Query Processing in Distributed Database with Non-disjointed Data Fragmentation

T.S. Narayanan

A Thesis

in

The Department

of

Computer Science

Presented in Partial Fulfillment of the Requirements for the Degree of Master of Computer Science at Concordia University
Montréal, Québec, Canada

August, 1985

ISBN 0-315-30629-7

ABSTRACT

Query Processing in Distributed Database with Non-Disjointed Data Fragmentation

T.S. Narayanan

This thesis emphasizes the importance of non-disjoint fragments in a distributed database environment. A classification for non-disjoint fragments is presented and this study examines three query processing and optimizing algorithms for non-disjoint fragmented and distributed databases.

The algorithms have the following features:

1. Make use of the redundant data to reduce Communication and I/O costs and allow parallel processing.

2. An efficient selection method of fragments referenced by a query.

Some simulated experimental results are given.

ACKNOWLEDGEMENT

I wish to express my deep gratitude to Dr. P. Goyal, whose guidance, encouragement, and advice has enlightened my way. I am especially indepted to him for many stimulating discussions and helpful suggestions.

I would like to thank Dr. B.C. Desai, Dr. C.K.S. Chong, and Dr. F. Sadri, for the valuable suggestions and discussions I had with them, while this work was being done.

I would also like to thank Dr. N. Srinivasan and Pervez Ahmed whose encouragement made this task simpler.

The research reported in this study was partially funded by the Natural Sciences and Engineering Research Council of Canada.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER I

INTRODUCTION

The evolution in data handling systems has progressed from the file management systems of the sixties to the data base management systems of the seventies and early eighties. Today developments in communications and price performance of desktop computers and workstations have led to the emergence of distributed databases[Rothnie and Goodman, 1977; Draffin and Poole, 1980; Bray, 1982] in an office environment. While computing with a standalone personal computer yields personal and local benefits, organisational productivity relies on linking with corporate resources. There are two different approaches in building distributed databases:

1. Integration of existing databases which are geographically dispersed [Landers and Rosenberg, 1982; Ferrier and Stangret, 1983].
2. Building a new non-centralized database to replace the existing centralized databases[Bernstein, et al., 1981; Williams, et al., 1982]

There are advantages and disadvantages in both approaches, but issues of integration become more central to the discussions in an office or already distributed environment.

The advantages of a distributed database are:

1. Increased availability
2. Better access time
3. Easy expansion

These benefits are obtained from fragmentation of related data units, and replication of data in a distributed database environment. This in turn requires insulation of the user from fragmentation and replication strategies i.e., provide replication and location transparency[Teorey and Fry, 1982; Ceri and Pelagatti, 1984]. This replication and geographic dispersion of data fragments make distributed database problems more complex than the centralized ones(though some of the basic problems are common). There are four major problems encountered in distributed databases:

1. Data allocation
2. Query processing and optimization
3. Concurrency control
4. Recovery

Data allocation deals with the problem of how to distribute the data among sites. Query processing problem includes finding a non redundant version of the database, deciding at which sites to execute what operations under the given constraints (data distribution, communication and processing capacities), and how to send the result to the required node. Concurrency control keeps multiple simultaneous transactions from interferring with each other. Finally, the recovery problem deals with the maintenance of database integrity in spite of possible system failures.

In this thesis we restrict our treatment to the relational data model[Date, 1982; Ullman, 1983]. The reader is assumed to be familiar with this model.

The existing distributed query processing techniques make the simplifying assumption that the fragments are disjoint. In this thesis, however, we emphasize the importance of non-disjoint fragments, a classification and a query processing technique making use of non-disjoint fragments are presented.

In the second chapter we introduce the terminology used, and some relevant distributed database concepts. In chapter three, we survey various query processing techniques for distributed databases, and other related literature. In chapter four, the classification of the non-disjoint fragments are presented. In chapter five, a query processing technique which takes into account the non-disjointness between fragments is introduced. This query processing technique is compared, in chapter six, with the general approach followed in existing techniques. In the last chapter, several open questions for future research are also presented.

CHAPTER II

REVIEW OF THE DATABASE CONCEPTS

This chapter presents a review of the database terminology and concepts. In Section 2.1 we present the features of relational data model briefly. In section 2.2 relational fragmentation is discussed. In section 2.3 a commonly agreed upon distributed database architecture is described. Finally in the last section, the problem of query optimization is presented.

2.1 RELATIONAL DATA MODEL

A formal definition of relation is based on the notion of domains. A domain is denoted by D. The cartesian product of domains $D_1, D_2, \ldots, D_n$, consists of all n-tuples $(d_1, d_2, \ldots, d_n)$, where $d_i \in D_i$ for $i=1,2,\ldots,n$. A relation R is a finite subset of cartesian products and can be represented by a table. Each row of the table is called a tuple, and each column stands for an attribute.

Two different languages are used to express the queries in relational data model - the relational algebra and the relational calculus. A query expressed in the relational calculus can be translated into an equivalent relational algebraic representation. The main difference between the two representations is, the way in which they describe the result of the query. The basic operations in the relational algebra are selection, projection, join, and the normal set operations. We follow the notation used by Ceri and Pelagatti [Ceri and Pelagatti, 1984] to represent these operations. Relational algebraic representation is followed throughout the text.

2.2 FRAGMENTATIONS

Relations in a distributed database are decomposed into fragments and replicated in order to improve the access time, reliability and local autonomy. There are basically two types of fragments generated by these decomposition process, they are:

1. Vertical fragments
2. Horizontal fragments

<u>Vertical fragments[Navathe, et al., 1982]</u>

Vertical fragments are generated by splitting a relation into a number of component fragments by projection, in such a way that the original relation can be recovered by joining these component fragments; either because of common attributes in fragments or the sharing of internal identifier values (TID's). A vertical fragmentation of a relation EMPLOYEE is given in figure 2.1. Vertical fragmentations are also at the heart of normalization techniques - to remove redundancy and update anomalies from the database. Vertical fragmentation basically groups attributes of a relation into different sets.

Global Relation : EMPLOYEE

EMP#	NAME	DEPT	SKILL	SALARY
1	JOHN	1	A	100
2	PAUL	2	C	70
3	ALEX	1	A	100
4	SAM	2	B	90
5	TOM	1	C	70
6	RON	2	B	90

Vertical Fragment V1=($PA_{EMP\#,NAME,DEPT,SKILL}$(EMPLOYEE)):

EMP#	NAME	DEPT	SKILL
1	JOHN	1	A
2	PAUL	2	C
3	ALEX	1	A
4	SAM	2	B
5	TOM	1	C
6	RON	2	B

Vertical Fragment V2=($PA_{SKILL,SALARY}$(EMPLOYEE)):

SKILL	SALARY
A	100
B	90
C	70

Figure 2.1 Vertical Fragments.

Assumption: Each employee has only one skill.

Horizontal Fragment H1=($SL_{DEPT=1}$(EMPLOYEE)):

EMP#	NAME	DEPT	SKILL	SALARY
1	JOHN	1	A	100
3	ALEX	1	A	100
5	TOM	1	C	70

Horizontal Fragment H2=($SL_{DEPT=2}$(EMPLOYEE)):

EMP#	NAME	DEPT	SKILL	SALARY
2	PAUL	2	C	70
4	SAM	2	B	90
6	RON	2	B	90

Figure 2.2 Horizontal Fragments.

Horizontal fragments

Horizontal fragments are generated by forming subsets of the tuples in a relation through a selection operation, and recovering the relation by a union of the subsets [Ceri, Negri and Pelagatti, 1982; Maier and Ullman, 1983]. A horizontal fragementation of the global relation EMPLOYEE is shown in figure 2.2.

Mixed Fragments

In addition to the two basic types, fragments are generated by combining vertical and horizontal fragments. Although horizontal and vertical fragmentations can be repeated ad infinitum, generating fragmentation scheme with more than two levels of fragmentation is, usually, not of practical interest[Ceri and Pelagatti, 1984]. Even if there were such a need it is possible to represent the original fragmentation scheme by an equivalent two level fragmentation scheme, whose lowest level fragments are logical fragments represented by an expression with union and join as operators, and physical fragments of the original fragmentation scheme as operands. Figure 2.3 illustrates mixed fragmentation defined on EMPLOYEE.

Mixed Fragment M1=($PA_{EMP\#,NAME,DEPT,SKILL}(SL_{DEPT=1}(EMPLOYEE))$):

EMP#	NAME	DEPT	SKILL
1	JOHN	1	A
3	ALEX	1	A
5	TOM	1	C

Mixed Fragment M2=($PA_{EMP\#,NAME,DEPT,SKILL}(SL_{DEPT=2}(EMPLOYEE))$):

EMP#	NAME	DEPT	SKILL
2	PAUL	2	C
4	SAM	2	B
6	RON	2	B

Mixed Fragment M3=($PA_{EMP\#,NAME,SALARY}(SL_{DEPT=1}(EMPLOYEE))$):

EMP#	NAME	SALARY
1	JOHN	100
3	ALEX	100
5	TOM	70

Mixed Fragment M4=($PA_{EMP\#,NAME,SALARY}(SL_{DEPT=2}(EMPLOYEE))$):

EMP#	NAME	SALARY
2	PAUL	70
4	SAM	90
6	RON	90

Figure 2.3 Mixed Fragments.

<u>Fragmentation tree[Ceri and Pelagatti, 1984]</u>

A fragmenatation scheme can be conveniently represented by a fragmentation tree. In a fragmentation tree, the root corresponds to a global relation, the leaf nodes correspond to the fragments, and the intermediate nodes correspond to the intermediate result of the scheme. The set of nodes which are children of a given node represent the decomposition of this node by a fragmentation operation (vertical or horizontal). Figure 2.4 illustrates the fragmentation trees for various fragmentations schemes of EMPLOYEE.

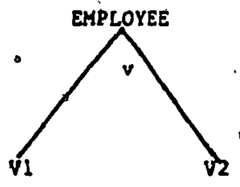
In existing systems, three rules are usually followed in defining these fragments[Ceri and Pelagatti, 1984].

C1: Completeness condition - All the data of the global relations must be mapped into the fragments.

C2: Reconstruction condition - It must be always possible to reconstruct each global relation from its fragments.

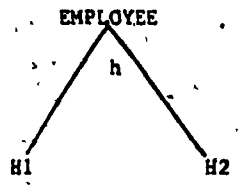
C3: Disjoint condition - Fragments of the relations are disjoint.

1. Vertical Fragmentation:



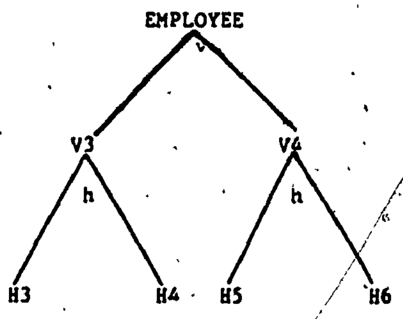
V1 : $PA_{EMP\#,NAME,DEPT,SKILL}$(EMPLOYEE)
V2 : $PA_{EMP\#,NAME,SALARY}$(EMPLOYEE)

2. Horizontal Fragmentation:



H1 : $SL_{DEPT=1}$(EMPLOYEE)
H2 : $SL_{DEPT=2}$(EMPLOYEE)

3. Mixed Fragmentation



V3: $PA_{EMP\#,NAME,DEPT,SKILL}$(EMPLOYEE)
V4: $PA_{EMP\#,NAME,SALARY}$(EMPLOYEE)
H3: $SL_{DEPT=1}$(V3)
H4: $SL_{DEPT=2}$(V3)
H5: $SL_{SALARY <95}$(V4)
H6: $SL_{SALARY>=95}$(V4)

Figure 2.4 Fragmentation trees.

