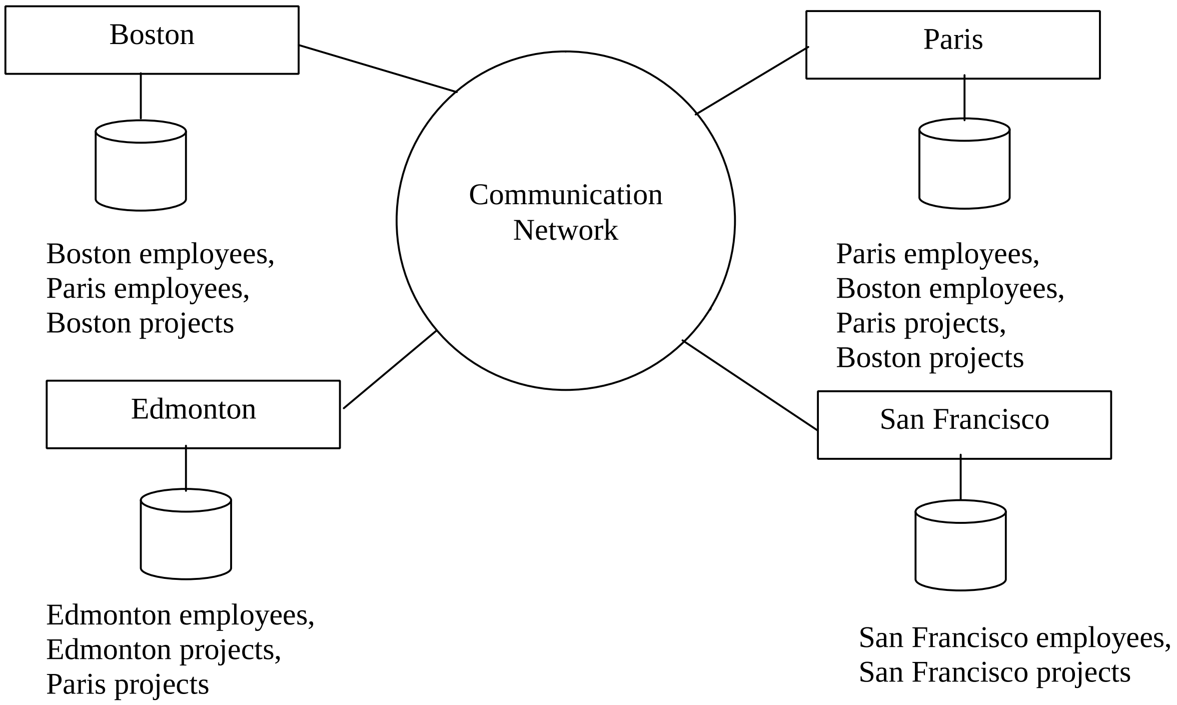

Boston
Paris
Communication Network
Boston employees, Paris employees, Boston projects
Paris employees, Boston employees, Paris projects, Boston projects
Edmonton
San Francisco
Edmonton employees, Edmonton projects, Paris projects
San Francisco employees, San Francisco projects

Figure 2.1 A Distributed Application

Furthermore, it may be preferable to duplicate some of this data at other sites for performance and reliability reasons. The result is a distributed database which is fragmented and replicated. Fully transparent access means that the user can still pose the query as before, without paying any attention to the fragmentation, location, or replication of data, and let the system worry about resolving the issues.

2.5 DDBS Architecture

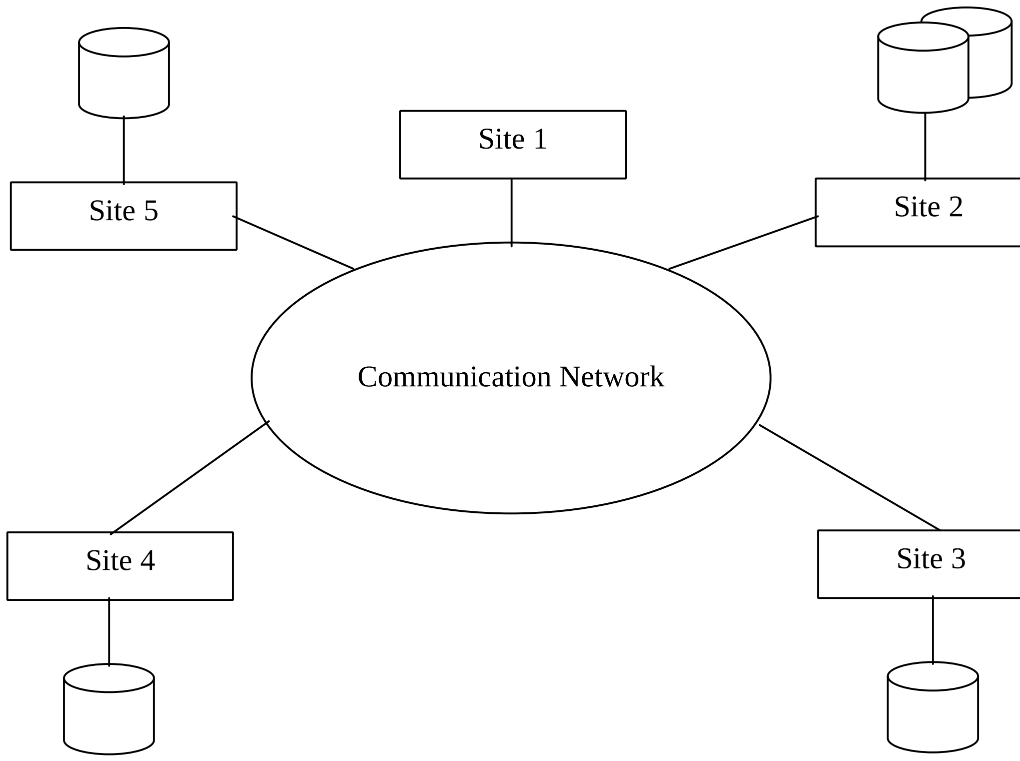


Figure 2.2 Distributed Database System Environment

2.6 DDBS Fundamental Principle

C.J. Dates [13] distributed database commandments describe a full distributed database, and, although no current DDBMS conforms to all of them, the rules do constitute a useful distributed database target. The commandments are:

RULE 1: Local Site Independence
Each local site can acts as an independent, autonomous, centralized DBMS. Each site is responsible for security, concurrency control, backup and recovery.

RULE 2: Central Site Independence
NO site in the network relies on a central site or any other site. All sites have the same capabilities.

RULE 3: Failure Independence

The system is not affected by node failures. The system is in continuous operation even in the case of a node failure or an expansion of the network.

RULE 4: Location Transparency

The user does not need to know the location of data in order to retrieve the data. The user is unaware of the fact that the data is distributed at all. What he sees is a single logical database like central database system and operates similarly.

RULE 5: Fragmentation Transparency

The user sees only one single logical database. Data fragmentation is transparent to the user. The user does not need to know the name of the database fragments in order to retrieve them.

RULE 6: Replication Transparency

The user sees only one single logical database. The DDBMS transparently selects the database fragments to access. The DDBMS manages all fragments transparently to the user.

RULE 7: Distributed Query Processing

A distributed query may be executed at several different data processor (DP) sites. Query optimization is performed transparently by DDBMS.

RULE 8: Distributed Transaction Processing

A transaction may update data at several different sites. The transaction is transparently executed at several different DP sites.

RULE 9: Hardware Independence

The system must run on any hardware platform.

RULE 10: Operating System Independence

The system must run on any operating system software platform.

RULE 11: Network Independence

The system must run on any network platform.

RULE 12: Database Independence

The system must support any vendor’s database product.

Examples:

Case 1: Database that supports fragmentation transparency

```
SELECT    *
FROM      EMP
WHERE     TITLE=’Manager’
```

Case 2: Database that supports location transparency

```
SELECT    *
FROM      BOSTON_EMP
WHERE     TITLE=’Manager’
UNION
SELECT    *
FROM      EDMONTON_EMP
WHERE     TITLE=’Manager’
UNION
SELECT    *
FROM      SAN_FRANCISCO_EMP
WHERE     TITLE=’Manager’
```

Case 3: Database that supports mapping transparency

```
SELECT    *
FROM      BOSTON_EMP NODE BOSTON
WHERE     TITLE=’Manager’
UNION
SELECT    *
FROM      EDMONTON_EMP NODE EDMONTON
WHERE     TITLE=’Manager’
UNION
```

```
SELECT    *
FROM      SAN_FRANCISCO_EMP NODE SAN FRANCISCO
WHERE     TITLE=’Manager’
```

LEVEL OF DISTRIBUTION TRANSPERANCY

HIGH $\longleftrightarrow$ LOW

Specify:	Fragmentation	Location	Local Mapping
Fragment?	No	Yes	Yes
Location?	No	No	Yes

Table 2.1 Summary of Transparency Features

2.7 Problem Definition

Distributed data processing is becoming a reality. Business wants to do it for many reasons and they often must do it in order to stay competitive. While much of the infrastructure for distributed data processing is already there (e.g. modern network technology), a number of issue make distributed data processing still a complex undertaking; (1) distributed system can become very large, involving thousands of heterogeneous sites including PCS and mainframe server machines; (2) the state of distributed system changes rapidly because the load of sites varies over time and new sites are added to the system; (3) legacy systems need to be integrated, such legacy systems usually have not been designed for distributed data processing and now need to interact with other (modern) system in a distributed environment.

With respect to time when the optimization is performed, query optimization algorithm can be static (compile time) or dynamic (execution time). According to the type of information that is use to optimize the query, query optimization techniques can be classified as: Statistically based query optimization algorithm, which uses statistical information about the database and a Rule-based query optimization algorithm, which is based on a set of user-defined rules to determine the best query access strategy. Chapter 5 describes it more precisely.

The time to retrieve the result of the generated query in a distributed database system is critical. So, to minimize the retrieval time is a key issue in the query processing of distributed database system. Some researchers have only taken communication cost in consideration while the other researchers have taken local processing cost in addition to communication cost and still the others have even considered the possibility of parallel processing. Nevertheless, minimizing communication cost will always be the target.

CHAPTER 3
LITERATURE REVIEW

3.1 Literature Review

Distributed database system (DDBS) has become the need of almost every organization to stay in the business. This shift in paradigm from centralized to distributed database systems have forced the researchers think harder than ever before as the opportunities and problems it bring with it is enormous. Numerous literatures have cited the various aspect of distributed database system. Distributed database design [3], query processing and transaction management [5] are few of it. Because of critical performance issue query processing has been the centre of attraction of many literatures. Objectives of distributed query processing are pointed out in [8]. A good, recent survey of distributed query processing is presented by Donald Kossmann in 'The State of the Art in Distributed Query Processing ', [24].

The cost of processing a query in a distributed database system consist sum of CPU, I/O and communication costs (i.e., cost of query processing in distributed database system = CPU cost + I/O cost + Communication cost). The CPU cost is incurred while performing operation on data in main memory. The I/O cost is the time necessary for disk input/output operations. This cost can be minimized by reducing the number of I/O operations through fast access methods to the data and efficient use of main memory (buffer management). The communication cost is the time needed for exchanging data between sites participating in the execution of the query. The cost is incurred in processing the message (formatting/ de-formatting) and in transmitting the data on the communication network.

The first two cost components (I/O and CPU cost) are the only factors considered by centralized DBMSs. The communication cost is probably the most important factor considered in distributed database. Most of the earlier proposals [3, 5, 9, 33] for distributed query optimization assume that the communication cost largely dominates local processing cost (I/O and CPU cost) and thus ignore the latter. The assumption of taking the communication cost as the most significant factor is valid for wide area network (WAN), where the limited bandwidth makes communication much more costly than local processing. Therefore, the aim of distributed query optimization reduces to the problem of minimizing communication cost generally at the expense of local processing [3, 5, 9]. However, modern distributed processing environments have much faster communication networks, whose bandwidth is comparable to that of disks. Therefore, a more recent research [23, 25, 28] consider a weighted combination of these three cost components since they all contribute significantly to the total cost of evaluating a query. More recent studies investigate the feasibility of retrieving data from a neighbouring nodes main memory cache rather than accessing them from a local disk [28].

Even if the relational distributed database system is assumed, there are numerous algorithms on the subject but they are not designed for the same environment. Some algorithms are designed for local network [19, 22], some for star network and most of the others algorithm for long haul network. Also some environments have no fragmented relations whereas in others some relations may be fragmented. Till date no one is able to show the superiority of one algorithm over all other algorithms for a given environment.

An important performance issue in distributed database systems is the implementation of logical relationship of data elements stored across sites. An example of this is the high cost of performing the join of relations stored at different sites. The straight forward approach to implement the join is to send one of the join participating relations to the site of the other relation and perform the join at that site. The objective of join query optimization is to reduce the cost of this inter-site data transmission and to move data in parallel so as to minimize the response time. An alternative way of performing join is to use semi-joins. Instead of performing joins in one step, semijoins are performed first to reduce the size of the relations. In the next step joins are performed on the reduced relations. The main value of the semijoin in a distributed system is to reduce the size of the join operands and then the communication cost. The theory of semijoins and their values for distributed query processing have been covered in [7, 11, 36]. Similarly, [10, 29] presented a distributed query processing that use joins instead of semijoins.

Wong [33] is the first researcher who proposes the algorithm for distributed query processing. Wong [33] algorithm is based on the hill climbing strategy. Wong algorithm translates a query into a sequence of two tactics: (1) move a sub-relation from one site to another, and (2) process data at one site using relational operations. The algorithm is a recursive optimization procedure. It begins by selecting a site S_a and then constructing the following initial solution.

1. Move all relations referenced by query to S_a.
2. Process at S_a as a local query, using the relations moved in step 1.

The initial solution is improved by recursively replacing individual "move" commands by lower cost sequences of "move" and "process" commands. The algorithm terminates when no "move" command can be replaced by a lower cost sequence. This algorithm produces increasingly efficient sequences of commands, although its hill climbing discipline is too weak to guarantee optimality. The main problem associated with this algorithm is a prior selection of S_a. Improved implementation of this algorithm based on semijoins is proposed in [7, 36].

Several distributed query processing algorithms have been proposed aiming at minimizing the amount of data transmission [7, 10, 11, 25, 29, 36]. A case study for distributed query processing is conducted by P. Agrawal et al. [1]. There are other query optimization strategies for distributed database systems some of which are extensions of centralized query processing [27].

C.T. Yu and C.C. Chang [36] describe the theoretical aspect of query processing in distributed database. They identified that only tree query can be fully reduced by semijoins and discuss about the algorithms that reduces a cyclic query into a tree query. Theirs semijoin strategy of query processing consists of three phase namely: copy identification phase, reduction phase and assembly phase, which is also quite similar in [11]. In copy identification phase, one or more copies of every relation appearing in the qualification of the query are identified and will be used to process the query. In reduction phase which is the most critical phase; semijoins are usually used to eliminate tuples of the relations that do not satisfy the qualification component of the query. In assembly phase, relations in the qualification component of the query are sent to one site to produce the output required. The assembly site is so chosen that the amount of data transfer between sites after reduction is minimal. Two steps are required to fully reduce a tree query. In first step, semijoin is applied from leaves to root of a tree which fully reduces the root relation where root of a tree query is arbitrarily chosen. In second step, the semijoin is applied from root to leaves which fully reduce the other relations, thus fully reducing all relations taking part in answering the query. Optimizing tree queries in distributed database is also studied in [37, 38].

Algorithm presented by Bernstein et al. [7] is a refinement of Wong's algorithm in which the concept of semijoin and reducer are used to abstract the main optimization problem. It considers only the communication cost, thus ignoring the local processing cost. This algorithm aims at minimizing intersite data transfer as it assumes that the network bandwidth to be the system bottleneck and seek to minimize use of this resource, all other resources are assumed to be free. The drawback of this algorithm is that it selects the semijoins that maximize immediate gain, ignoring the fact that the execution of one semijoin often decreases the cost and increases the benefit of other semijoins. This algorithm consists of reduction phase and assembly phase and is based on iterative hill climbing which is a kind of greedy algorithms. First phase executes relational operations at various sites of the distributed database in order to delimit a subset of database that contains all data relevant to the query and the latter one transmit the reduction to one designated site, and query is executed locally at that site. This first phase [7] can be taken as compact form of first two phases of [36] presented above. At the end, Bernstein et al. [7] suggest a variation for its enhancement.

A reduced cover set of the set of full reducer semijoin programs for an acyclic query graph for a distributed database system is covered by researchers [29]. They have presented an algorithm based on this reduced cover set which determines the minimum cost full reducer program. They have presented a low cost algorithm which determines a near optimal profitable semijoin program by converting a semijoin program into a partial order graph which expose the possibility of parallel execution of the semijoins in the program. LaFortune et al. [25] have presented a state transition model for the optimization of query processing in distributed database system. LaFortune et al. [25] have parameterized a problem by means of a state describing the amount of processing that has been performed at each site where database is located. A state transition occurs each time a new join or semijoin is executed.

