1

National Library
of Canada

Bibliothèque nationale
du Canada

Canadian Theses Service

Services des thèses canadiennes

Ottawa, Canada
K1A 0N4

CANADIAN THESES

THÈSES CANADIENNES

NOTICE

The quality of this microfiche is heavily dependent upon the quality of the original thesis submitted for microfilming. Every effort has been made to ensure the highest quality of reproduction possible.

If pages are missing, contact the university which granted the degree.

Some pages may have indistinct print especially if the original pages were typed with a poor typewriter ribbon or if the university sent us an inferior photocopy.

Previously copyrighted materials (journal articles, published tests, etc.) are not filmed.

Reproduction in full or in part of this film is governed by the Canadian Copyright Act, R.S.C. 1970, c. C-30. Please read the authorization forms which accompany this thesis.

**THIS DISSERTATION
HAS BEEN MICROFILMED
EXACTLY AS RECEIVED**

AVIS

La qualité de cette microfiche dépend grandement de la qualité de la thèse soumise au microfilmage. Nous avons tout fait pour assurer une qualité supérieure de reproduction.

S'il manque des pages, veuillez communiquer avec l'université qui a conféré le grade.

La qualité d'impression de certaines pages peut laisser à désirer, surtout si les pages originales ont été dactylographiées à l'aide d'un ruban usé ou si l'université nous a fait parvenir une photocopie de qualité inférieure.

Les documents qui font déjà l'objet d'un droit d'auteur (articles de revue, examens publiés, etc.) ne sont pas microfilmés.

La reproduction, même partielle, de ce microfilm est soumise à la Loi canadienne sur le droit d'auteur, SRC 1970, c. C-30. Veuillez prendre connaissance des formules d'autorisation qui accompagnent cette thèse.

**LA THÈSE A ÉTÉ
MICROFILMÉE TELLE QUE
NOUS L'AVONS REÇUE**

6-315-22241-7

National Library
of Canada

Bibliothèque nationale
du Canada

Canadian Theses
on Microfiche Service

Services des thèses canadiennes
sur microfiches

Ottawa, Canada
K1A 0N4

PERMISSION TO MICROFILM — AUTORISATION DE MICROFILMER

• Please print or type — Écrire en lettres mouées ou dactylographier

Full Name of Author — Nom complet de l'auteur

Peter BODORIK

Date of Birth — Date de naissance

SEPT. 27, 1985

Country of Birth — Lieu de naissance

CZECHOSLOVAKIA

Permanent Address — Résidence fixe

3429 ROWE AVE.
HALIFAX, N.S.
B3L-4C8

Title of Thesis — Titre de la thèse

QUERY PROCESSING STRATEGIES
IN A DISTRIBUTED DATABASE

University — Université

CARLETON UNIVERSITY

Degree for which thesis was presented — Grade pour lequel cette thèse fut présentée

Ph.D.

Year this degree conferred — Année d'obtention de ce grade

1985

Name of Supervisor — Nom du directeur de thèse

Dr. J.S. RIORDON

Permission is hereby granted to the NATIONAL LIBRARY OF CANADA to microfilm this thesis and to lend or sell copies of the film.

The author reserves other publication rights, and neither the thesis nor extensive extracts from it may be printed or otherwise reproduced without the author's written permission.

L'autorisation est, par la présente, accordée à la BIBLIOTHÈQUE NATIONALE DU CANADA de microfilmer cette thèse et de prêter ou de vendre des exemplaires du film.

L'auteur se réserve les autres droits de publication; ni la thèse ni de longs extraits de celle-ci ne doivent être imprimés ou autrement reproduits sans l'autorisation écrite de l'auteur.

Date

Sept. 27, 1985

Signature

x P. Bodorik

QUERY PROCESSING STRATEGIES IN A DISTRIBUTED DATA BASE

by

© PETER BODORIK, M. ENG.

A thesis submitted to the
Faculty of Graduate Studies and Research
in partial fulfillment of the requirements

for the degree of
Doctor of Philosophy

Faculty of Engineering
Department of Systems and Computer Engineering

Carleton University

September, 1985

1985, Peter Bodorik

The undersigned hereby recommends to the Faculty of Graduate
Studies and Research acceptance of this thesis,

"QUERY PROCESSING STRATEGIES IN A DISTRIBUTED DATA BASE"
submitted by PETER BODORIK, M. ENG., in partial fulfilment of the
requirements for the degree of Doctor of Philosophy.

Thesis Supervisor

External Examiner

Chairman,
Department of Systems
and Computer Engineering

Department of Systems and Computer Engineering

Faculty of Engineering

Carleton University

October, 1985

7

ABSTRACT

This research addresses the problem of query processing by joins in a relational Distributed Data Base (DDB). The problem of finding optimal network locations for execution of joins in a given sequence is shown to be polynomial. It is also shown that an attractive approach to the formulation of a query processing strategy is (i) first to find the sequence of joins which minimizes the total size of partial results and then (ii) determine the optimal network locations for their execution. It is shown through modelling that this two-phased approach yields strategies which are close to the minimal delay strategies.

A heuristic algorithm is developed to formulate strategies for queries defined on the distributed view of the DB in which relations can be partitioned horizontally and/or vertically. The algorithm iteratively creates a partial space of strategies represented by a graph consisting of a superposition of binary trees such that each tree corresponds to a unique strategy. Through the algorithm's parameter it is possible to alter the "depth" of the strategy space to be created at each iteration and thus decrease (increase) strategies' execution delay while increasing (decreasing) the delay incurred in their formulation. Consequently, a trade-off between execution time and formulation delay can be obtained. It is shown for a test-bed of queries that (i) the search for optimal strategies is undesirable due to the high strategy formulation delay and (ii) the heuristic algorithm incurs a query response time (combined delay due to formulation and execution) which is close to the execution time of optimal strategies.

1

An algorithm to find optimal strategies for processing queries defined on the distributed view of the DB is developed. One of its properties is that it can be set to minimize the cost of query execution using a number of measures such as a delay due to CPU processing or the dollar cost due to a network data transfer. Optimal strategies are employed in evaluating various cost measures which can be used in optimization of distributed queries. In particular, information which is necessary to examine the effect on the various costs associated with optimizing a particular query execution cost is identified and its use is demonstrated on a test-bed of queries. Nine measures of cost are defined and modelling results indicate that those measures which include costs due to both CPU processing and data transfer are more effective than those which include only CPU processing or data transfer cost.

ACKNOWLEDGEMENTS

I wish to express my appreciation for the support of my family, friends and colleagues during the course of my doctoral programme. In particular, I wish to express my gratitude to my supervisor, Dr. J.S. Riordon, Dean of the Faculty of Engineering, Carleton University, for his invaluable guidance, patience and encouragement without which completion of this dissertation would not have been possible.

I would also like to extend a special thanks to my colleagues at the Technical University of Nova Scotia and in particular to Professor S.H. Heaps, Director of the School of Computer Science, for their ongoing assistance and confidence in me.

The moral support and friendship of Joe and Anna-Marie Wolfert, Dennis MacKinnon, Mark Vigder, John Bryant and Bill Robertson are greatly appreciated. Joe and Anna-Marie are especially thanked for the numerous occasions on which they provided me with a roof over my head.

Last, but certainly not least, I would like to thank my best friend, Elizabeth Lane, for her patience and trust in me throughout the course of my studies.

TABLE OF CONTENTS

	PAGE
ABSTRACT	i
ACKNOWLEDGEMENTS	vii
TABLE OF CONTENTS	iv
LIST OF FIGURES	viii
LIST OF TABLES	ix
NOTATION GLOSSARY	x
 CHAPTER 1. INTRODUCTION	 1-1
1.1 INTRODUCTION	1-1
1.2 PARAMETERS FOR QUERY PROCESSING IN A DDB	1-6
1.3 LITERATURE REVIEW	1-16
1.3.1 APPLICATION OF RESULTS FROM CENTRALIZED QP	1-17
1.3.2 QPSs USING GENERALIZED JOINS	1-20
1.3.3 QPSs USING SEMI-JOINS	1-24
1.3.4 "OTHER" QPSs	1-30
1.3.5 FORMAL ASPECTS OF DISTRIBUTED QUERY PROCESSING	1-34
1.4 THESIS OBJECTIVES AND CONTRIBUTIONS	1-36
1.5 OUTLINE OF FURTHER SECTIONS	1-41
 CHAPTER 2. COMPLEXITY AND TWO-PHASED FORMULATION	 2-1
2.1 INTRODUCTION	2-1
2.2 THE QUERY PROCESSING PROBLEM	2-3
2.3 THE JOIN LOCATIONS PROBLEM	2-12
2.4 DISCUSSION	2-32
 CHAPTER 3. SEARCH FOR OPTIMAL STRATEGIES	 3-1
3.1 INTRODUCTION	3-1
3.2 DEPTH-FIRST SEARCH	3-3
3.3 DECOMPOSITION (DIVIDE AND CONQUER)	3-4
3.4 STRATEGY SPACE GRAPH REPRESENTATION	3-5
3.4.1 INTRODUCTION	3-5
3.4.2 GRAPH REPRESENTATION	3-9
3.4.3 ALGORITHM FOR A GLOBAL VIEW QUERY	3-15
3.4.4 VERTICALLY AND HORIZONTALLY PARTITIONED RELATIONS	3-22
3.4.5 ALGORITHM FOR A DISTRIBUTED VIEW QUERY	3-29
3.4.6 DESTINATION NODE	3-46
3.5 DISCUSSION	3-47

TABLE OF CONTENTS (continued)

	PAGE
CHAPTER 4. HEURISTIC ALGORITHM	4-1
4.1 INTRODUCTION	4-1
4.2 ALGORITHM HEU_DV	4-2
4.3 IMPROVED ALGORITHM HR_HEU_DV	4-11
4.4 NUMBER OF ITERATIONS	4-13
4.5 DISCUSSION	4-13
CHAPTER 5. MODEL FOR QUERY PROCESSING IN A RELATIONAL DDB	5-1
5.1 INTRODUCTION	5-1
5.2 MODEL PARAMETERS	5-8
5.3 DELAY AND DOLLAR COST OF TRANSFERRING RELATIONS	5-10
5.4 COST OF RELATIONAL OPERATIONS	5-11
5.5 CUMULATIVE COST OF PARTIAL RESULTS	5-20
5.6 SIZE OF PARTIAL RESULTS	5-20
5.7 MODELLED APPLICATION OF A DDB	5-31
5.8 MODELLING METHOD AND SOFTWARE	5-34
5.9 SUMMARY	5-38
CHAPTER 6. DDB APPLICATION MODELLING	6-1
6.1 INTRODUCTION	6-1
6.2 COST OF STRATEGIES WITH/WITHOUT OPTIMIZATION	6-8
6.3 COST MEASURE	6-12
6.4 COST MEASURE: TOTAL SIZE OF PARTIAL RESULTS	6-30
6.5 QPS FORMULATOR SELECTION	6-32
6.6 SUMMARY	6-50
CHAPTER 7. SUMMARY AND SUGGESTIONS FOR FURTHER RESEARCH	7-1
7.1 THESIS CONTRIBUTIONS	7-1
7.2 SUGGESTIONS FOR FURTHER RESEARCH	7-4
APPENDIX A	A-1
A-1. REASOURCE ALLOCATION AND CONCURRENCY CONTROL	A-2
A-2. CONJUNCTIVE FORM QUERY	A-7
APPENDIX B. SEMI-JOINS AS A TACTIC IN QUERY PROCESSING	B-1

TABLE OF CONTENTS (continued)

	PAGE
APPENDIX C	C-1
C-1. DEPTH-FIRST SEARCH	C-2
C-2. DECOMPOSITION (DIVIDE AND CONQUER)	C-12
APPENDIX D. CUMULATIVE COST FUNCTIONS	D-1
APPENDIX E. MODELLED DDB APPLICATION PARAMETERS	E-1
E-1. RELATIONS/ATTRIBUTES/DOMAINS	E-2
E-2. INFORMATION PROCESSORS PARAMETERS	E-8
E-3. NETWORK PARAMETERS	E-10
E-4. PARTITIONING AND DISTRIBUTION OF RELATIONS	E-11
E-5. QUERIES	E-14
APPENDIX F. SAMPLE INPUT AND OUTPUT	F-1
REFERENCES	R-1

LIST OF FIGURES

FIGURE NO.	PAGE
1-1 An architecture of a DBMS	1-3
1-2 Processors of ADD	1-24
1-3 Adopted assumptions	1-40
2-1 Query graph and join executions	2-5
2-2 Coalescence of query graph nodes	2-7
2-3 Strategy timing diagrams, a query and two strategies for its processing	2-10
2-- Algorithm LOCATIONS to find optimal network locations for execution of a given sequence of joins	2-23
3-1 Representation of a strategy	3-6
3-2 Two strategies to process $q = (R_1.a \theta_1 R_2.b) \wedge (R_2.c \theta_2 R_3.d)$ and their representation by a graph	3-8
3-3 A partial and a complete graph of strategies	3-11
3-4 Tree representation and a timing diagram of a strategy for w_{26}^R	3-13
3-5 Algorithm to develop the graph of strategies for processing a global view query	3-17
3-6 Modification of a query referencing a vertically partitioned relation by the Distributed View Processor	3-23
3-7 Modification of a query referencing a horizontally partitioned relation by the Distributed View Processor	3-26
3-8 Graph representation of fragmented joins and unions	3-31
3-9 Algorithm to develop the graph of strategies for processing a distributed view query	3-38

LIST OF FIGURES (continued)

FIGURE NO.	PAGE
4-1 First iteration of the heuristic algorithm HEU_DV	4-3
4-2 Second iteration of HEU_DV	4-5
4-3 Partitioned relations processed by HEU_DV	4-7
4-4 Heuristic algorithm HEU_DV	4-10
4-5 Two iterations of the algorithm NR_HEU_DV	4-12
4-6 The non-redundant heuristic algorithm NR_HEU_DV	4-14
6-1 Average delay of strategies formulated by NR_HEU_DV	6-36
6-2 Average delay to formulate strategies by NR_HEU_DV	6-39
6-3 Average query response time delay due to formulation and execution of strategies formulated by NR_HEU_DV	6-41
6-4 Formulation delay as a function of the number of operations	6-47
6-5 Average strategy execution delay and number of operations	6-49
3-1 Example of a depth-first search	3-3
3-2 Depth-first Search algorithm	3-9
3-3 Two identical solutions examined by Depth-first Search	3-11
3-4 Query, its graph, and a timing diagram for one of its strategies	3-13
3-5 Divide and Conquer algorithm	3-16
3-6 Decomposition of a query	3-19
F-1 Input of parameters for the network and information processors specification	F-4
F-2 Sample portion of the input file containing the specification of relations	F-5
F-3 Portion of the input file with the query specification	F-6
F-4 Sample of input commands to control modelling	F-7
F-5 Sample of statistics from modelling a query	F-8
F-6 Sample output of some of the statistics over all queries	F-9

LIST OF TABLES

TABLE NO.		PAGE
1-1	Classification of research on distributed QP	1-18
2-1	Possible join executions	2-20
5-1	Dollar cost and time delay for relational operations and data transfer	5-13
5-2	Cumulative cost functions	5-23
6-1	Average and maximum costs with and without optimization	6-10
6-2	Average cost of strategies optimizing h	6-13
6-3	Maximum cost (in g) of strategies optimizing h	6-18
6-4	Average ratios of cost (in g) of strategies optimizing h	6-21
6-5	Maximum ratios of cost (in g) of strategies optimizing h	6-25
6-6	Standard deviation of ratios of cost	6-28

NOTATION GLOSSARY

CARDINALITY/WIDTH/SIZE

$\text{Card}(R)$... cardinality of a relation R
 $\text{Width}(R)$... tuple length (in bytes) of a relation R
 $s_i = \text{Card}(R_i) \cdot \text{Width}(R_i)$... the total number of bytes of a relation R_i

RELATIONS, ATTRIBUTES, ONE AND TWO-VARIABLE TERMS

R_1, R_2, R_i, R_j ... relations
 $R_i^{(k)}$... horizontal partition/fragment "k" of a relation R_i
 $(k)_{R_i}$... vertical partition/fragment "k" a relation R_i
 $R_i.a$... attribute "a" of relation R_i
 a, b, c ... attributes
 $\theta, \theta_1, \theta_2$... arithmetic relational operators;
 $\theta \in \{=, >, <, \geq, \leq, \neq\}$
 $(R_i.a > k_1), (R_j.b \theta k_2)$... one-variable terms;
 k_1, k_2 ... constants
 V'_1, V'_2 ... one-variable terms
 $(R_i.a \theta R_j.b)$... two-variable term
 V''_1, V''_2 ... two-variable terms
 $(R_i.a \theta_1 k_1) \wedge (R_j.b \theta_2 R_k.c) \wedge \dots$
 ... conjunction of terms, where " $\wedge$ " represents the AND Boolean

operator

QUERY DEFINITION RELATED

q ... a given query "q"

$\{V\}^q$... set of one-variable terms for a query q

$\{V_1 V_2\}^q$... set of two-variable terms for a query q

$(x, c, I)_i$... triplet "i" specifying information about a non-partitioned relation R_i , where

- c_i is the cost (cumulative) of R_i
- x_i is the network location of R_i
- I_i is information necessary for size estimation of partial results

{Note that the cumulative cost includes all processing and data transfer to derive R_i and that the measure of cost (time delay, dollar cost due to CPU processing, etc.) is not specified.}

$(x, c, I)_i^{(k)}$... triplet specifying information about a horizontal fragment $R_i^{(k)}$, where

- $c_i^{(k)}$ is the cost (cumulative) of $R_i^{(k)}$
- $x_i^{(k)}$ is the network location of $R_i^{(k)}$
- $I_i^{(k)}$ is information necessary for size estimation of partial results

$\{(x, c, I)\}^q$... set of triplets containing information about relations referenced by the query q

Z_i ... information processor storing the relation R_i

$W, X, \dots$... network nodes; thus if R_i is stored by an information processor Z_i , connected to the network node W , then $x_i = W$

NUMBER OF ENTITIES

- N_j ... number of joins necessary to process a global view query
- N_o ... number of operations (joins and/or unions) necessary for a distributed view query execution
- N_h ... number of horizontally partitioned relations
- N_v ... number of vertically fragmented relations
- N_q ... number of queries
- n_i ... number of unique strategies to process a query q_i

(Note that a sub-script of "N" is never used as an index, but as a reminder to the reader about the use of the symbol. The sub-script "j" in " N_j " is, therefore, not used as an index but as an indicator that " N_j " denotes a number of joins. The sub-script "i" in " n_i ", however, is an index which specifies that " n_i " denotes the number of unique strategies to process a query q_i .)

RELATIONAL OPERATIONS, DATA TRANSFER AND THEIR COST

$\delta\{(R_i, R_r), Z\}$... restriction(s) on R_i giving R_r , executed by the information processor Z, where all restrictions specified by the query one-variable terms are cascaded and hence inferred by this notation

$\pi\{(R_i, R_r), Z\}$... projection on R_i giving R_r , executed by the processor Z

$\omega\{(R_i, R_r), Z\}$... initial processing (all specified restrictions and a projection to remove attributes of R_i which are neither target nor joining attributes), executed by the information processor Z

$\ast\{(R_i, R_j, R_r), Z\}$... join of R_i with R_j giving R_r , executed by the information processor Z

$U\{(R_i^{(k)}, R_p), Z\}$... union, of all horizontal fragments of R_i giving a non-partitioned relation R_p , executed by the information processor Z , where R_i is horizontally partitioned into $R_i^{(k)}$, $k=1, 2, \dots, m$

$\alpha\{Z\}$... processing of a work unit (byte) by information processor Z

$\phi\{(X, Y), R_i\}$... transfer of a relation R_i from network node X to Y

$\beta\{X, Y\}$... transfer of a unit of data (1 byte) from the network node X to Y

Φ ... denotes one of the relational or data transfer operations listed above

$D[\Phi]$... is the dollar cost of operation Φ

$T[\Phi]$... is the delay of operation Φ

CUMULATIVE COST OF A RELATION

The cumulative cost of a relation R using a specific cost measure is denoted by the following:

$T_a[R]$... cumulative delay due to CPU processing and data transfer

$T_p[R]$... cumulative delay due to CPU processing only

$T_x[R]$... cumulative delay due to data transfer only

$D_p[R]$... cumulative dollar cost due to CPU processing only

$D_x[R]$... cumulative dollar cost due to data transfer only

$D_a[R]$... cumulative dollar cost due to CPU processing and data transfer

$S_p[R]$... cumulative volume of data processed by CPUs

$S_x[R]$... cumulative volume of transferred data

$S_a[R]$... cumulative sum of the size of partial results

NOTATION FOR PRESENTATION OF MODELLING RESULTS

N_q ... number of queries

$q_i, i=1,2,\dots,N_q$... a query i

n_i ... number of unique strategies for processing a query q_i

$A_i^j, B_i^k, j, k=1,2,\dots,n_i$... unique strategies "j" and "k", respectively, for processing the query q_i ; the super-script identifies the strategy to process a query which is identified by the sub-script

$g[A_i^j]$... denotes the cumulative cost of the strategy A_i^j in the measure g , where g represents $T_a, T_p, T_x, D_a, D_x, D_s, S_p, S_x$ and S_a . For instance, assuming that a relation R represents the query q_i result obtained by a unique strategy A_i^j ,

$$\text{if } g=D_a \text{ then } g[A_i^j] = D_a[A_i^j] = D_a[R]$$

represents the cumulative dollar cost due to both CPU processing and data transfer for the query q_i result, relation R , derived by the strategy A_i^j .

h ... is defined similar to g . It also represents any one of $T_a, T_p, \dots, S_a$. If B_i^k is a unique strategy "k" to process a query q_i and $h=T_p$, then

$$h[B_i^k] = T_p[B_i^k]$$

denotes the total time delay due to CPU processing incurred by the strategy B_i^k in processing q_i .

$\hat{A}_{i|g}, \hat{A}_{i|h}, \dots$ strategies, for processing q_i , which minimize the cost in measures g and h , respectively:

$$g[\hat{A}_{i|g}] = \min_{\{j\}} \{ g[A_i^j], j=1,2,\dots,n_i \}$$

$g[\hat{A}_{i|h}] \dots \hat{A}_{i|h}$ is a strategy, to process q_i , which is minimal with respect to the measure h , and $g[\hat{A}_{i|h}]$ denotes its cumulative cost, in the measure g .

$\bar{c}[g] \dots$ is the average cost, in g , over all strategies available for processing all queries, ie.,

$$\bar{c}[g] = \sum_{i=1}^{N_q} \left\{ \sum_{j=1}^{n_i} g[A_i^j] / n_i \right\} / N_q,$$

where the outer summation is for queries $i = 1, 2, \dots, N_q$ and the inner one is for all strategies A_i^j , $j=1, 2, \dots, n_i$, of a query q_i ;

$c_m[g] \dots$ is the cost of the worst strategy, over all queries, for the measure g , ie.,

$$c_m[g] = \max_{\{i,j\}} \{ g[A_i^j]; i=1,2,\dots,N_q; j=1,2,\dots,n_i \} = \max_{\{i\}} \{ \max_{\{j\}} (g[A_i^j], j=1,2,\dots,n_i); i=1,2,\dots,N_q \};$$

$\bar{c}[h,g] \dots$ average cumulative cost, measured in g , over all queries q_i , $i=1,2,\dots,N_q$, such that each query q_i is processed by a strategy which minimizes the cumulative cost measured in h , $\hat{A}_{i|h}$:

$$\bar{c}[h,g] = \sum_{i=1}^{N_q} g[\hat{A}_{i|h}] / N_q$$

$r[i, h, g]$... ratio of the cost (in g) of a strategy $\hat{A}_{i|h}$, which minimizes the cost in h for the query q_i , to the cost of the strategy $\hat{B}_{i|g}$, which minimizes g for q_i , i.e.,

$$r[i, h, g] = g[\hat{A}_{i|h}] / g[\hat{B}_{i|g}]$$

$\bar{r}[h, g]$... average of ratios $r[i, h, g]$, over all queries q_i , $i=1, 2, \dots, N_q$, i.e.,

$$\bar{r}[h, g] = \frac{1}{N_q} \sum_{i=1}^{N_q} r[i, h, g]$$

$c_m[h, g]$... maximum cumulative cost (in g) over all queries such that each query q_i is processed by its best strategy with respect to h , $\hat{A}_{i|h}$:

$$c_m[h, g] = \max_{\{i\}} \{ g[\hat{A}_{i|h}], i=1, 2, \dots, N_q \}$$

$r_m[h, g]$... maximum ratio, over all queries, of the cumulative cost (in g) incurred by a strategy which minimizes h , to the cost (in g) of a strategy which minimizes g :

$$r_m[h, g] = \max_{\{i\}} \{ r[i, h, g], i=1, 2, \dots, N_q \} = \max_{\{i\}} \{ g[\hat{A}_{i|h}] / g[\hat{B}_{i|g}], i=1, 2, \dots, N_q \}$$

$\sigma(r[i,h,g])$... standard deviation of the distribution of ratios $r[i,h,g]$, $i=1,2,\dots,N_q$:

$$\sigma(r[i,h,g]) = \sqrt{\frac{1}{n} \sum_{i=1}^n (x_i - \bar{x})^2 / n},$$

which is the standard deviation formula where

$n=N_q$ is the number of queries;

$\bar{x}$ is the average over all ratios, i.e., $\bar{x} = \bar{r}[h,g]$; and

x_i represents the ratio $r[i,h,g] = g[A_{i|h}] / g[B_{i|g}]$ for the query q_i where, as usual,

$A_{i|h}$... is a strategy, to process q_i , which is best with respect to h and

$B_{i|g}$... is a strategy, to process q_i , which is best with respect to g .

$T_f[i,N]$... time it takes to formulate a strategy to process q_i by the algorithm NR_HEU_DV using a particular value of N which is the number of graph levels to be developed.

$T_e[i,N]$... cumulative execution delay of a strategy, formulated by the algorithm NR_HEU_DV using a particular value of N , to process the query q_i

$T_f[N]$... average delay incurred in formulating strategies by the algorithm NR_HEU_DV, using the value of N appearing in $T_f[N]$, to process queries q_i , $i=1,2,\dots,N_q$, i.e.,

$$\bar{T}_f[N] = \frac{\sum_{i=1}^N T_f[i, N]}{N_q}$$

$\bar{T}_f[N]$... average delay incurred by execution of strategies, formulated by the algorithm NR_HEU_DV with the value of N, to process queries q_i , $i=1, 2, \dots, N_q$, ie.,

$$\bar{T}_e[N] = \frac{\sum_{i=1}^N T_e[i, N]}{N_q}$$

$T_t[i, N]$... denote the total query response time for the query q_i . This is the total delay, due to the strategy formulation and execution, such that the strategy is formulated by the algorithm NR_HEU_DV, using the value of N, to process the query q_i :

$$T_t[i, N] = T_f[i, N] + T_e[i, N]$$

$\bar{T}_t[N]$... denote the average query response time, over all queries q_i , $i=1, 2, \dots, N_q$, such that each query q_i is processed by a strategy formulated by NR_HEU_DV, using the value of N appearing in $\bar{T}_t[N]$:

$$\bar{T}_t[N] = \frac{\sum_{i=1}^N T_t[i, N]}{N_q} = \bar{T}_f[N] + \bar{T}_e[N]$$

$\hat{N}$... the best value of N, from $N=1, 2, 3, 4, 5, 6$ and 7, for use by the algorithm NR_HEU_DV to formulate strategies for all queries q_i :

$$\hat{T}_t[\hat{N}] = \min_{\{N\}} \{ \bar{T}_t[N], N=1, 2, \dots, 7 \}$$

$\hat{T}_t[i]$... optimal response time, for query q_i , which can be achieved using the algorithm NR_HEU_DV. It is the minimal delay, due to both formulation and execution, for strategies formulated by NR_HEU_DV with $N=1,2,\dots,7$, i.e.,

$$\hat{T}_t[i] = \min_{\{N\}} \{ T_t[i, N]; N=1,2,\dots,7 \}$$

$\hat{T}_f[i], \hat{T}_e[i]$... respective formulation and execution components of the total delay $\hat{T}_t[i]$:

$$\hat{T}_t[i] = \hat{T}_f[i] + \hat{T}_e[i]$$

$\hat{T}_t[N^*]$... average query response time such that each query q_i is processed by a strategy formulated by NR_HEU_DV with that value of N which is best for q_i , i.e., which leads to $\hat{T}_t[i]$:

$$\hat{T}_t[N^*] = \sum_{i=1}^N \hat{T}_t[i] / N_q \quad \text{and}$$

$\hat{T}_f[N^*], \hat{T}_e[N^*]$... respective average formulation and execution components of $\hat{T}_t[N^*]$:

$$\hat{T}_t[N^*] = \hat{T}_f[N^*] + \hat{T}_e[N^*], \text{ where}$$

$$\hat{T}_f[N^*] = \sum_{i=1}^N \hat{T}_f[i] / N_q \quad \text{and}$$

$$\hat{T}_e[N^*] = \sum_{i=1}^N \hat{T}_e[i] / N_q$$

CHAPTER 1

INTRODUCTION

1.1 INTRODUCTION

A typical approach to implementation of an information system is the creation of a collection of related files and application programs which maintain and access data stored in files. This approach suffers from an obvious deficiency of data dependence, in that any change in physical organization or layout of files necessitates changes to the application programs as well. In addition, application programmers must, or at least should, concern themselves with problems of integrity, recoverability, and concurrency.

The concept of an integrated Data Base Management System (DBMS) is a step toward the removal of data dependence of application programs and provision of an environment in which application programmers are free from the aforementioned problems. A DBMS provides users with a unified data model; for example relational, hierarchical, or network, together with Data Description and Manipulation languages. The conceptual view (schema) of information about entities and their relationships which are of interest to a particular enterprise/organization are defined by the use of a Data Description language.

Various users belonging to an enterprise which uses a DBMS may need to use only a small portion of the total data and, in many cases, they should be restricted from access to information for which they

lack authorization. For this purpose, users are provided with a particular "view" (sub-schema) of the total information contained in the data base, and a Data Manipulation Language (DML) for access to data as defined by their sub-schema. The mapping of user requests expressed in a DML into operations on the physical implementation of the data base is performed by the DBMS; the physical organization of data is under the control of the Data Base Administrator (DBA) who is provided with a set of tools to ensure efficient access in order to satisfy the, at times contradictory, requirements of various users. The problems arising due to concurrent access, data integrity, and software/hardware failures are handled by the DBMS itself, perhaps under the control of the DBA. A particular architecture (Date, 1982) for a DBMS based on a network data model is shown in Figure 1-1.

Advantages that can be gained by the use of a DBMS are (Date, 1982):

- data independence of application programs is gained;
- data redundancy can be reduced;
- inconsistency can be avoided in some cases;
- data can be shared by various users;
- standards can be enforced by the DBA;
- security can be applied under the DBA's control;
- integrity can be maintained;
- concurrency control for shared access is provided;
- conflicting requirements of various users can be balanced by the DBA; and
- rollback and recovery procedures are provided.

Technological advances in computer networking stimulated great interest in distributed processing for the purpose of sharing resources and data. Arguments of both a quantitative and qualitative nature were put forward in support of, and in opposition to, distribution; the

Figure 1-1 An architecture of a DBMS (adopted from (Date, 1982))

problem became known as "Centralization vs. Decentralization" ((Reynolds, 1977), (Rockart, 1976)). A great interest has also been shown in the application of the concept of a DBMS in a distributed environment supported by a network of geographically separate information processors. As in a centralized DB, users are still provided with a unified data model, together with data description and manipulation languages, while the fact that data itself may be geographically distributed and supported by a network of cooperating information processors is user transparent.

A user-issued request is first analyzed by the local DBMS in which the request originates in order to determine the sequence of operations, which are necessary to process the query and also the network locations in which the operations are to be executed. Instructions are then sent to the appropriate local DBMSs to execute the sequence of operations and perform the necessary transfer of relations.

Advantages that can be gained by the use of a Distributed Data Base (DDB) in comparison to a centralized DB are, ((Rothnie, 1977), (Toth, 1980)):

- increased availability;
- improved response time;
- modularity and upward scaling;
- cost reduction.

To realize the aforementioned benefits, problems that also arise due to the data distribution must be solved. The problems are caused by the fact that:

- information exchange over a network leads to unpredictable delays;
- one information processor does not know the exact state of processing at other network nodes;
- communication, or any information processor, can fail at any time;
- in a point-to-point network, delivery of a set of messages in the sequence in which they were injected into the network can only be guaranteed between two information processors.

Toth (1980) groups existing technical problems in a DDB into categories of Database Structure and Organization, and Operation and Control. The second category, augmented by the Resource Allocation problem, contains the following problem areas:

- A. Resource Allocation (RA)

- B. Concurrency Control (CC)
- C. Query Processing Strategies (QPS)

Although this thesis deals with the topic of distributed query processing, a brief discussion about the Resource Allocation and Concurrency Control problems is included in Appendix A. The aim of the discussion is to outline the problems and thrust of current research efforts in these two areas.

A user specified query issued against a DDB is initially transformed either by a query processor or a pre-compiler, into relational algebra or relational calculus. In either case, the query is answered by execution of relational operators (selection, projection, join, union, etc.) on relations specified by the query. Since the ordering of operator execution is not fixed and relations are distributed over the network of information processors, the QPS formulator must determine, prior to the query execution, the sequence in which operators are executed, the location of their execution, and, of course, the movement of data/relations among processors. Formulation of the QPS comprises the query "strategy formulation" phase.

Once the QPS is determined, instructions are sent to local DBMSs for operator execution and transfer of data. The cooperation of local DBMSs by information exchange over the network is necessary to process the query. The strategy's execution comprises the "query execution" phase.

4.2 PARAMETERS FOR QUERY PROCESSING IN A DDB

To facilitate an appropriate frame of reference for discussion of the QPS formulators proposed by various researchers thus far, it is necessary to discuss several key issues which will also serve as parameters in categorizing the research on distributed query processing.

A. Environment

The design of a QPS formulator is targetted for the specific environment in which it operates. The specific parameters identifying the operational environment are discussed below.

a) Data Model (Relational)

The majority of work on DDBs appears within a framework of the Relational Data Model. This work also concentrates on a Relational DDB and, therefore, the remainder of this thesis assumes the Relational Data Model only. Query processing optimization in a DB based on hierarchical or network (DBTG) data models are not considered or discussed.

b) Data Base Structure and Organization

Three types of DB organizations can be assumed in relation to the design of a CC mechanism:

(i) All CC is performed by one information processor (Menasce, 1979). All requests for retrieval or update are submitted to this one processor which is in control of operations at all times. This is an uninteresting case which is no longer discussed in the literature,

since the one controlling processor poses a threat to reliability, integrity, and acts as a bottleneck as well.

(ii) Information processors distributed over the network form a virtual ring in which retrieval and update is controlled by circulation of a control token (LeLann, 1977, 1978, 1980).

(iii) Retrieval and update is assumed to function under distributed control as, for example, in ADD ((Mahmoud, 1979), (Toth, 1980)), INGRES (Stonebraker, 1976), or SDD-1 (Bernstein, 1979).

c) Network

Networks can be grouped into three major categories; point-to-point (e.g., ARPANET, DATAPAC), broadcast (e.g., ETHERNET), or star network. A private network supported by leased lines can be considered to be an appropriate subset of one of the above categories.

Most QPS formulations are assumed to exist in a network environment in which the exchange of information from one node to another is assumed without reference to the medium used for data transfer. An additional assumption is that the cost of communication between any two points can be estimated. Therefore, by default, a point-to-point network is assumed.

If a broadcast type network is assumed ((Yu, 1982a, 1982b), (Gouda, 1981)), its property, in that the same data may be transferred to many nodes simultaneously without an increase in data transfer cost, can be exploited in processing of a distributed query; however, parallel transfer of different data elements is not available.

A DDB supported by a star network has a particular architecture in which the information processor in the hub of the network is usually the most powerful processor in the system. It is particularly suited to perform the function of concurrency control and QPS formulation and monitoring. This architecture can be exploited in processing of distributed queries. (Kerschberg, 1982).

d) Existence of Fragmented Relations

In a relational DDB relations may be fragmented vertically and/or horizontally (Mahmoud, 1979). Horizontal fragmentation partitions a relation into subrelations according to a value of a partitioning key. All subrelations have tuples with mutually exclusive keys and identical attributes. Vertical fragmentation, on the other hand, is such that the key attribute(s) are replicated in all subrelations, but the remaining non-key attributes are mutually exclusive.

It will later be shown that vertical fragmentation can be easily handled by additions of equi-joins into the query which references vertically fragmented relations. Existence of horizontally fragmented relations, however, necessitates the use of a union operator, which complicates processing of a relational calculus query expressed in the conjunctive form. Most QPS formulators do not assume the existence of horizontally fragmented relations.

B. Query Form

As was mentioned earlier, a user query is transformed into either relational algebra or relational calculus. A relational algebra query is an expression which indicates the sequence of algebraic operators to be applied to DB relations and can be represented by a parse tree ((Smith, 1975), (Chu, 1982)). Relational algebra theory offers transformation laws by which the query can be transformed into an equivalent query by restructuring its parse tree or, equivalently, by manipulation of the algebraic expression. These laws form the basis for query optimization.

A relational calculus query simply expresses the conditions tuples must satisfy in order to be included in the query result. Although relational algebra and calculus are equivalent in that for any query expressed in relational algebra there exists an equivalent query in relational calculus and vice-versa (Ullman, 1982), research has concentrated on calculus query optimization. The reason for this is that relational calculus is particularly suited for a query language such as SQUARE, QBE, QUEL, SEQUEL and SQL (Ullman, 1982).

The optimization of a QPS for a relational calculus query has been concentrated on a query expressed in a restricted CONJUNCTIVE FORM ((Smith, 1975), (Ullman, 1982)). Informally, it is a sequence of formulas/terms concatenated by AND Boolean operators (hence conjunction). For example, if R_i are relations where $i=1,2,\dots$; $a,b,c,\dots$ are attributes; $k_1, k_2,\dots$ are constants; and $\theta_1, \theta_2, \dots$ are arithmetic relational operators, then a conjunctive form query is:

$$(R_1.a \theta_1 R_2.b \theta_2 R_3.c \theta_3 \dots) \wedge \dots \wedge (R_4.d \theta_4 R_5.e \theta_5 \dots),$$

where " $\wedge$ " represents the AND Boolean operator.

Moreover, each term having more than one arithmetic relational operator can be transformed into a conjunction of terms in which each term has exactly one arithmetic relational operator ((Smith, 1975), (Ullman, 1982)). A query is therefore a conjunction of one (restrictions) or two-variable terms, for instance:

$$(R_1.a > R_2.b) \wedge (R_2.c = R_3.d) \wedge (R_3.e < 4).$$

Appendix A contains a mapping of the informal representation of a query, used herein, to the formal definition of a predicate calculus query presented in (Ullman, 1982) and (Date, 1982).

The conjunctive form query, also referred to as a SELECT-PROJECT-JOIN query, is an important class of queries for two reasons. First, its form is not too restrictive as it is sufficient to express many user queries (Ullman, 1982). Secondly, it greatly simplifies the problem of QPS optimization. Only one two-variable operator, generalized join, together with one-variable operators of restriction and projection are required to process a query expressed in conjunctive form. This is under the assumption that horizontally fragmented relations are not considered as their processing necessitates the use of a union in addition to the select, project and join operators.

C. Cost Objective

Three cost functions which have traditionally been used in analyses of DBs and Distributed Information systems may also be adopted for QPS optimization:

(i) Communication Cost - monetary cost which depends on the volume of all data transmitted due to the query processing.

(ii) CPU Processing Cost - monetary cost of CPU processing for a query in all participating network nodes.

(iii) Query Response Time - total time elapsed from query submission to delivery of the response. Optimization of the query response time is usually simplified by making an assumption that one of the time delays, communication or CPU processing, is dominant and then neglecting the other.

It is interesting to note that although the query response time is claimed to be optimized by authors of various QPS formulators, in most cases the cost that is actually minimized is the volume of data either transferred over the network or processed by the CPUs. For example, in an environment of high capacity processors which communicate by use of standard services provided by a public network, the communication delay may be considered to be the most critical factor. Minimization of the total volume of data transferred over the network should also decrease the query response time.

D. Optimal vs. Sub-optimal Strategy

A QPS can be formulated by an algorithm which finds the optimal strategy or by a heuristic algorithm which finds a good but sub-optimal strategy. A heuristic algorithm decreases the cost of a query's formulation phase at the expense of an increase in its execution phase.

E. Static vs. Dynamic Strategy

A static QPS assumes a priori knowledge of the execution cost of different sequences of the query's relational operators. Since a static strategy is based on a priori cost estimates of partial results, an integral part of a static QPS should be some form of "threshold" mechanism (Toan, 1981). Should the cost of partial results in the execution phase deviate too much from the estimates established in the formulation phase -- beyond a certain threshold (ie., when a threshold is broken) -- that portion of the strategy which is yet to be executed should be redefined/reformulated. Establishment of thresholds for partial results, monitoring of the execution phase to detect a broken threshold, and reformulation of the remainder of the strategy comprise the "Threshold Mechanism".

A dynamic QPS avoids a priori cost estimation as much as possible. A complete QPS is not formed prior to the execution phase but it evolves with the processing of relational operators. After each operator is processed, giving a partial result of known cost, a decision is made on which operator(s) is to be executed next and the location of its execution. Redundant processing and/or communication may be introduced so that an optimal QPS, which minimizes a given cost objective, is achieved. If the cost objective is the total query response time, both redundant processing and communication may be introduced to gain benefits of parallelism. Similarly, redundant processing may be introduced to minimize communication volume and redundant communication may be introduced to minimize the CPU processing cost.

A synthesis of both static and dynamic strategies may be the best approach. In static strategies based on a priori cost estimates, dynamics may be introduced with the use of the previously described threshold mechanism. Similarly, dynamic strategies use some form of a

priori estimation, an element of a static approach.

F. Single Query vs. Multi Query Environment

Optimization of a processing strategy has thus far been concentrated on a single query only; the existence of other queries within the system has been ignored. This is a restriction in the sense that a query competes with other queries or applications co-existing within the environment for both access to data items and resources (processing power, network, etc.).

Assume that the cost of a query is defined to be the execution delay of the strategy used for its processing. If a single query environment is assumed, optimization minimizes the strategy execution delay for a given query. Other queries and applications, which may create a temporarily high CPU processing and/or communication load, are ignored even though they may delay the query under consideration. In addition, an optimal strategy for each individual query does not result in optimal system performance -- instead of minimization of the response time for each individual query the objective should be the minimization of the AVERAGE RESPONSE TIME FOR ALL QUERIES (Cellary, 1980). As will be shown in Chapter 2, the formulation of an optimal strategy for a single query is a difficult problem (NP-hard) and adapting it for a multi-query environment further increases its complexity. Consequently, only when the QPS formulation for a single query has been mastered attention should be turned to the multi-query environment.

Note that if the measure of cost is defined in terms of the communication and/or CPU processing volume, the multi-query environment is irrelevant. The total volume of data transferred and/or processed for an individual query is independent of the co-existence of other

queries or applications; the cost minimization for each individual query results in optimal performance in processing all queries issued against the DDB. For example, minimization of the total volume of data transferred over the network for each individual query also minimizes the total volume of data transfer over all queries.

G. Load balancing

Competition for resources can be viewed as internal or external. Internal loading is created by the queries' demand for resources. It can be "controlled" by a DB processor and hence by a QPS formulator. It is within the realm of a QPS to move data to the node of its choice and thus choose a node with a light load. External loading, on the other hand, arises due to the competition for resources by other applications co-existing within the environment. It cannot be controlled by the DB processor.

Research on QPS optimization has thus far allowed only a limited choice of the location of operator execution. A one-variable operator should not be moved to a different network node for its execution. As most of the time it reduces the size of its operand, it is executed at the network node where its operand resides. A two-variable relational operation (e.g., join, union) is assumed to be executed in the network node of one of its operands. The movement of all operands into some other, lightly loaded node is not considered for a single query optimization.

H. Size Estimation

A static QPS assumes a priori knowledge of the execution costs of different sequences of relational operators. The a priori cost of processing and data transfer can be determined with acceptable degree of accuracy, given that the size of data elements is known. The critical, and very difficult, parameter to estimate is the size of temporary relations (partial results) formed during query execution. Even if accurate knowledge of the cardinalities of relations in a distributed environment is assumed, once a relational operator is executed, a priori knowledge of the partial result size cardinality is not possible. Consequently, the size of partial results must be estimated for the purpose of a QPS formulation. A size estimate is based on the knowledge of cardinalities of operand(s) of the operation under consideration, expected distribution of the attributes of a relation on which the operation applies, and, a selectivity function ((Demolombe, 1980), (Toth, 1980), (Epstein, 1980), (Wong, 1982), (Christodoulakis, 1983, 1984a, 1984b), (Gardy, 1984)). The selectivity function determines the cardinality of the resulting relation based on the information listed above.

To keep this research within reasonable bounds, size estimation is not investigated. Formulators suggested thus far simply assume that a subsystem for size estimation exists which, if given a one or a two-variable term with information on the cardinality and tuple sizes of specified relations, will return an estimate of the size of the resulting relation if that term is processed. This is a typical approach as QPS formulators should not exploit any assumptions made on the method of estimation of the size of partial results. If such assumptions made by a QPS formulator do not hold for a particular application of a DB, the formulator's optimization is no longer valid. For example, assume that for some DB application the size estimate for an equi-join of relations R_i and R_j which results in relation R is obtained with acceptable level of accuracy by

$$\text{card}(R) = \min \{ \text{card}(R_i), \text{card}(R_j) \}.$$

If the QPS formulator utilizes the knowledge of how this estimate is made, the strategy formulation becomes application dependent. The QPS formulator does not create optimal QPSs for DDB applications in which the aforementioned estimation is invalid.

I. Redundant Data Elements

Redundant data elements (relations) may be created to increase the reliability and/or availability of data (Rothnie, 1977). The number of redundant copies may be changed periodically in order to respond to changes in the profile of transactions. Optimization of distributed queries uses multiple copies in two ways:

- (i) first a minimum number of sites required to process a query is determined before a QPS is formulated; and
- (ii) it is assumed that the "closest" copy of a data element is always accessed.

1.3 LITERATURE REVIEW

Although the section 1.2 identifies many parameters for distributed query processing, the scientific literature on distributed query processing can be broadly categorized into works on:

- application of solutions for centralized query processing;
- QPSs using generalized joins;
- QPSs using semi-joins;
- "other" QPSs; and

theoretical aspects of distributed query processing.

The review is performed by categories. Assumptions and objectives of each work discussed below are summarized in Table 1-1.

1.3.1 APPLICATION OF RESULTS FROM CENTRALIZED QP

Optimization of QP in a centralized DB ((Smith, 1975), (Wong, 1976), (Youssefi, 1979), to list a few) has led to a generally accepted set of rules (Ullman, 1982) which apply to a DDB as well, and which are assumed herein. The rules are:

- (i) One-variable operators (restrictions and projections) should be performed as early as possible as they usually reduce the size of relations which are operands of more costly two-variable operators. Note that a condition of the form " $R_1.a \theta R_1.b$ " contains only one tuple variable and it is a one-variable term which is classified as a restriction. As it reduces the size of its operand it should be executed as early as possible.
- (ii) Where applicable, one-variable operators should be performed (grouped/cascaded) together. Obviously, if there are several restrictions to be applied on one relation these should be applied simultaneously in one pass through the relation. For example, " $(R.a > 2) \wedge (R.b < 4)$ " specifies two restrictions on a relation R. Clearly, both of these restrictions can be applied simultaneously on one pass through the relation with only an additional negligible cost of a comparison per tuple. Simultaneous execution of such one-variable terms is called a cascaded one-variable term.
- (iii) Similarly, if there are several two-variable terms specified for the same two relations, these should be processed simultaneously by a

TABLE 1-1 CLASSIFICATION OF RESEARCH ON DISTRIBUTED QP

REFERENCE/ AUTHOR	QUERY FORM ALGEBRA/CALC	DDB ORG.	NET- WORK	HORIZ. FRAG./ MULTI COPY	STATIC/ DYNAMIC	SINGLE/ MULTI- QUERY
Wong (76) Youssefi(79) Epstein(78,80)	calculus conj.norm.form	INGRES	p-to-p/ br.cast	yes/ no	static	single
Toth(80,82) Mahmoud (79)	conj.norm.form	ADD	p-to-p	no	static	single
Daniels (82) Selinger (80)	conj.norm.form	System R (star)	local br.cast	yes/ no	static	single
Hevner(79,80) Ayers (83) Perizzo (84)	conj.norm.form		p-to-p	no	static	single
Toan (79,81)	conj.norm.form	MICROBE vir.ring	br.cast	no =	static + threshold	single
Bernstein(81)	conj.norm.form (envelope)	SDD-1	p-to-p	yes/ no	static	single
Chen, Li (83)	conj.norm.form		p-to-p	no/no	static	single
Black (82)	conj.norm.form			no	static	single
Yu (82a,82b, 83)	conj.norm.form		p-to-p	no/ yes	static	single
Krishna (84)	conj.norm.form		p-to-p	no	static	single
Valduries (84)	conj.norm.form	SABRE		no	static	single
Kambayashi(84)	conj.norm.form			no	static	single
Pelagatti (79)	conj.norm.form		p-to-p	yes/yes	static	single
Paik (79)	conj.norm.form		p-to-p	no/yes	static	single
Cellery(80)				no	static	multi
Kerschberg (82)	calc.- disjun. norm. form		star	no/yes	static	single
Chu (1979) Chu (1982)	algebra		p-to-p	horiz. frag.	static	single
Gavich (1982)	calculus			yes/no	static	single
Daniell(83), Wong(83)				no/yes	dynamic	single
Maier (83)				yes/no		
Ceri (83)						
Christo (83, 84a, 84b), Gardy (84) Toth (80), Epstein (80), Wong (82)			size estimation for partial results of relational operators			
Pramanik (84), Willard (84) Ibaraki (84), Ullman(82), Smith(75)			opt. of QP in a centralized DB (complexity results for sub-problems)			

TABLE 1-1 Continued

REFERENCE/ AUTHOR	OPT./ HEUR.	COST	JOIN/ SEMI-J.	METHOD
Wong (76) Youssefi(79) Epstein(78,80)	heur.	xmit vol.	join	decomposition of queries into "ir-reducible" sub-queries followed by sequencing and opt. sub-query processing
Toth(80,82) Mahmoud (79)	heur.	xmit vol.	join	decomposition of queries in "class A" sub-queries executed optimally + heur.
Daniels (82) Selinger (80)	heur.	delay	join	operator tree examination "bottom up" to evaluate alternatives; consolidation of hor.frag. relation before a join
Havnes(79,80) Aperi (83) Perizzo (84)	heur.	xmit vol.	semi-j.	opt. semi-join sequencing for "simple" queries; adapted for general queries; post-optimization for improvement
Toan (79,81)	heur.	xmit vol.	semi-j.	semi-join reduction for each relation + updatable thresholds
Bernstein(81)	heur.	xmit vol.	semi-j.	semi-join sequence is determined by iterative hill-climbing algorithm
Chen, Li(83)	heur.	xmit vol.	semi-j.	accumulation of distinct values for joining attributes; reduction
Black (82)	heur.	xm.vol.	semi-j.	heur. to determine the semi-join program
Yu (82a,82b, 83)	heur.	xmit vol.	semi-j.	several fold static opt.; site location, semi-join prog. + improvements
Krishna (84)	heur.	xm.vol.	semi-j.	bucket semi-joins for partial reduction
Valduriez (84)	heur.	xm.vol.	semi-j.	semi-j. prog. for multiproc. DB machine
Kambayashi(84)				using tempor. funct. dependencies in QP
Pelagatti (79)	opt.	xmit vol.	join	calc. min. transm. requirements given a seq. of joins; int.prog.model
Paik (79)		xmit vol.	join	two-phase opt. in a highly redundant DB; site-location to min. "logical" distance
Cellery(80)	heur.	ave.resp.time of all queries		dynamic scheduling of CPU operations and data transfer
Kerschberg (82)	heur.	disk access + xmit vol.		separate modelling of disk access; local vs. remote decision making
Chu (79, 82)	opt. exh. or CPU search	xm.vol vol.		permutation of algebraic operators
Gavich (1982)	heur.	xm.vol		optimization of a "set query" processing
Daniell(83), Wong(83)				dynamic rematerialization (duplication)
Maier (83)				theory for horizontal fragmentation
Ceri (83)				proving QPS to be correct by equivalence

cascaded join.

An additional well accepted tactic which involves initial execution of all local operations ((Wong, 1976), (Mahmoud, 1979), (Epstein, 1978), (Chu, 1982)) was derived from the rule (i) listed above. Those two-variable operations, which reduce the size of intermediate results and can be executed without any transfer of data, are executed first. For example, if a query contains a two-variable term " $R_1.a = R_2.b$ ", such that attributes "a" and "b" are keys for relations R_1 and R_2 respectively, then if the two relations are located in the same network node, the join of R_1 with R_2 should be executed locally. As the size of the join result is likely to be of a smaller size than the size of either R_1 or R_2 , the cost of further processing and data transfer is reduced. The tactic is particularly suitable for strategies which minimize the volume of transferred data.

1.3.2 QPSs USING GENERALIZED JOINS

One of the earliest works on QPS optimization in a centralized DB was done by Wong and Yossefi (1976) for INGRES relational DBMS. Their strategy consists of (i) a decomposition phase and (ii) a sub-query sequencing phase. The purpose of the decomposition is to reduce a conjunctive form query into a unique sequence of one-variable and irreducible two-variable sub-queries. They show that most of the time it is profitable to perform one-variable queries as soon as possible (rule (i) above). A two-variable query is reduced into a set of one-variable queries by a tuple substitution. In other words, one tuple is taken from one relation at a time and applied to the other relation, thus creating a one-variable query. A two-variable query is in this way transformed by a tuple substitution into a sequence of one-variable queries.

The objective of the decomposition is to divide a query into a unique sequence of smaller sub-queries which can be processed by tuple substitution. Informally, two-subqueries are irreducible if they cannot be further decomposed into components which overlap in one relational variable only. The sequence in which the results of sub-queries should be joined is not optimized; optimization concentrates on the variable to be chosen for the tuple substitution for each sub-query. The method was later evaluated on a test bed of queries and positive results were reported in (Youssefi, 1979).

In a network environment, the tuple substitution method to process irreducible sub-queries may lead to unacceptable delays. To alleviate this problem, Epstein, Stonebraker, and Wong (1978) abandon tuple substitution and suggest that a feasible strategy for an irreducible sub-query is to be created first and then improved by a "greedy" algorithm. The feasible strategy moves all results of one-variable processing to one node where they are processed. This feasible strategy is improved upon by a "greedy" algorithm. Recall that a sub-query is irreducible and that it consists of a few two-variable terms. Each term is examined and it is decided to process it by a join if the size of its result is smaller than the volume of data transmitted in order to process it.

The final step in optimization consists of the site selection for join processing and equalization of the load on processors in which operands are located. For example, if a join is to be performed between a horizontally partitioned relation and a non-partitioned relation, a decision is made whether the union of the partitioned relation should be performed first or whether the non-partitioned relation should be joined with each fragment and the results consolidated by a union. Moreover, tuples may be moved from one horizontal fragment to another in order to balance the load on the local CPUs. For a DDB supported by a high band-width local network it

is assumed that an increase in the network traffic, in order to achieve a higher degree of parallelism in CPU processing, will improve the query response time.

Optimization in the last step discussed above is performed for both a point-to-point and a broadcast network. The objective is either minimization of the volume of transferred data or CPU processing time which is approximated by the maximum CPU time delay.

The aforementioned algorithm was later evaluated by an experiment (Epstein, 1980) in which several key issues were investigated. The cost metric used was the number of bytes transferred during query processing. Two conclusions were offered: (i) good size estimates are crucial, and (ii) a limited search through the space of strategies for each irreducible sub-query performs poorly, suggesting that the search for optimal strategy is desirable.

Daniels et al. (1982) describe processing of queries in System R*, which is a distributed version of the System R relational DBMS, developed by IBM Research Laboratory at San Jose. System R* is intended for environment in which the communication is supported by a local high band-width network. The CPU processing delay is, therefore, considered to be significant. Optimization concentrates on evaluation of possible alternatives for concurrent transfer and CPU processing in a join execution.

Although System R* does not support fragmentation of relations, Selinger and Adiba (1980) propose a method for optimizing System R* queries using a union operator in addition to the restriction, projection and a join. Their method can be used for processing horizontally partitioned relations. A query is represented by a tree of relational operators. Optimization analyzes the operator tree bottom up and selects the best alternative for each sub-tree; however,

a join involving a horizontally fragmented relation is always preceded by a union in order to consolidate fragments.