The third rule has been basically, imposed for the sake of convenience[Ceri and Pelagatti, 1984], so that i) replication of data can be handled explicitly at the allocation level, and ii) it results in simple query processing strategies. However, this rule is not strictly applicable to vertical fragmentations, because it is usual that these fragments inherit the common tuple identifier values. The objective of this thesis is to study query processing algorithms, assuming the existence of non-disjoint horizontal fragments.

Non-disjoint fragments

When the fragments of a global relation overlap with each other, we call them non-disjoint or overlapping fragments. Due to the requirements of various applications, vertical and horizontal fragmentations may be required to be non-disjoint[Narayanan, et al., 1985]. Non-disjointness in vertical fragmentation has been extensively discussed in the literature, whereas non-disjointness among horizontal fragments has been rarely studied[Maier and Ullman, 1983].

Non-disjoint Vertical fragments[Navathe, et al., 1982]

As we have seen above, vertical fragmentation basically groups attributes of the global relation into different subsets. If one of the attributes is found in more than one fragment, then the fragments are called vertical clusters, while disjoint vertical fragments are called vertical partitions.

Non-disjoint horizontal fragments

If a tuple is found in more than one horizontal fragment then the horizontal fragments are considered to be non-disjoint [Maier and Ullman, 1983]. More details about non-disjoint horizontal fragments are given in chapter 4.

2.3 DISTRIBUTED DATABASE ARCHITECTURE

A general multilevel architecture of the distributed database system software[Teorey and Fry, 1982] is shown in figures 2.5. There are five levels to this architecture, divided into two major parts:

1. Distributed database management system
2. Local database management system

Each of the five levels supports a different view of the database. The global view defines all the data which are contained in the distributed database as if the database were not distributed at all. Global view of a hypothetical distributed database is shown in figure 2.6. The mapping between global relations and fragments is defined in the fragmentation view. Distributed view defines a mapping between fragments and nodes. Fragmented view and distributed view for the hypothetical distributed database are shown in figures 2.7 and 2.8 respectively. Local view defines all the data which are contained in the local database and managed by local DBMS (figure 2.9). User view defines the data which are accessible to a user. Figure 2.10 shows one of the user's veiw of the hypothetical database.

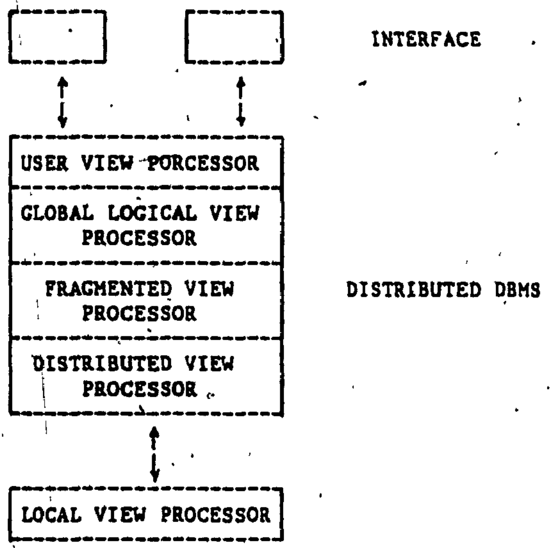

INTERFACE
USER VIEW PORCESSOR
GLOBAL LOGICAL VIEW PROCESSOR
FRAGMENTED VIEW PROCESSOR
DISTRIBUTED VIEW PROCESSOR
DISTRIBUTED DBMS
LOCAL VIEW PROCESSOR

Figure 2.5 Distributed DBMS architecture.

EMPLOYEE

EMP#	NAME	DEPT	SALARY
1	JOHN	1	100
2	PAUL	1	70
3	ALEX	2	100
4	SAM	2	90
5	TOM	3	70
6	RON	3	90

DEPT-CITY

DEPT	CITY
1	MONTREAL
2	OTTAWA
3	TORONTO

RAW MATERIALS

DEPT	ITEM	QUANTITY
1	P_1	500
1	P_2	100
2	P_3	940
3	P_4	100

Figure 2.6 Global Logical view.

EMPLOYEE

EMP#	NAME	DEPT	SALARY	
1	JOHN	1	100	Fragment 1
2	PAUL	1	70	
3	ALEX	2	100	Fragment 2
4	SAM	2	90	
5	TOM	3	70	Fragment 3
6	RON	3	90	

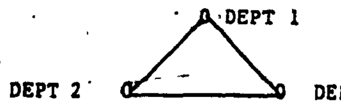
DEPT-CITY

DEPT	CITY
1	MONTREAL
2	OTTAWA
3	TORONTO

Fragment a

PNO	ITEM	QUANTITY	
1	P_1	500	Fragment A
1	P_2	100	
2	P_3	940	Fragment B
3	P_1	100	Fragment C

Figure 2.7 Fragmented View.



RELATION	FRAGMENT	LOCATION
EMPLOYEE	1	1
	2	1,2
	3	1,3
PLANT	a	1,2,3
RAW MATERIALS	A	1
	B	2
	C	3

Figure 2.8 Distributed View.

EMPLOYEE

EMP#	NAME	DEPT	SALARY
5	TOM	3	70
6	RON	3	90

Fragment 3

DEPT-CITY

DEPT	CITY
1	MONTREAL
2	OTTAWA
3	TORONTO

Fragment a

RAW MATERIALS

DEPT	ITEM	QUANTITY
3	P_1	100

Fragment C

Figure 2.9 Local view at Dept 3.

EMPLOYEE

EMP#	NAME	DEPT
1	TOM	3
6	RON	3

DEPT-GITY

DEPT	CITY
1	MONTREAL
2	OTTAWA
3	TORONTO

Figure 2.10 User View.

Fragmented view and distributed view are characteristics of the distributed DBMS. Fragmented view processor provides fragmentation transparency, and replication transparency; while the distributed view processor provides the location transparency to the end user.

2.4 QUERY PROCESSING

Query is a language expression that describes the data to be retrieved or updated. Query processing in distributed database refers to retrieving or changing data from several geographically distributed sites, in computer network. Query optimization refers to finding an optimal schedule in executing the query. Different parameters can be used in the optimization phase and these include response time, total time, and total transmission time etc. Exact query optimization procedure is in general computationally intractable[Hevener, 1979] and is hampered further by the lack of precise statistical information about the database. Most of the query optimization algorithms rely on heuristics, nevertheless, the term 'query optimization' will be used to refer to strategies intended to improve the efficiency of query processing procedures.

2.4.1 Query Languages and Query Graphs

There are many procedural, and non-procedural query languages. Non-procedural query languages do not specify the way the data has to be retrieved. An example of a non-procedural query Q is shown in figure 2.11. When the same query Q is expressed in a procedural language, it will appear as shown in figure 2.12.

```
SELECT CITY
    FROM  DEPT-CITY
    WHERE EMPLOYEE.NAME  =  'PAUL' AND
          EMPLOYEE.DEPT  =  DEPT-CITY.DEPT
```

Figure 2.11 Non-Procedural Query Q.

$$PA_{CITY}(SL_{NAME='PAUL'}\ (EMPLOYEE\ JN_{E.DEPT=D.DEPT}\ DEPT\text{-}CITY))$$

PA	: Project
SL	: Select
JN	: Joint
E	: EMPLOYEE
D	: DEPT-CITY

Figure 2.12 Procedural Query Q.

QUEL, and PLAIN are some of the commercially available query languages. QUEL has been implemented on INGRES[Held, Stonebraker and E. Wong, 1975], and is based on the relational calculus. PLAIN[Wasserman, et al., 1981] is based on relational algebra. More details about the query languages can be found in the literature [Astrahan and Chamberlin, 1975; Czarnik, Schuster and Tsichritzis, 1975; Schmid and Bernstein, 1975].

In order to have easier manipulation, queries are represented using graphical notations. Operator tree and optimization graph are two such frequently used graphical notations.

Operator tree[Ullman, 1983]

Operator tree specifies a partial order in which the operations must be applied; the operator tree for Q is shown in figure 2.13. In this case, the join is applied first, followed by selection and projection.

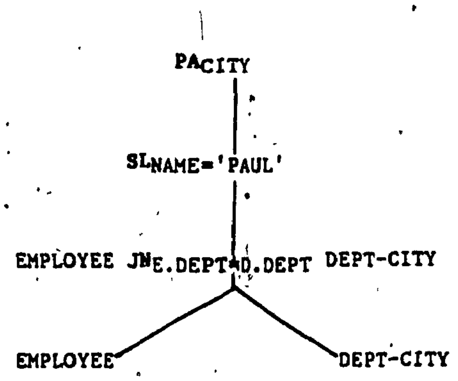


Figure 2.13 Operator tree of Q.

Optimization Graph[Ceri and Pelagatti, 1984]

Optimization graph is another graphical representation for the query, at a a higher level of abstraction, more suitable for distributed databases. It represents only the binary operations joins and unions. In this model nodes represent the reduced fragments, joins are represented by edges between nodes which are labelled with the join specification, and union is represented by hypernodes enclosing their operands.

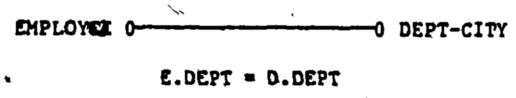


Figure 2.14 Optimization Graph of Q.

Depending upon the nature of the optimization graph, a query can be classified into two types:

Cyclic Queries

Queries whose optimization graph contains one or more closed paths between nodes are known as cyclic queries.

Tree Queries

Query whose optimization graph is in tree structure, is called a tree query, or acyclic query.

2.4.2 Query Scheduling

Irrespective of the type of language used to express the query, and nature of the database(centralized/distributed), the query processing in general consists of three phases:

1. Parsing the query
2. Determining a schedule
3. Execution of the schedule.

In parsing phase, query is checked for syntax and semantics with the aid of the dictionaries. If the query is correct, then the second phase is executed. In this phase a query optimization algorithm, will determine a schedule to process the query from a set of possible schedules. A schedule contains the basic operations, and the sites at which these are to be executed. Basic operations are relational algebraic operations and data transmissions. In the last phase data is retrieved from the database according to the schedule decided in phase two. These three phases are illustrated in figure 2.15[Apers, 1983].