All the algorithms before assumed that the strategy formulation delay is negligible in comparison to its execution delay. It is these two issues that are primarily investigated by P. Bodorik et al. [9] for a distributed database in which partitioned relations are permitted. So, response time include not only delays due to execution of strategies but also delays due to their formulation. P. Bodorik et al. [9] only concentrates on an important class of queries, the Select-Project-Join (SPJ) queries. They use a heuristic algorithm for processing distributed queries using generalized join whose overhead can be controlled. The trade-off between the strategy's execution and formulation delays is investigated. Their experimental result support the notion that simple greedy heuristic algorithms proposed by many researchers [7, 11, 36] are sufficient in a sense that they are likely to lead to near optimal strategies and that increasing the overhead in forming strategies is only marginally beneficial.

Similarity and distinction between distributed and parallel database system and the challenges offered by these techniques is discussed in [26]. Shared nothing architecture of parallel database system which closely resembles with distributed database system, differs it from mode of operation. In shared nothing multiprocessor systems, there is a symmetry and homogeneity of nodes; this is not true in distributed environment where heterogeneity of hardware and operating system at each node is a common place. Number of query processing techniques for the World Wide Web (WWW) which consist of semi-structured data distributed through out the world is addressed by Florescuu et al. [17]. They propose several techniques to manage websites and query a network for web pages as well as to manage and query a XML data. As our concern is only with the structured data, interested readers are referred to [17] for more detail survey on semi-structured data.

The effect of fragment and replicated strategy, local reduction strategy and integration of these two strategies are analyzed by P. Agrawal et al. [1]. They take into consideration all the factors that affect the distributed query processing i.e. local processing time, data communication time as well as parallel computing. Experimental result conducted by them conclude that fragment and replication strategy along with local reduction strategy (best) is better than fragment and replicated strategy along with all local reduction strategy and fragment and replicated strategy with no local reduction (worst) in all the cases. However, in reality this is not always true.

The problem of optimizing the processing of a single isolated query in a distributed database system has received a great deal of attention [1, 7, 9, 11, 33, 36]. Many algorithms have been devised for minimizing the cost associated with obtaining the answer to a single isolated query in a distributed database system. However, if more than one query may be processed by the system at the same time and if the arrival times of the queries are unknown the determination of optimal query processing strategies becomes a stochastic optimization problem. This stochastic optimization problem is the issue of [14]. In order to cope with such problems, P.E. Drenick and E.J. Smith [14] present a theoretical state transition model that treats the system as one operating under a stochastic load. Query processing strategies may then be distributed over the processors of a network as probability distributions in a manner which accommodates many queries over time.

An overview of the state of the art in distributed query processing is presented by Donald Kossmann [24]. He discussed various query processing techniques developed for the recent products and research prototypes and showed how they can be applied to different types of distributed systems. Much architecture can roughly be characterized by their communication paths (client-server, peer-to-peer, or multi-tier) and by the capabilities of the sites of the system (homogeneous or heterogeneous). For each category, Donald Kossmann [24] presented and discussed the set of query processing techniques which are particularly effective. For instance, [24] showed how to exploit the query capabilities of individual sites in a heterogeneous system. Independent of the specific architecture, all distributed database and information systems today are based on the two principles namely: best effort and flexible data placement.

In best effort the query processor always tries to execute a query as fast as possible or with as little cost as possible. At the heart of this strategy is a query optimizer, which decides for every query which query execution methods to use (e.g., which join method), where to execute these methods, and in which order to execute these methods. The optimizer can be used statically in order to compile a query once and for all times. The optimizer can also be used dynamically just before a query instance is executed or on the fly while the query is executed in order to adjust to the current state of the system.

While in a flexible data placement, in order to improve the performance of a whole query workload, caching and/or replication can be used in order to place data at or near sites where the data are frequently asked.

A large variety of economic model for various aspects of distributed computing have been studied since the mid-1980s (e.g. economic model for resource allocation, load balancing, flow control and quality of service). The motivation to use an economic model is that distributed systems are too complex to be controlled by a single centralized component with a universal cost model. Systems based on an economic model rely on the “magic of capitalism.” Every server that offers a service (data, CPU cycles etc.) tries to maximize its own profit by selling its services to clients. The hope is that the specific needs of all the individual clients are best met if all servers act this way [24]. Pentaris and Ioannidis [28] presents the e-commerce style of query processing (trading) in distributed database which include the bidding, bargaining and auction way of negotiation between the buyer and seller nodes viewing answer to the queries as commodities.

Mariposa is the first distributed database system based on economic paradigm. Mariposa process queries by carrying out auctions. In such auction, every server can bid to execute parts of a query and clients pay for the execution of their queries. More precisely, query processing in Mariposa works as follows [24, 28]:

1. Queries originate at clients and clients allocate a budget to every query. The budget of a query depends on the importance of the query and how long the client is willing to wait for the answer. A client in Las Vegas could, for example be willing to pay $5.00 if the client gets the latest World Cup football results within a second, but only 10 cents if the delivery of the results takes one minute.

2. Every query is processed by a broker. The broker parses the query and generates a plan that specifies the join order and joins methods.
3. The broker starts an auction. As part of this auction every server that stores copies of parts of the queried data or is willing to execute one or several of the operator specified in the broker's plan is asked to give bids in the form of <Operator o, Price p, Running Time t, Expiration Date x>. In other words, with such a bid a server indicates that it will be willing to execute operator o for p dollars in t seconds and that this offer is valid until the expiration date x.
4. The broker collects all bids and makes contracts with servers to execute the queries. Doing so, the broker tries to maximize its own profit. If, for example, the broker finds a way to execute the Las Vegas query from above in a second, paying only $1.00 to servers, the broker will pursue this way and keep $4.00 of the budget as profit. If the query cannot be evaluated with acceptable cost in one second, the broker will try to find a very cheap way to execute the query in a minute and keep a couple of cents as profit. If the broker finds no way to execute the query within time budget limitations, the broker will reject the query. In this case, the client must raise the budget, revise the response time goals, or just be happy without the answer.

The beauty of this strategy is that different servers can flexibly establish different bidding strategies in order to achieve high revenue and dynamic data placement nicely fits into it.

How non uniform bandwidth affects the query optimization in a distributed database system is investigated by Ip Alex et al. [23]. The cost based optimizer developed by Ip Alex et al. [23] manipulates both operator order and the physical locations at which these operators are evaluated and provides a basis for the detailed examinations of resource contention issues. They aim at enhancing the optimization strategy for distributed systems to account for non-uniform communication costs; particularly where the system is bottlenecked by lack of bandwidth in particular links. Ideas of how to tackle this bottleneck using the data shipping, query shipping and hybrid shipping strategy of query processing is also presented. Accelerating the query processing in distributed database system by using partially encoded record filter

(PERF) is studied by Haraty and Fany [20]. PERF is a new two way semijoin implementation primitive. The basic idea of PERF is as follows:

1. Project R (sending relation) on a joining attribute and get P_R.
2. Ship P_R to S (relation to be reduced).
3. Reduced S by a semijoin with P_R.
4. Send back to R, a bit vector (the PERF) that contains one bit for every tuple in P_R and in the same order. If the tuple is matching then send a 1 or else send a 0.

The fourth step is known as backward phase and which distinguish it from ordinary semijoin. The main utility of PERF is that it minimizes this phase and hence makes the forward phase (step 2) cost greater than the backward phase. PERF joins can be better enhanced by sending back to R not all the bit vector corresponding the P_R but only 0s part or 1s part according to which one is less in size and hence has lower transmission cost [20].

Yoo and LaFortune [35] have presented a heuristic method based on the A* algorithm that efficiently finds an optimal sequence of semijoins for a given query on a distributed database. The method generates new states from the given initial state by repeatedly doing semijoin on the relations. Each new state n is evaluated by estimating function $f(x) = g(x) + h(x)$. By providing that the admissibility and consistency conditions are satisfied, they have shown that an optimal sequence of semijoins can be found with the proposed function. Their solution on average search less than five percent of the search space before an optimal solution is found.

Basic concept about the distributed database system and its distinction from the centralized database system can be found in [13, 15, 30, 32] and core concept of it can be found in [27].

CHAPTER 4
RELATIONAL DATA MODEL

4.1 Introduction

The relational model was first introduced by Ted Codd of IBM Research in 1970 in a classic paper [Codd 1970], and attracted immediate attention due to its simplicity and mathematical foundations. The model uses the concept of a mathematical relation - which looks somewhat like a table of values - as its basic building block and has its theoretical basis in set theory and first order predicate logic. The relational data model provides a formal high level description of a collection of data items (i.e. a database) in terms of relations. The terms domain, attribute, relation, relation schema, relation state, contribute to the definition of the relational data model. These terms are defined as follows:

1. Domain: Domain is a set of data values. In other words, a domain D is a set of atomic values where atomic means that each value in the domain is indivisible as far as the relational model is concerned. A common method of specifying a domain is to specify a data type from which the data values forming the domain are drawn.
2. Attribute: An attribute is a name given to the set of data values. An attribute name which describes the set of data values is a subset of the possible values in a specific domain of values. Hence, each attribute is uniquely defined on one domain of values.
3. Relation: A relation can be thought of as a table in which each row of the table is called a tuple and each column of the tables is labeled by the attributes comprising the relation. No two tuples in the relation are identical.
4. Relation Scheme: A relation scheme names the relation and the attributes in that relation. The degree of a relation is the number of attributes of its relation schema.
5. Relation State: The contents of a relation at some moment in time, is called the relation state. Answer to any query presented at that time is dependent in that relation state.
6. Database Schema: Database schema is a set of relation schemas.

7. Database State: Database state is a set of relation states such that there is one relation state per relation schema.

In this work, relation name are capital letters from the latter part of the alphabet (e.g., Q, R) or the capital letter R, indexed by the integers (e.g., R1, R2 etc.). Attribute names are either capital letters from the first part of the alphabet (e.g., A, B) or the capital letter A, indexed by the integers (e.g., A1, A2 etc.). Some specific relation example use more descriptive relation and attribute names.

Given the preceding definitions, we can define following relation and attribute parameters.

For a relation R, let:
n = number of tuples (i.e. cardinality of a relation).
a = number of attributes.
For an attribute Ai, let:
Di = number of possible domain values (i.e. the domain cardinality).
Ci = number of distinct values currently in Ai (i.e. the attribute cardinality).
Wi = size of a data item in Ai.
Size of R (sR) = n * $\prod_{i=1}^{a}$ Wi.
Projected size of the attribute with no duplicate values (sAi) = Wi * Ci.

It is assumed Wi the size of a data item in Ai, is one unit of data in size. This assumption does not limit the generality of our work since the projected size of an attribute is always directly proportional to the cardinality of the attribute and the size of a relation is always directly proportional to the cardinality of that relation. Furthermore, this assumption allows us to make the following simplifications on the relation and attribute parameters.

For a relation R, then:
n = number of tuples.
a = number of attributes.
sR = size of R, sR = n *a.

For an attribute Ai, then:
Di = number of possible domain values.
Ci = number of distinct values currently in Ai.
sAi = projected size of the attribute with no duplicate values, sAi = Ci.

4.2 Relational Operations

The relational operations enable the user to specify basic retrieval requests. Each node in a distributed database contains a local database. Each local database is described by a database schema and database state. Relational operations in particular the join operation can be defined intra-nodally (within a node) or inter-nodally (between nodes).

Relational operations concerned with this work are discussed next.

4.2.1 Selection

A selection also known as restriction because of its behaviour, is a intra-nodal operation and can be represented as R[A=x]. Attribute A in relation R is to be restricted to those values in column A such that they equal the value x. Formally:
$R[A=x] = \{ r \in R \mid r.A=x\}$

In relational algebra it is represented as:
$\sigma_{\text{<selection condition>}} (R)$
where the symbol σ (sigma) is used to denote the SELECT operator and the selection condition is a Boolean expression specified on the attributes of relation R. The Boolean expression specified in <selection condition> is made up of a number of clauses of the form
<attribute name> <comparison op> <constant value> or
<attribute name> <comparison op> <attribute name>
where <attribute name> is the name of an attribute of R, <comparison op> is normally one of the operators $\{=, <, \leq, >, \geq, \neq\}$ and <constant value> is a constant value from the attribute domain.

In SQL; ‘WHERE’ part represents the restriction.

4.2.2 Projection

A projection is an intra-nodal operation and can be represented as R[A]. The projection of relation R on attribute A is obtained by eliminating all columns of R not labeled by A and then eliminating duplicate tuples. Formally:
$R[A] = \{ r.A \mid r \varepsilon R\}$

In relational algebra it is represented as:
$\pi_{<\text{attribute list}>} (R)$
where the symbol π (pi) is the symbol used to represent the ‘PROJECT’ operation and <attribute list> is a list of attributes from the attributes of relation R.

In SQL, ‘SELECT’ part represents the projection.

4.2.3 Join

A join operation can be represented as R[A=B]S. The join operation is used to combine two relations. For the join of two relations to happen, the joining attributes of these relations must be defined on the same domain. For the above case, attributes A and B must be defined on the same domain. Value of A in R is compared with a value of B in S. If the two values have the relationship specified in the join operation (e.g. ‘=’, equality), then the tuples of the relations are combined to form a third relation. Formally:
$R[A=B]S = \{r.s \mid r \varepsilon R, s \varepsilon S \text{ and } r.A=s.B\}$

In relational algebra, join is denoted by:
$R \bowtie_{<\text{join condition}>} S$

The join is inter-nodal if the two relations are at different nodes in the network. To perform an inter-nodal join, one of the two relations must be moved to the node where the other relation resides. There are different ways of joining the relations such as equi-join, left outer join, right outer join, full outer join. However, this work is concerned only with the equi-join.

4.2.4 Semi-join

Semi-join is used as an inter-nodal operation and can be represented as R<A=B]S, where S is the sending relation and R is a reduced relation. The semi-join operation may be used to perform the join of the two relations without moving an entire relation to perform the join. The semi-join is performed by first projecting S.B at the node where relation S resides. S.B is then transferred to the node where relation R resides. R is then restricted to those values of R.A equal to those values in S.B., which reduce the relation R. Let that reduced relation R be R'. R' is then sent to the site where S resides to perform the join of R' and S in order to obtain the final result. The main motivation to use semijoin is that the semi-join R<A=B]S will significantly reduce R so that transferring the reduced relation R (i.e. R') has a greater impact on the communication cost in comparison to unreduced one (i.e. original relation R). Formally:
R<A=B]S= {r | r ε R and r.A ε S[B]}

For joining two relations R and S located at different sites, there are two possibility
R<A=B]S
or
S<B=A]R.

Our semi-join tactics will consider both but only add one of them in a semi-join program at a time that is more beneficial than other in terms of average percentage gain in reduction of joining attributes cardinalities (covered in chapter 7). A semijoin program is a sequence of semijoins which describes the order in which the semijoin is to be performed in order to obtain the result of the requested query. There is slightly one more variation of semi-join which is called 2-way-semijoin in which the reduced relation again send the projection of the joining attribute to the sending relation such that the sending relation is also reduced i.e., if R<A=B]S is the first semijoin applied and R' be the reduced relation R then semijoin S<B=A]R' is applied after that so that

the relation S is reduced to S'. It is clear from the discussion that R' $\subseteq$ R i.e., reduced relation is a proper subset of original relation.

The cost of a semijoin R<A=B]S is defined to be the cost of transferring S[B] from the site containing S to the site containing R (if the two site are identical, the cost is zero). The benefit of the semijoin is the size of R before the operation minus the size of R after the operation i.e., the benefit of semijoin is the amount of data it eliminates for inter-nodal data transfer. A semijoin is said to be profitable if its cost is less than its benefit, i.e. cost of semijoin < benefit of semijoin. It is not always the case that semijoin is beneficial. Sometimes simple join is beneficial than semijoin. Our semijoin program only adds semijoin that is beneficial to semijoin program. Semijoin, unlike join, is asymmetric; that is, R<A=B]S is not equal to S<B=A]R. The former reduces the R, while the latter reduces S.

The preference of semijoin over join as indicated by [7], is mainly for three reasons. First, R<A=B]S $\subseteq$ R, and so semijoins monotonically reduce the size of the database. By contrast joins can increase the size of database, in worst case |R[A=B]S| = |R|*|S| . Second, semijoin can be computed with less data transfer than joins. To compute semijoin R<A=B]S, we need to transfer only the projection of S on attribute B i.e. S[B] whereas to compute join R[A=B]S, we must transfer an entire relation. Of course, semijoins may also have less effect than the join, since R<A=B]S only reduces R, whereas R[A=B]S simultaneously reduces R and S. However, the third advantage of semijoins is that the "reduction effect" of any single join can be attained by two semijoins. An optimal query processing algorithm would almost certainly include both joins and semijoins.

The following example illustrates the above operations. Given the two relations R1 and R2 with relation states:

R1:

A	B
1	Unary
2	Binary
3	Ternary

R2:

C	D
-	1
-	2
*	2
/	2
+	2

The restriction R2[D=2] has the following result.

R2':

C	D
-	2
*	2
/	2
+	2

The projection R2[C] has the following result.

R2':

C
-
*
/
+

The intra or inter-nodal join, R1[A=D]R2 has the following result.

R3:

A	B	C	D
1	unary	-	1
2	binary	-	2
2	binary	*	2
2	binary	/	2
2	binary	+	2

The semi-join R1<A=D]R2, has the following result at the node at which R1 resides.

R1':

A	B
1	unary
2	binary

To complete the join, this reduced relation is then transferred to the site containing relation R2 and join is performed over there to obtain final result R3 as in the normal join.

The below example taken from [4] illustrates where join is more beneficial than semijoin.