A different approach to processing of distributed queries was taken for Architecture for Distributed Databases (ADD) ((Mahmoud, 1979), (Toth, 1980, 1982)). In (Mahmoud, 1979) a system architecture for DDBs is presented together with a formal representation of distributed queries by graphs. In addition, an outline of a processing strategy is given. In (Toth, 1980), a strategy for processing "global view" queries based on decomposition of queries into sub-queries is proposed. A global view query is a query expressed on the global view of the DDB in which relations are not partitioned horizontally or vertically. The proposed decomposition differs from that utilized in INGRES in that it is based on rules for permutation of relational operations (projection, restriction, generalized join) and it strives to decompose a query into a set of special type queries called "Class A" queries. A Class A query is actually a superset of a "simple" query defined by Hevner and Yao (Hevner, 1979, 1980).

Class A sub-queries are processed first by an optimal strategy, thus reducing the size of intermediate results which are processed by a sub-optimal heuristic. The objective of the strategy is the minimization of the total size of data transferred under a maximum response-time "phase" constraint. A phase constraint is defined to be the maximum number of arcs on the QPS graph from any node to the destination node.

In (Toth, 1982), an improved ADD architecture is presented together with a transaction management facility (processor) for each layer of the architecture (Figure 1-2). In addition, an outline of how the strategy for a "global view" query can be adapted for a "distributed view" query, is also described. A distributed view query is the result of translation of the global view query by the "distributed view processor" (see Figure 1-2) and it therefore references relations which may be partitioned horizontally and/or vertically.

Figure 1-2 Processors of Layered ADD Architecture
(adopted from (Toth, 1982))

1.3.3 QPSa USING SEMI-JOINS

Recognition that the problem of QP in a DDB is difficult steered the research effort to specialized sub-problems (e.g. processing of a "set query" (Gavish, 1982)) and utilization of semi-joins in distributed query processing. Use of semi-joins assumes that each term in the conjunctive form query is of the form " $R_i.a = R_j.b$ " (equi-join) and that the objective of the distributed query processing is the minimization of the total volume of data transferred over the network.

Consider a term " $R_i.a = R_j.b$ " of a conjunctive form query such that relations R_i and R_j are located in network nodes X and Y respectively. The term can be processed by moving R_i to Y and executing a join of the two relations; however, it may be beneficial to precede this join by a semi-join. The joining attribute of R_j , attribute b, is first moved to X where a semi-join of R_i by R_j is performed. The semi-join is actually a natural join of R_i with the joining attributed b of R_j . It is executed in order to remove those tuples of R_i which do not contribute to the result of the join of R_i with R_j .

The relation R_i is thus reduced in size by the semi-join. It is then moved to Y for a join with R_j . Consequently, the semi-join is beneficial if the size reduction in R_i is greater than the total size of the joining attribute b, of R_j , which was transferred from Y to X for the purpose of a semi-join. Appendix B offers further details on semi-joins.

A strategy which uses semi-joins consists of two phases:

- (i) After initial processing of relations by restrictions and projections, semi-joins are used to reduce the size of relations to be processed by joins.
- (ii) After the size reduction by semi-joins, relations are moved to one site (the destination site) where they are processed by joins to derive the query answer. The sequence of semi-joins used to reduce the size of a relation is also referred to as a semi-join program.

Hevner and Yao (1979) proposed a polynomial-time algorithm for constructing semi-join programs for a "simple" query. A simple query is such that each relation has only a single joining attribute. They incorporate this algorithm into a heuristic to find semi-join programs

for non-simple queries. A semi-join program is separately constructed for each relation using their algorithm for simple queries; there is no effort made to integrate the semi-join programs. Later they make further improvements in cooperation with Apers (Apers, 1983).

Perizzo (1984) makes further improvements to the algorithm proposed in (Apers, 1983) by including a post optimization step which may delay execution of semi-joins. Consider the two-variable term " $R_i.a = R_j.b$ " processed by a semi-join discussed above. If there is a scheduled semi-join on R_j , its execution will likely decrease the cardinality of the joining attribute b . Consequently, the transfer of the joining attribute b , of R_j , for a semi-join on R_i should be delayed until the completion of the semi-join on R_j .

Optimization of a semi-join program adopted for SDD-1 (Bernstein, 1981) is a greedy, iterative hill climbing algorithm. After initial local processing is complete, each individual semi-join is examined and the most profitable semi-join is incorporated into the semi-join program. The cardinality of the affected relation is updated and the process repeats until no more profitable semi-joins are found. As usual, after a reduction by semi-joins all relations are scheduled to be transmitted into the destination node where they are processed by joins to derive the query answer.

Black and Luk (1982) propose a heuristic for generating a strategy for semi-joins. Its property is that an exhaustive search may be performed for moderate size queries. They cite the result of a simulation, which compares their algorithm with those proposed in (Hevner, 1979) and (Bernstein, 1981), in support of their algorithm.

A simple approach to using semi-joins is proposed in (Toan, 1981) for the MICROBE relational DDB which uses a circulating token for concurrency control and is supported by a fast broadcast local network.

A user query is first translated into a relational algebra binary tree. The tree is then restructured using relational algebra properties to determine the sequence of algebraic operators and then a decision is made on the site of the execution of relational operators. In addition, each equi-join is examined in order to determine whether it is profitable to precede it by a semi-join(s).

Perhaps the most advanced work on application of semi-joins in distributed query processing appears in (Yu, 1982a, 1982b, 1983). The authors note (Yu, 1982a) that under certain conditions processing of a two-variable term " $R_i.a = R_j.b$ " by a semi-join of R_i by R_j may be more profitable if the complement of the joining attribute b is used. Assuming that

- (i) $R_j[b]$ denotes the attribute b projected from R_j and $\text{Size}(R_j[b])$ denotes its total size;
- (ii) R_i and R_j are in two different network nodes;
- (iii) the domain for attribute b is finite and its complement, denoted by $\text{Compl}(R_j[b])$, can be derived; and
- (iv) $\text{Size}(R_j[b]) > \text{Size}(\text{Compl}(R_j[b]))$,

then it is more profitable to perform a semi-join of R_i by R_j by transmitting $\text{Compl}(R_j[b])$ to the network node where R_i is located and then joining it (equi-join) with R_i . The result is then complemented to derive a relation which is equivalent to the result of a semi-join of R_i by R_j . Since the size of $\text{Compl}(R_j[b])$ is less than that of $R_j[b]$ and the cost of CPU processing is ignored, the overhead cost is reduced in comparison to a semi-join of R_i by R_j which uses the attribute b , $R_j[b]$.

Yu and Chang utilize this property in a design of two algorithms for processing of previously discussed simple queries (Hevner, 1979). One algorithm is targetted for the environment of a point-to-point network and is shown by simulation to perform better than the algorithm

proposed by Hevner and Yao (1979). The other algorithm is targetted for query processing in a DDB supported by a broadcast type local network (e.g., Ethernet) and they show that it performs better than one proposed by Gouda and Dayal (1981).

Yu and Chang later adapted their algorithm for simple queries for use on queries which reference relations having more than one attribute (Yu, 1982b, 1983). In addition they consider relations which are replicated or horizontally fragmented. In the first phase of optimization, the minimum number of sites which contain all relations referenced by the query is determined. In the second phase, adaption of their "simple queries" algorithm (Yu, 1982a) determines a semi-join program which may be sub-optimal. The program is then improved in two steps. First, relations which do not have to be sent to the destination for final processing are identified. Secondly, a semi-join may be replaced by a better one, if such can be found.

The use of semi-joins for horizontally fragmented relations is restricted to consolidation of a fragmented relation by a union. Optimization determines the location in which the union of fragments should be performed.

Duplicates of a relation are used to decrease the communication cost. One copy of a relation is "primary" while the remaining copies are secondary. If a replicated relation is reduced by a semi-join, its primary copy must be used in all subsequent processing; otherwise a secondary copy may be used.

Semi-joins were applied to processing of queries in a multiprocessor database machine by Valduriez and Gardarin (1984). Processors for join executions, also called join processors, are connected to a fast cache. The resulting network is such that join processors can access the cache in parallel and the cache has the

ability to broadcast data to all join processors. Memories (disks) are connected to the cache via a special bus. Data is transferred from a disk to a cache via a filter which is a special purpose processor whose role is to execute restrictions and projections. In addition to the special purpose architecture which differs from a general DDB environment, optimization objectives differ as well. First, semi-joins are used to reduce the size of join operands because of the size limitations of cache and not because the time delay due to a data transfer is more critical than CPU processing delays. Secondly, an attempt is made to distribute the processing load amongst the join processors; this aspect is ignored in the application of semi-joins to query processing in a general DDB.

Chen and Li (1983) use an execution graph to examine semi-join programs. They propose three rearrangement techniques for execution graphs in order to produce optimal semi-join programs with the following properties. First, they show that the execution graph of an optimal semi-join program cannot be further rearranged. Secondly, there may be more than one occurrence of each semi-join in the optimal program. Finally, each end node of the optimal execution graph must be a final relation. Using an example they show that (i) the second property indicates that general algorithms which allow only one occurrence of a particular semi-join may not yield the optimal semi-join program even when exhaustive search is used; and (ii) the existing approach of processing all two-variable terms at one site after semi-join reductions may not yield the optimal semi-join program either.

A unique and interesting technique which uses "bucket" semi-joins was proposed by Krishnamurthy and Morgan (1984). They suggest that if a full reduction of relations by semi-joins is not cost-effective then it should be replaced by a partial reduction with bucket semi-joins. The partial reduction does not eliminate all tuples which do not.

contribute to the join result as a full reduction by semi-joins does; however, it can be attained at a fraction of overhead incurred by a full reduction.

Consider an equi-join term " $R_i.a = R_j.b$ ". A semi-join of R_i by R_j reduces the size of the relation R_i . It consists of first transferring all unique values of the joining attribute b , of R_j , to the information processor storing R_i and then removing all tuples from R_i which do not contribute to the join of R_i with R_j . A bucket semi-join of R_i by R_j is similar to the semi-join in that the values of the attribute b are also transferred to the information processor storing the relation R_i ; however, not all unique values of the attribute b are transferred. "Bucketing functions" based on probabilistic techniques are used to "pack" the values of the attribute b into highly compressed forms before transmission. Consequently, the cost of the semi-join reduction is lower; however, only a partial reduction is possible, as the bucket semi-join does not eliminate all tuples, of the relation R_i , which do not contribute to the join of R_i with R_j . The authors argue that the bucket semi-join method is suited for achieving a trade-off, between the cost of a bucket semi-join and its benefit, by using the well-known economic principle of diminishing returns.

1.3.4 "OTHER" QPSs

In this sub-section, research is discussed which is either unique in its approach to the processing of queries or it makes specialized assumptions on environment or the query form. The literature is examined in chronological order.

Chandra and Merlin (1976) describe optimization of query processing which can be used in a distributed environment. It is based on the fact that a query Q can be replaced by an equivalent query Q' .

which may have a different number of joins, unions, and cartesian products. The optimization concentrates on the minimization of the number of join and union operators. It should be noted, however, that this approach is considered to be more academic than practical since the replacement of the query Q by an equivalent query Q' involves showing their equivalence ($Q \subseteq Q'$ and $Q' \subseteq Q$) by containment through the use of foldings which are mappings of symbols from one query to another. In addition, optimization of the number of operators does not necessarily imply optimization of real costs because cardinalities of original relations and partial results are ignored altogether.

Pelagatti and Schreiber (1979) developed a mathematical model to evaluate transmission requirements for access to distributed data. The model is targetted for the application design stage, thus the speed at which a QPS for any query is arrived at is unimportant. Given a sequence of operations and specifications for the distribution of data files over the network, the model is used to determine the minimum volume of transmitted data. Essentially, the model can be used for determining the locations of operator executions. Data files can be replicated or fragmented horizontally.

Paik and Delobel (1979) broadly discuss a strategy for processing a query in a highly replicated DDB, i.e., a DDB in which many relations are replicated many times. The first phase of their optimization is similar to the first phase of optimization proposed in (Yu, 1983). It differs from Yu's phase in that the objective is to minimize the total "distance" of all pairs of relations referenced by the query. The result of this optimization is a cluster of relations which is used for the query processing. The second phase consists of execution of restrictions and projections and all two-variable operations which can be processed locally. For the third and final phase of optimization they propose that one of two alternatives can be used. One alternative is to execute all two-variable operations in one site, called centroid.

The centroid is determined in such a way that the volume of data transfer is minimized. The second alternative is to proceed with a distributed execution such as in distributed version of INGRES (Stonebraker, 1976) or ADD (Mahmoud, 1979).

Cellery and Meyer (1980) consider query processing in a multi-query environment. Their strategy's objective is the minimization of the mean response time of queries, the only such attempt that could be found. Optimization is performed by the scheduling of data transfer and local work-load on processors. A heuristic is proposed which optimizes queries' response time at the expense of redundant processing and communication. Their method suffers from two drawbacks. First, they assume that the state of processing is known at any point in time, which, in a network environment, is not only difficult to achieve by information exchange, but it also increases the query cost. Secondly, they ignore the fact that a concurrency control mechanism may use serialization (sequencing) of queries and updates (Bernstein, 1979) using time-stamps in order to maintain integrity of data. If serialization is used, it greatly restricts the freedom of scheduling access to local DBs.

Query optimization in a star computer network is investigated by Kerschberg, Ting and Yao (1982). The queries are assumed to be in a disjunctive form where each disjunction is a "sub-query" in a conjunctive form with one or two-variable terms. The QPS optimizes the query response time by minimizing secondary storage data access performed by information processors. The effect of the communication band-width on the strategy's performance is also investigated.

Gavish and Segev (1982) argue that since the problem of distributed query processing is difficult, concentration should be made on optimal solutions to special types of queries, as for example a simple query (Hevner, 1979), in order to gain further understanding of

the problem. They address the problem of processing a special class of "set queries" on a horizontally partitioned relation. They formulate the problem into an integer programming model and propose two heuristics for its solution. The cost metric under consideration is the volume of transferred data.

Relational algebra query processing optimization is considered by Chu and Hurley (1979, 1982) under special assumptions that the unit processing cost is identical for all information processors and that the unit cost of data transfer between any two network nodes is also identical. A tree representing the original query's algebraic operations is constructed. The first step in optimization involves moving all unary relational operators (selections, projections) as far down the tree as possible. As these reduce the size of operands most of the time, they should be executed as soon as possible. Following this, operations in the tree are permuted to investigate alternate strategies. For that purpose rules, based on the permutability properties of relational algebra, for identifying available correct permutations are given. Finally, the iterative construction model (ICM) generates the set of local operation groups and determines the best sites for their execution.

A unique approach to distributed data processing, which exploits temporal affinity of data, is proposed by Daniell et. al. (1983). They note that DDBs are designed to take advantage of "geographical locality of reference" to data elements, also called geographic affinity. Geographic affinity means that access to a given data item tends to cluster geographically. They argue that access to data items also tends to have a temporal affinity; access to a given data item tends to cluster in time. They suggest that both of these affinities of data access can be exploited by a simple, yet powerful technique of "dynamic replication" which dynamically replicates data items to which access is required.

-34

Each network node can store some number of data elements. If access to a particular data item which does not exist locally is required, a replica of the data item is created. The replica exists until room is needed for replicas of some other data items, in which case replicas of data items not used for the longest period of time are deleted first. Note that a static primary copy of a data item does not exist; rather, it is considered to be that copy which was most recently updated. Both concurrency control and query processing heavily rely on data dictionary which they broadly discuss (Daniell, 1983).

1.3.5 FORMAL ASPECTS OF DISTRIBUTED QUERY PROCESSING

This sub-section briefly identifies research topics on formal aspects of distributed query processing. Although theoretical developments in relational algebra/calculus and semi-joins are applicable to the distributed environment as well, they are not discussed; concentration is on work applicable only to a DDB.

Dynamic rematerialization applied to distributed data processing proposed in (Daniell, 1983) and discussed above is also advocated by Wong (1983). His contribution is not so much in discussing algorithms for concurrency control or processing of queries but in providing a conceptual framework for such algorithms.

A theory of horizontally fragmented relations was developed by Maier and Ullman (1983). Research on processing of distributed queries referencing horizontally fragmented relations assumes that the fragments are mutually exclusive. Maier and Ullman remove this restriction and show that retrieval, insertions and deletions are meaningful. They also show how to find a subset of fragments which suffices for interaction/processing with another relation.

A framework for proving the correctness of a QPS in a DB is developed by Ceri and Pelagatti (1983). For this purpose, a formal model, called Multirelational Algebra, is developed. The authors provide theorems on equivalence of strategies which are used to prove the correctness of a QPS by showing its equivalence to a strategy which is known to be correct.

Kambayashi and Yoshikawa (1983) suggest utilization of temporary functional dependencies for processing of cyclic queries. Since a tree query is easier to evaluate and semi-joins are applicable to tree queries only, transformation of a cyclic query into a tree query is desirable. Transformation is achieved by adding an additional attribute to some relations on the cycle. The additional attribute expresses a temporary functional dependency which often arises due to a restriction on a relation.

This sub-section concludes with the review of results on the complexity of various sub-problems in query processing:

1. Hevner (1980) shows that finding the optimal semi-join program for a given query is NP-hard.
2. Finding the optimal semi-join program is NP-hard for even a restricted class of queries, "tree queries" (Yu, 1982a).
3. Recall that Yu and Chang (1982a) propose that the first phase of static optimization on a replicated DB consists of locating the minimum number of sites which contain all referenced relations. In (Yu, 1982b) authors show that this problem is NP-hard.
4. Gavish and Segev (Gavish, 1982) show that optimization of a set query, which references exactly one horizontally fragmented relation, is NP-hard.

5. Several complexity results have recently appeared on sub-problems in query optimization for a centralized DB. Pramanik and Ittner (1984) studied two types of buffer allocation schemes which can be used in the minimization of page accesses in a join execution. They show that obtaining the upper bound on the buffer size, which is needed in order to avoid page reaccess, is an NP-hard problem. Willard (1984) shows that a subset of queries, based on the Range Query Theory Yao, (1983), can be processed in space $O(N)$ and time $O(\log^d N)$, where d is a small constant which depends on the particular predicate and N is the number of records stored in the DB. Finally, Ibaraki and Tiko (1984) minimize the number of page fetches for a conjunctive form query evaluated by a nested loop method. They propose a data structure which can be used to reduce the number of page fetches and develop a formula to derive the expected number of page fetches. They also show that although their data structure greatly reduces the expected number of page fetches, the minimization of access to pages is still NP-hard.

The classification of research on distributed query processing can be found in the previously shown Table 1-1.

1.4 THESIS OBJECTIVES AND CONTRIBUTIONS

If special sub-sets of queries are neglected, proposed algorithms for formulation of strategies can be classified into two groups depending on whether they use joins or semi-joins as a tactic for processing of distributed queries. Epstein (1980) and Bernstein (1981) point out that after solutions for use of joins and semi-joins are found, combining the two approaches will become necessary. Semi-joins are used to reduce the size of relations which are operands of equi-join operations. Current algorithms move relations reduced in

size by semi-joins to one node where they are processed by joins. Clearly, this phase should be replaced by algorithms for distributed query processing which utilize joins, such as those suggested for ADD (Mannoud, 1979, Toth, 1980, 1982)) or for distributed INGRES (Epstein, 1980).

In comparison to the semi-join tactic, relatively little attention has been paid to the formulation of strategies which use joins. Processing of queries which reference horizontally partitioned relations has received even less attention. Epstein (1978, 1980) considers horizontally partitioned relations in the last phase of optimization. Once a query is divided by a heuristic into irreducible components, and thus at least a partial sequence of joins has been established, a join involving a horizontally partitioned relation is considered for the purpose of load balancing. Selinger's (1980) optimization is such that a horizontally partitioned relation is always consolidated prior to its join with another relation. Finally, although Toth (1982) outlines processing of partitioned queries in the environment of ADD, no optimizing algorithms are offered. Processing of queries, which refer to relations partitioned horizontally and/or vertically, is one of the objectives of this thesis.

Since the problem of processing distributed queries is difficult, heuristic algorithms were proposed for formulation of strategies which use semi-joins or joins as a processing tactic. It is noteworthy that none of the proposed heuristics were proven to produce strategies which are within certain bounds of optimal strategies. In fact only Epstein (1980) compares the performance of a heuristic to that of an exhaustive search; however, this is only for the last phase of optimization which deals with load balancing. He concludes that an exhaustive search may be desirable because it leads to a lower cost strategy in comparison to that of a heuristic.

If strategies produced by heuristic algorithms are evaluated at all, in most cases they are compared to strategies formulated by some other heuristic algorithm. For example, Chang (1982) compares strategies formulated by his heuristic to those formulated by the heuristic algorithm used in SDD-1 (Bernstein, 1981). Even if search for optimal strategies may be too expensive due to the complexity of the problem at hand, optimal strategies are useful for evaluating strategies produced by heuristic formulators. Examination of the complexity of a QPS formulation, search for optimal strategies and their use in evaluating heuristic strategies are also objectives of this thesis.

As previously stated, most formulators adopt one of two objectives for formulation of a QPS. In a distributed environment supported by a high throughput local network, the maximum volume of data processed by any one CPU may be considered to be the critical parameter and hence it is a good candidate for adoption as a cost metric. If, however, not all information processors are of low processing capacities, the data transfer delay may be comparable to that of CPU processing and, consequently, it should not be ignored.

In an environment of high capacity processors which communicate by use of standard services provided by a public network, the communication delay may be considered to be the most critical factor. Minimization of the total volume of data transferred over the network should also decrease the query response time; however, a higher volume of transferred data does not necessarily result in an increase in the response time if the data is processed or transferred in parallel.

Selection of an appropriate measure of cost for a given application is also one of the objectives of this thesis.

THESIS OBJECTIVES

The objectives of this thesis, under the assumptions stated in Figure 1-3, are:

1. To examine the complexity of distributed query processing.
2. To develop a heuristic algorithm to formulate strategies for join processing of queries referencing relations fragmented horizontally and/or vertically.
3. To develop an algorithm which finds optimal strategies using a number of cost objectives.
4. To examine the use of optimal strategies in evaluation of heuristic strategy formulators and determination of a suitable measure of cost.

THESIS CONTRIBUTIONS

Determination of optimal network locations for execution of joins in a given sequence is shown to be a polynomial problem. This result is used in a new two-phased approach to formulation of a QPS. Modelling results indicate that for a test-bed of queries this approach leads to strategies of which execution delays are close to those of minimal delay strategies.

The sum of the size of partial results is shown to be a good alternative in comparison to other cost functions. The modelling results indicate that it should be considered for adoption as a measure of cost in optimizing a query processing strategy particularly in an environment in which the cost arising only due to a network data transfer does not dominate the cost arising only due to CPU processing and vice-versa.

ASSUMPTIONS

1. DDB Organization/Structure: Relational DDB is based on, or similar to, ADD ((Mahmoud, 1979), (Toth, 1980, 1982)), shown in Figure 1-2.
2. Network: point-to-point
3. DDB Relations: Relations may be partitioned horizontally and/or vertically; however, multi-copy relations or fragments are not considered.
4. Query: A conjunctive normal form query is assumed to be expressed on the distributed view of the DB. In addition, each term/formula is at most two-variable.
5. Distributed Query Processing Optimization: Optimization can be divided into four phases:
 - (i) Reduction in size of relations by execution of one-variable operations comprises the first phase. This phase is straightforward; the query restrictions and projections to remove attributes not necessary for query processing are executed first.
 - (ii) The second phase consists of using semi-joins to further reduce the size of relations. Since this thesis addresses the problem of the remaining two phases of optimization, to make the problem more tractable, semi-joins are not considered any further.
 - (iii) After the reduction phases, the strategy for processing the query by joins and unions is formulated.
6. QPS Formulation:
 - static QPS formulation combined with a threshold mechanism in the query execution phase;
 - single query;
 - the usual load balancing restriction is adopted in that a relational operation can be executed only in the network location of one of its operands
7. QPS Formulation Method: heuristic algorithm and exhaustive search
8. Objective: The objective of a QPS formulation is the minimization of the query execution cost where the cost can be expressed in a measure most suitable for the application environment.

Figure 1-3 Adopted Assumptions

An algorithm to formulate the optimal strategy for a distributed view query is developed (Chapter 3). The algorithm is a non-redundant exhaustive search through the complete space of query's strategies and it is used to find optimal strategies using a number of optimization objectives (cost measures).

A heuristic algorithm to formulate a sub-optimal strategy for a distributed view query is developed (Chapter 4). Overhead of this algorithm is "controllable"; its increase results in a lower cost strategy. Consequently, it enables balancing of the trade-off between the strategy formulation cost and execution costs. Results of modelling (Chapter 6) suggest that the heuristic produces strategies with delays which are close to those of optimal delay strategies.

A model of query processing in a relational DDB is developed and is integrated with the algorithm which finds optimal strategies. Optimal strategies are employed in evaluating various cost measures which can be used in optimization of distributed queries. In particular, information which is necessary to examine the effect on the various costs associated with optimizing a particular query execution cost is identified and its use is demonstrated on a test-bed of queries. Nine measures of cost are defined and modelling results indicate that those measures which include costs due to both CPU processing and data transfer are more effective than those which include only CPU processing or data transfer cost.

1.5 OUTLINE OF FURTHER SECTIONS

In Chapter 2, an algorithm is developed which finds optimal network

locations for executing joins in a given sequence. Since the algorithm is polynomial in time, the problem it solves is polynomial. This is used as a motivation for a two-phased approach to formulation of a QPS: First the sequence of joins is determined and then the optimal network locations of their execution are found.

The formulation of optimal strategies is examined in Chapter 3 and algorithm OPT_GV (OPTimal for a Global View), which develops a complete and "non-redundant" graph of strategies for a query expressed on the global view of the DB, is presented. That a query is expressed on the global view of the DB implies that it does not reference partitioned relations. This assumption is later removed and the algorithm is adapted in order to accomodate queries referencing relations which are partitioned either vertically and/or horizontally. The adaptation results in algorithm OPT_DV (OPTimal for a Distributed View).

A heuristic algorithm, called HEU_DV (HEUristic for a Distributed View), which formulates strategies for queries which reference vertically and/or horizontally partitioned relations is presented in Chapter 4. It has the property that the overhead cost of a QPS formulation can be increased in order to decrease the formulated strategy execution cost.

A model of QP in a relational DDB is presented in Chapter 5. Both algorithms, OPT_DV and HEU_DV, were implemented within the framework of the model. The model and algorithms are general in the sense that they make few assumptions and incorporate a variety of cost measures which can be realized in distributed query processing.

Chapter 6 reports the modelling results for a test bed of queries which can be issued on a particular application of a DDB. The modelling of queries is used to demonstrate how optimal strategies can be utilized in an evaluation of (i) heuristic QPS formulators and (ii) appropriate measures of cost to be adopted for optimization of distributed queries.

Finally, Chapter 7 offers concluding remarks and suggestions for further research.

CHAPTER 2

COMPLEXITY AND TWO-PHASED QPS FORMULATION

2.1 INTRODUCTION

The objective of this chapter is to examine the complexity of query processing by joins. Implications of results on complexities of distributed query processing are:

1. The optimal semi-join program generation is NP-hard (Hevner (1980), Chiu and Ho (1980)); thus, if semi-join programs are utilized in processing of distributed queries, their processing is NP-hard.
2. The Copy Location problem is NP-hard (Yu and Chang (1982b)); consequently, if a query references replicated relations, its processing is NP-hard.
3. Optimization of a "set query" is NP-hard (Gavish, 1982). Since a very simple query containing just one two-variable term " $R_1.a = R_2.b$ ", in which R_1 is a non-partitioned and R_2 is a horizontally partitioned relation, belongs to the class of set queries (Gavish, 1982), its processing is NP-hard. Consequently, processing of distributed queries referencing horizontally fragmented relations is also NP-hard.

Some complexity results on query processing in a centralized DB were already discussed in Chapter 1. Pramanik and Ittner (1984) showed that obtaining the upper bound on the buffer size, which is needed in order to avoid page reaccess in a join execution, is an NP-hard problem. This problem, however, is highly specialized and relevant to

an execution of one join; it has little relevance to distributed query processing. Ibaraki and Tiko (1984) minimize the number of page fetches for a conjunctive form query evaluated in a centralized DB by a nested loop method. They showed that this problem is NP-hard. Processing of queries by the nested loop method, however, is very similar to the tuple substitution proposed for INGRES relational DB (Wong, 1976, Youssefi, 1979). Both of these methods are inapplicable to the distributed environment as they may cause excessive network delays.

Judging by the results listed above, it would not be surprising if processing of queries which do not reference replicated or partitioned relations was, in fact, NP-hard; however, at this time this is still only a conjecture which is yet to be proven. What will be shown in this chapter is that the fact that a strategy is processed in a distributed environment does not seem to indicate an exponential explosion of the solution space. In particular, given a sequence of joins to process a query, determination of locations for their execution such that the total volume of data is minimized is a POLYNOMIAL problem.

Since determination of the optimal network locations for execution of a given sequence of joins is a polynomial problem, formulation of a Query Processing Strategy (QPS) in a DDB may be performed in two "phases": The first phase determines a sequence of joins while the second determines the network locations for their execution.

Decomposition of a problem into two sub-problems is the motivation for this approach. For the second phase of formulation, a polynomial-time algorithm, such as the one proposed in this chapter, can be used. In the first phase, a sequence of joins must be determined. If this is a polynomial problem, then the two-stage approach leads to a polynomial-time formulation of strategies. However, it was already stated that determination of a sequence of joins is likely to be an NP-hard problem. In this case the two-phased approach is also beneficial.

As the first phase ignores the distributed environment when determining the sequence of joins, which is likely to be an NP-hard problem, fewer variables and parameters are needed in comparison to a formulation method which does not include the two phases.

Before proceeding with the polynomial-time algorithm (section 2.3) which determines optimal locations for join executions, section 2.2 discusses the query processing problem in detail. In particular, it reviews representation of queries by graphs, examines the number of joins which are necessary to process a query and provides a comprehensive example of distributed query processing.

2.2 THE QUERY PROCESSING PROBLEM

Assume that $R_1, R_2, \dots$ are relations; $a, b, \dots$ are attributes; and $\theta_1, \theta_2, \dots$ are arithmetic relational operators. A query

$$q = (R_1.a \theta_1 R_2.d) \wedge (R_1.b \theta_2 R_3.e) \wedge (R_1.c \theta_3 R_4.f)$$

contains the set of two-variable terms $\{V''\}^q = \{V''_1, V''_2, V''_3\}$, with

$$V''_1 = R_1.a \theta_1 R_2.d;$$

$$V''_2 = R_1.b \theta_2 R_3.e; \text{ and}$$

$$V''_3 = R_1.c \theta_3 R_4.f.$$

Since the processing of one variable terms, restrictions and projections which remove unnecessary attributes, is always performed before the processing of two-variable terms begins, they are excluded from further considerations in this chapter.

QUERY GRAPH

The representation of the query q by a query graph in which nodes denote relations and edges denote two-variable terms. Mahmoud, 1975 is shown in Figure 1-1. If a join is executed to process the term $V_1 = R_1.a \Theta_1 R_2.b$, this processing can be represented in the query graph by a coalescence of the graph nodes representing the join operands. In other words, the nodes representing the relations R_1 and R_2 are replaced in the query graph by the join result. Any edges between R_1 and R_2 are removed as they represent the processed two-variable terms and edges connected to the coalesced nodes are connected to the node which replaces them. The modified query graph, which includes the representation of the join result of processing the term $(R_1.a \Theta_1 R_2.b)$, is shown in Figure 1-2. Note that the two-variable terms V_1 and V_2 were modified to $R_1.c \Theta_2 R_3.e$ and $R_1.c \Theta_3 R_4.f$ respectively. If V_3 is the next term to be processed by a join resulting in the relation R_5 , the query is modified again and its graph is shown in Figure 1-3.

NUMBER OF JOINS

The number of joins which are necessary to process a given query depends on the number of two-variable terms in the query definition and on the number of cascaded joins. It will be shown, however, that the number of joins simply depends on the number of relations referenced by the query.

One of the generally accepted rules for a QPS formulation, adopted here, is that two-variable terms referencing the same relations should be processed simultaneously by an execution of a "cascaded" join. For example, a query " $R_1.a \Theta_1 R_2.b \Theta_2 R_3.c \Theta_3 R_4.d$ " contains two

Query graph representation for query
 $Q = \{P_1, a_1, P_2, c\} \cup \{P_2, a_2, P_3, c\} \cup \{P_1, a_1, P_2, c\}$

Modified query graph to represent the query Q to process
the two-variable term P_1, a_1, P_2, c

Modified query graph to represent the query Q to process
the two-variable term P_2, a_2, P_3, c

Figure 1- Query graph and its execution

two-variable terms and normally requires two joins for its processing; however, both two-variable terms can be processed simultaneously by an execution of a cascaded join for, effectively, a cost of one join. When tuples of both relations are tested to determine whether they satisfy a predicate term, say " $R_1.t = R_2.d$ ", they might as well be tested, for a negligible cost of a comparison, against the other predicate " $R_1.a = R_2.s$ ".

A second example of a cascaded join is shown in Figure 3-3. After the execution of a join of R_2 with R_3 , the remaining two joins can be cascaded as they apply to the same relations R_1 and R_4 .

Theorem 3-1: If a query references N_q relations then the number of joins, of which some may be cascaded, necessary for the query processing is $N_j = N_q - 1$, where N_q is the number of referenced relations.

Proof: Consider the usual query representation by a graph as shown in figures 2-1 and 2-2. Recall that execution of a join of two relations is represented by a coalescence of the nodes which represent join operands; they are replaced by a node which represents the join result. An example of this process was already shown in figures 2-1 and 2-2. With this representation of query processing it becomes simple to derive the number of joins necessary to process a query. Any join execution removes two nodes from the query graph and replaces them by one node representing the join result. If there is more than one edge between the two join operands then it is a cascaded join. The query execution is complete when the query result is obtained. In a query graph representation this occurs when all nodes are coalesced into one. Since each join removes two nodes from the query graph and adds only one, $N_j = N_q - 1$ join executions are required. .EF.

Query: $(R_1 \bowtie G_1 R_2 \bowtie b) \bowtie (R_2 \bowtie c G_2 R_3 \bowtie d) \bowtie (R_3 \bowtie e G_3 R_4 \bowtie f) \bowtie (R_4 \bowtie g G_4 R_5 \bowtie h)$

Query graph:

Query graph after a join of R_2 with R_3 giving R_{12}

Query graph after a join of R_{12} with R_4 giving R_{124}

Query graph after a join of R_{124} with R_5 giving the query result, R_{1245}

Figure 2-2 Coalescence of the query graph nodes

Note that the number of joins which are necessary to process a query is denoted by N_j . The subscript "j" is not used as an index which varies over a range of values; rather, it is intended as an indication that N_j stands for a number of joins. As many symbols are needed to identify a number of different entities, notation follows the following convention: The number of entities is denoted by "N", subscripted by a letter which indicates the name of entities. For example, the number of joins which are required to process a given query is denoted by N_j , where "j" indicates "joins". If, however, the number of entities varies depending on some variable, it is denoted by an appropriately indexed lower case symbol. For example, the number of unique strategies which are available to process a query "i" is denoted by " n_i ", where index "i" identifies the query $q_i, i = 1, 2, \dots, N_q$, where N_q is the number of given queries. Recall that the notation glossary is provided for fast reference to symbols.

EXAMPLE OF DISTRIBUTED QUERY PROCESSING

Consider a query

$$q = R_1.a \Join R_2.b \Join R_3.c \Join R_4.d \Join R_5.e \Join R_6.f,$$

which references original relations R_1, R_2, R_3 and R_4 , located in network nodes A, X, Y and Z respectively. The query can be processed by various strategies one of which is:

- i) join relations R_1 and R_2 in A to give relation $R_{1,2}$;
- ii) join relations R_3 and R_4 in Z to give relation $R_{3,4}$; and
- iii) join $R_{1,2}$ and $R_{3,4}$ in X to give $R_{1,2,3,4}$, the query result.

The query graph and the timing diagram for the aforementioned strategy is shown in figures 2-3 b) and 2-3 c) respectively. The notation for strategy timing diagrams is defined in Figure 2-3(a).

Note the choice of subscripts for representation of original relations referenced by the query and for partial results. An original relation, i.e., relation which is not a join result, is always subscripted by a single digit ($R_1, R_2, \dots, R_9$). This implies that a query cannot reference more than 9 relations; however this is sufficient for explanatory purposes.

A partial result of some join executions is subscripted by two or more digits. The first digit represents the number of two-variable relational operations 'joins' to derive it. The remaining digits do not have any significance; however, the subscript must be unique. Since the first digit identifies the number of two-variable operations 'joins', it is tacitly assumed that any query can be processed by at most 9 joins; again, this is sufficient for explanatory purposes.

For the above strategy, a join of R_1 with R_2 results in a relation R_{12} . As R_1 and R_2 are not partial results of join processing, their subscript contains a single digit only. R_{12} is a result of one two-variable relational operation; therefore, the first digit of its subscript is a "1". R_{12} is also a result of one join; therefore, the first digit of its subscript is a "1". Finally, the relation R_{123} is a result of three joins; therefore, the first digit of its subscript is a "3".

NOTATION: $\xrightarrow{R}$... transfer of R

A join is executed to process a two-variable term $(R_1.a \text{ O } R_j.b)$. R_1 is located in X already, while R_j arrives just prior to the join execution. The join result is R_k which is then transferred to some other network location.

(a) Notation for strategy timing diagrams

(c) Strategy timing diagram

(d) Strategy timing diagram

Figure -- Strategy timing diagrams, a query and two strategies for its processing

Note that the right subscripts identify relations obtained by unique strategies. A relation is either an original relation referenced by the query or it is a result of a unique strategy/sub-strategy from which it is derived. Although relations R_{11} of Figure 2-3(c) and R_{12} of Figure 2-3(d) have identical content, i.e., they are comprised of identical tuples, as they are derived by different sub-strategies incurring different costs, they are considered to be two different partial results for the purpose of an OPS formulation.

The timing diagram of Figure 2-3(c) shows that R_2 is first transferred to the network node W where, immediately upon its arrival at time t_1 , it is joined with R_1 to process the two-variable term $(R_1.a \theta_1 R_2.b)$. The join is complete at time t_3 and it results in relation R_{11} which is then transferred to the network node Z. This transfer is complete at time t_5 .

R_3 is transferred to Z where, immediately upon its arrival at time t_2 , the join of R_3 with R_4 to process $(R_3.c \theta_2 R_4.d)$ commences. This join is complete at time t_4 .

Finally, when both R_{11} and R_{18} are available for processing in Z, which occurs at time t_5 , their join is executed. The query result is available in node Z at time t_6 .

Information about a relation is represented by a triplet $\langle x, c, I \rangle$, where

x_i is the network location of the relation R_i ;

c_i is the cumulative cost to derive R_i ; and

I_i is information about R_i which is necessary for the size and cost estimation of join processing.

Since in this section the cost is expressed in terms of the volume of data transferred over the network, the cumulative cost to derive R_k , c_k , includes not only the volume of data transferred to execute the join resulting in R_k but also the volume of all data transferred in derivation of the join operands. If R_k is the result of a join of R_1 with R_2 , c_k of x, c, I_k includes not only the transfer of R_1 or R_2 in order to execute the join which produces R_k but also all data transfer needed to derive R_1 and R_2 .

For the example of Figure 1-3, the cumulative cost c_3 of x, c, I_3 for the relation R_3 , includes the size of

- i) R_2 due to its transfer for a join with R_1 ;
- ii) R_3 due to its transfer for a join with R_4 ; and
- iii) R_1 due to its transfer for a join with R_8 .

Another strategy for processing the given query q is shown in Figure 1-4. It has the same sequence of joins as the strategy of Figure 1-3; however, the join of R_1 with R_2 is executed in X instead of W . Consequently, the query result is derived with a different cost than that of Figure 1-3.

2.3 JOIN LOCATION PROBLEM

The join location problem can be defined in the following way: "Given an ordered sequence of joins $S = (*_z, z=1, 2, \dots, n)$, where $n=N_j$, to process a query of which result is to be delivered to the destination node d , find the network location for execution of each join $*_z \in S$. The locations for join executions must be such that the total volume of data transferred over the network is minimized."

Given an ordered sequence of joins $S = \{ \sigma_z, z=1, 2, \dots, n \}$, the algorithm must determine the network location for execution of each $\sigma_z \in S$.

$n=N_J$ is the number of relations which are necessary to process a given query. It is determined by Theorem 2-1 which states that $N_J = N_R - 1$, where N_R is the number of original relations referenced by the given query. However, for the remainder of this chapter, the number of joins in the sequence S will be denoted by "n" in order to avoid double subscripts such as the subscript in " σ_{N_J} ".

Also, recall that the sequence of joins is given and that the cost is measured by the volume of transferred data; sufficient information in I_1 of $(X, c, I)_1$ is the total size (in bytes), s_1 , of the relation R_1 . Consequently, I_1 is replaced by s_1 and the triplet for R_1 becomes $(X, c, s)_1$. For example, the relation R_3 has a triplet $(x, c, s)_3$, where x_3 is the network location of R_3 , c_3 is the cumulative cost of R_3 , and s_3 is the total size of R_3 .

Without loss of generality assume the following:

1. The $N_R = n+1$ original relations referenced by the query are located in distinct network locations $W, X, Y, \dots$. Let X_R denote the set of N_R network locations in which the original relations are located.
2. Each join $\sigma_z \in S$ can be executed in any one of the network locations in X_R . This is a slight departure from assumptions summarized in Figure 1-3 which state that a join can be executed only in one of the network locations of its operands.

ALGORITHM LOCATIONS VIA EXAMPLE

The polynomial time algorithm which determines the optimal locations for join executions creates sets in order $E_1, E_2, \dots, E_n$, where each set E_z corresponds to a join, $*_z$, $z = 1, 2, \dots, n$, respectively. For each network location $x \in X$, a set E_z contains one relation R_x which is the result of the join $*_z$ execution in the network location x . R_x is such that it has the minimal cumulative cost of all possibilities for execution of the join $*_z$ in the network location x .

Consider a sample query

$$q = R_1.a \theta_1 R_2.b \wedge R_2.c \theta_2 R_3.d \wedge R_3.e \theta_3 R_4.f$$

which is to be processed by the sequence of joins $S = *_1, *_2, *_3$, where

- $*_1$... is the join of R_1 with R_2 ;
- $*_2$... is the join of R_3 with R_4 ; and
- $*_3$... is the join of the result of $*_1$ with that of $*_2$.

Relations' sizes, in bytes, are assumed to be

$$s_1 = 100; s_2 = 200; s_3 = 300; s_4 = 400;$$

the size of the join $*_1$ result is 150;

the size of the join $*_2$ result is 250 and

the size of the join $*_3$ result is 100.

Assume that the network locations of relations R_1, R_2, R_3 and R_4 are W, X, Y and Z , respectively. X_j is the set of locations in which joins can be executed and it is defined by the set of locations in which original relations are located, i.e., $X_j = \{W, X, Y, Z\}$.

As operands of $*_1$ are original relations R_1 and R_2 , creation of E_1 is straight forward. Assume that the result of a join

of R_1 with R_2 , in the network location W , is $R_{1,1}$;
 of R_1 with R_2 , in the network location X , is $R_{1,2}$;
 of R_1 with R_2 , in the network location Y , is $R_{1,3}$; and
 of R_1 with R_2 , in the network location Z , is $R_{1,4}$.

The only possibility for execution of $*$, in the network node $x = X$, is to move the original relations R_1 and R_2 , if they are not located there already, to x and then form their join. For the network location $x = W$, R_2 is moved to W where it is joined with R_1 , which is already located in W . As

the size of R_2 is s_2 and

the cost of transferring R_2 , from X to W , is the total volume of transferred data (in bytes),

the cumulative cost of the join result, relation $R_{1,1}$, is
 $c_{1,1} = s_2 = 200$.

Similarly, there is only one possibility for execution of $*$, in each of the network locations X , Y and Z :

i) The result of $*$, execution in X is $R_{1,2}$ with the cumulative cost $c_{1,2} = s_1 = 100$.

ii) The result of $*$, execution in Y is $R_{1,3}$ with the cumulative cost $c_{1,3} = s_1 + s_2 = 300$. In this case both relations R_1 and R_2 must be transferred to Y for the join execution.

iii) The result of $*$, execution in Z is $R_{1,4}$ with the cumulative cost $c_{1,4} = s_1 + s_2 = 300$.

As there is only one possibility for execution of $*$, in each of the network locations in X_r ,

$E_1 = \{R_{1,1}, R_{1,2}, R_{1,3}, R_{1,4}\}$

Note that all relations in E_1 have identical size of 300 bytes, as they all represent the join $*$, of R_1 and R_2 . Although their content is

R_2, R_3, R_4 is added to L_1 .

Returning back to determination of the network locations of join executions from the selected minimum cost element $R_2 = R_{34}$.

1. the location of the $*$ execution is determined by the network location of the first relation in the pair $R_2, R_{34} \in L_1$, which is $x_2 = X$; and

2. the location of the $*$ execution is determined by the network location of the second relation in the pair R_2, R_{34} , which is $x_3 = Z$.

Now that the algorithm to find optimal network locations for join executions has been demonstrated through an example, it can be formally presented.

ALGORITHM LOCATIONS

Input to the polynomial-time algorithm LOCATIONS consists of:

1. Ordered sequence of joins $S = (*_z, z=1, 2, \dots, n)$.
2. Original relations $R_i, i=1, 2, \dots, N_r$, referenced by the query, together with their "triplets" $(x, c, s)_i$. Each relation R_i is located in a distinct network location identified by x_i of the triplet $(x, c, s)_i$. The locations of original relations define the set X_r in which joins can be executed.
3. For each join $*_z$, the size of the join result.
4. Set H , such that for each join $*_z \in S$ there is an ordered pair $[o_1, o_2]_z \in H$ which identifies operands of $*_z$.

Application of the above steps to creation of E_1 is now demonstrated.

1. For $*_1$, steps 2 to 3 are executed.
2. $*_2$, the join under consideration, is $*_1$. As the two operands of this join are original relations R_1 and R_2 , two sets, $E_u = \{R_1\}$ and $E_v = \{R_2\}$, are created. The triplet $(x, c, s)_1$ for the relation R_1 is such that

$x_1 = W \dots$ is the network location of R_1 ;

$c_1 = 0 \dots$ is the cumulative cost of R_1 ; and

$s_1 = 100 \dots$ is the size of R_1 .

Similarly, the triplet $(x, c, s)_2$, for the relation R_2 is such that $x_2 = X$, $c_2 = 0$ and $s_2 = 200$.

For each network location x in $X_r = \{W, X, Y, Z\}$, Step 3 is executed.

3. For each combination of $R_i \in E_u$ and $R_j \in E_v$, R_k is created and its cumulative cost is calculated. R_k is the result of $*_1$ execution in the network location x . As sets E_u and E_v have only one element each, execution of this step, for the network location $x = W, X, Y$ and Z respectively, results in the following:

(i) For $x=W$, R_{11} is added to E_1 . R_{11} has a cumulative cost

$$c_{11} = c_1 + c_2 + s_2 = 200, \text{ as } x_1 = W \text{ and } x_2 = W.$$

(ii) For $x=X$, R_{12} is added to E_1 . R_{12} has a cumulative cost

$$c_{12} = c_1 + c_2 + s_1 = 100, \text{ as } x_1 = X \text{ and } x_2 = X.$$

(iii) For $x=Y$, R_{13} is added to E_1 . R_{13} has a cumulative cost

$$c_{17} = c_1 + c_2 + s_1 + s_2 = 300, \text{ as } x_1 = Y \text{ and } x_2 = Y.$$

iv For $x=E$, R_{12} is added to E . R_{12} has a cumulative cost

$$c_{18} = c_1 + c_2 + s_1 + s_2 = 300, \text{ as } x_1 = E \text{ and } x_2 = E.$$

Now consider application of the above steps to creation of E_2 for the join $*_2$. As operands of this join are also original relations, relations R_3 and R_4 , creation of E_2 is similar to that of E_1 . Assuming that the result of the join $*_1$ is

of R_3 with R_4 , executed in X is R_{15} ;

of R_3 with R_4 , executed in Y is R_{16} ;

of R_3 with R_4 , executed in Z is R_{17} ; and

of R_3 with R_4 , executed in E is R_{18} .

$E_2 = (R_{15}, R_{16}, R_{17}, R_{18})$. Recalling that the cumulative costs of R_3 and R_4 are $c_3=0$ and $c_4=0$ respectively, the cumulative costs of relations in E_2 are:

$$(i) \quad c_{15} = c_3 + c_4 + s_3 + s_4 = 700$$

$$(ii) \quad c_{16} = c_3 + c_4 + s_3 + s_4 = 700$$

$$(iii) \quad c_{17} = c_3 + c_4 + s_3 + s_4 = 400$$

$$(iv) \quad c_{18} = c_3 + c_4 + s_3 + s_4 = 300$$

The case of the join $*_3$ is more interesting as its operands are not original relations but results of previous join executions. One of the $*_3$ operands is the result of $*_1$ while the other is the result of $*_2$. Application of the above steps 1-3, to creation of E_3 , is now shown.

For the join $*_2 = \pi_1$, steps 1 to 3 are executed.

2. As neither of the two operands of $*_2$ are original relations, $E_1 = E_2$ and $E_V = E_2$. For each network location x in V_2 , Step 3 is executed.

3. Each combination of $R_1 \in E_1$ and $R_2 \in E_2$ represents one of the possibilities for operands of $*_2$ execution in the network location x . For each such operand combination calculate the cumulative cost of R_2 , the result of their join in the network location x . Table 10 summarizes these combinations of $*_2$ operands for the case of $x=W$. For each combination it also shows the cumulative cost of the join $*_2$ execution in the location $x=W$. An examination of the table reveals that the minimal cumulative cost for the join $*_2$ execution in the network location W is for the combination of operands R_{11} and R_{18} . Assuming that the result of joining these two relations in the network location W is R_{31} , R_{31} is added to the set E_2 . The cumulative cost of R_{31} is $c_{31} = 750$.

Execution of this step for $x = X, Y$ and Z is similar.

In summary, $E_3 = \{R_{31}, R_{32}, R_{33}, R_{34}\}$, such that the minimal cumulative cost of $*_3$ execution in the network location

(i) W is $c_{31} = c_{11} + c_{18} + s_{18} = 750$, for the relation R_{31} , which is the result of a join of R_{11} and R_{18} ;

(ii) X is $c_{32} = c_{12} + c_{18} + s_{18} = 650$, for the relation R_{32} , which is the result of a join of R_{12} and R_{18} ;

(iii) Y is $c_{33} = c_{12} + c_{17} + s_{12} = 650$, for the relation R_{33} , which is the

TABLE - POSSIBLE JOIN EXECUTIONS

The table lists possible operands for a σ_j execution in the network node x_{ij} . For each combination of operands P_i ($i = 1, 2$) and P_j ($j = 1, 2$), the cumulative cost of their join, which is executed in the network node x_{ij} and results in P_{ij} , is shown.

P_i	$x_{i,1}$	P_j	$x_{j,2}$	P_{ij}	cost of P_{ij}
P_1	$(X, 200, 150)$	P_1	$(X, 700, 250)$	$P_{1,1}$	400
P_1	$(X, 200, 150)$	P_2	$(X, 700, 250)$	$P_{1,2}$	500
P_1	$(X, 200, 150)$	P_3	$(Y, 400, 250)$	$P_{1,3}$	950
P_1	$(X, 200, 150)$	P_4	$(Z, 300, 250)$	$P_{1,4}$	750
P_2	$(Y, 300, 150)$	P_1	$(X, 700, 250)$	$P_{2,1}$	950
P_2	$(Y, 300, 150)$	P_2	$(X, 700, 250)$	$P_{2,2}$	200
P_2	$(Y, 300, 150)$	P_3	$(Y, 400, 250)$	$P_{2,3}$	400
P_2	$(Y, 300, 150)$	P_4	$(Z, 300, 250)$	$P_{2,4}$	800
P_3	$(Y, 300, 150)$	P_1	$(X, 700, 250)$	$P_{3,1}$	1150
P_3	$(Y, 300, 150)$	P_2	$(X, 700, 250)$	$P_{3,2}$	1400
P_3	$(Y, 300, 150)$	P_3	$(Y, 400, 250)$	$P_{3,3}$	1100
P_3	$(Y, 300, 150)$	P_4	$(Z, 300, 250)$	$P_{3,4}$	1000
P_4	$(Z, 300, 150)$	P_1	$(X, 700, 250)$	$P_{4,1}$	1150
P_4	$(Z, 300, 150)$	P_2	$(X, 700, 250)$	$P_{4,2}$	1400
P_4	$(Z, 300, 150)$	P_3	$(Y, 400, 250)$	$P_{4,3}$	1100
P_4	$(Z, 300, 150)$	P_4	$(Z, 300, 250)$	$P_{4,4}$	1000

result of a join of R_{12} and R_{13} .

iv. E is $R_{14} = R_{12} \bowtie R_{13}$, $s_{14} = 550$, for the relation R_{14} , which is the result of a join of R_{12} and R_{13} .

Relations in E_1 have the following properties

i. All R_i relations in E_1 are the results of Q_1 execution. Since Q_1 is the last join in the sequence Q , each relation in E_1 represents the query result.

ii. Each relation is located in a distinct location $x_i \in V_N$.

iii. $R_i \in E_1$ is the minimum cumulative cost result of Q_1 , executed in the network location $x_i \in V_N$.

Once the set E_1 is determined, it is necessary to consider the network destination to which the query result should be delivered. The minimum cumulative cost of each $R_i \in E_1$ is updated to include the cost of transferring R_i to the query destination location. If the destination is the network location d , the cost of each R_i in E_1 is updated by

$$c_i = c_i + s_i, \text{ if } x_i \neq d.$$

Assuming that the destination location for the sample query is $d=W$, the cumulative cost of relations in E_1 now become:

$$c_{31} = 750 \text{ remains unchanged as } x_{31} = W = d;$$

$$c_{32} = c_{32} + 100 = 750, \text{ as } x_{32} \neq d \text{ and the size of } R_{32} = 100;$$

$$c_{33} = c_{33} + 100 = 750, \text{ as } x_{33} \neq d \text{ and the size of } R_{33} = 100;$$

$$c_{34} = c_{34} + 100 = 650, \text{ as } x_{34} \neq d \text{ and the size of } R_{34} = 100.$$

Once the cumulative cost of relations in E_z is updated to include the cost of transfer to the destination location, it is a simple matter to find that relation R_z in E_z , which has the minimum cumulative cost. The strategy which derives R_z also identifies the optimal network locations for execution of joins specified in the sequence S .

For the sample query, the selected relation R_z is R_{34} with the cumulative cost $c_{34}=25$. Clearly, the network location for execution of the final join in the sequence S is identified by the found element R_z . Since $R_z=R_{34}$, the final join, $*_1$, should be executed in the network location $x_{12}=C$. As R_{34} is the result of the join $*_1$ with operands R_{12} and R_{23} ,

- i. the network location of $*_1$ execution is X , as R_{12} is the result of $*_2$ execution and the location of R_{12} is $x_{12}=X$, and
- ii. the network location of $*_2$ execution is C , as R_{23} is the result of $*_3$ execution and the location of R_{23} is $x_{23}=C$.

From the above determination of the network locations for $*_1$ and $*_2$ executions it is evident that the algorithm must be able to determine operands of the join $*_3$ execution which results in the selected, minimum cost element R_{34} . This is achieved by expanding Step 3. Any time a relation R_k is added to E_z , operands of $*_z$ execution, which results in R_k , will be stored for later use. For this purpose, for each set E_z a corresponding set D_z is created. When a relation R_k , which is the result of a join of R_i with R_j , is added to E_z , a pair $(R_i, R_j)_k$ is added to D_z .

Let

D_z ... be a set which corresponds to the join $*_z$ and its set E_z . For each relation R_k in E_z there is a pair of relations R_1, R_2 in E_z . The pair identifies those operands of $*_z$ which are used to create R_k .

For the sample query, there are three sets D_1, D_2 and D_3 which correspond to sets E_1, E_2 and E_3 , respectively. The set $E_1 = \{R_1, R_2, R_3, R_4\}$ is created for the join $*_1$. As operands of $*_1$ are original relations R_1 and R_2 .

$D_1 = \{R_1, R_2, \dots, R_1, R_2, \dots, R_1, R_2, \dots, R_1, R_2, \dots\}$

Similarly, as operands of $*_2$ are original relations R_5 and R_6 , and $E_2 = \{R_5, R_6, R_7, R_8\}$.