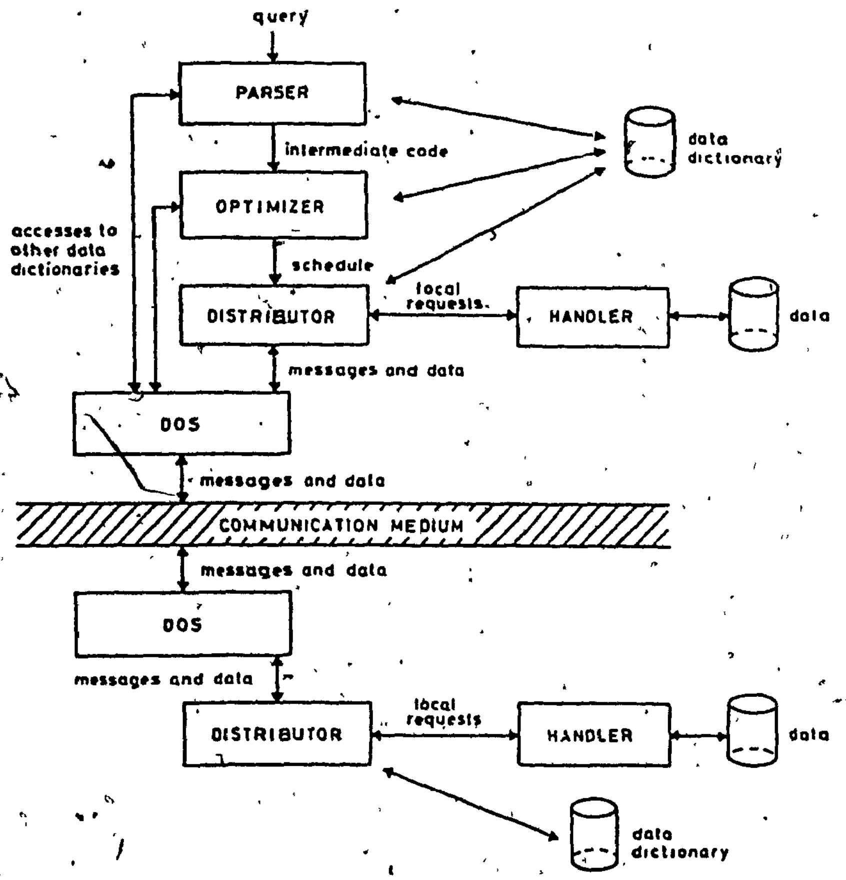


Figure 2.15. Three phases of query processing [Apers, 1983].

2.4.3 Query Processing in Centralized Database

Most of the query processing algorithms for distributed databases are based on centralized query processing algorithms. In centralized database systems, the query evaluation cost depends on the number of I/O operations, and CPU utilizations. The DBMS provides various mechanisms to efficiently access the data in a relation. The query optimizer selects between the different access mechanisms to achieve the desired goal; this combination of access mechanisms is known as the access path. The choice of an appropriate access path for a given query is to be decided by the query optimizer, and is part of the schedule.

In general it is easier to schedule a query which is referring to fewer relations. Two strategies are used in transforming a query with many relations(multivariable query) to queries with fewer relations.

1. Substitution

N-relation query $Q(R_1,R_2,\ldots,R_N)$ can be transformed, by substituting for R_i, its actual values, into K_i (N-1)-relation queries $Q(R_1,R_2,\ldots,a,\ldots,R_N)$[Pecherer, 1976; Selinger, et al., 1979]. K_i denotes the cardinality of R_i and 'a' stands for a particular tuple value of R_i. Repeated reduction eventually reduces the complexity of a query to a set of one variable queries.

2. Reduction[Wong and Youssefi, 1976]

A query $Q(R_1, R_2, \ldots, R_N)$ can be subdivided into subqueries $Q1(R_i, \ldots R_n)$, and $Q2(R_1, \ldots \underline{R}_i)$, where the predicates of the two subqueries overlap on R_i only. The result of Q1 is denoted by $\underline{R}_i$, and it is substituted into Q2 for further evaluation. Q1 and Q2 must be evaluated serially. This process can be recursively applied to each subquery, until subqueries are irreducible.

Normal forms

One variable query can be represented in disjunctive/conjunctive normal forms. In disjunctive normal form, the selection condition is expressed as conjunctions of disjunctive clauses. In conjunctive normal form, selection predicate is expressed as disjunctions of conjunctive clauses.

Selectivity factor

Selectivity factor is the ratio of relation size reduction after an operation. Estimating the selectivity factor for various operations has been based on uniform distribution and independent distribution assumptions [Rowe, 1983; Piatetsky, 1984].

Selection Selectivity factor

Selection selectivity factor can be estimated either by keeping profiles of attribute value distribution or by assuming a uniform distribution [Hevener and Yao, 1979; Selinger, et al., 1979]. The selectivity factor for selection of one value on a single attribute defined over a domain of size v, and uniformly distributed over v is simply 1/v.

Projection Selectivity factor

The selectivity factor for projection is equal to the tuple size reduction ratio. This estimate is exact if duplicate tuples are not removed.

Join Selectivity factor

Attribute independence is generally assumed in estimating the selectivity factor of joins. For each join attribute, the join selectivity gives the selection selectivity of this domain on any other joinable attribute. For example, consider the attribute desnsity $v(R_A)/D(A)$ of attribute A in a relation R, where $v(R_A)$ is the cardinality of attribute A in R, and D(A) is the cardinality of its domain. In case of a join $R\mathbf{JN}_{A=B}R'$, the attribute density for A gives the percentage of tuples of R' which can be joined with tuples of R.

2.4.4 Query Processing in Distributed Database

Data transmission is the characteristic feature of distributed databases, because of this the query evaluation cost in distributed database contains a component due to transmission cost, in addition to I/O and processing cost. Many of the query optimization algorithms suggested in the early stages of distributed databases focussed their strategies on reducing the transmission cost. I/O and processing costs were ignored in these algorithms. This was justified on the grounds that compared to data transmission cost, the other two costs had second order effect on the query evaluation cost. With the introduction of high speed networks this justification is no longer valid. Also the processing cost for join operation is non-linear with respect to the size of the relations, while transmission cost is basically assumed to be linear. These two facts have led to the introduction of new approaches based on transmission cost and I/O and processing costs.

In general query optimization algorithms consider various possible schedules for a given query, and choose the best schedule after considering their estimated evaluation costs. Cost of a schedule is usually estimated in terms of time. Processing time of an operation, denoted by $PT(R)=P_0+PC(R)$ where P_0 is the start-up queueing time and $PC(R)$ is the processing time, R stands for the operation. The transmission time of data, denoted by $TT(R)=T_0+TC(R)$, where T_0 is the start-up time, TC(R) is the transmission time, and R stands for the amount of data to be transmitted.

Total time

Total time of a schedule is equal to sum of the processing time, I/O time, and transmission time taken by the schedule in executing the query.

Response time

Response time of a schedule is the time elapsed between the start of the first basic opeation and the moment the required data is presented to the user.

Cost of a schedule also known as cost function can be used to measure either response time or total time or any other parameter which is to be optimized. The cost function used in our algorithms is the total time, and the start-up cost of the processing time is ignored in the calculation of the total time.

Semi-join[Bernstein and Chiu, 1981]

In order to send only the necessary data, a semijoin operation has been proposed. The semijoin of relation R2 by relation R1 on attribute A can be expressed as

$$R1 \ SJ_{R1.A=R2.A} \ R2$$

Informally, the semijoin between R1 and R2 eliminates from

R2 all the tuples which do not join with R1 on attribute A. The role of the semijoin operation is to minimize the transmission of data irrelevant to the solution of the query. A distributed join between R1 and R2 can then be performed by the following steps:

1. Send $PA_A(R1)$ to R2
2. Locally execute $PA_A(R1)$ JN_A R2 = R
3. send R to R1
4. locally execute R JN_A R1

Materialization

Materialization is assuming that only one copy of each relation exists in the database. Redundancy due to replication of the fragments are ignored in query processing by accessing a pre-selected copy. Most of the query processing algorithms work with pre-determined materialization.

Transmission Cost Model

The time required to transmit data from one site to another depends on the network topology and the bandwidth of the communication channels. If all sites are directly connected with each other, and when the queueing delays before transmission are, on the average, the same, then it is possible to have Equal Transmission Cost Model. If, on the other hand, the network has an arbitary topology or queueing delays vary too much we will speak of the Arbitrary

Transmission Cost Model.

Two Stages of Query Scheduling

Most of the scheduling is done in two stages. First the query is transformed into disjunctive normal form. The conjuncts can be treated as subqueries which can be executed independently. In the first phase these subqueries are executed in parallel at the required node. In the second phase some of the results of the first phase are joined recursively at some common node till we get the final result.

CHAPTER III

DISTRIBUTED QUERY PROCESSING ALGORITHMS

The poineering work in distributed query processing is due to E. Wong. His algorithm published in 1977 was designed for SDD-1[Wong, 1977]. Since then many algorithms have been suggested. Because of the variety of communication and cost models used, a qualitative and quantitative comparison of these algorithms is difficult. The criterion to be used for the comparison of these algorithms is presented in section 3.1. Some of the algorithms and other related developments are briefly surveyed in section 3.2.

3.1 A QUALITATIVE COMPARISON OF QUERY PROCESSING ALGORITHMS

Query processing algorithms can be compared by analysing their handling of the following aspects:

1. Materialization
2. Type of Scheduling
3. Objective of scheduling
4. Transmission cost model
5. Type of Communication model
6. Inclusion of the processing cost
7. Result site of the join
8. Capability to handle ad-hoc queries
9. Fragmentations permitted

Some of these terminologies have been discussed in chapter two; the others are described below.

Type of Scheduling (dynamic/static)

In static scheduling, the schedule for query evaluation is determined only once - at the beginning - utilising previously gathered statistics, eg., selectivity factor. Whereas in dynamic scheduling the schedule may be altered depending upon the intermediate results. Dynamic scheduling, though a complicated process, in general, gives better results[Epstein, 1980].

Objective of scheduling

Objective of query processing algorithms is usually to optimize one or more of the following cost functions:

1. Response time[section 2.4.4]
2. Total time[section 2.4.4]
3. Total network traffic
4. Total resources used

Type of Communication model

Some of the query processing algorithms are suitable for either broadcast or site-to-site model, while a few of the algorithms have options to handle both the communication models.

Result site of join

There are two options in performing the join operation between data from several nodes: i) all of the data are moved to the result node, and then joined; or ii) data are moved to intermediate nodes, joined, and the result is transmitted to the result node.

3.2 A SURVERY OF SOME EXISTING QUERY PROCESSING ALGORITHMS

1. Wong[Wong, 1977]

This algorithm, known as 'hill climbing' algorithm, computes progressive refinements on an initially feasible solution, until no beneficial modification of the evaluation plan is possible. An improved version of the algorithm uses semi-joins to reduce the amount of data that needs to be transmitted.

2. Hevener and Yao[Hevener and Yao, 1979]

Hevener et al., have suggested a family of algorithms to optimize either response time or total time. These algorithms are based on optimal solutions for a restricted class of queries called 'simple queries'. Simple queries are defined as queries for which, after the execution of the local processing phase, each relation contains only one common joining attribute, which is also the only output of the query. To solve the simple query, only the set intersections on the common domain need to be performed.

3. Epstein, Stonebraker, and Wong

This algorithm, used in distributed INGRES, is partly based on a centralized query evaluation algorithm[Stonebraker and Neuhold, 1977]. The main idea is to choose the fragments of the relations to be processed against all other 'complete' relations; this requires the replication of the 'other' relations at all nodes at which fragments of the 'chosen' relation reside, permitting a join operation in parallel at a number of nodes. Two different algorithms[Epstein, 1980] are suggested, one for broadcast and one for point-to-point network. Performances of these algorithms were computed for a variety of test cases.