Let the relation R1, R2, R3 and R4 are stored in Site1, Site2, Site3 and Site4 respectively with following relation state:

R1:

A	B
0	1
1	0

Site1

R2:

B	C
1	1
0	0

Site2

R3:

A	C
1	1
0	0

Site3

R4:

C	D	E	F	G	H	I
0	0	0	0	0	0	0
1	1	1	1	1	1	1

Site4

Let the 'WHERE' part of the SQL query is:

R1.A=R3.A

AND

R1.B=R2.B

AND

R2.C=R4.C

AND

R3.C=R4.C

Now using the semijoins, the optimal evaluation is to move R1, R2 and R3 to site 4, i.e.; no semijoin should be used as there is no benefit from any of it and our semijoin program would not add any. If we assume that each data item is 1 unit in length, then this require 12 units to be transferred to answer the query. Using the join the optimal evaluation is:

R12 = R2[B=B]R1 at Site2. This cost 4 units of data transfer.

R123 = R12[A=A AND C=C]R3 at Site2. This cost 4 units of data transfer.

Note that R123 = { }.

R4 = R4[C=C]R123 at site 4. This cost 0 unit of data transfer because R123 is empty.

So in total, the cost of answering the query with join is 4 + 4 + 0 = 8 units.

4.3 Relational Queries

Queries in a relational database system may be expressed using high-level non-procedural query language. Codd's original paper [Codd 70] set the basis for two families of relational query languages, relational calculus and relational algebra. The definitions of restriction, projection, join and semijoin presented earlier are expressed in relational algebra. SQL has been used to illustrate example in this work. The distributed aspect of relational queries is discussed in chapter 5.

4.4 Reasons for Using Relational Data Model

The reasons for choosing the relational data model as the underlying formulation are: the mathematical foundation of the relational model makes it a good candidate for theoretical treatment, the relational DBMS market has matured and is now sizable and finally most distributed database systems are also relational.

CHAPTER 5
QUERY PROCESSING IN DISTRIBUTED DATABASE

5.1 Query Processing Problem

The main function of a relational query processor is to transform a high level query (typically, in relational calculus) into an equivalent low-level query (typically, in some variation of relational algebra). The low level query actually implements execution strategy for the query. The transformation must achieve both correctness and efficiency. It is correct if the low level query has the same semantics as the original query, that is, if both queries produce the same result. The well defined mapping from relational calculus to relational algebra makes the correctness issue easy. But producing an efficient execution strategy is more involved. A relational calculus query may have many equivalent and correct transformation into relational algebra. Since each equivalent execution strategy can lead to very different consumptions of computer resources, the main difficulty is to select the execution strategy that minimizes resource consumption.

For example: Suppose the database have following tables:
EMP (<u>ENO</u>, ENAME, TITLE)
ASG (<u>ENO, PNO</u>, RESP, DUR)
And the following simple user query:
"Find the name of employees who are managing a project."
The expression of the above query in relational calculus using SQL syntax is:

```
SELECT    ENAME
FROM      EMP, ASG
WHERE     EMP.ENO=ASG.ENO AND RESP='Manager'
```

Two equivalent relational algebra queries that are correct transformation of the query above are:

$\Pi_{ENAME}(\sigma_{RESP='Manager' \wedge EMP.ENO=ASG.ENO}(EMP \times ASG))$

and

$\Pi_{ENAME}(EMP \bowtie_{ENO} (\sigma_{RESP='Manager'} (ASG)))$.

It is intuitively obvious that the second query which avoids the cartesian product of EMP and ASG consumes much less computing resources than the first and thus should be retained (extracted from [27]).

In a centralized context, query execution strategies can be well expressed in an extension of relational algebra. The main role of a centralized query processor is to choose, for a given query, the best relational algebra query among all equivalent ones. Since the problem is computationally intractable with a large number of relations, it is generally reduced to choosing a solution close to optimum.

In a distributed system, relational algebra is not enough to express execution strategies. It must be supplemented with operations for exchanging data between sites. Besides the choice of ordering relational algebra operations, the distributed query processor must also select the best sites to process data, and possibly the way data should be transformed. This increases the solution space from which to choose the distributed execution strategy, making distributed query processing significantly more difficult.

The below example illustrates the importance of site selection and communication for a chosen relational algebra query against a fragmented database. We consider the following query of above example

$\Pi_{ENAME}(EMP \bowtie_{ENO} (\sigma_{RESP='Manager'} (ASG)))$

We assume that relations EMP and ASG are horizontally fragmented as follows:

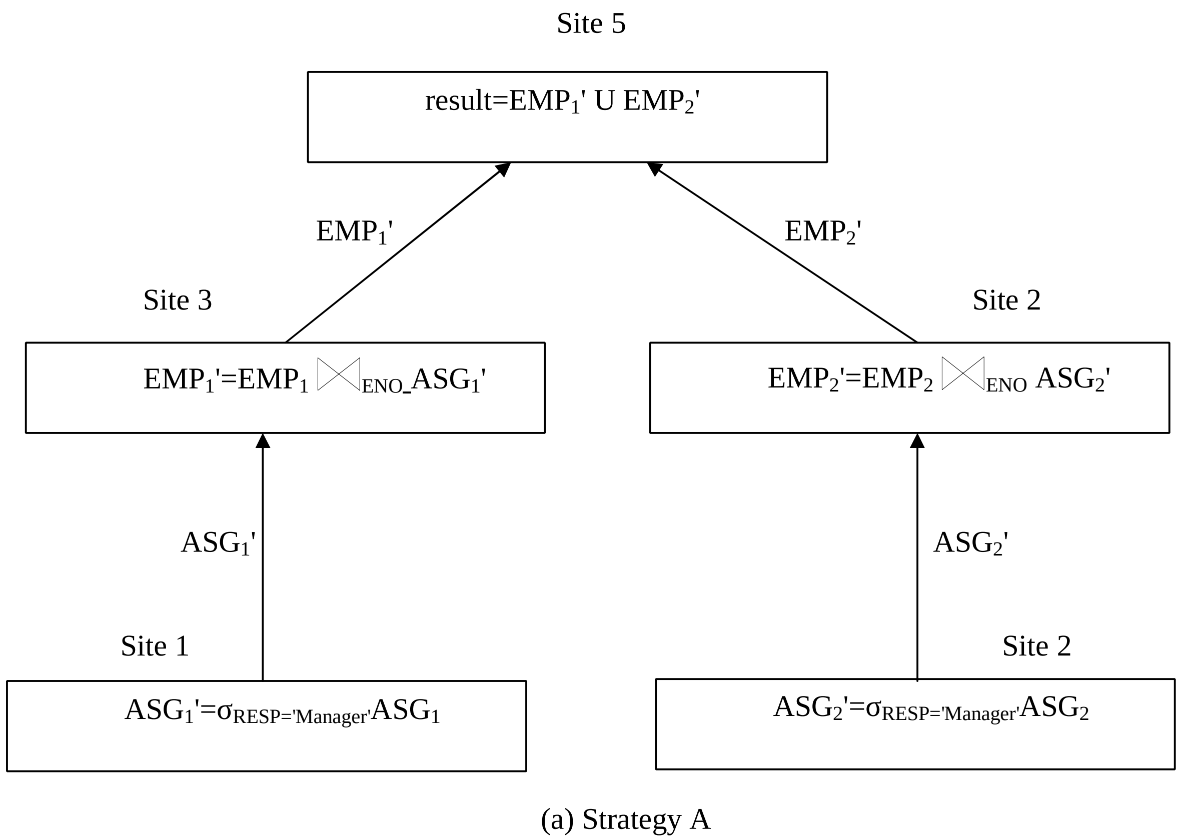
$EMP_1=\sigma_{ENO\leq"E3"}(EMP)$

$EMP_2=\sigma_{ENO>"E3"}(EMP)$

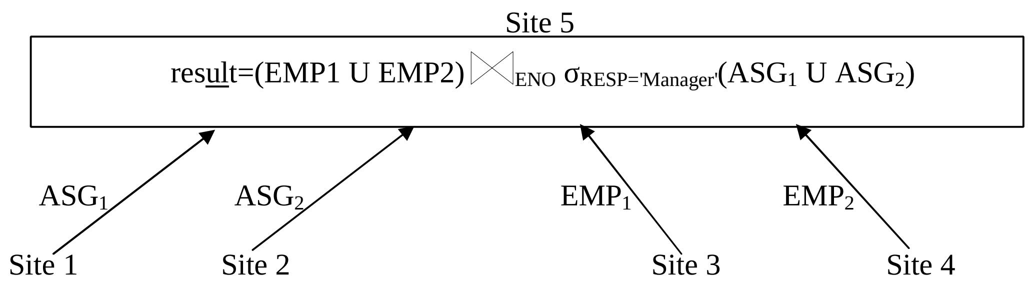
$ASG_1=\sigma_{ENO\leq"E3"}(ASG)$

$ASG_2=\sigma_{ENO>"E3"}(ASG)$

Fragments ASG_1, ASG_2, EMP_1 and EMP_2 are stored at site 1, 2, 3 and 4 respectively, and the result is expected at site 5.



(a) Strategy A



(b) Strategy B

Figure 5.1 Equivalent Distributed Execution Strategies

For the sake of simplicity, we ignore the project operation in the Figure 5.1 which show two equivalent distributed execution strategies for that above query. An arrow from site i to site j labeled with R indicates that relation R is transferred from site i to site j. Strategy A exploits the fact that relations EMP and ASG are fragmented the same way in order to perform the select and join operation in parallel. Strategy B centralizes all the operand data at the result site before processing the query.

To evaluate the resource consumption of these two strategies, we use a simple cost model. Let us assume that a tuple access denoted tupacc, is 1 unit (which is left unspecified) and a tuple transfer, denoted tuptrans, is 10 units. Let us assume that relations EMP and ASG have 400 and 1000 tuples, respectively, and that there are 20 managers in relations ASG. Similarly, assume that data is uniformly distributed among sites. Finally let us assume that relations ASG and EMP are locally clustered on attributes RESP and ENO respectively. Therefore, there is direct access to tuples of ASG (respectively, EMP) based on the value of attribute RESP (respectively, ENO) [27].

The total cost of strategy A can be derived as follows:

1. Produce ASG' by selecting ASG requires (10+10) * tupacc = 20
2. Transfer ASG' to the site of EMP requires (10+10) * tuptrans = 200
3. Produce EMP' by joining ASG' and EMP requires (10 +10) * tupacc * 2 = 40
4. Transfer EMP' to result site requires (10+10) * tuptrans = 200

Total cost is 460.

Similarly, the cost of strategy B can be derived as follows:

1. Transfer EMP to site 5 requires 400 * tuptrans = 4,000
2. Transfer ASG to site 5 requires 1000 * tuptrans =10,000
3. Produce ASG’ by selecting ASG requires 1000 * tupacc =1,000
4. Join EMP and ASG’ requires 400 * 20 * tupacc =8,000

Total cost is 23,000.

In strategy B we assumed that the access methods to relations EMP and ASG based on attributes RESP and ENO are lost because of data transfer. This is reasonable assumption in practice. Strategy A is better by a factor of 50, which is quite significant. Furthermore, it provides better distribution of work among sites. The difference would be even higher if we assumed slower communication and/or higher degree of fragmentation.

5.2 Query Processing Objectives

The objective of query processing on distributed database systems is to facilitate users to express queries as if the distributed database were a single unified database. The fact that data is actually distributed physically is transparent to the user. The user should however be able to direct the result of a query to any node (site) in the network. This node (site) is termed the 'result node'.

The main objective of query processing is query optimization. Because many execution strategies are correct transformation of the same high level query, the one that optimizes (minimizes) resource consumption should be retained.

A good measure of resource consumption is the total cost that will be incurred in processing the query. Total cost is sum of all times incurred in processing the operations of the query at various sites and in inter-site communication. Another good measure is the response time of the query [16] which is the time elapsed for executing the query. Since operations can be executed in parallel at different sites, the response time of a query may be significantly less than its total cost. Finding the optimal solution is computationally intractable (being NP HARD), so the goal of query processing is to find an execution strategy for the query which is close to optimal.

There are different approaches in finding this close to optimal solution like dynamic programming, heuristic based programming which is based on statistics of a database which will be explored in coming section.

5.3 Types of Optimization

Conceptually, query optimization aims at choosing the best point in solution space of all possible execution strategies. An immediate method for query optimization is to search the solution space exhaustively, predict the cost of each strategy and select the one with the minimum cost. Although this method is effective in selecting the best strategy, it may incur a significant processing cost for the optimization itself. The problem is that the solution space can be large; that is, there may be many equivalent strategies even with a small number of relations. The problem becomes worse as the number of relations or fragments increases (e.g. becomes greater than 5 or 6) [27]. Having high optimization cost is not necessarily bad, particularly if query optimization is done once for many subsequent executions of the query. Therefore, an exhaustive search approach is often used whereby (almost) all possible execution strategies are considered [31].

To avoid the high cost of exhaustive search, randomized strategies, such as iterative improvement and simulated annealing have been proposed. They try to find a very good solution, not necessarily the best one, but avoid the high cost of optimization, in terms of memory and time consumption.

Another popular way of reducing the cost of exhaustive search is the use of heuristics, whose effect is to restrict the solution space so that only a few strategies are considered. In both centralized and distributed systems, a common heuristic is to minimize the size of intermediate relations. This can be done by performing unary operations first, and ordering the binary operations by the increasing sizes of their intermediate relations. An important heuristic in distributed systems is to replace join operations by combinations of semi-joins to minimize data communication.

A query may be optimized at different time relative to the actual time of query execution. Optimization can be done statically before executing the query or dynamically as the query is executed. Basically, there are three types of query optimization in accordance to optimization time. They are:

5.3.1 Static Query Optimization

Static query optimization is done at query compilation time. Thus, the cost of optimization may be amortized over multiple query execution. Therefore, this query optimization is quite appropriate for use with the exhaustive search method. Since the size of the intermediate relations of a strategy is not known until run time, they must be estimated using database statistics. Errors in these estimates can lead to the choice of suboptimal strategies.

5.3.2 Dynamic Query Optimization

Dynamic query optimization proceeds at query execution time. At any point of execution, the choice of the best next operation can be based on accurate knowledge of the results of the operations executed previously. Therefore, database statistics are not needed to estimate the size of intermediate results. However, they may still be useful in choosing the first operations. The main advantage of dynamic query optimization is that the actual sizes of intermediate relations are available to the query processor, thereby minimizing the probability of a bad choice. The main shortcoming is that query optimization an expensive task, must be repeated for each execution of the query. Therefore, this approach is best for ad-hoc queries. One way of performing the dynamic query optimization in distributed database systems is to use dynamic programming. Dynamic programming, like the divide-and-conquer method, solves problems by combining the solutions to sub-problems. Dynamic programming is applicable when the subproblems are not independent, that is when subproblems share subsubproblems [12].

5.3.3 Hybrid Query Optimization

Hybrid query optimization attempts to provide the advantages of static query optimization while avoiding the issues generated by inaccurate estimates. In other words, it tries to exploit the advantages of both static and dynamic query optimization. This approach is basically static, but dynamic query optimization may take place at run time when a high difference between predicted sizes and actual size of intermediate relations is detected.

5.4 Statistics

The effectiveness of query optimization relies on statistics on the database. Dynamic query optimization requires statistics to choose which operation should be done first. Static query optimization is even more demanding since the size of intermediate relations must also be estimated based on statistical information. In a distributed database, statistics for query optimization typically bear on fragments, and include fragment cardinality and size as well as size and number of distinct values of each attribute. To minimize the probability of error, more detailed statistics such as histograms of attribute values are sometimes used at the expense of higher management cost. The accuracy of statistics is achieved by periodic updating. With static optimization, significant changes in statistics used to optimize a query might result in query reoptimization.

