THE UNIVERSITY OF MICHIGAN

COMPUTING RESEARCH LABORATORY

THE ALLOCATION OF DATA AND PROGRAMS IN DISTRIBUTED DATA PROCESSING ENVIRONMENTS

Amjad Umar

CRL-TR-22-84

MARCH 1984

Room 1079, East Engineering Building
Ann Arbor, Michigan 48109
USA
Tel: (313) 763-8000

THE ALLOCATION OF DATA AND PROGRAMS

IN DISTRIBUTED DATA PROCESSING ENVIRONMENTS

by

Amjad Umar

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
(Industrial and Operations Engineering)
in The University of Michigan
1984

Doctoral Committee:

Associate Professor Toby J. Teorey, Chairman
Assistant Professor John R. Birge
Professor Keki B. Irani
Associate Professor Devinder S. Kochhar

Dedicated to My Father

ACKNOWLEDGEMENTS

I wish to express my deepest gratitude to Professor Teorey for his guidance, encouragement, critique, and patience throughout this research. Despite his busy schedule, he spent a great deal of time reviewing and analyzing several versions of this dissertation. This work could have never been materialized without his help. I also wish to thank Professor Birge for introducing me to the analytical analysis of heuristics, an interesting research area. His review of optimization and queuing models is also greatly appreciated. Professor Irani's analysis of my derivations and the ensueing discussions have been of great help. Professor Kochhar's critique of the overall dissertation is also greatly appreciated. Last but not least, I am indebted to my wife Dolorese for her patience and understanding throughout this research.

PREFACE

A generalized model and algorithm for the allocation of programs and data to a set of interconnected computers is presented. This problem is of key importance in the design and operation of several computing systems, commonly termed distributed data processing environments, in which a user at one computer can access, in addition to the data at the local computer, the data and computing facilities of remotely located computers. Simply stated, this problem is concerned with the allocation of P programs and D data files to N computers which minimizes the cost of storing and accessing the data while still satisfying the access time and availability requirements of a diverse set of users. This problem is complicated by several management and technical factors which take into account the policies, capacities and interfaces of the environment.

Currently available techniques concentrate on optimization by ignoring a large number of complex but important factors in real systems. This dissertation first presents a simplified model which is systematically extended to include the complex technical and management issues of transaction processing models, update synchronization algorithms, data communication options (network topologies and routing algorithms), queuing considerations for local and network physical resources, and software development costs. The extensions are synthesized into a generalized allocation algorithm which yields near-optimal allocations. This algorithm integrates a large number of techniques including clustering, greedy search routine, decomposition, hierarchical heuristics, spatial dynamic programming, and distributed optimization into a single procedure.

The model is applicable in a variety of ways. First, it demonstrates that several realistically complex program and data allocation problems

can be described and solved efficiently, in polynomial time. Second, the model shows quantitavely the combined effect of several key factors (program/data relationships, update synchronization algorithms, transaction processing models, network configurations etc) on the allocation of programs and datasets, and thus provides insights regarding interdependencies in distributed data processing systems. Third, it is shown that several existing allocation schemes can be considered as special cases and/or transformations of our model. Finally, due to its generality, the model can be used in the design of new application systems, the evaluation of existing centralized or distributed applications, and the effect of support system variations on various applications.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF APPENDICES

MAIN PARAMETERS AND TERMS

. $\alpha(d,i)$ = 1 if dataset d is allocated to node i, 0 otherwise.
This parameter is used as a decision variable.

. $\alpha_s(k,p,d,i)$ = 1 if node i is the site selected for dataset d from program p, whenever a transaction arrives at node k, 0 otherwise. This parameter is used to show the site selected for datasets.

. $\beta(p,i)$ = 1 if program p is allocated to node i, 0 otherwise
This parameter is used as a decision variable.

. $\beta_s(k,p,i)$ = 1 if node i is the site selected for program p, when a transaction arrives at node k
0 otherwise. This parameter is used to show the program site selection.

. $\delta(k,i)$ = a delta function = 1 if k=i, 0 otherwise

. $A_a(z)$ = the availability of communication link z

. $B_{tp}(i,j)$ = the network topology
= z, the communication link between nodes i and j
= 0 otherwise

. B_{nc} = the network control=1 (distributed), 0 (centralized).
This parameter shows if all the network control (starting of sessions, terminating sessions, traffic throttling) is centralized or distributed.

. $B_{rt}(x)$ = the routing frequencies used in the network for link x:
= 1 means that this link is the primary path
= 2 means that this link is a secondary path

. $C_L(i)$ = cost, in cents, per local service at node i.

. $C_R(i,j)$ = cost, in cents, per exchange of remote message between nodes i and j.

. $C_S(i)$ = cost, in cents, for storing 1 megabyte at node i.

. COPY = the maximum number of dataset copies allowed

. D = set of all the datasets in the application system

. D_n = number of datasets in the application system.

. D_c(d1,d2)= consistency relationship between dataset
d1 and d2; implies that d2 must
be updated whenever d1 is updated .

. D_s(d,x) = size (x=1) and occurrence (x=2) of
tuples in dataset d.

. E(x) = the mean service time of device x

. F_D(d,i) = 1 if dataset d can be allocated to node i, 0 otherwise
This is a policy restriction.

. F_p(p,i) = 1 if program p can be allocated to node i, 0 otherwise
This is a policy restriction.

. L(t,k,i) = local service requests executed at node i
due to arrival of transaction t at node k

. N = set of all the nodes (computers) in the DDP network

. P = set of all the programs in the application system

. P_n = number of programs in the application system

. P_r(p,d) = number of reads issued from program p to
dataset d, per input message.

. P_w(p,d) = number of updates issued from program p to
dataset d, per input message.

. P_c(p,d) = number of consistency updates issued from
program p to dataset d, per input message.

. P_I(p) = average number of input records read by program p

. P_o(p) = display messages written by program p, per input message

. P_m(p,p1)= No. of temporary records written by program p
that are to be received by by program p1. Note
that p and p1 can execute in parallel if P_m(p,p1)=0.

. Q_{rt} = the routing graph: parallel (1) or ring (0).
.Parallel routing graph means that the arrival node controls the

operation, i.e. sends/receives direct messages from program sites

Serial routing means that the transaction is circulated through the program sites.

. Q_{da} = the data access mode: remote access (1) or local access(0)

.Remote data access implies that a program at node i can access data from node j, even if i $\neq$ j.

.Local data access implies that program at node i can only access data at node i, i.e. programs and data exist at the same node

. R(t,k,i,j) = remote service requests exchanged between nodes i and j due to arrival of transaction t at node k

. S(x) = storage occupied at node x

. STOR(i) = storage capacity of node i

. SRCH(k,x) = the search strategy for the queries arriving at node k. x=1 shows the first node searched, x=2 is the second node etc.

. T = set of all the transactions in the application system

. T_n = total number of transaction types in the application system.

. T_e(t,p)= no. of times program p is executed per invocation of transaction t.

. T_a(t,k) = Mean arrival rate of transaction t at node k for a given observation period.

. U_{cp} = location of the commit-processor: distributed (1) or centralized (0). The commit processor performs all the duplicate and consistency related data.

. U_{lm} = location of locks/stamps: distributed (1) or centralized (0). The lock management performs all the locking unlocking in the network.

. U_{pg} = the amount of "piggybacking" in synchronization:

= 0 complete piggyback

= 1 partial piggyback (request or response)

= 2 no piggybacking - all locks/unlocks sent separetly

. U_{cr} = the locks/stamps are core resident (0) or not (1)

. U_{sl} = the lock/unlock requests are broadcast (0) or addressed (1). In case of broadcast, the messages are sent to all the nodes. In case of addressing, the messages are only sent to the nodes where the data to be locked/unlocked is kept.

CHAPTER 1
INTRODUCTION

1.1 Background

Advances in the computer and communication technologies have led to *distributed data processing (DDP) networks* which interconnect many computers to satisfy the data processing needs of users in several enterprises. Such systems are also referred to as computing networks, distributed computing systems, distributed computing networks, distributed systems, and the like. The major appeal of DDP networks is that a user can perform many functions locally, perhaps on an inexpensive computer, and use the facilities of remotely located computers for other functions. Bennet <BENN83a>[1] reports that DDP networks of varying capabilities have grown from 500 in 1973 to over 1800 in 1978, and are presently growing so rapidly that even an approximate count is difficult.DDP networks can be described in terms of monolithic and cooperative systems. Monolithic systems are constructed for a single oragnization for an explicit purpose. Examples of such systems are:

---

1. The bibliography is organized in 6 sections (section a through f). Within each section, the references are sorted in ascending sequence by author name. The reference BENN83a indicates that the paper is listed in section a of the bibliography.

1. Banking systems in which each branch office of a bank has its own computer for branch office work. However, the branch office computers are connected to the head office computer for the inter-branch and corporate work.
2. University computing systems, where each campus, or department, has its own computer. These computers may be interconnected so that the files and special facilities of some computers can be shared. For example, the statistics department may house an extensive array of statistical packages, which may be used by other departments on an "as needed" basis.
3. Inventory systems where each plant has its own computer to keep inventory of the local products with possible access to other plant computers about the products which are not available at the local plant.

In contrast to the monolithic systems, the cooperative systems are formed between different organizations to share information and computing resources. Examples of such systems are:

1. Alliances of several approximately equal partners such as the MERIT computer network between the University of Michigan, Wayne State University and Michigan State Univeristy. The North Carolina Triangle Universities Computing Center (TUCC) is another example.
2. Facilities introduced by legislation (such as the state network in New Jersey) to consolidate state computing facilities between different agencies. Similar facilities have been implemented in Michigan for school districts and are being expanded.
3. Special purpose systems constructed to acquire experience such as ARPANET for agencies participating in the U.S. Department of Defense Advanced Research Project (ARPA), and free enterprise fa-

cilities like Tymnet and Edunet where a user pays a fee to join a network.

It can be expected that a DDP network may consist of cooperative sytems between several organizations where some of the cooperating organizations may have internal monolithic DDP networks. While DDP systems appeal to many enterprises, as evidenced by their growth, the design and development of such systems presents several serious technical as well as management challenges. Examples of the main issues are: interfacing different type of computers, allocation of computing resources to various sites, design and implementation of algorithms to retrieve and maintain duplicate information, enforcements of standards, personnel, etc. Several authors have discussed various aspects of these, and other related, issues <ROTH77a, MART81a, ADIB78a, DOWN76a, FRY76a, KIMB75a, OKEE78a, ROME80a, TRIP80a>.

The focus of this research is the technical problem of allocating programs and data to the computers. This problem is of great significance in the design and operation of DDP networks due to the effect of these decisions on cost savings and user satisfaction in terms of response time and availability. Simply stated, we are interested in the study, formulation and solution of the following problem:

DETERMINE:

. Allocation of P programs to N computers (nodes)

. Allocation of D data files to N computers

TO MINIMIZE: cost of storing and accessing data and programs

SUBJECT TO:

. User requirements of response time (the time needed to access the information) and availability (the percentage of time the information can be accessed).

. Technical constraints of the DDP network, and

. Management constraints of policies, standards etc.

We wish to translate this problem into a mathematical programming problem and develop an algorithm to solve it. This is not a trivial task since so many different views and approaches can exist. For example, variants of this problem have been addressed by over twenty researchers with different objective functions, decision variables and constraints. We wish to develop a generalized problem statement. The objective of this chapter is to identify the generalized program and data allocation problem and to demonstrate that this problem has not been solved.

The basic systems and terms used in this dissertation are introduced and defined in section 1.2. In section 1.3, a framework is introduced to help define the objective function and constraints precisely by identifying the surrounding activities and systems which influence the program and data allocation decisions. This section also shows a simple example to illustrate the main tradeoffs of the program and data allocation decisions. The actual problem to be solved is stated and discussed in section 1.4, and the literature is reviewed in section 1.5 to show that this problem has not been addressed before. The research approach and outline of this dissertation are given in section 1.6.