4. Baldissera, Bracchi and Ceri

The starting point for this algorithm is an operation tree. A recursive algorithm[Baldissera, Bracchi and Ceri, 1979] prunes the tree by removing subtrees; removing a subtree means deciding how its operations are to be performed. Choices are made at each 'branch' node of the tree, where the children of the branch node can be reduced using semi-joins; the method determines at each branch the cheapest sequence of semi-joins. The algorithm is also able to reduce a cyclic query.

5. Chu and Hurley[Chu and Hurley, 1979]

The complexity of the algorithm suggested by Chu and Hurley grows exponentially with the number of sites. In this algorithm query trees are formed for a given query using equivalence transformations, and only 'promising' trees are retained for processing; for each of these trees several 'query processing graphs' are obtained by merging contiguous operations which can be performed at the same site into a single node. Since each operation can be allocated to several sites, many different groupings are possible. Finally, a formula is given for evaluating the cost, in terms of file transmission for each such groupings.

6. Pelagatti and Schreiber

The complexity of this algorithm also grows exponentially with the number of sites. It is designed for queries that are stated quite frequently on a slowly changing database. The design [Pelagatti and Schreiber, 1979] consists of the following three phases:

1. Determination of a 'logical strategy'; this involves equivalence transformation of operator tree over the global schema.
2. Determination of a 'distribution strategy'; each global relation is now considered as a fragmented relation and global queries are translated to fragmented queries.
3. Determination of an 'execution strategy'; this involves the determination of the nodes upon which each operation is executed and of transmissions for moving 'operands'.

The three phases are executed in cascade with feedback between them. The third phase involves the solution of a linear integer programming problem.

7. Toth, Mahamoud and Riordon

In this algorithm[Toth, Mahmoud and Riordon, 1978] a query is expressed in relational algebra and will appear like this:

$$r \leftarrow ((R_1 x R_2 x \ldots x R_n)|C)[Z]$$

Where R's are the relations of the distributed database, C is a selection predicate on the tuples of $(R_1 x R_2 x \ldots x R_n)$, and Z is a projection. C and Z can be transformed such that operations on single relation results these are the subqueries. Every site will execute its subquery. Determining the order in which the results are combined together is done heuristically based upon a special type of query called 'class A' query. For the class A query an optimal solution can be found for minimizing the total network traffic subject to a given constraint.

8. Toan[Ngyuen Gia Toan, 1979]

This algorithm uses dynamic scheduling, but is limited to the queries which are representable in a tree structure. The query is represented by an operator tree that operates on a predetermined materialization. The operation tree is partitioned into local subtrees. Each site processes its subtree. Then data transfers are made in a decentralized manner to obtain the final result.

9. Apers[Apers, 1979]

This algorithm is a variation of Hevener and Yao's algorithm [Hevener and Yao, 1979], for delay(response time) minimization. Apers et al., [Apers, Hevener and Yao, 1983] suggested two other algorithms for total cost minimization, which are heuristic in nature, and show that the total time minimization is a harder problem than delay minimization.

10. Sacco[Sacco, 1984]

Set of heuristic strategies to solve multirelational queries in local area networks using broadcast routing are presented in this work. In broadcast routing data transmited by one site are received by all the other network sites. This fact is exploited to derive new strategies to minimize a linear combination of query response and total time.

Other than the above algorithms, the following publications have also been found to be relevant.

11. Selinger et al.[Selinger, et al., 1979]

The problem of redundant data is solved by extending the query tree such that a choice is made among non-redundant materializations of non-partitioned data[Selinger, et al., 1979]. Selinger and Adiba showed [Selinger and Adiba, 1980] that the I/O and processing time cannot be neglected over communication time in calculating the total cost function.

12. Maier and Ullman[Maier and Ullman, 1983]

This work is concerned with the theory of fragments. Algorithms for inserting and deleting from physical fragments are discussed. These algorithms do not assume disjoint fragments, and this aspect makes this paper unique.

13. In-Sup Paik and Delobel

This algorithm [In-sup Paik and Delobel, 1979] is concerned with the choice of materialization. It is done in three phases. In the first phase, known as clustering phase all possible materializations are investigated. In the second phase unwanted fragments are filtered out. In the final phase several alternatives for processing the query are compared to find the best one.

3.3 COMPARISON TABLE OF ALGORITHMS

Table 3.1 gives a qualitative comparison of various algorithms surveyed in previous s[illegible] All the query processing algorithms assume equal transmission cost model, and hence it is not listed in the table.

Algorithm	Wong77	Hevener79	Epstein80	Baldissera79	Chu-Hurley79	Pelagatti79	Toth81	Toan79	Sacco84	Maier83
Materialization assumed	Yes	Yes	Yes	No	Yes	No	Yes	Yes	No	-
Objective function	Total time	Response or Total time	Response or Total time	Total trans. time	Total trans. + Proc. time	Response time @	Response time	Response time @	Response time	-
Can handle - query type	Any type	'Simple' queries	Any type	Tree query	Tree Queries	Non ad-hoc queries	Class A queries	Tree queries	Any type	-
Scheduling	Static	Static	Dynamic	Static	Static	Static	Static	Dynamic	Dynamic	-
Communication Model	Site-to-Site	Site-to-Site	Site-to-Site Broadcast	Site-to-Site Broadcast	Broadcast	Site-to-Site Broadcast	Site-to-Site Broadcast	Broadcast	Broadcast	
Join Operation at the	Result Site	Result site	Arbitrary site	Depends	Arbitrary site	Arbitrary site	Depends	Arbitrary site	Arbitrary site	-
Includes the Processing cost	Arbitrary cost function	No	Yes	No	Yes	No	No	Yes	No	-
Fragments	Disjoint	Disjoint	Disjoint	Disjoint	Disjoint	Disjoint	Disjoint	Disjoint	Disjoint	Non Disjoint

@ subject to the minimization of the transmission time

Table 3.1

CHAPTER IV

NON-DISJOINT FRAGMENTS

Since the non-disjointness in vertical fragmentation has already been dealt with in the literature[Navathe, et al., 1982], we emphasize the importance of non-disjoint horizontal fragments in this chapter. In the first section we describe a process used in defining the disjoint horizontal fragments, to allow comparison with non-disjoint horizontal fragmentation scheme in the next section. In the last section, a classification of the non-disjoint fragments is presented. Unless otherwise stated the horizontal fragment refers to the non-derived (fragments which are defined using their own attributes and not on the attributes of the other fragments/relations) horizontal fragments only.

4.1 DESIGN OF DISJOINT HORIZONTAL FRAGMENT

Disjoint horizontal fragmentation requires that each tuple of the global relation be in a unique fragment. Thus, it involves determining a set of disjoint fragmentation predicates. This is usually done by considering the requirements of some of the 'important' applications accessing the relation[Ceri and Pelagatti, 1984]. Let R be a relation with k important applications accessing it.

Though there are other factors which affect the design of horizontal fragments, we consider only the effect of applications alone, and these can be made to include the other factors.

Let the r^th application prefer a horizontal fragmentation of the relation R into n_r fragments such that the tuples of the i^th fragment satisfy the conditions C_{ri}'s($1<=i<=n_r$); C_{ri}'s being mutually exclusive. If we define fragments to the preference of one of the k applications it may conflict with the requirements of the other applications, else if we define k sets of fragments to the preference of the k applications, we will be defining non-disjoint fragments. To overcome these difficulties, fragments are defined using the conjunction of conditions imposed by the k applications (eg., C_{11}/\\$C_{21}$/\\ ... /\\$C_{k1}$ is one such condition). We can define (n_1*n_2* ... *n_k) such fragments.

The above procedure is followed in defining the disjoint horizontal fragments for a relation. Example 4.1 illustrates this procedure for the EMPLOYEE relation.

By performing a union of the appropriate fragments we can form the fragments (derived fragments) reqired by the k applications.

The derived fragment F_{ri} satisfying the condition C_{ri} of the [illegible]ication r.

$$F_{ri} = \underset{m}{\mathrm{UN}}\ F_m$$

Where F_m is a fragment of R having condition ~~C~~$_{ri}$ in its definition, and the union operation is performed over m.

Example 4.1.

There are two important applications using the relation EMPLOYEE.

Application 1 needs two horizontal fragments of EMPLOYEE with C_{11} : DEPT=1, and C_{12} : DEPT=2.

Application 2 needs two horizontal fragments of EMPLOYEE with C_{21} : SALARY>=89, and C_{22} : SALARY<89.

The following four fragments of EMPLOYEE are defined using the conjunction of conditions imposed by applications 1 and 2.

F_1 = tuples satisfying DEPT=1 /\ SALARY<89

F_2 = tuples satisfying DEPT=1 /\ SALARY>=89

F_3 = tuples satisfying DEPT=2 /\ SALARY<89

F_4 = tuples satisfying DEPT=2 /\ SALARY>=89

F_{11} = F_1 UN F_2

F_{12} = F_3 UN F_4

F_{21} = F_1 UN F_3

F_{22} = F_2 UN F_4

4.2 DISJOINT AND NON-DISJOINT FRAGMENTS

There are in general, some limitations in the use of disjoint horizontal fragments. They include:

1. Determining the complete set of 'important' applications and C_{1i}'s at the time of design may not always be possible.

2. For a large number of important applications the number of fragments to be defined is equal to the cross product of the number of fragments due to each application, $(n_1*n_2*...*n_k)$. This large number of fragments increases the problem of global directory maintenance.

3. Whenever new important applications are introduced with non-existing fragmentation criteria, the existing fragments will be disturbed.

These limitations can be overcome, if non-disjoint fragments are permitted. However, handling of non-disjoint fragments introduces its own set of problems. Insertion, update, and deletion are far more involved in a non-disjoint fragmentation scheme. This is because the presence or absence of a tuple has to be checked in a set of fragments(>=1), unlike in the disjoint fragmentation scheme where there is only one effective fragment (replication is of no concern here). For this checking, two different procedures can be used. The first one uses a transfer matrix [Maier and Ullman, 1983] which indicates the tuples in fragment F_i which must also be in fragment F_j. In the second approach universal tuple-id's are used to search the presence or absence of a tuple in different fragments.

The decision to choose overlapping fragmentation depends upon the requirements. There are occasions, however, when one may be forced to handle overlapping fragments; one important situation arises when different centralized databases are integrated to form a distributed database. In such situations one does not have any prior control over the fragmentation scheme and the allocation of these fragments. Thus, it is essential to understand and deal with non-disjoint fragments.

Next we categorize the non-disjointness between the horizontal fragments, and then suggest some query processing strategies suitable for use with non-disjoint fragments.

4.3 CLASSIFICATION OF THE NON-DISJOINT HORIZONTAL FRAGMENTS

Non-disjointness in horizontal fragments can be classified into three basic categories:

1. Random Non-disjointness

Non-disjointness between fragments is random when fragments cannot be specified by 'simple' fragmentation trees. Reconstruction of the relation is only possible by performing the union of all the fragments; the absence of even one of the fragments may make the reconstruction impossible. This type of overlap is difficult to formalize. Every query referring to the relation has to be directed to all the fragments. Insertion can be made in any fragment while deletion and update require accessing every fragment.