5.5 Layers of Query Processing

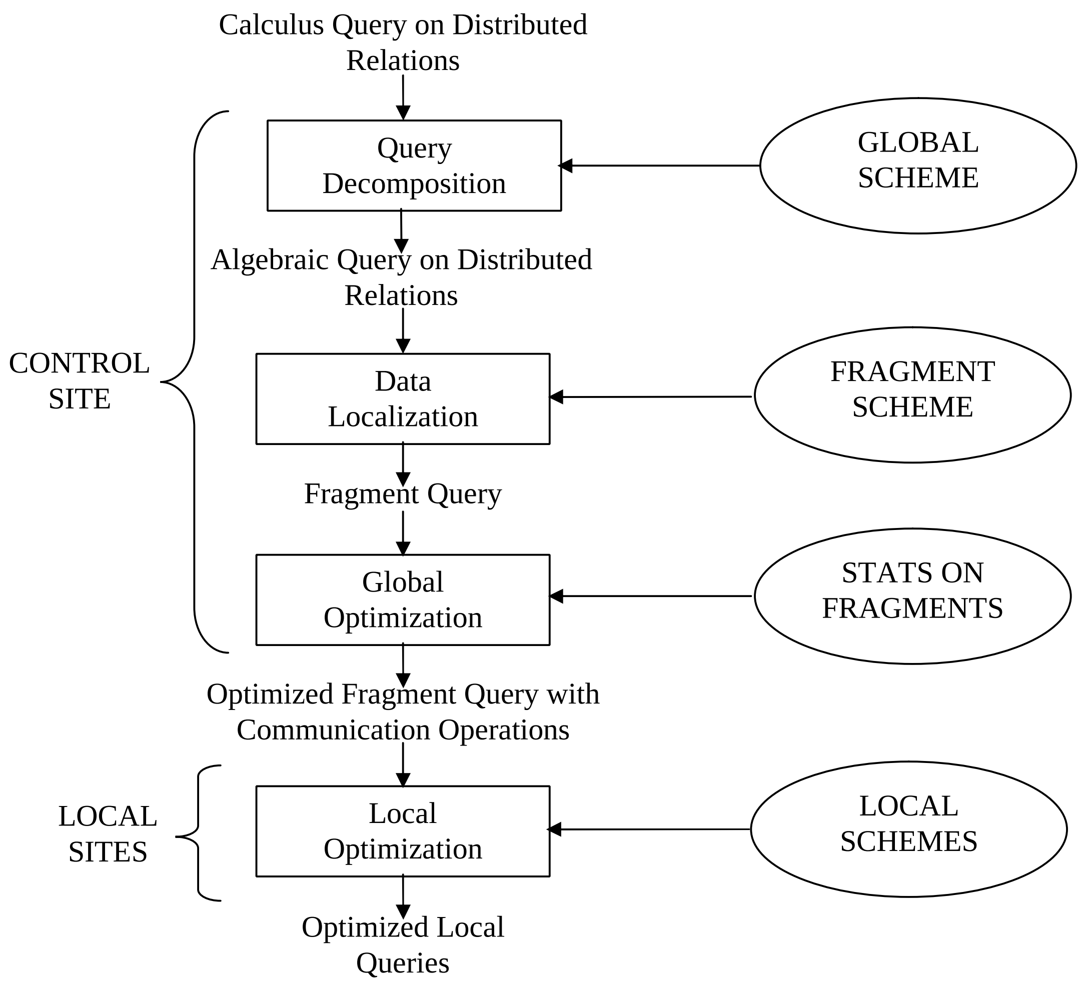


Figure 5.2 Generic Layering Scheme for Distributed Query Processing

As shown in the Figure 5.2 generic layering scheme for distributed query processing can be isolated into four main functions namely: query decomposition, data localization, global query optimization and local query optimization.

Query decomposition and data localization correspond to query rewriting. The first three layers are performed by central site and use global information, the fourth is done by the local sites.

Once a query has been expressed, we need to follow some query processing scheme to answer the query. The query processing scheme in general use [7, 21] is as follows:

1. Initial Local Processing

   The relational operations projection, selection and intra-site join are done first to reduce the amount of data before any data transmissions are made.

2. Processing Strategy

   A sequence of data transmission steps and local processing steps are done to further reduce the amount of data involved in answering the query. This step can be considered to be a preprocessing step (for the next step).

3. Final data transmission

   Data are then transmitted from the distributed nodes to the result node where final local processing is done to form the result of the query.

Steps 1 and 3 represent well known techniques in database management systems. Step 2 has been the focus of research [7, 21]. Without processing strategies, whole relations would always have to be transmitted on the network to answer queries. This would lead to excessive data communication costs and a high likelihood of a several congested communication network. This work is also focused on Step 2.

5.6 Assumptions

In this work, relational database systems are considered. Further, it is assumed that the cost of local processing is zero and all possible initial local processing like project, selection and intra-nodal join are performed before. Only one copy of the relation exists and each site can contain only one relation at a time. Only one query can be subjected to the system at a time and only retrieval based query is considered.

The cost measure is defined in terms of the total data transmission cost. The transmission cost of sending X bytes of data from site R to site S is assumed to be $C_0 + C_1 * X$, where C_0 is start-up cost of initiating transmission and C_1 is a proportionality constant.

Similarly to [7], the following assumptions are made for estimating the effect of a semi-join.

1. The distinct values in an attribute of a relation are assumed to be uniformly distributed. The probability that a tuple has a particular value is same as the probability that it has any other value. This is clearly a critical assumption and quite strong since it will be utilized in forming query processing strategies. If attribute value distribution is not uniformly distributed then the strategies formed will not perform as predicted.
2. If the number of distinct values in one attribute is reduced by a semijoin, the number of distinct values in each of other attributes in the same relation will also be reduced. Although there is no agreement in the literature regarding how cardinalities of attributes other than the joining attribute are updated with respect to the semi-join, this work uses approximation of Yao's function [34] which depends on the selectivity of the semijoin as in [7] (more will be covered in next chapter).

It is assumed that the system parameters like size of each relation, number of attributes in each relation, size of the attribute in each relation, cardinality of domain are contained in system catalog which is used for the cost-benefit estimation of semijoin. It is also assumed that there is only one join attribute between any two relations. All the assumptions are made to simplify the calculation. However, our assumption does not take anything away from generality as the final objective is to find a semi-join program with optimal cost-benefit.

5.7 Complexity Considerations

In the general query environment, the problem of determining an optimal semi-join processing strategy is a NP-Hard problem [21]. NP-Hard problems are those problems for which there exists no deterministic polynomial time algorithm to solve it. So, algorithms which generate these preprocessing strategies are necessarily heuristic in nature.

5.8 Database State Update Consideration

A problem is encountered when semi-join is used as query processing tactic. When a particular semi-join program is generated the cost beneficial semi-joins are added to it incrementally. For each semi-join added to the semi-join program, the database state needs updating to reflect the execution of this semi-join so that the next semi-join to be considered may have its correct cost and benefit determined. Research has shown that the history of previous semi-joins that are already in a semi-join program can affect the effect on the database state of a new semi-join that is being considered for addition to the semi-join program (more will be covered in next chapter).

CHAPTER 6
SEMIJOIN COST/BENEFIT ESTIMATION

6.1 Introduction

One of the main factors affecting the performance of a query processing strategy is the size of the intermediate relations produced during the query execution phase. Estimating the communication cost to send a stored relation to another site while executing a query is easy since the size of the relation is known. The situation become more complicated when we need to estimate the communication cost to send some intermediate relation to another site prior to the execution of the query. Since it is necessary to estimate the sizes of intermediate relations, this estimation is based on statistical information about the relations involved in the query and formula used to predict the cardinalities of the relations obtained from the sequence of database operations. There is a direct trade off between the precision of the statistical information and the cost of calculating such information.

When semijoin algorithm is applied, it looks for each possible semijoin i.e., candidate semijoin (CSJ) at each step for possible addition to the semijoin program (SJP). The cost of CSJ depends on the current state of the database. The benefit which is the amount of reduction that could be obtained after the semijoin is performed, will involve a hypothetical update of the database state. Thus, cost/benefit problem can be correspond to the problem of estimating the database state each time a semijoin has been added to SJP.

The current state of database consists of cardinalities (i.e., number of tuples) of each relation and the cardinalities (i.e., the number of distinct values) of each attribute. Given a new semijoin to be added to SJP, we only update the state of the relation that is being reduced.

6.2 Estimation

Performance of a distributed query processing algorithm depends to significant extent on the estimation algorithm used to evaluate the expected sizes of some intermediate relations. The choice of a reasonable estimation algorithm is therefore extremely important.

The cost of R<A=B]S is defined to be the amount of intersite data transfer required to compute it. This equals

$$\begin{cases} 0 \text{ if R and S are stored at the same site.} \\ \text{sB otherwise.} \end{cases}$$

where sB is the product of cardinality of the S projected on attribute B times width of the domain B.

A semijoin R<A=B]S is said to be beneficial if the size of the relation R before the semijoin is greater than the size of the relation R after the semijoin. Given the number of distinct value of an attribute say B on relation S and domain cardinality of an attribute B one can calculate the selectivity of an attribute B on relation S. Selectivity factor of an attribute B on relation S is the ratio of cardinality of distinct value of B in S by domain cardinality of attribute B. Selectivity factor determines how reducing relation gets affected on that join attribute. Formally,

$$\text{Selectivity (SF(Ri.Ai))} = \frac{|\text{ value of Ai in Ri}|}{|\text{domain value of Ai}|} \qquad \text{---------------------------} \; (6.1)$$

where SF(Ri.Ai) is the selectivity of an attribute Ai in relation Ri and $0<\text{SF(Ri.Ai)} \leq 1$.

If the SF(Ri.Ai) is close to 0, then it is said to have a good selectivity and if it is closer to 1, it is said to have bad selectivity because it will not reduce the relation significantly.

The estimation of the size of intermediate relation and distinct values of joining and non-joining attribute of the reduced relation due to the effect of CSJ added to the SJP is described below.

6.2.1 Effect of Semijoins on Relation and Joining Attribute Cardinality

Let R<A=A]S be a semijoin that is to be added to the semijoin program (SJP). Then the formula to calculate the new cardinality of R is as follow:

$$|R| = CC(R) * SF(S.A) \qquad \text{----------------------------} \ (6.2)$$

where CC(R) is the current cardinality of R i.e., cardinality of R before semijoin and SF(S.A) is the selectivity of relation S on attribute A.

Similarly, the new cardinality of the join attribute A of R is calculated as follow:

$$|R.A| = CC(R.A) * \frac{(\text{new cardinality of } R \text{ i.e., } |R|)}{(\text{old cardinality of } R \text{ i.e., } CC(R))} \qquad \text{----------------------------} \ (6.3)$$

where CC(R.A) is the number of distinct value of A in R before the semijoin and |R| is the cardinality of relation R after the semijoin.

New selectivity factor of relation R on join attribute A can be calculated as:

$$SF(R.A) = SF(R.A) \text{ before semijoin} * SF(S.A) \qquad \text{----------------------------} \ (6.4)$$

6.2.2 Effect of Semijoins on Non-joining Attribute Cardinality

There is no agreement in the literature regarding how cardinalities of attributes other than join attribute say A are updated with respect to the semijoin say R<A=A]S, reducing R.A. [21] mentions that they should remain unchanged, while Yu et al. [37] argue that they should be reduced in the same manner as R.A by applying the selectivity of the semijoin to the current cardinalities of each attribute. Bernstein et al. [7] use an approximation of Yao’s function [34] which depends on the selectivity of the semijoin. It is felt that Bernstein et al. [7] have the most realistic approach and will be used in this work.

Bernstein et al. [7] algorithm analyzes the effect on other attributes in R as a 'hit ratio' problem provided that the attributes are uniformly distributed among relations. If we are given n objects (corresponds to cardinality of R), distributed uniformly over m colours (corresponds to cardinality of R[B] where B≠A) then the hit ratio problem may be stated as, "How many distinct colours are we expected to hit if we randomly select r of the objects?".

The answer is given by Yao [34] as follow:

$$Y(m, n, r) = m * (1 - \prod_{i=1}^{r}[(nd - i + 1)/(n - i + 1)]), \text{ where } d = 1 - 1/m \quad \text{----------------} (6.5)$$

The computation of equation 6.5 is time consuming, so in practice Bernstein et al. [7] approximate Y(m, n, r) by:

$$Y(m, n, r) = \begin{cases} r & \text{if } r < m/2 \\ m & \text{if } m < r/2 \\ (r + m)/3 & \text{otherwise.} \end{cases} \quad \text{----------------------------} (6.6)$$

The above equation is simple yet powerful enough to estimate the reasonable reduction on other attributes other than the join attribute of a reduced relation.

Example:

Consider a query consisting of a join of relation R and S on attribute A i.e., R[A=A]S with the following database statistics as shown in Table 6.1 and 6.2.

X	A	B	C	D	E	F
|X|	10000	8000	9000	7000	7000	90000

Table 6.1 Domain Values

Ri	|Ri|	X	|Ri.X|
R	5680	A	360
		B	320
		D	1400
		E	45
S	5140	A	450
		C	360
		F	900

Table 6.2 Database Statistics

From the above Table 6.2 one can calculate the selectivity of each attribute for each relation using equation 6.1 and it is shown in table below.

Ri	X	SF(Ri.X)
R	A	0.036
	B	0.04
	D	0.2
	E	0.006
S	A	0.045
	C	0.04
	F	0.1

Table 6.3 Selectivity Factor of Each Attribute of Each Relation of Table 6.2

If for example, candidate semijoin (CSJ) R<A=A]S to be added to the semijoin program (SJP) then the database state of relation R need to be updated accordingly. The new cardinality of relation R can be calculated using equation 6.2.

$|R| = CC(R) * SF(S.A) = 5680 * 0.045 = 255.6 \approx 256$

Similarly, distinct value of attribute A in relation R after semijoin can be calculated using equation 6.3.

$$|R.A| = CC(R.A) * \frac{(|R| \text{ after semijoin})}{(|R| \text{ before semijoin})} = 360 * \frac{256}{5680} = 16.22 \approx 17$$

The effect of semijoin R<A=A]S on the other attributes other than the join attribute i.e., A can be computed using equation 6.6.

$$|R.B| = \frac{(256 + 320)}{3} = 192$$

|R.D| = 256

|R.E| = 45

In short when the semijoin R<A=A]S is added to semijoin program for the data given in Table 6.2, a hypothetical update to the database statistics of relation R has to be done and this hypothetical update is shown in Table 6.4.

Ri	|Ri|	X	|Ri.X|
R	256	A	17
		B	192
		D	256
		E	45

Table 6.4 Hypothetical Database Statistics Update of Relation R

CHAPTER 7
PREPROCESSING STRATEGIES ALGORITHMS

7.1 Introduction

Generally, in a distributed environment, the semijoin operation has been used as a preprocessing strategy for queries. The objective of semijoin preprocessing is to reduce the amount of data required to be transmitted on the network by first reducing the cardinalities of distributed relations using semijoin and then transmitting the resultant relations to the result node. Algorithm that utilizes the semijoin tactics produce as output a semijoin program that is to be executed on the network. Semijoin program signifies the order that needs to be carried out in order to obtain the result of a query.

In this chapter, we explore the two known algorithms in this field, algorithm SDD-1 [7] and algorithm AHY [2]. We next present algorithm LIGHT, a query preprocessing algorithm that represents a result from our work. We then present a concept of re-organization algorithm as explained in [36] which is proposed by Luk and Luk in 1980. This re-organization algorithm when given a semijoin program, produces a new semijoin program that is guaranteed to have a new cost less than or equal to the cost of the original semijoin program. In algorithm AHY, we only discuss one version that is relevant to us although [2] has presented 3 different types of algorithms. We also discuss how algorithm SDD-1 can be utilized to calculate the total time while simultaneous transmission of independent semijoin is permitted. Last section of this chapter; give an example query along with the semijoin produce by these first three algorithms on this example query.

It should be noted again that all these algorithms are heuristic in nature which use database statistics, as the determination of an optimal semi-join program is NP-HARD problem. To date, no algorithms producing optimal semijoin programs are known.

7.2 Initial Feasible Solution

The initial feasible solution (IFS) for a given query is to first perform all initial local processing of relations taking part in the query and then to transmit all relations to the result node.

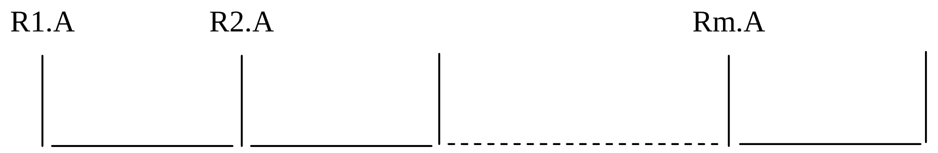
7.3 Algorithm AHY

Only algorithm GENERAL: Total Time as presented in [2] is discussed in this work. Hevner and Yao were the first researchers to determine an optimal solution to simple queries (queries with only one joining attribute and no output attributes other than the joining attribute). Hevner proved that finding an optimal solution for more general queries is NP-HARD problem. So, algorithm AHY utilizes the concept developed for simple queries to solve general queries. For these general queries, algorithm AHY ordered the relations according to their cardinalities. Starting from the smallest, a relation is sent serially to the next smallest one in order to perform semijoin. At starting step, transmission cost is guaranteed to be minimal. Since, at each step a minimal size relation is created which in turns guarantees minimal transmission cost at next step. Thus, at each step only transmission cost is guaranteed to be minimal. Algorithm AHY utilizes the concept of relation schedule, i.e., a sequence of semijoin to be executed in linear order to that relation. The construction of relation schedule is based on simple query tactics. This is the main reason why this schedule results in minimum transmission cost. Although, a problem associated with this tactics is that relation schedules are independent. That is, if R2 is reduced in cardinality in R1's schedule, information of this reduction in cardinality is not utilized while forming a schedule of R2's relation. In fact, these independencies of relations schedule make it worse than algorithm SDD-1 and algorithm LIGHT (covered in next section).

Before introducing the algorithm AHY, terminology used in the algorithm will be explained next.

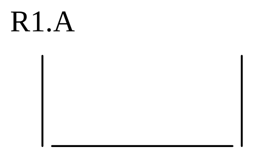
For each attribute A in the query we can define its 'simple query solution' in the following way. For those relations, say {R1, R2,….., Rm} with an attribute A, ordered the relations according to their cardinality of an attribute A. It is assumed that all relations are stored at different nodes in the network.