1.2. Basic Definitions

The program and data allocation problem will be viewed in terms of the interactions between the following two principal systems:

- An *application system* which is a collection of user defined data *(database)* , software instructions *(programs)* to manipulate the user data and user instructions *(transactions)* to activate the programs. An application system may be a simple query or a large and complex system. An example of the application system is a student information system where all the student data comprises the database, the computer instructions which read and update the informa-

tion are the programs, and the user instructions (commands, jobs) to print reports are the transactions.

- A *support system* which provides and manages the resources needed by the application system. Support systems consist of a *hardware subsystem* (processors, disks, links) and *control subsystem* (operating systems, database managers, communication managers, control information) to manage the interactions between the application systems and hardware devices. Example of a support system is the University of Michigan computing system which consists of the hardware subsystem (AMDAHL computer, terminals), and the control subsystem (MTS, UMMPS). Note that this support system provides and manages resources for a very large number of statistical, instructional, word processing and research applications.

These two definitions allow us to introduce the following definition of DDP networks (for other definitions see <BAGL79a>):

- A *Distributed data processing (DDP) network* provides an environment in which components of an application system can be distributed to, and accessed from, various computers *(nodes)* without manual intervention.

This definition of DDP networks allows us to define the following important terms:

- An application system whose programs and data exist at more than one node is called a *distributed application system*. If, however, all the components of an application system exist at one node, then these applications are termed *centralized application systems*.
- A support system which provides all the hardware and software for DDP networks is called a *DDP support system*.

In order to illustrate these definitions, consider a DDP newtork at a three campus university which interconnects the computers housed at the three campuses. Then all the hardware and control subsystems define the DDP support system and the many applications used in the network (student information, payroll, accounting, etc) are the application systems. Let us assume that the programs and data of the student information system exist at the three campuses, but all other applications exist only at the central campus. In this case, the student information application is distributed and the payroll and accounting applications are centralized. Note that in a given DDP network, some applications can be distributed, while others can be centralized.

1.3 Framework for Defining Program and Data Allocation

To formulate the program and data allocation problem precisely, we need to define the context and the interfaces of the allocation decisions in the DDP network life cycle. These decisions occur whenever:

1. A new application is designed in a DDP
2. An existing application in a DDP is modified and expanded
3. A new DDP support system is installed
4. An existing DDP support system is modified

We contend that the design of new applications for given DDP support systems (case 1) can represent most practical situations. Case 2 is a special situation of case 1. The third and fourth cases can be considered as special situations of case 1, as long as a wide range of DDP support systems can be correctly represented in the formulation of case 1. For example, if a new computer was being considered for case 3, then this situation can be addressed by studying case 1 for each of the application system influenced by this change.

Our next step is to identify the main activities needed to design new applications in DDP (case 1) so that we can determine the interfaces of

the program and data allocation decision and thus precisely define the input and output variables. Figure 1.1 proposes the location of this decision in the application system design process (see section 1.3.1 for discussion). A simple example is used to illustrate the allocation problem in section 1.3.2.

1.3.1 Application System Design in DDP

Application design in DDP is closely related to known techniques in centralized data processing (CDP) systems, where the design process can be described in terms of several phases. Figure 1.1 shows a generalization of the traditional CDP design approaches to take into account the DDP considerations.

The *requirement formulation* and the *logical design* phases are concerned with the formulation and refinement of user functional requirements, are independent of the support system considerations and thus the same for CDP and DDP. Basically, requirement formulation consists of determining the long range and short range scope of the system, the reports to be generated, the frequency of the reports to be generated and the major data elements involved, while the logical design consists of building the conceptual data and program models which can produce the desired results. The *physical design phase* is concerned with developing implementable structures of the database and the programs. In a DDP environment, the physical design phase can be decomposed into the following two non-overlapping subphases <TEOR82e>:

. *Distributed Design* : The internode allocations and the interfaces of the application system components are determined in this subphase. This subphase is concerned with the "global" problems of subdividing programs and datasets into partitions and then allocating these partitions to the nodes. This subphase does not exist in CDP.

. *Local Design* : The detailed physical design of the assigned components is conducted to determine:a) the database design which consists of record structure design, record clustering, access method design and index selection steps and b) the program design which consists of decomposing programs into modules, developing the flow and sequence of the modules and determining the conventions for program input/output and parameter exchange.

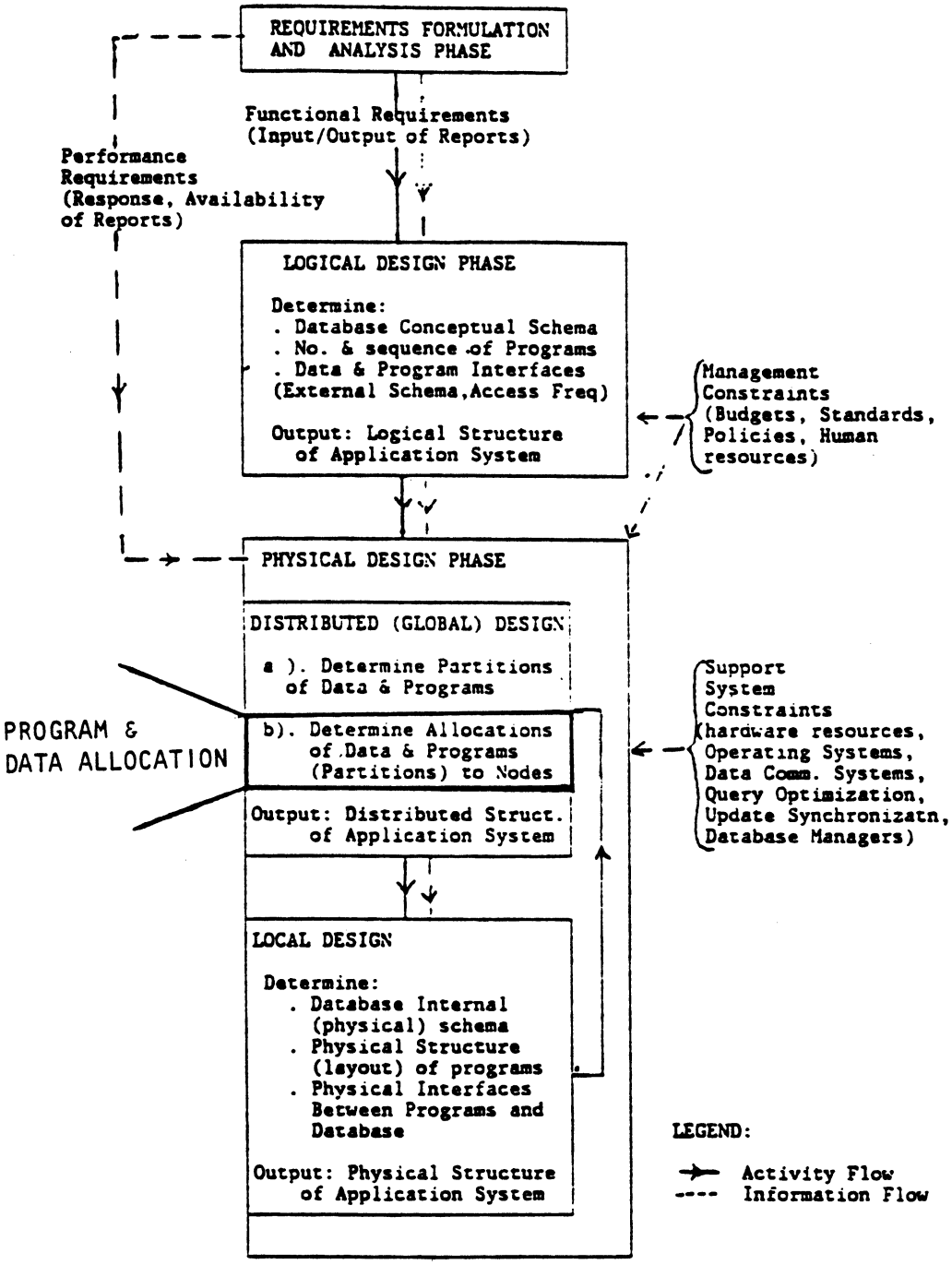
We exclusively concentrate on the distributed design subphase, since this subphase represents the differential of activities between CDP and DDP. The following two major steps are performed in the said subphase, each resulting in a different view (structure) of the application system:

. PARTITIONING which addresses the problem of dimensionality in distributed systems <SMOL79c, PALM78c>. The problem of dimensionality is concerned with instances of data and programs that can be allocated to different nodes. The result of this decision is that the logical database produced by the logical design phase is decomposed into partitions.

. ALLOCATION which assigns the partitions produced by the partitioning step to nodes. The program and data allocation problem is addressed in this step. As can be seen from figure 1.1, this step receives the partitioned programs and data, and generates a distributed (allocated) structure of the application system which shows the node assignments for each program and data partition (see the example below for explanation). It can also be seen from figure 1.1 that this step is constrained by the user performance requirements, management constraints and support system constraints. A detailed formulation and discussion of this step is given in section 1.4.

Figure 1.1 shows that the output of the allocation step is used as an input to the local design phase in which the local database and program

FIG (1.1) : THE APPLICATION SYSTEM DESIGN IN DDP



NOTE: It is assumed that every (sub)phase receives the output from the previous (sub)phase in addition to other input.

design decisions are made for each node. This observation emphasizes the importance of the program and data allocation decisions since an error at this level can have serious cascading effects on subsequent design and implementation decisions.

1.3.2 Example of Program and Data Allocation

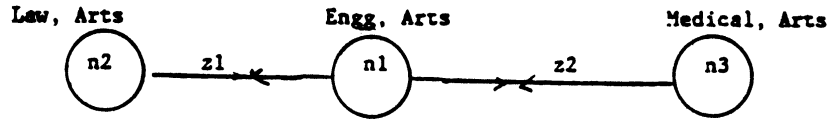
The program and data allocation within the context of application design in DDP can be illustrated through a student application design in a three campus university. The purpose of the application system is to provide information about student names, ids and schools being attended. Each campus houses a computer and the computers are interconnected, thus each campus can be viewed as a node. Let us designate these nodes as n1, n2, n3 where each campus has an arts school. In addition, n1 houses an engineering school, n2 houses a law school, and n3 houses a medical school. The support system environment is shown in figure 1.2a. The main question is: where should the student information be located? In other words, we need to know if information about all the students, irrespective of the school, should be allocated to all nodes, one node, or should the information be fragmented in some fashion? In order to answer these questions, let us follow the application design process shown in figure 1.1.

In the requirements analysis and design phases, the student information and the accessing programs are determined. Let us ignore the programs, for simplicity, and assume that the student information consists of student id, student name and school (see figure 1.2b). This information describes a logical view of data and shows the output of the logical design phase.

In the distributed design phase the student data is partitioned and allocated to the three nodes. Figure 1.2c shows the data partitioned in terms of the schools. Now the allocation step determines how the four

FIGURE 1.2: EXAMPLE OF APPLICATION DESIGN IN DDP

a). Support System Network



b). Logical View

STUDENT ID	STUDENT NAME	SCHOOL
302500369	Umar, A.	Engg
300762728	Duke, P.	Engg
900540983	Harris, T.	Engg
789283830	Burke, F.	Law
376890323	Peter, H.	Law
547689000	Garw, Y.	Medical
653637208	Wilson, F.	Medical
376399783	Jenkin, C.	Arts
012389000	Mach, W.	Arts
834902208	Levy S.	Arts

c). Partitioned Views(partitioning by school)

STUDENT ID	STUDENT NAME	SCHOOL
302500369	Umar, A.	Engg
300762728	Duke, P.	Engg
900540983	Harris, T.	Engg

STUDENT ID	STUDENT NAME	SCHOOL
789283830	Burke, F.	Law
376890323	Peter, H.	Law

STUDENT ID	STUDENT NAME	SCHOOL
547689000	Garw, Y.	Medica
653637208	Wilson, F.	Medica

STUDENT ID	STUDENT NAME	SCHOOL
376399783	Jenkin, C.	Arts
012389000	Mach, W.	Arts
834902208	Levy, S.	Arts

d). Distributed (Allocated) View

Three nodes (campusses). Node 1 offers engineering, node 2 offers law, node 3 offers medicine courses. The arts courses are offered at all campusses.