2. Non-disjointness due to multiple fragmentaion trees

When multiple fragmentation trees are defined on the same global relation, fragments in one of the fragmentation trees may overlap with the fragments from the other trees. For example in figure 4.1, some of the tuples found in fragment F_1 may also be found in fragments F_3, F_4, and F_5. In this case, global relation can be constructed in more than one way. A given query may be directed to one of the many sets of fragments. For example a query with the selection condition 'DEPT=1 /\ SKILL=A' can be directed to either fragment F_1 or F_3. This type of overlap can be formally stated as

$F_i \cap F_j$ = null

if F_i and F_j belong to the same fragmentation tree

$F_i \cap F_j$ may be non-null

if F_i and F_j belong to different fragementation trees

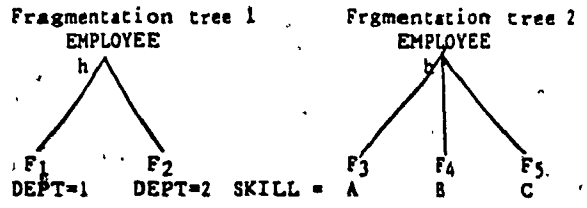


Figure 4.1
Non-disjointness due to multiple fragmentation trees

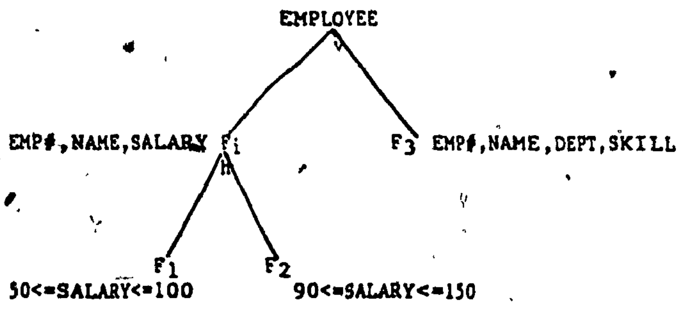


Figure 4.2
Non-disjointness between fragments of the same tree

3. Non-disjointness between fragments of a single fragmentation tree

When the fragmentation predicates of a given fragmentation tree are not mutually exclusive the same tuple may appear in more than one fragment of the tree. Here also, it may be possible to form the global relation in more than one way. Figure 4.2 illustrates non-disjointness of this type; the predicates used in defining fragments F_1 and F_2 are not mutually exclusive; they overlap in the range 90=<SALARY<100.

> $F_i \cap F_j$ may be non-null
> where F_i and F_j are different fragments of the
> same fragmentation tree.

Global relations of a database may be fragmented according to these three types or their combinations. Since type one is difficult to formalize, it is treated separately, and only the other two types of non-disjointness are treated in the next few sections.

Redundancy in a distributed database can be due to two reasons:

1. Replication
2. Non-disjointness

Redundancy due to non-disjointness is more general of the two; replication is 100 percent non-disjointness. Thus, in the rest of the text we deal with redundancy due to non-disjointness alone.

CHAPTER V

QUERY PROCESSING WITH NON-DISJOINT FRAGMENTS

A single variable query Q can, in general, be expressed as [Sacco, 1984]

$$Q=PA_A(SL_C(R))$$

Where R is a global relation,

A is the target list of attributes

and C is the qualification.

In general, C can be expressed in the disjunctive normal form [Bernstein and Chiu, 1981]

$$\bigvee_{i=1}^{M} \bigwedge_{j=1}^{N_i} C_{ij}$$

i.e., disjunction of conjunctions ($C = \bigvee_{i=1}^{M} C_i$)

where each $C_i = \bigwedge_{j=1}^{N_i} C_{ij}$.

C_{ij} is an expression with attributes and constants as operands, and comparison operators as operators(e.g., SALARY <= 100).

The optimal processing of Q, shown to be NP-hard[Hevener, 1979] involves:

1. Finding the set S_i of tuples satisfying the disjunct C_i for all $i=1,2,\ldots,M$.
2. Projecting the S_i's on A to get $S_i(A)$

$$S_i(A)=PA_A(S_i)$$

3. Forming the union of the $S_i(A)$ to give the required result S

$$S = \bigcup_{i=1}^{M} S_i(A)$$

5.1. TARGET FRAGMENTS

In this chapter, we shall present some heuristic techniques which reduce the query evaluation complexity.

<u>Selection Condition(SCON)</u>

$$SCON = \bigwedge_{k=1}^{p} C'_{ik}$$

where $p=1,2,\ldots,Ni$ and $C'_{ik} = C_{ij}$

for some $j,k = 1,2,\ldots,Ni$

i.e., SCON is conjunction of some of the terms in C_{ij}

<u>Target Fragments</u>

A fragment, f (any node in the fragmentation tree) of R, is called a **target fragment** with respect to a given disjunct C_i if it satisfies the following conditions:

C1: A is a subset of Attr(f)

C2: $\exists$ f (f|SCON)/\!(R-f|SCON); ! stands for NOT

i.e., f contains all the tuples of R satisfying an SCON of C_i.

C3: No other fragment in the subtree of f is a target fragment of C_i.

For a given disjunctive term C_i there can be many different SCON's and each SCON can be satisfied by more than one target fragment of C_i. TF_i, represents the set of target fragments of C_i, where

$$TF_i = \{t_f | t_f \text{ is a target fragment of } C_i\}$$

Example 5.1:

Given an EMPLOYEE relation with the fragmentation schemes shown in figure 5.1 and the query Q1

" Get the number, name, dept and salary of employees with skill = 'A' who work either in dept = 1 or dept = 2"

i.e., $Q = PA_A(SL_C(\text{EMPLOYEE}))$

where A = (EMP#,NAME,DEPT,SALARY)

C = (SKILL='A'/\DEPT=1)\/(SKILL='A'/\DEPT=2)

We wish to find the target fragments of EMPLOYEE for the given C.

C_1 = (SKILL='A' /\ DEPT=1)

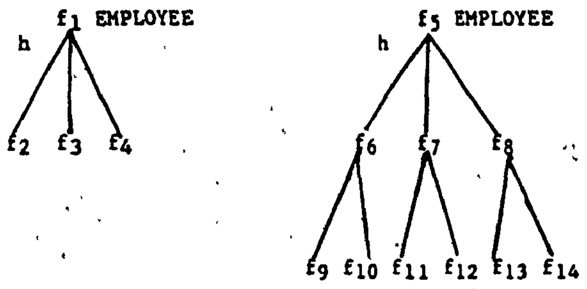
C_2 = (SKILL='A' /\ DEPT=2)

For C_1 the possible SCON's are: DEPT=1,SKILL='A', and C_1 itself; and for C_2 the possible SCON's are: DEPT=2, SKILL='A', and C_2 itself.

fragments f_2 and f_5 satisfy the conditions C1 and C2 for

SCON : (DEPT = 1).

fragments f_1 and f_6 satisfy the conditions C1 and C2 for SCON : (SKILL = 'A').



f_2 : $SL_{DEPT=1}$(EMPLOYEE)
f_3 : $SL_{DEPT=2}$(EMPLOYEE)
f_4 : $SL_{DEPT=3}$(EMPLOYEE)
f_6 : $SL_{SKILL='A'}$(EMPLOYEE)
f_7 : $SL_{SKILL='B'}$(EMPLOYEE)
f_8 : $SL_{SKILL='C'}$(EMPLOYEE)
f_9 : $PA_a(f_6)$
f_{10}: $PA_b(f_6)$
f_{11}: $PA_a(f_7)$
f_{12}: $PA_b(f_7)$
f_{13}: $PA_a(f_8)$
f_{14}: $PA_b(f_8)$

a : (EMP#,NAME,DEPT,SKILL)
b : (EMP#,NAME,SALARY)

Figure 5.1 A non-disjoint fragmentation scheme of EMPLOYEE

fragments f_1 and f_5 violate the condition C3,
thus, fragments f_2 and f_6 satisfy all the conditions for
SCON : C_1,
therefore $TF_1 = (f_2,f_6)$
TF_2 can be computed similarly
and $TF_2 = (f_3,f_6)$
In terms of physical fragments

$TF_1 = (f_2, f_9 \; JN_{EMP\#} \; f_{10})$

$TF_2 = (f_3, f_9 \; JN_{EMP\#} \; f_{10})$

From the definition it is clear that each target fragment contains all the tuples of R satisfying the corresponding SCON. A tuple satisfying C_i has to satisfy each one of its SCON's. So the fragment containing all the tuples satisfying an SCON, also contains all the tuples satisfying the C_i. Thus, it is enough to examine one of the target fragments of TF_i; the choosing of the best target fragment t_f, from TF_i is defferred to section 5.3. Though the number of possible SCON's are (2^{Ni}), the problem of finding TF_i is of the order O(NF). The following theorem proves this.

Theorem 5.1:

Time complexity of finding TF_i for a given disjunct C_i, from the NF fragments of a given relation R (includes all the nodes of the fragmentation trees of R) is of the order O(NF).

Proof:

$$C_i = \bigwedge_{j=1}^{Ni} C_{ij}$$

There are, thus, ${}^{Ni}C_r$ SCON's, each with r conjuncts from C_i, r varies from 0 to Ni.

Number of SCON's

$$= ({}^{Ni}C_0 + {}^{Ni}C_1 + \ldots + {}^{Ni}C_{Ni})$$

$$= (2^{Ni})$$

$$= (2^{Ni})$$

Which is obviously exponential.

We can find all the target fragments of C_i by finding the target fragments with r=Ni. We prove below that SCON with any other value for r is not going to return unique target fragment, which is not returned by the SCON with r=Ni.

On the contrary, let us assume that an SCON of C_i, $SCON_x$, with r=x (<Ni), returns an unique target fragment f' which is not returned by the SCON with r=Ni.

By definition, the f' contains all the tuples satisfying the $SCON_x$.

Since f' satisfies condition C1 for $SCON_x$, it has to satisfy this condition for the SCON with r=Ni.

Since f' satisfies the condition C2 for $SCON_x$, it has to satisfy this condition for the SCON with r=Ni.

Therefore in the subtree f', SCON with r=Ni must have a target fragment.

Since f' satisfies the condition C3, it implies that no other fragment in the subtree f' is a target fragment of C_i.

The last two statements contradict each other.

Therefore either the target fragment returned by the SCON with r=Ni is same as f' or our assumption that $SCON_x$ returns a unique target fragment is wrong.

Therefore it is enough if we check all the fargments of the relation R with SCON of r=Ni to find all the target fragments of the C_i.

Thus the time complexity of finding the target fragments is O(NF).

5.2 QUERY OPTIMIZATION COST MODEL

The cost of processing, TOT, the fragment t_f is the sum of the following

1. Processing cost, (PT): cost of using the CPU.
2. Secondary storage access or I/O cost (IOT): the cost of loading data pages from the secondary storage into main memory.
3. Communication (or transmission) cost (TT) : The

cost of transmitting data from the stored site to the computation site.