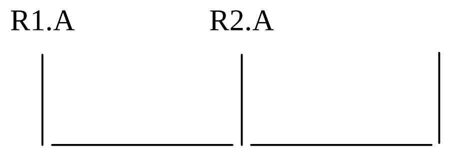
For instance, if ordering of relations happened to be R1, R2, ….., Rm, then the simple query schedule is

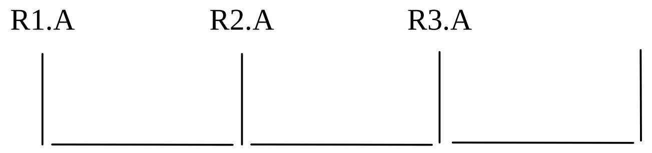


The schedule is executed from left to right. The semijoins are R2<A=A]R1, R3<A=A]R2, ………, Rm<A=A]Rm-1. The last transmission is to result node. R1.A has the smallest cardinality among all other relations on an attribute A and the above solution is the optimal solution if this is a simple query. In the above semijoin attribute A in R2 gets reduced in cardinality by the semijoin R2<A=A]R1. The reduced R2.A reduces R3, and so on.

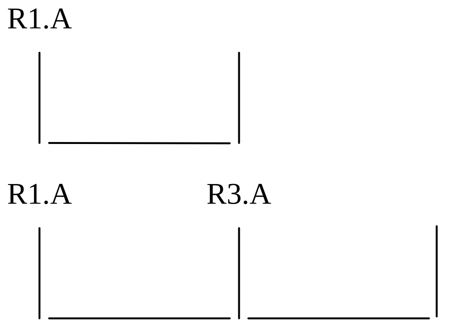
For each attribute, CASE 1 serial schedules are formed. For example, let there be three relations R1, R2 and R3, each at the different node in a network and let the cardinality of joining attribute be R1.A<R2.A<R3.A. Then, the CASE 1 serial schedules are defined as follows:







If we are trying to determine the relation schedule to relation Ri in algorithm AHY, we first need to form a CASE 2 serial schedules for relation Ri. For example the CASE 2 serial schedules for relation R2 from the above example will be:



This is done by eliminating R2.A's transmission from the CASE 1 serial schedules and eliminating duplicate schedules. Algorithm AHY then utilizes the CASE 2 serial schedule in forming relation schedules.

We now present algorithm AHY [2].

Algorithm AHY

1. Generate candidate relation schedules: Isolate each of the joining attributes and consider each to define a simple query with an undefined result node. Form CASE 1 serial schedule.
2. Select the best candidate schedule: For each relation, form the CASE 2 serial schedules for each attribute. Each schedule in the set of CASE 2 serial schedules is considered to be a 'candidate schedule' to the relation for that attribute. If the cost of a candidate schedule plus the final transmission to the result node has less cost than the initial feasible solution for that relation and is the least cost candidate schedule then save that candidate schedule for that attribute for that relation.
3. Integrate the schedules: If only one schedule has been saved for relation Ri, then this is the schedule to relation Ri. Otherwise the saved schedules need integration. This is done by Procedure TOTAL given below.
4. Remove schedule redundancies: Eliminate schedules for relations which have been transmitted in the schedule for another relation.

Procedure TOTAL

1. Candidate schedule ordering: For each relation, say Ri, order the saved schedules in increasing order of total cost (i.e. the cost of the schedule plus the cost of the transmission of Ri to the result node).
2. If S1, S2, ……, Sn are the saved schedules to relation Ri, in order of increasing total cost, then form the integrated schedules SI1, SI2, ……., SIn, which consist of the parallel transmission of the saved schedules to relation Ri, such that S1 is the only schedule in SI1, S1 and S2 are the only schedules in SI2. In general, Si ($i \leq j$) are the only schedules in SIj. Select the integrated schedule SIj that results in the minimal total time value.

The time complexity of this algorithm in the worst case is $O(nm^2)$, where n is the number of attributes and m is the number of relations in the query.

7.4 Algorithm SDD-1

Algorithm SDD-1 presented by Bernstein et al. [7] is a greedy optimization algorithm. Algorithm SDD-1 chooses the semijoins to include in semijoin program (SJP) is always the least cost semijoins as determined by the SDD-1 model of a distributed database system. There are only two requirements for a semijoin to be included in SJP. The first is that it must have the least cost in terms of amount of data transmitted and the second is that the cost must be less than benefit (i.e., benefit>cost), which is quite reasonable. The benefit is the reduction in relation size before applying the semijoin and after applying it. Clearly with these conditions in place, the semijoin program produced will always lead to a preprocessing strategy with total cost (the cost to execute the semijoins plus the final transmission to the result node) less than or equal to initial feasible solution. In one of their two papers, they have changed the algorithm to select the semijoin at each step having the highest (benefit - cost) which they called it algorithm OPT. The cost and benefit of the remaining semijoin which are affected by the addition of this semijoin to SJP is updated according to the cost/benefit estimation as presented in chapter 6.

Algorithm SDD-1 considers only those semijoins for inclusion in SJP which are implied by transitive closure of the join clauses expressed in the initial query. A semijoin once included in SJP is not considered again for possible addition to SJP.

Now we present algorithm SDD-1 which paper [7] refers as algorithm OPT.

Algorithm SDD-1

Input: Relation taking part in the query and database statistics.

Output: Semijoin program (SJP) and commands to move reduced relations to result node.

```
1. STEP 1: Initialization
   1.1. Perform local reduction permitted by query.
   1.2. Estimate the cost and benefit of all non-local semijoins permitted by the
        query.
2. STEP 2: Main Loop
   2.1. Do while some non-local semijoin permitted by query has benefit>cost.
   2.2. Select the most profitable semijoin and add to SJP and marked as selected.
   2.3. Estimate the effect of this addition of semijoin to SJP and update the cost and
        benefit accordingly.
   2.4. end
3. STEP 3: Termination
   3.1. The reduced relations are now transmitted to the result node. We assume that
        the result node is none of the relations referenced in a join clause in the query
        and all reduced relations are transmitted to this node.
```

Algorithm SDD-1 starts with performing local reductions using selections and projections permitted by the initial query. Semijoin within the same site can also be executed to reduce the size of the relations. Then all the semijoins across sites are identified which is permitted by the query. The cost and benefit of each semijoin is then calculated. Then the iteration process starts with choosing the most profitable semijoin. This most profitable semijoin is then added to semijoin program (SJP) and marked as selected and the remaining semijoins affected by this inclusion in SJP are updated. The reason for marking the semijoin added to SJP is that in the coming iteration it is not considered to be added to SJP any further. The iteration process repeats until no profitable semijoin is found. Algorithm then terminates by sending all these reduced relations to the result node to perform join in order to obtain the answer of imposed query. Total cost can be calculated from this algorithm by adding the cost of each semijoin that is in SJP and transmission of the reduced relations taking part in the query except for those relations which are in result node. The time complexity of this algorithm in the worst case is $O(nm^2)$, where n is the number of attributes and m is the number of relations in the query.

The semijoin program generated by above algorithm can be used to re-calculate the total cost in which simultaneous transmission is allowed. For this, a partial order graph is generated from the semijoin program which exposes the semijoins that can be carried simultaneously. Out of the semijoins which can be carried simultaneously, it is enough to add only the cost of semijoin having the highest cost. For example, if a semijoin program is somewhat like this R1<A=A]R2, R3<A=A]R2 then it is enough to add only the cost of one semijoin with highest cost in order to calculate total time (i.e. cost to add to calculate total cost = max{cost R1<A=A]R2, cost R3<A=A]R2}).

7.5 Algorithm LIGHT

Algorithm LIGHT is similar in style to algorithm SDD-1; however there is one major difference in terms of adding a semijoin in a semijoin program. Algorithm SDD-1 overlooked the principle of basic mathematics. Algorithm SDD-1 includes most beneficial semijoin at each step. Basic mathematics which is always in favor of taking the average of values is never considered in algorithm SDD-1. Algorithm SDD-1 only takes into account the semijoin that is most beneficial at each step and never looks ahead of time. In other words, algorithm SDD-1 is not futuristic in the sense that it never cares about consequences of adding semijoin in semijoin program. This lack of future vision is tried to be exploited in algorithm LIGHT.

A new heuristic based on principle of basic mathematics is utilized in algorithm LIGHT. Algorithm LIGHT includes highest average percentage gain in reduction of joining attributes cardinalities semijoin among the beneficial semijoins at each step. Here joining attributes refer to all those attributes of a relation going to be reduced which can form a bond with any relations taking part to answer the imposed query. The condition for a semijoin to be included in a semijoin program as in SDD-1 is still valid in the sense that only beneficial semijoin average percentage gain in reduction of joining attributes cardinalities is calculated. This heuristic is based on the notion that, adding the highest average percentage gain in reduction of joining attributes cardinalities among the beneficial semijoins have greater impact in reducing the cardinality of a relation (thus size) in coming iteration than merely adding the greatest beneficial semijoin. Gain in reduction of non-joining attributes cardinalities is not considered, mainly because it is unlikely to have any impact (effect) on future semijoins to be added to semijoin program (if any).

Now we present algorithm LIGHT.

<u>Algorithm LIGHT</u>

Input: Relation taking part in the query and database statistics.

Output: Semijoin program (SJP) and commands to move reduced relations to result node.

1. STEP 1: Initialization
    1.1. Perform local reduction permitted by query.
    1.2. Estimate the cost and benefit of all non-local semijoins permitted by the query.
2. STEP 2: Main Loop
    2.1. Do while some non-local semijoin permitted by query has benefit>cost.
    2.2. Out of the profitable semijoin, add that semijoin to the SJP that has the highest average percentage gain in reduction of joining attributes cardinalities and marked as selected.
    2.3. Estimate the effect of this addition of semijoin to SJP and update the cost and benefit accordingly.
    2.4. end
3. STEP 3: Termination
    3.1. The reduced relations are now transmitted to the result node. We assume that the result node is none of the relations referenced in a join clause in the query and all reduced relations are transmitted to this node.

Algorithm LIGHT performs almost identical as algorithm SDD-1 except for the step 2.2. Algorithm SDD-1 adds that semijoin to SJP which is most profitable whereas algorithm LIGHT differs slightly to add that semijoin to the SJP that has the highest average gain in reduction of joining attributes cardinalities. The time complexity of this algorithm is same as algorithm SDD-1 which in the worst case is $O(nm^2)$, where n is the number of attributes and m is the number of relations in the query. The reason for naming this algorithm LIGHT is mainly because this work tried to put some light in algorithm SDD-1.

7.6 Algorithm Luk

Algorithm Luk is presented by Luk and Luk in 1980. Algorithm Luk is an algorithm that seeks to improve a semijoin program produced by some arbitrary heuristic. The program is transformed into one with no-increasing cost and non-decreasing benefit. Let SJP and SJP' be a semijoin programs input to and output from the algorithm Luk respectively. Then SJP' satisfies the following conditions:

1. SJP' and SJP give the same answer to any given query.
2. SJP' has a cost no greater than SJP for all instances.
3. SJP' has a benefit not less than SJP for all instances.
4. SJP' is produced given only the SJP; no additional information such as cardinalities of the attributes and relations is necessary.

Furthermore, it has been proved that the improvement of SJP' over SJP is optimal in the sense that there will not be another SJP'' which is better than SJP' and still satisfies conditions 1 to 4. The worst case time complexity of this algorithm is $O(m^3)$, where m is the number of relations in a query.

7.7 An Example

An example is now presented to illustrate the results produced by algorithms AHY, SDD-1 and LIGHT. Let the initial database state is as given in Table 7.1.

Relation	Relation Size	Joining Attribute Cardinalities		
		P	S	A
R1	1000	400	100	
R2	2000	400	450	100
R3	3000	900		300

Table 7.1 Initial Database Statistics

Let the domain cardinalities of P, S, A be 1000, 500 and 300 respectively. Let the relations R1, R2 and R3 are at different nodes in the network. Let the where part of the SQL query contains the following conditions:

R1.P=R2.P

AND

R1.P=R3.P

AND

R1.S=R2.S

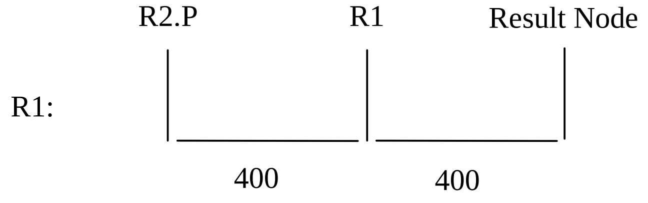
AND

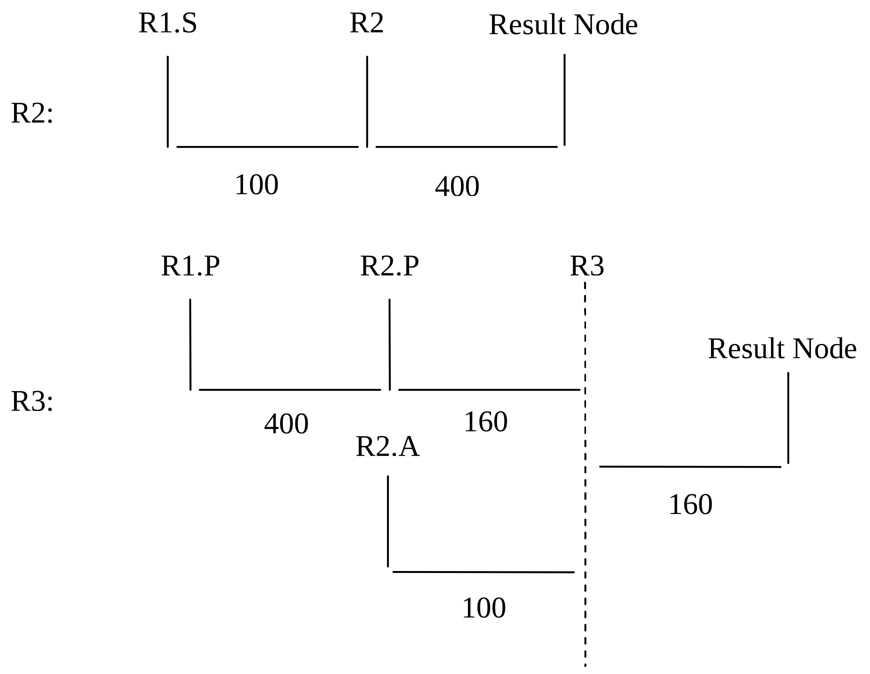
R2.A=R3.A

The cost in data communication for the initial feasible solution (IFS) for this query would be, 1000 + 2000 + 3000 = 6000 assuming that the C0=0 and C1=1 and the result is expected in another node in the network rather than the node containing R1, R2 and R3. The results presented below may be compared to the IFS cost to see how well the semijoin preprocessing strategies perform.

7.7.1 Algorithm AHY Result

Algorithm AHY when given this query produces the following relation schedules.





The total cost of these schedules is 400 + 400 + 100 + 400 + 400 + 160 + 100 + 160 = 2120.

7.7.2 Algorithm SDD-1 Result

Algorithm SDD-1 when given this query produces the following semijoin program.

1. R3<A=A]R2
2. R2<S=S]R1
3. R1<S=S]R2
4. R3<P=P]R1
5. R2<P=P]R3
6. R3<P=P]R2
7. R1<P=P]R3
8. R2<P=P]R1

The total cost of these semijoins plus the transmission of all reduced relations to result node is 634.

7.7.3 Algorithm LIGHT Result

Algorithm LIGHT when given this query produces the following semijoin program.

1. R3<A=A]R2
2. R2<S=S]R1
3. R1<S=S]R2
4. R2<P=P]R1
5. R3<P=P]R2
6. R2<P=P]R3
7. R1<P=P]R2
8. R3<P=P]R1
9. R2<A=A]R3

The total cost of these semijoins plus the transmission of all reduced relations to result node is 558. For the stepwise execution of these algorithms refer Appendix A.

CHAPTER 8
IMPLEMENTATION AND ANALYSIS

8.1 Implementation

A simulation program to test the performance of the algorithm SDD-1 and algorithm LIGHT introduced in the previous chapter has been developed. The program has been developed using Ruby on Rails. Ruby is a powerful and beautiful programming language; Rails is a web framework built on top of it. The reason for implementing it on Ruby on Rails is to make it platform independent and easily accessible via network.

This program accepts as input the number of relations taking part to answer the query and database description (i.e., relation cardinality, attributes cardinalities) for each relation. The program outputs for each algorithm the average total communication cost per query and the semijoin program with total execution trace (if any). Average communication cost includes the cost of executing the semijoin program and the final transmission of relations to the result node. The cost in data communication on the network of the semijoin program is the sum of the costs for each semijoin in the semijoin program. The cost of the final transmission of relations to the result node is the sum of the relation sizes after the semijoin program has been executed on the database.

It is assumed that each site can hold no more than one relation at a time. So, the number of inputted relations actually signifies the number of sites taking part to answer the generated query. It is also assumed that there is only one joining attribute between any two relations. So, whenever these two relations get involved to generate the query, the joining condition remains same. For simplicity, the start up cost of initiating transmission (C_0) is considered to be 0 (zero) and proportionality constant (C_1) to be 1 (i.e., cost of transmitting fixed amount of data between any two sites is equal).