$D_2 = \{R_5, R_6, \dots, R_5, R_6, \dots, R_5, R_6, \dots, R_5, R_6, \dots\}$

When E_3 is created, relations R_{31}, R_{32}, R_{33} and R_{34} are added to E_3 by repetitive execution of Step 3. As

(i) R_{31} is created by execution of $*_3$ with operands R_1 and R_8 , $(R_1, R_8)_{31}$ is added to D_3 ;

(ii) R_{32} is created by execution of $*_3$ with operands R_{12} and R_{18} , $(R_{12}, R_{18})_{32}$ is added to D_3 ;

(iii) R_{33} is created by execution of $*_3$ with operands R_{12} and R_{17} , $(R_{12}, R_{17})_{33}$ is added to D_3 ;

(iv) R_{34} is created by execution of $*_3$ with operands R_{12} and R_{18} ,

R_2, R_3, R_4 is added to L_1 .

Returning back to determination of the network locations of join executions from the selected minimum cost element $R_2 = R_3, R_4$.

i. the location of the $*$ execution is determined by the network location of the first relation in the pair $R_2, R_3, R_4 \in L_1$, which is $x_2 = X$; and

ii. the location of the $*$ execution is determined by the network location of the second relation in the pair R_2, R_3, R_4 , which is $x_3 = Z$.

Now that the algorithm to find optimal network locations for join executions has been demonstrated through an example, it can be formally presented.

ALGORITHM LOCATIONS

Input to the polynomial-time algorithm LOCATIONS consists of:

1. Ordered sequence of joins $S = (*_z, z=1, 2, \dots, n)$.
2. Original relations $R_i, i=1, 2, \dots, N_r$, referenced by the query, together with their "triplets" $(x, c, s)_i$. Each relation R_i is located in a distinct network location identified by x_i of the triplet $(x, c, s)_i$. The locations of original relations define the set X_r in which joins can be executed.
3. For each join $*_z$, the size of the join result.
4. Set H , such that for each join $*_z \in S$ there is an ordered pair $[o_1, o_2]_z \in H$ which identifies operands of $*_z$.

a. o_1 of $[o_1, o_2]_z \in H$ identifies the first operand of $*_z$. This operand is either

(i) an original relation R_1 , in which case $o_1 = (R_1, A)$, or

(ii) a result of a join $*_u$, which precedes $*_z$ in the sequence

ie., $u < z$, in which case $o_1 = (A, *_u)$.

Consequently, o_1 is an ordered pair of two elements such that

the first element of o_1 is a relation R_1 if the first operand of $*_z$

is an original relation R_1 ; otherwise it is A nil, and

the second element of o_1 is a join $*_u$, if the first operand of $*_z$

is a result of join $*_u$ execution; otherwise it is A nil.

b) o_2 of $[o_1, o_2]_z$ identifies the second operand of $*_z$. This operand is either

(i) an original relation R_2 , in which case $o_2 = (R_2, A)$, or

(ii) a result of $*_{v_2}$ which precedes $*_z$ in the sequence S ie.,

$v < z$, in which case $o_2 = (A, *_u)$.

5. The destination location d to which the query result should be delivered.

The set H , for the sample query which is to be processed by the sequence $S = \{*_1, *_2, *_3\}$, has three pairs, one for each join:

$$H = \{[(R_1, A), (R_2, A)]_1, [(R_3, A), (R_4, A)]_2, [(A, *_1), (A, *_2)]_3\}$$

For each join $*_z$, $z=1, 2, \dots, n$, the algorithm creates sets E_z and D_z by execution of steps 2 to 3. In Step 2,

(i) a dummy set $E_u = \{R_1\}$ is created in case the first operand of $*_z$ is an original relation R_1 , ie., if o_1 of $[o_1, o_2]_z \in H$, is $o_1 = (R_1, A)$; and

if a dummy set $E_v = R_j$ is created in case the second operand of $*_z$ is an original relation R_j , i.e., if ρ_2 of $\{\rho_1, \rho_2\}_z \in H$ is $\rho_2 = R_j$.

Step 3 is then executed for each network location x in N in order to find the minimum cumulative cost relation which is a result of $*_z$ execution in x .

Creation of the set E_z uses sets E_u and E_v . If the join $*_z$ has operands which are results of joins $*_u$ and $*_v$, i.e., $\{\rho_1, \rho_2\}_z \in H$ is such that $\rho_1 = *_u$ and $\rho_2 = *_v$, then E_u and E_v are those of $*_u$ and $*_v$ respectively. If one or both operands of $*_z$ are original relations then dummy set s , E_u and/or E_v , created by Step 2 are used.

For all combinations of $R_i \in E_u$ and $R_j \in E_v$, the algorithm derives the cumulative cost c_k for the join $*_z$ of R_i with R_j , executed in the network location x . That relation R_k is added to the set E_z for which c_k is minimum. In addition, the pair $(R_i, R_j)_k$ is included in the set D_z . The pair identifies operands of $*_z$ execution which derives the chosen, minimum cost relation R_k which is added to E_z .

Once the sets E_n and D_n are determined for the final join in the sequence S , cost of relations in E_n is updated to include the cost of the final transfer of the query result to the destination location. Assuming that the query result is to be delivered to the network location d , the minimum cost c_i , for each relation R_i in E_n , is updated to include the transfer of the query result to d .

Optimal locations for join executions are determined by first finding that relation R_n , in E_n , which has the minimum cumulative cost. Following this, locations for join executions are found by, respectively, following operands stored in sets E_u , starting with E_1 , which corresponds to the found, minimum cost relation R_n .

Algorithm LOCATIONS is shown in Figure 1.

Note that the Step 3 cost calculation of c_k can be modified to accommodate any measure of cost without invalidating the algorithm. For example, assume that $\beta(X,Y)$ represents the transfer of a unit of data byte, from a network location X to Y and $D[\beta(X,Y)]$ is its dollar cost. For the cost measured in terms of the dollar cost only due to a data transfer, the cost calculation performed in Step 3 becomes:

$$c_k := c_1 + c_2 + \begin{cases} 0; & \text{if } x_1 = x_2 = x \\ s_1 \cdot D[\beta(x_1, x)]; & \text{if } x_1 \neq x \text{ and } x_2 = x \\ s_2 \cdot D[\beta(x_2, x)]; & \text{if } x_1 = x \text{ and } x_2 \neq x \\ s_1 \cdot D[\beta(x_1, x)] + s_2 \cdot D[\beta(x_2, x)]; & \text{if } x_1 \neq x \text{ and } x_2 \neq x \end{cases}$$

CORRECTNESS

Obviously, E_1 , the set for the first join in the sequence S , is derived correctly. Assume that sets E_u , $u=1,2,\dots,z-1$, are correct, i.e., that E_u contains the minimum cumulative cost relations which are results of join $\ast_u$ execution in network locations of X_r . If sets E_u , $u=1,2,\dots,z-1$, are correct, derivation of E_z is also correct as it uses enumeration. Consequently, by induction, E_n , the set for the last join in the sequence S , is also correct.

ALGORITHM LOCATIONS TO FIND NETWORK LOCATIONS FOR EXECUTION OF A GIVEN SEQUENCE OF JOINS

INPUT

ordered sequence of joins $S = \langle *_1, *_2, \dots, *_{n-1} \rangle$ where $n-1$ is the number of joins required to process the given query.

original relations $R_1, R_2, \dots, R_m$ referenced by the query, together with their triplets $\langle x, c, s \rangle$ recall that $m \leq n-1$. The relations are located in N distinct network locations which form the set X in which joins can be executed.

For each $*_z$ in S , the size of its result.

Let $\langle a, b \rangle$ such that for each join $*_z \in S$ there is an ordered pair $\langle a, b \rangle$ in $\langle a, b \rangle$ which identifies operands of $*_z$. $\langle a, b \rangle$ is an unordered pair of two elements such that

i. the first element of $\langle a, b \rangle$ is a relation R_i if the first operand of $*_z$ is an original relation R_i , otherwise it is Λ nil, and

ii. the second element of $\langle a, b \rangle$ is a join $*_{j_1}$ which precedes $*_z$ in the sequence S $\langle *_{j_1}, *_{j_2}, \dots, *_{j_{z-1}} \rangle$ if the first operand of $*_z$ is a result of join $*_{j_1}$ execution, otherwise it is Λ nil.

$\langle a, b \rangle$ of $\langle a, b \rangle_{z-1}$ identifies the second operand of $*_z$. This operand is either

i. an original relation R_i , in which case $a_{z-1} = R_i, \Lambda$, or

ii. a result of $*_{j_1}$ which precedes $*_z$ in the sequence S $\langle *_{j_1}, *_{j_2}, \dots, *_{j_{z-1}} \rangle$ in which case $a_{z-1} = \Lambda, *_{j_1}$.

5. The destination location d to which the query result should be delivered.

OUTPUT: Network locations for execution of joins S , such that the total volume of data transferred over the network is minimized.

METHOD: For each join $*_z \in S$, $z=1, 2, \dots, n$, sets E_z and D_z are created.

E_z contains $n+1$ elements, triplets of the form $\langle x, c, s \rangle_k$ representing the result of the join $*_z$ execution in a distinct network location $x \in X$, such that c_k has the minimum cumulative cost for $*_z$ execution in x .

D_z contains a pair $\langle R_i, R_{j_1} \rangle_k$ for each relation R_k in E_z . The pair of relations R_i and R_{j_1} identifies the operands of $*_z$ execution which derives the relation R_k .

STEP

1. For each join $*_z \in S$, $z=1, 2, \dots, n$, create sets E_z and D_z by executing steps 2-3.

Figure 2-4 Algorithm LOCATIONS to find optimal network locations for execution of a given sequence of joins

10. Create a dummy set $E_u = R_u$ in case the first operand of $\ast_z$ is an original relation R_u , i.e., if o_1 of $[o_1, o_2]_z \in H$, is $o_1 = R_u \wedge$.

11. Create a dummy set $E_v = R_v$ in case the second operand of $\ast_z$ is an original relation R_v , i.e., if o_2 of $[o_1, o_2]_z \in H$, is $o_2 = R_v \wedge$.

Execute Step 7 for each network location x in V in order to find the minimum result of execution of $\ast_z$ in x .

12. Create an empty set E_z . In the following it is assumed that the join $\ast_z$ as operand which are results of the joins $\ast_u$ and $\ast_v$ which have respective sets E_u and E_v . For all combinations of $R_u \in E_u$ and $R_v \in E_v$, derive the cumulative cost c_k for the join $\ast_z$ with R_u executed in the network location x

$$c_k = \begin{cases} 0 & \text{if } x = x_u \wedge x \\ s & \text{if } x_u = x \text{ and } x = x_v \\ s & \text{if } x_v = x \text{ and } x = x_u \\ s_u + s_v & \text{if } x \neq x_u \text{ and } x \neq x_v \end{cases}$$

Append that R_k to E_z for which c_k is minimum. Also include in the set E_z the pair $(R_u, R_v)_k$ which identifies operands of $\ast_z$ execution which derives the chosen, minimum cost relation R_k .

13. Add the cost of transfer, of the query result to the destination node, by executing the following statement for each $R_k \in E_z$, where E_z is the set for the last join $\ast_z$ in the given sequence S:

If $x_d \neq i$, where i is the destination node, then $c_k = c_k + s$.

14. Determine the optimal location of join executions.

a. Find that relation R_m in E_z , where E_z corresponds to the last join in S, for which c_m is minimum.

b. Find the optimal location for execution of each join by invoking the recursive procedure LOCATE by "LOCATE $\ast_n, R_m$ ".

RECURSIVE) PROCEDURE LOCATE $\ast_z, R_m$:

Comment -- $\ast_z$ is a join for which the network location is to be determined.

R_m is the relation in E_z which determines the location of $\ast_z$ execution.

BEGIN

print "optimal execution for join", $\ast_z$, "is the network location", x_m .

Comment -- Let R_u and R_v be those of the pair $(R_u, R_v)_m \in E_z$.

IF $(R_u$ is not an original relation) THEN LOCATE $\ast_u, R_u$;

IF $(R_v$ is not an original relation) THEN LOCATE $\ast_v, R_v$;

(Comment -- $\ast_u$ is the second element in o_1 of $[o_1, o_2]_z \in H$;

$\ast_v$ is the second element in o_2 of $[o_1, o_2]_z \in H$.)

END of the procedure LOCATE.

E_n contains relations which are results of $*_n$ executions in distinct network locations as defined by X_F . Each relation $R_i \in E_n$, located in $x_i \in X_F$, is the join $*_n$ result obtained in x_i with minimum cumulative cost. As Step 4 simply adds the cost of transferring the query result to the destination for each $R_i \in E_n$, Step 5 will correctly find that relation $R_m \in E_n$, which has the minimum cumulative cost. This relation R_m identifies the minimum cumulative cost for obtaining the query result in the destination node.

Finally, once the minimum cost relation R_m is found, the location of R_m is also the location of $*_n$ execution. In addition, $(R_i, R_j)_m \in D_n$ identifies the operands of that $*_n$ execution which is used to derive R_m . Locations of R_i and R_j are also the locations for execution of $*_u$ and $*_v$ respectively, where

- i) $*_u$ execution results in R_i and
- ii) $*_v$ execution results in R_j .

Using sets D_u and D_v , locations for operands of $*_u$ and $*_v$ are located and so on.

Note that exhibition of the algorithm LOCATIONS is for the purpose of showing that the problem of finding the optimal locations for execution of a given sequence of joins is polynomial. Consequently, the algorithm is neither claimed to be efficient nor the best for its task.

LOCATIONS -- POLYNOMIAL TIME ALGORITHM

Examine the complexity of the algorithm LOCATIONS.

1. Step 1 causes n executions of Steps 2 to 4, once for each join in S .
2. Step 2 just prepares for creation of sets E_z and D_z and it causes execution of Step 3 once for each location in X_r , where X_r is the set of locations in which joins can be executed. As the cardinality of X_r is $|X_r| = n+1$, each execution of Step 2 causes $n+1$ executions of Step 3.
3. The cardinalities of E_z and D_z , created by Step 3, are $|E_z| = |D_z| = |X_r| = n+1$. Each element in E_z and D_z is derived by examining all combinations of elements of two sets E_u and E_v . Since $|E_u| \leq n+1$ and $|E_v| \leq n+1$, determination of each of relations in E_z requires at most $O((n+1)^2)$ operations.
4. Step 4 adds the cost of transferring the query result to the destination node for each relation in E_n . This requires $O(n+1)$ operations.
5. Finally, Step 5 first finds the minimum cost element of E_n ($O(n+1)$ operations) and then it determines the location of execution for each join in S which requires $O(n)$ operations.

From 1 to 5 above it is evident that the order of complexity is determined by Step 3. Since

2

1.0

1.1

1.25

1.5

1.4

1.6

1.8

2.0

2.2

2.5

2.8

3.2

3.6

4.0

4.5

5.0

5.6

6.3

7.1

8.0

9.0

10.0

11.2

12.5

14.0

16.0

18.0

20.0

22.4

25.0

28.0

31.5

36.0

40.0

45.0

50.0

each execution of Step 3 has a complexity of $O((n+1)^2)$;

Step 3 is executed $n+1$ times for each execution of Step 2; and

Step 2 is executed n times,

the algorithm LOCATIONS complexity is of $O((n+1)^4)$. As the input to the algorithm is the sequence of n joins to process $n+1$ original relations, the algorithm is P-complete.

Theorem 2-2: The join locations problem is polynomial.

Proof: The polynomial-time algorithm LOCATIONS solves any instance of the join locations problem.

QED.

2.4 DISCUSSION

It was already stated that because the problem of finding optimal locations for execution of joins in a given sequence can be solved by a polynomial-time algorithm, formulation of a QPS should be performed in two phases: The first phase determines a sequence of joins while the second determines the network locations for their execution.

As was pointed out in the previous section, the second phase of optimization can utilize whatever minimization objective is deemed appropriate, such as minimization of the total volume of data transferred over the network or the dollar cost incurred by a data transfer.

For the first phase of optimization, an attractive objective is minimization of the total size of partial results. Although most query optimizers are claimed to minimize the query response time, in most cases the cost that is actually minimized is the volume of data either transferred over the network or processed by CPUs.

For example, in an environment of high capacity processors which communicate by use of standard services of a public network, the communication delay may be considered to be the most critical factor. Minimization of the total volume of data transferred over the network should also decrease the query response time. For medium capacity processors connected by a high band-width local network, the CPU delay may be considered to be the most critical factor. Minimization of the total volume of data processed by CPUs should also decrease the query response time.

Minimization of the total size of partial results should be applicable to both of the above cases. It may be argued that if the total size of partial results is minimized, both the volume of data (i) processed by CPUs or (ii) transferred over the network is reduced. Consequently, in the first phase of the proposed two-phase approach the optimization objective should be minimization of the total size of partial results.

Although this two-phase approach may look promising, it does not guarantee optimal strategies even if the first phase uses an optimal algorithm and not a heuristic. This is because the distributed environment is ignored in the determination of the sequence of joins. A distributed environment offers the potential for parallel data transfer and join executions by CPUs and an optimal QPS may not be the one comprised of a sequence of joins which minimizes the total size of partial results; and it is exactly such a sequence which is determined in the first phase.

Most QPS formulators, proposed in the literature or incorporated in experimental DDBs, minimize the total volume of data transferred over the network; they may not derive optimal strategies which minimize the total query response time either. Consequently, it is necessary to compare the response time of

- (i) strategies derived by the "two-phased" approach, discussed above, and
- (ii) strategies which minimize the total volume of data transferred over the network

with the minimum response time possible. This is one of the objectives of subsequent chapters.

CHAPTER 3

SEARCH FOR OPTIMAL STRATEGIES

3.1 INTRODUCTION

Referring back to the classification Table 1-1 it can be observed that QPS formulation algorithms intended for an operational environment are heuristic. This seems to imply that researchers agree that formulation of a QPS is a problem of which complexity precludes the use of algorithms which find the optimal strategy; therefore, the use of heuristic approaches is justified even if they may lead to a sub-optimal strategy. Indeed, it was shown in Chapter 2 that processing of queries is NP-hard. Nonetheless there is a two-fold impetus for finding optimal strategies:

a) Epstein (1980) suggests that an exhaustive search may be desirable because it leads to a lower cost strategy in comparison to that of a heuristic. Black and Luk (1982) proposed a heuristic algorithm to find semi-join programs. The heuristic is such that for queries with a small number of two-variable terms it derives optimal semi-join programs. Consequently, search for optimal strategies should be considered at least for queries with a small number of two-variable terms.

b) It is difficult to evaluate strategies produced by heuristic formulators without comparing them to optimal strategies. None of the proposed heuristics were proven to produce strategies which are within certain bounds of optimal strategies. In fact only Epstein

(1988) compares the performance of a heuristic to that of an exhaustive search for the last phase of his optimization method.

The problem of finding optimal strategies is addressed in this chapter. Three general techniques were applied to the formulation of optimal strategies. The first is a "backtracking" approach ((Horowitz, 1978), (Aho, 1974), (Reingold, 1977)), while the second is based on the "dynamic programming" technique ((Horowitz, 1978), (Reingold, 1977)). It will be shown that QPS formulation algorithms developed by applying these two techniques suffer from the same deficiency in that they examine many strategies/sub-strategies redundantly.

The last technique applied to the formulation of QPSs creates the solution space in levels (Reingold, 1977). Algorithm OPT_DV, which is based on this technique, has two desirable properties. First, it avoids redundancy exhibited by algorithms using the other two techniques. Secondly, as it creates the complete strategy space, the optimal strategy for any measure of cost can be found.

The following assumptions, in addition to those stated in Chapter 1 (see Figure 1-3), are accepted:

a) The first assumption refers to a priori cost and size estimation of partial results. Given two relations R_i and R_j , a QPS formulation algorithm is able to estimate the size and cost of the partial result of joining R_i with R_j to process a two-variable term ($R_i.a \Join R_j.b$). Relations R_i and R_j can be either original/elementary relations referenced by the given query or partial results of processing of some of the two-variable terms. Included in the estimation is the removal of joining attributes (attributes 'a' and 'b' for the above relations R_i and R_j) by a projection. A projection is performed to remove joining attributes only if they are not 'target' attributes and are no longer necessary in processing of the query's further two-variable terms.

b) The second assumption refers to the destination of the query result. When a query is issued against a DDB, the desired destination to which the query result should be delivered is also specified. The destination node is typically, but not necessarily, the node of the query's origin. The strategy to processing a given query must be such that the query result is delivered to the specified destination node. It is assumed that when a strategy, under examination by any one of the algorithms presented in this chapter, is such that the query result is not obtained in the destination node, the strategy is augmented by the final transfer of the query result to the destination node. In addition, the total cost of the strategy is increased by the cost of this final data transfer.

c) Finally, it is assumed that relations referenced by a query are not partitioned either horizontally or vertically. A query may therefore be considered to be expressed on the global view of the data base (Toth 1980, 1982). However, this assumption is removed for the last algorithm presented in this chapter.

3.2 DEPTH-FIRST SEARCH

A depth-first search is a variation of a backtrack algorithm in which the choices under examination can be represented by a decision tree. The edges of a decision tree represent choices under examination while its nodes are used as states to keep track of choices made thus far. The Depth-first Search, as its name implies, proceeds to develop a tree toward the leaves which represent solutions while retaining only one path from the root to a leaf at any one time. Once a solution is found, the algorithm backtracks one level up and again proceeds toward the leaves to examine the remaining options. Backtracking from any node occurs only if all available options from the state represented by

the nodes have been exhausted.

A depth-first search, as is the case in its application to the problem of a QPS strategy formulation, may often suffer from the pitfall of redundancy (Reingold, 1977). This redundancy is the reason that this approach is considered to be unsatisfactory and is not pursued in the main text of this dissertation any further; however, further details about its application to the formulation of a QPS can be found in Appendix C.

3.3 DECOMPOSITION (DIVIDE AND CONQUER)

A "divide and conquer" approach is actually a dynamic programming technique. When applied to the problem of a QPS formulation, it is based on the observation that the query result is obtained by a final join of two relations which are partial results of some previous processing (joins). Therefore, the problem can be divided into two sub-problems of strategy formulation for the two partial results (sub-queries) followed by the final join. The sub-problems can be similarly sub-divided which results in a recursive solution.

Detailed discussion of, and an algorithm based on this approach can be found in Appendix C. The main difficulty with the application of this approach to the problem of finding the optimal strategy for a given query is the formulation of the "principle of optimality". The division of a query into two sub-queries must be such that their optimal strategies are to be sub-strategies of the query's optimal strategy. However, there appear to be no properties of distributed query processing which can be exploited to find such sub-queries without examining all decompositions of queries into pairs of sub-queries. Consequently, not only does the algorithm based on this approach result in an exhaustive search but it also examines many strategies/sub-strategies redundantly as is demonstrated in Appendix C.

3.4 STRATEGY SPACE GRAPH REPRESENTATION

3.4.1 INTRODUCTION

Algorithms based on techniques discussed in previous sections suffer from redundancy in that they examine the same strategies or sub-strategies redundantly. Reingold (1977) points out that to avoid this type of redundancy, a solution space may be built in levels and this approach is utilized by an algorithm presented in this section.

Consider a simple query $q = (R_1.a \theta_1 R_2.b) \wedge (R_2.c \theta_2 R_3.d)$ which references relations R_1 , R_2 , and R_3 , located in network locations W , X , and Y , respectively. The query graph is shown in Figure 3-1(a). Let the left sub-script of a relation, such as in " ${}_WR_1$ ", identify the relation's network location. One of the strategies for processing the query q is

join ${}_WR_1$ with ${}_XR_2$ in W to give ${}_WR_{11}$ and

join ${}_WR_{11}$ with ${}_YR_3$ in W to give ${}_WR_{21}$.

The usual representation of this strategy is shown in Figure 3-1(b). Since one of the assumptions is that restrictions and projections are executed as soon as possible, the strategy representation does not show execution of one variable relational operations. In addition, as the query is assumed to be defined on the global view of the DB, relations are not partitioned horizontally, or vertically, and only one type of two-variable relational operations, a join, is needed to process the query. Consequently, the strategy representation can be simplified, by the removal of relational operators which results in the strategy represented by the tree shown in Figure 3-1(c). The root of the tree represents the relation derived by the strategy while the leaves represent original relations referenced by the query. Each tree element, which is not a leaf, represents a partial result of a join of two operands represented by

(a) Query graph for $q = (R_1.a \theta_1 R_2.b) \vee (R_2.c \theta_2 R_3.d)$

(b) Strategy representation showing join operators

(c) Strategy representation without join operators

Figure 3-1 Representation of a strategy

the node's children. For the example tree of Figure 3-1(c), relation R_{11} is the result of a join of R_1 with R_2 . Since R_2 is transferred from X to W , the edge connecting R_2 with R_{11} is a "resistor" symbol.

An alternate strategy, which differs from that shown in Figure 3-1(b) only in the location of the last join execution, is

join $W R_1$ with $X R_2$ in W to give $W R_{11}$ and

join $W R_{11}$ with $Y R_3$ in Y to give $Y R_{22}$.

The strategy is shown in both representations in Figure 3-2(a). If the strategies of figures 3-1(c) and 3-2(a) are examined, it becomes obvious that some of their portions are identical. In particular, the sub-strategy for the relation R_{11} is identical in both cases. Both strategies are juxtaposed in Figure 3-2(b).

Instead of representing each strategy separately, their identical portions can be merged together to obtain a graphical representation of both strategies as shown in Figure 3-2(c). Note that the graph is organized in levels, where

level zero contains original relations referenced by the query;

level one contains relations which are partial results of one join execution; and

level two contains relations which are partial results of two join executions.

In general, level n contains relations which are partial results of n join executions.

The concept of merging identical portions of trees representing different strategies is utilized in the representation of the query's strategy space by a graph.

(a) Second strategy to process $q = (R_1 \cdot a \cdot Q_1 \cdot R_2 \cdot b) \cdot (R_2 \cdot c \cdot Q_2 \cdot R_3 \cdot d)$

(b) Identical sub-strategies of two strategies

(c) Combined strategies

Figure 3-2 Two strategies for processing $q = (R_1 \cdot a \cdot Q_1 \cdot b) \cdot (R_2 \cdot c \cdot Q_2 \cdot R_3 \cdot d)$ and their representation by a graph

3.4.2 GRAPH REPRESENTATION

Development of a graph representing the space of strategies is demonstrated for a query q of Figure 3-2. The strategy space must be such that it reflects a choice of the sequence of joins and their locations of execution. Options for selection of the first join together with the location of its execution are:

- (i) R_1 and R_2 are joined in W (ie.,
 R_2 is moved to W) to give a relation R_{11} ;
- (ii) R_1 and R_2 are joined in X to give R_{12} ;
- (iii) R_2 and R_3 are joined in X to give R_{13} ; and
- (iv) R_2 and R_3 are joined in Y to give R_{14} .

The stated options can be represented by a graph in which nodes of level zero represent original relations, ie., relations referenced by the query, and nodes of a level $p > 0$ represent relations which are results of p join executions.

To avoid confusion between network nodes and nodes of a graph, the graph nodes will be referred to as elements of a graph or as elements. Since these elements represent relations, they will sometimes be referred to simply as relations so that a repetition of the phrase "element representing a relation" is avoided.

A relation R_k of a level $p > 0$ has two edges connecting it to two relations R_i and R_j , of levels p_x and p_y respectively, from which it is derived by a join. If one of R_i and R_j is transferred for the purpose of a join execution, the edge connecting it with R_k is a "resistor" symbol.

If processing of a query requires $n=N_j$ joins, the graph has levels $p = 0, 1, 2, \dots, n$, i.e., it has $n+1$ levels. Elements of the level n are called "final" elements/nodes; they represent the query result, each obtained by a unique strategy. Elements of level zero, which represent original relations referenced by the query, are called "initial" elements.

Figure 3-3(a) shows, using a graphical representation, the above options for the first join execution in processing a query shown in Figure 3-1(a) and to which figures 3-1 and 3-2 also apply. The graph contains elements representing original relations and also relations which are results of one join execution. The graph elements are labelled using the usual notation R_{rsb} , in which "rsb" is the right subscript which is a unique identifier, or ID for short. Recall from Chapter 2 that the right subscript consists of either one digit, in which case it is for an original relation, or it consists of more than one digit, in which case the first digit denotes the number of joins used to derive the relation. In addition, if it is necessary to identify the network location of a relation, a left subscript is added for that purpose. For instance, if a graph element is labelled by $W_{R_{21}}$, the relation is located in the network location W and is a result of two joins.

Continuing with the example, consider relations which are results of exactly two joins:

- (i) Y_{R_3} is joined with $W_{R_{11}}$ in W to give $W_{R_{21}}$
- (ii) Y_{R_3} is joined with $W_{R_{11}}$ in Y to give $Y_{R_{22}}$
- (iii) Y_{R_3} is joined with $X_{R_{12}}$ in X to give $X_{R_{23}}$

(a) Partial (levels 0 and 1) graph of strategies

(b) Complete (levels 0, 1, and 2) graph of strategies - it is derived from the partial graph shown in (a), whose edges are not shown

Figure 3-3 A partial and a complete graph of strategies for $(R_1, a, O_1, R_2, b) \cdot (R_2, c, O_2, R_3, d)$

(iv) YR_3 is joined with XR_{12} in Y to give YR_{24}

(v) WR_1 is joined with XR_{13} in X to give XR_{25}

(vi) WR_1 is joined with XR_{13} in W to give WR_{26}

(vii) WR_1 is joined with YR_{14} in Y to give YR_{27}

(viii) WR_1 is joined with YR_{14} in W to give WR_{28}

The graph representation for the above options is derived from a graph shown in Figure 3-3(a). For each relation which is a result of two joins a new element is added to the graph level two. Each relation of the level two has two edges connected to relations of lower graph levels from which it is created. The graph is shown in Figure 3-3(b).

Given a final element, i.e., an element representing the query result, its strategy can be obtained simply by tracing edges to the lower graph levels. For the example of Figure 3-3, element WR_{26} identifies a strategy shown in Figure 3-4; is determined by following all edges from the final element/relation WR_{26} to the initial elements in the graph level zero.

Obviously, the graph is connected, as there is at least one edge for every element. A graph element/node is either

(i) an initial element representing an original relation referenced by the query, in which case there must be at least one edge connected to an element of a higher graph level as the relation must be processed by a join, or

(ii) an element representing a relation which is a result of some join executions and therefore it must have two edges connecting it to elements of lower graph levels.

(a) Strategy timing diagram for v^R_{26}

(b) Tree representation of the strategy for v^R_{26}

Figure 3-4 Tree representation and a timing diagram of a strategy for v^R_{26}

Definition 3-1: Level p of the graph representing a query's strategy space is a set of elements which represent relations processed by exactly p joins.

The following observations can be made about the graph properties:

Observation 3-1: Levels are mutually exclusive sets of graph elements/nodes.

Observation 3-2: The graph has levels $0, 1, 2, \dots, n$, where n is the number of joins necessary to process a query, i.e., the graph has $n+1$ levels.

Observation 3-3: A node of a graph level $p > 0$ has exactly two edges which connect it to lower graph level elements. One connects it to a node of level p_x while the other to a node of level p_y . The levels p_x and p_y must be such that $p_x, p_y < p$ and $p = 1 + p_x + p_y$.

Observation 3-4: Every element which is not a final element must have at least one edge connecting it to a node of a higher graph level.

Observation 3-5: Every final element representing the query result obtained by a unique strategy is a root of a binary tree which defines the strategy to derive it (for an example see Figure 3-4).

Each graph element of level p represents the result of p joins. The number of graph levels, therefore, depends on the number of joins necessary to process a given query and is determined by Theorem 2-1.

3.4.3 ALGORITHM FOR A GLOBAL VIEW QUERY.

Observation 3-5 implies that a graph representing a query's strategy space can be viewed as a superposition of binary trees, each representing a unique QPS. The trees are such that they may "share" some of their nodes. The problem is to construct the graph of strategies for any given query in the most efficient way and, preferably, without redundancies exhibited by the Depth-first Search and Divide and Conquer algorithms shown in Appendix C. If such a graph is constructed for a given query and the cost of relations represented by the graph elements is known, it is a simple task to examine all final elements to find the one which has minimum cost. It identifies the minimum cost strategy while the strategy itself is defined by a tree of which the minimum cost element is a root.

Formally, if relations R_i and R_j are results of processing of a set of original relations denoted by $\{r\}_i$ and $\{r\}_j$, respectively, relations R_i and R_j can be joined only if

- (i) $\{r\}_i \cap \{r\}_j$ is empty and
- (ii) there exists a two-variable term " $R_s \theta R_t$ " specified by the query such that $R_s \in \{r\}_i$ and $R_t \in \{r\}_j$.

Relation R , which is a result of p joins and is therefore an element of level p , can be created by a join of two relations R_i and R_j of levels p_x and p_y , respectively. As R_i and R_j are at respective levels p_x and p_y , R_i is a result of p_x joins, while R_j is a result of p_y joins. If R_i and R_j are joined to give R , then R will be a result of $p_x + p_y + 1$ joins and therefore the relationship $p = p_x + p_y + 1$ must be satisfied.

To summarize, the level by level graph construction is started by the creation of level 0, whose elements represent relations referenced by the query. Then levels 1, 2, ..., n are built in that order, where

n is the number of joins which are necessary to process the given query. Nodes of level p represent relations which are results of exactly p join executions. An element of level p is constructed from elements of levels p_x and p_y such that $p = 1 + p_x + p_y$. The number of graph levels is equal to one plus the number of two-variable relational operations (joins) necessary to process the query and it is determined by the use of Theorem 2-1. Any time a new element representing a relation R is constructed from elements representing relations R_1 and R_2 , its size and cumulative cost are estimated. This conforms to the first assumption of the section 3.1. In addition, pointers to elements representing R_1 and R_2 are added; in the pictorial representation, they represent the graph edges connected to relations of lower levels from which the new element is created.

Since each element of the final level represents the query result obtained by a different strategy, in order to find the optimal strategy, the highest/final level is searched for the minimum cost element. The optimal strategy is found by following its pointers to the initial elements/sinks representing the query's original relations.

Algorithm OPT_GV (Optimal for a Global View), which constructs the graph of available strategies for a given query, is shown in Figure 3-5. Consider a query

$$q = (R_1.a \theta_1 R_2.b) \theta_2 (R_2.c \theta_3 R_3.d),$$

shown in Figure 3-1(a). The first step of the algorithm creates level 0 elements which correspond to the original relations R_i , $i=1,2,3,4$ (see level 0 of Figure 3-3(a)). The cumulative cost of an original relation is set to the cost of processing its restriction(s) and a projection to remove its unnecessary attributes. Since the query is processed by two joins, Step 3 is executed twice.

**ALGORITHM OPT_DV TO CREATE THE GRAPH OF AVAILABLE STRATEGIES
FOR A GLOBAL VIEW QUERY**

INPUT: Query q defined by $\{V'\}^q$, $\{V''\}^q$ and $\{(x,c,I)\}^q$, where

- (i) $\{V'\}^q$ is the set of one-variable terms;
- (ii) $\{V''\}^q$ is the set of two-variable terms; and
- (iii) $\{(x,c,I)\}^q$ is the set of triplets $(x,c,I)_i$, $i=1,2,\dots,N_r$, where N_r is the number of distinct relations referenced by the query's two-variable terms. The triplet $(x,c,I)_i$ conveys information about relations R_i :
 x_i is the network location of R_i ;
 c_i is the cumulative cost of R_i which is zero; and
 I_i is information about relation R_i which is necessary for the size and cost estimation of join processing.

OUTPUT: The graph of strategies and their costs for processing q .

METHOD: The graph of strategies is created by levels. Each graph element represents either the original relation referenced by the query (elementary relation) or a relation which is a partial result of processing of one or more joins. Level 0 represents the original relations with processed restrictions and projections which remove unnecessary attributes (i.e., attributes which are neither target nor joining attributes), while level $p>0$ represents relations which are partial results of exactly p joins. Each graph node representing relation R_i contains

- a) two pointers to its predecessors from which it is created;
- b) pointer to the next element of the same level;
- c) $\{r\}_i$, a set of elementary relations processed by joins to derive R_i . For example, if R_i is the result of a join of R_1 with R_2 , followed by a join with R_3 , then $\{r\}_i = \{R_1, R_2, R_3\}$; and
- d) triplet $(x,c,I)_i$ which conveys the following information about R_i :
 x_i is the network location of R_i ;
 c_i is the cumulative cost expended to derive R_i ;
 I_i is information about R_i which is necessary for the size and cost estimation of join processing.

STEPS

1. Create elements of level 0 directly from relations referenced by the query and set the cost of each element to the cost of its restrictions and a projection to remove unnecessary attributes.

Figure 3-5 Algorithm to develop the graph of strategies available for processing a global view query (OPT_DV)

ALGORITHM OPT_{GV} TO CREATE THE GRAPH OF AVAILABLE STRATEGIES
FOR A GLOBAL VIEW QUERY

2. Do step (1) for $p = 1, 2, \dots, n$, to create levels 1, 2, ..., n , where n is the number of joins necessary to process the query.
3. Insert elements representing results of p joins to level p by execution of step (4) for $p_x = p-1, p-2, \dots, \lfloor p/2 \rfloor$.
4. Let level p_y be such that $p = p_x + p_y + 1$, i.e., $p_y = p - p_x - 1$.
 - a) If $p_x \neq p_y$ then for each graph element R_1 of level p_x , search level p_y for those elements R_j which can be joined with R_1 .
 - b) If $p_x = p_y$ then assume that elements of level $p_x = p_y$ are $R_1, R_2, \dots, R_u$. For each element $R_i, i=1, 2, \dots, u-1$, of level p_x , search $R_j, j=i+1, i+2, \dots, u$ of the same level p_x for those relations R_j which can be joined with R_i .

Note that relation R_i can be joined with R_j only if $\{r\}_i \cap \{r\}_j$ is empty and there exists " $R_i \Join R_j$ " $\in \{V'\}^q$ such that $R_i \in \{r\}_i$ and $R_j \in \{r\}_j$.

For each element R_j found by either (a) or (b) above, insert a new element R_k to level p . It represents relation R_k arrived at by a join of R_i with R_j in the network node where R_i is located. Set the triplet $(x, c, I)_k$ of the new element representing the relation R_k to:

- x_k is set to x_i ;
- c_k is set to the cumulative cost of R_k ;
- I_k is derived from I_i and I_j .

In addition, $\{r\}_k$ is set to the union of $\{r\}_i$ and $\{r\}_j$. The pointers of the new node to lower graph levels are set to identify the elements representing R_i and R_j . If $x_i \neq x_j$, i.e., R_i and R_j are not located in the same network node, insert an additional graph element to level p . The element represents the join of R_i with R_j executed in the network node where R_j is located, i.e., in the network node x_j .

Figure 3-5 Continued

For the first iteration through Step 3, Step 4 is executed four times in order to create elements of the level 1 which represent results of exactly one join execution (see Figure 3-3(a)). For the second iteration through Step 3, Step 4 is executed 8 times to create elements of the level 2 which represent the results of exactly two join executions (see Figure 3-3(b)). Since the query q processing requires only two joins, the created graph, consisting of levels 0, 1 and 2, represents all strategies available for its processing. Each element of the final level, level 2, represents the result of a unique strategy. For instance, element R_{26} represents a strategy shown in Figure 3-4.

The algorithm OPT_DV creates the graph representing all available strategies under assumptions of Chapter 1 (see Figure 1-3) and section 3-1. In addition, the graph is such that it does not contain any redundant elements. The algorithm avoids redundancy by taking into consideration all of the available information any time a new graph element, which represents a sub-strategy, is created.

Two theorems are presented in support of the above statements which claim that the graph is complete and that no graph element is redundant.

Theorem 3-1: Under the assumptions of Figure 1-3 and section 3-1, Algorithm OPT_GV creates the complete graph of strategies for a given query.

Proof: The proof is given using induction on levels by showing that level "p" contains elements which represent results of all strategies to process p joins.

(1) Level 0 contains all elements representing the original relations referenced by the query - it is complete.

(ii) Level 1 is created from level 0 by comparing each element of level 0 with all of its other elements. If a join of two relations R_i and R_j can be performed then two new elements, which represent the result of their join executed at network locations of R_i and R_j , are added to level 1. Since any two relations can be joined only at one of their locations, level 1 is complete.

(iii) Assume that levels 0, 1, ..., $p-1$ are complete. Level p contains elements representing relations which are results of processing exactly p joins. Any element of level p can only be created from two already existing elements of lower graph levels such that the total number of join executions is exactly p . In other words, if the two elements are at levels p_x and p_y respectively, then it must be that $p_x + p_y + 1 = p$. But steps (3) to (4) ensure that all such elements at level p are created and therefore the level p is complete. QED.

Theorem 3-2: Algorithm OPT_GV does not create any redundant graph nodes.

Proof: It must be shown that each graph element represents a relation which is a partial result of at least one strategy and that all graph elements represent relations obtained by unique strategies.

Level 0 represents original relations referenced by the query and each element of a level $p > 0$ represents a relation which is a result of p joins. Consequently, every graph element represents a relation which is a partial result for at least one strategy.

Two graph elements of different levels represent relations arrived at by a different number of joins, therefore, they represent different partial results. What remains to be shown is that if two or more graph elements represent a relation with identical tuples then they represent different strategies to derive that relation. The proof is by

induction on levels.

(i) Level 0 contains elements representing different original relations with processed restrictions and projections.

(ii) Level 1 contains elements representing relations which are results of one join by comparing all pairs of relations at level 0 in such a way that each pair is compared exactly once. If two relations of level 0 can be joined then two new elements are added to level one. One element represents the result of a join execution in the network node where one operand is located, while the second represents the result of a join execution in the network node of the other operand. If both operands are located in the same network node, only one element is added to level one. Consequently, level 1 contains only elements which represent relations obtained by unique sub-strategies.

(iii) Assume that levels 0, 1, ..., $p-1$ contain elements representing relations obtained by unique strategies. Level p is created from elements of levels p_x and p_y such that $p = p_x + p_y + 1$. Steps (3) and (4) ensure that any element at level p_x is considered for a join with an element of level p_y , such that $p = p_x + p_y + 1$, only once. In addition, any element of level p_x is "joined" with an element of level p_y only once in the network location of the relation of level p_x and once in the location of the relation of level p_y . If the relations of levels p_x and p_y are located in the same network node, only one element is added to level p . Consequently, all elements of level p represent relations obtained by unique strategies. QED.

Theorem 3-1 states that a graph of available strategies developed by Algorithm OPT_GV is complete. If the graph of strategies is created for a given query, then Theorem 3-1 guarantees that it contains the optimal strategy which can be found by a search of the final level for the minimum cost element. The graph of available strategies does not contain any redundant elements according to Theorem 3-2 and, therefore,

redundancy is avoided.

3.4.4 VERTICALLY AND HORIZONTALLY PARTITIONED RELATIONS

Algorithm OPT_GV, which develops the graph of strategies for a given query, assumes that the query is expressed on the global view of the DB. This implies that the query does not reference relations which are partitioned horizontally or vertically. Since realistic applications of a DDB necessitate the use of partitioned relations, the assumption is removed in order to allow a query to be expressed on the distributed view of the DB and Algorithm OPT_GV is adapted for relations which are partitioned horizontally and/or vertically.

A query expressed on the global view of the DB references relations which are called original relations. Once the query is processed by the Distributed View Processor (Toth, 1980, 1982), the query is modified to reference relations of the distributed view of the DB, called elementary relations. Global view relations may be partitioned horizontally and/or vertically as is reflected by the distributed view of the DB. Elementary relations, i.e., relations defined by the distributed view of the DB are not sub-divided any further.

VERTICALLY PARTITIONED RELATIONS

Vertical partitioning ((Chang, 1980), (Mahmoud, 1979)) fragments a relation into sub-relations by attributes. Key attribute(s) are replicated in each sub-relation, while the remaining, non-key attributes are mutually exclusive, i.e., a non-key attribute appears in one sub-relation only. All sub-relations have equal cardinalities determined by the cardinality of the key attribute(s).

Query $q: (R_i.a > R.b) \wedge (R.c < R_j.d)$

Relation R is vertically partitioned into $(1)R$ and $(2)R$.

Query graph (global view):

(a) Query q and its graph (global view of the DB)

(b) Query q modified by the Distributed View Processor

Figure 3-6 Modification of a query referencing a vertically partitioned relation by the Distributed View Processor

Consider a simple query

$$(R_i.a > R.b) \wedge (R.c < R_j.d)$$

shown in Figure 3-6(a) and assume that it is expressed on the global view of the DB. The query references the original relations R_i , R_j , and R . Assume that the relation R has a key attribute "f" and that it is vertically partitioned into sub-relations $(1)R$ and $(2)R$. The key attribute f is replicated in both relations $(1)R$ and $(2)R$, while the

remaining attributes of $(1)R$ are mutually exclusive with those of $(2)R$. When the query is processed by the Distributed View Processor (Toth, 1982), it is modified into an equivalent query expressed on the distributed view of the DB in the following way. An equi-join is included in the query by addition of a two-variable term " $(1)R.f = (2)R.f$ ". In addition, any reference to the relation R in the global view query is replaced by a reference to either $(1)R$ or $(2)R$, depending on which one contains the joining attribute.

Assuming that, in the example query of Figure 3-6, $(1)R$ contains the attribute b while $(2)R$ contains the attribute c , the query becomes

$$((1)R.f = (2)R.f) \wedge (R_i.a > (2)R.b) \wedge ((2)R.c < R_j.d)$$

with its query graph shown in Figure 3-6(b).

The method for transformation of a query's reference to a vertically partitioned relation R from the global view into the partitioned view is now given.

For every relation R with a key attribute(s) f , which is vertically partitioned into sub-relations $(k)R$, $k=1,2,\dots,m$, modify the query in the following way:

(i) Add the following two-variable terms

$$((1)R.f = (2)R.f) \wedge \dots \wedge ((m-1)R.f = (m)R.f)$$

to the query.

(ii) Replace every reference to the relation R within any two-variable term " $R.a \Theta R_i.b$ " by a reference to that fragment $^{(k)}R$, which contains the joining attribute a . The two-variable term becomes " $^{(k)}R.a \Theta R_i.b$ ". If the joining attribute is the key attribute f then the reference to R can be replaced by a reference to any one of the fragments $^{(k)}R$, $k=1,2,\dots,m$.

Theorem 3-3: If a query q expressed on the global view DB is transformed into a query q' expressed on the distributed view of the DB in the way described above then the queries q and q' are equivalent.

Proof: It must be shown that the result of query q applied against the global view of the DB is the same as the result of query q' applied to the distributed view of the DB.

Every relation R with a key attribute(s) f , which is vertically partitioned into sub-relations $^{(k)}R$, $k=1,2,\dots,m$, results in addition of the following two-variable terms to the query:

$$(^{(1)}R.f = ^{(2)}R.f) \wedge \dots \wedge (^{(m-1)}R.f = ^{(m)}R.f)$$

If these additional two-variable terms are processed first, then after the removal of unnecessary attributes their result is exactly the relation R . The remainder of the processing is exactly the same as if the query q was executed on the global view of the DB. QED.

HORIZONTALLY PARTITIONED RELATIONS

Horizontal fragmentation partitions a relation into sub-relations according to the value of the relation's key. Attributes of all sub-relations are identical while tuples of one sub-relation are

mutually exclusive with tuples of any other sub-relation. The cardinality of the relation is equal to the sum of the cardinalities of its sub-relations.

Unfortunately, horizontal partitioning can not be processed as elegantly as vertical partitioning. The Distributed View Processor expands each horizontally fragmented relation as shown in Figure 3-7(a). The QPS formulator considers the following options when a join of two relations under examination has one or both of its operands partitioned horizontally.

Query q : $(R_1.a > R.b) \wedge (R.c < R_2.d)$

Relation R is horizontally partitioned into R' and R'' .

Query graph (global view):

(a) Query q and its graph (global view of the DB)

(b) Query q modified by the Distributed View Processor

Figure 3-7 Modification of a query referencing a horizontally partitioned relation by the Distributed View Processor

CASE 1: Let the two-variable term to be processed by the join execution be

$$(R_1.a \theta R.b), \quad \dots (3-1)$$

where R_1 is not partitioned and R is horizontally partitioned into

sub-relations $R^{(k)}$, $k=1,2,\dots,m$. The formulator chooses one of the following two options:

- (i) A union of fragments is executed in one of the fragments' locations followed by a join, i.e.,

$$R = \bigcup_{k=1}^m R^{(k)}$$

and (3-1) becomes

$$(R_1.a \Theta (\bigcup_{k=1}^m R^{(k)}.b)).$$

- (ii) A fragmented join is executed. The non-fragmented relation is moved to all horizontal fragments where it is joined with them. The result is a relation which is still horizontally partitioned and is ready for further processing, perhaps by another fragmented join. Eventually, however, before the end of query execution the partitions will be combined into one relation by a union operation. In this case (3-1) becomes

$$(\bigcup_{k=1}^m (R_1.a \Theta R^{(k)}.b)).$$

CASE 2: Both of the join operands are partitioned. In this case a union of all fragments is formed for one of the operands. For instance, if the two-variable term is

$$(R_1.a \Theta R_j.b) \quad \dots (3-2)$$

where R_1 is horizontally partitioned into $R_1^{(x)}$, $x=1,2,\dots,m_1$ and R_j into $R_j^{(y)}$, $y=1,2,\dots,m_j$ then either

(i) a union of fragments of R_i is formed in a network location of one of the fragments of R_i , in which case (3-2) becomes

$$\left(\bigcup_{y=1}^m \left(\left(\bigcup_{x=1}^m R_i^{(x)} \right) .a \ \Theta \ R_j^{(y)} .b \right) \right), \text{ or}$$

(ii) a union of fragments of R_j is formed in a network location of one of the fragments of R_j , in which case (3-2) becomes

$$\left(\bigcup_{x=1}^m \left(R_i^{(x)} .a \ \Theta \ \left(\bigcup_{y=1}^m R_j^{(y)} \right) .b \right) \right)$$

CASE 2 is thus reduced to CASE 1.

Theorem 3-4: Processing of a query defined on the distributed view of the DB, as described above, is equivalent to the processing of the same query but defined on the global view of the DB. In other words, the processing described above is correct.

Proof: Consider CASE 1 (i). A union of fragments is formed first, followed by a join of two, non-fragmented relations. Obviously, this is correct.

Consider CASE 1 (ii). A fragmented join is formed first, which is eventually followed by a union of the fragmented result. This sequence of operations is correct due to the distributivity of the join over a union. For a two-variable term " $R_i .a \ \Theta \ R_j .b$ ", where R_i is horizontally partitioned into sub-relations $R_i^{(k)}$, $k=1,2,\dots,m$, due to the distributivity:

$$\left(\left(\bigcup_{k=1}^m R_i^{(k)} \right) .a \ \Theta \ R_j .b \right) = \left(\bigcup_{k=1}^m \left(R_i^{(k)} \ \Theta \ R_j .b \right) \right)$$

Finally, in CASE 2 both operands of a join under consideration are horizontally fragmented. This case is reduced to CASE 1 by a union of one of the partitioned operands. QED.

3.4.5 ALGORITHM FOR A DISTRIBUTED VIEW QUERY

Algorithm OPT_GV is now adapted to create a graph of available strategies for a query expressed on the distributed view of the DB. The adapted algorithm is called OPT_DV. (Optimal for a Distributed View). A query can reference relations which are partitioned either horizontally and/or vertically. The Distributed View Process sufficiently modifies a query (section 3.4.4) before it is submitted to the QPS formulator, so that its vertical partitioning is accounted for by the use of equi-joins. Consequently, a QPS formulator does not need to take any special action due to the vertical partitioning of relations and the Algorithm OPT_GV does not need to be modified either. However, the algorithm must be changed so that the graph of available strategies it produces represents processing of horizontally partitioned relations. Since the following discussion pertains to the horizontal partitioning of relations only, any subsequent reference to a partition/fragment implies a horizontal, not a vertical, partition/fragment.

To represent partitioned relations, the elements of a graph of available strategies represent relations or partitions. In a graphical representation, a partition is identified in the usual way by a right super-script such as in $R_i^{(2)}$, which is a second partition of R_i .

A graph level p consists of elements representing relations which are results of exactly p joins and/or unions. Recall from the section 3.4.4 that a fragmented relation can be processed in one of two ways. Either it is an operand of a fragmented join whose result is also partitioned, or all of its fragments are consolidated by a union.

(i) Fragmented Join: Algorithm OPT_GV creates an element, which represents a relation R , at level p from two elements R_i and R_j of levels p_x and p_y , respectively. Relations R_i and R_j must be such that

they can be joined to give R and, in addition, p_x, p_y must satisfy the relationship $p = p_x + p_y + 1$. These two conditions also hold if partitioned relations are allowed. Each level p contains elements which represent results of exactly p join and/or union executions. If a relation R , which is at level p , is a result of a join of non-partitioned relations R_i and R_j at levels p_x and p_y respectively then p_x and p_y must satisfy $p = p_x + p_y + 1$. Assuming that R_i is partitioned into $R_i^{(k)}$, $k=1,2,\dots,m$, and that a join under consideration is to process " $R_i.a \Theta R_j.b$ ", then a fragmented relation R is created at level p by a fragmented join. It consists of m partitions $R^{(k)}$ such that

$$R^{(k)} = (R_i^{(k)}.a \Theta R_j.b), k=1,2,\dots,m.$$

The network location of $R^{(k)}$ is the same as that of $R_i^{(k)}$.

(ii) Union: When a new graph level p is created, for any relation R_i , which consists of fragments $R_i^{(k)}$, $k=1,2,\dots,m$, at level $p-1$, m new elements are created at level p . Each element represents the union of fragments

$$R = \left(\bigcup_{k=1}^m R_i^{(k)} \right),$$

such that each union is executed in one of the network locations of fragments $R_i^{(k)}$, $k=1,2,\dots,m$.

The graph representation of fragmented joins and unions is shown in Figure 3-8. Each graph element of level 0 represents one of the elementary relations referenced by the query. An element/relation of level $p > 0$ represents a result of one of the following operations:

Figure 3-8 Graph representation of fragmented joins and unions

(i) a join of two non-partitioned relations, in which case it has two edges to lower graph levels, connecting it to the operands of the join which produces it;

(ii) a fragmented join, in which case it also has two edges to lower graph levels which connect it to the join operands which produces it; one edge connects it to a non-fragmented relation while the other to a single partition; and

(iii) a union of fragments, in which case the element has one edge to each of the partitions at level $p-1$ from which it is created.

Note that the graph representation of a fragmented join result is similar to that of a join of non-partitioned relations; however, since one of the operands of a fragmented join is a partition, its result has exactly one edge which connects it to that partition.

Also note that a graph element at level p which represents the result of a union of fragments R_k^k , $k=1,2,\dots,m$, has m edges connecting it to the partitions from which it is created. Each edge connects it to one of the fragments R_k at level $p-1$. In a pictorial representation,

- (i) if a fragment is moved for the purpose of a union, the edge between the union result and that fragment is a resistor symbol, and
- (ii) all m edges pass through a rectangle which is marked with a "union" (see Figure 3-8).

The graph of strategies, shown in Figure 3-8, is for a query referencing relations R_1 , R_2 , R_3 and R_4 . Relation R_4 is horizontally partitioned into $R_4^{(1)}$ and $R_4^{(2)}$, located in Z and W respectively.

Relations R_1 , R_2 and R_3 are located in network nodes W, X and Y, respectively.

Only three elements (out of many) of level 1 are shown. One element is R_{11} , which is the result of a join of R_1 with R_2 in the network node X. The other two elements represent results of a union. R_{12} represents the union of fragments of $R_4^{(1)}$ and $R_4^{(2)}$, executed in Z, while R_{13} represents a union of the same fragments but executed in W.

Note that R_{12} has one edge connecting it to the fragment $R_4^{(1)}$ and one edge connecting it to $R_4^{(2)}$. As R_{12} is the result of a union of the two fragments, executed in the network location Z, the edge between R_{12} and $R_4^{(2)}$ is a "resistor" symbol. It denotes the transfer of $R_4^{(2)}$ from W to Z for the purpose of a union execution.

Similarly, R_{13} also has two edges to lower graph levels, to the same fragments $R_4^{(1)}$ and $R_4^{(2)}$. However, since R_{13} represents the union executed in W, the edge between R_{13} and $R_4^{(1)}$ is a resistor symbol; it represents the transfer of $R_4^{(1)}$ from Z to W, for the purpose of a union.

The result of a fragmented join of R_{11} with R_4 is shown at level 2. $R_{21}^{(1)}$ represents the transfer of R_{11} to the network node Z, followed by its join with the fragment $R_4^{(1)}$. $R_{21}^{(2)}$ represents the transfer of R_{11} to W, followed by its join with $R_4^{(2)}$. Both, $R_{21}^{(1)}$ and $R_{21}^{(2)}$ are fragments of R_{21} . As they are results of two joins, of which one is fragmented, they are at the graph level 2. Each has two edges connecting it to the operands (of level 1) from which it is created.

Relation R_{22} at level 2 represents the result of a join of R_2 with R_{12} in the network node Z.

Elements $R_{31}^{(1)}$ and $R_{31}^{(2)}$, shown at level 3, represent the result of a fragmented/join of R_3 with a partitioned relation R_{21} .