These costs are composed of the communications setup cost and delay cost(Section 2.4.4). Each of these costs is further composed of two components:

1. The cost of developing the target fragment, if it is an intermediate fragment or part of an intermediate fragment.

2. The cost of evaluating the S_i's from the target fragments.

5.2.1 Processing Time(PT)

The cost, PTF, of developing the intermediate fragments from the leaf nodes is the cost of performing the requisite union or join operations. Union operation does not take much of processing time, if redundant tuples are not removed from the result. On the other hand, the join operation between fragments takes considerable processing time. Join between vertical fragments f_i and f_j is performed by reading a tuple from one of the fragments, and searching the other fragments to find the tuple which satisfies the joining conditions. This search time constitutes the processing time.

$$t_f = \mathop{\mathrm{UN}}_{i=1}^{H} \mathop{\mathrm{JN}}_{j=1}^{V_i} f_{ij}$$

or $$= \mathop{\mathrm{JN}}_{i=1}^{V} \mathop{\mathrm{UN}}_{j=1}^{H_i} f_{ij}$$

where f_{ij}'s are leaf node fragments, and PTF is the cost of developing t_f then

$$\underline{PTF} = \sum_{k=1}^{J} K_{av} * |f_k| \text{ units of processing time(u.p.t)}$$

Where J is the number of join operations performed in developing the intermediate fragment, assuming all the joins are two-way joins; $|f_k|$ is the cardinality of the fragments involved in the join, usually smaller of the joining fragments; K_{av} is the average search time taken for a tuple.

This assumes that there is a primary structure on one of the pairwise joining domains.

To calculate the cost, PTS, of evaluating S_i's,let us assume that the target fragment, t_f, is common to C_D disjuncts and on the average there are C_V conjuncts per disjunct of C. To evaluate S_i's corresponding to the C_D disjuncts, each tuple of t_f needs to be tested against each of the C_D disjuncts. In the worst case this requires C_D*C_V comparisons per tuple and, could, in the best case require $\min(C_D,C_V)$ comparisons per tuple.

Average number of comparisons/tuple

$= (C_D*C_V + \min(C_D,C_V))/2$

$C_D*C_V/2$

and

$$PTS = |t_f|*(C_D*C_V)/2 \text{ u.p.t}$$

$$PT = PTF + PTS$$

$$PT = \sum_{k=1}^{J} K_{av}*|f_k| + |t_f|*(C_D*C_V)/2 \text{ u.p.t} \qquad ....5.1$$

<u>Example 5.2:</u>

Given the cardinality of the target fragments compute the processing time components of f_2, f_3, and f_6.

Let us assume

$|f_2|=|f_3|=|f_6|=|f_9|=|f_{10}|=1000$; $K_{av}=9$

and join selectivity factor between f_9 and f_{10} = 1.0

$$PTS_6 = |t_f|*(C_D*C_V+\min(C_D,C_V))/2 \text{ u.p.t}$$

$$= 1000*(2*2+2)/2 = 3000 \text{ u.p.t}$$

(Since $C_D = 2$, $C_V = 2$, and $|f_6|=1000$)

therefore $PTS_6 = 3000$ u.p.t

$$PTF_6 = \text{Min}(|f_9|,|f_{10}|)*K_{av}$$

$$= 1000*9 \text{ u.p.t}$$

$$= 9000 \text{ u.p.t}$$

Similarly we can compute the second component of the processing time for the fragment f_2, and f_3.

$PTS_2 = 1500$ u.p.t

$PTS_3 = 1500$ u.p.t

Usually the processing time component PTF is dominated by the I/O time, so it's contribution to the total time or response time can be ignored.

5.2.2 I/O time (IOT)

Usually the I/O times are measured in terms of number of pages [Epstein, 1980] We shall name the same units for the measurement of I/O time(u.i.t). The I/O cost, IOTF, involved in developing the intermediate fragments is the sum of the I/O costs for performing the union, $IOTF_U$, and join, $IOTF_J$, operations.

$$IOTF = \sum_{i=1}^{U} IOTF_{Ui} + \sum_{j=1}^{J} IOTF_{Jj} \text{ u.i.t}$$

Where $IOTF_{Ui}$ is the time taken for the ith union operation

$IOTF_{Jj}$ is the time taken for the jth join operation

U the number of unions required to develop t_f

J the number of joins required to develop t_f

The I/O cost, IOTS, involved in evaluating S_i's from the target fragment t_f is given by

$$IOTS = tp \text{ u.i.t}$$

Where tp is the number of pages in t_f

$$IOT = IOTF + IOTS \text{ u.i.t}$$

$$= \sum_{i=1}^{U} IOTF_{Ui} + \sum_{j=1}^{J} IOTF_{Jj} + tp \text{ u.i.t} \qquad \text{......5.2}$$

Example 5.3:

Given that 100 tuples are stored per page compute the IOT for the target fragments computed in example 5.1 and example 5.3.

$IOTF_{U6} = 0$; no union operation required

$IOTF_{J6} = p_9 + |f_9|$ units of I/O time[Epstein, 1980]

It is assumed that the vertical fragments have primary structure on their joining domain.

Where p_9 is the number of pages in f_9

$= 10 + 1000$ u.i.t

$= 1010$ u.i.t

Therefore $IOTF = 1010$ u.i.t

$IOTS_6 = tp_6 = 10$ u.i.t

$IOT = 1010 + 10 = 1020$ u.i.t

Similarly the I/O time for f_2, and f_3 can be computed

$IOT_2 = 10$ u.i.t

and $IOT_3 = 10$ u.i.t

5.2.3 Communication (Transmission) Cost(TT)

The communication cost is the sum of the communication costs for building the intermediate fragments (TTF) and for transmitting the S_i's to the query originating node (TTS), and are measured in transmission time units (u.t.t).

Fragments would need to be transmitted amongst nodes for evaluating the union and join operations. Epstein[Epstein, 1980] has analysed the various fragment processing strategies and fomulated the communication cost for a broadcast model. Conditions for an optimal data transmission are given. It is sufficient for our purposes to take the cost as proportional to the number of bytes transmitted.

$$TTF = \sum_{i=1}^{U}(C_0 + C_1 * b'_i) + \sum_{j=1}^{J}(C_0 + C_1 * b''_j)$$

units of transmission time(u.t.t)

Where b'_i is the bytes moved across the nodes for the ith union operation in developing t_f, b''_j ~~is the~~ bytes moved across the nodes for the jth join operation in developing t_f.

The communication cost, TTS, taken to transmit the S_i's to the query originating node is

$$TTS = \sum_{i=1}^{C_D} C_0 + C_1 * |S_i| * k_i \text{ u.t.t}$$

If t_f is not at the query originating node

= 0 otherwise

where C_D : number of S_i's evaluated from the t_f

k_i : number of bytes in a tuple of S_i

Example 5.4:

Given that f_9 and f_{10} are at the query originating node; f_2 and f_3 at some other node, compute the TT's for the target fragments.

Selectivities are given:

Selectivity of (DEPT=10) = 1/3

Selectivity of (SKILL=1) = 1/3

Selectivity of (DEPT=1) = 1/3

$C_0 = 0$; $C_1 = 1$

sizes of the various fields of EMPLOYEE in bytes

EMP# : 10; NAME : 20; DEPT : 1; SKILL : 1; SALARY : 10

therefore $k_1 = k_2 = 41$ bytes

f_9 and f_{10} are at the same node

therefore $TTF_6 = 0$

f_6 is also at the query originating node

therefore $TTS_6 = 0$ u.t.t

$TT_6 = 0$ u.t.t

$|S_1|$ = Sel.(DEPT=1)*Sel.(SKILL='A')*|EMPLOYEE|

= 1/3*1/3*3000

≃ 334

$TTS_2 = 334*k_i$ u.t.t

= 334*41 u.t.t

= 13694 u.t.t

$TTF_2 = 0$ u.t.t

$TT_2 = 13694$ u.t.t

Similarly TT_3 can be computed

$TT_3 = 13694$ u.t.t

For a single disjunct in C, the optimal choice of a target fragment is the one with minimum total time. However, when two or more disjuncts have a common target fragment, it may be beneficial to choose the common fragment rather than the optimal target fragments for the individual disjuncts. For example, let TOT_1 and TOT_2 be the respective total costs for processing the optimal target fragments for the disjuncts C_1 and C_2; and TOT_c be the cost of processing the fragment f_c satisfying both the disjuncts C_1 and C_2. The common fragment is preferable if $TOT_c < TOT_1 + TOT_2$.

Theorem 5.2:

The time complexity of finding the optimal set T_{opt} of M target fragments (not necessarily distinct) from TF_i's($1<=i<= M$), choosing not more than one fragment from each TF_i, is non-deterministic polynomial.

Proof:

Let TF_i denote the set of all target fragments for C_i.

Let TF_i be Wi tuple,

say $TF_i = (t_{i1},t_{i2},...,t_{iWi})$

Fixing one t_{ij} in TF_i we have to search for all other M-1 t_{rk}'s, such that r != i. Thus, for each t_{ij} from TF_i we can form W1*W2*...*Wi-1*Wi+1*...*WM combinations of t's.

But TF_i itself contains Wi t's. Hence the total number of combinations to be searched to find the T_{opt}.

$$= W1*W2*...Wi-1*Wi*Wi+1*...*WM$$

Let Wk be min of {W1,W2,..,WM}.

Therefore total number of combinations to be searched to find T_{opt}.

$$>= Wk^M$$

As the searching of T_{opt} is non-deterministic polynomial, we suggest three heuristic approaches which might give suboptimal solutions for T_{opt}. In all these approaches a binary matrix CT representing the relation between the disjuncts and target fragments is manipulated. CT contains M rows and L columns. L is equal to the number of elements in T,

$$\text{where } T = \bigcup_{i=1}^{M} TF_i$$

$$= \{t_1, t_2, \ldots. t_L\}$$

Let TOT_i denote the total cost in processing fragment i of T, Without loss of generality let us assume that $TOT_i < TOT_{i+1}$, $1<=i<=L$.

$$CT(r,c) = 1 \quad \text{if } t_c \in TF_r$$
$$= 0 \quad \text{otherwise}$$
$$1<=r<=M, \ 1<=c<=L$$

5.3. QUERY PROCESSING ALGORITHMS

The first approach attempts to minimize I/O and communication costs by distributing these costs over the maximum number of disjuncts. While the second approach attempts a partial optimization by choosing the minimum of two, intuitively sub-optimal, costs, one of the costs being that obtained in the first approach. In the third approach target fragments which are are having lower average total time are selected. Average total time is computed by dividing the total time of the fragment by the number of ones in its column in matrix CT.

Associated Row

A row, r, is said to be associated with a column, c, if CT(r,c)=1.

5.3.1 Minimization of Costs

The aim is to distribute the evaluation costs over as many disjuncts as possible. The target fragments common to the maximum number of disjuncts are selected by locating the column with the maximum number of associated rows; when there is more than one such column, the column with the lower total time is selected. The target fragment corresponding to the selected column satisfies the disjunctive conditions corresponding to the associated rows. To avoid duplicate evaluation of disjunctive conditions, CT is modified by deleting the selected column and its associated rows. When a row is deleted from CT, the evaluation costs of the remaining columns are affected; thus, after each column deletion, the costs for the remaining columns are recalculated. Any remaining columns with no associated rows are also deleted; this results when all the disjuncts can be evaluated by the already selected fragments. The above procedure of selecting columns is repeated with the modified CT until all rows from CT have been eliminated. The union of the tuples from the selected fragments is the result, S, and the sum of the costs of evaluating the selected fragments is the anticipated evaluation cost.