Conceptually, the program starts with taking the basic information (i.e., number of relations, relation cardinality, attributes cardinalities of each relation) inputted by user and form a query based on these inputs. Then, the program generates the possible list of semijoins that have to be considered. Initially, the ordered list of profitable semijoin i.e., a semijoin program; say it omega-profitable, is empty. At each step, program calculate the required information like cost of each semijoin, benefit of each semijoin, benefit minus cost of each semijoin etc. in order to determine whether any semijoin is to be added to the omega-profitable. If the STEP 2 of algorithm SDD-1 (similar for algorithm LIGHT) detects a semijoin that satisfies the required condition, it is added to omega-profitable, marked it as selected and do the necessary database statistics update operation on reduced relation as stated in chapter 6. When no more beneficial semijoin is found, main loop terminates and the program outputs the semijoin program with total execution trace (if any) and average total communication cost incurred in order to answer the generated query. Our implementation consists of two class controllers namely: Statistic and Algorithm which encapsulate the detail of database statistics inputted and discussed algorithms respectively. Class Algorithm controller has three main methods to evaluate each strategy. Methods ifs, sdd-1 and light of class Algorithm controller, evaluates the initial feasible solution (IFS), algorithm SDD-1 and algorithm LIGHT respectively. Each of these methods respective view files output the results in the browser.

Figure 8.1, 8.2 and 8.3 shows the sample output of a program execution (just an instance) for IFS, algorithm SDD-1 and algorithm LIGHT respectively. All these outputs are generated under identical conditions i.e., same number of relations and same initial database statistics (to be specific when number of relation equals to 3 and average selectivity factor is 0.05).

```
                ===========
                    IFS
                ===========
   Average total communication cost is 705000.
```

Figure 8.1 Sample Output of IFS

===========================

Algorithm SDD-1

===========================

Iteration 1

Semijoin	Cost	Relation Reduced To	Benefit	Benefit-Cost
R1<A3=A3]R2	850	1215	146355	145505
R2<A3=A3]R1	900	900	102300	101400
R2<A5=A5]R3	1800	1400	100800	99000
R3<A5=A5]R2	400	1334	445998	445598

Iteration 2

Semijoin	Cost	Relation Reduced To	Benefit	Benefit-Cost
R1<A3=A3]R2	850	1215	146355	145505
R2<A3=A3]R1	900	900	102300	101400
R2<A5=A5]R3	16	13	104961	104945
R3<A5=A5]R2	Not Considered			

Iteration 3

Semijoin	Cost	Relation Reduced To	Benefit	Benefit-Cost
R1<A3=A3]R2	Not Considered			
R2<A3=A3]R1	22	22	104934	104912
R2<A5=A5]R3	16	13	104961	104945
R3<A5=A5]R2	Not Considered			

Iteration 4

Semijoin	Cost	Relation Reduced To	Benefit	Benefit-Cost
R1<A3=A3]R2	Not Considered			
R2<A3=A3]R1	22	1	36	14
R2<A5=A5]R3	Not Considered			
R3<A5=A5]R2	Not Considered			

Iteration 5

Semijoin	Cost	Relation Reduced To	Benefit	Benefit-Cost
R1<A3=A3]R2	Not Considered			
R2<A3=A3]R1	Not Considered			
R2<A5=A5]R3	Not Considered			
R3<A5=A5]R2	Not Considered			

```
Semijoin Program is
===============
1.R3<A5=A5]R2
2.R1<A3=A3]R2
3.R2<A5=A5]R3
4.R2<A3=A3]R1

Average total communication cost is 8938.
```

Figure 8.2 Sample Output of Algorithm SDD-1

```
===========================
      Algorithm LIGHT
===========================
```

<u>Iteration 1</u>

Semijoin	Cost	Relation Reduced To	Benefit	Benefit-Cost	Avg. % gain in joining attributes cardinalities
R1<A3=A3]R2	850	1215	146355	145505	97.5555555555556
R2<A3=A3]R1	900	900	102300	101400	48.7058823529412
R2<A5=A5]R3	1800	1400	100800	99000	48.0
R3<A5=A5]R2	400	1334	445998	445598	99.1111111111111

<u>Iteration 2</u>

Semijoin	Cost	Relation Reduced To	Benefit	Benefit-Cost	Avg. % gain in joining attributes cardinalities
R1<A3=A3]R2	850	1215	146355	145505	97.5555555555556
R2<A3=A3]R1	900	900	102300	101400	48.7058823529412
R2<A5=A5]R3	16	13	104961	104945	98.25
R3<A5=A5]R2	Not Considered				

<u>Iteration 3</u>

Semijoin	Cost	Relation Reduced To	Benefit	Benefit-Cost	Avg. % gain in joining attributes cardinalities
R1<A3=A3]R2	13	19	149943	149930	99.8888888888889
R2<A3=A3]R1	900	1	36	Negative	
R2<A5=A5]R3	Not Considered				
R3<A5=A5]R2	Not Considered				

Iteration 4

Semijoin	Cost	Relation Reduced To	Benefit	Benefit-Cost	Avg. % gain in joining attributes cardinalities
R1<A3=A3]R2	Not Considered				
R2<A3=A3]R1	1	1	36	35	46.1538461538462
R2<A5=A5]R3	Not Considered				
R3<A5=A5]R2	Not Considered				

Iteration 5

Semijoin	Cost	Relation Reduced To	Benefit	Benefit-Cost	Avg. % gain in joining attributes cardinalities
R1<A3=A3]R2	Not Considered				
R2<A3=A3]R1	Not Considered				
R2<A5=A5]R3	Not Considered				
R3<A5=A5]R2	Not Considered				

Semijoin Program is

===============

1.R3<A5=A5]R2

2.R2<A5=A5]R3

3.R1<A3=A3]R2

4.R2<A3=A3]R1

Average total communication cost is 4492.

Figure 8.3 Sample Output of Algorithm LIGHT

8.2 Analysis

The interest of this work is to find how each algorithm performs in a general query environment. To be able to compare the performance of each algorithm it is necessary to average total communication cost for each query for each algorithm over a large number of queries. To have some feeling of the range that possible results could have, the number of join clauses in the queries is varied. This variation occurred in the range of 2 to 6. Similarly, for each number of relations taking part in the query the average selectivity factor is varied from 0.05, 0.1, 0.2, ……….., 0.5 to explore the impact of this variation on implemented algorithms. For the simulation purpose the cardinality of relations varied from 35k to 200k and domain cardinalities from 9k to 100k (k represents thousand).

We have considered IFS as the worst case strategy and used it to compare, how well other algorithms preformed with respect to it. Clearly, IFS does not play significant role as a query processing strategy when a good semijoin preprocessing strategies are available. We only conducted small scale simulation and large scale simulation is left as future work.

8.2.1 Cost Comparison of IFS to Semijoin Preprocessing Strategies

For each query, the cost in data communication of the IFS is the sum of relation sizes involved in the query. Algorithm SDD-1 and algorithm LIGHT only produces a semijoin program if it can improve upon the cost of IFS. When IFS cost (IFS_COST) for a query is compared to the average cost of query preprocessing strategy (AVG_COST) by algorithm SDD-1, the result shown in Table 8.1 is obtained.

Number of Relations	IFS_COST/AVG_COST
2	11.55
3	10.74
4	17.54
5	13.56
6	25.03

Table 8.1 Ratio of IFS_COST/AVG_COST for Different Number of Relations for Algorithm SDD-1

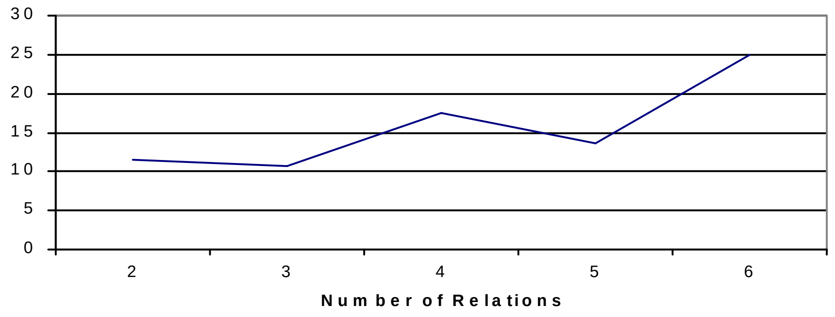


Figure 8.4 IFS_COST/AVG_COST versus No. of Relations

When Table 8.1 is plotted, line diagram as shown in the Figure 8.4 is obtained. In Figure 8.4 as the number of relations increases the ratio of the IFS_COST to AVG_COST increases except for where the number of relations is 3 and 5. We believe that this fluctuation in values will get remedies in large scale simulation. The ratio of IFS_COST to AVG_COST of algorithm SDD-1 is never below 10.74:1.

Similarly, when IFS cost (IFS_COST) for a query is compared to the average cost (AVG_COST) of query processing strategy by algorithm LIGHT, the results shown in Table 8.2 is obtained.

Number of Relations	IFS_COST/AVG_COST
2	11.55
3	17.70
4	21.12
5	18.78
6	26.99

Table 8.2 Ratio of IFS_COST/AVG_COST for Different Number of Relations for Algorithm LIGHT

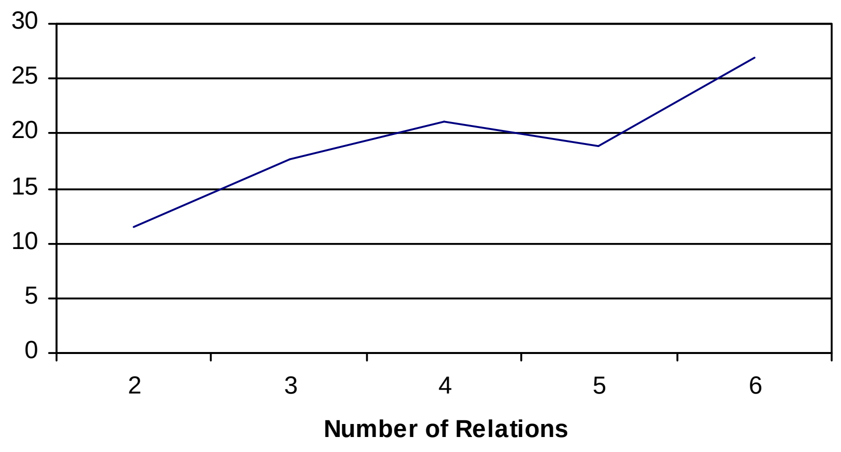


Figure 8.5 IFS_COST/AVG_COST versus No. of Relations

When Table 8.2 is plotted, line diagram as shown in the Figure 8.5 is obtained. In Figure 8.5 as the number of relations increases the ratio of the IFS_COST to

AVG_COST increases except for where the number of relations is 5 in which a slight downfall is noticed. The line in Figure 8.5 are smoother than those for the Figure 8.4, which signifies that there is more smooth increase in IFS_COST/AVG_COST when number of relation increases in algorithm LIGHT than algorithm SDD-1. The ratio of IFS_COST to AVG_COST of algorithm LIGHT is never below 11.55:1.

Selectivity Factor (S.F.)	IFS_COST/AVG_COST	
	For Algorithm SDD-1	For Algorithm LIGHT
0.05	29.4	117.3
0.1	52.58	67.73
0.2	29.88	44.70
0.3	18.13	22.63
0.4	12.79	16.70
0.5	11.85	13.24

Table 8.3 IFS_COST/AVG_COST versus Average Selectivity

Table 8.3 presents the data obtained when IFS_COST/AVG_COST versus average selectivity is calculated for both the algorithms and Figure 8.6 is the line diagram when data of Table 8.3 is plotted.

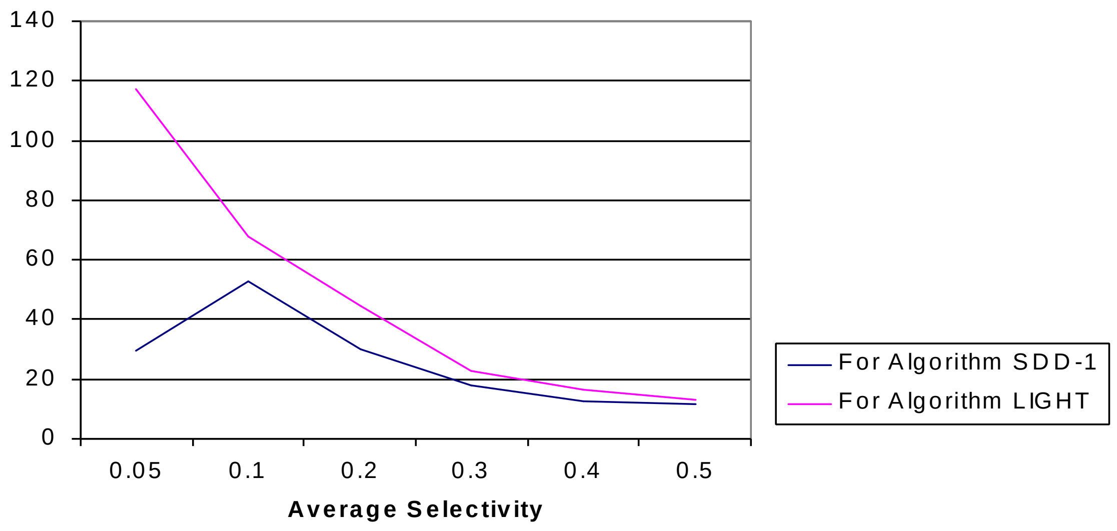


Figure 8.6 IFS_COST/AVG_COST versus Average Selectivity

Figure 8.6 supports the notion that one can have maximum benefit when the selectivity factor is as close to zero as possible. As the average selectivity factor increases the ratio of IFS_COST to AVG_COST decreases smoothly for algorithm LIGHT but for algorithm SDD-1 on can see the fluctuation when average selectivity is 0.1. From the Figure 8.6 one can also conclude that the gap (i.e., benefit obtained) between algorithm SDD-1 and algorithm LIGHT decreases as the average selectivity increases and became almost identical when average selectivity is 0.5.

8.2.2 Cost Comparison of Algorithm SDD-1 and Algorithm LIGHT

Algorithm SDD-1 and algorithm LIGHT performs almost identical in our case when the number of joining attribute is 2, no matter whatever be the selectivity. As the number of relations (and hence sites) taking part to answer the query increases, algorithm LIGHT starts to show its effect over algorithm SDD-1. Our simulation have found algorithm SDD-1 to performs well than algorithm LIGHT 7.7% of the times whereas algorithm LIGHT performs better than algorithm SDD-1 72.3% of the times. Algorithm SDD-1 and algorithm LIGHT performs identical 20% of the times. Algorithm SDD-1 only generates fewer numbers of semijoins in a semijoin program 2.2% of the times than algorithm LIGHT. So, the advantage obtained from algorithm LIGHT is not taken away by the preprocessing time required to formulate the schedule. It is believed that the fluctuation in above line diagram is mainly due to the small scale simulation and will be removed when large scale simulation is carried out.

CHAPTER 9
CONCLUSION

9.1 Conclusion

The shift in paradigm from centralized to its counter part distributed database systems have brought enormous opportunities and problems along with it. In this present world of globalization, distributed database system is no more an obsession but a need of almost every organization to remain competitive and more importantly to survive in this competition. Because of critical performance issue, query processing has always been the centre of attraction. This study also addressed query processing in distributed database systems.

This study is mainly focused on the semijoin query preprocessing strategies for distributed database systems which utilized the database statistics to generate the semijoin program. The study has presented an overview of an existing semijoin preprocessing algorithms and their downfall.

The parallel independent relation schedules that algorithm AHY produces are the main reasons for its failure to perform worse when compared to other two algorithms (SDD-1 and LIGHT).

Algorithm SDD-1, includes the most profitable semijoin in semijoin program at each step and never cares about the consequences of adding it on the impact of future semijoins to be added to the semijoin program (if any). This is believed to be the root cause for the failure of algorithm SDD-1 to perform worse than algorithm LIGHT.

Algorithm LIGHT exploits what algorithm SDD-1 lacks by including that semijoin to the semijoin program whose average percentage gain in reduction of joining attributes cardinalities is highest among the profitable semijoin at each step. Our simulation has showed that algorithm LIGHT is significantly better than algorithm SDD-1 in producing a good semijoin preprocessing strategies. However, this does not minimize the importance of algorithm SDD-1 as it forms the basis for algorithm LIGHT.

One question concerning this work that has not been answered is, how close are our results to optimal solution? Even if the results presented here are within the factor of 2 to the optimal solution, it can be considered as good solution because an optimal solution algorithm will necessarily be exponential (if any).

9.2 Future Direction

Only small scale simulation of algorithm SDD-1 and algorithm LIGHT has been conducted. The results obtained are really promising. However, validity of this work on large scale simulation is left as a future work.

No doubt, any good distributed query processing strategy always include both join and semijoin strategies. In this regard, this study only considered semijoin strategies. Graceful integration of join and semijoin in distributed query processing is still an open research problem. Lately, few researchers have initiated a work on it; but still a lot more is to be done. Query processing in World Wide Web (WWW) is probably the most interesting, challenging and future work in this field.