Finally, there are two elements shown at the final level, level 4. Element R_{41} represents the union of fragments of R_{31} performed in Z; R_{42} represents the union of the same fragments but executed in W.

Each element of level 4 represents the query result obtained by a unique strategy. The strategy which is represented by an element of level 4 can be obtained by recursively following its edges towards the initial level (level 0). For instance, element R_{42} represents the strategy:

move R_1 to X and join it with R_2 in X to give R_{11} ;
 move R_{11} to Z and W where it is joined with $R_4^{(1)}$ and $R_4^{(2)}$ to give $R_{21}^{(1)}$ and $R_{21}^{(2)}$, respectively;
 move R_3 to Z and W where it is joined with $R_{21}^{(1)}$ and $R_{21}^{(2)}$ to give $R_{31}^{(1)}$ and $R_{31}^{(2)}$, respectively;
 move fragment $R_{31}^{(1)}$ to W and perform a union of fragments $R_{31}^{(1)}$ and $R_{31}^{(2)}$ to give the query result represented by R_{42} .

NUMBER OF LEVELS - NUMBER OF REQUIRED JOINS AND/OR UNIONS

The graph of strategies for a query referencing horizontally partitioned relations includes elements representing results of fragmented joins and unions. Obviously, a fragmented join does not

increase the number of graph levels in comparison to the graph of strategies for the same query expressed on the global view of the DB, but a union does. If a query references N_h horizontally fragmented relations then the number of graph levels is increased by N_h .

Note that in this context a union of fragments of a relation R_i , consisting of m horizontal partitions, is considered to be one operation, even though it consists of a sequence of $m-1$ unions. Similarly, although a fragmented join includes a number of join executions, one per fragment, it is also considered to be one operation. For example, consider a relation R_i , partitioned into horizontal fragments $R_i^{(k)}$, $k=1,2,\dots,m$. A union of fragments $\bigcup_{k=1}^m R_i^{(k)}$ is considered to be one operation. Similarly, a fragmented join $(R_i^{(k)} \Join R_j)$, $k=1,2,\dots,m$, is also considered to be one operation.

Theorem 3-5: Assuming that

- (i) a query q references N_r relations out of which N_h are horizontally and N_v are vertically partitioned and
- (ii) the number of partitions of a vertically partitioned relation R_i is denoted by n_i

then the number of joins, of which some may be cascaded, and/or unions which are necessary to process the query q , is:

$$N_o = N_h + N_s - 1, \text{ where } N_s = N_r + \sum_{i=1}^{N_v} (n_i - 1)$$

Proof: Theorem 2-1 states that a query q referencing N_r relations expressed on the global view of the DB requires $N_j = N_r - 1$ joins for its execution. If the query is modified by the Distributed View Processor to account for vertically partitioned relations by insertion of

equi-joins which reference vertical partitions as described in the section 3.4.4, then the number of relations, of which some are vertical partitions, referenced by q becomes:

$$N_s = N_r + \sum_{i=1}^{N_v} (n_i - 1) = N_r + \sum_{i=1}^{N_v} (n_i) - N_v$$

If q is now processed then the number of joins, N_j , of which some may be cascaded, necessary to process the query has increased due to the inserted equi-joins:

$$N_j = N_s - 1 = (N_r + \sum_{i=1}^{N_v} (n_i) - N_v) - 1$$

Assume that q is further modified to include horizontal partitions of N_h relations. If q is now processed in such a way that any fragmented relation is first consolidated by a union then the number of operations, N_o , ie., the number of unions and/or joins, of which some may be cascaded, is:

$$N_o = N_h + N_s - 1$$

Consider, however, a simple two-variable term " $R_i.a \Theta R_j.b$ " in which R_i is horizontally fragmented into $R_i^{(k)}$, $k=1,2,\dots,m$. It can be processed in two ways:

(i) a fragmented join is executed first, followed by a union, ie.,

$$(R_i.a \Theta R_j.b) = \bigcup_{k=1}^m (R_i^{(k)}.a \Theta R_j.b), \text{ or}$$

(ii) a union of fragments of R_i is formed first, followed by a join, ie.,

$$(R_i.a \Theta R_j.b) = (\bigcup_{k=1}^m R_i^{(k)}).a \Theta R_j.b.$$

In either case, the total number of operations is two, one fragmented join and one union of fragments. The total number of operations does not depend on the sequence in which they are executed and, as a result, the total number of operations, i.e., unions and/or joins, of which some may be cascaded and/or fragmented, is:

$$N_0 = N_h + N_s - 1, \quad \text{where } N_s = N_r + \sum_{i=1}^N (n_i) - N_v$$

QED.

Level p of the graph of query strategies represents relations which are results of exactly p operations. A graph will, therefore, have $N_0 + 1$ levels, where N_0 is determined by Theorem 3-5.

ALGORITHM

Algorithm OPT_DV, shown in Figure 3-9, constructs a graph of available strategies for a given query which may reference partitioned relations. Each graph element consists of:

- a) Pointers to elements of lower graph levels from which the element is created. If a relation is the result of a join then there are two pointers which identify the join operands. If the relation is a result of a union of horizontal partitions, there is a pointer to each of the partitions from which it is created by a union.
- b) Pointer to the next element of the same level.
- c) If the element represents a horizontal partition, it contains two pointers of a two-way linked list of partitions. In other words, partitions of a horizontally partitioned relations are chained together on a two-way linked list;

ALGORITHM OPT_DV TO CREATE THE GRAPH OF AVAILABLE STRATEGIES
FOR A DISTRIBUTED VIEW QUERY

③

INPUT: Query q defined by $\{V\}^q$, $\{V''\}^q$ and $\{(x,c,I)\}^q$, where

- a) $\{V\}^q$ is the set of one-variable terms for query q ;
- b) $\{V''\}^q$ is the set of two-variable terms for query q . Note that the query is assumed to be modified to account for vertically partitioned relations by inclusion of equi-joins as described in the section 3.4.4;
- c) $\{(x,c,I)\}^q$ is a set of triplets (x,c,I) , one per each non-partitioned relation or a horizontal fragment.

(i) The triplet for a non-partitioned relation R_1 is denoted by $(x,c,I)_1$ and it conveys information about relations R_1 , where

x_1 is the network location of R_1 ;

c_1 is the cumulative cost of R_1 which is zero; and

I_1 is information about relation R_1 which is necessary for the size and cost estimation.

(ii) The triplet for a partition $R_1^{(k)}$ of relation R_1 , horizontally partitioned into $R_1^{(k)}$, $k=1,2,\dots,m$, is denoted by $(x,c,I)_1^{(k)}$ and it conveys information about the fragment $R_1^{(k)}$, where

$x_1^{(k)}$ is the network location of $R_1^{(k)}$;

$c_1^{(k)}$ is the cumulative cost of $R_1^{(k)}$ which is zero; and

$I_1^{(k)}$ is information about $R_1^{(k)}$ which is necessary for the size and cost estimation.

OUTPUT: The graph of strategies for processing q .

METHOD: The graph of strategies is created by levels. Each graph element represents either a non-partitioned relation or a fragment. Elements of level 0 represent elementary relations with processed restrictions and projections which remove unnecessary attributes (i.e., attributes which are neither target nor joining attributes). An element of level $p > 0$ represents a relation or a partition, which is a result of p joins (some of which may be cascaded and/or fragmented) and/or unions. A graph element represents either a non-partitioned relation R_1 or a fragment $R_1^{(k)}$ of relation R_1 which is horizontally partitioned into $R_1^{(k)}$, $k=1,2,\dots,m$. Each element contains:

- a) two pointers to graph elements of lower levels from which it is created;
- b) pointer to the next element of the same level;
- c) two pointers of a two-way chain of elements representing a fragmented relation;
- d) set of processed relations $\{r\}$ and a triplet (x,c,I) such that

Figure 3-9 Algorithm to develop the graph of a query strategies for a distributed view of the DB

ALGORITHM/OPT DV TO CREATE THE GRAPH OF AVAILABLE STRATEGIES
FOR A DISTRIBUTED VIEW QUERY

- (i) if the element represents a non-partitioned relation R_i then
 -- the set of processed relations to derive R_i is denoted by $\{r\}_i$ and
 -- the triplet is denoted by $(x, c, I)_i$ and it is defined as above (see input definition);
 (ii) if the element represents a horizontal fragment $R_i^{(k)}$ then
 -- the set of processed relations to derive $R_i^{(k)}$ is denoted by $\{r\}_i^{(k)}$ and
 -- the triplet is denoted by $(x, c, I)_i^{(k)}$ and is defined as above (see input definition).

STEPS

1. Create level zero of the graph of strategies directly from relations referenced by the query and set the cost of each element to the cost of its restrictions and a projection to remove unnecessary attributes.
2. Do step (3) for $p = 1, 2, \dots, N_0$ to create levels $1, 2, \dots, N_0$; where N_0 is the number of necessary operations (joins and/or unions) to process the query.
3. a) Search level $p-1$ for partitioned relations. For each partitioned relation R_i consisting of fragments $R_i^{(k)}$, $k=1, 2, \dots, m$, at level $p-1$, add m elements to level p . Each new element represents $\bigcup_{k=1}^m R_i^{(k)}$ executed at one of the locations of $R_i^{(k)}$. For each new element R_j representing the union of fragments executed at $x_i^{(k)}$ derive the cumulative cost c_j of the triplet $(x, c, I)_j$:
 set x_j to $x_i^{(k)}$;
 set $\{r\}_j$ to $\{r\}_i^{(k)}$;
 I_j is derived from $I_i^{(k)}$, $k=1, 2, \dots, m$.
 b) Insert elements which represent results of join executions, to level p by execution of step (4) for $p_x = p-1, p-2, \dots, \lfloor p/2 \rfloor$.
4. Let level p_y be such that $p = p_x + p_y$, i.e., $p_y = p - p_x - 1$.
 (i) If $p_x \neq p_y$ then for each graph element R_i of level p_x , search level p_y for those elements R_j which can be joined with R_i .
 (ii) If $p_x = p_y$ then assume that elements of level $p_x = p_y$ are $R_1, R_2, \dots, R_u$. For each element R_i , $i=1, 2, \dots, u-1$, of level p_x , search elements R_j , $j=i+1, i+2, \dots, u$, of the same level p_x for those relations R_j which can be joined with R_i . Note that relation R_i or R_j can be fragmented.

ALGORITHM OTD_{DV} TO CREATE THE GRAPH OF AVAILABLE STRATEGIES
FOR A DISTRIBUTED VIEW QUERY

Relation R_i can be joined with R_j only if $|r_i| \cap |r_j|$ is empty and there exists " $R_i \Join R_j$ " $\in |V'|^Q$ such that $R_i \in |r_i|$ and $R_j \in |r_j|$. In addition, at most one of R_i and R_j can be horizontally partitioned.

Assume that the relation R_j found by (i) or (ii) above is such that a join of R_i with R_j is to process a two-variable term $(R_i.a \Join R_j.b)$.

(i) If neither of R_i or R_j is partitioned then insert a new element R_v to level p . It represents the result of a join of R_i with R_j in the network node where R_i is located. Set the triplet $(x, c, I)_v$ of the new element representing the relation R_v to:

x_v is set to x_i ;
 c_v is set to the cumulative cost of R_v ;
 I_v is derived from I_i and I_j .

In addition, $|r|_v$ is set to the union of $|r|_i$ and $|r|_j$. The pointers of the new node to lower graph levels are set to identify the elements representing R_i and R_j . If $x_i \neq x_j$, i.e., R_i and R_j are not located in the same network node, insert an additional graph element to level p . The element represents the join of R_i with R_j executed in the network node where R_j is located, i.e., in the network node x_j .

(ii) If one of R_i or R_j is partitioned then insert elements, which represent the result of a fragmented join of R_i with R_j , to level p . Assume that R_i is partitioned into $R_i^{(k)}$, $k=1, 2, \dots, m$. If it is R_j that is partitioned then the roles of R_i and R_j are exchanged in the remaining portion of this step. There are m new elements inserted to level p ; each represents a fragment $R_v^{(k)} = (R_i^{(k)} .a \Join R_j .b)$, $k=1, 2, \dots, m$, obtained by a join of $R_i^{(k)}$ with R_j executed at the network location $x_i^{(k)}$. For each new element/fragment $R_v^{(k)}$, set the triplet $(x, c, I)_v^{(k)}$ of the new element representing the fragment $R_v^{(k)}$ to

$x_v^{(k)}$ is set to $x_i^{(k)}$;
 $c_v^{(k)}$ is set to the cumulative cost of $R_v^{(k)}$; and
 $I_v^{(k)}$ is derived from $I_i^{(k)}$ and I_j .

In addition, $|r|_v^{(k)}$ is set to the union of $|r|_i^{(k)}$ and $|r|_j$. The pointers of the new node to lower graph levels are set to identify the elements representing $R_i^{(k)}$ and R_j .

d) $\{r\}$... a set of processed elementary relations to derive the relation represented by the element. For example, if the relation is a result of a join of R_1 with R_2 , followed a join with R_3 , then $\{r\} = \{R_1, R_2, R_3\}$.

e) A triplet (x, c, I) conveying information about the relation the node represents, where

x is the network location;

c is the cumulative cost of all CPU processing and data transfer; and

I is information about the relation necessary for the size and cost estimation.

A graph element may either represent a non-partitioned relation or a fragment.

CASE 1: The graph element represents one of the partitions $R_i^{(k)}$, $k=1,2,\dots,m$, of a fragmented relation R_i . In this case its set of processed relations is denoted by $\{r\}_i^{(k)}$ and its triplet by $(x, c, I)_i^{(k)}$. The partition is either an elementary relation or it is the result of a fragmented join. If it, ie., $R_i^{(k)}$, is one of the partitions derived by a fragmented join to process a two-variable term such as " $R_{11}.a \Join R_{12}.b$ ", where R_{11} is horizontally partitioned into $R_{11}^{(k)}$, $k=1,2,\dots,m$, then

$x_i^{(k)} = x_{11}^{(k)}$, $k=1,2,\dots,m$, ie., the network location of a partition

$R_i^{(k)}$ is the same as that of $R_{11}^{(k)}$;

$\{r\}_i^{(k)} = \{r\}_{11}^{(k)} \cup \{r\}_{12}$, ie., the set of processed relations of R_i is the union of sets of relations from which it is derived by a fragmented join; and

$\{r\}_i^{(k)} = \{r\}_i^{(j)}$, $k, j=1, 2, \dots, m$, i.e., the set of processed relations is identical for all fragments.

CASE 2: The graph element represents a non-partitioned relation R_i . In this case its set of processed relations is denoted by $\{r\}_i$ and its triplet by $(x, c, l)_i$. The relation is

- (i) either a result of a join of two non-partitioned relations such as R_{13} and R_{14} , in which case the two pointers of R_i identify R_{13} and R_{14} and $\{r\}_i = \{r\}_{13} \cup \{r\}_{14}$, or
- (ii) it is a union of fragments $R_i^{(k)}$, $k=1, 2, \dots, m$, in which case $\{r\}_i = \{r\}_i^{(k)}$ for all $k=1, 2, \dots, m$.

Algorithm OPT_DV is similar to OPT_GV in that it develops the graph of strategies in levels. Elements of level 0 are created first; they represent the query's elementary relations with processed restrictions and a projection to remove attributes which are neither target nor joining. Then, levels 1, 2, ..., N_0 are created, where N_0 is the number of joins and/or unions necessary to process the query and it is determined by utilization of Theorem 3-5.

Elements of level $p > 0$ are created in two stages. In the first stage, for every partitioned relation R_i , consisting of fragments $R_i^{(k)}$, $k=1, 2, \dots, m$, at level $p-1$, m elements are created at level p . Each represents the union of fragments of R_i , i.e., $\bigcup_{k=1}^m R_i^{(k)}$, but each obtained at a different network location of fragments $R_i^{(k)}$, $k=1, 2, \dots, m$.

In the second stage, elements representing the results of joins of elements of levels p_x and p_y are created. This stage is similar to that of the algorithm OPT_GV with the enhancement that a join may be fragmented. A fragmented join causes an addition of a partitioned relation, consisting of fragments which are chained together, to level p . Two relations R_i and R_j of levels p_x and p_y respectively, result in an element(s) augmented to level p if

$$p = p_x + p_y + 1; \text{ and}$$

R_i and R_j can be joined, i.e.,

if $\{r\}_i \cap \{r\}_j$ is empty;

there exists a two-variable term " $R_s \Theta R_t$ " which is specified by the query such that $R_s \in \{r\}_i$ and $R_t \in \{r\}_j$; and at most one of R_i and R_j is partitioned.

If neither one of R_i and R_j is partitioned then two elements are added to level p ; they represent the join of R_i with R_j executed at network locations of R_i and R_j , i.e., at x_i and x_j respectively.

If one of R_i and R_j is partitioned then elements representing the result of a fragmented join are added to level p . If R_i consists of horizontal partitions $R_i^{(k)}$, $k=1,2,\dots,m$, then m elements are added at level p , each representing a join to process " $R_i^{(k)} \cdot a \Theta R_j \cdot b$ ", executed at a network location of $R_i^{(k)}$, i.e., the network location $x_i^{(k)}$.

Algorithm OPT_DV has the same desirable properties as its less general predecessor, algorithm OPT_GV; it represents all query strategies with no redundant graph nodes. This statement is supported by the following two theorems. Recall that assumptions applicable to algorithm OPT_GV are those of Chapter 1 (Figure 1-3) and sections 3.1 and 3.4.4. However, as the purpose of the algorithm OPT_DV is to

create a graph of strategies for a query expressed on the distributed view of the DB, the assumption of the section 3.1, which specifies that a query is expressed on the global view of the DB, no longer applies.

Theorem 3-6: Algorithm OPT_DV creates the complete graph of strategies to process a given query.

Proof: Algorithm OPT_DV is a modification of algorithm OPT_GV which was shown to produce the complete graph of query strategies. Two modifications were made to OPT_GV.

a) Step (3) was modified to include representation for the processing of fragmented relations by a union. The first stage of the creation of a level $p > 0$ consists of a search of the level $p-1$ for partitioned relations. For each relation R_i , consisting of fragments $R_i^{(k)}$, $k=1,2,\dots,m$, at level $p-1$, m new elements are added to level p . Each new element represents $\bigcup_{k=1}^m R_i^{(k)}$, executed at one of the network locations of fragments $R_i^{(k)}$. Consequently, the graph represents all possible ways to process any partitioned relation by a union.

b) Step (4) was modified to include a representation for processing of fragmented relations by a fragmented join. An element R_i , of level p , which is not a result of a union, represents a join of two relations R_u and R_v of levels p_x and p_y , respectively, such that

$$p = p_x + p_y + 1;$$

$\{r\}_u \cap \{r\}_v$ is empty; and

there exists a two-variable term " $R_s \Theta R_t$ " $\in \{V'\}^Q$, such that

$R_s \in \{r\}_u$ and $R_t \in \{r\}_v$.

Recall that $\{V'\}^Q$ is the set of two-variable terms for the query

q and that if R_u is a relation which is horizontally partitioned into $R_u^{(k)}$, $k=1,2,\dots,m$, then

$$\{r\}_u = \{r\}_v^{(k)}, \text{ for all } k=1,2,\dots,m.$$

Whether relations R_u and R_v , of levels p_x and p_y respectively, can be joined to result in a relation of level $p = p_x + p_y + 1$ does not depend on whether one of R_u , R_v is partitioned or not.

If R_u is partitioned into $R_u^{(k)}$, $k=1,2,\dots,m$, then m elements are added to level p . They represent the result of a fragmented join which conforms to the assumption on a fragmented join execution, specified in the section 3.4.4.

Since

algorithm OPT_DV is a modification of algorithm OPT_GV which creates a complete graph of query strategies;

the first of the two modifications ensures that for any partitioned relation the union of its fragments is represented by the graph of strategies;

a decision whether two relations can or cannot be joined is independent of the fact whether one of them is partitioned; and

the result of a fragmented join is represented in the graph of strategies for any two relations which can be processed by a fragmented join.

the algorithm OPT_DV creates the complete graph strategies for processing a given query. QED.

Theorem 3-7: Algorithm OPT_DV does not create any redundant graph elements.

Proof: As OPT_DV is a modification of the algorithm OPT_GV which does not create any redundant graph elements, what needs to be shown is only

that the modifications do not cause any redundant graph elements.

The first of two modifications made to OPT_GV is in the step (3). For a partitioned relation of level p , a representation of the union of its fragments is inserted at level $p+1$. As the step (3) is executed only once for each graph level $p > 0$, this modification does not introduce any redundant graph elements.

The second modification allows representation of fragmented joins. If two relations R_u and R_v of levels p_x and p_y respectively, where R_u is partitioned into $R_u^{(k)}$, $k=1,2,\dots,m$, can be joined then m elements are inserted to level $p = p_x + p_y + 1$. They represent the result of a fragmented join of R_u with R_v . But steps (3) and (4) ensure that two relations R_u and R_v are considered for a join only once, therefore, any fragmented join is represented only once. QED.

3.4.6 DESTINATION NODE

Algorithms OPT_GV and OPT_DV create the query strategy space for a global and distributed view of the DB, respectively. Both algorithms, however, ignore the fact that a query specifies the network node to which the query result should be delivered. The space of strategies can be used in a QPS formulation provided that it is augmented to account for this destination node. This can be accomplished by inclusion of the following step into the algorithms which create the query strategy space (ie., algorithms OPT_GV and OPT_DV):

STEP 5:

"Examine every element of the final level of the graph of query strategies. Each element represents the query result obtained by a

unique strategy. If the result is not located in the destination node, augment the strategy and its cost by the transfer of the result to the destination node."

If algorithm OPT_DV is modified to include this final step, it can be used for the QPS formulation. For any given query, the QPS formulator first creates the graph of available strategies and then it searches the final graph level for the minimum cost element. The minimum cost element identifies the optimal strategy which can be found by tracing the pointers to the initial elements which are at level 0.

The reason that the above step was not directly included in algorithms OPT_GV and OPT_DV is that the graph of available strategies will be used in examination of some other issues in QP besides the strategy formulation such as utilization of various measures of cost. The inclusion of a query destination node would imply an additional parameter which would have to be considered in the examination of those issues.

3.5 DISCUSSION

This chapter addressed the problem of finding the optimal QPS. Three approaches were examined. The depth-first search and divide and conquer approaches resulted in algorithms which were shown to examine many sub-strategies redundantly. Then two algorithms were proposed; they develop the query strategy space represented by a graph for a global view and for a distributed view of the DB. Algorithms were shown to create the complete space of query strategies and, in addition, all sub-strategies/strategies represented by the graph are unique.

The algorithm OPT_DV creates a space of strategies for a distributed view of the DB, i.e., relations partitioned vertically and/or horizontally can be referenced by queries. The only assumption made on the cost functions was that given information about relations and the cost of their data transfer then the cost of a join or a union result can be estimated. Since the algorithm creates the complete space of query strategies and the cost of each strategy can be calculated using various cost functions, the graph can be utilized in examination of various aspects of QP in a DDB, as will be seen in the following chapters.

In common with most QPS formulators in the literature, this chapter assumes the existence of an accurate estimate of the size of the partial results. In fact, accurate size estimation of the results of relational operations is a difficult problem which is currently an area of active research. Effective implementation of these processing strategies in a working environment therefore depends upon further progress in methods of size estimation.

CHAPTER 4

HEURISTIC ALGORITHM

4.1 INTRODUCTION

Even if a user-issued query does not contain many two-variable terms, the number of strategies available for its processing may be large. Recall from Chapter 3 that if a query references a relation partitioned into N_v vertical fragments, the Distributed View processor inserts $N_v - 1$ equi-joins into the query and thus enlarge the space of the query strategies. Similarly, a horizontally partitioned relation referenced by the query can be processed by either a union or a fragmented join. This further enlarges the space of strategies. Consequently, a user query which references partitioned relations may have a large strategy space and it may be necessary to use a heuristic algorithm for a QPS formulation. Indeed, if proposed algorithms for formulation of a QPS in an operational environment are examined (see Table 1-1), it can be observed that all algorithms are heuristic in nature. The use of a heuristic algorithm for formulation of a QPS is also supported by the fact that the problem of QPS formulation is NP-hard.

A simple heuristic algorithm for a QPS formulation, derived from the algorithm OPT_DV which creates a graph of all strategies available for processing of a given query, is presented in this chapter. The algorithm is called HEU_DV (HEUuristic for Distributed View) and it is based on a partial graph of strategies.

4.2 ALGORITHM HEU_DV

The heuristic is derived from algorithm OPT_DV and, provided that the algorithm OPT_DV is understood, it is quite simple. It first creates levels 0, 1, ..., N of the graph of query strategies in exactly the same way as the algorithm OPT_DV does. Each graph element of level N represents a partial result obtained by a unique strategy comprised of N joins and/or unions. Level N is searched for the minimum cost element and the unique sub-strategy it represents is adopted as a part of the strategy for processing the given query. The query itself is then modified in such a way that relations processed by the adopted sub-strategy are replaced by the sub-strategy's result. In addition, two-variable terms processed by the adopted sub-strategy are removed and the modified query is then subjected to the same process described above. N+1 graph levels are created for the modified query and the minimum cost element of level N is found. The sub-strategy represented by the minimum cost element is adopted as a part of the final strategy, the query is modified and the process repeats again; it is illustrated by the following example.

Consider a query

$$(R_1.a \theta_1 R_2.b) \wedge (R_2.c \theta_2 R_3.d) \wedge (R_3.e \theta_3 R_4.f),$$

whose graph is shown in Figure 4-1(a) and whose processing strategy is to be determined by the heuristic algorithm with N=2. The heuristic constructs levels 0, 1 and 2 of the graph of query strategies (see Figure 4-1(b)) and searches level 2 for the minimum cost element. The sub-strategy represented by the minimum cost element and shown in Figure 4-1(c) is adopted as a part of the QPS and the query is modified. Since the adopted sub-strategy processes two-variable terms $(R_1.a \theta_1 R_2.b)$ and $(R_2.c \theta_2 R_3.d)$, they are removed from the query and relations R_1 , R_2 and R_3 are replaced by the sub-strategy result,

a) Query graph for $(R_1.a \oplus R_2.b) \oplus (R_2.c \oplus R_3.d) \oplus (R_3.e \oplus R_4.f)$

b) Levels 0, 1, and 2 of the graph of strategies for the query in (a)

c) Timing diagram for the adopted sub-strategy

Figure 4-1 First iteration of the heuristic algorithm HBU_DV

relation R_{25} . The modified query is shown in Figure 4-2(a).

The process described above is then repeated for the modified query. Levels 0, 1, ..., N, where $N=2$, are constructed as shown in Figure 4-2(b). However, as the modified query is processed by one join, only levels 0 and 1 are constructed. The minimum cost element of level 1 is found and its unique sub-strategy (see Figure 4-2(c)) is adopted as a part of the QPS. The combined sub-strategies of figures 4-1(c) and 4-2(c) constitute the strategy for processing the given query; it is shown in Figure 4-2(d).

Level N of a partial graph of query strategies may contain a elements which represent a horizontally partitioned relation spanning several information processors. When the heuristic searches the final level of the graph of strategies for the minimum cost element, it considers a fragmented relation as one relation whose cost is defined to be the combined cost of its fragments. In this way, the heuristic makes a trade-off among the following possible alternatives in processing a fragmented relation:

- (i) the fragments may be consolidated by a union whose result is then an operand of a join;
- (ii) the partitioned relation is an operand of a fragmented join followed by a union; and
- (iii) the partitioned relation is an operand of a fragmented join which is then followed by another fragmented join.

For instance, consider a query

$$q = (R_1.a \theta_1 R_2.b) \wedge (R_2.c \theta_2 R_3.d),$$

minimum cost element

(b) Graph of strategies for the modified query of (a)

(c) Timing diagram for the adopted sub-strategy

(d) Timing diagram for the OPS -- combined sub-strategies

Figure 4-2 Second iteration of algorithm HEU_DV

where R_1 is horizontally partitioned into $R_1^{(1)}$ and $R_1^{(2)}$, located in W and X respectively, and relations R_2 and R_3 are located in Y and Z respectively. The first three levels of the query's strategies graph is shown in Figure 4-3(a). Note that only some of the elements of levels 1 and 2 are shown.

(i) Element R_{23} represents a relation obtained by a union of fragments followed by a join.

(ii) Elements $R_{22}^{(1)}$ and $R_{22}^{(2)}$ represent a partitioned relation obtained by two fragmented joins.

(iii) Element R_{21} represents a relation obtained by a fragmented join followed by a union.

After the partial graph of strategies is constructed, a search of the final graph level is performed to locate the minimum cost partial result. If the result is horizontally fragmented, it is considered to be one relation of which cost is the combined cost of its fragments. In this way a trade-off among the alternatives for the processing of a fragmented relation as listed above, can be made.

Assume that the minimum cost partial result is that of a sub-strategy which executes a fragmented join of R_1 with R_2 followed by another fragmented join (see Figure 4-3(a)), i.e., case (ii) above. The sub-strategy for the fragmented partial result, comprised of $R_{22}^{(1)}$ and $R_{22}^{(2)}$, and of which timing diagram is shown in Figure 4-3(b), is adopted as a part of the QPS. The query is modified by replacement of relations processed by the sub-strategy with its partial result consisting of fragments $R_{22}^{(1)}$ and $R_{22}^{(2)}$. Again, levels 0, 1, ..., N of a graph of strategies are constructed for the modified query as shown in Figure 4-3(c). The final portion of the strategy is then determined by locating the minimum cost element of the final graph level. It

minimum cost partial result partitioned relation.

LEVEL 1:

R_1 is the result of a union of R_1 and X^{R_2} , executed in W

R_2 and X^{R_2} are the result of a fragmented join of R_2 with R_1 and X^{R_2}

I^R_3 is the result of a union of R_1 and X^{R_2} , executed in X

LEVEL 2:

W^R_2 is the result of a union of R_2 and X^{R_2} , executed in W

W^R_22 and X^R_22 are the result of a fragmented join of R_2 with R_2 and X^{R_2}

Y^R_23 is the result of a join of X^{R_2} with R_2 , executed in Y

a. Partial graph of strategies for a query referencing a horizontally fragmented relation

Figure 4-3 Partitioned relations processed by HEU_DV

(b) Timing diagram for the chosen sub-strategy whose result is a horizontally partitioned relation R_{22}

(c) Graph of strategies for the modified query with $R_{22}^{(1)}$ and $R_{22}^{(2)}$

(d) Timing diagram for the adopted sub-strategy

identifies a sub-strategy shown in Figure 4-3(d). The QPS is the synthesis of the two, found sub-strategies of figures 4-3(b) and 4-3(d), respectively.

The heuristic algorithm HEU_DV for formulation of a QPS is shown in Figure 4-4. Algorithm OPT_DV which is used to construct a partial graph of query strategies necessitates three modifications, one to its input, one to its first step, and one to its second step.

1. Algorithm OPT_DV, shown in Figure 3-9, has as its input a query q which is defined by $\{V'\}^q$, $\{V''\}^q$ and $\{(x,c,I)\}^q$. $\{(x,c,I)\}^q$ specifies information about elementary relations referenced by q while $\{V'\}^q$ and $\{V''\}^q$ are sets of the query's one and two-variable terms, respectively. There are two modifications in the input specification:

- (i) The query is defined in exactly the same way for the modified algorithm OPT_DV with one exception. The cumulative cost $c_i \in (X,c,I)_i$ of a relation R_i (could be a fragment) is no longer assumed to be zero; it may have some initial value.
- (ii) There is an additional input value N which specifies that $N+1$ levels of the graph of query strategies should be created.

2. Step (1) of the modified algorithm OPT_DV creates level 0 whose elements represent elementary relations referenced by one and two-variable terms in $\{V'\}^q$ and $\{V''\}^q$, respectively. The cost of each relation R_i (which may be a fragment) represented by an element of level 0 is its initial cost $c_i \in (X,c,I)_i$ plus the cost of any restrictions and/or projections to remove attributes which are not necessary for further processing of q .

HEURISTIC ALGORITHM HEU_DV FOR QPS FORMULATION

INPUT: N ... the number of levels of a partial graph of strategies and a query q defined as for algorithm OPT_DV, i.e., q is defined by $\{V\}^q$, $\{V''\}^q$ and $\{(x,c,I)\}^q$, where

- a) $\{V\}^q$ is a set of one-variable terms for query q ;
- b) $\{V''\}^q$ is a set of two-variable terms for query q ;
- c) $\{(x,c,I)\}^q$ is a set of triplets (x,c,I) , one per each non-partitioned relation or a horizontal fragment.

Note that the query is assumed to be modified to account for vertically partitioned relations by inclusion of equi-joins as described in the section 3.4.4. For further details about the input, see algorithm OPT_DV in Figure 3-10.

OUTPUT: QPS and its cost.

METHOD: There are three repeatedly executed phases.

- (i) Let algorithm OPT_DV construct levels $0, 1, \dots, N$ of the graph of strategies for query q .
- (ii) Find the minimum cost partial result of the final level of the graph of strategies. Save the sub-strategy for the minimum cost partial result and adopt it as a portion of the QPS.
- (iii) Modify the query by removal of relations and two-variable terms processed by the adopted sub-strategy and repeat (i) to (iii).

STEPS:

1. Invoke the modified algorithm OPT_DV to construct levels $0, 1, \dots, N$ of the graph of strategies.

Input: N , and the query q defined by $\{V\}^q$, $\{V''\}^q$ and $\{(x,c,I)\}^q$.

Output: Levels $0, 1, \dots, N$ of the graph of query strategies, such that

$N_g = N$, or

$N_g < N$, in which case the graph of strategies for the input query is complete. It represents all available and complete strategies for processing the input query q and $N_g - N_0$ is the number of joins and/or unions necessary to process the query input to the algorithm OPT_DV.

To ensure that restrictions one-variable terms in $\{V\}^q$ are applied only once, take $\{V''\}^q$ into an empty set.

2. Search level N_g for the minimum cost partial result; at level N_g horizontal partitions at level N_g are considered to be one partial result whose cost is a combined cost of all of its fragments. Save the sub-strategy for the minimum cost partial result and adopt it as a part of the QPS. Denote the chosen sub-strategy by QPS'. Assume that relations processed by the sub-strategy QPS' are $R_j, j=1,2,\dots,n$ and that the result of QPS' is a relation R_1 . Modify $\{(x,c,I)\}^q$ and $\{V''\}^q$ in the following way.

(i) Replace information in $\{(x,c,I)\}^q$ for relations processed by the chosen sub-strategy by information about its (sub-strategy's) result.

(ii) Remove from $\{V''\}^q$ two-variable terms processed by the adopted sub-strategy QPS'.

(iii) For any two-variable term $R_j \theta R_k \in \{V''\}^q$, if $R_j \in \{R_j, j=1,2,\dots,n\}$, i.e., if R_j is one of the relations processed by QPS', then replace R_j with R_1 . In other words, any reference in the query's two-variable terms $\{V''\}^q$ to a relation processed by the adopted sub-strategy is replaced by a reference to R_1 , the sub-strategy's result.

3. If $\{V''\}^q$ is not empty, repeat steps (1) to (3); otherwise stop -- the strategy for processing the query q is the synthesis of sub-strategies adopted in executions of step (2).

Figure 4-4 Heuristic algorithm HEU_DV

3. Finally, since only $N+1$ levels of the graph of query strategies are to be created, step (2) is modified to "Do step (3) for $p=1,2,\dots,N$ ".

4.3 IMPROVED ALGORITHM NR_HEU_DV

The heuristic algorithm HEU_DV repetitively builds $N+1$ levels of the graph of query strategies, each time choosing and adopting a sub-strategy for a minimum cost partial result at level N . The query is then modified to remove two-variable terms processed by the adopted sub-strategy and relations processed by the sub-strategy are replaced by its result. On the next iteration, $N+1$ levels of the graph of strategies are again built for the modified query; however, some of the new elements will be identical to those of the previous iteration and are created redundantly. The graph elements which do not vary from one iteration to another are those that are not a part of the adopted sub-strategy.

Consider a query

$$(R_1.a \theta_1 R_2.b) \wedge (R_2.c \theta_2 R_3.d) \wedge (R_3.e \theta_3 R_4.f),$$

shown in Figure 4-5(a). On the first iteration of the HEU_DV algorithm, with $N=1$, levels 0 and 1 of the graph of strategies are constructed as shown in Figure 4-5(b). The minimum cost partial result at level 1 represents a sub-strategy to process the two-variable term $(R_1.a \theta_1 R_2.b)$. The query is modified to reflect the adopted sub-strategy as shown in Figure 4-5(c). Instead of re-constructing all elements of level 1 on the next iteration of the heuristic, the graph can be modified by removal of elements which represent relations processed by the adopted sub-strategy and then use the modified graph of the query strategies as input for the next iteration. For the

(a) Query $(R_1.a \ Q_1 \ R_2.b) \ \wedge \ (R_2.c \ Q_2 \ R_3.d) \ \wedge \ (R_3.e \ Q_3 \ R_4.f)$

(b) Partial graph of strategies created by the first iteration of NR_HEU_DV

(c) Modified query with R_{12} replacing R_1 and R_2

(d) Modified graph of strategies with R_{12}

(e) Partial graph of strategies of the second iteration

Figure 4-5 Two iterations of algorithm NR_HEU_DV

example query, the chosen sub-strategy processes relations R_1 and R_2 . Consequently, elements of the graph that represent processing of any one of relations R_1 or R_2 will be removed from the graph and the renamed sub-strategy's result is moved to level 0 as shown in Figure 4-5(d).

On the next iteration, only those elements of level 1 which represent processing of the element newly inserted to level 0, i.e., the chosen, minimum cost element of the previous iteration, must be created. The partial graph of strategies is shown in Figure 4-5(e).

The improved, non-redundant heuristic called NR_HEU_DV (Non-Redundant HEuristic for Distributed View) is shown in Figure 4-6. Although it is still based on the algorithm OPT_DV, the modifications necessary to avoid the described redundancy are such that the steps of algorithm OPT_DV must be included in the heuristic algorithm itself.

4.4 NUMBER OF ITERATIONS

Assume that processing of a query requires N_0 joins and/or unions. Since each iteration executing steps (2) to (6) determines a sub-strategy consisting of N joins and/or unions, the total number of iterations through steps (2) to (6) is N_0/N .

4.5 DISCUSSION

The heuristic algorithm can justifiably be called a "greedy" heuristic. It develops a partial graph of strategies for a given query and adopts the sub-strategy with a minimum cost result as a part of the QPS. The query is modified to reflect the commitment to the sub-strategy. This modification results in a decrease of the query's

NON-REDUNDANT HEURISTIC ALGORITHM NR_HEU_DV FOR QPS FORMULATION

INPUT: N ... the number of levels of a partial graph of strategies and a query q defined as for algorithm HUE_DV of Figure 4-4 or algorithm OPT_DV of Figure 3-10.

OUTPUT: QPS and its cost.

METHOD: There are three repeatedly executed phases:

- (i) Construct levels $0, 1, \dots, N$ of the graph of query strategies.
- (ii) Find the minimum cost partial result of the final level of the graph of strategies. Save the sub-strategy for the minimum cost partial result and adopt it as a portion of the QPS.
- (iii) Modify the query by removal of relations and two-variable terms processed by the adopted sub-strategy. Remove elements derived from any relations processed by the adopted sub-strategy from the partial graph of strategies and insert the minimum cost result of the sub-strategy to level 0. Repeat (i) to (iii).

Each graph element contains the same information as before (see algorithm OPT_DV of Figure 3-10) plus a flag "OLD/NEW" which specifies whether the element is retained from the previous execution of steps 2) to 6) shown below.

STEPS

1. Create level zero of the graph of strategies directly from relations in $\{(x, c, l)^q\}$ and set the cost of each element to the cost of its restrictions specified in $\{V^q\}$ and a projection. Mark all elements as "NEW".
2. Do step (3) for $p = 1, 2, \dots, N_g$ to create graph levels $1, 2, \dots, N_g$. Let N_g be the number of necessary operations to process the query specified by the current $\{(x, c, l)^q\}$ and $\{V^q\}$. If $N < N_g$ then $N_g = N$, otherwise $N_g = N$.
3. a) Search level $p-1$ for partitioned relations marked as "NEW". For each "NEW" partitioned relation R_i at level $p-1$, consisting of fragments $R_i^{(k)}$, $k=1, 2, \dots, m$, add m elements to level p . Each new element represents $\bigcup_{k=1}^m R_i^{(k)}$ executed at one of the locations of $R_i^{(k)}$, $k=1, 2, \dots, m$. For each new element R_i representing the union of fragments executed at $x_i^{(k)}$
 - derive the cumulative cost c_i from the cost of $R_i^{(k)}$, $k=1, 2, \dots, m$;
 - set x_j to $x_i^{(k)}$;
 - set $\{r\}_j$ to $\{r\}_i^{(k)}$;
 - I_j is derived from $x_i^{(k)}$, $k=1, 2, \dots, m$.
- b) Insert elements which represent results of join executions, to level p by execution of step (4) for $p_x = p-1, p-2, \dots, \lfloor p/2 \rfloor$.
4. Let level p_y be such that $p = p_x + p_y - 1$, i.e., $p_y = p - p_x + 1$.
 - (i) If $p_x \neq p_y$ then for each graph element R_i of level p_x which is marked as "NEW" element, search level p_y for those elements R_j which can be joined with R_i .
 - (ii) If $p_x = p_y$ then assume that elements of level $p_x = p_y$ are $R_1, R_2, \dots, R_u$. For each element R_i , $i=1, 2, \dots, u-1$, of level p_x , if R_i is marked as a "NEW" element, search elements R_j , $j=i+1, i+2, \dots, u$, of the same level p_x for those relations R_j which can be joined with R_i . Note that one of relations R_i and R_j can be fragmented.

Figure 4-6 The non-redundant heuristic algorithm NR_HEU_DV

NON-REDUNDANT HEURISTIC ALGORITHM NR_HEU_DV FOR QPS FORMULATION

Also note that the relation R_1 can be joined with R_2 only if $|r|_1 \cap |r|_2$ is empty and there exists " $R_3 \in R_1 \cup R_2$ " $\in V^{(1)}(Q)$, such that $R_3 \in |r|_1$ and $R_3 \in |r|_2$. In addition, at most one of R_1 and R_2 can be horizontally partitioned.

Assume that the relation R_j found by i) or ii) above is such that a join of R_1 with R_j is to process a two-variable term $R_3(a \in R_1, b)$.

i) If neither of R_1 or R_j is partitioned then insert a new element R_v to level p and mark it as "NEW". It represents the result of a join of R_1 with R_j in the network node where R_1 is located. Set the triplet x, c, l_v of the new element representing the relation R_v to:

- x_v is set to x_1 ;
- c_v is set to the cumulative cost of R_v ;
- l_v is derived from l_1 and l_j .

In addition, $|r|_v$ is set to the union of $|r|_1$ and $|r|_j$. The pointers of the new node to lower graph levels are set to identify the elements representing R_1 and R_j . If $x_1 \neq x_j$, i.e., R_1 and R_j are not located in the same network node, insert an additional graph element to level p and mark it as "NEW". It represents the join of R_1 with R_j executed in the network node where R_j is located, i.e., in the network node x_j .

ii) If one of R_1 or R_j is partitioned then insert elements, which represent the result of a fragmented join of R_1 with R_j , to level p . Assume that R_1 is partitioned into $R_1^{(k)}$, $k=1,2,\dots,m$. If it is R_j that is partitioned then the roles of R_1 and R_j are exchanged in the remaining portion of this step. There are m new elements inserted to level p and each is marked as "NEW"; each represents a fragment $R_v^{(k)} = (R_1^{(k)} \cup R_j, b)$, $k=1,2,\dots,m$, obtained by a join of $R_1^{(k)}$ with R_j executed at the network location $x_1^{(k)}$. For each new element $R_v^{(k)}$ Set the triplet $x, c, l_v^{(k)}$ of the new element representing the fragment $R_v^{(k)}$, such that:

- $x_v^{(k)}$ is set to $x_1^{(k)}$;
- $c_v^{(k)}$ is set to the cumulative cost of $R_v^{(k)}$;
- $l_v^{(k)}$ is derived from $l_1^{(k)}$ and l_j .

In addition, $|r|_v^{(k)}$ is set to the union of $|r|_1^{(k)}$ and $|r|_j$. The pointers of the new node to lower graph levels are set to identify the elements representing $R_1^{(k)}$ and R_j .

NON-REDUNDANT HEURISTIC ALGORITHM NE_HEU_DV FOR QPS FORMULATION

5. Search level N_k for the minimum cost partial result at level N_k . Horizontal partitions at level N_k are considered to be one partial result whose cost is a combined cost of all of its fragments. Save the sub-strategy for the minimum cost partial result and adopt it as a part of the QPS. Denote the chosen sub-strategy by QPS'. Assume that relations processed by the sub-strategy QPS' are R_j , $j=1,2,\dots,n$ and that the result of QPS' is a relation R_1 . Modify $\{(x,c,I)\}^q$ and $\{V''\}^q$ in the following way:

- (i) Replace information in $\{(x,c,I)\}^q$ for relations processed by the chosen sub-strategy by information about its (sub-strategy's) result.
- (ii) Remove from $\{V''\}^q$ two-variable terms processed by the adopted sub-strategy QPS'.
- (iii) For any two-variable term " $R_j \theta R_k$ " $\in \{V''\}^q$, if $R_j \in \{R_j, j=1,2,\dots,n\}$, i.e., if R_j is one of the relations processed by QPS', then replace R_j with R_1 . In other words, any reference in the query's two-variable terms $\{V''\}^q$ to a relation processed by the adopted sub-strategy is replaced by a reference to R_1 , the sub-strategy's result.

Also modify the partial graph of strategies for query q in the following way:

- (i) remove elements derived from any relations processed by the adopted sub-strategy from the partial graph of strategies;
- (ii) mark all graph element as "OLD"; and
- (iii) insert the minimum cost result of sub-strategy q to level 0 and mark it as "NEW".

6. If $\{V''\}^q$ is not empty, repeat steps (2) to (6); otherwise stop -- the strategy for processing of query q is the synthesis of sub-strategies adopted in executions of step (5).

two-variable terms. The next portion of the strategy is determined in the same way from the modified query.

The value of N can be set for any particular query. It is expected that a larger value for N will, on the average, determine a strategy of a smaller cost at the expense of a higher cost of strategy formulation. The discussion on the input parameter N is postponed to Chapter 6.

CHAPTER 5

MODEL FOR QUERY PROCESSING IN A RELATIONAL DDB

5.1 INTRODUCTION

One of the objectives of this dissertation is to examine how optimal strategies can be used in (i) evaluation of heuristic formulators and (ii) choosing a suitable measure of cost for any given application of a DDB. The instrument used in obtaining these objectives is modelling of query processing for an application of a DDB. This chapter presents the necessary background material; it describes a model of a relational DDB and an application of a DDB of which modelling results are reported in the next chapter.

A relatively simple but detailed model of a relational DDB is developed for modelling distributed queries in order to evaluate the QPS formulators proposed in previous chapters. It therefore has parameters which identify all information necessary for a QPS formulation and its cost estimation. In fact, the model's parameters are very close to those which must be available to the operational QPS formulators, namely:

- (i) parameters on relations and their attributes, for instance, cardinalities, horizontal/vertical partitioning, distribution of relations on information processors;

- (ii) parameters identifying information processors, their processing costs and distribution over network nodes;
- (iii) parameters identifying the network, ie., time delays and dollar transfer costs;
- (iv) parameters necessary for partial result size estimation; and
- (v) cost functions for determining the cost of data transfer and relational operations (restriction, projection, join, fragmented join and a union).

Section 5.2 describes the model parameters used to identify the DDB environment. Functions used to calculate the dollar cost and delay of transferring a relation between any two network nodes are described in the section 5.3. Functions which determine the time delay and dollar cost of execution of individual relational operations (restrict, project, join, union, fragmented join), under the assumption that cardinalities of partial results are known, are presented in the section 5.4.

A query processing strategy has been defined as a specification of a sequence of relational operations and their execution locations. For instance, for a simple query discussed in Chapter 3 (see Figure 3-2),

$$(R_1.a \theta_1 R_2.b) \wedge (R_2.c \theta_2 R_3.d),$$

where R_1 , R_2 and R_3 are located in network nodes W, X, and Y, respectively, the following was given as one of the strategies available for its processing:

- (i) join relations R_1 and R_2 in W to give a relation R_{11} ;

(ii) join relations R_{11} and R_3 in Y to give the query result, relation R_{21} .

This is a rather simple presentation of a query processing strategy which tacitly assumes that additional operations are appropriately included. For the example query these assumed, unspecified operations are now described.

a) Since the join of R_1 with R_2 is executed in W , the relation R_2 must be transferred from X to W .

b) Before R_2 is transferred, it is assumed that its attributes, which are neither target nor "joining", are removed by a projection. Joining attributes for the example query are "a", "b", "c", and "d", i.e., they are attributes specified in the query's two-variable terms. In fact, as specified in assumptions of Chapter 1, every relation referenced by the query is processed by a projection to remove attributes which are not necessary for its processing.

c) Before a join of R_1 with R_2 can commence, necessary fast access paths to R_1 and/or R_2 must be created if they do not exist already.

d) If it is assumed that the "joining" attribute "a" is not a target attribute, then it is removed by a projection from R_1 , the result of a join of R_1 with R_2 . Similarly, if "b" is not a target attribute, it is removed together with the attribute "a".

e) R_{11} is transferred to the network node W .

f A fast access path is built on R_1 or R_2 in preparation for a join of R_1 with R_2 .

g After the join of R_1 with R_2 , non-target attributes are removed from the query result by a projection.

All of the above operations are included in the model for processing of a query. If the cost is measured in terms of the dollar cost due to CPU processing and data transfer then the strategy's cost is derived in the following way.

i The cost of R_1 , R_2 and R_3 is the cost of a projection on each relation to remove attributes which are not necessary for the query processing.

ii The cumulative cost of R_1 is the sum of the costs of
 R_1 and R_2 ;
 transfer of R_2 from X to W;
 building an access path(s) for a join of R_1 with R_2 ;
 join of R_1 with R_2 in W; and
 projection on R_1 to remove unnecessary "joining" attributes.

iii The cumulative cost of R_2 , which is the query result, is the sum of costs of
 R_1 and R_3 ;
 transfer of R_1 from W to Y;
 building an access path(s) for a join of R_1 with R_3 ;
 join of R_1 with R_3 in the network node Y; and

projection on R_2 to remove non-target attributes.

For cost measures other than the combined dollar cost due to CPU processing and data transfer it is necessary to modify the cost calculation described above. For example, for the dollar cost due to CPU processing only, the cost of a data transfer, in the above calculation, is ignored.

The execution of the above strategy includes data transfer and processing of the elementary relations referenced by the query (R_1 , R_2 and R_3) and partial results (R_{12} and R_{21}). Since R_{12} is the result of a join of R_1 with R_2 its cumulative cost is derived from the cumulative costs of R_1 and R_2 together with the cost of any further CPU processing and data transfer which are necessary to produce R_{12} . Similarly R_{21} is the result of a join of R_1 with R_3 ; therefore, its cumulative cost is derived from the cumulative costs of R_1 and R_3 together with any further CPU processing and data transfer costs which are necessary to derive R_{21} . Consequently, functions are necessary to derive the cost of a partial result in terms of various measures. Since a QPS specifies operations (restriction, join, fragmented join, union), their operands and the location of their executions, the cost functions must be such that they also include the cost of data transfer, projections and creation of necessary fast access paths.

Functions used to calculate the cumulative cost for a result of

- (i) initial processing of elementary relations (restrictions and projections);
- (ii) join;
- (iii) fragmented join; and

iv union

are presented in the section 5.5. They will be utilized to determine the cumulative cost of the strategy's partial results and, consequently, the cumulative cost of any strategy.

Since the model is to be used in a selection of a suitable cost measure for a given application of a DDB, it must be sufficiently general to determine the cost of a QPS in terms of various measures. Of particular interest are:

- i) dollar cost due to CPU processing only;
- ii) dollar cost due to data transfer only;
- iii) combined dollar cost due to both CPU processing and data transfer (sum of (i) and (ii) above);
- iv) total size of partial results;
- v) total volume of data transferred over the network;
- vi) total volume of data processed by CPUs;
- vii) total delay due to CPU processing;
- viii) total delay due to data transfer; and
- ix) total delay due to CPU processing and data transfer.

How partial result sizes are determined by the model is described in the section 5.6. Together, sections 5.2 to 5.6 define a model of relational DDB which can be used for any application of a DDB under the assumptions stated in Figure 1-3.

The application of a DDB, chosen for modelling, is described in section 5.7. Not only is the application outlined but the distributed environment parameters are specified and the modelled queries identified. Finally, section 5.8 outlines how algorithms for formulation of a QPS are integrated within the framework of the model, briefly comments on the mechanics of modelling software and software

organization, and provides a sample input/output for a simple query.

Notation which is relevant to subsequent sections is now reviewed.
Recall that

$\text{Card}(R)$... is the cardinality of relation R ;

$\text{Width}(R)$... is the "width" of relation R , i.e., it is the sum of the domain field lengths (in bytes) of the relation's attributes;

$s_i = \text{Card}(R_i) \cdot \text{Width}(R_i)$... is the size (in bytes) of R_i ;

$(x, c, I)_i$... is a triplet containing information about a non-partitioned relation R_i :

x_i ... is the network location of R_i

c_i ... is the cumulative cost of R_i

I_i ... is information about R_i which is necessary for the cost and size estimation;

$(x, c, I)_i^{(k)}$... is a triplet containing information about a horizontal fragment $R_i^{(k)}$:

$x_i^{(k)}$... is the network location of $R_i^{(k)}$

$c_i^{(k)}$... is the cumulative cost of $R_i^{(k)}$

$I_i^{(k)}$... is information about $R_i^{(k)}$ which is necessary for the cost and size estimation;

Z_i ... is the information processor in which R_i is stored; and

$Z_i^{(k)}$... is the information processor in which a fragment $R_i^{(k)}$ is stored.

5.2 MODEL PARAMETERS

To enable modelling of query processing, parameters identifying the DDB environment are necessary. They can be divided into the categories of

- i, parameters about relations (cardinalities, attributes, network locations, ... ;
- ii) parameters about information processors (processing costs, network locations); and
- iii) parameters identifying network costs.

PARAMETERS FOR RELATIONS

Information about relations is contained in two tables. One describes domains for attribute values, while the other contains detailed information about relations.

Domain Specification Table

A relation R can be defined as a set of ordered n -tuples $\langle d_1, d_2, \dots, d_n \rangle$ such that $d_1, d_2, \dots, d_n$ are values for the relation attributes, drawn from sets $D_1, D_2, \dots, D_n$, respectively (Date, 1982). $D_1, D_2, \dots$ are domains, which are not necessarily distinct, for attribute values of all relations in the DB.

For each domain of the DDB, the table specifies its field length in bytes.

Relation Specification Table

For each relation in the DB, the table specifies

- (i) the number of attributes;
- (ii) a list of attributes, and for each attribute the domain from which its values are drawn by pointing to the appropriate entry in the Domain Specification Table;
- (iii) the number of vertical partitions;
- (iv) a list of vertical partitions together with their attribute composition and network locations, i.e., for each vertical fragment there is a list of the attributes it is comprised of and its location;
- (v) the number of horizontal partitions;
- (vi) a list of horizontal partitions, their cardinalities and locations;
- (v) locations of non-fragmented relations; and
- (vi) the relation cardinalities.

INFORMATION PROCESSOR PARAMETERS

For each information processor the parameters specify its network location and the dollar cost and delay for processing a work unit of data. Let

$\alpha\{Z\}$... denote processing of a work unit of data (byte) by information processor Z;

$T[\alpha\{Z\}]$... denote the delay to process one unit of data (1 byte) by information processor Z; and

$D[\alpha\{Z\}]$... denote the dollar cost to process one unit of data (1 byte) by information processor Z.

NETWORK PARAMETERS

The parameters specify the dollar cost and delay for transferring a unit of data between any two network locations. Let

$\beta\{X,Y\}$... denote the transfer of a unit of data (1 byte) from the network node X to Y;

$T[\beta\{X,Y\}]$... denote the delay to transfer one unit of data (1 byte) from the network node X to Y; and

$D[\beta\{X,Y\}]$... denote the dollar cost to transfer one unit of data (1 byte) from X to Y.

5.3 TIME DELAY AND DOLLAR COST OF TRANSFERRING RELATIONS

The dollar cost and delay to transfer a relation R_i between any two network nodes is assumed to be directly proportional to its size. Let

$\phi\{(X,Y),R_i\}$... denote the transfer of a relation R_i from the X to Y.

Recall that $D[\beta\{X,Y\}]$ and $T[\beta\{X,Y\}]$ are the respective dollar cost and delay to transfer a unit of data from X to Y. The dollar cost and time delay to transfer a relation R_i from X to Y are

$$D[\phi\{(X,Y),R_i\}] = D[\beta\{X,Y\}] \cdot s_i \quad \text{and}$$

$$T[\phi\{(X,Y),R_i\}] = T[\beta\{X,Y\}] \cdot s_i,$$

respectively, where s_i is the total size of R_i .