NODE 1

STUDENT ID	STUDENT NAME	SCHOOL
302500369	Umar, A.	Engg
300762728	Duke, P.	Engg
900540983	Harris, T.	Engg

NODE 2

STUDENT ID	STUDENT NAME	SCHOOL
789283830	Burke, F.	Law
376890323	Peter, H.	Law

NODE 3

STUDENT ID	STUDENT NAME	SCHOOL
547689000	Garw, Y.	Medica
653637208	Wilson, F.	Medica

NEED TO BE ALLOCATED
(single node, two nodes, three nodes)

STUDENT ID	STUDENT NAME	SCHOOL
376399783	Jenkin, C.	Arts
012389000	Mach, W.	Arts
834902208	Levy, S.	Arts

partitions, where each partition can be viewed as a data file, are to be allocated to three nodes. This problem is in practice quite difficult and is the main challenge of this research. We can assume that the student data of a given school is frequently referenced at the campus where the courses are offered. In this case, the ENGG partition can be allocated to node n1, LAW can be allocated to n2, etc (see figure 1.2d). Now we are left with the ARTS partition which can be allocated to all nodes, node n1 only, node n2 only etc. Let us review two extreme allocations:

1. ARTS is allocated to all the nodes (replicated). This allocation has the advantage that data can be quickly retrieved at nodes n1, n2 and n3, without any remote access (an access from n1 to n2, say). However, there are two potential disadvantages of this allocation: the storage cost of duplicating data can be high and the cost to update duplicate information (whenever information is updated at n1, it must be also updated at n2 and n3) may be high.
2. ARTS is allocated to one node, say n1. In this case the storage cost is minimal and there is no cost for duplicating data, however, every request for data at node n2 and n3 results in remote accesses which are usually slow and expensive.

It can be seen that tradeoffs exist for data allocations. Similar situation also exists for program allocations. Several factors, to be studied in detail later in this dissertation, further complicate these decisions. For example, the storage capacity of n3 may be small to inhibit the storage of large amount of data, n1 may have a very fast computer which can outweigh the remote access delay, the administration may want to restrict sensitive data to one node for security purposes, relationships may exist between data files which complicate the measurement of costs, etc. The purpose of the allocation algorithm is to determine the optimal, or near optimal, allocation of programs and data

to minimize the cost subject to the various factors. The generalized problem is now stated.

1.4 Thesis Objective

The objective of this research is the formulation and solution of the following allocation problem encounterd in the design of new application systems in DDP: [1]

GIVEN:

1). Application system information at a partitioned/logical level, consisting of:

- The database which shows the size and relationships of user data.
- The programs which show the operations to be performed on the database and the relationships between programs.
- The transactions submitted by the users which activate the programs.

2). DDP information which describes the support system under which the application system is to be designed, installed and used. This information specifies:

- The number, speed and capacity of nodes, where each node represents a computer.
- The network topology (interconnection between nodes) and control (rules for the exchange of messages between nodes).
- The transaction processing options that are used to access data from programs and select sites for data and program operations.
- The update synchronization algorithms that are used to

---

1. The exact parameters to represent this model are defined later and a formal problem statement is given in chapter 6.

update and manage duplicate data.

3). Costs for utilization of storage (mega bytes used), node processing (node i/o) and communication links (remote message).

DETERMINE:

. Allocation of application data to nodes.

. Allocation of application programs to nodes.

WHICH MINIMIZES: the total cost of running an application system

= Communication cost (cost for sending/receiving internode messages)

+ Node processing cost (cost for local i/o and processing)

+ Storage cost (cost of storing data and programs on nodes)

SUBJECT TO:

1). User performance requirements:

a.Transaction response time (time elapsed between transaction submission and display of results) must not exceed a user specified value. The response time is a sum of local processing delays, communication (remote) processing delays and the queuing delays due to device busy and locking conflicts.

b.Transaction availability (percentage of time the transaction executes properly) must not be less than a user specified value. The transaction availability is a function of the availability (reliability) of all the components that are used by the transaction.

2). Support system/management restrictions:

a.The storage occupied at node i must not exceed the storage capacity of node i.

b.The data must be allocated to at least one node but may be restricted to a few nodes due to the policy and/or security considerations. For example, sensitive data may be restricted to a single node.

c.The programs must be allocated to at least one node but may be restricted to a few nodes due to the

technical (processor limitations) and/or program duplication cost constraints. For example, program duplication costs may be prohibitively large in terms of time and money in a heterogeneous network.

EXPLANATIONS AND COMMENTS:

It should be emphasized that the the local database and program design issues are beyond the scope of this research. It also follows that, in the absence of local design decisions, the calculations of the costs and constraints are only approximate. This issue will be examined in chapter 7.

In ideal environments, the programs can be replicated to all nodes, thus eliminating the program allocation decision. However, fo r several practical situations which involve heterogeneous systems, the program allocation is important because program replication incurs considerable software development and maintenance costs. Program allocation may also be restricted due to the capacity of processors. For example, if a program needs floating point arithmetic, then it should be allocated to the nodes with floating point hardware feature.

Data duplication decreases data retrieval cost but increases cost of updating and maintaining redundant data. It follows that an optimal data allocation always exists. This property of data distribution has provided the main motivation for the large number of data (file) allocation schemes. Basically there are four data distribution strategies (see <TEOR82e>):

. Centralized (single copy of the database located at 1 node).

. Unique (single copy of the database located at several nodes).

. Replicated (complete copy of the database at several nodes).

. Hybrid (partial copy of the database at several nodes).

As can be expected, different tradeoffs exist for each strategy. This issue, discussed in detail by Teorey and Fry <TEOR82e> will be treated in chapter 4.

It is important that the application variables used must be able to represent complex application systems in which each transaction may activate several programs, each program may access several data files and relationships may exist between data. The support system variables must be able to specify a large number of practical DDP configurations so that changes in the support system can be taken into account (case 3 and 4 in section 1.1). Thus systems involving heterogeneous components (micros, minis, large scale systems) interconnected in various fashions and with different software facilities (database managers, transaction managers, communication managers) must be represented.

The objective function represents the total cost of running an application system. In practice, this cost may be a utility function. The user performace requirements represent the total transaction response time and the transaction availability. These variables are dependent on a large number of factors and are difficult to calculate.

The support system and management considerations restrict the decision variables in a large number of practical situations. For example, a hybrid data distribution strategy may be eliminated if the support system does not facilitate this strategy.

1.5 Literature Overview

The program and data allocation problem stated above involves several rapidly advancing areas in computing and communication technologies. A large body of literature exists in these areas and articles on different aspects of distributed data processing continue to appear regularly in many computing journals. The bibliography lists main references applicable to this research and is organized in six sections, a

through f. At present, the references listed in section c are of particular interest to us because it lists the existing allocation algorithms since the publication of the first paper by Chu <CHU69a> in 1969.

Let us determine if existing work can be used to solve the program and data allocation problem stated above. Table 1.1 serves as a basis for evaluating current allocation schemes which are listed as rows. The various factors of the problem stated in section 1.4 (objective function, application description, support system description, management and support system restrictions and user performance restrictions) are shown as columns of table 1.1. The allocation work is categorized into: program/data allocation schemes which attempt to solve the problem of program and data allocation, data allocation schemes which only concentrate on allocation of data to nodes, program allocation schemes which focus on the program allocation problem, and the data/communication schemes which attempt to solve the combined problem of data and communication system allocation. A detailed analysis of these categories is given in appendix I. We can make the following observations from table 1.1:

1. None of the currently available schemes address the problem stated in section 1.2.
2. Only four researchers <MORG77c, BUCK79c, FISH80c, MARC81c> have considered the problem of program and data allocation so far. It can be seen that most work has been done on the data allocation problem, a subproblem of program and data allocation.
3. Most schemes ignore the local processing cost. The reason for this is not known.
4. The application description is usually restricted to simple cases. For example, most schemes cannot describe the relationships between data, programs and transactions. Thus these models cannot describe realistically complex application systems where one

TABLE 1.1: COMPARISON OF ALLOCATION SCHEMES

		OBJECTIVE FUNCTION			APPLICATION DESCRIPTION			SUPPORT SYSTEM DESCRIPTION			SUPPORT SYSTEM & MANAGMT. RESTRICTIONS				USER PERFORMANCE REQUIREMENTS			
		STOR COST	LOCAL PROC COST	RMT PROC COST	DATA DESC	PROG DESC	TRAN-SACT. DESC	TRAN PROC MODE	UPDATE SYNCH. ALGOR.	NETWK TOPOL. & CNTL	STORAGE CAPACITY	LINK CAPCTY	DATA RESTR	PROGM RESTR	TRANSACTION AVAILABILTY FILE	TRANSACTION AVAILABILTY TOTAL	TRANSACTION RESPONSE TIME COMMUNIC	TRANSACTION RESPONSE TIME TOTAL
Program/ Data Allocation	MORG77	*		*	1	2	2				*		*				9	
	BUCK79	*		*	1		3											
	FISH80	*		*	1	2	2											
	MARC81	*		*	1	*	3				*				12		10	
	CHU69	*		*	1		3				*						10	
Data Allocation	CASE72	*		*	1		3											
	LOOM76		*	*	*		3											
	CHAND76	*		*	1		3											
	TABA78			4			3											11
	SURI79	*		*	1		3				*	*			12		9	
	COFF80	*		*	1		3		5									
	CHANG81	*		*	*	6	6		5									
	CHEN83	*		4			*		*									*
Program Allocation	GYLY76			7		*					*						9	
	BRYA81			7		*												11
	MCENT81		*	7		*											9	
	DOTY82			7		*											9	
Data/ Communication Allocation	WHIT70	*		*	1		3			8								
	CASE73	*		*	1		3			8					*		9	
	MAHM76	*		*	1		3			8					12		10	
	CHANG77	*		*	1		3			8					12		9	
	IRAN81	*		*	1		3			8		4			12		10	

LEGEND FOR TABLE ENTRIES:
- * indicates that the factor is fully included by the researcher
- n, where n is an integer, indiactes that the factor is partially considered by the researcher. See note n for explanation.
- blank indicates that the factor is ignored by the researcher

NOTES:
1. Only describes data (file) size. Relationships between data are ignored.
2. Only program to data read/update operations are defined. The relationships between programs are ignored.
3. The transaction is described in terms of node to file read/update traffic. Cannot model complex transactions which involve operations on more than 1 file.
4. The objective function is not a dollar cost. It represents the transaction turnaround time (response time).
5. Only simple/specialized cases of update synchronization can be handled, for example, Ellis ring algorithm.
6. Can describe relational view of data.
7. The objective function is not dollar cost. Instead, processing time or some "benefit" function is used.
8. Network design is a decision variable. May include topology design or channel capacity assignment.
9. Communication delay is expressed in terms of the "maximum expected internode delay", an input parameter.
10. Node to node communication delay; includes queuing at the communication links (channels).
11. Communications + local + queuing delays. Excludes delays due to locking.
12. Availability of a file from a node. Does not consider the node to program and program to file availability (total transaction availability).

transaction may activate several programs, one program may access many data files and relationships between datasets may exist.

5. The support system descriptions of existing models are also limited. For example, the transaction processing options, update synchronization algorithms and network topologies cannot be described adequetely.
6. The management and support system restrictions are rarely introduced in the existing models.
7. The user performance restrictions are also limited. For example, most schemes use communication delay as a constraint. In practice, the user is intersested in the total transaction response time which is given by the sums of all the communication, node processing and queueuing delays.

The main reason for these shortcomings is that the available schemes have been developed for theoretical investigation into particaular aspects of the allocation problems and are limited in practical applicability because very few parameters (in most cases about 4) are provided for describing the real life systems. Several other researchers <ROTH77a, WAH84a, HEVN79a, CHAND77a> have also reviewed this work and discovered similar limitations.