An algorithm for finding S using this approach is given in figure 5.2. The procedure FIND-COL-MAX1 returns the column number with the maximum number of associated rows and minimum cost. The procedure MODIFY eliminates the given column and associated rows, reevaluates costs for remaining columns and returns the modified matrix, CT, along with a count of the remaining number of rows with target fragments arranged in ascending order of their total time.

<u>Algorithm 5.1:</u>

```
        NROW := M; NCOL := L[illegible]S := null; TOT1:=0;
    {
     While NROW != 0 do
       {
         FIND-COL-MAX1(CT,COL-MAX1);
         for i = 1 to NROW
           {
            if CT(i,COL-MAX1) = 1 then
              S := S UN PA_A SL_Ci(t_COL-MAX1);
           }
         TOT1 := TOT_COL-MAX1 + TOT1;
          /* TOT := PT + IOT + TT - all the components */
          /*   expressed in the same units             */
          /*   TOT1 is the total time to evaluate S    */
         MODIFY(CT,COL-MAX1,NROW);
       }
}
```

Figure 5.2

5.3.2 Partial Optimization of Costs

The previous approach does not exploit fragments with low costs and neither does it check for other possible optimizations. In the scheme presented in this section another set of target fragments with individual minimum costs are selected and the costs of evaluating S using this scheme is compared with the cost obtained using Algorithm 5.1.

The individual minimum cost target fragments are identified by selecting K columns, in turn, from CT, suitably modified after each selection, such that each of the M rows is an associated row of atleast one of the K columns. After each column selection CT is modified as in

Algorithm 5.1. Algorithm 5.2 is shown in figure 5.3.

```
 Algorithm 5.2:

  CT1 := CT ; TOT1 := 0 ; TOT2 := 0 ;
  NROW := M ; NCOL := L; SET1 := null ; SET2 := null;
{
  /* Approach  #1  */
  While NROW != 0 do
  {
    FIND-COL-MAX1(CT,COL-MAX1);
    SET1 := SET1 UN $t_{COL-MAX1}$;
    TOT1  := TOT1 + $TOT_{COL-MAX1}$;
    /* TOT := PT + IOT + TT - all the components */
    /*  expresssed   in the same units           */
    /* Total time to evaluate S using approach 1 */
    MODIFY(CT,COL-MAX1,NROW);
  }
    /* Select individual min. cost target fragments */
  i := 1; NROW := M ; NCOL := L ;
  while NROW !=  0 do
  {
    SET2 := SET2 UN $t_i$ ;
    TOT2  := TOT2 + $TOT_{COL-MAX1}$:
    /*Total time to evaluate S using individual minimum*/
    /*                                        cost set */
    MODIFY(CT1,COL-MAX1,NROW);
  }
  if TOT1 < TOT2 then
      use SET1 to find S
  else
      use SET2 to find S
  /* order of the target fragments in SET1 and SET2  */
 /* is important, as it relates disjuncts           */
/*  with the target fragments                      */
}
```

Figure 5.3

5.3.3 Minimal Average cost disjuncts

Average cost per disjunct of a target fragment, D_{av}, is calculated by dividing the total cost, TOT, of the target fragment by the number of disjuncts to which the target fragment is common.

$$D_{av} = TOT/C_D$$

In the previous approach we selected the target fragments which are common to the maximum number of disjuncts, irrespective of the total cost of the target fragment. This may not assure minimum cost per disjunct. In the present approach we select the target fragments which give the minimum average cost per disjunct. Algorithm 3 shown in figure 5.4 uses this principle to select the target fragments. The selected fragment is used to evaluate the associated disjuncts. To avoid duplicate processing of the disjuncts, CT is modified by deleting the selected column and its associated rows. When a row is deleted the average costs of the remaining column will be recalculated. Remaining column with no associated rows are also deleted; this procedure of selecting the target fragment(Columns of CT) is repeated with the modified CT until all the rows from CT have been eliminated. The procedure MODIFY modifies the average cost per disjunct, instead of the total cost of the target fragment as in the previous approach. FIND-MIN-DAV is a procedure whose inputs are CT and <u>TOT</u> (a vector, with L elements containing total costs of the target fragments corresponding to the L columns of the CT). This procedure finds the column of CT, COL-MIN, corresponding to the minimum average cost per disjunct.

Algorithm 5.3:

```
NROW:=M; NCOL:=L; SET:=null; TOT:=0
{
 While NROW != 0 do
  {
   FIND-MIN-DAV(CT,TOT,COL-MIN);
   SET := SET UN t_COL-MIN;
   TOT := TOT + T_COL-MIN;
   MODIFY(CT,COL-MIN);
  }
 }
Using the target fragments in SET find the S'
```

Figure 5.4

It may be true that a non-optimal selection of fragments has been made, but it should be remembered that an optimal selection complexity is non-deterministic polynomial and the goal here is to reduce the complexity of finding intuitively optimal solutions. When one of the target fragment selected in T_{opt} is the relation R itself, then all other fragments of T_{opt} can ignored, and S can be computed by examining R.

Example 5.6:

Assuming a high bandwidth network where the nominal I/O speed is equal to the data transmission speed, find the total time, TOT, for the target fragments given in the previous example. Assume that the unit of processing time is 10 times smaller than the unit of transmission time (1 u.t.t=10 u.p.t).

Express u.i.t in bytes instead of pages

Number of tuples per page = 100

therefore 1 u.i.t = 100*41 bytes

where 41 is the number of bytes per tuple

$= .41*10^4$ u.t.t

Since I/O time is equal to the data transmission time

$TT_2 = 334*41$ u.t.t

$= .136*10^5$ u.t.t

$IOT_2 = 10$ u.i.t

$= 10*100*42$ u.t.t ; tuple size of $f_2 = 42$ bytes

$= .42*10^5$ u.t.t

$IOT_2 + TT_2 = (.42 + .136)*10^5$ u.t.t

$= .556*10^5$ u.t.t

$= .556*10^6$ u.p.t

$TOT_2 = .556*10^6 + PT_2$ u.p.t

$= .556*10^6 + 1500$ u.p.t

$= .557*10^6$ u.p.t

Similarly we can compute the TOT_3, and TOT_6

$TOT_3 = .557*10^6$ u.p.t

$TOT_6 = 1020$ u.i.t + 9000 u.p.t

$\simeq 1020*10*100*31$ u.t.t ; tuple size of $f_{10}=31$ bytes

$= 32*10^6$ u.p.t

Using algorithm 5.1 and algorithm 5.2 find the T_{opt} for the previous examples.

$TF_1 = \{f_2, f_6\}$

$TF_2 = \{f_3, f_6\}$

$$T = \underset{i=1}{\overset{2}{\mathrm{UN}}}\ TF_i$$

$= \{f_2, f_6, f_3\}$

after arranging the elements in the order of their cost

$= \{f_2, f_3, f_6\}$

L = number of elements in T = 3

M = number of disjuncts in C = 2

$$CT = \begin{array}{|c|c|c|} \hline 1 & 0 & 1 \\ \hline 0 & 1 & 1 \\ \hline \end{array}$$

Algorithm 5.1: When we execute FIND-COL-MAX1 for the first time, it returns COL-MAX1 = 3 and NROW = 0.

therefore $S = PA_A(SL_{C1}(f_6)\ \mathrm{UN}\ SL_{C2}(f_6))$

$TOT1 = TOT_6$ u.p.t

$= 32*10^6$ u.p.t

Algorithm 5.2: Algorithm 5.2 returns SET1, and SET2

SET1 = $\{f_6\}$

SET2 = $\{f_2, f_3\}$

TOT1 = $32*10^6$ u.p.t

TOT2 = $TOT_2 + TOT_3$ u.p.t

$= 1.1140*10^6$ u.p.t

Note: Though f_6 is at the query originating node, it is not efficient to use f_6, compared to using f_2, and f_3. Intuitively working with either fragementations shown in figure 5.1, one would choose either f_6 or f_2 and f_3.

CHAPTER VI

PERFORMANCE STUDY AND CONCLUSION

It is impossible to analytically compare the non-disjoint fragment query processing techniques presented in this thesis and the query processing strategy, for use in disjointed fragmentation environments, presented in the literature[Wong, 1977; Epstein, 1980; Apers, 1983]. We shall, however, use an example to show that the data retrieval can be more efficient in a non-disjoint fragmentation environment then in a disjoint one.

6.1 PERFORMANCE STUDY

Performance of the three algorithms, the solution with disjoint case, and the optimal solution for non-disjoint case are shown in table 6.1 for a set of sample queries given below. The ratio of time taken in the disjoint case and the best non-disjoint case, TOT_d/TOT_{nd}, are also shown in the table. Performance of the algorithms are measured for a sample database with various data allocation and sample queries. The example 6.1 shows how the time taken by an equivalent disjoint case is computed.

Q2 : Find the Employee number, dept, name, and salary where employee skill is 'A' or 'C' and employee belongs to department 2.

Q3 : Find the Employee number, name, and salary where employee name is 'PAUL' or 'JOHN' and skill is 'C'.

Q4 : Find the Employee number,dept, and salary where employee number is 3 and employee name is 'JOHN'

Example 6.1:

Find the total time taken to evaluate S for the query given in example 5.1 for an equivalent disjoint fragmentation scheme.

The equivalent disjoint fragmentation is shown in figure 6.1

- f_{11} and f_{12} are available on the same node
- f_{13} and f_{14} are available on the same node
- $|f_{11}| = |f_{12}| = |f_{13}| = |f_{14}| = 334$
- $|f_2| = |f_3| = 334$
- Query originating node is different from that of f_{11} and f_{13}

From the fragmentation scheme it is clear that

$$S = f_2 \text{ UN } f_3$$

$TOT_2 = IOTF_2 + TTS_2$ u.p.t

$IOTF_2 = p_2 + |f_2|$ u.i.t

$= .32(4 + 334)*10^5$ u.p.t

$= 10.7*10^6$ u.p.t

$TTS_2 = 334*41$ u.t.t

$$= 1.36*10^5 \text{ u.p.t}$$

therefore $TOT_2 \simeq 11*10^6$ u.p.t

similarly $TOT_3 \simeq 11*10^6$ u.p.t

Fragments selected in answering the sample queries and the corresponding total time taken for a particular data allocation.