5.4 COST OF RELATIONAL OPERATIONS

The dollar cost and delay of relational operations (restrict, project, join, union) are specified in this section. Let

$\delta\{(R_i, R_r), Z\}$... denote restriction(s) on R_i giving R_r , executed by the information processor Z , where all restrictions as specified by the query one-variable terms are cascaded and hence inferred by this notation;

$\pi\{(R_i, R_r), Z\}$... denote a projection on R_i giving R_r , executed by the information processor Z ;

$\omega\{(R_i, R_r), Z\}$... denote initial processing (all specified restrictions and a projection to remove attributes of R_i which are neither target nor joining attributes), executed by the information processor Z ;

$\ast\{(R_i, R_j, R_r), Z\}$... denote a join of R_i with R_j giving R_r , executed by the information processor Z ; and

$U\{(R_i^{(k)}, R_r), Z\}$... denote a union, of all horizontal fragments of R_i giving a non-partitioned relation R_r , where R_i is horizontally partitioned into $R_i^{(k)}$, $k=1, 2, \dots, m$, executed by the information processor Z .

The equations for determining the dollar cost and time delay of each of the above operations are developed in this section. Let

Φ ... denote one of the relational operations listed above. Then

$D[\Phi]$... denotes the dollar cost of operation Φ and

$T[\Phi]$... denotes the delay of operation Φ .

The equations for the dollar cost, $D[\theta]$, and delay, $T[\theta]$, are developed for each operation θ . Their summary can be found in Table 5-1.

RESTRICTION

All restrictions specified by the query on any relation are assumed to be processed before processing of two-variable terms begins. The cost of a restriction, or cascaded restrictions, on a relation depends on the availability of the relation's fast access paths. For example, if a fast access path exists on attribute "a" for relation R, then a restriction specified on that attribute, such as "R.a = 1", may be processed for the cost of access to only those tuples which satisfy the restriction. If a fast access path on that particular attribute does not exist, however, then there is no advantage to building one: Building an access path requires access to all tuples of the relation and one pass through that relation also determines all tuples satisfying the restriction. Modelling assumes that all restrictions on one relation require one pass through that relation.

Recall that

i. $\pi[D]$ denotes processing of a unit byte of data by the information processor D;

ii. $T[\pi[D]]$ and $D[\pi[D]]$ are the respective delay and dollar cost to process a unit of data and

iii. s_1 and s_2 are the respective sizes of r_1 and r_2 .

The dollar cost and delay of a restriction on R_1 , resulting in R_2 and executed by the information processor D, are defined as

$$D[\theta(R_1, R_2, D)] = T[\pi[D]] \cdot s_1 - s_2 \quad \text{and}$$

TABLE 1. DOLLAR COST AND DELAY FOR RELATIONAL OPERATIONS AND DATA TRANSFER

OPERATION ρ	DOLLAR COST $D(\rho)$ AND TIME DELAY $T(\rho)$
RESTRICTION ρ $\rho = R_1 \wedge R_2 \wedge \dots \wedge R_n$	$D(\rho) = D(R_1) + D(R_2) + \dots + D(R_n)$ $T(\rho) = T(R_1) + T(R_2) + \dots + T(R_n)$
PROJECTION ρ $\rho = \pi_{A_1, A_2, \dots, A_n}(R)$	$D(\rho) = D(R)$ $T(\rho) = T(R) + \sum_{i=1}^n \text{width}(A_i) \cdot \log_2 \text{card}(R)$
RELATIONAL PROCESSING ρ $\rho = \sigma_{A_1, A_2, \dots, A_n}(R)$	$D(\rho) = D(R)$ $T(\rho) = T(R) + \sum_{i=1}^n \text{width}(A_i) \cdot \log_2 \text{card}(R)$
JOIN ρ $\rho = R_1 \Join R_2$	$D(\rho) = D(R_1) + D(R_2)$ $T(\rho) = T(R_1) + T(R_2) + \text{width}(R_1) \cdot \log_2 \text{card}(R_2)$
DATA TRANSFER ρ $\rho = \tau(R)$	$D(\rho) = D(R)$ $T(\rho) = T(R) + \text{width}(R) \cdot \log_2 \text{card}(R)$

- $D(\rho)$ is the dollar cost of operation ρ . $T(\rho)$ is the time delay of operation ρ . $\text{width}(R)$ is the width of relation R . $\text{card}(R)$ is the cardinality of relation R . $\log_2$ is the base-2 logarithm. $\sum$ is the summation operator. $\Join$ is the join operator. τ is the transfer operator. σ is the selection operator. π is the projection operator. $\wedge$ is the logical AND operator. ρ is a constant for an algorithm to select a set access set.
- $D(\rho)$ denotes the dollar cost of operation ρ .
- $T(\rho)$ denotes the time delay of operation ρ .
- ρ denotes the following relational or data transfer operations:
- $\rho = R_1 \wedge R_2 \wedge \dots \wedge R_n$ denotes a restriction ρ of relation R by predicates $R_1, R_2, \dots, R_n$. All restrictions are stored in the left-hand side terms and are listed in the order of their restriction.
 - $\rho = \pi_{A_1, A_2, \dots, A_n}(R)$ denotes a projection ρ of relation R by attributes $A_1, A_2, \dots, A_n$. All projections are stored in the left-hand side terms and are listed in the order of their projection.
 - $\rho = \sigma_{A_1, A_2, \dots, A_n}(R)$ denotes a selection ρ of relation R by predicates $A_1, A_2, \dots, A_n$. All selections are stored in the left-hand side terms and are listed in the order of their selection.
 - $\rho = R_1 \Join R_2$ denotes a join ρ of relations R_1 and R_2 . All joins are stored in the left-hand side terms and are listed in the order of their join.
 - $\rho = \tau(R)$ denotes a transfer ρ of relation R . All transfers are stored in the left-hand side terms and are listed in the order of their transfer.
- ρ denotes a non-partitioned relation ρ executed by the information processor. ρ denotes a horizontally partitioned relation ρ executed by the information processor. ρ denotes the transfer of relation ρ for network use.

$$T[\delta(R_1, R_r), Z] = T[\alpha(Z)] \cdot (s_1 + s_r),$$

respectively.

Both relations R_1 and R_r have identical attributes. The cost of a restriction includes retrieval and a search through R_1 to find tuples satisfying the given predicate(s). This cost is directly proportional to the size of R_1 . In addition, tuples of the result must be stored -- a process incurring a cost which is directly proportional to the size of R_r , the restriction's result. As can be deduced from the above equations, modelling of a restriction ignores possible existence of fast access paths.

PROJECTION

A generally accepted rule for processing of queries (see Chapter 1) is that attributes which are not necessary in any subsequent processing of the query should be removed by a projection. Before processing of any two-variable terms begins all attributes which are not joining or target attributes should be removed from relations to decrease their size, and thus decrease the cost of subsequent processing and data transfer. In addition, if a joining attribute is not used in further processing and is not a target attribute, it should be removed from the join result. To remove attributes by a projection, one pass through the relation is sufficient; however, removal of an attribute(s) may cause duplicate tuples which must be removed. Removal of duplicate tuples is achieved by first sorting the relation or building an access path on it and removing duplicate tuples on one pass through the relation. Consequently, the cost of a projection depends not only on the size of the original relation but also on its size without the removed attributes.

If a relation R_i , stored by the information processor Z is processed by a projection which results in R_r , the dollar cost and delay of a projection are defined to be:

$$D[\pi((R_i, R_r), Z)] = D[\alpha\{Z\}] \cdot s_i + \\ D[\alpha\{Z\}] \cdot \text{Width}(R_r) \cdot K_s \cdot \log_2(\text{Card}(R_r)) + \\ D[\alpha\{Z\}] \cdot s_r;$$

$$T[\pi((R_i, R_r), Z)] = T[\alpha\{Z\}] \cdot s_i + \\ T[\alpha\{Z\}] \cdot \text{Width}(R_r) \cdot K_s \cdot \log_2(\text{Card}(R_r)) + \\ T[\alpha\{Z\}] \cdot s_r;$$

The first term of the sum in the above equations is the cost of retrieval of R_i and removal of unnecessary attributes. The second term represents the cost of building an access path which is used in the removal of possible duplicate tuples. It is directly proportional to the width of a tuple in R_r times the complexity of the algorithm to build the access path, which is assumed to be " $K_s \cdot \log_2(\text{Card}(R_r))$ ". K_s is a constant for the particular algorithm used to build the access path. Finally, the third term represents the cost of removal of duplicate tuples and storing the result.

DOLLAR COST AND DELAY DUE TO INITIAL.

PROCESSING ON A RELATION (restriction(s) AND projection)

Since restrictions specified on a relation are processed first, followed by a projection to remove unnecessary attributes, the dollar cost and delay for this initial processing is now derived. If a restriction(s) applied to R_i results in the relation R_j and then a projection on R_j results in R_r , the dollar cost and delay are defined

to be:

$$D[U(R_1, R_2), Z] = D[\sigma(R_1, R_2), Z] + D[\pi(R_1, R_2), Z]$$

$$T[U(R_1, R_2), Z] = T[\sigma(R_1, R_2), Z] + T[\pi(R_1, R_2), Z]$$

Note that if a one-variable term is not specified on the relation R_1 then the cost of a restriction on R_1 is defined to be zero. Similarly, if all attributes of R_1 are necessary for the QPS execution then a projection is unnecessary and its cost is defined to be zero. Recall that if more than one restriction is specified on a relation, they are cascaded and executed for the cost of one restriction.

UNION

In this model, a union is executed to consolidate a horizontally fragmented relation such as R_1 consisting of fragments $R_1^{(k)}$, $k=1, 2, \dots, m$. The consolidation is assumed to be executed in the following way. First, a temporary, empty relation R_{temp} is created with attributes identical to those of $R_1^{(k)}$ and each fragment is "appended" to R_{temp} . In other words, as each fragment is received from the network, it is directly appended to R_{temp} . Finally, possible duplicate tuples are removed utilizing a projection which results in the relation R_1 . Consequently, the dollar cost and delay of a union of fragments are defined to be:

$$D[U(R_1^{(k)}, R_1), Z] = D[\pi(R_1, R_1), Z]$$

$$T[U(R_1^{(k)}, R_1), Z] = T[\pi(R_1, R_1), Z]$$

Since each fragment is appended directly to the result, as it is received from the network the dollar cost and delay to execute the union of a fragmented relation is directly proportional to the dollar cost and delay, respectively, of the projection to remove possible duplicate tuples.

JOIN

A join is assumed to be processed in the following way. If relations R_1 and R_2 are to be joined, first a fast access path is built on the joining attribute of the operand having the smaller cardinality. Assume that, from operands R_1 and R_2 , it is the relation R_1 which has the smaller cardinality. Once the access path for R_1 exists, the processor steps through the relation R_1 . For each tuple in R_1 , utilizing the fast access path the processor finds all tuples in R_2 which satisfy the two-variable term, such as $R_1.a = R_2.b$, being processed. Tuples in R_2 are accessed exactly once, while tuples in R_1 more than once, depending on the cardinality of the result. At the end of the join the cost of a projection to remove unnecessary joining attributes is added to the cost of the join result.

Note that this model of a join execution includes the process of building an access path. Normally a fast access path must be created. Elementary relations may already have access paths, however, relations are first processed by restrictions to process one-variable terms and by projections to remove unnecessary attributes. Consequently, access paths, which are necessary for further processing of the query, must be created after a restriction and/or projection. This model differs from query processing in a centralized DB in the stage when access paths are created.

Smith [1978] proposes that in a centralized DB a fast access path should not be built in the beginning of a join execution but in the last stages of operations which produce the join operands. For instance, consider a join of R_1 with R_2 such that R_1 is a result of a projection. If it is assumed that a fast access path on R_1 is needed for join execution then it should be created in the "output" stage of the projection on R_1 and not as a first stage of the join. This results in a decrease of access to the secondary storage devices.

In a distributed environment, however, an access path should be created in the first stage of a join if relation R_1 is not located where the join is executed. Assume that R_1 is located in X while the join execution occurs in Y . If the fast access path on R_1 is built at the "output" stage of the projection on R_1 in node X , then the transfer of R_1 from X to Y must also include the fast access path. This increases the cost to transfer R_1 , which may outweigh the savings gained by building the access path in the last stage of the projection. Moreover, it is not likely that a fast access path on R_1 built in X can be utilized in node Y . Even if relational DBMSs in different network nodes are homogeneous, it may be that stored access methods [Date, 1982], or their parameters, are different in various network locations. Consequently, the fast access path on R_1 must be built in node Y .

Finally, the size estimation which includes the fast access path on R_1 is difficult and, therefore, the cost to transfer R_1 with its access path is difficult to estimate as well. As a consequence of the above arguments, the process of building an access path is integrated with the join execution in this model.

Assume that the join of R_1 with R_2 results in a relation R_r and that $\text{Card}(R_1) < \text{Card}(R_2)$. The dollar cost of a join of R_1 with R_2 resulting in R_r is defined to be

$$\begin{aligned}
D^*(R_1, R_2, R_P, Z) &= D[\alpha(Z)] \cdot s_1 + \\
&D[\alpha(Z)] \cdot \text{Card}(R_1) \cdot K_g \cdot \text{Width}(R_1) \cdot \log_2 \text{Card}(R_1) + \\
&D[\alpha(Z)] \cdot s_3 + D[\alpha(Z)] \cdot \text{Card}(R_P) \cdot \text{Width}(R_P) + D[\alpha(Z)] \cdot s_5 + \\
&D[\pi(R_1, R_P, Z)]
\end{aligned}$$

The first term of the sum in the above equation is the cost of retrieval of R_1 for the purpose of building an access path. The cost of building the access path is determined by the second term of the sum; it is derived in the same way as the cost of a projection.

The third, fourth and fifth terms of the sum for the dollar cost, represent the cost of determining a set of tuples of R_1 and R_P which satisfy the predicate of the two-variable term being processed such as $R_1.a = R_P.b$.

The third term, " $D[\alpha(Z)] \cdot s_3$ ", is the retrieval cost for tuples of R_P . For each tuple of R_P , the corresponding tuples of R_1 which satisfy the predicate must be found. The cost of this process is determined by the fourth term, " $D[\alpha(Z)] \cdot \text{Card}(R_P) \cdot \text{Width}(R_1)$ " in which the number of retrieved tuples of R_1 depend directly on the cardinality of the join result, $\text{Card}(R_P)$. The fifth term, " $D[\alpha(Z)] \cdot s_5$ ", represents the cost of storing the result. Finally, the last term approximates the cost of a projection to remove possible duplicate tuples from the result.

Assume that the join of R_1 with R_P results in R_P and that $\text{Card}(R_1) < \text{Card}(R_P)$. The delay of a join of R_1 with R_P resulting in R_P is defined below. Its derivation is similar to that of the join's dollar cost.

$$\begin{aligned}
 T^*(R_1, R_2, R_3, D) &= T^*(R_1, D) \cdot s_1 + \\
 &T^*(R_2, D) \cdot \text{Card } R_1 \cdot s_2 + \text{Width } R_1 \cdot \log_2 \text{Card } R_1 + \\
 &T^*(R_3, D) \cdot s_1 + T^*(R_1, D) \cdot \text{Card } R_2 \cdot \text{Width } R_2 + T^*(R_2, D) \cdot s_2 + \\
 &T^*(R_1, R_2, D)
 \end{aligned}$$

3.5 CUMULATIVE COST OF PARTIAL RESULTS

The cumulative cost of a relation R_i or a horizontal partition R_i^K has been denoted thus far by c_i and c_i^K , respectively. In this section, functions to determine the cumulative cost for partial results of various operations, in a number of cost measures, are presented. Specifically, given operands and their cumulative costs, functions to determine the cumulative cost of a result of the above operations are presented. Whereas the previous section presented cost determination of individual operations, for example a join, in this section the cumulative cost is determined for operations' results, for instance a join, which includes not only the cost of a join execution itself, but also the cumulative cost of the data transfer and any operations executed to derive the join operands. In other words, the cumulative cost of a result includes the total cost of all processing and data transfer performed in order to derive it.

The functions are such that information consisting of parameters about environment, operands and their cumulative costs is deemed to be sufficient to determine the cumulative cost of the operation's result. This is a reasonable assumption for a single query environment in which a query is not delayed due to execution of concurrent transactions. For instance, assume that relations R_1 and R_2 are located in network nodes X and Y, respectively. If it is assumed that the total dollar costs for R_1 and R_2 are known, the cumulative cost of R_{11} , which is the

result of a join of R_i with R_j executed in the network node X , can be accurately calculated by:

cumulative cost of R_i = cumulative cost of R_i +
 cumulative cost of R_j +
 cost of transfer of R_i from X to Y +
 cost of building of an access path +
 cost of a join execution +
 cost of a projection to remove unnecessary
 attributes

Nine measures were selected for the cumulative cost time size of a partial result. Let the cumulative cost of a relation R , where R could be a fragment of a horizontally partitioned relation, in a specifying cost measure be denoted by

$C_P[R]$... dollar cost due to CPU processing only;

$C_X[R]$... dollar cost due to data transfer only;

$C_A[R]$... dollar cost due to CPU processing and data transfer;

$T_P[R]$... delay due to CPU processing only;

$T_X[R]$... delay due to data transfer only;

$T_A[R]$... delay due to CPU processing and data transfer;

$S_P[R]$... volume of data processed by CPUs;

$S_X[R]$... volume of data transferred; and

$S_A[R]$... sum of the size of partial results.

Functions for the nine measures listed above are to be developed for each of the partial results of
 initial processing of an elementary relation;

- a join;
- a fragmented join; and
- a union of a horizontally partitioned relation.

Consequently, there is a total of 36 functions to be presented. As the presentation of all 36 functions is repetitious, it is included in Appendix B. This section develops equations only for the total combined dollar cost due to CPU processing and data transfer. The summary of all functions can be found in Table B-2.

INITIAL PROCESSING

Every elementary relation referenced by the query is first processed by restrictions and a projection to remove possible duplicate tuples and attributes which are not necessary for processing of two-variable terms. Consider an elementary relation R_i (which could be a fragment), located in the network node X with information processor Z and whose initial processing results in the relation R_r . Recall that $\omega\{(R_i, R_r), Z\}$ represents initial processing (restrictions) and a projection on R_i resulting in R_r . The cumulative dollar cost due to CPU processing and data transfer for R_r is defined as follows:

$$D_p[R_r] = D[\omega\{(R_i, R_r), Z\}]$$

As initial processing is not preceded by any other operation, its cumulative cost is simply the dollar cost of initial processing, $D[\omega\{(R_i, R_r), Z\}]$.

TABLE 5-2 CUMULATIVE COST FUNCTIONS

OPERATION ϕ PRODUCING R_i	FUNCTIONS FOR $C_p(R_i), C_x(R_i), C_a(R_i), C_s(R_i), S_p(R_i), S_x(R_i), S_a(R_i), S_s(R_i)$
Initial processing R_i, R_i, R_i	$C_p(R_i) = C(-1, R_i, R_i, 2)$ $C_x(R_i) = 0$ $C_a(R_i) = C_p(R_i) + C(-1, R_i, R_i, 2)$ $C_s(R_i) = C(-1, R_i, R_i, 2)$ $C_x(R_i) = 0$ $C_a(R_i) = C_p(R_i) + C(-1, R_i, R_i, 2)$ $S_p(R_i) = C_p(R_i)^2$ $S_x(R_i) = 0$ $S_a(R_i) = S_p(R_i) + S_p(R_i) + S_p(R_i)$
Join R_i, R_i, R_i	$C_p(R_i) = C_p(R_i) + C_p(R_i) + C(-1, R_i, R_i, 2)$ $C_x(R_i) = C_p(R_i) + C_p(R_i) + C(-1, Y, X, 2)$ $C_a(R_i) = C_p(R_i) + C_p(R_i) + C(-1, R_i, Y, X) + C(-1, R_i, R_i, 2)$ $C_s(R_i) = \max \{ C_p(R_i), C_p(R_i) + C(-1, R_i, R_i, 2) \}$ $C_x(R_i) = \max \{ C_p(R_i), C_p(R_i) + C(-1, Y, X, 2) \}$ $C_a(R_i) = \max \{ C_p(R_i), C_p(R_i) + C(-1, Y, X, 2) + C(-1, R_i, R_i, 2) \}$ $S_p(R_i) = C_p(R_i)^2$ $S_x(R_i) = C_p(R_i)^2$ $S_a(R_i) = S_p(R_i) + C_p(R_i)^2$
Fragmented join R_i, R_i, R_i	$C_p(R_i) = C_p(R_i) + C_p(R_i) + C(-1, R_i, R_i, 2)$ $C_x(R_i) = C_p(R_i) + C_p(R_i) + C(-1, Y, X, 2)$ $C_a(R_i) = C_p(R_i) + C_p(R_i) + C(-1, R_i, Y, X) + C(-1, R_i, R_i, 2)$ $C_s(R_i) = \max \{ C_p(R_i), C_p(R_i) + C(-1, R_i, R_i, 2) \}$ $C_x(R_i) = \max \{ C_p(R_i), C_p(R_i) + C(-1, Y, X, 2) \}$ $C_a(R_i) = \max \{ C_p(R_i), C_p(R_i) + C(-1, Y, X, 2) + C(-1, R_i, R_i, 2) \}$ $S_p(R_i) = C_p(R_i)^2$ $S_x(R_i) = C_p(R_i)^2$ $S_a(R_i) = S_p(R_i) + C_p(R_i)^2$
Join R_i, R_i, R_i	$C_p(R_i) = \sum_{k=1}^n C_p(R_i^{(k)}) + C(-1, R_i, R_i, 2)$ $C_x(R_i) = \sum_{k=1}^n C_p(R_i^{(k)}) + \sum_{k=1}^n C(-1, R_i^{(k)}, X, 2)$ $C_a(R_i) = \sum_{k=1}^n C_p(R_i^{(k)}) + \sum_{k=1}^n C(-1, R_i^{(k)}, Y, X) + C(-1, R_i, R_i, 2)$ $C_s(R_i) = \max \{ C_p(R_i^{(k)}), C_p(R_i^{(k)} + C(-1, R_i, R_i, 2) \}$ $C_x(R_i) = \max \{ C_p(R_i^{(k)}), C_p(R_i^{(k)} + C(-1, R_i^{(k)}, X, 2) \}$ $C_a(R_i) = \max \{ C_p(R_i^{(k)}), C_p(R_i^{(k)} + C(-1, R_i^{(k)}, Y, X) + C(-1, R_i, R_i, 2) \}$ $S_p(R_i) = C_p(R_i)^2$ $S_x(R_i) = C_p(R_i)^2$ $S_a(R_i) = \sum_{k=1}^n S_p(R_i^{(k)}) + S_p(R_i)$

TABLE 5-2. CONTINUED

* The total volume of data processed by CPUs is derived in the same way as the total dollar cost due to CPU processing evaluated with $D_{\text{CPU}}[i, j]$ and $T_{\text{CPU}}[i, j]$ for any relational operation executed by any information processors i .

** The total volume of data transferred over the network is derived in the same way as the total dollar cost due to a data transfer evaluated with $D_{\text{NET}}[i, j]$ and $T_{\text{NET}}[i, j]$ for all i and j .

*** $T_{\text{NET}}[i, j]$ is the transmission time, i.e., time when all fragments have been received. If fragments of R are horizontally fragmented into R^k , $k=1, \dots, m$.

NOTATION

$C_{\text{CPU}}[i]$... dollar cost to process a unit of data (byte) by processor i .

$T_{\text{CPU}}[i]$... time delay to process a unit of data (byte) by processor i .

$C_{\text{NET}}[i, j]$... time delay to transfer a unit of data (byte) from i to j .

$D_{\text{NET}}[i, j]$... dollar cost to transfer a unit of data (byte) from i to j .

$\sigma_{\text{CPU}}[i, R]$... denotes a projection on R , giving R_1 , executed by the processor i .

$\rho_{\text{CPU}}[i, R]$... denotes initial processing and specified restrictions and a projection to remove attributes of R , which are neither target nor joining attributes, executed by the information processor i .

$\sigma_{\text{NET}}[i, j, R]$... denotes a projection on R , giving R_1 , executed by the information processor i , storing R_1 connected to the network node j , such that it is assumed that R_1 is stored at j .

$\rho_{\text{NET}}[i, j, R]$... denotes a subset of R , giving a non-partitioned relation R_1 , where R is horizontally partitioned into R^k , $k=1, \dots, m$, and

$\tau_{\text{NET}}[i, j, R]$... denotes the transfer of a relation R_1 from the network node j to i .

$C_{\text{CPU}}[i, j]$... cumulative dollar cost due to CPU processing only;

$C_{\text{NET}}[i, j]$... cumulative dollar cost due to data transfer only;

$C_{\text{CPU,NET}}[i, j]$... cumulative dollar cost due to CPU processing and data transfer;

$T_{\text{CPU}}[i, j]$... cumulative time delay due to CPU processing only;

$T_{\text{NET}}[i, j]$... cumulative time delay due to data transfer only;

$T_{\text{CPU,NET}}[i, j]$... cumulative time delay due to CPU processing and data transfer;

$S_{\text{CPU}}[i, j]$... cumulative volume of data processed by CPUs;

$S_{\text{NET}}[i, j]$... cumulative volume of data transferred; and

$S_{\text{CPU,NET}}[i, j]$... cumulative sum of the size of partial results.

RESULT OF A JOIN

Consider relations R_i and R_j , located in U and V respectively and with their respective cumulative costs c_i and c_j expressed in an appropriate measure of cost. A join of R_i with R_j is executed in one of the network nodes X or Y and, therefore, if $U \neq V$ then one of R_i and R_j must be transferred. The cumulative cost c_r of the join resulting relation R_r , is derived from the cumulative costs of the join operands c_i and c_j , and the cost for the data transfer and a join execution. If both relations are located in the same network node $U = V$, the cost of the data transfer in the following equations is defined to be zero. It is assumed for the following equations that the join is executed in the network node $x_r = X$ with information processor I_r ; therefore, if $U \neq X$ that R_i is transferred from U to X . If the join is executed in Y then R_j is replaced by R_i in any cost pertaining to the data transfer.

Recall that

- * $\{R_i, R_j, R_r, Z\}$... denotes the join of R_i with R_j , resulting in R_r and
- $\phi\{R_j, (Y, X)\}$... denotes the transfer of R_j from Y to X .

The cumulative dollar cost due to both CPU processing and data transfer is defined for R_r as:

$$D_a[R_r] = D_a[R_i] + D_a[R_j] + D[\phi\{R_j, (Y, X)\}] + D[*\{R_i, R_j, R_r, Z\}]$$

$D_a[R_r]$ is the sum of the cumulative costs of R_i and R_j , the dollar cost to transfer R_j from Y to X and the dollar cost to execute the join of R_i with R_j .

Note that the cost of creation of last access paths is included within the cost of a join execution. See section 4.4.

FRAGMENTED JOIN

Consider relations R and S such that R is located in the network node x_1 and S is horizontally partitioned into fragments $S^{(k)}$, $k=1,2,\dots,m$, located in their respective network nodes $x_i^{(k)}$, $k=1,2,\dots,m$, each with an information processor $Z_i^{(k)}$. A fragmented join of R with S is executed in two stages. First, the non-partitioned operand, relation R , is transmitted to the network nodes $x_i^{(k)}$, $k=1,2,\dots,m$, i.e., it is transmitted to those nodes in which the partitions of S are located. In a point-to-point network, this is accomplished in the following way. The information processor Z_1 in which R is stored, submits R to the network node to which it is connected, Y , for a broadcast to the nodes $x_i^{(k)}$, $k=1,2,\dots,m$. The network delivers the relation R to the destination node $x_i^{(k)}$, $k=1,2,\dots,m$, with a dollar cost of $C(R, Y, x_i^{(k)})$ and a time delay of $T(R, Y, x_i^{(k)})$.

In the second stage, each partition $R_i^{(k)}$ is joined with $S^{(k)}$ by the information processor $Z_i^{(k)}$ which is connected to the network node $x_i^{(k)}$. The fragmented join result is the relation R_r , horizontally partitioned into $R_r^{(k)}$, $k=1,2,\dots,m$, such that the location of $R_r^{(k)}$ is $x_r^{(k)} = x_i^{(k)}$.

Upon examination of the fragmented join execution described above it becomes evident that the cumulative cost for a partition $R_r^{(k)}$ is derived in exactly the same way as that of a join of two non-partitioned relations.

$$R_1^k = R_1^k \cup R_2^k \cup \dots \cup R_m^k$$

$$R_1^k = R_1^k \cup R_2^k \cup \dots \cup R_m^k$$

UNION OF A HORIZONTALLY PARTITIONED RELATION

Consider a relation R_1 horizontally partitioned into m fragments $R_1^1, \dots, R_1^m$, such that each partition is located in the network node x^k . A union of partitions executed in a network node x^k with an information processor Z is executed as described in the section 5.4. To review the process, first, an empty relation R_2 is created and then as fragments of R_1 are received from the network, they are directly "appended" to R_2 . Finally, when all fragments of R_1 are consolidated in this way, a projection is executed to remove possible duplicate tuples.

$$D[R_2] = \sum_{k=1}^m D[R_1^k] + \sum_{k=1}^m D[U\{R_1^k, x^k, Z\}]$$

$$D[U\{R_1^k, R_2, Z\}]$$

The total dollar cost is the sum of the fragments' cumulative costs, plus the cost to transfer partitions to the node x^k and the cost of the union, $D[U\{R_1^k, R_2, Z\}]$.

As previously stated, the cumulative cost functions are developed in Appendix D. Only one note is added here, and that is on the cost measures S_p , S_x and S_a .

The dollar cost due to CPU processing by a processor Z is directly proportional to the volume of processed data. Consequently, if it is assumed that $D[\alpha\{Z\}] = \$1.00/\text{byte}$ for all information processors Z , i.e., the cost to process a unit (byte) of data by any information processor Z is one dollar, the total volume of data processed by CPUs, $S_p[R]$, is

derived in the same way as the total dollar cost due to CPU processing,

$$S_P[R] = C_P[R] \cdot \text{byte } \$$$

Similarly, the cumulative cost of data transferred from a network node X to Y is directly proportional to the volume of transferred data. If it is assumed that the total dollar cost due to transferring any relation R is equal to the size of R , i.e., the unit cost of a data transfer between any two network nodes X and Y , $X \neq Y$, is one $C_B[X,Y] = \$1.00/\text{byte}$, then the cumulative volume of data transferred over the network in deriving the relation R is

$$S_X[R] = C_X[R] \cdot \text{byte } \$$$

Consequently, the total volume of data transferred over the network, $S_X[R]$, is derived in the same way as the total dollar cost due to a data transfer, $C_X[R]$, by using $C_B[X,Y] = \$1.00/\text{byte}$, for all network nodes X and Y , $X \neq Y$.

The total size of partial results for a relation R is the sum of the size of any operands used in deriving R and the size of R .

5.6 SIZE OF PARTIAL RESULTS

It is not the objective of this dissertation to examine size estimation of partial results. The assumption is made that a mechanism for size estimation of partial results would exist in a DB system. However, the model of a DDB must determine the size of partial results of restrictions, projections, joins and unions; given that enough information about the operand(s) of these operations is available. Specifically, the model must determine

- (i) the proportion of tuples removed by a projection;
- (ii) the proportion of tuples which satisfy a restriction;
- (iii) the cardinality of a join result; and
- (iv) the cardinality of a union.

How the model determines the size of partial results for each of the above cases is described in this section.

RESTRICTION

A restriction never increases, and usually decreases, the cardinality of its operand. The proportion of tuples which satisfy a restriction is specified directly by the query specification. A query q specification includes a set of one-variable (restrictions) terms, $\{V'\}^q$.

For each one-variable term V'_j there is a constant which specifies the proportion of the relation's tuples which satisfy it. If a restriction, specified by the one-variable term V'_j , results in a relation R_r , the cardinality of R_r is determined by

$$\text{Card}(R_r) = K_j^1 \cdot \text{Card}(R_i), \quad 0 < K_j^1 \leq 1$$

$K_j^1 \dots$ is a constant specifying the proportion of tuples which satisfy the restriction V'_j , where the super-script "1" specifies that the constant is for a one-variable term. With which one-variable term is the constant to be used is identified by the sub-script.

3

PROJECTION

Removal of an attribute(s) by a projection may cause duplicate tuples which must be removed. The cardinality of the result is determined by additional parameters stored in the "Relation Specification Table" described in the section 5.2. For each attribute of every relation, there is an additional parameter identifying the proportion of tuples which are removed from that relation if the attribute is removed by a projection.

UNION

The cardinality of a union is given by the sum of the cardinalities of its operands.

JOIN

A modelled query definition, which is discussed in the section 5.8, includes the query's two-variable terms. For each of the two-variable terms, such as $V''_k = "R_i.a \theta R_j.b"$, the query's definition input to the modelling software also includes a constant " K_k^2 " which is used for size estimation. If the two-variable term V''_k is processed by a join of R_i with R_j giving a relation R_r , the cardinality of the result is determined by

$$\text{Card}(R_r) = K_k^2 \cdot \text{Card}(R_i) \cdot \text{Card}(R_j), \quad 0 < K_k^2 \leq 1,$$

where the super-script of "K" specifies that the constant is for a two-variable term while with which two-variable term it is to be used is specified by the sub-script. The constant K_k^2 is equivalent to the selectivity factor ((Epstein, 1980), (Ibaraki, 1984)) of the join

predicate $V''_k = "R_i.a \Theta R_j.b"$. It specifies the expected fraction of tuple pairs from R_i and R_j which satisfy V''_k .

While sections 5.2 to 5.6 defined a model of a relational DDB which can be used for any application of a DDB, section 5.7 describes the application which was chosen for modelling. It defines the parameters identifying the application's distributed environment and lists modelled queries.

5.7 MODELLED APPLICATION OF A DDB

Firm MONAX designs and builds chemical process plants. It has four offices, two main offices and two field headquarters. One main office, called MAIN-WEST, is located in Calgary, while the other, called MAIN-CENTRAL, is located in Toronto. The two main offices are autonomous in the sense that they have their own administration, management and associated mobile field teams. They also work on their own projects which are, primarily, located in their respective regions.

Each main office has a field headquarters associated with it. Field headquarters called HEADQ-NORTH, located in Edmonton, is under the management of the office MAIN-WEST, while the headquarters HEADQ-EAST, located in Halifax, is under the management of the office MAIN-CENTRAL.

The company has recently acquired a subsidiary, called MONREAL, which has one office located in Calgary, several mobile field teams, but no field headquarters.

Cooperation exists between offices MAIN-WEST and MAIN-CENTRAL but primarily at the managerial and design levels. Cooperation between their field teams is limited due to the prohibitive costs arising from

their geographical separation. Cooperation between companies MONAX and MONREAL, located in Calgary, consists of sharing the field headquarters, manpower and, occasionally, material.

The companies use a relational DDB to support their management information system. Each main office has its own local DB containing information about office related matters, i.e., employees, payroll and design specifications for jobs. The field headquarters also have their local DBs containing information about equipment and material. The shortage of material due to the design or delivery problems is of a great concern as a one day delay on a large job may cost up to a half million dollars.

The local DBs are autonomous but it is necessary to support queries which view the collection of local DBs as a unified, integrated DDB.

The model parameters of the DDB application are included in Appendix E. Relations, attributes and domains are displayed in the section E-1 of the appendix. Elements of a domain are assumed to have equal size; this is shown in bytes.

Each of the main offices has an information processor. Section E-2 of the appendix displays the parameters identifying the delay and cost of processing a unit of data for each information processor.

Each information processor is connected to a public network (Datapac). As the volume of data transferred between two cities, in pairs Calgary-Edmonton and Toronto-Halifax, is higher in comparison to that between other combinations of cities, higher rates of data transfer were arranged for with Datapac. In addition, due to the close proximity of offices of companies MONAX and MONREAL, which are both located in Calgary, a local connection with a high data transfer rate

was obtained. The dollar cost and delay for a data transfer between any pair of processors are shown in the section E-3 of Appendix E.

The distributed view of the DB, i.e., the partitioning and distribution of relations, is presented in the section E-4. Finally, the modelled queries are shown in English and also in their conjunctive normal form in the section E-5 of the appendix.

The selection of queries for modelling was performed using the following two guidelines:

1. Local queries are ignored as they are processed by local data base processors only.
2. An effort was made to choose queries that span several processors in such a way that the selection would contain queries with a variety of one and two-variable terms, and referencing a different number of partitioned relations.

All queries are assumed to be equally "important", their frequencies are ignored. It may be argued that the QPS for a frequent query is more significant than a QPS for an infrequent query. However, a formulator optimizes a QPS regardless of the query frequency. Furthermore, the frequency of queries determines access rates to relations and their attributes which is the key parameter for distributing relations and which may affect the logical and physical design of the DB itself.

5.8 MODELLING METHOD AND SOFTWARE

Now that the model and the chosen application have been described, integration of the strategy formulation algorithms within the framework of the model and modelling software organisation are briefly discussed.

INTEGRATION OF ALGORITHMS WITHIN THE FRAMEWORK OF THE MODEL

Algorithm OPT_DV, presented in Chapter 3, creates the space of available strategies for processing a given query. The model of a relational DDB was applied to this algorithm in the following way.

Recall that the input to the algorithm is comprised of

- (i) $\{V'\}^q$, a set of query one-variable terms;
- (ii) $\{\bar{V}''\}^q$, a set of query two-variable terms; and
- (iii) $\{(x,c,I)\}^q$, a set of triplets containing information about relations referenced by the query's one and two-variable terms.

The query one and two-variable terms contain references to the relations described in the Relation Specification Table. Information in $\{(x,c,I)\}^q$ corresponds to parameters of the Relation Specification table. A sample input of the specification for a simple query and the application's distributed environment parameters are shown in Appendix F.

The algorithm creates a graph which represents the space of the query strategies. The graph level 0 elements represent elementary relations with processed restrictions and projections. The cost of each relation is determined by utilisation of cost functions for initial processing, presented in the section 5.5; the cost is obtained in terms of each of the nine measures.

An element of level $p > 0$ represents a relation which is a result of p operations (joins and/or unions). It is either

- (i) the result of a join of two relations which are not partitioned horizontally;
- (ii) a union of fragments; or
- (iii) a partition of a fragmented join result.

The cumulative cost of a result of any of the above operations is derived for all measures by cost functions presented in the section 5.5.

Elements of the "final" graph level represent the query result, each obtained by a unique strategy. The strategy is determined by following the "predecessor" pointers, i.e., pointers to elements of lower graph levels, from the element representing the query result to the elements of the graph level 0.

Application of the model to the heuristic algorithm `NR_HEU_DV` of Chapter 4 is similar.

MODELLING METHOD

The modelling of an application of a relational DDB includes three steps:

1. The parameters for the relational DB, processors and the network are prepared and input to the modelling software.
2. Queries are prepared for input to the modelling software.
3. Modelling of query processing is performed.

The first step may require a lot of effort, but it is

straightforward. The second step requires some explanation. A query is translated from its English form into a conjunctive normal form expressed on the global view of the DB. The query is then input to the modelling software.

For instance, consider the following simple query on two relations:

"List all jobs for all customers. Print customer and job numbers and the job location."

The conjunctive normal form of the query is:

(JOB_CUSTOMER.JOB# = JOB_OFFICE.JOB#)

The query references two relations, JOB_CUSTOMER AND JOB_OFFICE, both of which are horizontally partitioned. Assume that the domains, cardinalities, partitioning of relations, distribution of relations, CPU parameters and network parameters for the above relations are the same as shown in Appendix E. The input file containing the parameters relevant to the above relations and the distributed environment is shown in Appendix F. Also is shown the input file containing the query and a sample output of modelling results.

SOFTWARE ORGANIZATION

The modelling software was developed using Pascal on a VAX/780 running under the VMS operating system. It is composed of a number of modules which, excluding comments, total approximately 3500 source lines of code. The software modules are briefly described below.

GEN, NPSIOR, NPSIOE, NPSQG: These four modules provide for input/output of the distributed environment and queries. GEN receives input in a symbolic form (for an example see Appendix F) and converts

it into an encoded form. Encoded parameters about the network and relations are then input by modules NPSIOR and NPSIOR/ respectively. The module NPSQC is used to input the encoded specification of queries.

NPSDEF: This module is the control module in which the tables identifying the distributed environment are defined. It controls input/output of the distributed environment parameters by calling procedures in NPSIOR and NPSIOR/, inputs each query using the procedures in NPSQC and for each query it controls invocation of algorithms OPT_DV and NR_HEU_DV. In addition, it also controls various levels of output for debugging purposes such as output of the graph of strategies, optimal strategies for chosen measures of cost and individual graph elements.

NTREE: The module contains implementations of algorithms OPT_DV and NR_HEU_DV. It uses procedures of other modules for cumulative cost calculations, debug output and creation of elements representing results of fragmented joins and unions.

NTREEPROC: Functions to determine the cumulative cost of partial results are contained in this module.

NTREEPRAG: Procedures which create representation for results of unions and fragmented joins are contained in this module.

NPSDSP, NPSESA: The modules contain service procedures for locating optimal strategies and a selective output such as printing of selected strategies graph levels and of elements for a selected strategy.

NPSSTATS: For each query, the optimal strategy cost is stored for each cost measure. The role of this module is to output this information and also to calculate and print statistics which are discussed in the next chapter.

5.9 SUMMARY

The parameters and cost functions which are needed to model a relational DDB were defined in this chapter. How algorithms OPT_DV and NR_HEU_DV are integrated with the model was also outlined.

The parameters and queries of an application of a DDB were described in order to prepare for a discussion on results of its, i.e., DDB application's, modelling which will be offered in the next chapter. Finally, the modelling software organization was overviewed.

In summary, the chapter presented all material which is necessary for discussing modelling results of which may be issued by users of a particular DDB application.

CHAPTER 6

DDb APPLICATION MODELLING

6.1 INTRODUCTION

At the risk of redundancy, the terminology associated with a QPS and its formulation is reviewed. Recall that a strategy to process a given query has already been defined as a specification of a sequence of relational operations and the locations of their executions. The strategy also implicitly defines the transfer of relations and partial results.

The processing of a user-issued query is composed of two phases. First, a strategy for its processing is determined in the "strategy formulation" phase. Subsequently, the strategy is executed in the "strategy execution" phase, i.e., relational operations and the necessary data transfer are performed.

In the "QPS formulation" phase an optimizing algorithm, such as NR_HEU_DV described in Chapter 4, is used. Its purpose is to formulate a strategy to process a given query, i.e., it is to determine a QPS. The strategy is such that its execution minimizes some measure of cost, for instance the total dollar cost or the strategy's execution delay. The algorithm is also called a formulator, as it formulates a strategy which is used in the "strategy execution" phase.

Algorithms OPT_DV and NR_HEU_DV, described in chapters 3 and 4 respectively, are examples of formulators, i.e., they are used to determine a strategy for processing a given query.

The design of a DDBMS includes the choice of an optimization algorithm for the formulation of QPSs. The designers must either choose one of the algorithms (formulators) proposed in the scientific literature or face the task of creating their own. If the designers adopt the former then they must address several issues arising in making the selection, amongst formulators, for a particular application of a DDB:

1. The cost function for the QPS formulation may not be obvious. If a strategy formulator which produces strategies minimizing one particular measure of cost is chosen, then it is imperative for the designers also to examine the cost of those strategies in other measures. For example, consider strategies which minimize their execution delay. The total dollar cost of such strategies may be prohibitively high due to the parallelism in data transfer and CPU processing which might be introduced in order to reduce the execution delay. Perhaps strategies which minimize the total size of partial results are a good compromise if they incur acceptable delay at a lower total dollar cost.
2. An optimizing algorithm considered for adoption by a designer may not minimize the strategy's cost which is suitable for the intended application. Most algorithms proposed in the scientific literature for a QPS formulation minimize the strategy's total volume of data transferred over the network. If the DDB application is such that the query response time should be minimized, the designer should compare the delay of strategies produced by the formulator under consideration to that of minimum delay strategies.

3. To choose just between two formulators is not a simple task even if they minimize the strategies' cost measured in the same units, ie., even if they utilize the same measure of cost. As the two formulators are likely to be heuristic in nature, the designer must not only compare the cost of their strategies but also their overhead.

It is proposed here that modelling of QP, using a model of a relational DDB such as described in Chapter 5, can provide sufficient information to address the issues listed above. This proposal is supported by modelling of a hypothetical application of a DDB.

The model of a relational DB for QP in a distributed environment, presented in Chapter 5, was integrated with algorithms OPT_DV and NR_HEU_DV to enable modelling of applications of a DDB under the assumptions listed in Figure 1-3. A particular application, also described in Chapter 5, was modelled in order to

- (i) examine the relationship amongst the various measures of cost which can be adopted for the purpose of a QPS formulation;
- (ii) compare overhead incurred by QPS formulators under consideration;
- (iii) compare the cost of strategies produced by the formulators and
- (iv) demonstrate how information obtained through modelling can be used in evaluation of QPS formulators.

The first objective of modelling is to compare the cost of processing queries with and without optimization of strategies. If an algorithm for optimizing strategies is to be used there should be some evidence to support the fact that its inclusion in a DDBMS is worth the effort. For this purpose, section 6.2 examines the average and maximum

cost of strategies with and without their optimization.

The second objective is to demonstrate how optimal strategies can be used in choosing an appropriate measure of cost for an application of a DDB. Optimal strategies for various measures of cost were found and are compared, in section 6.3, in order to determine the most suitable measure of cost. Information and a method for its presentation is developed; they enable an examination of the effect of choosing one measure of cost, in a QPS optimization, on the cost of strategies in other measures.

The third objective relates to choosing the measure of cost and also to the "two-staged" approach which can be adopted in formulation of strategies (Chapter 2). It can be argued that the sum of the size of partial results can be used to minimize the total volume of data which is processed by CPUs and also transferred over the network. This argument is presented in the section 6.4 and whether or not it holds for the modelled application of a DDB is also examined.

In Chapter 2, it was proposed that a QPS formulation should be performed in the following two stages: (i) First, the sequence of relational operations, which minimizes the total size of partial results, is determined and then (ii) the optimal locations for the operations' execution is found. It has been explained already that this approach does not lead to the minimal time delay strategies. The delay of strategies, for the modelled application, generated by this method is compared to that of the minimal time delay strategies.

The final objective is to demonstrate how optimal strategies can be used as a tool in evaluation of strategy optimizing algorithms (QPS formulators) which are under consideration for inclusion into a DDEMS (section 6.5).

Before the material of subsequent sections is presented, notation, which is common to all sections, is defined. Let

N_q ... denote the number of queries for the modelled application of a DDB. Recall that if a capitalized "N" is used to denote a number of entities of a certain type, such as N_q , its sub-script is not used as an index but, rather, as a reminder to the reader about the symbol's use. In the context of modelling results, reported in this chapter, $N_q = 21$, i.e., processing of 21 queries was modelled;

$q_i, i=1,2,\dots,N_q$... denote a query i ;

n_i ... denote the number of unique strategies for processing a query q_i . Recall that a strategy is a specification of a sequence of relational operations and the locations of their executions. Also note that the sub-script "i", in " n_i ", is used as an index identifying a particular query;

$A_i^j, B_i^k, \dots$ denote unique strategies "j" and "k", respectively, for processing the query q_i . The super-script identifies the strategy to process a query which is identified by the sub-script. $j, k=1,2,\dots,n_i$.

Recall that the cumulative cost of a relation R using a specific cost measure is denoted by

$T_a[R]$... delay due to CPU processing and data transfer;

$T_p[R]$... delay due to CPU processing only;

$T_t[R]$... delay due to a data transfer only;

$D_p[R]$... dollar cost due to CPU processing only;

$D_x[R]$... dollar cost due to a data transfer only;

$D_a[R]$... dollar cost due to CPU processing and data transfer;

$S_p[R]$... volume of data processed by CPUs;

$S_x[R]$... volume of data transferred; and

$S_a[R]$... sum of the size of partial results.

Also recall the distinction between the measures S_p , the cumulative volume of data processed by CPUs, and S_a , the cumulative sum of the size of partial results. The measure, S_p includes the total volume of data transferred (retrieved and/or stored) by any CPU to/from its secondary storage device(s) while S_a includes only the size of results obtained by execution of relational operators.

For example, consider a join of R_1 with R_2 , resulting in the relation R . Using the measure S_a , this join contributes the relation's R size to the total sum of the size of partial results. Using the measure S_p , the total volume data, transferred (retrieved/stored) between the CPU, executing the join, and its secondary storage device(s), is contributed to the the total volume of data processed by CPUs.

Since this chapter deals with the cumulative cost of strategies, rather than those of partial results, it is necessary to develop notation representing the cost of a strategy in a particular measure of interest. Let

$g[A_i^j]$... denote the cumulative cost of the strategy A_i^j in the measure g , where g represents T_a , T_p , T_x , D_p , D_x , D_a , S_p , S_x and S_a . For instance, assuming that a relation R represents the query q_1 result obtained by a unique strategy A_1^j .

if $g=D_a$ then $g[A_i^j] = D_a[A_i^j] = D_a[R]$

and it represents the cumulative dollar cost due to both CPU processing and data transfer for the query q_i result, relation R , derived by the strategy A_i^j .

Let h be another cost function, similarly defined. It also represents $T_a, T_p, \dots, S_a$. If B_i^k is a unique strategy "k" to process a query q_i and $h=T_p$, then $h[B_i^k] = T_p[B_i^k]$ denotes the total time delay due to CPU processing incurred by the strategy B_i^k in processing q_i .

Also let

$\hat{A}_{i|g}, \hat{A}_{i|h} \dots$ denote strategies, for processing q_i , which minimize the cost in measures g and h respectively:

$$g[\hat{A}_{i|g}] = \min_{\{j\}} \{ g[A_i^j], j=1,2,\dots,n_i \}$$

As the strategy $\hat{A}_{i|g}$ minimizes the cost in g , its execution incurs the minimum cost, measured in g , over all n_i unique strategies A_i^j , $j=1,2,\dots,n_i$, which are available to process the query q_i . Similarly,

$$h[\hat{A}_{i|h}] = \min_{\{j\}} \{ h[A_i^j], j=1,2,\dots,n_i \}.$$

As $\hat{A}_{i|h}$ denotes a strategy, to process q_i , which is minimal with respect to the measure h , $g[\hat{A}_{i|h}]$ denotes its cumulative cost in the measure g . For instance, assume that $\hat{A}_{i|h}$ denotes a strategy, to process q_i , which minimizes the dollar cost due to both CPU processing and data transfer, i.e., $h=D_a$. This strategy is such that

$$h[\hat{A}_{i|h}] = D_a[\hat{A}_{i|h}] = \min_{\{j\}} \{ D_a[A_i^j], j=1,2,\dots,n_i \}$$

and for $g=T_x$,

$$g[\hat{A}_{i|h}] = T_x[\hat{A}_{i|h}],$$

is its total time delay due to transferring data over the network.

6.2 COST OF STRATEGIES WITH/WITHOUT OPTIMIZATION

The problem of optimizing strategies is difficult and inclusion of optimizing algorithms into a DDMS requires a certain amount of effort. To examine whether or not this effort is justified, the average and maximum cost of optimal strategies is compared to those of strategies without optimization.

Let

$\bar{c}[g]$... denote the average cost, in g , over all strategies available for processing all queries, i.e.,

$$\bar{c}[g] = \frac{1}{N_q} \sum_{i=1}^{N_q} \left\{ \frac{1}{n_i} \sum_{j=1}^{n_i} g[A_i^j] \right\}$$

where the outer summation is for queries $i = 1, 2, \dots, N_q$ and the inner one is for all strategies A_i^j , $j=1, 2, \dots, n_i$, of a query q_i ;

$c_m[g]$... denote the cost of the worst strategy, over all queries, for the measure g , i.e.,

$$c_m[g] = \max_{\{i,j\}} \{ g[A_i^j]; i=1,2,\dots,N_q; j=1,2,\dots,n_i \}$$

$$\max_{\{i\}} \{ \max_{\{j\}} (g[A_i^j], j=1,2,\dots,n_i); i=1,2,\dots,N_q \};$$

$\bar{c}[h,g]$... denote the average cumulative cost, measured in g , over all queries $q_i, i=1,2,\dots,N_q$, such that each query q_i is processed by a strategy which minimizes the cumulative cost measured in h :

$$\bar{c}[h,g] = \frac{\sum_{i=1}^{N_q} g[\hat{A}_{i|h}]}{N_q}$$

Recall that $\hat{A}_{i|h}$ denotes the minimum cost strategy, to process q_i , for the measure h and $g[\hat{A}_{i|h}]$ is its cost in g . $\bar{c}[g,g]$ is the average cost, in g , for strategies which are best with respect to g , i.e.,

$$\bar{c}[g,g] = \frac{\sum_{i=1}^{N_q} g[\hat{A}_{i|g}]}{N_q} =$$

$$\min_{\{h\}} \{ \bar{c}[h,g], h=T_a, T_p, \dots, S_a \};$$

$c_m[h,g]$... denote the maximum cost, in g , over all queries such that each query q_i is processed by its optimal strategy with respect to the measure $h, \hat{A}_{i|h}$:

$$c_m[h,g] = \max_{\{i\}} \{ g[\hat{A}_{i|h}], i=1,2,\dots,N_q \}$$

Each query q_i is processed by a strategy $\hat{A}_{i|h}$ which minimizes the cost measured in h .

Table 6-1(a) displays the average cost, in all measures over all queries, of strategies with and without optimization. It shows the average costs $\bar{c}[g]$ and $\bar{c}[g,g]$ for each measure of cost $g, g = T_a, T_p$.

..., S_a . Note that on the average there are 1215 unique strategies to process each of the $N_q = 21$ queries.

Table 6-1(b) shows the worst cost, over all queries, of strategies with and without optimization, i.e., it shows $c_m[g]$ and $c_m[g,g]$ for each measure of cost g , $g = T_a, T_p, \dots, S_a$.

An examination of these two tables reveals that, if the formulation overhead is neglected, optimization of strategies is desirable for this example no matter which one of the measures of cost are considered. Without optimization, depending on the measure g , the average cost of strategies is between 2.56 to 9.81 times higher than the average cost with optimization. The average delay (cost measure $g=T_a$) of strategies without optimization is about 6.4 times higher than that of optimized strategies. The maximum cost, over all strategies without optimization, is between 8.2 to 34.6 times higher than the maximum cost of optimized strategies.

Note that the reported average and maximum delays are for the strategy execution phase; the overhead/delay due to formulation of strategies is ignored. It will later be shown that the formulation of optimal strategies is undesirable precisely because of its high delay which is inherent due to its probable NP-hardness (see Chapter 2). Consequently, optimization of strategies is desirable if an optimizing algorithm can be found such that

- (i) its overhead is low and
- (ii) its strategies' execution cost is close to that of optimal strategies.

It will be shown in subsequent sections that the heuristic algorithm NR_HEU_DV (Chapter 4) satisfies the above two conditions.

6.3 COST MEASURE

As stated previously, although most QPS formulators are claimed to minimize the query response time, the variable which is actually optimized is the total volume of data transferred over the network (see Table 1-1). Other cost measures can also be used for optimization of a QPS. If the ultimate objective of the QPS formulation is the minimization of the query response time, but for various reasons some other variable is actually optimized, it is important to find how close the strategies' delay is to that of optimal delay strategies.

Even if the optimized variable is the strategy execution delay it is important and instructive to find the cost in terms of other measures; for example in terms of the network or CPU processing dollar costs.

This section demonstrates how optimal strategies and modelling of queries can be used to address the issues raised above for a particular environment of a DDB.

AVERAGE CUMULATIVE COST

Recall that

$\bar{c}[h,g]$... denotes the average cumulative cost, measured in g , over all queries q_i , $i=1,2,\dots,N_q$, such that each query q_i is processed by a strategy which minimizes the cumulative cost measured in h , $A_{i|h}$:

$$\bar{c}[h,g] = \frac{\sum_{i=1}^{N_q} g[A_{i|h}]}{N_q}$$

$A_{i|h}$ is the minimum cost strategy, to process q_i , for the measure h and

TABLE 6-2 AVERAGE COST (IN ϵ) OF STRATEGIES OPTIMIZING h $\bar{c}[h, g]$ is shown for each combination of h and g

AVERAGE COST MEASURED IN ϵ

$h \backslash g$	STRATEGIES ARE OPTIMIZED WITH RESPECT TO h									
	T_a	T_p	T_z	D_p	D_z	D_a	S_p	S_z	S_a	
T_a	20.36	11.02	10.58	5.47	0.40	3.87	180323.95	3044.50	31000.90	
T_p	32.34	9.41	23.90	3.74	0.78	4.53	195058.80	7365.09	34931.77	
T_z	22.56	13.30	10.15	3.71	0.35	4.06	191744.95	3159.67	32369.27	
D_p	49.96	13.06	38.88	2.55	1.02	3.57	143079.84	9215.24	28295.85	
D_z	23.14	13.53	11.18	3.19	0.33	3.52	170350.78	3030.03	30078.79	
D_a	28.20	12.72	15.86	2.55	0.45	2.99	143079.84	3994.55	28295.85	
S_p	49.96	13.06	38.88	2.55	1.02	3.57	143079.84	9215.24	28295.85	
S_z	23.22	13.33	11.41	3.19	0.34	3.53	170377.11	3020.32	30081.84	
S_a	25.71	11.91	14.29	2.62	0.44	3.06	145191.06	3819.29	27619.45	

$g[\hat{A}_{1|h}]$ is its cost in g .