It is obvious that the currently available allocation schemes address certain portions of the generalized program and data allocation problem, but the combined effect of the large number of actual factors which influence the assignment of programs and data in real systems are not included. Thus, although these schemes have provided significant insight into the nature of the allocation problem, the practical application is severely limited. This finding is the main motivation for our research.

1.6 Research Approach and Outline of Dissertation

Formulation and solution of the program and data allocation problem stated in section 1.4 requires formal representation of, and derivation of the intricate relationships between, the application and support systems. The approach to be taken in this research has three phases: the effect of application system characteristics on program and data allocation is studied in depth by assuming a simplified support system (chapters 2 and 3); the effect of support systems is introduced by successively including transaction processing options, update synchronization algorithms, physical resource configurations, response time and queuing considerations (chapters 4 and 5), and the extensions and techniques introduced in the first two phases are integrated into a single generalized program and data allocation algorithm, the main output of this research, which is implemented as a computer program and tested on an actual system (chapter 6). The major advantage of this approach is that we can first concentrate on application system factors and then succesively include the complex support system factors to extend our model, thus providing a systematic mechanism to study, evaluate and parameterize the variables that need to be included in our model.

Chapter 7 concludes this dissertation by discussing significant results of this research, proposing a distributed application system design methodology, showing possible applications of the model and methodology, and listing research limitations. The primary contribution of this research is that the problem of program and data allocation is formulated, solved, and evaluated in its most general form.

CHAPTER 2
THE UNCONSTRAINED PROBLEM

Our first major objective is to study in detail the effect of application system factors on the program and data allocations by assuming an idealized support system. In section 2.2, we define and parameterize generalized application systems and introduce the concept of service requests to measure the interplays between the application and support systems. The unconstrained allocation problem is stated mathematically in section 2.3 and the major underlying assumptions are listed in section 2.4. A predictive model is introduced in section 2.5 to calculate the total cost (objective function). The mathematical programming problem is analyzed in section 2.6 and the solution algorithm is presented in section 2.7.

2.1 Symbols and Notations

Our approach to the data and program allocation problem classifies four type of variables:

i). **Independent (external) variables** which define the application system and the DDP support system.

ii). **Decision (control) variables** which show the allocation of data and programs to nodes.

iii). **Dependent (observable) variables** which show information like the communication cost, node processing cost, transaction response time, etc. These variables are functions of the independent and decision variables.

iv). Intermediate Variables which are used internally for calculations etc.

All the independent and dependent variables are designated by single capital letters A thru Z. Subscripts are used to represent similar variables. For example, C_S represents the storage cost, C_L shows the local processing cost and C_R exhibits the remote processing cost. Parenthesis are used to identify matrix elements. For example, $C_S(i)$ shows the cost of storing data at node i.

The decision variables are represented by α and β. The variables closely associated with the decision variables are identified by using subscripts. For example, α_s is used to show the site selected due to the decision variable α. Small and/or Greek letters are used to designate intermediate variables. Throughout:

. d indicates a dataset

. p indicates a program

. t indicates a transaction

. i and j indicate nodes

. k indicates a transaction arrival node

. Ξ is used to designate elements of a set. For example, $p \Xi P$ shows that p is an element of set P.

. D is the set of all the datasets (files) in the application system. Thus $d \Xi D$.

. P is the set of all the programs in the application system. Thus $p \Xi P$.

. T is the set of all the transactions in the application system. Thus $t \Xi T$.

. N is the set of all the nodes. Thus, $i \Xi N$, $j \Xi N$ and $k \Xi N$.

For notational simplicity, we will use $\Sigma_x f(x)$ to indicate that f is summed over all possible values of x, whenever the possible values of x are clear from the context. For example, $\Sigma_i f(i)$ shows that f is summed over all possible nodes i in the network. In most cases, A(i)B(j) will

indicate a multiplication of A(i) by B(j). Occasionally, the symbol × will be used for clarity.

2.2 The Parameters

2.2.1 The Application System Parameters

The basic components of an application system are the database, the programs and the transactions, as shown in figure 2.1a. Users submit transactions, transactions activate programs and programs access the database(s). One to one, one to many, many to one and many to many relationships may exist between any two of the three components.

The relational model <COD70e, DATE75d> is employed to help discuss and explain the application systems and associated parameters. Data is represented in this model as a collection of *relations (tables)* which consist of unique and fixed length *tuples (rows)*. Each tuple consists of several *attributes (columns)*. Transactions perform three fundamental operations on the data: i)*Selection* to choose certain rows from a relation, ii)*Projection* to choose certain columns, and iii)*Join* to concatenate two relations based on common attributes. Although the relational model is usually restricted to data retrievel, we assume that a *modify* operation is also available which changes the database through insertions, updates and deletions. The relational model is used here only to discuss and explain application systems due to its ease of use; the scope of this research is applicable to both relational and nonrelational systems systems.

DATABASE PARAMETERS: A database is a collection of data items. We assume that the database can be viewed in terms of D_n "datasets" where a dataset is the smallest unit of data allocation. Thus a dataset $d \in D$ may be a relation (file), a tuple (record) or an arbitrary collection of tuples. Let the matrix $D_s(d,x)$ represent the tuple size (x=1) in bytes,

and tuple occurrence (x=2) in thousands, of dataset d. A dataset d1 can be related to another dataset d2 through a *consistency relationship* $D_c(d1,d2)$ which means that d2 must be modified whenever d1 is modified. $D_c(d1,d2)$ can be represented to show any relationships between d1 and d2 which results in a modification (deletion, insertion and update) of d2 whenever d1 is modified. Examples of such relationships can be parent/child relationships in hierarchical database models and indices maintained in many file systems. Note that, depending upon the type of normalization, the consistency relationship can be one to one, one to many, many to one or many to many. Simply stated, a database is *consistent* if it correctly represents, at time t1, real life information at time t0, where $t1 \neq t0$.

PROGRAM PARAMETERS: The database is accessed and manipulated by P_n programs, where a program $p \in P$ is a collection of software instructions that may perform one or more operations (selection, projection, join, modify) on the database. Let $P_I(p)$ represent the number of input (keyboard) messages received by program p per execution of p and let $P_O(p)$ represent the number of output (display) messages generated per input message. Let $P_r(p,d)$ symbolize the average number of reads issued from program p to dataset d per input message. Note that in this model, the number of output messages generated and the number of reads issued by a program are dependent on the number of input messages. We observe that typically the number of input messages received by a program dictate the file accesses as well as the size of the reports generated by the program. A program p may modify dataset d by insertions, deletions and updates. Some of these modifications necessitate "secondary modifications" in consistency related datasets. For example, deletion of a tuple from a course relation may require update of other tuples from the student relation. Let $P_w(p,d)$ and $P_c(p,d)$, respectively, represent the first order (no secondary effect) and second order (consistency related secondary changes) writes/modifications applied to dataset d from

program p. In order to minimize number of parameters and simplify analysis, this model does not facilitate "currency" type program-to-database relationships (next, last, next within parent, unique etc). This simplification can be eliminated by introducing new parameters which can be used to calculate P_r, P_w and P_c, and thus currency relationships can be easily included in the model (see example below). Programs can be interrelated to other programs in a complex manner. For example, a program p1 may pass $P_m(p1,p2)$ temporary records (intermediate datasets) to program p2. Note that p1 and p2 can execute in parallel if $P_m(p1,p2) = 0$. We assume, for the sake of simplicity, that all the input messages, displays and temporary file records are of same fixed length, say 80 bytes. This assumption can be relaxed by introducing three additional parameters: average size of input messages, average size of displays and average record size of the temporary files.

TRANSACTION PARAMETERS: A program is activated by a transaction t ∈ T which is essentially the command(s) submitted by the user. For example, a SEQUEL SELECT statement from a user can be viewed as a transaction. In case of simple queries, a transaction may activate one program, while a series of programs may be activated to satisfy complex requests. The notion of transaction used in this research conforms to the one used by Eswaran <ESWA76> who defines a transaction to be a unit of consistency, i.e. a transaction starts with a consistent state of the database and terminates with another consistent state of the database, even though the database consistency may be violated temporarily when the transaction is executing. Let $T_e(t,p)$ indicate the number of times program p is executed by transaction t. $T_a(t,k)$, the number of times transaction t arrives at node k, determines the workload generated by the application system. A transaction can be decomposed into subtransactions, where each subtransaction executes at a single node.

DISCUSSION OF THE APPLICATION SYSTEM PARAMETERS: The application system parameters defined above, and shown in table 2.1, can be used ef-

fectively to describe a large number of complex application systems. To illustrate the use of these parameters, let us consider a computerized student registeration system as a sample application system. This application system will be used throughout this thesis for illustration. Assume three relations DEPARTMENT (Dno, Dptname), COURSE (Cno, Cname, Dno, Inst) and STUDENT(Sno, Sname, Cno), with the information shown in figure 2.1b.

These three relations can be viewd as three datasets d1 (department), d2 (course) and d3 (student). Note that a consistency relationship exists between the course and student datasets (if a course is dropped, then the student relation needs to be updated). Similarly, a consistency relationship exists between the department and the course datasets.

Consider the following transaction which deletes (drops) all courses from the Mathematics department taught by the instructor Woo:

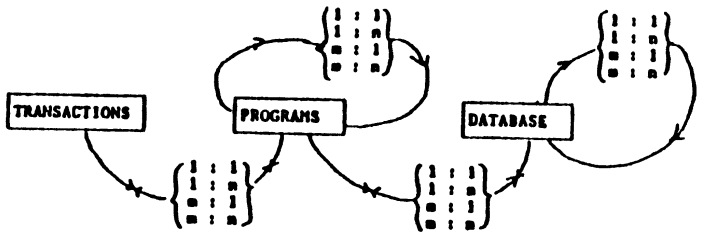
DROP course WHERE (Inst='Woo') and (Dptname.Department='Mathematics')

We assume that this transaction also prints the department tuple for the Mathematics department and the course tuples deleted. Two major operations are performed by this transaction: i) determination of the department number (Dno) for Dptname = 'Mathematics' from the department dataset and ii) deletion of the course tuples with Dno = '40' and Inst = 'Woo'. Each of these operations can be described in terms of relational algebra (selection, projection, join).