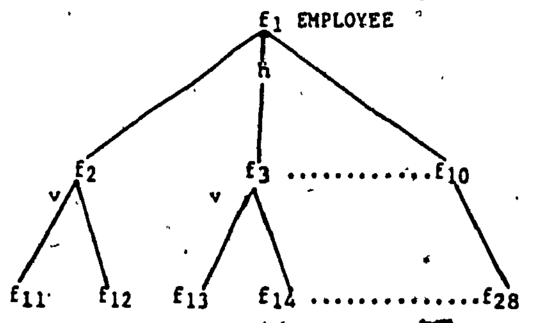
	Q1	time	Q2	time	Q3	time	Q4	time
Algorithm 5.1	f6	32	f3	.140	f10	.140	f1	.800
Algorithm 5.2	f2,f3	.28	f3	.140	f10	.140	f1	.800
Algorithm 5.3	f2,f3	.28	f3	.140	f10	.140	f1	.800
Disjoint ●	f2,f3	22	f3,f9	22	f8,f9,f10	33	f1	99
Optimal - nd	f2,f3	.28	f3	.140	f10	.140	f1	.800
TOTnd/TOTd		80		160		240		124

d - disjoint; nd - non-disjoint

● - fragments of disjoint fragmentation

time measured in units of 10^6 u.p.ts

Table 6.1



f_2 : SL(DEPT=1 /\ SKILL='A')(EMPLOYEE)
f_3 : SL(DEPT=2 /\ SKILL='A')(EMPLOYEE)
....
....
f_{10}: SL(DEPT=3 /\ SKILL='C')(EMPLOYEE)
f_{11}: $PA_a(f_2)$
f_{12}: $PA_b(f_2)$
f_{13}: $PA_a(f_3)$
f_{14}: $PA_b(f_3)$
......
......
f_{28}: $PA_b(f_{10})$

a : (EMP#,NAME,DEPT,SKILL)
b : (EMP#,NAME,SALARY)

Figure 6.1 A disjoint fragmentation of EMPLOYEE

Time taken to evaluate $\mathcal{S}$ = $22*10^6$ u.p.t

This time is approximately 20 times the time taken using the second approach in Example 5.7. In the latter case there is, however, a processing overhead to find the target fragments and to compute T_{opt}. Considering the enormous differences between the two results it would still be beneficial to have a non-disjoint fragmentation scheme. It should also be remembered that in a non-disjoint scheme fragments are defined as required by the concerned applications. In contrast, as shown earlier, with a disjoint scheme where the fragmentation increases exponentially and fragments not required by any application have to exist in the database. The problem of handling multiple copies, a complex topic, though common to both fragmentation schemes may be more complex in the non-disjoint scheme and is beyond the scope of this work.

6.2 MULTIVARIABLE QUERIES

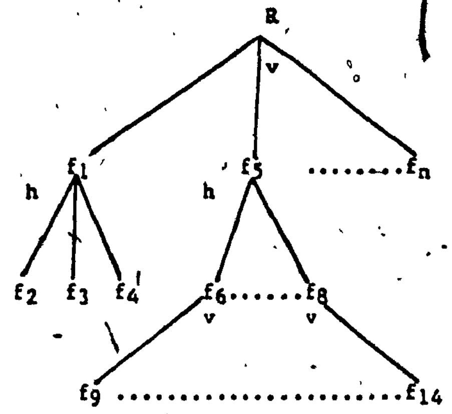
In the last chapter approaches for processing single variable queries were discussed. These strategies can be extended to handle multivariable queries by assuming a universal relation, R, [Toth, Mahmoud and Riordon, 1978; Sacco, 1984] to represent the database. This introduces one more level to the fragmentation tree as the individual relations can be considered to be vertical fragments or clusters of the universal relation R. Figure 6.2

illustrates this principle.
Note: EMPLOYEE1, and EMPLOYEE2 are two vertical clusters of the same relation fragmented further, by different fragmentation criteria.

6.3 CONCLUSION AND FURTHER WORK

This thesis has presented a set of heuristic algorithms for distributed query processing, making use of redundancy in non-disjoint fragments of data. The algorithms are presented for a single variable query, but can be extended to multivariable queries. The performance of these algorithms for multivariable queries is worth further investigation.

The question of finding the optimal set of target fragments is shown to be non-deterministic polymonial. We have verified that for some problems the polynomial. Given by our heuristic algorithms are closer to the optimal solution. We feel that many of the existing problems associated with non-disjoint fragments, can be investigated in the context of the algorithms given in this thesis.



Fragments are defined as in figure 5.1

Figure 6.2 Representation of R

REFERENCES

1. Apers, P.M.G., "Distributed Query Processing, Minimum Response time Schedules for Relations", IR 50, V[illegible]e Universteit, Amsterdam(1979).

2. Apers, P.M.G., "Query Processing and Data Allocation in Distributed Database Systems", PhD. Thesis, Vrije Universteit, Mathematisch Centrum Amsterdam, (1983).

3. Apers, P.M.G., A.R. Hevener, and S.B. Yao, "Optimization Algorithms for Distributed Queries", IEEE Transactions on Software Engineering, SE-9(1), (1983).

4. Astrahan, M.M. and D.D. Chamberlin, "Implementation of a Structured English Query Language", Communications ACM 18(10), pp.580-588, (1975).

5. Baldissera, C., G. Bracchi, and S. Ceri, "A Query Processing Strategy for Distributed Data Bases", Proc. EURO-IFIP 1979, pp. 667-677, North-Holland, Amsterdam, (1979).

6. Bernstein, P.A. and D.W. Chiu, "Using Semi-joins to Solve Relational Queries", Journal of the ACM, 28(1), pp. 25-40, (1981).

7. Bernstein, P.A., N. Goodman, E. Wong, C.L. Reeve, and J.B. Rothnie, "Query Processing in SDD-1 : A System for Distributed Databases", ACM-TODS, 6(4), pp. 602-625,

(1981).

8. Bray, O.H., Distributed Database Management Systems, Lexington Books, 1982.

9. Ceri, S., M. Negri, and G. Pelagatti, "Horizontal Partitioning in Database Design", Proc. ACM-SIGMOD, (1982).

10. Ceri, S., G. Gottlob, G. Pelagatti, "Joining Fragmented Relations in Distributed Databases", Report No. DEPM-83-9, Dept. Of Electronics Polittecnic, Milano, (1983).

11. Ceri, S. and G. Pelagatti, Distributed Databases - Principles and Systems, McGraw-Hill, New York, 1984.

12. Chu, W.W. and P. Hurley, "A Model for Optimal Processing for Distributed Databases", Proc. 18th IEEE Computer, pp. 116-122, (1979).

13. Czarnik, B., S. Schuster, and D. Tsichritzis, "ZETA: A Relational Data Base Management System", Proc. ACM Pacific Regional Conf., pp. 21-25, (1975).

14. Date, C.J., An Introduction to Database Systems, 3 ed., Addision-Wessley, Reading, Mass. 1982.

15. Draffan, W. and F. Poole, Distributed Databases - An Advanced Course, Cambridge Univ. Press, 1980.

16. Epstein, R., "Query Processing Techniques for

Distributed Data Base Systems", PhD. Thesis Memorandum No. UCB/ERL M80/9, Univ. Calif. Berkeley, (1980).

17. Ferrier, A. and C. Stangret, "Heterogeneity in Distributed Database Management System SIRIUS-DELTA", Eighth VLDB, Mexico City, (1983).

18. Held, G.D., M.R. Stonebraker, and E. Wong, "INGRES - A Relational Data Base System", Proc. NCC 44, pp. 409-416, (1975).

19. Hevener, A.R. and S.B. Yao, "Query Processing in Distributed Database Systems", IEEE Transactions on Software Engineering SE-5(3), pp. 177-187, (1979).

20. Hevener, A.R., "The Optimization of Query Processing on Distributed Database Systems", PhD Thesis. Purdue University, (1979).

21. Landers, T. and R.L. Rosenberg, "An Overview of MULTIBASE", Distributed Databases, H.J. Schneider, ed., North-Holland, 1982.

22. Maier, D. and J.D. Ullman, "Fragmentes of Relations", Proc. ACM-SIGMOD RECORD, Vol. 13(4), pp. 15-22, (1983).

23. Narayanan, T.S., P.Goyal, and B.C. Desai, "An Approach to Integrating Heterogeneous Distriuted Database Systems", Proc. ACM-SIGSMAL/PC Conference, Danvers, Mass., pp. 10-17, (1985).

24. Navathe, S.B., S. Ceri, G. Wiederhold, and J. Dou, "Vertical Partitioning for Physical and Distribution Design of Databases", Report No. STAN-CS-82-957, Stanford University, Stanford, (1982).

25. Nguyen Gia Toan, "Decentralized Dynamic Query Decomposition for Distributed Database Systems", Proc. ACM Pacific '80, pp. 55-60, (1979).

26. Paik, I.S. and C. Delobel, "A Strategy for Optimizing the Distributed Query Processing", Proc. First Int. Conf. On Distributed Computing Systems, pp. 686-698, (1979).

27. Pecherer, R.M., "Efficient Exploration of Product Spaces", ACM SIGMOD, pp. 169-177, (1976).

28. Pelagatti, G. and F.A. Schreiber, "A Model of an Access Strategy in a in a Distributed Database", IFIP-TC2, Data Base Architecture, G. Bracchi and G.M. Nijssen eds., North-Holland, pp. 55-71, (1979).

29. Piatetsky-Shapiro, G.I., "A Self-Organizing Database System - a Different Approach to Query Optimization", PhD. Thesis, Courant Institute, New York University, (1984).

30. Rothnie, J.B. and N. Goodman, "An overview of the Preliminary Desing of SDD-1: A System for Distributed Databases", Proc. 2nd Berkeley Workshop on Distributed Data Management and Computer Networks, pp. 39-57, (1977).

31. Rowe, N.C., " Top-Down Statistical Estimation on a Database", Proc. Of Annual Meeting, ACM-SIGMOD, pp. 135-145, (1983).

32. Sacco, G.M., "Distributed Query Evaluation in Local Area Networks", IEEE Data Engineering, pp. 510-516, (1984).

33. Schmid, H.A. and P.A. Bernstein, "A Multi-Level Architecture for Relational Data Base Systems", Proc. Int. Conf. Very Large Data Bases, pp. 202-226, (1975).

34. Selinger, P.G., M.M. Astrahan, D.D. Chamberlin, R.A. Lorie, and T.G. Price, "Access Path Selection in a Relational Database Management System", Proc. ACM Conference, pp.23-34, (1979).

35. Selinger, P.G. and M.E. Adiba, "Access Path Selections in Distributed Data Base Management Systems", Proc. Int. Conf. On Databases, pp. 204-215, (1980).

36. Stonebraker, M.R. and E. Neuhold, "A Distributed Version of INGRES", Proc. 2nd Berkeley Workshop on Distributed Data Management and Computer Networks, pp. 19-36, (1977).

37. Teorey, T.J. and J.P. Fry, Design of Database Structures, Prentice-Hall, 1982.

38. Toth, K.C., S.A. Mahmoud, J.S. Riordon, and O. Sherif, "The ADD system: An Architecture for Distributed

Databases", Proc. 4th Int. Conf. Very Large Data Bases, pp. 462-471, (1978).

39. Toth, K.C., S.A. Mahmoud, J.S. Riordon, "Query Processing Strategies in a Distributed Database Architecture", Proc. 2nd Seminar on Distributed Data Sharing Systems, pp. 117-134, (1981).

40. Ullman, J.D., Principles of Database Systems, 2 ed., Computer Science Press, Rockwell, MY., (1983).

41. Wasserman, A.I. Et al., "Revised Report on the Programming Language PLAIN", SIGPLAN 16(5), pp. 59-80, (1981).

42. Williams, R. Et al., "R* : An Overview of the Architecture", RJ 3325, IBM Research Laboratory, San Jose, Calif., (1981).

43. Wong, E. and K.Youssefi, "Decomposition - A Strategy for Query Processing", ACM Trans. On Database Systems 1(3), pp. 223-241, (1976).

44. Wong, E., "Retrieving Dispersed Data from SDD-1 : A System for Distributed Data Bases", Proc. 2nd Berkeley Workshop on Distributed Data Management and Computer Networks, pp. 217-235, (1977).