Table 6-2 displays the average cumulative cost, in each of the measures $g = T_a, T_p, \dots, S_a$, over all queries processed by strategies which are minimum with respect to h . In other words, $\bar{c}[h,g]$ is tabulated for all combinations of h and g . The table element in a row corresponding to the optimized measure h , and a column, corresponding to the measure g , shows the average cumulative cost, in g , for best/minimum strategies with respect to h .

For instance, the fourth row of the table shows the average cumulative cost in $g = T_a, D_p, D_x, \dots, S_a$, for strategies which minimize the queries' total dollar cost due to CPU processing; the row corresponds to the optimized measure $h=D_p$. Each query q_i , $i=1,2,\dots,N_q$, is assumed to be processed by the strategy $\hat{A}_{1|h}$ which minimizes the total dollar cost due to CPU processing. The first element of this row shows $\bar{c}[h=D_p, g=T_a]$, the average delay, over all queries, due to both CPU processing and data transfer.

Note that any diagonal element $\bar{c}[g,g]$ is the minimum element in its column as it shows

$$\begin{aligned} \bar{c}[g,g] &= \min_{\{h\}} \{ \bar{c}[h,g], h=T_a, T_p, \dots, S_a \} = \\ &= \sum_{i=1}^{N_q} g[\hat{A}_{1|g}] / N_q \end{aligned}$$

ie., it shows the average minimum cost, in g , for strategies which are best with respect to g .

Most algorithms proposed for a QPS formulation minimize the total volume of data transferred over the network and neglect the CPU processing delay altogether; they assume that the data transfer

cost/delay is dominant. It is instructive to examine whether this assumption also applies to the modelled application of a DDB which is described in Chapter 5. The first row elements of Table 6-2 correspond to the average cost, measured in $g = T_a, T_p, \dots, S_a$, of strategies which minimize the total delay due to CPU processing and data transfer, i.e., which minimize the measure $h = T_a$. The first three elements of this row are:

$\bar{c}[T_a, T_a] = 20.36$ sec. ... is the average delay for strategies which minimize the delay due to both CPU processing and data transfer.

$\bar{c}[T_a, T_p] = 11.02$ sec. ... is the average delay for the same strategies only this time the measured delay is only due to CPU processing.

$\bar{c}[T_a, T_x] = 10.58$ sec. ... is the average delay for the same strategies only this time the measured delay is only due to transferring data over the network.

The elements indicate that the delay due to CPU processing is of the same significance as that due to transferring data over the network. This observation is also supported by the following elements of Table 6-2. Element

$\bar{c}[h = T_p, g = T_p] = 9.41$ sec. ... is the average delay, due to CPU processing only, for strategies which optimize only the CPU processing delay and

$\bar{c}[h = T_x, g = T_x] = 10.15$ sec. ... is the average delay, due to transferring data over the network, for strategies which optimize the delay only due to a data transfer.

Again, these averages support the observation that delays due to CPU processing and delays due to a data transfer are of approximately the same significance. This seems to imply that if a choice had to be made between two QPS formulators, one which minimizes the strategies' CPU delay and one which minimizes their delay due to transferring data, either formulator could be used. This conclusion, however, is incorrect.

Examine the second and third row of the table. The second row shows the average cost, for each measure g , of strategies which minimize the delay only due to CPU processing, i.e., it shows $\bar{c}[h=T_p, g]$, for $g = T_a, T_p, \dots, S_a$. The third row shows the average cost, for each g , of strategies which minimize the delay only due to a data transfer, i.e., it shows $\bar{c}[h=T_x, g]$, for $g = T_a, T_p, \dots, S_a$. With the exception of the cost measured in $g=T_p$, the average costs for all other measures g are higher in the second row than those in the third row; the average cost of strategies optimized for the measure $h=T_p$ is higher, in any measure g but $g=T_p$, than those of strategies optimized for the measure $h=T_x$. Consequently, of the two QPS formulators, one which minimizes the strategies' delay due to CPU processing and one which minimizes the delay due to a data transfer, the results of this example indicate that the latter should be chosen. This is, of course, under the assumption that the formulators have approximately the same overhead.

MAXIMA

Since each element of Tables 6-2 represents an average over all modelled queries, it is instructive to find some characteristics for the distributions of which the averages are presented. Recall that $c_n[h, g] \dots$ denotes the maximum cumulative cost measured in g over all

queries processed by best strategies with respect to h:

$$c_m[h,g] = \max_{\{i\}} \{ g[\hat{A}_{i|h}], i=1,2,\dots,N_q \}$$

Each query q_i is processed by a strategy $\hat{A}_{i|h}$ which minimizes the cost measured in h. Table 6-3 shows $c_m[h,g]$ for each combination of h and g. The element of row "h" and column "g" shows $c_m[h,g]$. It represents the maximum cumulative cost, in g, over all queries processed by best strategies with respect to h. Consequently, the diagonal elements $c_m[h,h]$ are smallest in their columns, as they represent the maximum cost, measured in h, for strategies which are best with respect to h.

Since elements $c_m[h,g]$, of Table 6-3, show the maximum cumulative cost over all queries, they are likely to be dominated by "large sized" queries, i.e., queries which have many two-variable terms referring to large sized relations.

Consider a strategy $\hat{A}_{i|h}$, which minimizes the measure $h=S_a$ for a query q_i , i.e., which minimizes the q_i 's total size of partial results. Assume that the query q_i is by far the "largest" query, i.e., it has many more two-variable terms referring to large sized relations than any other query. Although this strategy may incur a total time delay which is close, or even equal, to the minimum time delay for this query, its total time delay is likely to appear in Table 6-3 for the element $c_m[h=S_a, g=T_a]$.

Similarly, consider a strategy $\hat{A}_{k|h}$, $h=S_a$, which also minimizes the total size of partial results but this time for a "small sized" query q_k . In other words, the query q_k has a small number of two-variable terms which refer to only small sized relations. Although this strategy may incur a total time delay which is many times higher than the minimum time delay for this query, its total delay is not

TABLE 6-3 MAXIMUM COST (IN \$) OF STRATEGIES OPTIMIZING h $c_a[h, g]$ is shown for each combination of h and g

MAXIMUM COSTS IN \$										
$g \backslash h$	T_a	T_p	T_x	D_p	D_x	D_a	S_p	S_x	S_a	
T_a	161.59	67.79	93.80	17.56	2.80	18.82	909704.50	21640.63	135200.00	
T_p	298.04	47.20	261.30	17.56	6.05	18.82	909704.50	46500.00	154000.02	
T_x	166.63	74.01	93.80	17.56	2.80	18.82	909704.50	21500.00	135200.00	
D_p	280.59	74.01	242.88	8.73	6.01	14.72	657560.13	66750.00	119000.04	
D_x	166.63	74.01	93.80	11.61	2.80	12.91	657560.13	21500.00	119000.01	
D_a	166.63	74.01	93.80	8.73	2.80	11.53	657560.13	21500.00	119000.01	
S_p	280.59	74.01	242.88	8.73	6.01	14.72	657560.13	66750.00	119000.01	
S_x	166.63	74.01	93.80	11.61	2.80	12.91	657560.13	21500.00	119000.01	
S_a	161.59	67.79	93.80	9.47	2.80	11.53	657560.13	21500.00	119000.01	

STRATEGIES ARE OPTIMIZED WITH RESPECT TO h

likely to appear in Table 6-3 for the element $c_m[h-S_a, g-T_a]$.

From the above discussion it becomes apparent that the cost of one large sized query may dominate/appear in all elements of Table 6-3. The effect of large sized queries is even more important on the averages shown in Table 6-2. Large sized queries are likely to significantly effect the average cost over all measures. In order to examine the effect of using one measure of cost in optimization of strategies on the cost of strategies in other measures, some other form for presentation of information should be used. A suitable form of information presentation is using ratios which are discussed and defined below.

RATIOS

One of the most important aspects of the use of an algorithm, which minimizes a cost measure h , is the effect of that algorithm on a different cost measure, g . For example, given that a QPS formulator produces strategies which minimize the total size of partial results, it is important to examine how close, or how much worse, is the total delay of these strategies with respect to that of the minimum delay strategies. This effect can be examined by using ratios defined below.

Let

$r[i, h, g]$... denote the ratio of the cost, measured in g , of a strategy $A_{i|h}$, which minimizes the cost in h for the query q_i , to the cost of the strategy $B_{i|g}$, which minimizes g for q_i , i.e.,

$$r[i, h, g] = g[A_{i|h}] / g[B_{i|g}] \text{ and}$$

$\bar{r}[h, g]$... denote the average of ratios $r[i, h, g]$, over all queries q_i ,

$i=1, 2, \dots, N_q$, i.e.,

$$\bar{r}[h,g] = \frac{\sum_{i=1}^{N_q} r[i,h,g]}{N_q}$$

Table 6-4 shows the ratios for all combinations of h and g . For example, the table element $\bar{r}[h=S_a, g=T_a]$ shows how much higher the average delay of strategies, which are best with respect to the total size of partial results, is in comparison to strategies which incur minimal delay. The diagonal elements, $\bar{r}[h,h]$, are, of course, equal to 1. The average for each row, i.e., for each optimized measure $h = T_a, T_p, \dots, S_a$, is also shown.

Note that an element of Table 6-4, such as $\bar{r}[h=S_a, g=T_a]$, is not formed directly from elements of Table 6-2 by a simple evaluation

$$\bar{r}[h=S_a, g=T_a] = \bar{c}[h=S_a, g=T_a] / \bar{c}[h=T_a, g=T_a]$$

As the definition of $\bar{r}[h,g]$ indicates, it is the average of ratios over all queries, such that for each query q_i the ratio $r[i,h,g] = g[\hat{A}_{i|h}] / g[\hat{B}_{i|g}]$ is used. Consequently, $\bar{r}[h=S_a, g=T_a]$ is evaluated as

$$\bar{r}[h,g] = \frac{\sum_{i=1}^{N_q} r[i,h,g]}{N_q} = \frac{\sum_{i=1}^{N_q} g[\hat{A}_{i|h}] / g[\hat{B}_{i|g}]}{N_q}$$

where $h=S_a$ and $g=T_a$.

Examination of Table 6-4 confirms that the results are as expected. Since both CPU processing and data transfer are significant, the rows corresponding to optimized measures which include both CPU processing and data transfer, i.e., which correspond to

TABLE 6-4 AVERAGE RATIOS OF COSTS (IN g) OF STRATEGIES OPTIMIZING h $\bar{r}[h, g]$ is shown for each combination of h and g

AVERAGE RATIOS OF COSTS MEASURED IN g											row average
$h \backslash g$	T_a	T_p	T_x	D_p	D_x	D_g	S_p	S_x	S_g		
T_a	1.00	1.26	1.07	1.14	1.21	1.11	1.11	1.18	1.06	1.13	
T_p	1.67	1.00	2.90	1.54	4.69	1.67	1.40	4.14	1.34	2.27	
T_x	1.45	2.28	1.00	1.49	1.15	1.43	1.51	1.08	1.27	1.41	
D_p	1.85	1.34	7.01	1.00	10.34	1.10	1.00	10.16	1.02	3.86	
D_x	1.46	2.34	1.05	1.43	1.00	1.37	1.46	1.01	1.24	1.37	
D_g	1.22	1.33	1.38	1.00	1.26	1.00	1.00	1.26	1.08	1.16	
S_p	1.85	1.34	7.01	1.00	10.34	1.10	1.00	10.16	1.02	3.86	
S_x	1.49	2.23	1.13	1.43	1.11	1.39	1.46	1.00	1.24	1.38	
S_g	1.17	1.31	1.30	1.02	1.30	1.03	1.02	1.25	1.00	1.15	

STRATEGIES ARE OPTIMAL WITH RESPECT TO h

T_a ... delay due to both CPU processing and data transfer;
 D_a ... dollar cost due to both CPU processing and data transfer;
 and
 S_a ... total size of partial results,

have the lowest average values, values 1.13, 1.16 and 1.15, respectively. This means that the cost of strategies, which are formulated to minimize any one of the three measures T_a , D_a and S_a , is also favourable in other measures. For instance the cumulative dollar cost due to CPU processing of strategies which minimize the total delay due to CPU processing and data transfer, element $\bar{r}[h=T_a, g=D_p]$, is only slightly higher in comparison to that of strategies which minimize the dollar cost only due to CPU processing, element $\bar{r}[h=D_p, g=D_p]$.

Note how close the average values are to each other. This indicates that the use of these three measures in optimization of QPSs leads to strategies of comparable cost; the three cost measures may be considered to be equivalent/interchangeable for the purpose of optimization.

The rows corresponding to optimized measures relating to a data transfer, i.e., which correspond to optimized measures

T_x ... delay due to transferring data;
 D_x ... dollar cost due to transferring data; and
 S_x ... total volume of data transferred over the network,

have average values of 1.41, 1.37 and 1.38, respectively.

Again, note how close these averages are. Strategies optimized using one of the three measures T_x , D_x and S_x , will, on the average, result in costs, in other measures, which are about 1.4 higher than the

cost of optimal strategies for those other measures.

Finally, the rows corresponding to optimized measures relating only to CPU processing, i.e., which correspond to optimized measures

T_p ... delay due to CPU processing;

D_p ... dollar cost due to CPU processing; and

S_p ... total volume of data processed by CPUs.

have average values of 2.27, 3.86 and 3.86, respectively.

Obviously, these measures should not be chosen for optimization of strategies in this application of a DDB. Although these averages are not as close to each other than those of the previous two groups of measures, two of these measures, D_p and S_p , yield strategies of comparable costs when used in formulation of QPSs.

Now examine the columns of Table 6-4. Each column represents elements $\bar{r}[h,g]$, where $h = T_a, T_p, \dots, S_a$. Each element of a column corresponding to the cost measure g , represents the average ratio of the cost, in g , for strategies which minimize h , to that of strategies which minimize g . For instance, the second column contains elements $\bar{r}[h,g=T_p]$, $h = T_a, T_p, \dots, S_a$. The first element of this column, $\bar{r}[h=T_a,g=T_p]$, is the average ratio of the cost, in $g=T_p$ (delay due to CPU processing), of strategies which minimize $h=T_a$ (delay due to CPU processing and data transfer), to that of strategies which minimize $g=T_p$.

Examine the columns for $g = T_p, D_p$ and S_p , i.e., the columns corresponding to measures only in terms of CPU processing. The smallest elements in these columns are in the rows which correspond to optimized measures $h = T_a, D_a, T_p, D_p, S_p$ and S_a . The largest elements

are those for the optimized measure $h = T_x, D_x$ and S_x , measures which ignore the cost due to CPU processing. As expected, minimizing the cost, measured only in terms of a data transfer, results in the worst cost measured in terms of the CPU processing.

Similarly, examine the columns for $g = T_x, D_x$ and S_x , ie., the columns which correspond to measures only in terms of transferring data over the network. The worst/highest average ratios are in the rows which correspond to the optimization of measures $h = T_p, D_p$ and S_p . As expected, minimizing the cost, measured only in terms of CPU processing, results in the worst cost measured in terms of data transfer.

MAXIMUM RATIOS

Let

$r_m[h,g]$... denote the maximum ratio over all queries of the cumulative cost, in g , incurred by a strategy which minimizes h , to the cost, in g , of a strategy which minimizes g :

$$r_m[h,g] = \max_{\{i\}} \{ r[i,h,g], i=1,2,\dots,N_q \} = \max_{\{i\}} \{ g[\hat{A}_{i|h}] / g[\hat{B}_{i|g}], i=1,2,\dots,N_q \}$$

Table 6-5 presents $r_m[h,g]$ for all combinations of h and g . For example, $r_m[h=S_a, g=T_a]$ shows the maximum ratio of the total delay of strategies, which minimize the total size of partial results, to that of strategies which incur the minimum total delay. This ratio is the maximum for all queries $q_i, i=1,2,\dots,N_q$. In other words it shows the maximum/worst ratio of the total time delay, for a strategy which optimizes the measure $h=S_a$, to that of the best delay strategy for that

TABLE 6-5 MAXIMUM RATIOS OF COST (IN ϵ) OF STRATEGIES OPTIMIZING h
 $r_m[h, \epsilon]$ is shown for each combination of h and ϵ

MAXIMUM RATIOS OF COSTS IN ϵ

$\epsilon \backslash h$	T_a	T_p	T_x	D_p	D_x	D_a	S_p	S_x	S_a
T_a	1.00	2.62	1.47	2.33	2.17	1.94	2.09	2.17	1.51
T_p	5.72	1.00	22.01	3.47	51.36	3.70	2.75	56.89	2.47
T_x	5.78	9.39	1.00	4.88	2.02	4.72	5.78	1.68	3.26
D_p	5.60	2.71	80.04	1.00	166.87	1.68	1.00	166.88	1.23
D_x	5.78	11.13	1.66	4.88	1.00	4.72	5.78	1.13	3.26
D_a	3.18	2.97	5.45	1.00	4.09	1.00	1.00	4.04	1.23
S_p	5.60	2.71	80.04	1.00	166.87	1.68	1.00	166.88	1.23
S_x	5.78	9.39	2.34	4.88	1.71	4.72	5.78	1.00	3.26
S_a	3.16	2.62	5.45	1.13	4.09	1.11	1.13	4.04	1.00

STRATEGIES ARE OPTIMIZED WITH RESPECT TO h

query.

Information represented by Table 6-5 confirms observations made from Table 6-4 which shows averages of ratios. In general, the smallest maximum ratios can be found in the rows for optimized measures in terms of both CPU processing and data transfer, i.e., for $h = T_a, D_a$ and S_a and the worst for the rows which correspond to optimized measures $h = T_p, D_p$ and S_p .

The worst ratios, for measures only in terms of CPU processing, i.e., for $g = T_p, D_p$ and S_p , can be found in the rows for optimized measures in terms of the network data transfer, i.e., for $h = T_x, D_x$ and S_x .

Similarly, the worst ratios, for measures only in terms of the network data transfer, i.e., for $g = T_x, D_x$ and S_x , can be found in the rows for optimized measures in terms of CPU processing, i.e., for $h = T_p, D_p$ and S_p .

STANDARD DEVIATIONS

Recall that each element $\bar{r}[h,g]$, of Table 6-4, represents the average cost ratios over all queries:

$$\bar{r}[h,g] = \frac{\sum_{i=1}^N r[i,h,g]}{N_q} = \frac{\sum_{i=1}^N s[\hat{A}_{i|h}]}{s[\hat{B}_{i|g}]}$$

Maxima of these ratios are shown in Table 6-5 for all combinations of h and g . To examine the dispersion of these ratios, the standard deviation of discrete values

$$r[i, h, g] = \hat{g}[\hat{A}_{i|h}] / \hat{g}[\hat{B}_{i|g}], i=1, 2, \dots, N_q,$$

are reported in Table 6-6 for each combination of h and g .

Let

$\sigma(r[i, h, g])$... denote the standard deviation of the distribution of ratios $r[i, h, g]$, $i=1, 2, \dots, N_q$:

$$\sigma(r[i, h, g]) = \sqrt{\frac{\sum_{i=1}^n \{x_i - \mu\}^2}{n}},$$

which is the standard deviation formula where

$n=N_q$ is the number of queries;

μ is the average over all ratios, i.e., $\mu = \bar{r}[h, g]$; and

x_i represents the ratio $r[i, h, g] = \hat{g}[\hat{A}_{i|h}] / \hat{g}[\hat{B}_{i|g}]$ for the query q_i , where

$\hat{A}_{i|h}$... is a strategy, to process q_i , which is best with respect to h and

$\hat{B}_{i|g}$... is a strategy, to process q_i , which is best with respect to g .

The standard deviations just confirm the previously made observations using tables 6-4 and 6-5, i.e., using ratios' averages and maxims, respectively. At the risk of redundancy, they are repeated.

The smallest deviations can be found in the rows for optimized measures in terms of both CPU processing and data transfer, i.e., for $h = T_a, D_a$ and S_a and the worst for the rows which correspond to optimized measures $h = T_p, D_p$ and S_p .

TABLE 6-6 STANDARD DEVIATION OF RATIOS OF COSTS

 $\sigma(r[i, h, g])$ is shown for each combination of h and g
STANDARD DEVIATION OF RATIOS OF COSTS IN g

$h \backslash g$	T_h	T_p	T_x	D_p	D_x	D_h	S_p	S_x	S_h
T_h	0.00	0.37	0.13	0.32	0.37	0.24	0.25	0.34	0.13
T_p	1.07	0.00	4.41	0.78	10.57	0.86	0.60	7.59	0.44
T_x	1.09	2.28	0.00	0.95	0.26	0.88	1.10	0.19	0.54
D_p	1.27	0.40	17.24	0.00	35.10	0.15	0.00	35.14	0.05
D_x	1.14	2.56	0.15	0.93	0.00	0.87	1.09	0.03	0.54
D_h	0.50	0.43	0.97	0.00	0.69	0.00	0.00	0.68	0.05
S_p	1.27	0.40	17.24	0.00	35.10	0.15	0.00	35.14	0.05
S_x	1.11	2.26	0.31	0.93	0.23	0.87	1.09	0.00	0.54
S_h	0.48	0.37	0.95	0.04	0.68	0.04	0.03	0.66	0.00

STRATEGIES ARE OPTIMIZED WITH RESPECT TO h

An examination of the columns of Table 6-6 reveals that the worst deviations, for measures only in terms of CPU processing, i.e., for $g = T_p, D_p$ and S_p , can be found in the rows for optimized measures in terms of the network data transfer, i.e., for $h = T_x, D_x$ and S_x .

Similarly, the worst ratios, for measures only in terms of the network data transfer, i.e., for $g = T_x, D_x$ and S_x , can be found in the rows for optimized measures in terms of CPU processing, i.e., for $h = T_p, D_p$ and S_p .

SUMMARY

This section examined how optimal strategies and their cumulative costs, obtained through modelling, can be used in a selection of the cost measure which should be used for an application of a DDB. The type of information displayed in tables 6-1 to 6-6 can be obtained for any application of a DDB. It can be applied to gain insight on the use of various measures of cost in optimization of strategies. Most importantly, it can be used to find how optimization of one measure can affect the cost of strategies in other measures.

For the particular application of the modelled DDB, it is clear that optimization of strategies yields best results for measures in terms of both CPU processing and data transfer, i.e., for measure T_a, D_p and S_a .

That optimization of the delay (T_a) and dollar cost (D_a), due to both CPU processing and data transfer, should yield best results is not surprising; however, that minimization of the total size of partial results leads to costs equivalent to those of T_a and D_a is somewhat surprising and it will be elaborated upon in the next section.

Optimization of a measure only in terms of CPU processing, T_p , D_p and S_p , leads to higher costs measured in terms of a data transfer, T_x , D_x and S_x , and vice-versa; however, optimization of strategies in terms of the network data transfer is preferred to that in terms of CPU processing.

Although a particular application was used for demonstration purposes, it is clear that it could be replaced by any other application. The results may differ for other applications; however, the method of comparing the cost measures and the type of information (tables 6-1 to 6-6) needed for such a comparison remain valid.

6.4 COST MEASURE: TOTAL SIZE OF PARTIAL RESULTS

This section is devoted to utilization of the total size of partial results as a measure of cost in optimization of distributed query processing strategies. The previous section has already pointed out that, for the modelled application of a DDB, the measure S_a (total size of partial results) yields excellent results. When used as a cost measure in optimization of strategies, it results in strategies' costs which are comparable to those of strategies which minimize the cost due to both CPU processing and data transfer (measures T_a and D_a).

Using this cost measure implies that strategies should be optimized in two stages as was outlined in Chapter 2: (i) First, the sequence of relational operations, which minimizes the size of partial results, should be determined and then (ii) the optimal locations for the execution of the operations should be found.

It has been explained already, in Chapter 2, that this approach does not lead to the minimal time delay strategies. The results of the previous section, however, indicate that, at least for the modelled

application of a DDB, optimization using this two-staged approach yields strategies which perform close to those which are minimal with respect to the delay due to both CPU processing and data transfer.

This is not surprising as it can be argued that the minimization of the total size of partial results minimizes either the total volume of data processed by information processors, or transmitted over the network, depending on whether communication or CPU processing is considered to be the critical factor.

For example, in a distributed environment comprised of lower capacity CPUs connected by a high throughput local network, the total volume of processed data may be considered to be the critical parameter, and hence it is a good candidate for adoption as a measure of cost. It can be argued that, by minimizing the size of partial results, the total volume of processed data is also be minimized.

Similarly, in an environment of high capacity processors, which communicate by using the standard services provided by a public network, the total volume of data transferred over the network may be considered to be the most critical factor. It can be argued that, by minimizing the total size of partial results, the total volume of data transferred over the network is also minimized.

Although most QPS formulators are claimed to minimize the query response time, the variable which is actually optimized is the total volume of data either (i) transferred over the network or (ii) processed by any one CPU. In the first case the time delay due to a data transfer is considered to be critical, while in the second case, the CPU processing is considered to be critical.

Instead of minimizing the volume of data transferred over the network, or processed by CPUs, it may be more effective to minimize the total size of partial results. This variable may be particularly suitable in case the network cost does not clearly dominate the CPU processing cost, or vice-versa, as is the case for the application of a DDB described in Chapter 5 and of which modelling results were examined in this chapter.

6.5 QPS FORMULATOR SELECTION

An examination of Table 1-1 reveals that most algorithms proposed for formulation of strategies are heuristic in nature. The objective of this section is two-fold. The first objective is to demonstrate how optimal strategies and modelling of distributed queries can be used as a tool in evaluation of heuristic QPS formulators which are under consideration for integration with a DDBMS. The second objective is to examine delays of strategies, produced by the heuristic formulator proposed in Chapter 4, algorithm NR_HEU_DV, and overhead it (NR_HEU_DV) incurs in their formulation.

The suggested method for evaluation of heuristic QPS formulators is very simple. It is comprised of finding the delay of strategies produced by the heuristic algorithms and comparing them to delays of optimal strategies. In addition, the time it takes to formulate strategies is also taken into consideration.

Recall from Chapter 4, that the heuristic algorithm NR_HEU_DV determines a QPS by repetitively developing N levels of the query's strategy space and finding the minimum cost partial result. The developed levels of the strategy space are such that they contain all unique sub-strategies consisting of N relational operations (joins and/or unions). At each iteration, the sub-strategy for the minimum

cost partial result is adopted as a portion of the QPS. It is expected that a higher value of N yields a lower cost strategy at the expense of higher overhead, i.e., a higher strategy formulation delay.

For demonstration purposes it is assumed that the heuristic QPS formulation algorithms to be examined are

NR_HEU_DV with $N=1$;

NR_HEU_DV with $N=2$;

.....; and

NR_HEU_DV with $N=7$.

For each value of N , $N = 1, 2, \dots, 7$, the algorithm NR_HEU_DV is considered to be a different QPS formulator with its overhead and strategies it produces. The assumption, that the formulators selected for the demonstration and modelling are variations of the algorithm NR_HEU_DV, presented in Chapter 4, is not restrictive, as they could be replaced by some other algorithms proposed in the scientific literature.

NOTATION

Consider a strategy formulated by the algorithm NR_HEU_DV, set to minimize the strategy's delay and using a particular value of N , which is the number of strategy's space levels to be developed. Let

$T_f[i, N]$... denote the time it takes to formulate this strategy to process q_i and

$T_e[i, N]$... denote the cumulative delay for that strategy's execution.

Also let

$\bar{T}_f[N]$... denote the average delay incurred in formulating strategies by the algorithm NR_HEU_DV, using the value of N appearing in $\bar{T}_f[N]$, to process queries q_i , $i=1,2,\dots,N_q$:

$$\bar{T}_f[N] = \frac{1}{N_q} \sum_{i=1}^{N_q} T_f[i,N];$$

$\bar{T}_e[N]$... denote the average delay incurred by the execution of strategies, formulated by the algorithm NR_HEU_DV with the value of N, to process queries q_i , $i=1,2,\dots,N_q$:

$$\bar{T}_e[N] = \frac{1}{N_q} \sum_{i=1}^{N_q} T_e[i,N];$$

$T_t[i,N]$... denote the total query response time for the query q_i . This is the total delay, due to the strategy formulation and execution, such that the strategy is formulated by the algorithm NR_HEU_DV, using the value of N, to process the query q_i :

$$T_t[i,N] = T_f[i,N] + T_e[i,N] \quad \text{and}$$

$\bar{T}_t[N]$... denote the average query response time, over all queries q_i , $i=1,2,\dots,N_q$, such that each query q_i is processed by a strategy formulated by NR_HEU_DV, using the value of N appearing in $\bar{T}_t[N]$:

$$\bar{T}_t[N] = \frac{1}{N_q} \sum_{i=1}^{N_q} T_t[i,N] = \bar{T}_f[N] + \bar{T}_e[N]$$

QUERY RESPONSE TIME -- DELAY DUE TO STRATEGY'S FORMULATION AND EXECUTION

Figure 6-1 shows the average execution delay, $\bar{T}_e[N]$, for strategies formulated by the algorithm NR_HEU_DV, with values of $N = 1, 2, \dots, 7$, in a graphical form and also in the form of a table. Recall from Chapter 4 that when N is sufficiently large, the algorithm produces optimal strategies and, in fact, it becomes equivalent to the algorithm OPT_DV.

The modelled queries are such that with $N \geq 7$, the algorithm NR_HEU_DV finds optimal strategies:

$$\bar{T}_e[N \geq 7] = \bar{c}[g, g] = \frac{\sum_{i=1}^N g[\hat{A}_{i|g}]}{N_q},$$

where $g = \bar{T}_a$. It is the average delay of strategies, over all queries q_i , $i=1, 2, \dots, N_q$, such that each strategy $\hat{A}_{i|g}$ is minimal with respect to the delay due to both CPU processing and data transfer.

The queries of the modelled application were chosen to give a representative range for the number of operations (joins and unions) which are necessary for their processing. From the $N_q=21$ modelled queries,

- 3 queries require 2 operations;
- 6 queries require 3 operations;
- 5 queries require 4 operations;
- 3 queries require 5 operations;
- 2 queries require 6 operations; and
- 2 queries require 7 operations.

N=1	N=2	N=3	N=4	N=5	N=6	N=7
24.79	23.76	21.20	20.96	20.36	20.36	20.36

Average execution delay, $\bar{t}_0[N]$ in seconds, for strategies formulated by NR_HEU_DV with N=1, 2, 3, 4, 5, 6 and 7

Figure 6-1 Average execution delay of strategies formulated by the algorithm NR_HEU_DV

In the author's opinion, the above selection of queries is representative for a wide variety of applications with respect to the number of operations they require. One would expect that, for a general application, most queries would require three to five operations (joins and unions) for their processing, while the number of queries, which require two, six or seven operations, would be small. There would be a few queries which require only one operation (join and/or union) or more than seven operations; no modelled query falls into this category.

As no modelled query requires more than seven operations, the delays shown in column "N=7" of Figure 6-1 are those of optimal strategies. In fact, for the modelled queries NR_HEU_DV finds optimal strategies if it uses $N \geq 5$. However, there may be a query which was not modelled such that it requires six or more operations and for which NR_HEU_DV with $N=5$ would not find the optimal delay strategy.

It is evident from Figure 6-1 that, as expected, an increase in N results in strategies of lower execution delay. It may, therefore, be tempting to choose that formulator which produces the minimum execution delay strategies, i.e., NR_HEU_DV with a large value of N which makes it produces optimal strategies and become equivalent to the algorithm OPT_DV. As stated before, as none of the queries require more than seven operations, $N \geq 7$ is sufficiently large to ensure that NR_HEU_DV produces optimal delay strategies.

Finding optimal strategies is appropriate in case a QPS is prepared and stored in a library and whenever the query is issued by a user, the strategy is retrieved and executed. Since the strategy has been prepared and stored prior to the query's "invocation", the delay incurred by its formulation does not contribute to the query response time. Moreover, the strategy is formulated only once while it may be executed many times, each time the query is issued by a user.

There are two drawbacks to this approach. First, queries must be known in advance; "ad-hoc" queries are excluded. Secondly, a query may refer only to relations whose cardinalities and attribute distributions do not vary over time. In other words, relations' characteristics used in estimation of the size of partial results may not vary over time; for if this is not the case, the formulated strategy will not be optimal for each of its execution. For the two reasons stated above it is assumed that each time a query is issued, a strategy for its processing is formulated.

Figure 6-2 shows the average delay incurred in formulation of strategies, $\bar{T}_f[N]$. This is the average delay incurred by the algorithm NR_HEU_DV, using $N = 1, 2, \dots, 7$, in formulating strategies of which average delay is shown in Figure 6-1. Clearly, it is unreasonable to accept an average delay of about 26 seconds in formulating strategies of which average execution delay is about 20 seconds. These values are shown in figures 6-2 and 6-1, respectively, for $N=7$.

Increasing the value of N does decrease the execution delay of strategies formulated by NR_HEU_DV (see Figure 6-1); however, this decrease is at the expense of increased delay in their formulation.

The application of the modelled DDB assumes information processors comparable to the VAX family of computers (see section 2 of Appendix E). Since the formulation of strategies and modelling was performed using a VAX-780 computer system, it may be assumed that if the algorithm NR_HEU_DV was included as a QSP formulator for the modelled application of a DDB, it would incur formulation delays which are comparable to those reported in Figure 6-2. This enables estimation of the total query response time which includes delays due to both formulation and execution of strategies.

N=1	N=2	N=3	N=4	N=5	N=6	N=7
0.19	0.35	0.79	2.01	5.81	13.59	23.56

Average delay, $\bar{T}_p[N]$ in seconds, for formulation of strategies
by the algorithm NR_HEU_DV with N=1, 2, 3, 4, 5, 6 and 7

Figure 6-2. Average delay to formulate strategies by the algorithm NR_HEU_DV

Figure 6-3 shows the total strategy formulation and execution delay, $\bar{T}_t[N]$, for $N = 1, 2, \dots, 7$. It indicates that NR_HEU_DV should be used with a value of $N=3$. If $\hat{N}$ is that value of N for which

$$\bar{T}_t[\hat{N}] = \min_{\{N\}} \{ \bar{T}_t[N], N=1, 2, \dots, 7 \}, \text{ then}$$

$$\bar{T}_t[\hat{N}] = \bar{T}_t[N=3] = \bar{T}_f[N=3] + \bar{T}_e[N=3] = 0.79 + 21.2 + 21.99 \text{ sec. (6-1)}$$

Using NR_HEU_DV with $N=3$, results in the minimum query response time, $\bar{T}_t[\hat{N}] = \bar{T}_t[N=3] = 21.99$ seconds, from all $\bar{T}_t[N]$, $N=1, 2, \dots, 7$. On the average, about 4 percent of the query response time is spent in the formulation of strategies while 96 percent of the time is spent in the strategies' execution. This query response time is only 1.08 times higher than the execution delay of optimal strategies, i.e., $\bar{T}_e[\hat{N}] / \bar{T}_e[N=7] = 1.08$.

Note that the formulation delay, reported in Figure 6-2, includes delays due to CPU processing and access to the secondary storage devices, but it excludes the delay due to other programs running on the VAX-780, with the VMS the time-sharing operating system, utilized for modelling. The parameters for information processors of the modelled application of a DDB include a "slow-down" factor of 1.5 due to the co-existence of other applications or processing of queries (see section 2 of Appendix E). To be prudent, the average formulation delay, reported in Figure 6-2, should also have been increased by this slow-down factor of 1.5. Even if this was the case, however, $N=3$ would still have been the optimal value for use with the heuristic algorithm NR_HEU_DV.

N=1	N=2	N=3	N=4	N=5	N=6	N=7
24.98	24.11	21.99	22.97	26.17	33.95	43.92

Average query response time, $\bar{T}_t[N] = \bar{T}_f[N] + \bar{T}_e[N]$

in seconds, due to formulation and execution delay of strategies produced by the algorithm SR_HEU_DV with N=1, 2, 3, 4, 5, 6 and 7

Figure 6-3 Average query response, delay due to formulation and execution, of strategies formulated by the algorithm SR_HEU_DV

It has been assumed thus far that whenever any one of the N_q queries is issued, the algorithm NR_HEU_DV is used to formulate a strategy for its processing such that the same value of N is used for all queries. This is not the best policy, as different values of N are best for use, by NR_HEU_DV, for different queries. Let

$\hat{T}_t[i]$... denote the optimal response time, for query q_i , which can be achieved using the algorithm NR_HEU_DV. It is the minimal delay, due to both formulation and execution, for strategies formulated with $N=1,2,\dots,7$, i.e.,

$$\hat{T}_t[i] = \min_{\{N\}} \{ T_t[i,N], N=1,2,\dots,7 \}, \text{ and let}$$

$\hat{T}_f[i], \hat{T}_e[i]$... denote the respective formulation and execution components of the total delay $\hat{T}_t[i]$. Obviously,

$$\hat{T}_t[i] = \hat{T}_f[i] + \hat{T}_e[i].$$

Also let

$\hat{T}_t[N^*]$... denote the average query response time such that each query q_i is processed by a strategy formulated by NR_HEU_DV with that value of N which leads to $\hat{T}_t[i]$, i.e.,

$$\hat{T}_t[N^*] = \frac{\sum_{i=1}^{N_q} \hat{T}_t[i]}{N_q} \text{ and}$$

$\hat{T}_f[N^*], \hat{T}_e[N^*]$... denote the average delays spent in formulation and execution, respectively:

$$\hat{T}_t[N^*] = \hat{T}_f[N^*] + \hat{T}_e[N^*], \text{ where}$$

$$\hat{T}_f[N^*] = \frac{\sum_{i=1}^{N_q} \hat{T}_f[i]}{N_q} \quad \text{and}$$

$$\hat{T}_e[N^*] = \frac{\sum_{i=1}^{N_q} \hat{T}_e[i]}{N_q}$$

For the modelled application,

$$\hat{T}_t[N^*] = 21.52 \text{ seconds}; \quad (6-2)$$

$$\hat{T}_f[N^*] = 0.53 \text{ seconds; and}$$

$$\hat{T}_e[N^*] = 20.99 \text{ seconds.}$$

The above average query response time, $\hat{T}_t[N^*] = 21.52$ seconds, is the best that could be possibly achieved with the algorithm NR_HEU_DV for the modelled application of the DDB. On the average, 2.5 percent of the query response time is spent in formulating strategies while 97.5 percent is spent in the strategies' execution.

Achieving this query response time (equation 6-2), however, is not possible. The problem is due to the assumption that the optimal value of N to be used by the algorithm in formulating a strategy for a given query is known before its execution commences. In reality, which one of the values, from $N=1, 2, \dots, 7$, should be used by NR_HEU_DV for a particular query is not known. Consequently, it appears that the algorithm must be used with the same value of N for all queries.

It has already been shown, however, that algorithm NR_HEU_DV, used with $N=3$ for all queries, leads to the average query response time of

$$\hat{T}_t[N] = \hat{T}_t[N=3] = 21.99 \text{ seconds (see equation 6-1),}$$

which is only 1.02 times higher than the best possible response time which can be achieved by NR_HEU_DV, i.e., it is only 1.02 times higher than

$$\hat{T}_t[N^*] = 21.52 \text{ seconds (see equation 6-2).}$$

The modelling results indicate that a properly chosen value of N , used by NR_HEU_DV for all queries, results in the average query response time, $\bar{T}_t[N]$, which is close to the minimal response time which can be achieved by NR_HEU_DV, $\hat{T}_t[N^*]$. In fact, it is close to the average execution delay, $\bar{c}[T_a, T_a]$, of optimal execution delay strategies.

It has been demonstrated in this section that optimal strategies are a useful tool in evaluation of QPS formulators. The execution delay of optimal strategies can be compared to that of strategies which are produced by heuristic formulators. This comparison shows how much worse is the delay of heuristic strategies in comparison to that of optimal strategies. In addition, a comparison of execution delays, of strategies produced by heuristic algorithms, is insufficient; the overhead/delay incurred in the strategies' formulation must also be considered.

It has been shown that the execution delay of strategies produced by the heuristic algorithm NR_HEU_DV is comparable to those of optimal strategies and that the algorithm has a desirable property in that, by a manipulation of its parameter N , a trade-off, between the formulation and execution of delays, can be made.

FORMULATION DELAY AND A NUMBER OF OPERATIONS

The $N_q = 21$ modelled queries are such that

- 3 require 2 operations;
- 6 require 3 operations;
- 5 require 4 operations;

- 3 require 5 operations;
- 2 require 6 operations; and
- 2 require 7 operations

for their processing, where operations include joins and unions but exclude restrictions and projections. It is instructive to examine the average formulation and strategy execution delays as a function of the number of operations.

Recall that

q_i ... denotes one of the $N_q = 21$ modelled queries; $i = 1, 2, \dots, N_q$.

Let

Q_k ... denote the set of indices for those queries which require k operations for their execution and

m_k ... denote the number of those queries which require k operations for their execution. For the modelled queries, $m_2=3$, $m_3=6$, $m_4=5$, $m_5=3$, $m_6=2$ and $m_7=2$ and $|Q_k| = m_k$, where $|Q_k|$ is the cardinality of the set Q_k .

Also let

$\bar{T}_f[k]$... denote the average delay in formulating strategies, by the algorithm NR_HEU_DV using $N=k$, for the m_k queries which require k operations for their processing:

$$\bar{T}_f[k] = \sum_{i \in Q_k} T_f[i, N=k] / m_k,$$

where $T_f[i, N=k]$ is the delay incurred by the algorithm NR_HEU_DV using the value of $N=k$ in formulating a strategy for the query q_i ;

$\bar{T}_e'[k]$... denote the average execution delay of strategies, formulated by the algorithm NR_HEU_DV using $N=k$, for the m_k queries which require k operations:

$$\bar{T}_e'[k] = \sum_{i \in Q_k} T_e[i, N=k] / m_k,$$

where $T_e[i, N=k]$ is the execution delay of a strategy, formulated by the algorithm NR_HEU_DV using the value of $N=k$, for the query q_i .

Note that as queries q_i , $i \in Q_k$, require k operations, for their processing, NR_HEU_DV, with $N=k$, formulates their optimal strategies. Consequently, $\bar{T}_f'[k]$ is the average delay incurred by NR_HEU_DV in formulating optimal strategies for queries q_i , $i \in Q_k$. Similarly, $\bar{T}_e'[k]$ is the average execution delay of optimal strategies used to process queries q_i , $i \in Q_k$.

The average formulation delay, $\bar{T}_f'[k]$, is shown as a function of the number of operations, k , in Figure 6-4(a) and it is also shown using the semi-log scale in Figure 6-4(b). It appears to have an exponential shape and thus supporting the conjecture that formulation of a QPS is NP-hard and that the search for optimal strategies in an operational environment is not desirable.

The average strategy execution delay, $\bar{T}_e'[k]$, is shown as a function of the number of operations, k , in Figure 6-5. There appears to be no dependency, of the strategy execution delay, on the number of required operations. This is to be expected as a query may refer to "large sized" relations and its processing, although requiring only a small number of operations, may incur a high delay. Conversely, a query which refers only to "small sized" relations may incur a low

k=2	k=3	k=4	k=5	k=6	k=7
0.14	0.41	1.79	11.87	24.16	24.22

Average delay, $\bar{T}_p[k]$, in seconds, for formulation of strategies, by the algorithm NR_HEU_DV with $N=k$, for formulation of strategies for queries q_i , $1 \leq i \leq Q_k$, where k is the number of joins and unions required to process the queries q_i , $1 \leq i \leq Q_k$.

Figure 6-4(a) Formulation delay as a function of the number of operations

Figure 6-4(b) Formulation delay as a function of the number of operations
Semi-Log Scale

$k=2$	$k=3$	$k=4$	$k=5$	$k=6$	$k=7$
21.04	31.46	7.46	6.46	27.27	32.20

Average execution delay, $\bar{t}_0[k]$, in seconds, of strategies, formulated by the algorithm NR_HEU_DV with $N=k$, for queries $q_1, \dots, q_k$, where k is the number of joins and unions required to process the queries $q_1, \dots, q_k$.

Figure 6-5 Average strategy execution delay and number of operations

delay even though it may require many operations (joins and unions) for its processing.

6.6 SUMMARY

The model of a relational DDB and algorithms, proposed in chapters 3 and 4, were used for modelling of an application of a DDB. The modelling was used to demonstrate how optimal strategies can be used in an examination of the cost of strategies with and without optimization;
a selection of an appropriate measure of cost and
an examination of the effects of an algorithm, which minimizes a particular cost measure, on the strategies' cost in other measures.

Delays of strategies produced by algorithms NR_HEU_DV and OPT_DV were examined and it was shown that overhead (delay) due to the formulation of strategies should be taken into consideration when selecting an optimizing algorithm. It was also exhibited that, using the ability of the heuristic algorithm NR_HEU_DV to achieve a trade-off between the delay in formulation and execution of strategies, it is possible to obtain an average query response which is close to the execution delay of optimal strategies.

Finally, it was shown that, for the modelled application of a DDB, the total size of partial results is a good candidate for adoption as a measure of cost and a minimization objective. It confirms that the "two-staged" approach to formulation of strategies, proposed in Chapter 2, should lead to strategies of which delay is close to that of strategies which yield the minimum delay.

It should be stressed that the reported modelling results and their discussion apply to the particular application of a DDB which was described in Chapter 5. The application was chosen to be representative of general applications in that (i) the information processors incur delays and costs which correspond to the VAX family of computers and (ii) the delay and cost of transferring data between two information processors are comparable to those of a public data network such as DATAPAC. In addition, the queries were also chosen to be representative with respect to the number of operations (joins and/or unions) which they require for their processing.

In spite of this, in the author's opinion, representative selection of the modelled application, there is no claim made that the results of modelling apply to applications of a DDB in general. Modelling of an application was used for two reasons. First, it was used as a vehicle to demonstrate that in conjunction with optimal strategies it is a useful tool in addressing many issues which arise in a distributed query processing. Secondly, information (tables 6-1 to 6-6) needed to address these issues, and how it should be used were identified in this chapter; modelling of a particular application enabled the presentation of this material with some substance.

An alternative to modelling of an application of a DDB is to randomly select queries such as in Look (1982). The difficulty with this approach is that not only does the number of a query two-variable terms varies randomly, but so do the number of relations each query references, the cardinalities and attribute field widths of each relation and also the parameters associated with the selectivity functions used in the size estimation of partial results. Consequently, the number of randomly varying variables is large and what distributions, for random variables, should be used in order to simulate a "general" DDB application is yet to be addressed.

CHAPTER 7

SUMMARY AND SUGGESTIONS FOR FURTHER RESEARCH

7.1 THESIS CONTRIBUTIONS

The major contributions of this research are as follows:

1. QPS FORMULATION -- COMPLEXITY: The proof that finding optimal locations for executions of joins in a given sequence is a problem which is polynomial.
2. TOTAL SIZE OF PARTIAL RESULTS: Showing that the size of partial results is a good alternative to other measures of cost in optimizing distributed queries.
3. OPTIMAL STRATEGIES: The development of an algorithm which finds optimal strategies for the distributed view of a DB.
4. HEURISTIC QPS FORMULATOR: The development of a heuristic algorithm to formulate strategies for queries defined on the distributed view of a DB.
5. MODELLING OF DISTRIBUTED QUERY PROCESSING: The utilization of a relational DDB model and optimal strategies in a method for (i) choosing an appropriate measure of cost for distributed query optimization and (ii) evaluation of QPS formulators.

1. QPS FORMULATION -- COMPLEXITY

The problem of determination of optimal network locations for execution of joins in a given sequence was proved to be a polynomial problem by finding a polynomial-time algorithm for its solution.

2. TOTAL SIZE OF PARTIAL RESULTS

The result, listed above, implies that the likely NP-hardness of processing queries in a DDB by joins arises due to the freedom of choice in the sequence in which joins can be executed and not due to the nature of the distributed environment. Consequently, queries can be processed in two phases: In the first phase, the sequence of relational operations, which minimizes the total size of partial results, is determined. In the second phase, the locations for the relational operations' execution is found. This "two-phased" approach, however, may not lead to strategies which minimize their execution delay.

Modelling of query processing for an application of a DDB indicates that the use of the total size of partial results, as a minimization objective in a QPS formulation, leads to strategies with delays which are close to minimal delays. The modelled DDB application is such that the cost measured in terms of a network data transfer does not dominate that in terms of CPU processing and vice-versa. Consequently, the use of the total size of the partial results as a measure of cost and the "two-phased" approach to the formulation of strategies are viable and promising alternatives for adoption particularly in a distributed environment in which the network data transfer cost/delay does not clearly dominate the cost/delay due to CPU

processing and vice-versa.

3. OPTIMAL STRATEGIES

Algorithm OPT_DV was developed for formulation of optimal strategies for processing of distributed view queries. Under the adopted assumptions, the algorithm was shown to be non-redundant, i.e., it does not consider any strategies/sub-strategies redundantly, and complete in that it searches through the space of all strategies available for a query processing. Because of this search through the complete space of strategies, it incurs high overhead/delay in the formulation of strategies. It is, therefore, not intended for use as an operational QPS formulator but, instead, as a tool to be utilized in the distributed query processing modelling for evaluation of (i) cost measures and (ii) heuristic QPS formulators. For this reason, the algorithm is capable of finding optimal strategies for various measures of cost.

4. HEURISTIC QPS FORMULATOR

An efficient heuristic algorithm, algorithm NR_HEU_DV, was developed for formulation of strategies for distributed view queries. It has a desirable property in that, by manipulation of its parameter (parameter "N"), it is also possible to manipulate its overhead/delay in-formulation of strategies. With an increase in its overhead the algorithm produces strategies of lower execution delay. Using this algorithm it is possible to attain a trade-off between overhead/delay in the formulation of strategies and execution delay of those strategies.

5. MODELLING OF DISTRIBUTED QUERY PROCESSING

Algorithm OPT_DV was developed to find optimal strategies in order to enable evaluation of cost measures and heuristic strategy formulators. This algorithm and also the heuristic formulator, algorithm NR_HEU_DV, were integrated within the framework of a model of a relational DDB. Information needed for the evaluation of cost measures and heuristic formulators was defined and its use was demonstrated through modelling of queries for an application of a DDB.

7.2 SUGGESTIONS FOR FURTHER RESEARCH

Although this research has made some progress in solving problems arising in distributed query processing and much research has been published on this topic, there are still many problems to be solved:

1. There appear to be two approaches in research on this topic. In one approach, semi-joins are utilized as a tactic in optimization of queries while the other utilizes joins as a processing tactic. As has been stated before, these two approaches need to be integrated. The first problem to be addressed in their integration is which one of these tactics should be used first. One alternative is first to find an optimal strategy which uses joins as a processing tactic and improve upon it by utilization of semi-joins. The second alternative is first to reduce relations by utilization of semi-joins and then consider optimization of their processing by joins. Whether one alternative leads to lower cost strategies than the other is an open question.
2. This research suggests that formulation of strategies, which use joins as a processing tactic, for processing distributed view queries can be performed by a previously discussed "two-phased"

approach; algorithms utilizing this approach need to be developed.

3. Formulation of QPSs relies heavily on size estimation of results of relational operations. Although some progress has been done on this topic, further research is needed.
4. Presently, there are no methods available for correcting strategies which are developed through use of incorrect size estimation. Since a strategy is formulated on the basis of a priori size and cost estimates, the actual cost of its execution may be significantly different from its estimate and some corrective action should be taken. The strategy execution should be monitored and if the actual cost differs from its estimate the unexecuted portion of the strategy should be reformulated.
5. Algorithms OPT_DV and NR_HEU_DV are used to formulate strategies for queries defined on the distributed view of the DB. Relations may be partitioned vertically and/or horizontally. Horizontal fragmentation is assumed to be such that all partitions are assumed to be mutually exclusive. Maier and Ullman (1983) developed a theory for processing horizontally partitioned relations such that fragments of a relation need not be mutually exclusive. Algorithms which optimize queries referencing such relations need to be developed.
6. Very little attention has been paid to the integration of the concurrency control mechanisms and the QPS formulation and monitoring/coordination. How they affect each other requires an in-depth examination.

APPENDIX A

The appendix contains two sections. Section A-1 briefly discusses two categories of technical problems in a DDB (Toth, 1980), Resource Allocation and Concurrency Control. The formal definition of a conjunctive form query and its mapping to the informal representation used herein is given in the section A-2.

A-1. RESOURCE ALLOCATION AND CONCURRENCY CONTROL

The aim of discussing the two categories of technical problems arising in a DDB is to outline the problems and offer reference to some of the literature which can be used as a starting point for research.

RESOURCE ALLOCATION IN NETWORKS

The problem of Resource Allocation (RA) in Distributed Information Systems can be formulated in the following way: Given a set of resources and parameters that identify environment (data items/fragments, links and their capacities, processor capacities, ...) and a set of constraints (query response time, constraints on resource assignments, ...) find the optimal allocation of resources so that a chosen measure of cost, such as volume of data transferred over the network, is optimized.

It is noteworthy that most approaches to the RA problem allow multiple copies of data items to improve availability, query response time or reliability. Some models allow dependencies of access between data items; however, dependencies are at most pairwise.

The above definition of the RA problem is in its fullest form. Most of the mathematical models that have been developed to solve this problem are subsets of this definition, stressing some parameters and constraints while neglecting others ((Cassey, 1972), (Mahmoud, 1975), (Morgan, 1977), (Eswaran, 1974), (Akoka, 1980)). Eswaran (1974) showed that the problem is NP-complete and therefore heuristic approaches are justified. Ramamorthy (1980) established the equivalence of this problem to that of the "Warehouse Location Problem" found in the Operations Research area, therefore, numerous solutions to the "Warehouse Location Problem" can be applied to the RA problem as well.

Riordon (1982) points out that the RA problem, as defined above, may be unsuitable for use in the design of an integrated DDB as the distribution of data elements (relations and their fragments) depends on the CC mechanism and strategies for execution of distributed queries. As further understanding of these two problems is gained, attention will again turn to the problem of distribution of data elements of an integrated DDB.

Indeed, researchers are starting to turn their attention to the problem of distribution of relations and their fragments in integrated relational DDB ((Wong, 1983), (Ceri, 1983)). The problem definition differs from the general RA problem in that a completely connected network is assumed to exist and that the decision on distribution of relations and their fragments is incorporated into the overall distributed database design methodology.

CONCURRENCY CONTROL

A Concurrency Control (CC) mechanism is necessary in a central DBMS because several users may share access to the same data items simultaneously. Matters are further complicated in a DDB where a transaction may access data items in several network nodes. Due to the distribution of data, the CC mechanism must cope with variable message delays, the fact that ordering of messages can be guaranteed between two parties only, and failures of hardware/software.

The CC mechanism must ensure that (Toth, 1980)

- a transaction has a consistent view of the data base;
- mutual consistency of multicopy data is preserved;
- danger of deadlocks is properly handled;
- a transaction eventually completes;

the mechanism is robust in the face of failures at several levels.

To provide the CC, the mechanism may introduce:

- additional messages into the network to provide coordination if a transaction spans more than one node;
- delays due to concurrent access (e.g., a transaction may be locked out when trying to access a data item);
- additional storage requirements to maintain lock tables, queues, and back-ups.

Many authors addressed the problem of CC with the objective of finding a mechanism which functions correctly, which is usually proven by the use of 'serialization' ((Thomas, 1978), (Bernstein, 1979), (Bernstein, 1983a)), while incurring minimum cost - message traffic, delays, and storage requirements. The most prominent methods are:

- (i) Centralized Locking (Menasce, 1979)
- (ii) Majority Consensus Algorithm (Thomas, 1978)
- (iii) Virtual Ring Methods (LeLann, 1977, 1978, 1980)
- (iv) Conflict-driven Restarts ((Rozencrantz, 1978), (Decitre, 1981))
- (v) SDD-1 Synchronization Scheme (Bernstein, 1977, 1979)
- (vi) ADD Synchronization Protocol ((Mahmoud, 1979), (Toth, 1980))

The proposed mechanisms for CC have different complexities and incur different overhead. To enable their comparison, a formal framework of reference is required and the methods themselves should be analysed, if possible, analytically. Research in this direction includes:

Wilms (1980) compares several CC methods on the basis of the number of network messages, update response time and resiliency to failure;

a. Query graph for $R_1.a \oplus R_2.b$, $R_2.c \oplus R_3.d$, $R_3.e \oplus R_4.f$.

b. Coalesced graph representing processing of $R_1.a \oplus R_2.b$.

c. Coalesced graph representing processing of $R_3.e \oplus R_4.f$.

network
node

(d) Strategy timing diagram

Figure C-4 Query, its graph, and a timing diagram for one of its strategies

In summary, after the initial identification of problems associated with the concurrency control, many CC mechanisms were proposed. Currently, an effort is being made to quantitatively analyse the proposed mechanisms and to develop formal models for their comparison. However, due to the complexity of proposed mechanisms and variations in assumptions and approaches, this is proving to be difficult.