Let us assume that the first operation is performed by program p1 and the second operation is performed by program p2. Then p1 receives one input record (Dptname='Mathematics'), reads the department dataset, say, 2 times causing two physical i/o's, prints (displays) the mathematics department tuple and passes 1 record (tuple with Dno='40') to p2. The program p2 also receives one input record (Inst='Woo'), reads the temporary record generated by p1, reads the course dataset, say, 6 times, deletes one tuple (course MA05) and displays the tuple deleted,

FIG 2.1 : THE APPLICATION SYSTEM

a). COMPONENTS OF AN APPLICATION SYSTEM



b). EXAMPLE OF AN APPLICATION SYSTEM DATABASE

DEPARTMENT

DNO	DPTNAME
10	Arts
20	Management
30	Engineering
40	Mathematics

COURSE

CNO	CNAME	DNO	INST
ARO1	Painting	10	Doe
MAO2	Algebra	40	Jone
ENO5	Design	30	Woo
MGO1	Accntg	20	Chen
EN10	Circuit	30	Jone
MAO5	Calculus	40	Woo

STUDENT

SNO	SNAME	CNO
170	Mark	MAO2
232	Sue	MAO5
536	Sam	MGO1
221	Phil	EN10
346	Mary	MAO5
482	Dave	ENO5

c). PARAMETERS DESCRIBING THE APPLICATION SYSTEM

i). DATABASE PARAMETERS:

$D_n = 3$

$D_c(d1,d2)$ =

d1 \ d2	1	2	3
1	0	1	0
2	0	0	1
3	0	0	0

$D_s(d,x)$ =

d \ x	1	2
1	40	6
2	80	6
3	60	6

ii). PROGRAM PARAMETERS: $P_n = 2$

$P_r(p,d)$=

p \ d	1	2	3
1	4	0	0
2	0	6	0

$P_w(p,d)$=

p \ d	1	2	3
1	0	0	0
2	0	0	0

$P_c(p,d)$=

p \ d	1	2	3
1	0	0	0
2	0	1	0

$P_I(p)$=

p	1	2
	1	2

$P_o(p)$=

p	1	2
	1	1

$P_m(p1,p2)$=

p1 \ p2	1	2
1	0	1
2	0	0

iii). TRANSACTION PARAMETERS:

(We assume that the drop transaction arrives 10 times per hour at node1 and 20 times at node 5, in a 5 node network):

$T_n = 1$

$T_e(p)$ =

p	1	2
	1	1

$T_a(k)$ =

k	1	2	3	4	5
	10	0	10	0	20

TABLE (2.1) : APPLICATION SYSTEM PARAMETERS

a). DATA DESCRIPTION

(D = set of all the datasets in the application system)

- D_n = number of datasets in the application system.
- $D_c(d1,d2)$= consistency relationship between dataset d1 and d2; implies that d2 must be updated whenever d1 is updated .
- $D_s(d,x)$ = size (x=1) and occurrence (x=2) of tuples in dataset d.

b). PROGRAM DESCRIPTION

(P = set of all the programs in the application system)

- P_n = number of programs in the application system
- $P_r(p,d)$ = number of reads issued from program p to dataset d.
- $P_w(p,d)$ = number of updates issued from program p to dataset d.
- $P_c(p,d)$ = number of consistency updates issued from program p to dataset d.
- $P_I(p)$ = average number of input records read by program p
- $P_o(p)$ = display messages written by program p.
- $P_m(p,p1)$= No. of temporary records written by program p that are to be received by by program p1.

c). TRANSACTION DESCRIPTION

(T = set of all the transactions in the application system)

- T_n = total number of transaction types in the application system.
- $T_e(t,p)$= no. of times program p is executed per invocation of transaction t.
- $T_a(t,k)$ = No. of times transaction t arrives at node k for a given observation period.

one line display. Notice that due to the deletion of this course tuple, two tuples in the student dataset (students 232 and 346) need to be updated to maintain database consistency (secondary updates). Figure 2.1c describes this application system in terms of the parameters shown in table 2.1.

In addition to describing relational systems, these parameters can also be used to specify accesses to a hierarchical database. For example, consider a database schema consisting of two record types: the parent record type A and the child record type B. Let us assume that there are 100 occurrences of record type A and 200 occurrences of record type B per A. Each record type can be viewed as a separate dataset, thus d1= all occurrences of A and d2= all occurrences of B. Consider a program p1 which finds random occurrence of record B. Assuming that record type A is the main entry point and only sequential accesses are allowed between record occurrences, we can represent the database accesses from this program in terms of the $P_r(p,d)$ parameter which shows the number of reads issued from program p to dataset d:

$$P_r(p1,d1) = 50, \quad P_r(p1,d2) = 100$$

By using this procedure, currency related data access can be represented in terms of the application parameters presented above.

2.2.2 The DDP Support System Parameters

A support system consists of two major subsystems:

1. *Hardware Subsystem* which is a collection of all the computing devices (CPUs, storage, i/o devices, i/o interfaces) and communication devices (communication controllers, terminals, links, modems).
2. *Control Subsystem* which can be decomposed into i) *control programs* (operating systems to manage the hardware resources, que-

ry/transaction managers to control the flow of transactions in the system, the database managers to administer the access to the database and communication managers to provide interaction between remotely located devices) and ii) *control database* (data/network directories to show the structure and location of data and network resources, control blocks to describe the resources allocated by the tasks in the system, etc.).

The components of the DDP hardware and control subsystems exist at several geographical locations. This leads to two views of DDP support systems: the local view which describes the local hardware and control subsystem and global (distributed) view which describes the hardware as well as control subsystems which are used to provide and manage internode interactions.

As can be expected, we exclusively focus on the global view. Figure 2.2a shows and illustrates a simplified DDP support system. The system consists of several computers (nodes) interconnected through communication links. At each node, control programs CP and control database CD facilitate the interactions between the application programs AP and application database AD. Depending upon the location of these programs and databases, a given interaction, say a read, between AP and AD can generate different amount of local as well as remote activities. In order to measure these activities precisely, we introduce the important concept of service requests in the next section.

The discussion of support systems is intentionally brief and simplified in this section since we are presently only interested in application systems. Our current objective is to consider simplified support systems which can be represented by only one parameter - the number of nodes N_n. A notable simplification is that the control program CP is represented by a single layer in figure 2.2a. However, several layers of CP have been proposed and implemented in the industry. (See, in fig-

FIG 2.2a: THE DDP SUPPORT SYSTEM MODEL - SIMPLIFIED VIEW

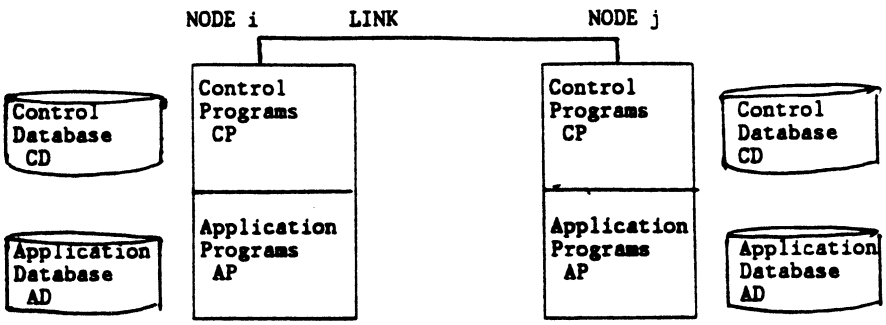
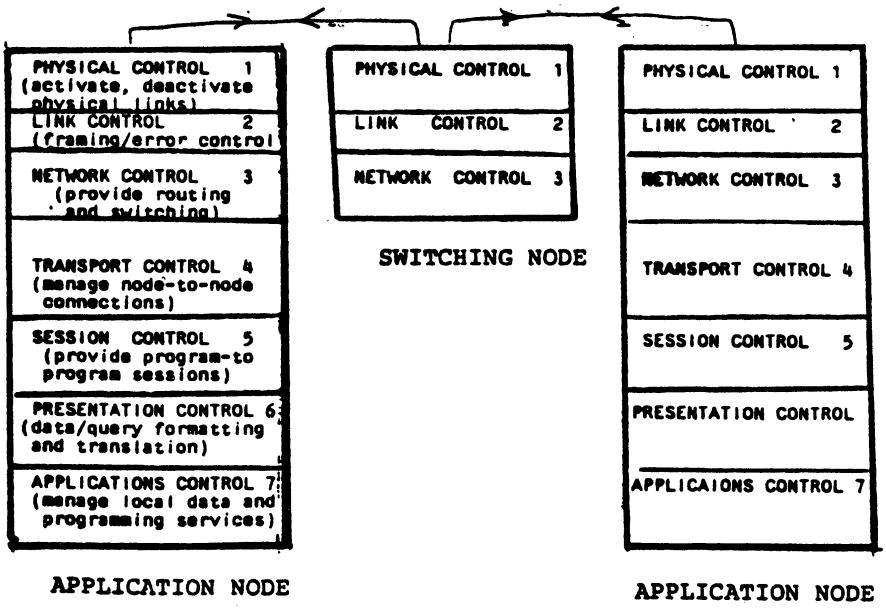


FIG 2.2b: THE ISO/ANSI DISTRIBUTED REFERENCE MODEL



ure 2.2b, the seven layers of CP proposed by the Distributed Reference Model of the ANSI/ISO SPARC subcommittee <BACH78d>.)

2.2.3 Measuring The Interplays - The Service Requests

A *Service Request* (SR) is defined as an interaction between two programs or a program and a dataset. For example, reading a dataset is a service request and exchanging information between two programs in execution is also a service request. Figure 2.3a illustrates this concept by showing that 6 SR are generated by a program when it reads a dataset.

If we assume that service requests between two entities in core can be ignored in figure 2.3a, then for core resident programs AP and CP, the number of SR are reduced to 4 (SR'a and d =0). Also, if the control database CD is core resident, then the SR b can be ignored, thus reducing the SR number to 2. In addition, if the record to be retrieved from AD is in core, then no SR are issued for this read.

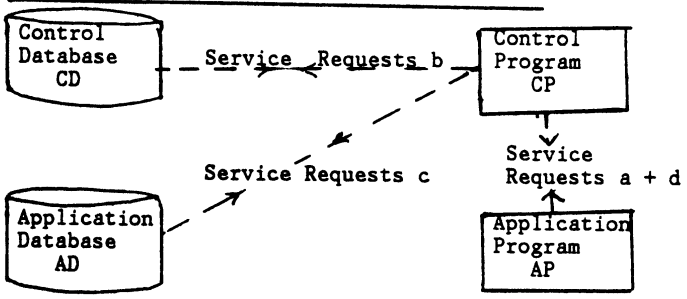
A service request can be local (LSR) or remote (RSR) depending on the location of the engaging parties(see figure 2.3b):

- *Local Service Request (LSR)* is defined as an interaction between two programs or between a program and a dataset at the same node. For example, all reads/updates issued to a dataset from a program at the same node can be viewed as LSR.
- *Remote Service Request (RSR)* is defined as an interaction between two programs or between a program and a dataset when the two parties exist at different nodes. For example, all internode messages can be viewed as RSR.

Figure 2.3b illustrates the LSR and the RSR by showing the access of a dataset at node j from a program at node i. Note that due to the issue of a request at one node, local service requests are generated at other nodes. The main advantage of inroducing SR is that for a given support system, appliction system and program/data allocations, the LSR and RSR show the local and remote activity and can be computed precisely in terms of the support system assumptions. Thus these quantities play key role in the calculation of dependent variables like local processing costs, remote processing costs, transaction availability and transaction response time. These quantities can also be easily translated into physical workload characteristics like disk i/o, communications messages, transport volumes, cpu instructions and delays at the local and network levels. In this sense, the service requests are similar to

FIG (2.3): THE SERVICE REQUESTS

a). ACCESS OF A LOCAL DATASET BY A PROGRAM



Read of one Record from AD by the Program AP causes 6 Service Requests (SR):

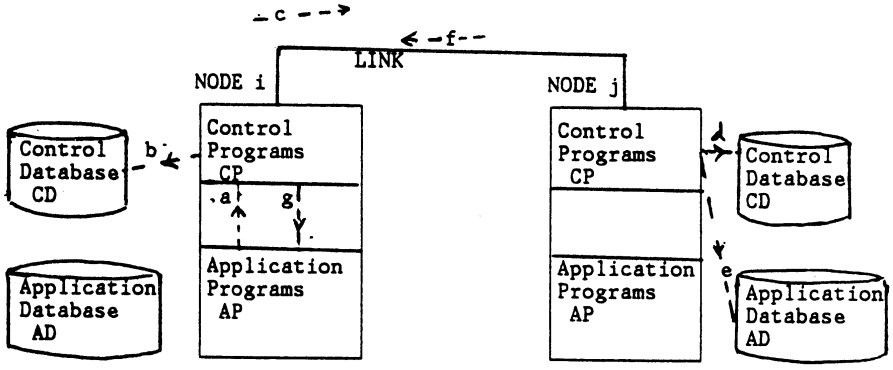
a. Application program AP issues the read to control program CP (1 SR).

b. CP accesses the control database AD to locate the record to be referenced. (2 SR - one access, one response).

c. CP now reads AD (2 SR - one access, one response).

d. CP sends the retrieved record to application program AP (1 SR).

b). LOCAL AND REMOTE SERVICE REQUESTS (LSR & RSR)



--<->-- Service Requests

Reconsider the example shown in figure 2.3 a. However, assume that AP and AD reside at two different nodes, i.e. AP_i, the application program at node i, wants to read application database AD_j residing at node j, where $i \neq j$. In this case, 10 SR are issued (2 remote - RSR and 8 local - LSR):

a. An LSR is issued from AP_i to CP_i, the control program at node i.

b. CP_i accesses CD_i to determine the node which houses AD_j. (Two LSR - one access, one response.)

c. CP_i transfers control to CP_j by issueing a Remote SR (1 RSR).

d. CP_j accesses CD_j to locate the record to be accessed (2 LSR).

e. CP_j now accesses the desired record from AD_j (2 LSR).

f. CP_j routes the retrieved record back to CP_i (1 RSR).

g. Finally, CP_i returns the desired record to AP_i (1 LSR).

the "logical record access" concept used by Teorey & Fry <TEOR80d> for database design.