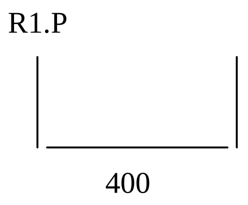
APPENDIX A

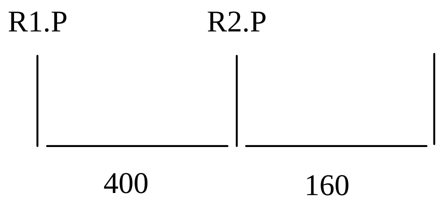
A.1 Stepwise Execution of 7.7 Example Using Algorithm AHY

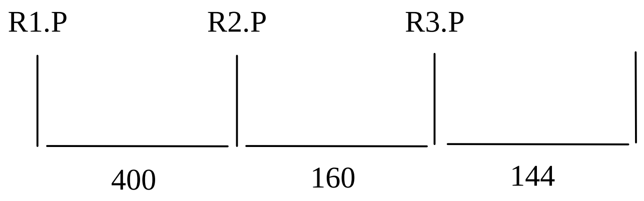
Algorithm AHY proceeds as given below:

Step 1: Generating candidate relation schedules

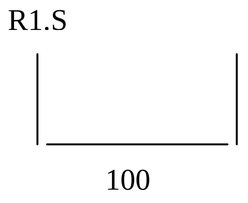
For attribute P

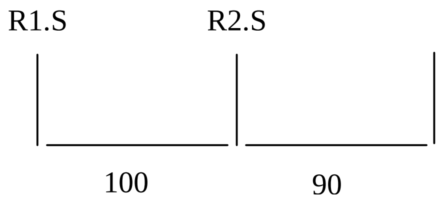




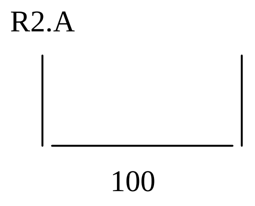


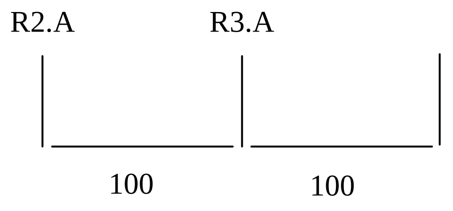
For attribute S





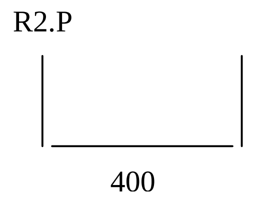
For attribute A

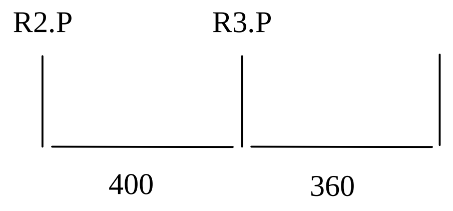




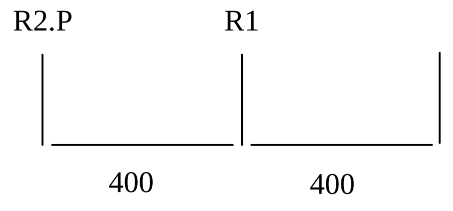
Step 2: Selecting the best candidate schedule

For attribute P of relation R1

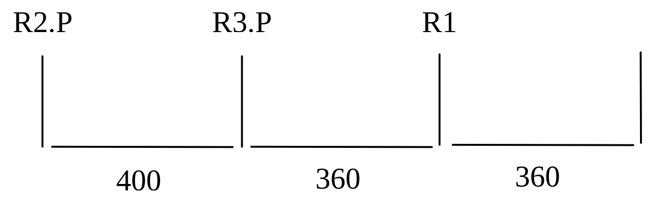




Total cost calculation of candidate schedules for relation R1 on attribute P



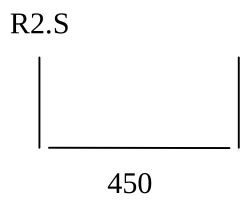
Total Cost = 400 + 400 = 800 (Saved)



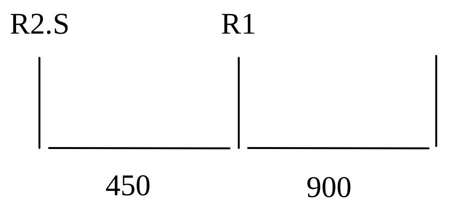
Total Cost = 400 + 360 + 360 = 1120 (Rejected)

Where ‘Saved’ means that candidate schedule is saved for that relation for that attribute because cost of a candidate schedule plus the final transmission to the result node has less cost than the initial feasible solution for that relation and is the least cost candidate schedule. Similarly, ‘Rejected’ means that candidate schedule is not even considered for the save because cost of a candidate schedule plus the final transmission to the result node is greater than the initial feasible solution for that relation. From here on to end of this appendix ‘Saved’ and ‘Rejected’ carry the same meaning as described above.

<u>For attribute S of relation R1</u>

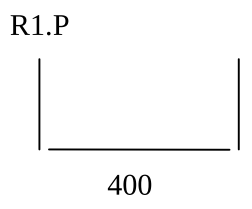


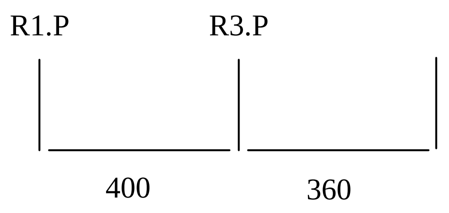
<u>Total cost calculation of candidate schedules for relation R1 on attribute S</u>



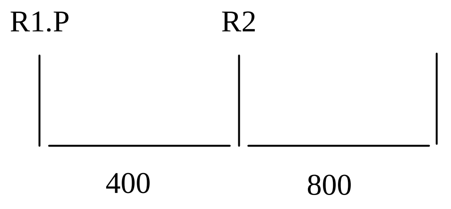
Total Cost = 450 + 900 = 1350 (Rejected)

<u>For attribute P of relation R2</u>

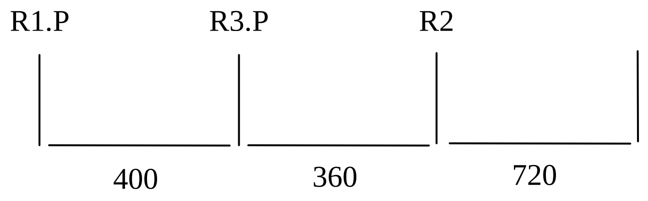




Total cost calculation of candidate schedules for relation R2 on attribute P

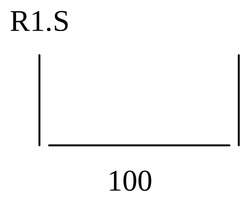


Total Cost = 400 + 800 = 1200 (Saved)

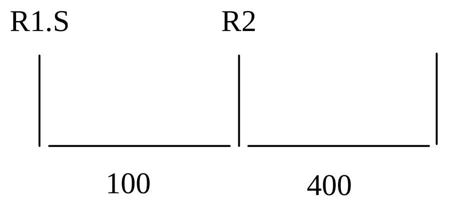


Total Cost = 400 + 360 + 720 = 1460

For attribute S of relation R2

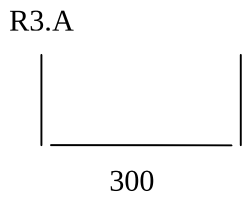


Total cost calculation of candidate schedules for relation R2 on attribute S

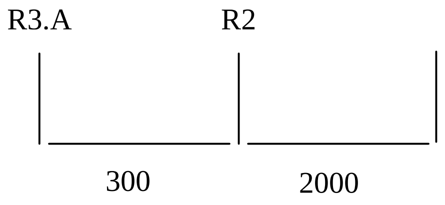


Total Cost = 100 + 400 = 500 (Saved)

For attribute A of relation R2

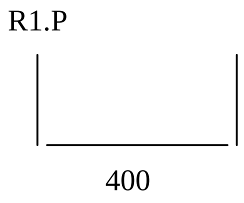


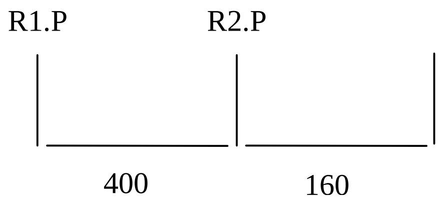
Total cost calculation of candidate schedules for relation R2 on attribute A



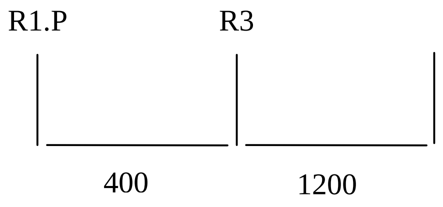
Total Cost = 300 + 2000 = 2300 (Rejected)

For attribute P of relation R3

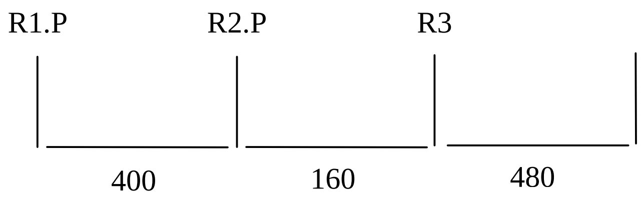




Total cost calculation of candidate schedules for relation R3 on attribute P

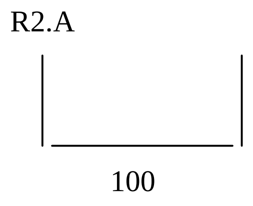


Total Cost = 400 + 1200 = 1600

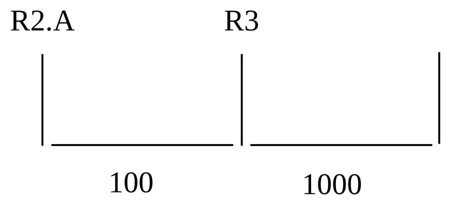


Total Cost = 400 + 160 + 480 = 1040 (Saved)

For attribute A of relation R3



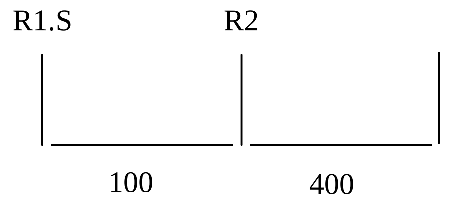
Total cost calculation of candidate schedules for relation R3 on attribute A



Total Cost = 100 + 1000 = 1100 (Saved)

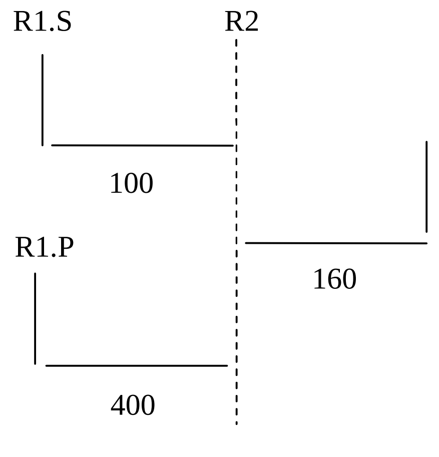
Applying Procedure Total for saved schedules of R2

SI1 of R2



Total Cost = 100 + 400 = 500 (Selected)

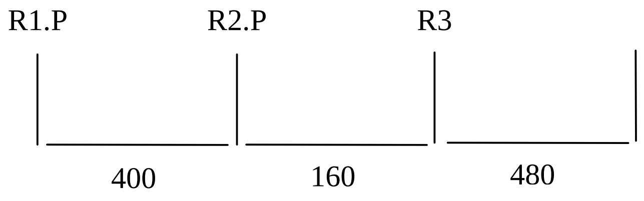
SI2 of R2



Total Cost = 100 + 400 + 160 = 660

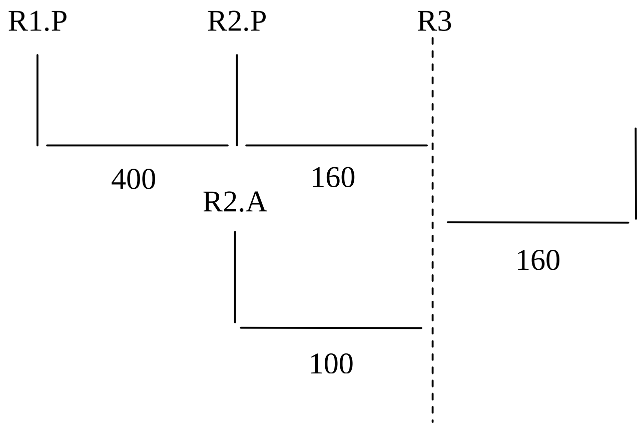
Applying Procedure Total for saved schedules of R3

SI1 of R3



Total Cost = 400 + 160 + 480 = 1040

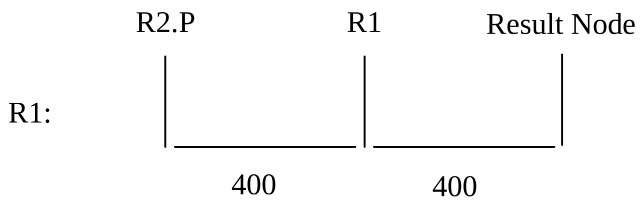
SI2 of R3

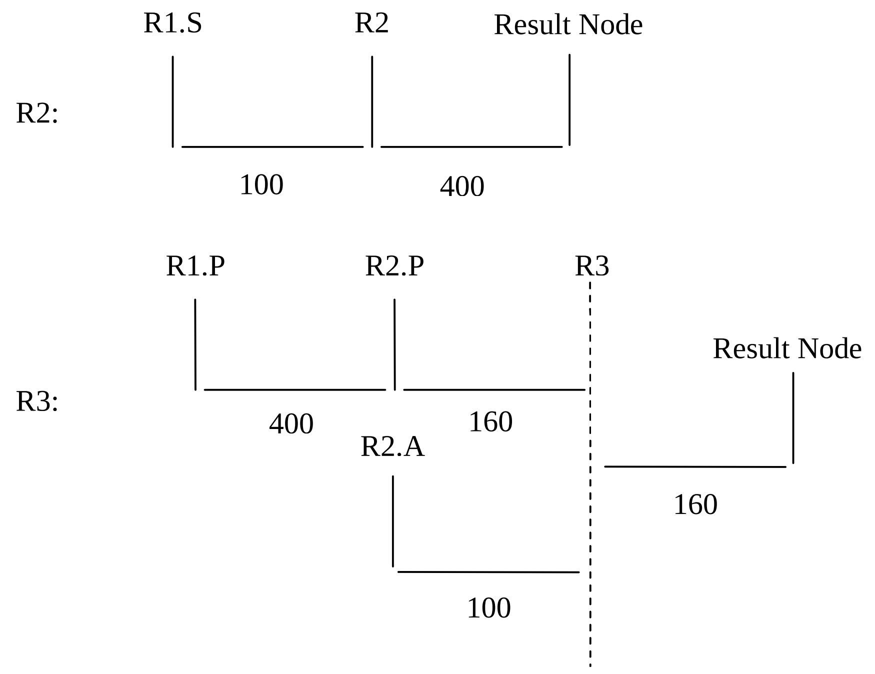


Total Cost = 400 + 160 + 100 + 160 = 820 (Selected)

Step 4: Eliminate the schedule which is not marked as selected for the relation R2 and R3 (i.e., SI2 of R2 and SI1 of R3).

So the final relational schedules produced by AHY is





The total cost of these schedules is 400 + 400 + 100 + 400 + 400 + 160 + 100 + 160 = 2120.

A.2 Stepwise Execution of Example 7.7 Using Algorithm SDD-1

Iteration 1

Semijoin	Cost	Relation Reduced To	Benefit	Benefit - Cost
R1<P=P]R2	400	400	600	200
R2<P=P]R1	400	800	1200	800
R1<P=P]R3	900	900	100	Negative
R3<P=P]R1	400	1200	1800	1400
R2<P=P]R3	900	1800	200	Negative
R3<P=P]R2	400	1200	1800	1400
R1<S=S]R2	450	900	100	Negative
R2<S=S]R1	100	400	1600	1500
R2<A=A]R3	300	2000	0	Negative
R3<A=A]R2	100	1000	2000	1900

Semijoin R3<A=A]R2 has the highest (Benefit - Cost) so added to SJP.

Iteration 2

Semijoin	Cost	Relation Reduced To	Benefit	Benefit - Cost
R1<P=P]R2	400	400	600	200
R2<P=P]R1	400	800	1200	800
R1<P=P]R3	634	634	366	Negative
R3<P=P]R1	400	400	600	200
R2<P=P]R3	634	1268	732	98
R3<P=P]R2	400	400	600	200
R1<S=S]R2	450	900	100	Negative
R2<S=S]R1	100	400	1600	1500
R2<A=A]R3	100	667	1333	1233
R3<A=A]R2		Not Considered		

Semijoin R2<S=S]R1 has the highest (Benefit - Cost) so added to SJP.

Iteration 3

Semijoin	Cost	Relation Reduced To	Benefit	Benefit - Cost
R1<P=P]R2	267	267	733	466
R2<P=P]R1	400	160	240	Negative
R1<P=P]R3	634	634	366	Negative
R3<P=P]R1	400	400	600	200
R2<P=P]R3	634	254	146	Negative
R3<P=P]R2	267	267	733	466
R1<S=S]R2	90	180	820	730
R2<S=S]R1		Not Considered		
R2<A=A]R3	100	134	266	166
R3<A=A]R2		Not Considered		

Semijoin R1<S=S]R2 has the highest (Benefit - Cost) so added to SJP.

Iteration 4

Semijoin	Cost	Relation Reduced To	Benefit	Benefit - Cost
R1<P=P]R2	267	49	131	Negative
R2<P=P]R1	180	72	328	148
R1<P=P]R3	634	115	65	Negative
R3<P=P]R1	180	180	820	720
R2<P=P]R3	634	254	146	Negative
R3<P=P]R2	267	267	733	466
R1<S=S]R2		Not Considered		
R2<S=S]R1		Not Considered		
R2<A=A]R3	100	134	266	166
R3<A=A]R2		Not Considered		

Semijoin R3<P=P]R1 has the highest (Benefit - Cost) so added to SJP.