A-2. CONJUNCTIVE FORM QUERY

Chapter 4 presents an informal representation of a conjunctive form query which is generally used in the scientific literature dealing with optimization of distributed query processing and which is also assumed for this dissertation. This section defines mapping of this informal representation to the formal definition of a conjunctive normal form query. The formal definition of a tuple calculus expression and the conjunctive form is adopted from Date, 1982, and Wilman, 1982.

A query can be defined as a tuple calculus expression which is the primary construct of the tuple-oriented relational calculus. A tuple calculus expression is formed from Tuple Variables, Conditions and Well-formed Formulas which are now defined.

TUPLE VARIABLE: Tuple variables are represented by $S, T, U, \dots$ and each tuple variable is constrained to range over some named relation or a union of relations. If T represents a tuple t then " $T.a$ " represents the " a " attribute value of t , where " a " is an attribute of the relation over which T ranges.

CONDITION: A condition has a form $x \theta y$, where θ is a relational arithmetic operator from a set of operators $=, <, >, \leq, \geq$ and at least one of x and y , or both, are expressions of the form $T.a$, where T is a tuple variable and a is an attribute of the relation over which T ranges. One of x and y may be a constant.

WELL-FORMED FORMULA (WFF): A WFF is constructed from conditions, Boolean operators AND, OR, and NOT, and quantifiers " $\exists$ " (there exists) and " $\forall$ " (for all), according to the following rules:

1. Every condition is a WFF.
2. If f is a WFF, then so are f and NOT f .

C A-8

3. If f and g are WFFs, then so are $f \text{ AND } g$ and $f \text{ OR } g$.
4. If f is a WFF in which T occurs as a free variable, then $[T/f]$ and $\forall T(f)$ are WFFs.
5. Nothing else is a WFF.

Each occurrence of a tuple variable is free or bound according to the following rules:

1. Occurrences of tuple variables within a condition are free.
2. Occurrences of tuple variables in the WFFs f and $\text{NOT } f$ are free/bound according as they are free/bound in f .
3. Occurrences of tuple variables within the WFFs $f \text{ AND } g$ and $f \text{ OR } g$ are free/bound according as they are bound in f or g .
4. Occurrences of a tuple variable T which are free in f are bound in $[T/f]$ and $\forall T(f)$.

A tuple calculus expression is an expression of the form

$S.a.T.b. \dots U.c \text{ WHERE } f.$

where $S, T, \dots, U$ are tuple variables; $a, b, \dots, c$ are attributes of the associated relations; and f is a WFF containing exactly $S, T, \dots, U$ as free variables. " $S.a.T.b. \dots U.c$ " is called a target list.

A prenex normal form tuple calculus expression is such that all quantifiers appear at the front of the WFF.

conjunctive normal form tuple calculus expression is a prenex normal form restricted to the "there exists" existential quantifier, i.e., the "for all" universal quantifier is not allowed. In addition, the expression is of the form

$S.a.T.b. \dots U.c \text{ WHERE } \exists U_1 \dots \exists U_n \forall V_1 \dots \forall V_m f.$

where $U_1, U_2, \dots, U_m$ are tuple variables and $V_1, V_2, \dots, V_n$ are one or two-variable terms, i.e., they are conditions.

The representation of a conjunctive normal form query used herein is derived directly from the above definition. The target list and existential quantifiers are omitted and any tuple variable occurrence appearing in the qualification term is replaced by the name of the relation over which it ranges. For example, consider a conjunctive normal form query specified on relations R_1, R_2 and R_3 ,

$$S.a \text{ where } (\exists T) (\exists U) ((S.a = T.b) \wedge (T.c > U.d)),$$

where

S, T and U are tuple variables ranging over relations R_1, R_2 and R_3 respectively;

a is an attribute of R_1 ;

b and c are attributes of R_2 ; and

d is an attribute of R_3 .

The informal representation of the above query becomes

$$(R_1.a = R_2.b) \wedge (R_2.c > R_3.d).$$

Note that if two (or more) distinct tuple variables, such as S and T , range over the same relation R then, in the informal representation, S is replaced by R_1 and T by R_2 , where R_1 and R_2 are assumed to be two distinct relations with an identical content as R . (In a distributed environment, obviously, R_1 and R_2 are assumed to be stored by the same information processor.

The informal representation of a conjunctive normal form query is used in the scientific literature in order to simplify the presentation of material. It avoids repetitious identification of which tuple

variable ranges over which relation and specification of the target list which, most of the time, is unimportant when discussing the material on distributed query processing.

APPENDIX B

SEMI-JOINS AS A TACTIC IN DISTRIBUTED QUERY PROCESSING

Because the use of semi-joins received a great deal of attention, the basic concept of their use is discussed in detail. The discussion will also reveal the reason why semi-joins are particularly suitable for the minimization of the volume of transmitted data.

A strategy which uses semi-joins consists of two phases:

- (i) After initial processing of relations by restrictions and projections, semi-joins are used to reduce the size of relations to be processed by joins.
- (ii) After the size reduction by semi-joins, relations are moved to one site (the destination site) where they are processed by joins to derive the query answer. The sequence of semi-joins is also referred to as a semi-join program.

Assume that

$R[a]$... denotes a projection of attribute "a" on relation R, i.e., all other attributes of R are projected out;

$R_1[a=b]R_2$... denotes an equi-join of R_1 with R_2 on joining attributes a and b of R_1 and R_2 , respectively;

$\text{Card}(R)$... is the cardinality of relation R;

$\text{Width}(R)$... is the size (in bytes) of tuples of R;

$\text{Width}(R[a])$... is the size (in bytes) of the field/attribute R.a;

$\text{Size}(R) = \text{Card}(R) \cdot \text{Width}(R)$... is the size of the relation R; and

$\text{Size}(R[a]) = \text{Card}(R[a]) \cdot \text{Width}(R[a])$... is the size of attribute R.a.

Consider a two-variable term " $R_1.a = R_2.b$ " such that relations R_1 and R_2 are located in two different network nodes W and Z, respectively, and that the result is to be delivered to node Z. The

term can be processed by transmitting both R_i and R_j to the network node Z where they are joined to give a relation R_k . The cost to derive R_k , c_k , expressed in terms of the total volume of data transferred over the network is

$$c_k = \text{Size}(R_i) + \text{Size}(R_j).$$

R_k can also be derived by utilization of a semi-join. The semi-join of R_i by R_j on term $(R_i.a = R_j.b)$, denoted by $R_i \lt a=b \gt R_j$, results in a transmission of $R_j[b]$ to the network node W where it is joined with R_i to give R_1 which is then shipped to node Z. As $\text{Card}(R_1) < \text{Card}(R_i)$, the size of R_i which is to be transferred to Z is reduced. A semi-join is said to be profitable if its benefit out-weighs its cost, i.e., when $\{\text{Size}(R_i) - \text{Size}(R_1)\} > \text{Size}(R_j[b])$.

Similarly, the relation R_j can be reduced by a semi-join of R_j by R_i to give a relation R_2 ,

$$R_2 = R_j \lt b=a \gt R_i = R_j[b=a](R_i[a]),$$

by transmitting $R_i[a]$ to Y and executing the join $R_j[b=a](R_i[a])$. The resulting relation R_2 is then transmitted to the network node Z where it is joined with R_i to give R_k . The strategy which includes both semi-joins has a cost

$$c_k = \text{Size}(R_i[a]) + \text{Size}(R_j[b]) + \text{Size}(R_1) + \text{Size}(R_2).$$

All proposed strategies which utilize semi-joins contain the two phases listed above. Consequently they utilize semi-joins to decrease the network traffic at the expense of CPU processing: Relations reduced by semi-joins are all assembled at the destination node where

they are processed by joins.

It is important to note that since cost is assumed to be incurred only due to a data transfer, the two semi-joins in the above example should be executed in sequence. In other words, the semi-join of R_j by R_i should be replaced by a semi-join of R_j by R_i in order to gain further size reduction on R_j . Essentially, two semi-joins are then used to decrease the size of R_j .

The optimization of a semi-join program determines the sequence of semi-joins for each relation referenced by the query such that the volume of data transferred over the network is minimized.

APPENDIX C

The appendix presents application of two techniques to finding the optimal strategy for a given query. The first technique, included in the section C-1, is a variation of the "backtracking" approach (Horowitz, 1978), (Aho, 1974), (Reingold, 1977). The algorithm using this technique will be shown to examine many strategies sub-strategies redundantly.

The second technique utilizes the "divide and conquer" approach of dynamic programming. An algorithm based on this approach and a discussion of the reason why it is considered to be unsatisfactory is included in the section C-2.

2-1. DEPTH-FIRST SEARCH

A depth-first search is a variation of a backtrack algorithm in which the choices under examination can be represented by a decision tree. The edges of a decision tree represent choices under examination while its nodes are used as states to keep track of choices made thus far. The Depth-first Search, as its name implies, proceeds to develop a tree toward the leaves which represent solutions while retaining only one path from the root to a leaf at any one time. Once a solution is found, the algorithm backtracks one level up and again proceeds toward the leaves to examine the remaining options. Backtracking from any node occurs only if all available options from the state represented by the node have been exhausted.

For the problem of QPS formulation the choices/options under consideration are: i. "Which of the remaining two-variable terms should be processed by a join?", and ii. "In which network location should the join be executed?". For example, let a simple query with its graph as shown in Figure C-1a be

$$R_1.a \Theta R_2.b \wedge R_2.c \Theta_2 R_3.d \wedge R_3.e \Theta_3 R_4.f,$$

where relations R_1 , R_2 , R_3 , and R_4 are located in network locations W, X, Y, and Z respectively. The choice of the first join is made from

i. join R_1 with R_2 in the network location W;

ii. " R_1 " R_2 " " X;

iii. " R_2 " R_3 " " X;

iv. " R_2 " R_3 " " Y;

v. " R_3 " R_4 " " Y; and

vi. " R_3 " R_4 " " Z.

Every Graph

First Strategy

Back-track and Forward

Back-track Twice

Level

Level 2

Level 3

Figure 10. Example of the depth-first search

Recall that a relational operation can be executed only in one of the locations of its operands. The join of R_1 with R_2 can therefore be executed only in one of the locations where R_1 or R_2 reside. Since there are three two-variable terms in the original query and they reference four relations located in different network nodes, there are six choices for the selection of the first join and its execution location.

Recall that the relation's right sub-script is a unique identifier. For an original relation referenced by the query it is a single digit. For a partial result the unique right sub-script has two or more digits. The first digit identifies the number of join executions used to derive the partial result. In addition, if it is desirable to show a relation's network location, the left sub-script is used for that purpose. For example, a relation W_{R_3} is located in the network location W and it is a result of three join executions as the first digit of its right sub-script is "3".

Once a choice is made to examine case (1), further choices are selected for examination while developing a path from the root to a leaf as shown in Figure C-1(b). The path shown represents the following strategy:

Join R_1 with R_2 in W to give R_{11} ;
 Join R_3 with R_4 in Y to give R_{12} ; and
 Join R_{11} with R_{12} in W to give R_3 .

If the destination node to which the query result is to be delivered is not the node W then the strategy is augmented by the final data transfer of the relation R_3 to the destination node and the cost of the strategy is increased by the cost of this data transfer. If the strategy cost is less than that of the current minimum cost strategy, it is saved and declared to be the current minimum cost strategy. At

this point backtracking occurs one level up and the next strategy examined is the one shown in Figure C-1(c). Backtracking occurs one level up again, but, since all choices for the tree node "3" were examined already, there is an additional backtrack. The next tree node created represents the result of a join of R_3 with R_4 in the network location Z (see tree node "6" of Figure C-1(d)); its sub-tree is examined next and so on.

Each node "u" of the decision tree must contain information about which two-variable terms are yet to be examined and parameters about relations, which are necessary to estimate the cost and size of partial results. This information is represented by sets $\{V''\}^u$ and $\{(x,c,I)_i\}^u$:

$\{V\}^u$ is the set of two-variable terms to be examined from the tree node u.

$\{(x,c,I)_i\}^u$ is a set of triplets conveying information about relations referenced by the two-variable terms in $\{V''\}^u$. For each distinct relation R_i , $i=1,2,\dots,m$, referenced by two-variable terms in $\{V''\}^u$, there is a triplet $(x,c,I)_i \in \{(X,c,I)_i\}^x$, such that

- (i) x_i is the network location of R_i ;
- (ii) c_i is the cumulative cost of all processing and data transfer necessary to derive R_i ; and
- (iii) I_i is information about relation R_i which is necessary for the size and cost estimation of join processing.

For the root of the decision tree, information about relations and two-variable terms to be examined is derived directly from the query. Consider node 2 of the decision tree of Figure C-1(b), which represents a join of R_1 with R_2 in order to process the two-variable term

$R_1.a \odot R_2.b$. If the query is modified to represent this processing, it becomes

$$R_1.c \odot_2 R_3.d \wedge R_3.e \odot_3 R_4.f.$$

Consequently, $\{V'\}^2$ and $\{x.c, y.d\}^2$ for the tree node 2 are derived from information of node 1, the root, in the following way:

1. The set of two-variable terms to be examined for the tree node 2, $\{V'\}^2$, is derived from the set of the two-variable terms for the node 1, $\{V'\}^1$, by removing the chosen two-variable term, which, in this case, is $V'\{1\}$. In other words,

$$\{V'\}^2 = \{V'\}^1 - \{V'\{1\}\}$$

where $V'\{1\} = \{R_1.a \odot R_2.b\}$ is the two-variable term processed by a join of R_1 and R_2 . In addition, any reference to R_1 or R_2 in $\{V'\}^2$ is replaced by a reference to R_{11} , the join result. Since the two-variable term under the consideration is $V'\{2\} = \{R_1.a \odot R_2.b\}$ and the join of R_1 with R_2 executed in the network node W is R_{11} , the set of two-variable terms for the tree node 2 becomes

$$\begin{aligned} \{V'\}^2 &= \{(R_1.a \odot R_2.b), (R_2.c \odot_2 R_3.d), (R_3.e \odot_3 R_4.f)\} - \\ &\quad \{(R_1.a \odot R_2.b)\} = \{(R_2.c \odot_2 R_3.d), (R_3.e \odot_3 R_4.f)\} \end{aligned}$$

In addition any reference to R_1 or R_2 is replaced by R_{11} and $\{V'\}^2$ becomes

$$\{V'\}^2 = \{(R_{11}.c \odot_2 R_3.d), (R_3.e \odot_3 R_4.f)\}$$

(11) Since relations R_1 and R_2 are processed by the join, information about them is replaced by information about the join result:

$$\begin{aligned} \{(x, c, I)\}^2 &:= \{(x, c, I)\}^2 - \{(x, c, I)_{11}, (x, c, I)_{12}, \dots, (x, c, I)_{1n_1}\} \\ &\quad - \{(x, c, I)_{21}, (x, c, I)_{22}, \dots, (x, c, I)_{2n_2}\} \\ &\quad - \{(x, c, I)_{11}, (x, c, I)_{12}, \dots, (x, c, I)_{1n_1}\} \\ &\quad - \{(x, c, I)_{21}, (x, c, I)_{22}, \dots, (x, c, I)_{2n_2}\} \\ &\quad - \{(x, c, I)_{11}, (x, c, I)_{12}, \dots, (x, c, I)_{1n_1}\} \\ &\quad - \{(x, c, I)_{21}, (x, c, I)_{22}, \dots, (x, c, I)_{2n_2}\} \end{aligned}$$

$(x, c, I)_{ij}$ is information about the join result, relation R_{ij} , where

x_{ij} is the location of R_{ij} , network node i ;

c_{ij} is the cumulative cost of all processing and data transfer to obtain R_{ij} ; it is derived using x, c, I_{11} and x, c, I_{12} ; and

I_{ij} is information about R_{ij} necessary for the size and cost estimates.

In general, $\{V''\}^V$ and $\{(x, c, I)\}^V$ of the tree node v which represents the processing of $V''_m = (R_1.a \Theta R_2.b)$ by a join of R_1 with R_2 , is derived in the following way: Assuming that the predecessor node of v is u and that the join result is R_k , then

$$(1) \{V''\}^V := \{V''\}^u - \{V''_m\},$$

where $V''_m = (R_1.a \Theta R_2.b)$ and any reference to R_1 or R_2 in V''_m is replaced by a reference to R_k , the join result;

(ii) information about relations R_1 and R_2 is replaced by information about the join result, ie.,

$$\{(x, c, I)\}^V := \{(x, c, I)\}^u - \{(x, c, I)_{11}, (x, c, I)_{12}, \dots, (x, c, I)_{1n_1}\} \cup \{(x, c, I)_{k1}, (x, c, I)_{k2}, \dots, (x, c, I)_{kn_k}\}.$$

The left-to-right search algorithm is presented in both its recursive and non-recursive forms in Figure 1.

If a query has n two-variable terms referencing n relations located in different network nodes, then the first tree level contains n nodes which represent options for the selection of the first two-variable term to be processed and the location of its processing. In the above example, the query contains three two-variable terms and, therefore, there are six nodes in the first level of the decision tree. For each node in the first level, there will be $n-1$ nodes in the second level of the decision tree: each node represents a choice for processing one of the remaining $n-1$ terms by a join in one of the two network locations where join operands relations reside. Since there are $n-1$ nodes in the first level, the second level contains $2(n-1)$ nodes, etc. The number of nodes in each of the tree levels is:

LEVEL	NUMBER OF NODES
1	$2n$
2	$2n \cdot 2(n-1)$
3	$2n \cdot 2(n-1) \cdot 2(n-2)$
...	...
$n-1$	$2n \cdot 2(n-1) \cdot \dots \cdot 2(n-n+2)$
n	$2n \cdot 2(n-1) \cdot \dots \cdot 2(n-n+2) \cdot 2$

The number of nodes in a tree level p is

$$2^p n(n-1) \dots (n-p+1) = 2^p \frac{n!}{(n-p)!}$$

The total number of nodes in a tree created by the algorithm is

$$\sum_{p=1}^n \left(2^p \frac{n!}{(n-p)!} \right)$$

DEPTH-FIRST SEARCH ALGORITHM - NON-RECURSIVE

INPUT: Query Q , and a network node i - destination node for the query result.

The query is defined by $V = \{V_1, V_2, \dots, V_n\}$ and x, c, I where

$V_1, \dots, V_n$ is a set of one-variable terms for query Q .

x, c, I is a set of two-variable terms for query Q and

x, c, I is a set of triplets, one triplet for each of the original relations referenced by the query. A triplet (x, c, I) for a relation R is such that

x is the network location of R .

c is the cumulative cost of R which is zero, and

I is information about R for size and cost estimation.

OUTPUT: QPS - optimal strategy, cQPS - the cost of the optimal strategy QPS

PROCESS: A tree of strategies is developed and examined by a "depth-first" search. Each tree node n represents a state of the search by

V_n - a list of two-variable terms to be examined/processed

x, c, I_n - a set of triplets, one triplet per each distinct relation referenced by the two-variable terms in V_n such that each triplet is defined as above

a pointer to a parent node to enable back-tracking; and

the state of the search for the tree node, i.e., the set of all possible joins for processing of any one of terms in V_n and which of these joins were examined already

STEP 1: Create the tree root directly from the definition of query Q . Let n denote the tree root.

$V_n = V$, $x, c, I_n = x, c, I$

The cost of each relation R_i of x, c, I_n is set to the cost of one-variable processing on R_i , i.e., the cost of restrictions and initial projections.

QPS = "empty", cQPS = "infinity", cQPS is the cost of QPS.

Set "current" node to the tree root just created:

STEP 2: Let V_n and x, c, I_n be those of the current tree node.

IF V_n is empty THEN

BEGIN

* COMMENT - x, c, I_n contains only one triplet "i" which represents the query result, relation R_i , obtained by a particular strategy. *

IF $x \neq i$ THEN *i.e., the result is not in the desired destination node*

$c_i = c_n$ - the cost of transfer of R_i to the destination node;

IF $c_i < cQPS$ THEN

BEGIN

cQPS := c_i .

QPS := strategy represented by a path from the tree root to the "current" node. END

BACK TRACK: * i.e., set "current" node to be the parent node *

GO TO STEP 2.

END

ELSE

BEGIN

IF each possible join has been examined THEN

BEGIN

IF the current node is the tree root THEN

STOP (*QPS is the optimal strategy and cQPS is its cost*)

ELSE

BEGIN

BACK TRACK: * i.e., set "current" to be the parent node *

GO TO STEP 2

END

END

ELSE

BEGIN

Choose next join to be examined; assume that it is a join of R_i with R_j in a particular network node of R_i giving a relation R_k and that it is to process $V'_1 \in V_n$.

Create a new tree node n' with $V_{n'} = V_n - \{V'_1\}$ and $\{(x, c, I)\}^{n'}$ such that

(i) $\{V'_1\}^{n'} := \{V'_1\}^n - \{V'_1\}_i$, and

(ii) $\{(x, c, I)\}^{n'}$ is derived from $\{(x, c, I)\}^n$ by replacement of triplets "i" and "j" for relations R_i and R_j respectively, by a triplet for relation R_k , i.e.,

$\{(x, c, I)\}^{n'} := \{(x, c, I)\}^n - \{(x, c, I)_i, (x, c, I)_j\} \cup \{(x, c, I)_k\}$.

Note that c_k of $(x, c, I)_k \in \{(x, c, I)\}^{n'}$ represents the cumulative cost of R_k and includes the cost of all processing and data transfer to derive R_k .

Make the new node a "current" node.

GO TO STEP 2

END

END;

Figure C-2(a) Non-recursive Depth-first Search algorithm

RECURSIVE PROCEDURE DEPTH-FIRST-SEARCH

INPUT: query Q , and a network node i - destination node for the query result.

The query is defined by V^Q and $\{(x,c,I)^Q\}$, where

V^Q is a set of one-variable terms for query Q ,

$\{(x,c,I)^Q\}$ is a set of two-variable terms for query Q , and

$\{(x,c,I)^Q\}$ is a set of triplets, one triplet for each of the original relations referenced by the query. A triplet $\langle x,c,I \rangle$ for a relation R_i is such that

x is the network location of R_i ,

c is the cumulative cost of R_i which is zero; and

I is information about R_i for size and cost estimation.

OUTPUT: QPS - optimal strategy; c(QPS) - the cost of QPS.

* COMMENT - Depth-first Search algorithm of Figure C-2 is modified into a recursive form.

BEGIN

IF V^Q is not empty THEN

BEGIN

The cost c_i of $\langle x,c,I \rangle \in \{(x,c,I)^Q\}$ for relation R_i

is set to the cost of one-variable processing on R_i , i.e., the cost

of restrictions and initial projections on R_i .

$V^Q := V^Q - \{V_i^Q\}$; $\{(x,c,I)^Q\}$ is made empty.

END;

IF V^Q is empty THEN

BEGIN

* COMMENT - $\{(x,c,I)^Q\}$ contains only one triplet $\langle i, \dots \rangle$ which

represents the query result, relation R_i , obtained by a particular strategy.

QPS := "empty";

IF $x_i = i$ THEN

BEGIN

* i.e., the result is not located in the desired destination node*

c(QPS) := the cost of transfer of R_i to the destination node i ;

QPS := transfer of R_i to the destination node i ;

END

END

BEGIN

ELSE

c(QPS) := "infinity"; *The cost of the opt. str. is initialized to infinity.*

FOR each possible join to process one of two-variable terms $V_i^Q \in V^Q$ DO

BEGIN

Assume that the join to process V_i^Q is that of R_i with R_j in a network node of R_i or R_j , giving relation R_k .

Create $\{V\}^V$ and $\{(x,c,I)\}^V$ such that

(1) $\{V\}^V := \{V^Q\}^Q - \{V_i^Q\}$, i.e., the processed two-variable term is removed, and

(2) $\{(x,c,I)\}^V$ is derived from $\{(x,c,I)\}^Q$ by replacement of triplets $\langle i, \dots \rangle$ and $\langle j, \dots \rangle$ for relations R_i and R_j "processed" by a join under consideration, by a triplet for relation R_k , the join result, i.e.,

$\{(x,c,I)\}^V := \{(x,c,I)\}^Q - \{(x,c,I)_i, (x,c,I)_j\} \cup \{(x,c,I)_k\}$.

Note that c_k of $\langle x,c,I \rangle_k \in \{(x,c,I)\}^V$ represents the cost of R_k and

includes all CPU processing and data transfer cost to derive R_k .

CALL DEPTH-FIRST-SEARCH ($\{V\}^V$, $\{V^Q\}^Q$, $\{(x,c,I)\}^V$, QPS', c(QPS'));

(* note, $\{V\}^V = \emptyset$ *)

IF c(QPS') < c(QPS) THEN

BEGIN

c(QPS) := c(QPS');

QPS := QPS' augmented by transfer of R_i (if $x_i \neq x_k$) or R_j (if $x_j \neq x_k$) to x_k and a join of R_i with R_j ;

END;

END; (* FOR loop *)

END; (* else *)

END (* of procedure DEPTH-FIRST-SEARCH *);

Figure C-2(b) Recursive Depth-first Search

A depth-first search, as is the case in its application to the problem of a JPS strategy formulation, may often suffer from the pitfall of redundancy (Reingold, 1977). Figure 3-3 shows a portion of a decision tree used to examine two identical strategies which are represented by two sub-trees; there is no need to develop both sub-trees, one is sufficient. Since Depth-first Search develops only one path from the root to the last level at any one time, it is unable to recognize and remove this redundancy.

Figure 3-3 Two identical solutions examined by the Depth-first Search

The nodes of the last level represent examined strategies. The number of distinct strategies available for processing of a query is less than the number of nodes in the last level of the decision tree developed by the Depth-first Search algorithm. This is a result of redundancy.

C-2. DECOMPOSITION (DIVIDE AND CONQUER)

A "divide and conquer" approach used for finding the optimal strategy to process a given query is based on the observation that the query result is obtained by a final join of two relations which are partial results of some previous processing joins. Therefore, the problem can be divided into two sub-problems of strategy formulation for the two partial results followed by the final join. The sub-problems can then be similarly sub-divided resulting in a recursive solution.

For instance, assume that a simple query

$$(R_1.a \odot R_2.b) \odot R_2.c \odot R_3.d \odot R_3.e \odot R_4.f,$$

whose query graph is shown in Figure C-4(a), references relations R_1 , R_2 , R_3 , R_4 located in network nodes W, X, Y, Z, respectively. Assume that the query is processed by the following strategy and that the cost is measured in terms of the strategy execution delay:

- (i) R_1 is moved to X where it is joined with R_2 to give a partial result, relation R_{11} . The coalesced query graph to reflect this processing is shown in Figure C-4(b).
- (ii) R_4 is moved to Y where it is joined with R_3 to give a partial result, relation R_{12} . Again, the coalesced query graph is shown in Figure C-4(c).
- (iii) R_{11} is moved to Y where it is joined with R_{12} to give the query result, relation R_{31} .

The final join of relations R_{11} and R_{12} can commence only when R_{11} and R_{12} are both ready for further processing, while the join of R_1 with R_2 can proceed concurrently with the join of R_3 with R_4 as they are executed in different network nodes. The strategy timing diagram is shown in Figure C-4 d). Note that if instead of the transfer of R_1 to node X, R_2 is moved to node X where it is joined with R_1 , a new strategy with a different cost is created. As relations R_1 and R_2 have a different size, the cost of their transfer differs as well.

From the decomposition point of view, the original query can be divided into two sub-queries, q_x and q_y , whose processing results in relations R_{11} and R_{12} , respectively. In the example of Figure C-4, sub-query q_x is the two-variable term $R_1.a \theta_1 R_2.b$ which, when processed by a join, results in a relation R_{11} , located in X, while sub-query q_y is the two-variable term $(R_3.c \theta_2 R_4.d)$ which results in a relation R_{12} , located in node Y. The problem of processing the original query can be reduced into sub-problems of processing sub-queries q_x and q_y , followed by the final join of R_{11} and R_{12} . This conforms to the typical approach of "divide and conquer" in which a large problem is solved by its decomposition into a number of smaller sub-problems.

An additional strategy for the query processing is:

- (i) move R_1 to node X where it is joined with R_2 giving R_{11} ;
- (ii) move R_{11} to node Y where it is joined with R_3 giving R_{21} ;
- (iii) move R_4 to node Y where it is joined with R_{21} giving the query result.

The original query is divided into sub-queries $q_x = \{R_4\}$ and $q_y = \{(R_1.a \theta_1 R_2.b), (R_2.c \theta_2 R_3.d)\}$ which results in the relation R_{21} . A sub-query is either a relation or a set of two-variable terms.

The result of sub-queries, relations R_1 and R_2 , are joined to give the query result. Sub-query q_x may be further decomposed into sub-sub-queries $q_x' = (R_1.a \theta, R_2.b)$ and $q_x'' = R_2$.

The final join of the strategy stated above is different from that of the strategy shown in Figure 3-4. Furthermore, the total cumulative cost of relations, which may be partial results of previous processing, processed by the final join must be known before the cost of the total strategy can be found. The cost of the partial results can be found only by formulation of their optimal strategies. Consequently, when a query is divided into two sub-queries they must be such that their results can be joined to give the query result and all such divisions into sub-queries must be examined. This is the source of difficulties which make the application of the decomposition to the problem of QPS formulation result in an algorithm which is less elegant than desired.

The DIVIDE_AND_CONQUER algorithm, shown in Figure 3-5, assumes the existence of a procedure DIVIDE and a function DERIVE. Procedure DIVIDE is used to find all possible decompositions of query q into a pair of sub-queries (q_x, q_y) such that the results of q_x and q_y can be joined to give the result/answer for query q in the destination node d . Its input consists of a query q defined by

- (i) a network location d in which the final join should be processed;
- (ii) a set of two-variable terms $\{V''\}^q$ for query q ; and
- (iii) a set of triplets $\{(x, c, I)\}^q$ conveying information about relations referenced by the two-variable terms in $\{V''\}^q$; each triplet $(x, c, I)_i$ contains previously defined information about the relation R_i , i.e., the triplet contains the network location x_i , the cumulative cost c_i and information I_i which is used the size and cost estimation.

PROCEDURE DIVIDE_AND_CONQUER

INPUT: query q defined by i, V^i and x, c, I^i , where
 i is the destination node to which the query q result should be delivered.
 V^i is a set of two-variable terms for query q ; and
 x, c, I^i is a set of triplets x, c, I , one for each of the original relations referenced by the query.

OUTPUT: Triplet x, c, I_k for relation R_k which is the query result; and
 QPS_k , which is the optimal strategy to derive R_k .

* NOTE that the procedure assumes the existence of a global variable V , which is the set of one variable terms for the original query. *

BEGIN * DIVIDE_AND_CONQUER utilizes previously defined procedures DIVIDE and DERIVE *

IF V^i is empty THEN

BEGIN

* x, c, I^i contains only one triplet, x, c, I_k , which contains information about relation R_k , the query result. *

c_k of x, c, I_k is set to the cost of one variable processing, i.e., the cost of restrictions on R_k as specified by V^i and the cost of initial projection.

RETURN x, c, I_k and QPS_k = empty;

END

IF V^i contains only one two-variable term $R_i, a \in R_i, b$ THEN

BEGIN

* processing of $R_i, a \in R_i, b$, by a join results in the relation R_k *

$x, c, I_k := \text{DERIVE}(d, x, c, I_i, x, c, I_j)$;

$QPS_k :=$ is the movement of that relation R_i or R_j to node i which is not located in i already, followed by a join of R_i with R_j .

RETURN x, c, I_k and QPS_k ;

END

ELSE

BEGIN

CALL DIVIDE

input: query q

output: $Q = \{(q_x, q_y), x=1, 2, \dots, n\}$ -- set of pairs of sub-queries such that each pair can be processed by the final join in the node i to give the query result;

$n := 0$;

FOR each pair $(q_x, q_y) \in Q = \{(q_x, q_y), x=1, 2, \dots, n\}$ DO

BEGIN

CALL DIVIDE_AND_CONQUER

input: q_x of (q_x, q_y)

output: $(x, c, I)_x$, a triplet for relation R_x which is the result of sub-query q_x and

QPS_x , optimal strategy for processing q_x ;

CALL DIVIDE_AND_CONQUER

input: q_y of (q_x, q_y)

output: $(x, c, I)_y$, a triplet for relation R_y which is the result of sub-query q_y

and QPS_y , optimal strategy for processing q_y ;

(* R_x and R_y are the results of processing q_x and q_y respectively

and R_k is the query q result *)

$n := n + 1$;

$(x, c, I)_k := \text{DERIVE}(d, (x, c, I)_x, (x, c, I)_y)$;

$QPS_k := QPS_x$ plus QPS_y plus data transfer of R_x or R_y to d if it

is not their already, followed by the join of R_x with R_y ; END;

FIND k such that $c_k = \text{minimum } \{c_p \text{ of } (x, c, I)_p, p = 1, 2, \dots, n\}$;

RETURN $(x, c, I)_k$ and QPS_k ;

END

END {procedure DIVIDE_AND_CONQUER}

The procedure's output is $\{q_x, q_y, i\}, i=1,2,\dots,n$. It is a set of pairs q_x, q_y of sub-queries q_x and q_y such that q_x and q_y are joined in i to give the result for query q . Since a join can be executed only in the location of one of its operands, one of the results of sub-queries q_x or q_y must be located in the network node i .

Function DERIVE is used to derive information about the result of a join of relations R_1 and R_2 in a given network node. Input to the function consists of

1. the network location of the join execution and
2. two triplets (x, c, I_1) and (x, c, I_2) for relations R_1 and R_2 respectively.

The value of the function is a triplet (x, c, I_K) for relation R_K which is the result of the join of R_1 and R_2 , executed in the network location which is an input argument of DERIVE. The cost c_K of (x, c, I_K) includes the cumulative cost of processing and data transfer to derive R_K .

The recursive procedure DIVIDE_AND_CONQUER, which finds the optimal QPS using a "divide and conquer" approach is shown in Figure C-5. Input to the procedure is a query q and a node d in which the final join should be executed. Output is an optimal strategy for processing of the query q and its cost. If the query passed to the procedure DIVIDE_AND_CONQUER has only one two-variable term to be processed in the network location d , its cost is calculated in the "then" portion of the "if" statement. Otherwise, the query is divided into all pairs of sub-queries (q_x, q_y) such that the final join can be executed in the network location d . This is performed by the invocation of the procedure DIVIDE. Recursion is used to determine the optimal strategies for processing of q_x and q_y for each pair (q_x, q_y) . Finally, the optimal strategy for q is determined by calculating the

cost of the final join of the results of sub-queries q_x and q_y for each pair q_x, q_y returned by the procedure DIVIDE. From these final joins, the minimum cost strategy is determined and returned by the procedure.

The major drawback of this algorithm is similar to that of Depth-first Search; it examines many strategies redundantly. For example, consider a query referencing n relations. The query is divided by the algorithm into a set of pairs of sub-queries. Consider just two pairs of sub-queries, q_u, q_v and q_x, q_y . Assume that the sub-query q_u processes/contains relations $R_1, R_2, \dots, R_k$, while, q_v processes/contains relations $R_{k+1}, R_{k+2}, \dots, R_n$. Also assume that sub-queries q_x and q_y are such that q_x contains relations $R_1, R_2, \dots, R_{k+1}$, while q_y relations $R_{k+2}, R_{k+3}, \dots, R_n$. The two pairs of sub-queries differ in only one relation. Figure C-5(a) shows a query q referencing six relations, $R_i, i=1,2,\dots,6$. Figure C-5(b) shows the decomposition of the query into a pair of sub-queries (q_x, q_y) , while Figure C-5(c) shows the decomposition of the query into a pair of sub-queries (q_u, q_v) . In this particular case, $n=6$ and $k=4$.

At the next level of recursion, q_u, q_v, q_x, q_y are also sub-divided. Because q_u and q_x differ only in one relation, many of their sub-divisions are identical and optimal solutions for their executions are found redundantly. For example, q_u may be sub-divided into q_u' containing relations $R_1, R_2, \dots, R_{k-1}$ and q_u'' containing R_k . Query q_x will, amongst others, be sub-divided into sub-queries q_x' containing relations $R_1, R_2, \dots, R_{k-1}$ and q_x'' containing R_k and R_{k+1} . Sub-queries q_u' and q_x' are identical, but the algorithm will redundantly find the same optimal strategy for both of them.

Figure C-6 Decomposition of a query q

$$D_x[R_r] = \sum_{k=1}^m D_x[R_i^{(k)}] + \sum_{k=1}^m D[\phi(R_i^{(k)}, x_i^{(k)}, X)]$$

(iii) CUMULATIVE DOLLAR COST DUE TO CPU PROCESSING AND DATA TRANSFER:

The total dollar cost is the sum of the fragments' cumulative costs, plus the cost to transfer partitions to the node X and the cost of the the union, $D[U(R_i^{(k)}, R_r), Z]$.

$$D_x[R_r] = \sum_{k=1}^m D_x[R_i^{(k)}] + \sum_{k=1}^m D[\phi(R_i^{(k)}, x_i^{(k)}, X)] + D[U(R_i^{(k)}, R_r), Z]$$

(iv) CUMULATIVE TIME DELAY DUE TO CPU PROCESSING ONLY:

The projection to remove possible duplicate tuples is executed only when all fragments are consolidated, i.e., received. Consequently, the cumulative time delay for R_r is the sum of the maximum time delay of all fragments plus the delay to execute the union, $T[U(R_i^{(k)}, R_r), Z]$.

$$T_p[R_r] = \max \{ T_p[R_i^{(k)}], k=1, 2, \dots, m \} + T[U(R_i^{(k)}, R_r), Z]$$

(v) CUMULATIVE TIME DELAY DUE TO DATA TRANSFER ONLY:

Consider a relation R_i which is horizontally partitioned into $R_i^{(k)}$, $k=1, 2, \dots, m$, such that each fragment has an identical cumulative time delay of $T_x[R_i^{(k)}]$, i.e.,

$$T_x[R_i^{(1)}] = T_x[R_i^{(2)}] = \dots = T_x[R_i^{(m)}].$$

All fragments of R_i are completed in their respective locations $x_i^{(k)}$ at the same time. Assuming that the union of fragments is to be executed in the network node X ,

$$T_x[R_r] = T_x[R_i^{(1)}] + \sum_{k=1}^m T[\phi(R_i^{(k)}, x_i^{(k)}, X)].$$

For the example of Figure C-6, C-6(d) shows the decomposition of q_u into a pair of sub-queries (q_u' , q_u''), while C-6(e) shows the decomposition of q_x into a pair of sub-queries (q_x' , q_x'').

The application of a "divide and conquer" approach resulted in an algorithm exhibiting redundancy as described above. If it was possible to divide a query into two sub-queries without the examination of all possible decompositions produced by the procedure DIVIDE, not only would the redundancy be removed, but the algorithm would become efficient and elegant as well; however, it appears that no properties of distributed query processing can be exploited to find such a decomposition.

APPENDIX D

CUMULATIVE COST FUNCTIONS

Functions to determine the cumulative cost for partial results of relational operations, in a number of measures, are presented. A partial result's cumulative cost includes the total cost of all processing and data transfer to derive it; consequently, it also includes the cumulative cost of operands of the operation which derives it.

NOTATION

The cumulative cost of a relation R , where R could be a fragment of a horizontally partitioned relation, in a specific cost measure is denoted by

$D_p[R]$... dollar cost due to CPU processing only;

$D_x[R]$... dollar cost due to data transfer only;

$D_a[R]$... dollar cost due to CPU processing and data transfer;

$T_p[R]$... delay due to CPU processing only;

$T_x[R]$... delay due to data transfer only;

$T_a[R]$... delay due to CPU processing and data transfer;

$S_p[R]$... volume of data processed by CPUs;

$S_x[R]$... volume of data transferred; and

$S_a[R]$... sum of the size of partial results.

$T_{\alpha[Z]}$... is the delay to process a unit of data (byte) by the processor Z ;

$D[\alpha\{Z\}]$... is the dollar cost to process a unit of data (byte) by the processor Z;

$T[\beta\{X,Y\}]$... is the delay to transfer a unit of data (byte) from the network node X to Y;

$D[\beta\{X,Y\}]$... is the dollar cost to transfer a unit of data (byte) from the network node X to Y;

$\delta\{(R_i, R_r), Z\}$... denotes restriction(s) on R_i giving R_r , executed by the information processor Z, where all restrictions as specified by the query one-variable terms are cascaded and hence inferred by this notation;

$\pi\{(R_i, R_r), Z\}$... denotes a projection on R_i giving R_r , executed by the information processor Z;

$\omega\{(R_i, R_r), Z\}$... denotes initial processing (all specified restrictions and a projection to remove attributes of R_i which are neither target nor joining attributes), executed by the information processor Z;

$\ast\{(R_i, R_j, R_r), Z\}$... denotes a join of R_i with R_j giving R_r , executed by the information processor Z;

$U\{(R_i^{(k)}, R_r), Z\}$... denotes a union, of all horizontal fragments of R_i giving a non-partitioned relation R_r , where R_i is horizontally partitioned into $R_i^{(k)}$, $k=1,2,\dots,m$, executed by the information processor Z;

$\phi\{R_i, (X,Y)\}$... denotes the transfer of a relation R_i from the network node X to Y;

Φ ... denotes one of the relational operations or data transfer listed above;

$D[\Phi]$... denotes the dollar cost of the operation Φ ; and

$T[\Phi]$... denotes the time delay to perform operation Φ .

Note that $D[\Phi]$ and $T[\Phi]$ are not the cumulative cost and delay, respectively; they do not include the dollar cost and time delay of the operation's operands.

Functions are developed for the nine measures, $D_p[R]$, $D_x[R]$, ..., $S_a[R]$; for a partial result of

- initial processing of an elementary relation;
- a join;
- a fragmented join; and
- a union of a horizontally partitioned relation.

INITIAL PROCESSING

Initial processing includes all restrictions and a projection to remove unnecessary attributes. As

- (i) the initial processing is not preceded by any operations and
- (ii) its dollar cost ($D[\omega((R_i, R_r), Z)]$) and delay ($T[\omega((R_i, R_r), Z)]$) are assumed to be known,

the derivation of its cumulative cost functions is straight forward.

Consider an elementary relation R_i which could be a horizontal fragment, stored by the information processor Z which is connected to the network node X and assume that initial processing on R_i results in

the relation R_r . The cumulative cost for R_r is defined as follows.

- (i) CUMULATIVE DOLLAR COST DUE TO CPU PROCESSING ONLY:

$$D_p[R_r] = D[\omega\{(R_i, R_r), Z\}]$$

- (ii) CUMULATIVE DOLLAR COST DUE TO DATA TRANSFER ONLY:

$$D_x[R_r] = 0$$

As initial processing does not involve any data transfer, the cost due to a data transfer is zero.

- (iii) CUMULATIVE DOLLAR COST DUE TO CPU PROCESSING AND DATA TRANSFER:

$$D_a[R_r] = D_p[R_r] = D[\omega\{(R_i, R_r), Z\}]$$

- (iv) CUMULATIVE TIME DELAY DUE TO CPU PROCESSING ONLY:

$$T_p[R_r] = T[\omega\{(R_i, R_r), Z\}]$$

- (v) CUMULATIVE TIME DELAY DUE TO DATA TRANSFER ONLY:

$$T_x[R_r] = 0$$

- (vi) CUMULATIVE TIME DELAY DUE TO CPU PROCESSING AND DATA TRANSFER:

$$T_a[R_r] = T_p[R_r] = T[\omega\{(R_i, R_r), Z\}]$$

- (vii) CUMULATIVE VOLUME OF DATA PROCESSED BY CPUS:

Since the dollar cost due to CPU processing is directly proportional to the volume of processed data, this volume is derived in the same way as $D_p[R_r]$ evaluated with the cost to process a unit of data by any CPU

equal to one, i.e., with $D[\alpha(Z)] = \$1/\text{byte}$:

$$S_p[R_r] = D_p[R_r] \cdot (\text{byte}/\$) = D[\omega((R_1, R_r), Z)] \cdot (\text{byte}/\$)$$

(viii) CUMULATIVE VOLUME OF TRANSFERRED DATA:

$$S_x[R_r] = 0$$

(ix) CUMULATIVE SUM OF THE SIZE OF PARTIAL RESULTS:

$$S_s[R_r] = s_r = \text{Card}(R_r) \cdot \text{Width}(R_r)$$

RESULT OF A JOIN

Consider relations R_1 and R_j , located in the network nodes X and Y respectively, with their respective cumulative costs c_1 and c_j expressed in an appropriate measure of cost. A join of R_1 with R_j is executed in one of the network nodes X or Y and therefore, if $X \neq Y$, one of R_1 , R_j must be transferred. The cumulative cost c_r of the join result, relation R_r , is derived from the cumulative costs of the join operands c_1 and c_j , the cost of the data transfer and a join execution. If $X = Y$, i.e., both relations are located in the same network node, the cost of the data transfer in the following equations is defined to be zero. It is assumed for the following equations that the join is executed in the network node X with information processor Z ; therefore, it is R_j that is transferred from Y to X . If the join is executed in Y then R_j is replaced by R_1 in any cost pertaining to the data transfer.

(i) CUMULATIVE COST DUE TO CPU PROCESSING ONLY:

$D_p[R_r]$ is simply the sum of the cumulative costs of R_1 and R_j plus the dollar cost to execute the join of R_1 with R_j .

$$D_p[R_r] = D_p[R_1] + D_p[R_j] + D[\ast\{(R_1, R_j, R_r), Z\}]$$

(ii) CUMULATIVE DOLLAR COST DUE TO DATA TRANSFER ONLY:

$D_x[R_r]$ is the sum of the cumulative costs of R_1 and R_j plus the dollar cost to transfer R_j from Y to X.

$$D_x[R_r] = D_x[R_1] + D_x[R_j] + D[\ast\{R_j, (Y, X)\}]$$

(iii) CUMULATIVE DOLLAR COST DUE TO CPU PROCESSING AND DATA TRANSFER:

$D_a[R_r]$ is the sum of the cumulative costs of R_1 and R_j , the dollar cost to transfer R_j from Y to X and the dollar cost to execute the join of R_1 with R_j .

$$D_a[R_r] = D_a[R_1] + D_a[R_j] + D[\ast\{R_j, (Y, X)\}] + D[\ast\{(R_1, R_j, R_r), Z\}]$$

(iv) CUMULATIVE TIME DELAY DUE TO THE CPU PROCESSING ONLY:

The join of R_1 with R_j can be executed only when all processing to create both R_1 and R_j is complete. This occurs at the time which is the larger of the total time delays for R_1 and R_j , to which the time delay to execute the join is added.

$$T_p[R_r] = \max\{T_p[R_1], T_p[R_j]\} + T[\ast\{(R_1, R_j, R_r), Z\}]$$

(v) CUMULATIVE TIME DELAY DUE TO DATA TRANSFER ONLY:

Since the delay due to the CPU processing is ignored, the join of R_1 with R_j is considered to be complete when both R_1 and R_j are in the network node X.

$$T_X[R_r] = \max \{ T_X[R_1], T_X[R_j] + T[\phi[R_j, (Y, X)]] \}$$

(vi) CUMULATIVE TIME DELAY DUE TO CPU PROCESSING AND DATA TRANSFER:

The join of R_1 with R_j can commence only when both operands are complete and located in the node of the join execution, node X . This happens at time " $\max \{ T_X[R_1], T_X[R_j] + T[\phi[R_j, (Y, X)]] \}$ ", to which the time delay to join R_1 with R_j is added.

$$T_A[R_r] = \max \{ T_A[R_1], T_A[R_j] + T[\phi[R_j, (Y, X)]] \} + T[\psi((R_1, R_j, R_r), Z)]$$

(vii) TOTAL VOLUME OF DATA PROCESSED BY CPUs:

Since the dollar cost due to CPU processing is directly proportional to the volume of processed data, this volume, $S_p[R_r]$, is derived in the same way as $D_p[R_r]$ evaluated with the dollar cost to process a unit of data by any CPU set to one, i.e., with $D[a[Z]] = \$1/\text{byte}$, for all information processors:

$$S_p[R_r] = D_p[R_r] \cdot (\text{byte}/\$)$$

(viii) TOTAL VOLUME OF TRANSFERRED DATA:

Since the cumulative dollar cost due to data transfer, $D_p[R_r]$, is directly proportional to the volume of transferred data, the total volume of data transferred over the network, $S_x[R_r]$, is derived in the same way as $D_p[R_r]$ evaluated with the dollar cost to transfer a unit of data between any two network nodes set to 1, i.e., with $D[\phi[X, Y]] = \$1/\text{byte}$, for all X and Y , where $X \neq Y$.

$$S_x[R_r] = D_x[R_r] \cdot (\text{byte}/\$)$$

(ix) TOTAL SUM OF THE SIZE OF PARTIAL RESULT:

The total size of partial results is the sum of the size of partial

results for the join operands plus the size of the join result.

$$S_a[R_r] = S_a[R_i] + S_a[R_j] + s_r.$$

FRAGMENTED JOIN.

The cumulative cost for a partition $R_r^{(k)}$, which is the result of a fragmented join, is derived in exactly the same way as that of a join of two non-partitioned relations. In the following equations it is assumed that relations R_j and R_i are such that R_j is located in network node Y and R_i is horizontally partitioned into fragments $R_i^{(k)}$, $k=1,2,\dots,m$. Each horizontal fragment $R_i^{(k)}$, $k=1,2,\dots,m$, is assumed to be stored by the information processor $Z_i^{(k)}$ connected to the network node $x_i^{(k)}$. The result of the fragmented join is a relation R_r which is horizontally partitioned into $R_r^{(k)}$, $k=1,2,\dots,m$, such that the location of $R_r^{(k)}$ is $x_r^{(k)} = x_i^{(k)}$.

The following equations are those for the join result, such that the operand R_i is replaced by a partition $R_i^{(k)}$.

(i) CUMULATIVE DOLLAR COST DUE TO CPU PROCESSING ONLY:

$$D_p[R_r^{(k)}] = D_p[R_i^{(k)}] + D_p[R_j] + D[\{(R_i^{(k)}, R_j, R_r^{(k)}), Z\}]$$

(ii) CUMULATIVE DOLLAR COST DUE TO DATA TRANSFER ONLY:

$$D_x[R_r^{(k)}] = D_x[R_i^{(k)}] + D_x[R_j] + D[\{R_j, (Y, x_i^{(k)})\}]$$

(iii) CUMULATIVE DOLLAR COST DUE TO CPU PROCESSING AND DATA TRANSFER:

$$D_a[R_r^{(k)}] = D_a[R_i^{(k)}] + D_a[R_j] + D[\phi(R_j, (Y, x_i^{(k)}))] + \\ D[\phi(R_i^{(k)}, R_j, R_r^{(k)}, Z)]$$

(iv) CUMULATIVE TIME DELAY DUE TO CPU PROCESSING ONLY:

$$T_p[R_r^{(k)}] = \max \{ T_p[R_i^{(k)}], T_p[R_j] \} + T[\phi(R_i^{(k)}, R_j, R_r^{(k)}, Z)]$$

(v) CUMULATIVE TIME DELAY DUE TO DATA TRANSFER ONLY:

$$T_x[R_r^{(k)}] = \max \{ T_x[R_i^{(k)}], T_x[R_j] + T[\phi(R_j, (Y, x_i^{(k)}))] \}$$

(vi) CUMULATIVE TIME DELAY DUE TO CPU PROCESSING AND DATA TRANSFER:

$$T_a[R_r^{(k)}] = \max \{ T_a[R_i^{(k)}], T_a[R_j] + T[\phi(R_j, (Y, x_i^{(k)}))] \} + \\ T[\phi(R_i^{(k)}, R_j, R_r^{(k)}, Z)]$$

(vii) TOTAL VOLUME OF DATA PROCESSED BY CPU:

$$S_p[R_r^{(k)}] = D_p[R_r^{(k)}] \cdot (\text{byte/s}),$$

where $D[\alpha\{Z\}] = S/\text{byte}$ is used for derivation of $D_p[R_i^{(k)}]$, $D_p[R_j]$ and $D_p[R_r^{(k)}]$, for all information processors Z.

(viii) TOTAL VOLUME OF TRANSFERRED DATA:

$$S_x[R_r^{(k)}] = D_x[R_r^{(k)}] \cdot (\text{byte/s}),$$

where $D[S\{X, Y\}] = S/\text{byte}$ is used for derivation of $D_x[R_i^{(k)}]$, $D_x[R_j]$ and $D_x[R_r^{(k)}]$, for all X and Y, $X \neq Y$.

(ix) TOTAL SUM OF THE SIZE OF PARTIAL RESULT:

$$S_a[R_r^{(k)}] = S_a[R_i^{(k)}] + S_a[R_j] + s_r$$

UNION OF A HORIZONTALLY PARTITIONED RELATION

Consider a relation R_i , which is horizontally partitioned into $R_i^{(k)}$, $k=1,2,\dots,m$, such that each partition is located in the network node $x_i^{(k)}$. Execution of a union of partitions executed in a network node $X \in \{x_i^{(k)}, k=1,2,\dots,m\}$ with an information processor Z , is briefly reviewed. First, an empty relation R_r is created and as the fragments of R_i are received from the network they are directly "appended" to R_r . Finally, when all fragments of R_i are consolidated in this way, a projection is executed to remove possible duplicate tuples.

(i) CUMULATIVE DOLLAR COST DUE TO CPU PROCESSING ONLY:

The total dollar cost of R_r due to the CPU processing is the sum of the total costs of each of the fragments $R_i^{(k)}$, $k=1,2,\dots,m$, plus the sum of the cost to execute the union of fragments, $D[U\{(R_i^{(k)}, R_r), Z\}]$.

$$D_p[R_r] = \sum_{k=1}^m D_p[R_i^{(k)}] + D[U\{(R_i^{(k)}, R_r), Z\}]$$

(ii) CUMULATIVE DOLLAR COST DUE TO DATA TRANSFER ONLY:

The total cost due to a data transfer is the sum of the costs for each of the fragments of R_i plus the cost to transfer partitions to the node X . Note that the cost of data transfer from node $x_i^{(k)}$ to X is zero if $X = x_i^{(k)}$.

$$D_X[R_r] = \sum_{k=1}^m D_X[R_i^{(k)}] + \sum_{k=1}^m D[\phi(R_i^{(k)}, (x_i^{(k)}, X))]$$

(iii) CUMULATIVE DOLLAR COST DUE TO CPU PROCESSING AND DATA TRANSFER:

The total dollar cost is the sum of the fragments' cumulative costs, plus the cost to transfer partitions to the node X and the cost of the the union, $D[U\{R_i^{(k)}\}, R_r, Z]$.

$$D_a[R_r] = \sum_{k=1}^m D_a[R_i^{(k)}] + \sum_{k=1}^m D[\phi(R_i^{(k)}, (x_i^{(k)}, X))] + D[U\{R_i^{(k)}\}, R_r, Z]$$

(iv) CUMULATIVE TIME DELAY DUE TO CPU PROCESSING ONLY:

The projection to remove possible duplicate tuples is executed only when all fragments are consolidated, i.e., received. Consequently, the cumulative time delay for R_r is the sum of the maximum time delay of all fragments plus the delay to execute the union, $T[U\{R_i^{(k)}\}, R_r, Z]$.

$$T_p[R_r] = \max \{ T_p[R_i^{(k)}], k=1, 2, \dots, m \} + T[U\{R_i^{(k)}\}, R_r, Z]$$

(v) CUMULATIVE TIME DELAY DUE TO DATA TRANSFER ONLY:

Consider a relation R_i which is horizontally partitioned into $R_i^{(k)}$, $k=1, 2, \dots, m$, such that each fragment has an identical cumulative time delay of $T_X[R_i^{(k)}]$, i.e.,

$$T_X[R_i^{(1)}] = T_X[R_i^{(2)}] = \dots = T_X[R_i^{(m)}].$$

All fragments of R_i are completed in their respective locations $x_i^{(k)}$ at the same time. Assuming that the union of fragments is to be executed in the network node X ,

$$T_X[R_r] = T_X[R_i^{(1)}] + \sum_{k=1}^m T[\phi(R_i^{(k)}, (x_i^{(k)}, X))].$$

Since the time delay due to the CPU processing is ignored for the cost measure in terms of the total time delay due to the data transfer only, the total time delay is determined by the sum of the time when the fragments are ready, $T_X[R_i^{(1)}]$, and the total time it takes to transfer the fragments to the node X where the union is executed. The fragments are submitted to the network for transmission at the same time; however, the receiving node X is capable of receiving only one fragment at a time - hence the sum of the time delays to transfer all fragments.