2.3 Mathematical Formulation of the Unconstrained Problem

The unconstrained distributed data and program allocation problem can be stated as:

GIVEN:

1). N_n, the number of nodes in the network

2). The application system parameters shown in table (2.1).

DETERMINE: The allocation of datasets and programs to node i, given by the 0,1 decision variables:

. $\alpha(d,i)$ = 1 if dataset d is allocated to node i, 0 otherwise

. $\beta(p,i)$ = 1 if program p is allocated to node i, 0 otherwise

TO MINIMIZE: Total cost= Remote Processing Cost + Local Processing Cost + Data Storage Cost

1). Remote Processing Cost=$\Sigma_t\Sigma_k\Sigma_i\Sigma_j T_a(t,k)R(t,k,i,j)C_R(i,j)$....(2.1a)

Where:

. $T_a(t,k)$ = the arrival rate of transaction t at node k.

. $R(t,k,i,j)$ = no. of remote service requests exchanged between nodes i and j, when transaction t arrives at node k.

. $C_R(i,j)$ = the cost, in cents, for exchange of a message between nodes i & j,

2). Local Processing Cost = $\Sigma_t\Sigma_k\Sigma_i T_a(t,k)L(t,k,i)C_L(i)$ .....(2.1b)

Where:

. $L(t,k,i)$ = no. of local service requests executed at node i due to arrival of transaction t at node k.

. $C_L(i)$ = the cost, in cents, per local service at node i

3) Data Storage Cost = $\Sigma_i S(i)C_S(i)$ .....(2.1c)

Where:

. $S(i)$ = storage occupied at node i (in characters) .

. $C_S(i)$ is the cost, in cents, for storing 1 character at node i.

SUBJECT TO: the implicit constraints:

$\Sigma_i \alpha(d,i) \geq 1$ (each dataset must be assigned to at least one node)

$\Sigma_i \beta(p,i) \geq 1$ (each program must be assigned to at least one node)

2.4 Major Assumptions and Restrictions

We assume that the DDP support system satisfies the following simplifying assumptions:

1). The network consists of hardware components with infinite communication as well as node speed.

2). The nodes are homogeneous, reliable and fully connected computers with infinite storage and processor capacity.

3). All the nodes consist of homogeneous control programs and the control database is replicated and "core resident".

4). Due to homogeneous hardware and software (assumptions 2 and 3), we can deduce that there is no program development cost for duplicating a program. We also assume that program storage cost is negligible.

5). All service requests in main core are insignificant. Thus all local program to program service requests are ignored since it can be assumed that interacting programs are loaded in main storage. Also, due to the main storage residence of the control databases (assumption 3), all control program to local control databases service requests can be ignored.

6). The duplicate data is updated without any consistency/concurrency control (update synchronization) mechanism. This reflects a realistic situation if the updates are always applied to the local database and synchronized later by "store forward" appli-

cations. This also represents the "optimistic" update synchronization algorithms in idealized environments <BHAG82b>.

7). The transaction processing allows direct access of remotely located data based on the minimum access cost rule, i.e. the data is accessed from a node to minimze communication cost. It is also assumed that the transaction manager first chooses a program site from the arrival node, and then the data site from the program site.

8). There are no restrictions on data distribution, i.e. the data can be centralized, allocated uniquely, replicated or factored/partitioned in some fashion.

These assumptions play a key role in this research, since will later introduce the support system considerations by gradually relaxing these assumptions.

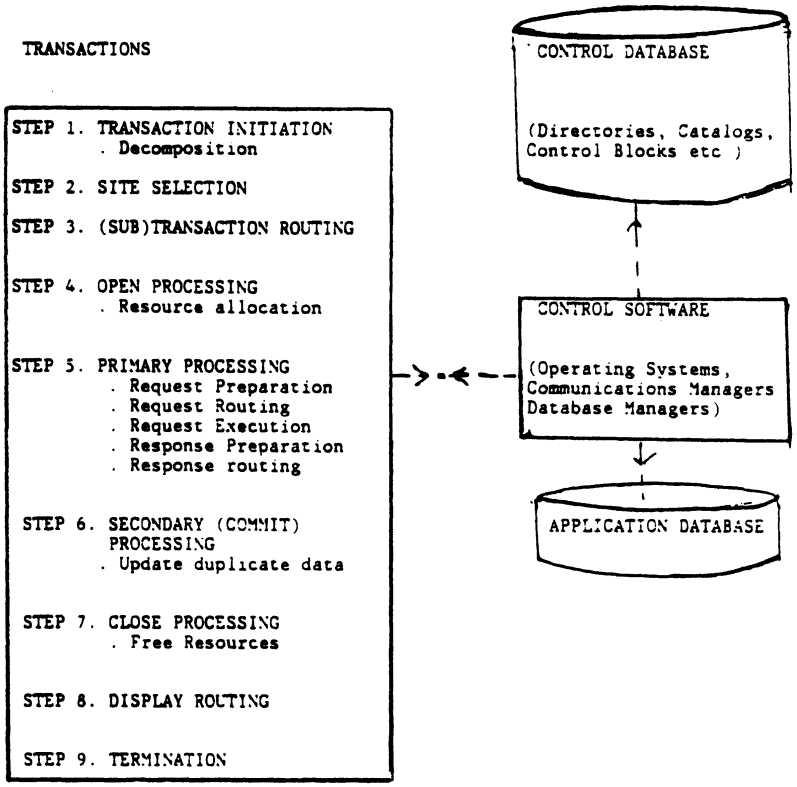
2.5 Calculation of Objective Function - The Predictive Model

The analytical expressions for dependent variables for storage (S), local processing (L) and remote processing (R) in equations 2.1a, 2.1b and 2.1c are needed. The expression for S(i), the amount of storage occupied at node i, is:

$$S(i) = \Sigma_d \alpha(d,i)\, D_s(d,1)\, D_s(d,2) \qquad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots (2.2)$$

Where $D_s(d,1)$ and $D_s(d,2)$ represent, respectively, the average size and number of tuples in dataset d, and $\alpha(d,i)$ is the data allocation decision variable. In order to estimate L and R, a predictive model is proposed which decomposes the transaction flow into several steps. Figure 2.4 shows the predictive model, section 2.5.1 gives the derivation of L and R and section 2.5.2 gives the final expressions.

FIGURE (2.4): THE PREDICTIVE MODEL - CALCULATION OF TOTAL COST



2.5.1 Calculation of Local and Remote Processing Cost

The predictive model shown in figure 2.4 decomposes the transaction flow into several steps: transaction initiation, site selection, (sub)transaction routing, open processing, primary processing, secondary (commit) processing, close processing and display routing. At each step, well defined activities are performed and L and R are computed based on the assumptions stated in section 2.4 (see below). In later chapters, these assumptions will be relaxed and the expressions for L and R will be recomputed. Thus this model is at the core of the optimal program/data allocation procedure.

To simplify notation, two zero-one variables are introduced to show the sites selected (primary nodes) for programs and data:

- $\beta_s(k,p,i)$ =1 if node i is the site selected for program p when a transaction arrives at node k, 0 otherwise. This variable is a function of the program allocation decision variable, $\beta(p,i)$, the arrival node k and the site selection "search" rule (minimum cost, minimum distance etc). Thus $\beta_s(k,p,i) = \beta(p,i)$ if i is "closest" to the arrival node k and i Ξ I, where I describes the set of nodes where program p resides.
- $\alpha_s(k,p,d,i)$ = 1 if node i is the site selected for dataset d when accessed through progrm p for a transaction arriving at node k, 0 otherwise. This variable is a function of the data allocation decision variable, $\alpha(d,i)$, the program site selection variable $\beta_s(k,p,j)$ and the site selection "search" rule (minimum cost, minimum distance). Thus $\alpha_s(k,p,d,j) = \beta_s(k,p,i)\ \alpha(d,j)$ where i is the program site selected for program p when accessed from node k, j is "closest" to the program node i, i Ξ I, j Ξ J, and I and J describe the set of nodes where program p and dataset d reside, respectively.

STEP 1:TRANSACTION INITITIATION

In this step, the transaction t is decomposed into t' subtransactions, where each subtransaction is a collection of programs which executes at the same node. We initially assume that every program can be treated as a subtransaction. There is no local or remote activity in this step:

$L(t,k,i) = 0$ and $R(t,k,i,j) = 0$ for all $k,i,j \Xi N$

STEP 2: SITE SELECTION

In this step, the sites are selected for all the programs (subtransactions) and datasets referenced by the transaction t, i.e. the site selection matrices α_s and β_s are determined:

. $\beta_s(k,p,i) = 1$ if i is the site selected for program p when the transaction arrives at node k.

. $\alpha_s(k,p,d,i) = 1$ if i is the site selected for dataset d when

from the program (subtransation) p.

There is no local or remote activity in this step since only the directories are accessed (see assumption 5 in section 2.4):

$$L(t,k,i) = 0 \quad \text{and } R(t,k,i,j) = 0 \text{ for all } k,i,j \in N$$

STEP 3: (SUB)TRANSACTION ROUTING

In this step, the subtransactions are routed to the sites selected in step 2. The service requests in this step depend upon the the size of input data and the size of the temporary records exchanged. The expressions for LSR and RSR are:

$$L(t,k,i) = 0$$

$$R(t,k,i,j) = 0 \text{ if } i=j$$

Else:

$$R(t,k,i,j) = \Sigma_p T_e(t,p) [P_I(p)\delta(k,i)\beta_s(k,p,j) + \Sigma_{p1} P_m(p,p1)\beta_s(k,p,i)\beta_s(k,p1,j)] \quad \text{when } i \neq j$$

Where $\delta(k,i) = 1$ if $k=i$, 0 otherwise. We will use δ frequently in these derivations.

STEP 4: OPEN PROCESSING

In this step, the (sub)transaction actually starts execution by checking authority and allocating resources. The subject of authority checking and resource allocation will be treated in chapter 4. The service requests in this step are mainly due to reading directories. Thus:

$$L(t,k,i) = 0 \quad \text{and } R(t,k,i,j) = 0 \text{ for all } k,i,j \in N$$

STEP 5 - PRIMARY PROCESSING

In this step, the application programs actually read/update the application database. This step is very complex and can be subdivided into several substeps. The details of this step will be presented in chapter 4. The LSR in this step is due to the read/update of data at the data sites selected (primary data):

$$L(t,k,i) = \Sigma_p T_e(t,p)P_I(p) \Sigma_d [P_r(p,d)+P_w(p,d)]\alpha_s(k,p,d,i)$$

Note that the data is read and updated only at the data sites selected, i.e. where $\alpha_s(k,p,d,i) = 1$. Also note that the input messages, $P_I(p)$,

are used as a multiplier because $P_r(p,d)$ and $P_w(p,d)$ represent the program reads and writes, respectively, per input message. The RSR in this step is due to the exchange of remote messages between program sites and data sites, when $i \neq j$:

$$R(t,k,i,j) = \Sigma_p\, T_e(t,p)P_I(p)\ \Sigma_d\, \beta_s(k,p,i)[P_r(p,d)+P_w(p,d)] \times \alpha_s(k,p,d,j)[1-\delta(i,j)]$$

STEP 6: SECONDARY (COMMIT) PROCESSING

In this step, all the copies of data updated in step 5 are updated (synchronized). This step takes place only if the updated data is duplicated and/or has consistency relationships. The actual synchronization mechanisms will be discussed in chapter 4. At present we simply take into account the updates:

$$L(t,k,i) = \Sigma_p\, T_e(t,p)\ P_I(p)\ \Sigma_d\ [\ \Sigma_{d1}\ P_c(p,d)\ D_c(d,d1)\alpha(d1,i) + P_w(p,d)\ \{\alpha(d,i) - \alpha_s(k,p,d,i)\}]$$

$$R(t,k,i,j) = \Sigma_p\, T_e(t,p)P_I(p)\ \Sigma_d\ [\Sigma_{d1}\ P_c(p,d)D_c(d,d1)\beta_s(\ k,p,i)\alpha(d1,j) + P_w(p,d)\ \{\alpha(d,j)-\alpha_s(k,p,d,j)\}\beta_s(k,p,i)][1-\delta(i,j)]$$

Basically, the P_c term shows the consistency related updates and the P_w term shows the updates to all the secondary (duplicate) data. Once again, $R(t,k,i,j) = 0$ if $i = j$.