Iteration 5

Semijoin	Cost	Relation Reduced To	Benefit	Benefit - Cost
R1<P=P]R2	267	49	131	Negative
R2<P=P]R1	180	72	328	148
R1<P=P]R3	115	21	159	44
R3<P=P]R1		Not Considered		
R2<P=P]R3	115	46	354	239
R3<P=P]R2	267	49	131	Negative
R1<S=S]R2		Not Considered		
R2<S=S]R1		Not Considered		
R2<A=A]R3	94	126	274	180
R3<A=A]R2		Not Considered		

Semijoin R2<P=P]R3 has the highest (Benefit - Cost) so added to SJP.

Iteration 6

Semijoin	Cost	Relation Reduced To	Benefit	Benefit - Cost
R1<P=P]R2	31	6	174	143
R2<P=P]R1	180	9	37	Negative
R1<P=P]R3	115	21	159	44
R3<P=P]R1		Not Considered		
R2<P=P]R3		Not Considered		
R3<P=P]R2	31	6	174	143
R1<S=S]R2		Not Considered		
R2<S=S]R1		Not Considered		
R2<A=A]R3	94	15	31	Negative
R3<A=A]R2		Not Considered		

Semijoin R3<P=P]R2 has the highest (Benefit - Cost) so added to SJP (if there are more than one semijoin having the highest (Benefit - Cost) then any one of them is randomly selected and included in SJP) .

Iteration 7

Semijoin	Cost	Relation Reduced To	Benefit	Benefit - Cost
R1<P=P]R2	31	6	174	143
R2<P=P]R1	180	9	37	Negative
R1<P=P]R3	4	1	179	175
R3<P=P]R1		Not Considered		
R2<P=P]R3		Not Considered		
R3<P=P]R2		Not Considered		
R1<S=S]R2		Not Considered		
R2<S=S]R1		Not Considered		
R2<A=A]R3	6	1	45	39
R3<A=A]R2		Not Considered		

Semijoin R1<P=P]R3 has the highest (Benefit - Cost) so added to SJP.

Iteration 8

Semijoin	Cost	Relation Reduced To	Benefit	Benefit - Cost
R1<P=P]R2	31	1	0	Negative
R2<P=P]R1	1	1	45	44
R1<P=P]R3		Not Considered		
R3<P=P]R1		Not Considered		
R2<P=P]R3		Not Considered		
R3<P=P]R2		Not Considered		
R1<S=S]R2		Not Considered		
R2<S=S]R1		Not Considered		
R2<A=A]R3	6	1	45	39
R3<A=A]R2		Not Considered		

Semijoin R2<P=P]R1 has the highest (Benefit - Cost) so added to SJP.

Iteration 9

Semijoin	Cost	Relation Reduced To	Benefit	Benefit - Cost
R1<P=P]R2	1	1	0	Negative
R2<P=P]R1		Not Considered		
R1<P=P]R3		Not Considered		
R3<P=P]R1		Not Considered		
R2<P=P]R3		Not Considered		
R3<P=P]R2		Not Considered		
R1<S=S]R2		Not Considered		
R2<S=S]R1		Not Considered		
R2<A=A]R3	6	1	0	Negative
R3<A=A]R2		Not Considered		

As there is no beneficial semijoin, main loop of the algorithm terminates.

At this point, size of relations R1, R2 and R3 reduced to 6, 1 and 6 respectively. So, the total cost = 100 +100+90+180+115 +31+4+1+6+1+6 = 634.

A.3 Stepwise Execution of Example 7.7 Using Algorithm LIGHT

Iteration 1

Semijoin	Cost	Relation Reduced To	Benefit	Benefit - Cost	Avg. % gain
R1<P=P]R2	400	400	600	200	30
R2<P=P]R1	400	800	1200	800	22.44
R1<P=P]R3	900	900	100	Negative	
R3<P=P]R1	400	1200	1800	1400	30
R2<P=P]R3	900	1800	200	Negative	
R3<P=P]R2	400	1200	1800	1400	30
R1<S=S]R2	450	900	100	Negative	
R2<S=S]R1	100	400	1600	1500	37.73
R2<A=A]R3	300	2000	0	Negative	
R3<A=A]R2	100	1000	2000	1900	48.08

From here on Avg. % gain means average percentage gain in reduction of joining attributes cardinalities. Semijoin R3<A=A]R2 has the highest avg. % gain so added to SJP.

Iteration 2

Semijoin	Cost	Relation Reduced To	Benefit	Benefit - Cost	Avg. % gain
R1<P=P]R2	400	400	600	200	30
R2<P=P]R1	400	800	1200	800	22.44
R1<P=P]R3	634	634	366	Negative	
R3<P=P]R1	400	400	600	200	29.95
R2<P=P]R3	634	1268	732	98	12.16
R3<P=P]R2	400	400	600	200	29.95
R1<S=S]R2	450	900	100	Negative	
R2<S=S]R1	100	400	1600	1500	37.73
R2<A=A]R3	100	667	1333	1233	31.3
R3<A=A]R2			Not Considered		

Semijoin R2<S=S]R1 has the highest avg. % gain so added to SJP.

<u>Iteration 3</u>

Semijoin	Cost	Relation Reduced To	Benefit	Benefit - Cost	Avg. % gain
R1<P=P]R2	267	267	733	466	36.625
R2<P=P]R1	400	160	240	Negative	
R1<P=P]R3	634	634	366	Negative	
R3<P=P]R1	400	400	600	200	29.95
R2<P=P]R3	634	254	146	Negative	
R3<P=P]R2	267	267	733	466	36.59
R1<S=S]R2	90	180	820	730	68.5
R2<S=S]R1			Not Considered		
R2<A=A]R3	100	134	266	166	44.15
R3<A=A]R2			Not Considered		

Semijoin R1<S=S]R2 has the highest avg. % gain so added to SJP.

<u>Iteration 4</u>

Semijoin	Cost	Relation Reduced To	Benefit	Benefit - Cost	Avg. % gain
R1<P=P]R2	267	49	131	Negative	
R2<P=P]R1	180	72	328	148	54.54
R1<P=P]R3	634	115	65	Negative	
R3<P=P]R1	180	180	820	720	43.93
R2<P=P]R3	634	254	146	Negative	
R3<P=P]R2	267	267	733	466	36.59
R1<S=S]R2			Not Considered		
R2<S=S]R1			Not Considered		
R2<A=A]R3	100	134	266	166	44.15
R3<A=A]R2			Not Considered		

Semijoin R2<P=P]R1 has the highest avg. % gain so added to SJP.

Iteration 5

Semijoin	Cost	Relation Reduced To	Benefit	Benefit - Cost	Avg. % gain
R1<P=P]R2	49	9	171	122	72.5
R2<P=P]R1			Not Considered		
R1<P=P]R3	634	115	65	Negative	
R3<P=P]R1	180	180	820	720	43.93
R2<P=P]R3	634	46	26	Negative	
R3<P=P]R2	49	49	951	902	72.95
R1<S=S]R2			Not Considered		
R2<S=S]R1			Not Considered		
R2<A=A]R3	100	24	48	Negative	
R3<A=A]R2			Not Considered		

Semijoin R3<P=P]R2 has the highest avg. % gain so added to SJP.

Iteration 6

Semijoin	Cost	Relation Reduced To	Benefit	Benefit - Cost	Avg. % gain
R1<P=P]R2	49	9	171	122	72.5
R2<P=P]R1			Not Considered		
R1<P=P]R3	32	6	174	142	81.66
R3<P=P]R1	180	9	40	Negative	
R2<P=P]R3	32	3	69	37	95.05
R3<P=P]R2			Not Considered		
R1<S=S]R2			Not Considered		
R2<S=S]R1			Not Considered		
R2<A=A]R3	49	12	60	11	78.67
R3<A=A]R2			Not Considered		

Semijoin R2<P=P]R3 has the highest avg. % gain so added to SJP.

Iteration 7

Semijoin	Cost	Relation Reduced To	Benefit	Benefit - Cost	Avg. % gain
R1<P=P]R2	2	1	179	177	99.44
R2<P=P]R1			Not Considered		
R1<P=P]R3	32	6	6	142	81.66
R3<P=P]R1	180	9	40	Negative	
R2<P=P]R3			Not Considered		
R3<P=P]R2			Not Considered		
R1<S=S]R2			Not Considered		
R2<S=S]R1			Not Considered		
R2<A=A]R3	49	1	2	Negative	
R3<A=A]R2			Not Considered		

Semijoin R1<P=P]R2 has the highest avg. % gain so added to SJP.

Iteration 8

Semijoin	Cost	Relation Reduced To	Benefit	Benefit - Cost	Avg. % gain
R1<P=P]R2			Not Considered		
R2<P=P]R1			Not Considered		
R1<P=P]R3	2	1	0	Negative	
R3<P=P]R1	1	1	48	47	97.41
R2<P=P]R3			Not Considered		
R3<P=P]R2			Not Considered		
R1<S=S]R2			Not Considered		
R2<S=S]R1			Not Considered		
R2<A=A]R3	49	1	2	Negative	
R3<A=A]R2			Not Considered		

Semijoin R3<P=P]R1 has the highest avg. % gain so added to SJP.

Iteration 9

Semijoin	Cost	Relation Reduced To	Benefit	Benefit - Cost	Avg. % gain
R1<P=P]R2		Not Considered			
R2<P=P]R1		Not Considered			
R1<P=P]R3	1	1	0	Negative	
R3<P=P]R1		Not Considered			
R2<P=P]R3		Not Considered			
R3<P=P]R2		Not Considered			
R1<S=S]R2		Not Considered			
R2<S=S]R1		Not Considered			
R2<A=A]R3	1	1	1	1	60.66
R3<A=A]R2		Not Considered			

Semijoin R2<A=A]R3 has the highest avg. % gain so added to SJP.

Iteration 10

Semijoin	Cost	Relation Reduced To	Benefit	Benefit - Cost	Avg. % gain
R1<P=P]R2		Not Considered			
R2<P=P]R1		Not Considered			
R1<P=P]R3	1	1	0	Negative	
R3<P=P]R1		Not Considered			
R2<P=P]R3		Not Considered			
R3<P=P]R2		Not Considered			
R1<S=S]R2		Not Considered			
R2<S=S]R1		Not Considered			
R2<A=A]R3		Not Considered			
R3<A=A]R2		Not Considered			

As there is no beneficial semijoin, main loop of the algorithm terminates. At this point, size of relations R1, R2 and R3 are all reduced to 1. So, the total cost = 100 +100+90+180+49+32+2+1+1+1+1+1 = 558.

REFERENCES

[1] Agrawal, P., Bitton, D., Guh, K., Liu, C., and Yu, C., "A Case Study for Distributed Query Processing", IEEE 1998, Pages 124-130.

[2] Apres, P., Hevner, A., and Yao, S.B., "Optimization Algorithms for Distributed Queries", IEEE Transactions on Software Engineering, January 1983, Pages 57-68.

[3] Apres, P.M.G., "Data Allocation in Distributed Database Systems", ACM Transactions on Database Systems, Vol. 13, No. 3, September 1988, Pages 263-304.

[4] Bernstein, P.A., and Chiu, D.W., "Using Semijoin to Solve Relational Queries", ACM Vol. 28, No. 1, January 1981, Pages 25-40.

[5] Bernstein, P.A., and Goodman, N., "An Algorithm for Concurrency Control and Recovery in Replicated Distributed Databases".

[6] Bernstein, P.A., and Goodman, N., "The Power of Natural Semijoins", SIAM J. Computing Vol. 10, No. 4, November 1981, Pages 751-771.

[7] Bernstein, P.A., Goodman, N., Wong, E., Reeve, C.L., and Rothnie, Jr., J.B., "Query Processing in a System for Distributed Databases (SDD – 1)", ACM Transactions on Database Systems, Vol. 6, No. 4, December 1981, Pages 602-625.

[8] Bodorik, P., and Riordon, J.S., "Distributed Query Processing Optimization Objectives", Proceedings of IEEE Fourth International Data Engineering Conference, Los Angeles, CA, February 2-4, 1998, Pages 320-329.

[9] Bodorik, P., and Riordon, J.S., "Heuristic Algorithms for Distributed Query Processing", IEEE 1988, Pages 144-155.

[10] Ceri. S., and Gotlob, G., “Optimizing Joins Between Two Partitioned Relations in Distributed Databases”, Journal of Parallel and Distributed Computing, Vol. 3, 1986, Pages 183-205.

[11] Chang, J.M., “A Heuristic Approach to Distributed Query Processing”, Proceedings of the Eight International Conferences on Very Large Data Bases, September 1982, Pages 54-61.

[12] Cormen, T.H., Leiserson, C.E., Rivest, R., and Stein, C., “Introduction to Algorithms”, Prentice-Hall, Second Edition, 2004.

[13] Date, C.J., “An Introduction to Database Systems”, Pearson Education, Seventh Edition, 2005.

[14] Drenick, P.E., and Smith, E.J., “Stochastic Query Optimization in Distributed Databases”, ACM Transactions on Database Systems, Vol. 18, No. 2, June 1993, Pages 262-288.

[15] Elmasri, R., and Navathe, S.B., “Fundamentals of Database Systems”, Pearson Education, Third Edition, 2003.

[16] Epstein, R., Stonebraker, M., and Wong, E., “Distributed Query Processing in a Relational Database System”, Proceeding of ACM SIGMOD Conference, June 1978, Pages 169-180.

[17] Florescuu, D., Levy, A., and Mendelzon, A., “Database Techniques for the Worldwide Web: A Survey”, ACM SIGMOD, September 1998, Vol. 27, No. 3, Pages 59-74.

[18] Fu, Q. “Distributed Query Optimization Using Multi-attribute Semijoin Operations”, Master Thesis, University of Windsor, Ontario. Canada, 1996.

[19] Gouda, M.G., and Dayal, U., “Optimal Semijoin Schedules for Query Processing in Local Distributed Database Systems”, Department of Computer Science, University of Texas at Austin, November 1980.

[20] Haraty, R.A., and Fany, R.C., “Query Acceleration in Distributed Database Systems”, Revista Comlombiana de Computacion, Vol. 2, No. 1, 2001, Pages 19-34.

[21] Hevner, A.R., and Yao. S.B., “Query Processing in Distributed Database Systems”, IEEE 1979, Transaction on Software Engineering, Pages 177-187.

[22] Hevner, A.R., Wu, O.Q., and Yao, S.B., “Query Optimization on Local Area Networks”, ACM Transactions on Office Information, Vol. 13, No. 1, January 1985, Pages 35-62.

[23] Ip, A., Rahayu, W., and Singh, S., “Query Optimisation in a Non-Uniform Bandwidth Distributed Database System”, IEEE 2000, Proceeding of the Fourth International Conference.

[24] Kossmann, D., “The State of the Art in Distributed Query Processing”, ACM Computing Surveys, Vol. 32, No. 4, December 2000, Pages 422-469.

[25] LaFortune, S., and Wong, E., “A State Transition Model for Distributed Query Processing”, ACM Transactions on Database Systems, Vol. 11, No. 3, September 1986, Pages 294-322.

[26] Ozsu, M.T., and Valduriez, P., “Distributed and Parallel Database Systems”, ACM Computing Surveys, Vol. 28, No. 1, March 1996, Pages 125-128.

[27] Ozsu, M.T., and Valduriez, P., “Principles of Distributed Database Systems”, Pearson Education, Second Edition, 2004.

[28] Pentaris, F., and Ionnidis, Y., “Query Optimization in Distributed Networks of Autonomous Database Systems”, ACM Transactions on Database Systems, Vo. 31, No. 2, June 2006, Pages 537-583.

[29] Pramanik, S., and Vineyard, D., "Optimization Join Queries in Distributed Databases", IEEE 1998, Pages 1319-1326.

[30] Ramakrishna, R., and Gehrke, J., "Database Management System", McGraw-Hill, Third Edition, 2003.

[31] Selinger, P.G., and Adiba, M., "Access Path Selection in Distributed Database Management Systems", Proceeding of International Conference on Databases, Aberdeen University, Aberdeen, Scotland, July 1980.

[32] Silberschatz, A., Korth, H., and Sudarshan, S., "Database System Concepts", McGraw-Hill, Fourth Edition, 2002.

[33] Wong, E., "Retrieving Dispersed Data from SDD-1: A System of Distributed Databases", Proceedings of Second Berkeley Workshop on Distributed Data Management and Computer Networks, 1977, Pages 217-235.

[34] Yao, S.B., "Approximating Block Access to Database Organizations", CAMCM, April 1977.

[35] Yoo, H., and LaFortune, S., "An Intelligent Search Method for Query Optimization by Semijoins", IEEE Transactions on Knowledge and Data Engineering, Vol. 1, No. 2, June 1989, Pages 226-237.

[36] Yu, C.T. and Chang, C.C, "Distributed Query Processing", ACM Computing Surveys, Vol. 16, No. 4, December 1984, Pages 399-433.

[37] Yu, C.T., Lam, K., and Ozsoyoglu, M., "An Algorithm for Tree-Query Membership of a Distributed Query", Proceedings of COMPSAC 1979, IEEE Computer Society, November 1979.

[38] Yu, C.T., Lam, K., and Ozsoyoglu, M.Z, "Distributed Query Optimization for Tree Queries", Technical Report, Department of Information Engineering, V.I.C.C., July 1980.