Now consider the same case, i.e., all partitions have identical cumulative time delay with the exception of the partition $R_i^{(1)}$ which has a much higher total time delay, i.e., $T_X[R_i^{(1)}] \gg T_X[R_i^{(k)}]$, $k=2,3,\dots,m$. Then

$$T_X[R_r] = T_X[R_i^{(1)}] + T[\phi(R_i^{(1)}, (x_i^{(1)}, X))].$$

In this case, the cumulative time delay of R_r is remarkably different.

Let

t_m ... be the time when all fragments are consolidated (received) in the node X .

At time t_m the projection to remove possible duplicate tuples is executed. However, for a total time delay only due to transferring data, $T[U\{(R_i^{(k)}, R_r), Z\}] = 0$ and

$$T_X[R_r] = t_m + T[U\{(R_i^{(k)}, R_r), Z\}] = t_m.$$

Since the receiving node X can receive only one fragment at a time, the following relationships, for the consolidation time t_m , hold:

$$t_m > \min \{T_X[R_i^{(k)}], k=1,2,\dots,m\} + \sum_{k=1}^m T[\Phi\{R_i^{(k)}, (x_i^{(k)}, X)\}]$$

$$t_m < \max \{T_X[R_i^{(k)}], k=1,2,\dots,m\} + \sum_{k=1}^m T[\Phi\{R_i^{(k)}, (x_i^{(k)}, X)\}]$$

The difficulty in the derivation of $T_X[R_r]$ for a union of fragments lies in the fact that the receiving node is assumed to be able to receive only one fragment at a time and that the fragments may be ready for the transfer at arbitrary times.

Assume that the fragments are labelled in such a way that for two fragments $R_i^{(k)}, R_i^{(n)}$, if $k < n$ then $T_X[R_i^{(k)}] < T_X[R_i^{(n)}]$, i.e., the fragment $R_i^{(k)}$ is ready for transmission sooner than $R_i^{(n)}$. $T_X[R_r]$ can be found using a recursive formula in the following way.

Let

$t_k \dots$ denote the time when fragments 1, 2, ..., k are consolidated, i.e., have arrived at the receiving node X . Clearly,

$$t_1 = T_X[R_i^{(1)}] + T[\Phi\{R_i^{(1)}, (x_i^{(1)}, X)\}]$$

To calculate t_2 , there are two cases:

$$a) T_X[R_i^{(2)}] < t_1: \quad t_2 = t_1 + T[\Phi\{R_i^{(2)}, (x_i^{(2)}, X)\}]$$

The fragment $R_i^{(2)}$ must wait for the completion of reception of the fragment $R_i^{(1)}$, which occurs at time t_1 .

$$b) T_X[R_i^{(2)}] \geq t_1: \quad t_2 = T_X[R_i^{(2)}] + T[\psi\{R_i^{(2)}, (x_i^{(2)}, X)\}]$$

The fragment $R_i^{(2)}$ can be received immediately as the reception of $R_i^{(1)}$ has been completed by the time $T_X[R_i^{(2)}]$ when $R_i^{(2)}$ is submitted for transmission.

In general, for $k=1, 2, \dots, m-1$:

$$a) T_X[R_i^{(k+1)}] < t_k: \quad t_{k+1} = t_k + T[\psi\{R_i^{(k+1)}, (x_i^{(k+1)}, X)\}]$$

The fragment $R_i^{(k+1)}$ must wait for the completion of reception of fragments $R_i^{(j)}$, $j=1, 2, \dots, k$, which occurs at time t_k .

$$b) T_X[R_i^{(k+1)}] \geq t_k: \quad t_k = T_X[R_i^{(k+1)}] + T[\psi\{R_i^{(k+1)}, (x_i^{(k+1)}, X)\}]$$

The fragment $R_i^{(k+1)}$ can be received immediately as the reception of $R_i^{(j)}$, $j=1, 2, \dots, k$, has been completed by the time $T_X[R_i^{(k+1)}]$ when $R_i^{(k+1)}$ is submitted for transmission.

Finally, the cumulative time delay for R_r is the time when all fragments has been received at node X, i.e.,

$$T_X[R_r] = t_m.$$

(vi) CUMULATIVE TIME DELAY DUE TO CPU PROCESSING AND DATA TRANSFER:

The total time delay for R_r , which is the union of fragments $R_i^{(k)}$, $k=1, 2, \dots, m$, due to the CPU processing and data transfer, is derived using the time when all fragments are consolidated/received in the node X and the delay to execute a projection to remove possible duplicate

tuples. The time when all fragments of R_i have been received in the node X is derived as for $T_X[R_i]$ (see (v) above) by replacement of $T_X[R_i^{(k)}]$ with $T_a[R_i^{(k)}]$. As above, assume that

the fragments of R_i are labelled in such a way that for two fragments $R_i^{(k)}$, $R_i^{(n)}$, if $k < n$ then $T_a[R_i^{(k)}] < T_a[R_i^{(n)}]$, i.e., the fragment $R_i^{(k)}$ is ready for transmission sooner than $R_i^{(n)}$; and

t_k denotes the time when all fragments $j=1,2,\dots,k$ have been received at the node X .

Then

$$t_1 = T_a[R_i^{(1)}] + T[\phi\{R_i^{(1)}, (x_i^{(1)}, X)\}]$$

and for $k=1,2,\dots,m-1$:

$$T_a[R_i^{(k+1)}] < t_k: \quad t_{k+1} = t_k + T[\phi\{R_i^{(k+1)}, (x_i^{(k+1)}, X)\}]$$

The fragment $R_i^{(k+1)}$ must wait for the completion of reception of fragments $R_i^{(j)}$, $j=1,2,\dots,k$, which occurs at time t_k .

$$T_a[R_i^{(k+1)}] \geq t_k: \quad t_k = T_a[R_i^{(k+1)}] + T[\phi\{R_i^{(k+1)}, (x_i^{(k+1)}, X)\}]$$

The fragment $R_i^{(k+1)}$ can be received immediately as the reception of $R_i^{(j)}$, $j=1,2,\dots,k$, has been completed by the time $T_a[R_i^{(k+1)}]$ when $R_i^{(k+1)}$ is submitted for transmission.

Finally, the cumulative time delay for R_i is the time when all fragments has been received at node X , i.e., t_m , plus the time to execute the union, $T[U\{(R_i^{(k)}, R_i), Z\}]$.

$$T_a[R_r] = t_m + T[U\{(R_i^{(k)}, R_r), Z\}].$$

(vii) TOTAL VOLUME OF DATA PROCESSED BY CPUs:

Since the cumulative dollar cost due to CPU processing, $D_p[R_r]$, is directly proportional to the volume of processed data, this volume, $S_p[R_r]$, is simply $D_p[R_r]$, derived with the dollar cost to process a unit of data set to 1, i.e., with $D[\alpha\{Z\}] = \$1/\text{byte}$, for all information processors.

$$S_p[R_r] = D_p[R_r] \quad (\text{byte}/\$),$$

where $D[\alpha\{Z\}] = \$1/\text{byte}$ is used for derivation of $D_p[R_i^{(k)}]$, $k=1,2,\dots,m$, and $D_p[R_r]$, for all information processors Z .

(viii) TOTAL VOLUME OF TRANSFERRED DATA:

Since the cumulative dollar cost due to data transfer, $D_x[R_r]$, is directly proportional to the volume of transferred data, the total volume of data transferred over the network to derive R_r , $S_x[R_r]$, is simply $D_x[R_r]$ derived with the dollar cost to transfer a unit of data between any two network nodes set to 1, i.e., with $D[\beta\{X,Y\}] = \$1/\text{byte}$, for all X and Y , where $X \neq Y$.

$$S_x[R_r] = D_x[R_r] \quad (\text{byte}/\$),$$

where $D[\beta\{X,Y\}] = \$1/\text{byte}$ is used for derivation of $D_x[R_i^{(k)}]$, $k=1,2,\dots,m$, and $D_x[R_r]$, for all X and Y , $X \neq Y$.

(ix) TOTAL SUM OF THE SIZE OF PARTIAL RESULTS:

The cumulative size of the partial results for R_r , which is the union of portions of $R_i^{(k)}$, $k=1,2,\dots,m$, is the sum of the cumulative size of

all operands and size of the union, s_r .

$$S_a[R_r] = \sum_{k=1}^n S_a[R_i^{(k)}] + s_r.$$

APPENDIX E

MODELLED DDB APPLICATION PARAMETERS

The appendix contains parameters defining the distributed environment of the modelled application of a DDB. It consists of five sections. Section E-1 describes the relations as they would be defined on the global view of the DB. Section E-2 specifies the parameters relevant to information processors. In particular, the dollar cost and delay to process a unit of data (byte) is shown for each information processor.

Network nodes to which information processors are connected are defined in the section E-3. The dollar cost and delay to transfer a unit of data (byte) between any two nodes is shown.

Partitioning of relations and the network locations of elementary relations is shown in the section E-4. Finally, section E-5 lists the queries and displays them in their conjunctive normal form.

Note that a sample input of parameters to the modelling software is shown in Appendix F.

E-1. RELATIONS/ATTRIBUTES/DOMAINS

Relations are specified using the following format. The relation name is specified first together with its cardinality and, if the relation name and name of the attributes are insufficient to indicate its use, the relation's content is briefly described. It is then followed by a list of its attributes. An attribute name is followed by "(key=true)" if it is the relation's primary key or a member of the composite primary key. For each attribute its width/length is shown in bytes. This is the byte length of values drawn from the attribute's domain. In addition, if the name of an attribute does not clearly indicate its use, a brief explanation is given.

The firm MONAX has two main offices, MAIN-WEST and MAIN-CENTRAL, while the company MONREAL, which is a subsidiary of MONAX, has only one office. The companies share two field headquarters. Attribute OFF# is used to identify both main offices and the field headquarters.

Note that "_C" at the end of an attribute name stands for a "_CODE".

Relation: OFFICE (cardinality=5)

There are three main offices (two for the firm MONAX and one for MONREAL) and two field headquarters.

OFF#	(key=true)	width=3
OFF_NAME		width=15
LOCATION		width=15

Relation: DEPT (cardinality=30)

There are six departments, on the average, in each of the five offices/field headquarters.

DEPT#	(key=true)	width=3
DEPT_NAME		width=15

Relation: OFFICE_DEPT (cardinality=30)

OFF# (key=true) width=3
DEPT# (key=true) width=3

Relation: DEPT_POSITION (cardinality=120)

There are approximately four position types per department. Note that there could be several positions of each type in each department. For example, there could be several secretaries in a department, each having a position of type secretary.

DEPT# (key=true) width=3
POSITION_C width=3
(POSITION_C is the code of a position. For example, code "PGM" could denote a programmer position.)
POSITION_NUM width=4
(POSITION_NUM is the number of positions with a code POSITION_C in the department with a number DEPT#.)
POS_NUM_FILL width=4
(The number of filled positions.)

Relation: EMP (cardinality=240)

This relation contains some information about each employee. There are 240 employees in offices, field headquarters and mobile teams which are attached to field headquarters.

EMP# (key=true) width=7
POSITION_C width=3
PAY_RATE width=4
START_DATE width=6
END_DATE width=6
TERMINATE_C width=3
EXPERIENCE_C width=3

Relation: DEPT_EMP (cardinality=240)

DEPT# (key=true) width=3
EMP# (key=true) width=7

Relation: JOB_CUSTOMER (cardinality=10)

There are 10 active jobs in various stages.

JOB# (key=true) width=5
 CUST# (key=true) width=5
 JOB_NAME width=15
 JOB_LOCATION width=15

Relation: JOB_OFFICE (cardinality=10)

The relation contains information about which offices are responsible for which jobs. Each job is performed by one of the main offices and one field headquarters.

JOB# (key=true) width=5
 OFF# (key=true) width=3
 HQ# (key=true) width=3
 (HQOFF# is the "OFF#" of the field headquarters which performs the job.)

Relation: EMP_JOB (cardinality=650)

Which employees work on which jobs is contained in this relation.

EMP# (key=true) width=7
 JOB# (key=true) width=5
 JOB_POS_C width=3
 (Position required by the job and performed by the employee with a number "EMP#".)
 PERCENT TIME width=4
 (An employee may not work full time one one particular job.)
 START DATE width=6
 END DATE width=6

Relation: JOB_PHASE (cardinality=100)

Phases of jobs are identified by this relation. There are ten jobs, each going through approximately 10 phases.

JOB# (key=true) width=5
 PHASE_C (key=true) width=3
 PROPOSE DATE width=6
 START DATE width=6
 END DATE width=6

Relation: JOB_MATERIAL (cardinality=2500)

Material requirements for jobs. There are about 250 different types of items required for ten jobs.

JOB#	(key=true)	width=5
ITEM#	(key=true)	width=8
QTY	(key=true)	width=4
PHASE_C	(key=true)	width=3
EXP_PRICE	(key=true)	width=4

Relation: JOB_MANPOWER (cardinality=300)

Manpower requirements for the ten jobs are identified by this relation. There are about 30 types of positions for one job.

JOB#	(key=true)	width=5
PHASE_C	(key=true)	width=3
POSITION_C	(key=true)	width=3
RATE CHARGED		width=4
HOURS		width=4

Relation: ITEMS (cardinality=7000)

There are about 7000 different items kept on a file. Any time a new item, which has not been used on previous jobs, is needed, it is added to this relation.

ITEM#	(key=true)	width=8
ITEM_NAME		width=15
WEIGHT		width=4
ITEM_DESCRIP		width=30

Relation: SUPPLY_ORDER (cardinality=1000)

Purchase orders for each job. There are, on the average, 100 purchase orders per each of ten jobs. There are ten items (item types), on the average, for each purchase order.

JOB#	(key=true)	width=5
PO#	(key=true)	width=10

("PO#" -- Purchase Order Number)

Relation: PO_SUPPLIER (cardinality=1000)

Suppliers of purchase orders are identified by this relation.

PO#	(key=true)	width=10
SUPPLIER#		width=5
PO_STATUS_C		width=3

("PO_STATUS_C" -- Purchase Order Status Code)

Relation: PO (cardinality=10000)

Purchase orders. There are about ten items per each of 1000 purchase orders.

PO#	(key=true)	width=10
ITEM#	(key=true)	width=8
QTY		width=4
ORDER_DATE		width=6
DELIVERY_DATE		width=6
PRICE		width=4

Relation: PO_RCVD (cardinality=7000)

Received shipments of ordered items. There are 1.5 deliveries, on the average, per purchase order; however, for about 50% of the purchase orders, any deliveries are yet to be made.

PO#	(key=true)	width=10
ITEM#	(key=true)	width=8
QTY		width=4
DATE	(key=true)	width=6

Relation: QUOTES (cardinality=10000)

ITEM#	(key=true)	width=8
SUPPLIER#	(key=true)	width=5
QTY	(key=true)	width=4
QUOTE_PRICE	(key=true)	width=4
DATE	(key=true)	width=6

Relation: SUPPLY (cardinality=10000)

ITEM#	(key=true)	width=8
SUPPLIER#	(key=true)	width=5

Relation: SUPPLIER (cardinality=50)

SUPPLIER#	(key=true)	width=5
SUP_NAME		width=15
ADDRESS		width=15
STATUS_C		width=3

Relation: CUSTOMER (cardinality=10)

CUST# (key=true) width=5
CUST_NAME width=15
ADDRESS width=15
STATUS_C width=3

E-2. INFORMATION PROCESSOR PARAMETERS

Each office and field headquarters have an information processor. For each processor the delay (in seconds) and dollar cost for processing a unit of data (byte) is shown.

OFFICE/HEADQ. INF. PROCESSOR	DOLLAR COST (to process a byte) (in \$)	DELAY (to process a byte) (in seconds)
MONAXC	0.00001	0.00005
MONAXW	0.000012	0.00004
MONREAL	0.0000095	0.00006
HQHALIFAX	0.000009	0.00009
HQEDMONTON	0.0000085	0.0001

MONAXW ... is the info. proc. for MONAX in Calgary
 MONAXC ... is the info. proc. for MONAX in Toronto
 MONREAL ... is the info. proc. for MONREAL in Calgary
 HQHALIFAX ... is the info. proc. for headq. in Halifax
 HQEDMONTON ... is the info. proc. for headq. in Edmonton

Note that information processors are assumed to be of capacities comparable to those of the VAX family of computers. The secondary storage devices attached to the processors are of storage capacities and delays comparable to those of RAS1 disk pack with

28 msec. average seek time,
 8.3 msec. average latency and
 17 Megabits/sec data transfer rate.

Data is stored in pages of 1024 bytes each. It is assumed that each 512 bytes of retrieved data incurs one disk access with one track seek (28 msec) and 8.3 msec. latency. This is prorated over one byte, giving a delay of approximately 0.000035 sec/byte due to access to secondary storage devices. Further slow down, by a factor of 1.5, is

due to the processing of data by the CPU itself and, in particular, due to other applications co-existing within the time-sharing environment of the VMS operating system.

The dollar cost ranges between \$0.9. and \$1.2 per 100,000 bytes of processed data. It is assumed that a system with lower delay is more expensive than one which incurs higher delay to process a unit of data.

E-3. NETWORK PARAMETERS

The delay (in seconds) and the dollar cost of transferring a unit of data (byte) between any two information processors is shown below. The dollar cost is comparable to that of the DATAPAC network. Delay is based on the assumption that information processors are connected to the network using the network's standard services with modems which have the following rates:

INFORMATION PROCESSORS CONNECTION TO THE NETWORK
(BITS/SEC)

MONAXW	...	2400
MONAXC	...	2400
MONREAL	...	2400
HQHALIFAX	...	1200
HQEDMONTON	...	1200

Note that MONAX and MONREAL offices, both located in Calgary, due to their close proximity have a local connection with a data transfer rate of 9600 bits/sec.

DOLLAR COST TO TRANSFER A UNIT OF DATA

	MONAXW	MONAXC	MONREAL	HQEDM.	HQHAL.
MONAXW	0.0	0.000089	0.000015	0.00003	0.0001
MONAXC	0.000089	0.0	0.000089	0.000095	0.00007
MONREAL	0.000015	0.000089	0.0	0.00003	0.0001
HQEDM.	0.00003	0.000095	0.00003	0.0	0.00013
HQHAL.	0.0001	0.00007	0.0001	0.00013	0.0

E-4. PARTITIONING AND DISTRIBUTION OF RELATIONS

The partitioning of relations and their distribution is shown in this section. Note that as there are no vertically partitioned relations in this application of a DDB, any fragmentation/partitioning of relations refers to horizontal fragmentation/partitioning. For each relation, its total cardinality is shown. It is then followed by indication whether it is fragmented (horizontally). If the relation is fragmented, the cardinality and locations of fragments are shown.

MONAXW ... is the info. proc. for MONAX in Calgary
 MONAXC ... is the info. proc. for MONAX in Toronto
 MONREAL ... is the info. proc. for MONREAL in Calgary
 HQHALIFAX ... is the info. proc. for headq. in Halifax
 HQEDMONTON ... is the info. proc. for headq. in Edmonton

RELATION	CARD. (total)	FRAGMENTED (horizontal) (true/false)	INFO. PROC. (storing a relation/ fragment)	CARD. (fragment)
OFFICE	5	false	MONAXC	
DEPT	30	true	MONAXC	12
			MONAXW	12
			MONREAL	6
DEPT_POSITION	120	true	MONAXC	80
			MONREAL	40
EMP	240	true	MONAXC	50
			MONAXW	50
			MONREAL	50
			HQHALIFAX	45
			HQEDMONTON	45

DEPT_EMP 240 true

MONAXC 50
 MONAXW 50
 MONREAL 50
 HQHALIFAX 45
 HQEDMONTON 45

JOB_CUSTOMER 10 true

MONAXC 7
 MONREAL 3

JOB_OFFICE 10 true

MONAXC 7
 MONREAL 3

EMP_JOB 650 true

MONAXC 150
 MONAXW 150
 MONREAL 100
 HQHALIFAX 100
 HQEDMONTON 100

JOB_PHASE 100 true

MONAXC 30
 MONAXW 40
 MONREAL 30

JOB_MATERIAL 2500 true

MONAXC 500
 MONAXW 500
 MONREAL 500
 HQHALIFAX 500
 HQEDMONTON 500

JOB_MANPOWER 300 true

MONAXC 60
 MONAXW 60
 MONREAL 60
 HQHALIFAX 60
 HQEDMONTON 60

ITEMS 10000 false

MONAXW

SUPPLY_ORDER 1000 true

HQHALIFAX 500
 HQEDMONTON 500

PO_SUPPLIER 1000 true

HQHALIFAX 500
 HQEDMONTON 500

PO	1000	true	HQHALIFAX	500
			HQEDMONTON	500
PO_RCVD	1000		HQHALIFAX	500
			HQEDMONTON	500
QUOTES	10000	false	MONAXW	
SUPPLY	5000	false	MONAXW	
SUPPLIER	50	false	MONAXW	
CUSTOMER	10	true	MONAXC	7
			MONREAL	3

E-5. QUERIES

Each modelled query is shown in English and also in its conjunctive normal form.

Query 1:

"Find departments which have programmer positions, ie., which have positions with a code 'PGM'. Also list their office #s."

```
(DEPT_POSITION.POSITION_CODE = 'PGM' ) AND
(OFFICE.OFF# = OFFICE_DEPT.OFF# ) AND
(OFFICE_DEPT.DEPT# = DEPT_POSITION )
```

Target attributes: OFFICE.OFF#, DEPT.DEPT#

Query 2:

"List all currently employed programmers. List their employee, department and office numbers."

```
(EMP.POSITION_CODE = 'PGM' ) AND
(OFFICE_DEPT.DEPT# = DEPT_EMP.DEPT# ) AND
(DEPT_EMP.EMP# = EMP.EMP# )
```

Target attributes: EMP.EMP#, DEPT.DEPT#, OFFICE_DEPT.OFFICE#

Query 3:

"List all currently employed programmers in offices MONAXW and MONREAL. List employee, department and office numbers and also department name."

```
(EMP.POSITION_CODE = 'PGM' ) AND
(OFFICE_DEPT.DEPT# = DEPT_EMP.DEPT# ) AND
(DEPT_EMP.EMP# = EMP.EMP# ) AND
(DEPT_EMP.DEPT# = DEPT.DEPT# ) AND
(OFFICE_DEPT.DEPT# = DEPT.DEPT# )
```

Target attributes: EMP.EMP#, DEPT.DEPT#, OFFICE_DEPT.OFF#, DEPT.NAME

Query 4: "Find employees who work on a job with a job number "job12" and are paid more than the rate charged to the customer". List employee number, position, pay_rate, and the rate charged to the customer."

```
(EMP.PAY_RATE > JOB_MANPOWER.RATE_CHARGED ) AND
(EMP.EMP# = EMP_JOB.EMP# ) AND
(EMP_JOB.JOB# = JOB_MANPOWER.JOB# ) AND
(EMP_JOB.JOB_POS_C = JOB_MANPOWER.POSITION_C ) AND
```

(EMP_JOB.JOB# = 'JOB12')

Target Attr: EMP.EMP#, EMP.POSITION, EMP.PAY_RATE,
JOB_MANPOWER.POSITION_C, JOB_MANPOWER.RATE_CHARGED

Query 5:

"Find employees who work on a job with a job number "job02". List employee number, department number, department name and office number."

(EMP.EMP# = EMP_JOB.EMP#) AND
(DEPT_EMP#.EMP# = EMP.EMP#) AND
(DEPT.DEPT# = DEPT_EMP.DEPT#) AND
(DEPT.DEPT# = OFFICE_DEPT.DEPT#) AND
(EMP_JOB.JOB# = 'JOB02')

Target Attr: DEPT_EMP.EMP#, DEPT.DEPT#, DEPT.NAME, OFFICE_DEPT.OFF#

Query 6:

"Find employees of the firm MONREAL who are seconded to jobs of the MONAX firm office MONAXW. Find employee numbers and jobs they work on. List employee number, department number and job number."

(OFFICE_DEPT.OFF# = 'MONR1') AND
(DEPT_EMP.DEPT# = OFFICE_DEPT.DEPT#) AND
(DEPT_EMP.EMP# = EMP_JOB.EMP#) AND
(JOB_OFFICE.JOB# = EMP_JOB.JOB#) AND
(JOB_OFFICE.OFF# = 'MONW2')

Target Attr: DEPT_EMP.EMP#, DEPT_EMP.DEPT#, JOB_OFFICE.JOB#

Query 7:

"Find jobs of employees for a given department of MONAXC office. List employee number, department # and department name. For each employee list the job #, job phase, job position code, percent time and start and end dates. Also list the # of the office managing the job."

(EMP_JOB.EMP# = DEPT_EMP.EMP#) AND
(DEPT_EMP.DEPT# = DEPT_.DEPT#) AND
(DEPT_EMP.DEPT# = 'dept2') AND
(EMP_JOB.JOB# = JOB_OFFICE.JOB#)

Target Attr: DEPT.DEPT NAME, DEPT_EMP.EMP#, JOB_OFFICE.JOB#,
JOB_OFFICE.JOB#, EMP_JOB.JOB_POS_C,
EMP_JOB.PERCENT TIME, EMP_JOB.START DATE,
EMP_JOB.END DATE

Query 8:

"List all information in the relation EMP_JOB, for a given time period, for employees of the office MONAXW."

(JOB_OFFICE.OFF# = 'MONO2') AND

```
(EMP_JOB.JOB#      = JOB_OFFICE.JOB# ) AND
(EMP_JOB.START_DATE > '270285'      ) AND
(EMP_JOB.END_DATE  < '270585'      )
```

Target Attr: EMP_JOB.JOB#, EMP_JOB.EMP#, EMP_JOB.POS_C,
EMP_JOB.PERCENT_TIME, EMP_JOB.START_DATE,
EMP_JOB.END_DATE

Query 9: "List the number, pay rate and experience of all headquarter employees."

```
(EMP.EMP#      = DEPT_EMP.EMP#      ) AND
(DEPT_EMP.DEPT# = OFFICE_DEPT.DEPT# ) AND
(OFFICE_DEPT.OFF# = 'HQEDM' OR OFFICE_DEPT.OFF# = 'HQH1F')
```

Target Attr: EMP.EMP#, EMP.POSITION_C, EMP.PAY_RATE,
EMP.EXPERIENCE_C

Query 10:

"The relation JOB_MANPOWER lists the position requirements for each job. For each position requirement for jobs under the management of the MONAXW office, find employees of the MONREAL office that are qualified to fill it."

```
(JOB_MANPOWER.POSITION_C = EMP.POSITION_C ) AND
(EMP.EMP#      = DEPT_EMP.EMP#      ) AND
(DEPT_EMP.DEPT# = OFFICE_DEPT.DEPT# ) AND
(OFFICE_DEPT.OFF# = 'MONR1' ) AND
(JOB_MANPOWER.JOB# = JOB_OFFICE.JOB# ) AND
(JOB_OFFICE.OFF# = 'MONX2' )
```

Target Attr: DEPT_EMP.EMP#, EMP.POSITION_C, JOB_OFFICE.JOB#

Query 11:

"Find all incomplete purchase orders for a job with a job# = 'job12'. For each purchase order, list the PO#, supplier # the supplier address."

```
(SUPPLY_ORDER.JOB# = 'JOB12' ) AND
(SUPPLY_ORDER.PO# = PO_SUPPLIER.PO# ) AND
(PO_SUPPLIER.PO_STATUS_C = 'INC' ) AND
(PO_SUPPLIER.S# = SUPPLIER.S# )
```

Target Attr: SUPPLY_ORDER.PO#, SUPPLIER.S#,
SUPPLIER.ADDRESS

Query 12: "Find all items on purchase orders issued to suppliers with a status code 'slw' (slow). List the supplier #, PO# and item #s."

```
(SUPPLIER.STATUS_C = 'slw' ) AND
(PO_SUPPLIER.S# = SUPPLIER.S# ) AND
```

(PO.PO# - PO_SUPPLIER.PO#)

Target Attr: SUPPLIER.S#, PO.ITEM#, PO.PO#

Query 13:

"Find all items on purchase orders issued to suppliers with a status code 'slw'. List the supplier #, PO#, item #s and also job #s for which the purchase orders were made."

(SUPPLIER.STATUS_C = 'slw') AND
 (PO_SUPPLIER.S# = SUPPLIER.S#) AND
 (PO.PO# = PO_SUPPLIER.PO#) AND
 (PO.PO# = SUPPLY_ORDER.PO#)

Target Attr: SUPPLIER.S#, PO.ITEM#, SUPPLY_ORDER.PO#,
 SUPPLY_ORDER.JOB#

Query 14:

"List numbers and status for suppliers who were issued purchase orders for any jobs managed by office MONAIC."

(JOB_OFFICE.OFF# = 'MONA1') AND
 (JOB_OFFICE.JOB# = SUPPLY_ORDER.JOB#) AND
 (SUPPLY_ORDER.PO# = PO_SUPPLIER.PO#) AND
 (PO_SUPPLIER.SUPPLIER# = SUPPLIER.SUPPLIER#)

Target Attr: SUPPLIER.S#, SUPPLIER.STATUS_C, PO_SUPPLIER.PO#

Query 15:

"Find the quantity of items ordered and received for all incomplete purchase orders (ie., purchase orders with SUPPLY_ORDER.PO STATUS_C = 'inc') issued for the job with a job# 'job12'. List the PO#, item #, supplier # and supplier address."

(SUPPLY_ORDER.JOB# = 'job12') AND
 (SUPPLY_ORDER.PO# = PO_SUPPLIER.PO#) AND
 (PO_SUPPLIER.SUPPLIER# = SUPPLIER.SUPPLIER#) AND
 (PO_SUPPLIER.PO STATUS_C = 'inc') AND
 (SUPPLY_ORDER.PO# = PO.PO#) AND
 (PO.PO# = PO_RCVD.PO#) AND
 (PO.ITEM# = PO_RCVD.ITEM#) AND
 (PO.QTY < PO_RCVD.QTY)

Target Attr: SUPPLIER.SUPPLIER#, SUPPLY_ORDER.PO#,
 PO.ITEM#, PO.QTY, PO_RCVD.QTY

Query 16:

"For each job list the quantity of items ordered, item price, supplier # and the purchase order date."

(SUPPLY_ORDER.PO# = PO.PO#)

Target Attr: SUPPLY_ORDER.JOB#, PO.PO#, PO.ITEM#, PO.QTY,
PO.ORDER_DATE, PO.QTY, PO.PRICE

Query 17:

"For each item required for a job with a # 'job12', list suppliers who supply that item. List the supplier #, name and address."

(JOB_MATERIAL.JOB# = 'job12') AND
(JOB_MATERIAL.ITEM# = SUPPLY.ITEM#) AND
(SUPPLY.SUPPLIER# = SUPPLIER.SUPPLIER#)

Target Attr: JOB_MATERIAL.ITEM#, SUPPLIER.SUPPLIER#,
SUPPLIER.SUP_NAME, SUPPLIER.ADDRESS

Query 18:

"For each job, performed by the firm MONAX for a customer with a customer # = 'cust1', list manpower requirements."

(JOB_CUSTOMER.CUST# = 'cust1') AND
(JOB_CUSTOMER.JOB# = JOB_OFFICE.JOB#) AND
(JOB_OFFICE.OFF# = 'monx1' OR JOB_OFFICE.OFF# = 'monx2') AND
(JOB_OFFICE.JOB# = JOB_MANPOWER.JOB#)

Target Attr: JOB_OFFICE.JOB#, JOB_MANPOWER.PHASE_C,
JOB_MANPOWER.JOB_POS_C, JOB_MANPOWER.HOURS,
JOB_MANPOWER.RATE_CHARGED

Query 19:

"For each job, performed by the firm MONAX for a customer with a customer # = 'cust1', list item requirements."

(JOB_CUSTOMER.CUST# = 'cust1') AND
(JOB_CUSTOMER.JOB# = JOB_OFFICE.JOB#) AND
(JOB_OFFICE.OFF# = 'monx1' OR JOB_OFFICE.OFF# = 'monx2') AND
(JOB_OFFICE.JOB# = JOB_MATERIAL.JOB#)

Target Attr: JOB_OFFICE.JOB#, JOB_MATERIAL.PHASE_C,
JOB_MATERIAL.ITEM#, JOB_MATERIAL.QTY,
JOB_MATERIAL.EXP_PRICE

Query 20:

"List all jobs for all customers. Print customer #, job # and location, and the office number managing the job."

(JOB_CUSTOMER.JOB# = JOB_OFFICE.JOB#)

Target Attr: JOB_OFFICE.JOB#, JOB_CUSTOMER.CUST#,
JOB_CUSTOMER.JOB_LOCATION

Query 21:

"List all jobs for the firm MONREAL. Print customer #, job # and location. Also print the customer name and address."

```
(JOB_CUSTOMER.JOB#      = JOB_OFFICE.JOB#) AND  
(JOB_OFFICE.OFF#        = 'monr1'      ) AND  
(JOB_CUSTOMER.CUST#     = CUSTOMER.CUST# )
```

Target Attr: JOB_OFFICE.JOB#, JOB_CUSTOMER.JOB LOCATION,
CUSTOMER.CUST#, CUSTOMER.CUST_NAME,
CUSTOMER.ADDRESS

APPENDIX F

SAMPLE INPUT AND OUTPUT

The appendix contains a sample input and output for the software modelling the application of a DDB described in Appendix E. First, a sample of parameters input to specify the distributed environment and also input for a sample query is shown. A sample output for the one sample query follows. Finally, a portion of output for statistics over all queries concludes this appendix.

A sample input is provided for the specification of relations, network parameters and a simple query which also appears as "Query 20" in Appendix E. Figure F-1 shows the input file containing the distributed environment specification. Each line may contain some explanation on its content by use of a comment. Comments begin by "--".

The first 5 lines shown in the figure are indigenous to the software. The first three lines state that input of the network environment, and relation and query specifications should be performed.

The fourth line states that there are 21 queries to be input while the next line lists the numbers of queries which should be modelled.

Next come the number of network nodes, network nodes names, the number of information processors and their names. Following is then the network dollar cost and delay (per byte of transferred data) for each pair of network nodes. The last 5 lines specify the dollar cost and delay for processing a unit of data (byte) for each information processor.

Figure F-2 shows a portion of the file containing the specification of relations and attributes. Note that only the two relations which are necessary for the sample query are listed. As before, each line may contain a comment which begins with "--". Both relations "JOB_CUSTOMER" and "JOB_OFFICE" are fragmented horizontally as specified in Appendix D.

Figure F-3 shows a portion of the input file containing the specification of queries. First, the query is described and its conjunctive normal form is shown using comments. Next, "20 2" is the query number and the number of the global view relations it references. The next two lines contain the names of the two referenced relations.

Next comes the specification of the one and two-variable terms. There are no one-variable terms in this query as specified by input "0". The input of two-variable terms starts with a "1" which states that there is only one two-variable term to be input. The term is input on one line. The line starts with the name of the relation and its joining attribute "JOB_CUSTOMER JOB#". This is followed by the second relation and its joining attribute, "JOB_OFFICE JOB#". The line ends with a "0.1" which is the selectivity factor used in the size estimation for the result of a join which is used to process this two-variable term. The attribute "JOB#" is the primary key for each of the two relations and as there are 10 active jobs, the cardinality of each relation is also 10. The selectivity factor is "0.1" which means that the cardinality of the join result is "Card(JOB_CUSTOMER) times Card(JOB_OFFICE) times 0.1 = 10". For each value of attribute "JOB#" in the relation "JOB_CUSTOMER" there is exactly one identical value of attribute "JOB#" in the relation "JOB_OFFICE"; consequently, the cardinality of the join result is 10.

Finally, following the specification of two-variable terms, target attributes are identified. For this query there are three target attributes and each is listed on a separate line using a format

"relation_name attribute_name".

Figure F-4 shows a section of the input file which contains various control parameters for printing and debugging purposes. Description of their use is beyond the scope of this discussion.

Output of modelling the previously shown, sample query, is shown in Figure F-5. Note that the figure shows some of the results for only modelling this query. A sample of the output of statistics over all 21 queries is provided in Figure F-6. Note that this figure is not provided for discussion of modelling results but only as a sample of output produced by the modelling software.

F-4

Figure F-1 Input of parameters for the network and information processors specification

```

JOB_CUSTOMER      4      10.0  ~ relation name, number of attributes, cardinality
JOB#              5.0      ~ attribute name and width
CUST#             5.0      ~ attribute name and width
JOB_NAME          15.0
JOB_LOCATION      15.0
true             ~ fragmented horizontally
2               ~ # of fragments
MONAXC
MONREAL

JOB_OFFICE        3      10.0  ~ relation name, number of attributes, cardinality
JOB#              5.0      ~ attribute name and width
OFF#              3.0
HQOFF#           3.0
true             ~ fragmented horizontally
2               ~ # of fragments
MONAXC
MONREAL

~ info. proc. storing the frag. and cardinality
3.0 ~ info. proc. storing the frag. and cardinality

7.0 ~ info. proc. storing the frag. and cardinality
3.0 ~ info. proc. storing the frag. and cardinality

~ info. proc. storing the frag. and cardinality
7.0 ~ info. proc. storing the frag. and cardinality
3.0 ~ info. proc. storing the frag. and cardinality

```

Figure F-2 A sample portion of the input file containing specification of relations

```

-- QUERY 20:
-- "List all jobs for all customers. Print customer #, job # and location,
-- and the office number managing the job."
-- (JOB_CUSTOMER.JOB# = JOB_OFFICE.JOB#)
-- Target Attr: JOB_OFFICE.JOB#, JOB_CUSTOMER.CUST#, JOB_CUSTOMER.JOB_LOCATION
JOB_CUSTOMER JOB# 2 ~ query id, # of referenced relations
JOB_OFFICE ~ name of referenced relation
0 ~ name of referenced relation
0 ~ # of one-variable terms
1 JOB_CUSTOMER JOB# ~ of two-variable terms JOB# 0.1
JOB_OFFICE JOB# ~ # of attr target attributes
JOB_CUSTOMER CUST# ~ target attribute: rel. name, attr. name
JOB_CUSTOMER JOB_LOCATION ~ target attribute: rel. name, attr. name
--
-- QUERY 21:
-- "List all jobs for the firm MCNREAL. Print customer #, job # and location.
-- Also print the customer name and address."

```

Figure F-3 A portion of the input file with the specification of queries

[illegible]

REFERENCES

AKOKA (1980)

Akoka, J., "Design of Optimal Distributed Database Systems", Distributed Data Bases, North Holland Publishing Company, 1980, 229-245.

APERS (1983)

Apers, P. M. G., Hevner, A. R., Bing Yao, S., "Optimization Algorithms for Distributed Queries", IEEE TSE, Vol. SE-9, No. 1, January 1983, pp. 57-68.

BERNSTEIN (1977)

Bernstein, P., A., et. al., "Analysis of Serializability in SDD-1", Proc. IEEE Comp. Software and Appl. Conf., Chicago, Nov. 1977, 579-585.

BERNSTEIN (1978)

Bernstein, P., A., et al., "Formal Aspects of Serializability in Data Base Concurrency Control", IEEE TOSE, Vol. SE-4, No. 3, May 1978, 203-216.

BERNSTEIN (1979)

Bernstein, P., A., "Approaches to Concurrency Control In Distributed Data Base Systems", AFIPS, 1979, 813-820.

BERNSTEIN (1981)

Bernstein, P. A., et. al., "Query Processing in a System for Distributed Databases (SDD-1)", ACM TDS, Vol. 6, No. 4, Dec. 1981, 602-625.

BERNSTEIN (1983a)

Bernstein, P. A., et. al., "Analyzing Concurrency Control Algorithms when User and System Operations Differ", IEEE TSE, Vol. SE-9, No. 3, May 1983, pp. 223-239.

BERNSTEIN (1983b)

Bernstein, P. A., Goodman, N., "Multiversion Concurrency Control-Theory and Algorithms", ACM TODS, Vol. 8, No. 4, December 1983, pp. 465-483.

BHARGAVA (1982)

Bhargava, B., "Performance Evaluation of the Optimistic Approach to Distributed Database Systems and its Comparison to Locking", 3rd Int. Conf. on Distributed Computing Systems, Florida, Oct. 1982, 508-517.

BHARGAVA (1983)

Bhargava, B., Hua, C. T., "A Causal Model for Analyzing Distributed Concurrency Control Algorithms", IEEE TSE, Vol. SE-9, No. 4, July 1983, pp. 470-486.

BLACK (1982)

Black, P. and Luk, W. S., "A New Heuristic for Generating Semi-Join Programs for Distributed Query Processing", Proc. of COMPSAC 82, IEEE Computing Society's 6th International Conf., 1982, 581-588.

CARIE (1983)

Carie, M. J., "An Abstract Model of Database Concurrency Control Algorithms", SIGMOD 83, San Jose, May 1983, pp. 97-107.

CASSEY (1972)

Cassey, R. G., "Allocation of Copies of a File in an Information Network", AFIPS 1972, Vol. 1, 617-625.

CELLARY (1980)

Cellary, W., and Meyer, D., "A Multi-Query Approach to Distributed Processing in Relational DBMS", Distributed Data Bases, North-Holland Publishing Company, 1980, 99-119.

CERI (1983)

Ceri, S. and Pelagatti, G., "Correctness of Query Execution Strategies in Distributed Databases", ACM, TODS, Vol. 8, No. 4, Dec. 1983, 577-607.

CERI (1985)

Ceri, S. and Gottlob, G., "Translating SQL into Relational Algebra: Optimization, Semantics, and Equivalence of SQL Queries", IEEE TOSE, Vol. SE-11, No. 4, April 1985, 324-332.

CHANDRA (1976)

Chandra, A. K., Merlin, P. M., "Optimal Implementation of Conjunctive Queries in Relational Data Bases", Proc. of the 9th Annual Symposium on the Theory of Computing, 77-90.

CHANDY (1983)

Chandy, K. M., et. al., "Distributed Deadlock Detection", ACM TODS, Vol. 1, No. 2, May 1983, pp. 144-156.

CHANG (1980)

Chang, S-K. and Cheng, W-H., "A Methodology for Structured Database Decomposition", IEEE TOSE, March 1980, Vol. SE-6, NO. 2, 205-218.

CHEN (1980)

Chen, P. S. and Akoka, J., "Optimal Design of Distributed Information Systems", IEEE Trans. on Computers, Dec. 1980, Vol. C-29, No. 12, 1068-1078.

CHEN (1983)

Chen, L. P. and Li, O. K., "Properties of Optimal Semi-join Programs for Distributed Query Processing", 0730-3157/83/00Q0/0467, IEEE, 1983, 476-483.

CHRISTODOULAKIS (1983)

Christodoulakis, S., "Estimating Record Selectivities", Information Systems, Vol. 8, No. 2, 1983, 105-116.

CHRISTODOULAKIS (1984a)

Christodoulakis, S., "Estimating Block Selectivities", Information Systems, Vol. 9, No. 1, 1984, 69-80.

CHRISTODOULAKIS (1984b)

Christodoulakis, S., "Implications of Certain Assumptions in Database Performance Evaluation", ACM TODS, Vol. 9, No. 2, June 1984, 193-186.

CHU (1979)

Chu, W. W. and Hurley, P., "A Model for Optimal Query Processing for Distributed Data Bases", CG1393-8/79, 1979 IEEE, 116-122.

CHU (1982)

Chu, W. W. and Hurley, P., "Optimal Query Processing for Distributed Database Systems", IEEE Trans. on Computers, Vol. C-31, No. 9, Sept. 1982, 835-850.

CODD (1971)

Codd, E. F., "A Database Sublanguage Founded on the Relational Calculus", Proc. ACM SIGFIDET Workshop on Data Description, Access and Control, New York, 1971.

COOPER (1977)

Cooper, L., Bhat, U. N., LeBlanc, L. J., "Introduction to Operations Research", Saunders Publ. Comp., 1977.

DANIELL (1983)

Daniell, T. P., Harding, R. C., Nauckhoff, S. H., "Dynamic Replication, an Overview", AFIPS, Proc. of 1983 NCC, 219-228.

DANIELS (1982)

Daniels, et al., "An Introduction to Distributed Query Compilation in R", Distributed Data Bases, Schneider, H. J., editor, North Holland, 1982, 247-290.

DATE (1982)

Date, C. J., "An Introduction to Database Systems", Addison-Wesley, 1982.

DECITRE (1981)

Decitre, P., "A Concurrency Control Algorithm in a Distributed Environment", AFIPS, Vol. 50, 1981, 473-479.

DEMOLOMBE (1980)

Demolombe, R., "Estimation of the Number of Tuples Satisfying a Query Expressed in A Predicate Calculus Language", 6th Conf. on Very Large Databases, Montreal, Quebec, Oct. 1980, 55-63.

EPSTEIN (1978)

Epstein, R., et. al., "Distributed Query Processing in a Relational Data Base System", ACM-SIGMOD, Proc. International Conf. on Management of Data, Austin, Texas, 1978, 169-180.

EPSTEIN (1980)

Epstein, R. and Stonebraker, M., "Analysis of Distributed Data Base Processing Strategies", 6th International Conf. on Very Large Data Bases, Montreal, Quebec, 1980, 92-101.

ESWARAN (1974)

Eswaran, K. P., "Replacement of Records in a File and File Allocation in a Computer Network", IFIP, 1974.

ESWARAN (1976)

Eswaran, K., P., et. al., "The notions of Consistency and Predicate Locks in a Database System", CACM, Nov. 1976, Vol. 19, No. 11, 624-634.

GARDY (1984)

Gardy, D. and Puech, C., "On the Sizes of Projections: A Generating Function Approach", Inform. Systems, Vol. 9, No. 3/4, 1984, 231-235.

GAVISH (1982)

Gavish, B. and Segev, A., "Query Optimization in Distributed Computer Systems", Management of Distributed Data Processing, North Holland Publ. Comp., 1982, 233-252.

GIFFIN (1971)

Giffin, W. C., "Introduction to Operations Engineering", Irving-Dorsey Ltd., 1971.

GOUDA (1981)

Gouda, M. G., Dayal, U., "Optimal Semi-join Schedules for Query Processing in Local Distributed Database Systems", ACM SIGMOD, May 1981, 164-175.

HAMMER (1980)

Hammer, M. M., Shipman, D. W., "Reliability Mechanisms for SDD-1: A System for Distributed Databases", ACM Trans. on DB, Vol. 5, No. 4, December 1980.

HEVNER (1979)

Hevner, A. R. and Yao, S. B., "Query Processing in Distributed Data Base Systems", IEEE TOSE, Vol. SE-5., No. 3, May 1979, 177-187.

HEVNER (1980)

Hevner, A. R., "Query Optimization in Distributed Database Systems", Ph.D. thesis, Univ. of Minnesota, 1980.

HOROWITZ (1978)

Horowitz, E. and Sahni, S., "Fundamentals of Computer Algorithms", Computer Science Press, 1978.

IBARAKI (1984)

Ibaraki, T. and Kameda, T., "On the Optimal Nesting Order for Computing N-Relational Joins", ACM TODS, Vol. 9, No. 3, Sept. 1984, 482-502.

KAMBAYASHI (1983)

Kambayashi, Y., Yoshikawa, M., "Query processing Utilizing Dependencies and Horizontal Decomposition", ACM SIGMOD, May 1983, 55-67.

KAMOUN (1981)

Kamoun, F., Kleinrock, L., Muntz, R., "Queueing Analysis of the Ordering Issue in a Distributed Database Concurrency Control Mechanism", Proc. of 2nd Int. Conf. of Distributed Computing Systems, Paris, France, April 1981, 13-23.

KERSCHBERG (1982)

Kerschberg, L., Ting, P. D. and Yao, B., "Query Optimization in Star Computer Networks", ACM TDS, Vol. 7, No. 4, Dec. 1982, 678-711.

KRISHNAMURTHY (1984)

Krishnamurthy, R. and Morgan, S. P., "Distributed Query Optimization: An Engineering Approach", Proc. of the First Int. Conf. on Data Engineering, April 24-27, 1984, Los Angeles, California, USA, 220-227.

LAUSEN (1983)

Lausen, G., "Formal Aspects of Optimistic Concurrency Control in a Multiple Version Database System", Information Systems, Vol. 8, No. 4, 1983, pp. 291-302.

Le LANN (1977)

Le Lann, G., "Distributed Systems - Towards a Formal Approach", IFIP 1977, 155-165.

Le LANN (1978)

Le Lann, G., "Algorithms for Distributed Data Sharing Systems which Use Tickets", Proc. of the 3rd Berkeley Workshop on Data Management and Computer Networks, August 1978, 259-272.

Le LANN (1980)

Le Lann, G., "Consistency, Synchronization and Concurrency Control", Distributed Data Bases, Cambridge University Press, 1980, 195-222.

LIN (1982)

Lin, Wen-Te, K. and Nelte, J., "Communication Delay and Two-phase Locking", Proc. of the 3rd Int. Conf. on Distributed Computing Systems, Florida, Oct., 1982, 502-507.

MAHMOUD (1976)

Mahmoud, S. A., Riordon, J. S., "Optimal Allocation of Resources in Distributed Information Systems", ACM TODS, Vol. 1, March 1976, pp. 66-78.

MAHMOUD (1979)

Mahmoud, S. A., Riordon, J. S. and Toth, K. C., "Distributed Database Partitioning and Query Processing", IFIP-TC-2, Venice, Italy, 1979, 32-51.

MAIER (1983)

Maier, D., Ullman, J. D., "Fragments of Relations", ACM SIGMOD, May 1983, 15-22.

MENASCE (1979)

Menasce, D., Muntz, R., R., "Locking and Deadlock Detection in Distributed Data Bases", IEEE TOSE, Vol. SE-5, No. 3, May 1979, 195-202.

MORGAN (1977)

Morgan, H. L. and Levin, K. D., "Optimal Program and Data Locations in Computer Networks", Communications of the ACM, May 1977, Vol. 2, No. 5, 315-322.

OBERMARCK (1981)

Obermarck, R., Beerli, C., "A Resource Class Independent Deadlock Detection Algorithm", 7th Int. Conf. on VLDB, Cannes, France, Sept. 1981, 166-178.

PAIK (1979)

Paik, I-S and Delobel, C., "A Strategy for Optimization of Distributed Query Processing", Proc. 1st Int. Conf. on Distributed Computing Systems, Huntsville, Alabama, Oct. 1979, 686-698.

PELAGATTI (1979)

Pelagatti, G. and Schreiber, F. A., "Evaluation of Transmission Requirements in Distributed Database Access", ACM-SIGMOD 1979, Int. Conf. on Management of Data, May 1979, Boston, Mass., 102-108.

PERIZZO (84)

Perizzon, W., "A Method for Processing Distributed Database Queries", IEEE TSE, Vol. SE-10, No. 4, July 1984, 466-471.

PRAMANIK (1985)

Pramanik, S. and Ittner, D., "Use of Graph Theoretic Models for Optimal Relational Database Accesses to Perform Join", ACM TODS, Vol. 10, No. 1, March 1985, 58-74.

RAMAMORTHY (1980)

Ramamorthy, C. W. and Wah, W. W., "The File Allocation Problem Revisited", University of California, 84720, Berkeley, California, 1980.

REINGOLD (1977) see page F-9

RIORDON (1982)

Riordon, J. S., "Chairman's Report: Resource Allocation and Query Processing in Distributed Databases", Distributed Data Sharing Systems, North Holland Publishing Company, 1982, 95-100.

REYNOLDS (1977)

Reynolds, C. H., "Issues in Centralization", Datamation, March 1977.

ROCKART (1976)

Rockart, J. F. and Leventer, J. S., "Centralization vs. Decentralization", Working Paper WP-845-76, April 1976, MIT Sloan School of Management, Cambridge, Mass., USA.

ROTHNIE (1977)

Rothnie, J., B., Goodman, N., Bernstein, P., A., "The Redundant Update Methodology of SDD-1", Proc. of IEEE Comp. Software and Applications Conf., Chicago, Nov. 1977, 586-593.

ROSENKRANTZ (1978)

Rosenkrantz, D., J., et al., "System Level Concurrency Control for Distributed Database Systems", ACM TODS, Vol.3, No.2, June 1978.

SKEEN (1983)

Skeen, D., Stonbraker, M. R., "A Formal Model of Crash Recovery in a Distributed System", IEEE, TSE, Vol. SE-9, No. 3, May 1983, pp. 219-228.

SELINGER (1980)

Selinger, P. G. and Adiba, M., "Access Path Selection in Distributed Data Base Management Systems", Proc. of the First Int. Conf. on Data Bases, Aberdeen, 1980.

SMITH (1975)

Smith, J. M. and Chang, P. Y., "Optimizing the Performance of a Relational Algebra Database Interface", CACM, Vol. 18, No. 10, Oct. 1975, 568-579.

STANKOVIC (1982)

Stankovic, J., A., et. al., "An Evaluation of the Applicability of different mathematical approaches to the analysis of decentralized control algorithms", COMPSAC 82, pp. 62-69.

STONEBRAKER (1976)

Stonebraker, M. and Neuhold, E., "A Distributed Database Version of Ingres", Memorandum No. ERL-M612, Sept. 1976, Eng. Research Lab., University of California, Berkeley, California.

STONEBRAKER (1979)

Stonebraker, M., "Concurrency Control and Consistency of Multiple Copies of Data in Distributed INGRES", IEEE TOSE, Vol. SE-5, No. 3, May 1979, 188-194.

THANOS (1982)

Thanos, C., et. al., "Synchronization Techniques Based on Two-phase Locking", COMPSAC 82, 358-367.

THOMAS (1978)

Thomas, R., H., "A Solution to the Concurrency Control Problem for Multiple Copy Databases", Proc. CONPCOM 1978, IEEE, N.Y.

TOAN (1979)

Toan, N. G., "A Unified Method for Query Decomposition and Shared Information Updating in Distributed Systems", First Int. Conf. on Distributed Computer Systems, Huntsville, Alabama, Oct. 1979, 679-685.

TOAN (1981)

Toan, N. G., "Distributed Query Management for a Local Network", Proc. 2nd Int. Conf. on Distributed Computing Systems, Paris, France, April 1981, 188-196.

TOTH (1980)

Toth, K. C., "Distributed Database Architecture and Query Processing", Ph. D. Dissertation, Carleton University, Ottawa, Ontario, 1980.

TOTH (1982)

Toth, K.C., Mahmoud, S. A., Riordon, J. S., "Query Processing Strategies in a Distributed Database Architecture", Distributed Data Sharing Systems, North Holland Publ. Comp., 1982, 117-134.

ULLMAN (1982)

Ullman, J. D., "Principles of Database Systems", Computer Science Press, 1982, 2nd edition.

VALDURIEZ (1984)

Valduriez, P., Gardarin, G., "Join and Semijoin Algorithms for a Multiprocessor Database Machine", ACM TODS, Vol. 9, NO. 1., March 1984, 133-156.

WILLARD (1984)

Willard, D., "Efficient Processing of Relational Calculus Expression Using Range Query Theory", SIGMOD 84, Proc. of the Annual Meeting, Boston, Ma., June 18-21, SIGMOD Record Vol. 14, No. 2, 164-175.

WILMS (1980)

Wilms, P., "Qualitative and Quantitative Comparison of Update Algorithms in Distributed Databases", Distributed Data Bases, North-Holland, 1980, 275-294.

WONG (1976)

Wong, E. and Youssefi, K., "Decomposition - A Strategy for Query Processing", ACM TDS, Vol. 1., No. 3., Sept. 1976, 223-241.

WONG (1982)

Wong, E., "A Statistical Approach to Incomplete Information in Database Systems", ACM TDS, Vol. 7, No. 3, Sept. 1982, 470-488.

WONG (1983)

Wong E., Katz, R. H., "Distributing a Database for Parallelism", SIGMOD 83, San Jose, May 1983, pp. 23-29.

YOUSSEFI (1979)

Youssefi, K. and Wong E., "Query Processing in a Relational Database Management System", IEEE, CH1406-8/79, 407-417.

YAO (1979)

Yao, S. B., "Optimization of Query Evaluation Algorithms", ACM TDS, Vol. 4, No. 2, June 1979, 133-155.

YAO (1983)

Yao, S. B., 15-th ACM STOC, (1983), 258-264.

YU (1982a)

Yu, C. T., Chang, C. C., Chang, Y. C., "Two Surprising Results in Processing Simple Queries in Distributed Databases", IEEE COMPSAC, Nov. 1982, 377-384.

YU (1982b)

Yu, C. T., et. al., "A Promising Approach to Distributed Query Processing", Berkeley Conf. on Distr. DBs, Feb. 1982, 363-390.

YU (1983)

Yu, C. T., Chang, C. C., "On the Design of a Query Processing Strategy in a Distributed Database Environment", ACM SIGMOD, May 1983, 30-39.

REINGOLD (1977)

Reingold, E., "Combinatorial Algorithms: Theory and Practice", Prentice-Hall, 1977, QA164.R43.

END

0.9 | 0.4 | 8.6

FIN