STEP 7: CLOSE PROCESSING

In this step, the resources and the locks acquired in step 5 are freed. The service requests mainly are due to interactions between the control programs. We ignore this activity:

$$L(t,k,i) = 0 \qquad \text{and } R(t,k,i,j) = 0 \text{ for all } k,i,j) \ \Xi\ N$$

STEP 8 - DISPLAY ROUTING

In this step, the results computed in steps 5 and 6 are routed back to the originating node. The service requests depend upon number of display messages generated by each program:

$$L(t,k,i) = 0$$

$$R(t,k,i,j) = \Sigma_p\, T_e(t,p)P_I(p)P_o(p)\beta_s(k,p,i)\delta(k,j)[1-\delta(i,j)]$$

Please note that the $P_I(p)$ is being used as a multiplier again since $P_O(p)$ is the number of output messages generated per input message.

STEP 9 - TERMINATION

This step terminates the processing of the entire transaction and displays the messages back to the user. We assume no activity in this step:

$$L(t,k,i) = 0 \qquad R(t,k,i,j) = 0 \quad \text{for all } k,i,j \in N$$

2.5.2 Expressions for L and R

The expression for L(t,k,i), the number of local service requests issued at node i due to the arrival of transaction t at node k, is obtained by calculating the read activity at the primary nodes and the update activity at all the nodes where the duplicate, and/or the consistency related, data resides:

$$L(t,k,i)=\Sigma_p\ P_I(p)T_e(t,p)\ \Sigma_d\ [P_r(p,d)\alpha_s(k,p,d,i)+P_w(p,d)\alpha(d,i) + P_c(p,d)\Sigma_{d1}\ D_c(d,d1)\alpha(d1,i)] \qquad \text{.....(2.3)}$$

where $P_I(p)$ shows the number of input messages received by program p, $T_e(t,p)$ shows the number of times transaction type t activates program p , $P_r(p,d)$, $P_w(p,d)$, $P_c(p,d)$ are the data manipulation parameters and $D_c(d,d1)$ is the consistency relationship.

The expression for R(t,k,i,j), the number of remote service requests exhanged between nodes i and j when a transaction t arrives at node k, is obtained by calculating the node to program, program to program and program to data traffic:

$$R(t,k,i,i)= 0 \text{ and } R(t,k,i,j) = R(t,k,j,i) \text{ for any } k,i,j \in N$$

$$R(t,k,i,j) = R_{np}(t,k,i,j)+R_{pp}(t,k,i,j)+R_{pd}(t,k,i,j) \qquad \text{.....(2.4)}$$

Where R_{np}, R_{pp} and R_{pd} are given by:

$R_{np}(t,k,i,j)$ = node to program remote services exchanged due to reading of input data, from nodes, and sending of display messages, to nodes, by programs

$$= \Sigma_p T_e(t,p)P_I(p)[\delta(k,i)\beta_s(k,p,j)+\delta(k,j)\beta_s(k,p,i)P_o(p)] \quad .....(2.4a)$$

$R_{pp}(t,k,i,j)$ = program to program remote services exchanged to transfer

temporary data between programs at different sites

$$= \Sigma_p T_e(t,p)\ \beta_s(k,p,i)\ \Sigma_{p1}\ P_m(p,p1)\ \beta_s(k,p1,j) \quad .....(2.4b)$$

$R_{pd}(t,k,i,j)$ = program to data traffic due to the read/update of remotely located data by programs. Note that the update traffic modifies the actual data plus the consistency related data.

$$= \Sigma_p T_e(t,p)P_I(p)[\Sigma_d P_r(p,d)\beta_s(k,p,i)\alpha_s(k,p,d,j)+\Sigma_d P_w(p,d)\beta_s(k,p,i)\alpha(d,j)$$
$$+ \quad \Sigma_d\ P_c(p,d)\beta_s(k,p,i)\Sigma_{d1}D_c(d,d1)\alpha(d1,j)] \quad .....(2.4c)$$

Where $P_I(p)$ and $P_o(p)$ show the number of input and output messages received/generated by program p, $P_m(p,p1)$ are the number of temporary records generated by program p for p1 and $\delta(k,i)$ is a 0-1 variable such that $\delta(k,i) = 1$ if $i=k$, zero otherwise.

Figure 2.5 illustrates the local and remote service requests for the student registeration system described in section 2.2.1 in a 6 node network. The node to program, program to program and program to dataset traffic is shown. The LSR and RSR shown are based on the equations 2.3 and 2.4. It should be emphasized that equations 2.3 and 2.4 show the local and remote activities primarily in terms of the application parameters and can thus be used to study the effect of application variations on data and program allocations.

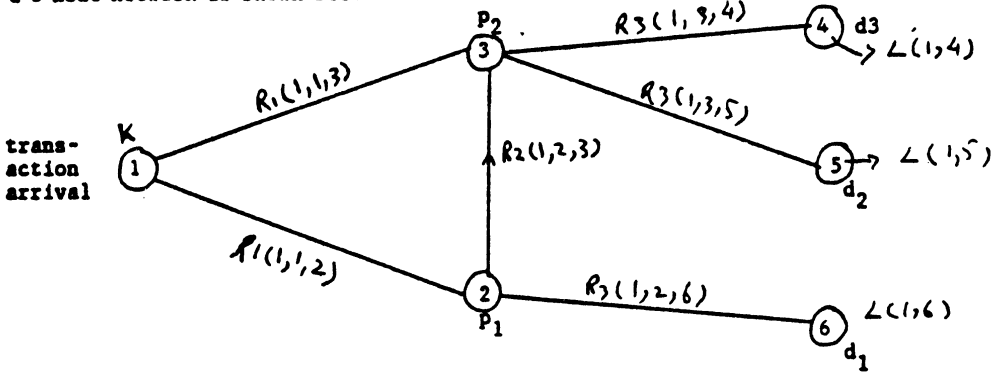
2.5.3 Discussion and Analysis of the Objective Function

The objective function given by equation 2.1 can now be written as:

$$=\Sigma_t\Sigma_k\Sigma_i\Sigma_j T_a(t,k)(R_{np}(t,k,i,j)+R_{pp}(t,k,i,j)+R_{pd}(t,k,i,j))C_R(i,j)$$
$$+ \Sigma_t\Sigma_k\Sigma_i\ T_a(t,k)\ L(t,k,i)\ C_R(i) +\Sigma_i\ S(i)\ C_S(i) \quad .....(2.5)$$

FIG 2.5 : ILLUSTRATION OF LOCAL AND REMOTE SERVICE REQUESTS

Consider the student registeration system described in section 2.2.1. Assume that the two programs p1 & p2 and the three datasets d1, d2, d3 are allocated to a 6 node network as shown below:



a). Decision (allocation) variables $\alpha(d,i)$ and $\beta(p,i)$:

$\alpha(d,i)$ =

i	1	2	3	4	5	6
1	0	0	0	0	0	1
2	0	0	0	0	1	0
3	0	0	0	1	0	0

$\beta(p,i)$ =

i	1	2	3	4	5	6
1	0	1	0	0	0	0
2	0	0	1	0	0	0

b). Site Selection Matrices $\alpha_s(p,d,i)$ and $\beta_s(k,p,i)$:

$\beta_s(1,1,2) = 1$ and $\beta_s(1,2,3) = 1$; zero for all other values

$\alpha_s(1,1,6) = 1$, $\alpha(2,1,6) = 1$

$\alpha_s(1,2,5) = 1$, $\alpha_s(2,2,5) = 1$

$\alpha_s(1,3,4) = 1$, $\alpha_s(2,3,4) = 1$

$\alpha_s(p,di,i) = 0$ otherwise

c). Local Service Requests L(k,i):

L(k,i) =

i	1	2	3	4	5	6
1	0	0	0	2	2	2
1	0	0	0	2	2	2
1	0	0	0	2	2	2
1	0	0	0	2	2	2
1	0	0	0	2	2	2
1	0	0	0	2	2	2

d). Remote Service Requests R(k,i,j):

R1(1,1,2) = node to program traffic exchanged between nodes 1,2 due to arrival of a transaction at node 1 = 1

R1(1,1,3)= node to program traffic exchanged between node 1,3 due to arrival of a transaction at node 1 = (1+1) = 2

R2(1,2,3)= program to program traffic between node 2,3 due to arrival of atransaction at node 1 = 1

R3(1,2,6) = program to dataset traffic between nodes 2 and 6 when the transaction arrives at node 1 = 2

R3(1,3,5) = 6

R3(1,3,4) = 2 .. due to the consistency updates to dataset 3.

where R_{np}, R_{pp}, R_{pd} are given by equations (2.4a, 2.4b, 2.4c), L(t,k,i) is given by equation (2.3), S(i) is given by equation (2.2), $C_R(i,j)$, $C_L(i)$ and $C_S(i)$ are given costs and $T_a(t,k)$ is given arrival rate. When this objective funtion is substituted in the mathematical programming problem stated in section 2.3, we obtain an integer programming problem with nonlinear objective function. We analyze the total cost to gain insights into the nature of the problem.

A few special cases of the cost function are studied next to show the effect of application system variations and the program/data allocations on the total cost of running an application system:

i). Local Data Access: Consider a situation where programs can access only local data. This may be due to support system restrictions (see for example <FOST76b>) or security considerations. Then $R_{pd}(t,k,i,j)= 0$ for all $i,j,k \in N$, but L(t,k,i) and S(i) are unaffected. Thus the remote services are now mainly due to node-to-program traffic (R_{np}) and the program to program traffic (R_{pp}) given by equations 2.4a and 2.4b. Note that in this situation if a program and data do not exist at the same node, either the program or data will have to be transfered to process a transaction.

ii). Remote Data Access: If a program can access data from anywhere in the system, then we can assume that the programs execute at the arriving node so that the node to program remote service rqeuests are eliminated. In this case, $R_{np}(t,k,i,j)=0$, $R_{pp}(t,k,i,j)=0$ and L(t,k,i) and S(i) are unaffected for all $i,j,k \in N$. Consequently the remote traffic, shown by equation 2.4c (R_{pd}), consists of the read and update requests sent from programs to data, a situation designated as node to file read and update traffic in the literature. Note that the communication cost in this situation would be entirely different from the communication cost obtained from (i).

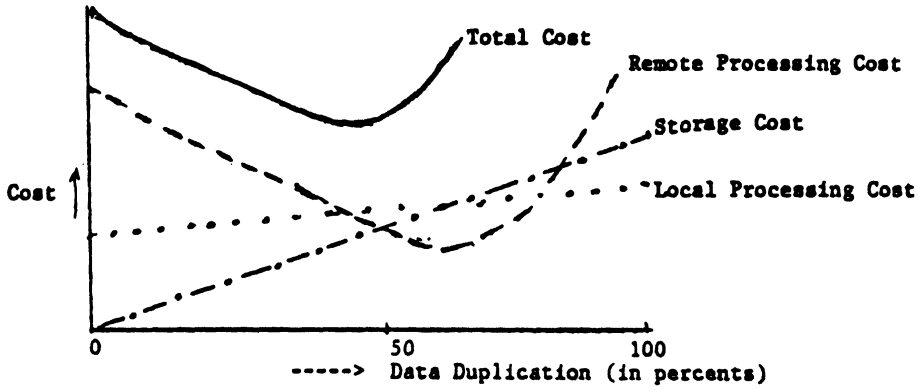
iii). Consistency Data Relationships: Existence of consistency relationships $D_c(d,d1)$ introduce overhead at local as well remote proc-

essing levels (see equations 2.3 and 2.4c with component $D_c(d,d1)$). An examination of equations 2.3 and 2.4c shows that the cost due to consistency updates is a monotonically increasing function of the allocation of consistency related data d1 because this cost is minimum when d1 is allocated to only one node and increases steadily as d1 is duplicated. Thus if several consistency relationships and secondary updates $P_c(p,d)$ exist in the application system, then it may be better to keep all the data at a single location.

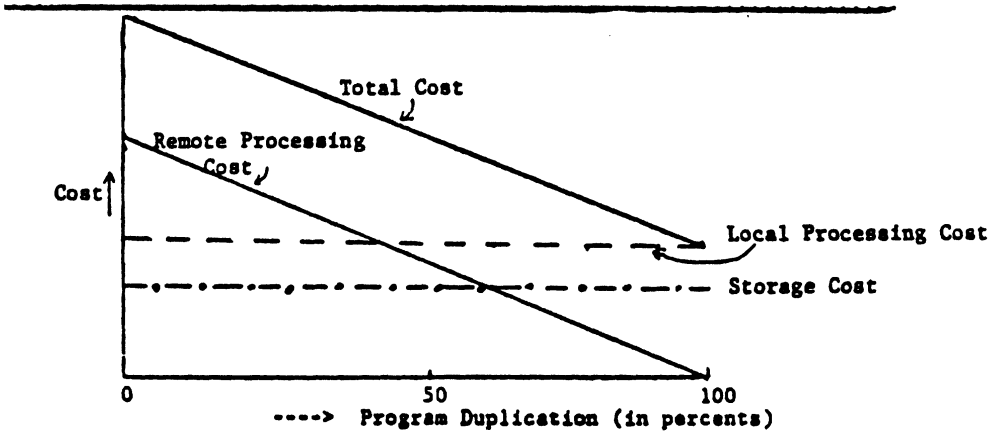
iv). Program to Program Exchanges: Existence of program to program exchanges $P_m(p,p1)$ only influence the remote traffic (see equation 2.4b). The local traffic is not affected since we ignore the interactions between local programs (see assumption 5 of section 2.4).

v). Data Duplication: Consider the influence of data duplication on the objective function by assuming that programs are replicated and that there are no consistency relationships. The results, based on a computer program which calculates the storage, local processing and remote processing costs given by equations 2.2, 2.3 and 2.4, are given in figure 2.6a. It can be seen that the storage occupied (S) is a monotonically increasing function of data duplication, the node processing activity (L) is also a monotonically increasing function of data duplication and the communication activity (R) first decreases and then starts increasing with data duplication. The initial drop in the communication cost is due to the increase in the availability of local data, which makes internode communication unnecessary. However, as the data duplication keeps on increasing, more internode communication is needed for updating duplicate data, eventually resulting in a net increase in the communication cost. The communication cost measured is only due to program-to-node traffic (the node to program and program to program traffic is zero since all programs are replicated). The objective function, as a result of the individual behaviours of S, L and R, is a convex function of the data duplication. This property of data duplication,

FIG (2.6). THE BEHAVIOUR OF THE OBJECTIVE FUNCTION (TOTAL COST)



a). THE EFFECT OF DATA DUPLICATION ON
STORAGE, LOCAL, REMOTE & TOTAL COSTS
(It is assumed that all the programs are replicated)
The write/read ratio is approximately 30%



b). THE EFFECT OF PROGRAM DUPLICATION ON
STORAGE, LOCAL, REMOTE & TOTAL COSTS.
(It is assumed that all the data is replicated)

albeit by ignoring L, has been studied extensively in the literature and has been the main motivation for the large number of file allocation algorithms.

vi). Program Duplication: Consider the influence of program duplication on the objective function (see figure 2.6b). To avoid unnecessary complications, we assume that the data is completely replicated. We have also assumed (assumption 4, section 2.4) that the programs occupy negligible amount of storage (a practical assumption in view of cheap storage and small amount of program storage as compared to data storage). Now, let us examine figure 2.6b. The storage cost is constant because we assume programs do not occupy any disk storage (disk storage used by programs is usually negligible as compared to the stoarge used by files), the local processing is fixed because at one time only one copy of the program is at use and the communication costs drop due to the elimination of node to program traffic (there is no program to data traffic since EVERY node has a copy of data). Thus the overall objective function decreases monotonically in figure 2.6b. This implies that the program optimization is trivial for this problem. However, if the cost of program replication (duplicate software development) is included in the objective function, then the objective function also shows convexity in heterogeneous systems. In our model, the software development cost is a management constraint which may restrict program replication. This cost can be included in the objective function by using a Langrange multiplier. This topic will be treated in chapter 6.

2.6 Optimization: The Unconstrained Allocation Algorithm

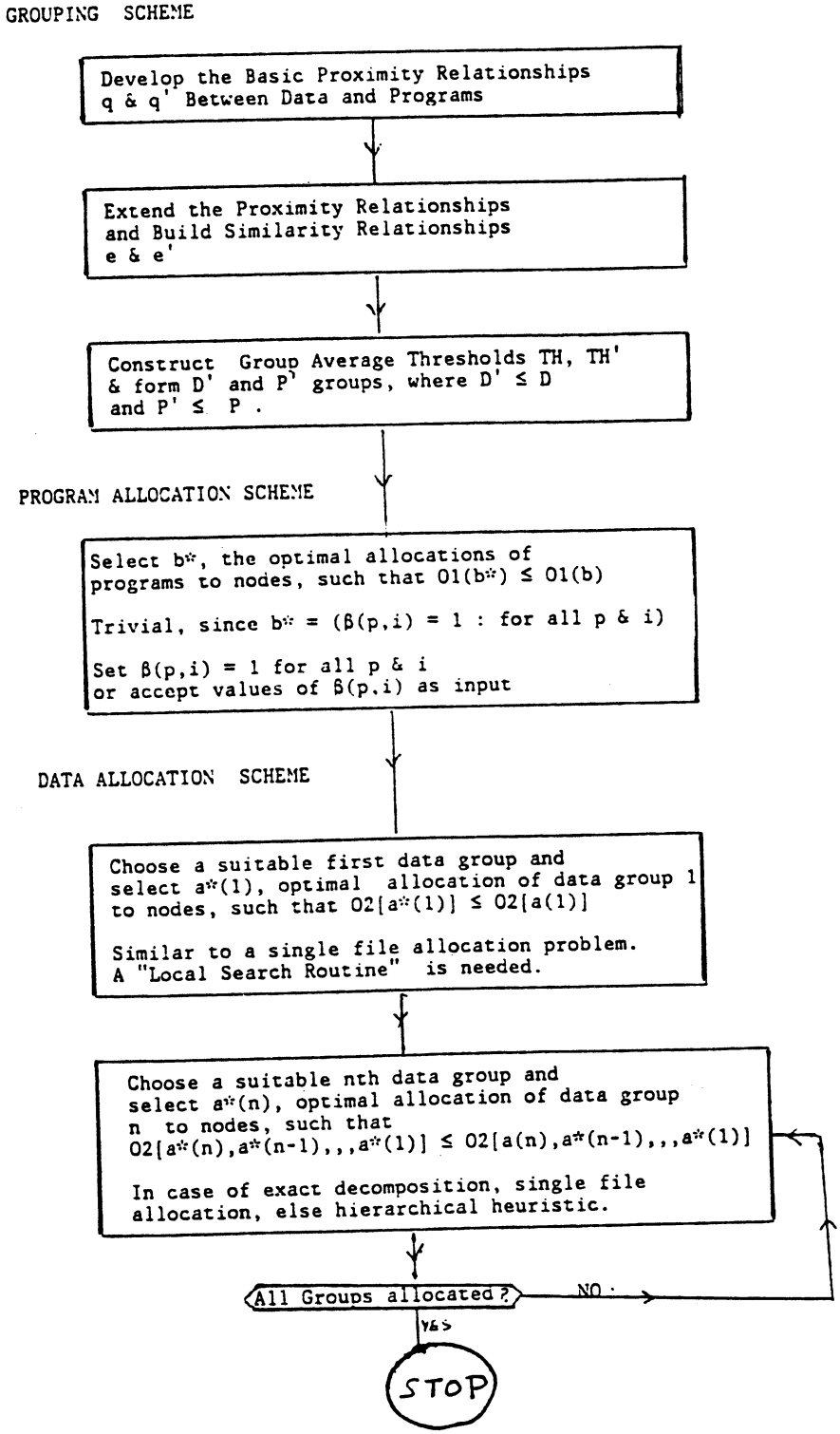
The purpose of the allocation algorithm is to determine the decision variables α (allocation of data to nodes) and β (allocation of programs to nodes) to minimize the objective function shown in equation 2.5. The allocation of x datasets and y programs to n nodes requires evaluation

of $2^{(x+y)n}$ choices. The following observations reduce the number of choices:

- The problem stated in section 2.3 can be decomposed into two subproblems: a program allocation and a data allocation subproblem. Thus the choices are reduced to $2^{xn} + 2^{yn}$. Proof of this decomposition is given in Appendix A.
- The programs can be assigned to all the nodes since there is no additional cost incurred due to program replication (see assumption 4 in section 2.4). Thus only data needs to be allocated, i.e. the choices are reduced to 2^{xn}.
- The datasets and programs can be grouped together to further reduce the number of choices. Several clustering techniques can be employed to form groups. For example, the x datasets can be clustered together to form g groups, where $g < x$, thus reducing the choices to 2^{gn}.
- The data allocation can be decomposed to subproblems. Appendix B shows that the allocation of g datasets, or groups, can be decomposed into g independent allocation subproblems, thus reducing the choices to $g\ 2^{n}$.
- A further reduction in the data allocation is achieved by using Casey's <CASE72c> "local search routine" which quickly finds the optimal allocation of a single file in an n node network within z evaluations where $z < n$. Appendix C shows that the data allocation problem at this level can be transformed to Casey's formulation, thus enabling us to use his results.

In short, the above stated decompositions and approximations reduce the maximum number of choices (allocations) to be evaluated from $2^{(x+y)n}$ to $g \times 2^{z}$, where $g < x$ and $z < n$. Figure 2.7 shows the optimal allocation algorithm which uses these approximations and decompositions to solve

FIG (2.7) : THE UNCONSTRAINED ALLOCATION ALGORITHM



the problem stated in section 2.3. As can be seen the algorithm consists of three parts: the grouping scheme, the program allocation scheme and the data allocation scheme.

2.6.1 The Grouping Scheme

The grouping scheme is based on standard clustering techniques (see for example <EVER74f, DURA74f>). Although the literature on clustering techniques is scattered among several different fields, the basic procedure consists of choosing and computing an appropriate similarity or dissimilarity (distance) function and then utilizing hierarchical, mathematical programming, mode-seeking or clumping techniques for "fusing" similar objects. Unfortunately, no general guidelines for choosing proper similarity/ distance functions and/or the "fusing" techniques exist. However, Everret <EVER74f> presents several practical hints and tradeoffs. Figure 2.8 shows a three step grouping scheme which builds proximity and similarity relations for programs and datasets in the first two steps and uses a hierarchical clustering technique in the third step.

STEP 1).A proximity relationship is developed which shows the commonality between various programs and datasets. Within the context of this research, the proximity between two programs p and p1 can be expressed in terms of common data accessed, information exchanged $P_m(p,p1)$ between p & p1 and commonality of activating transactions $T_e(t,p)$. Basically, p & p1 should be grouped if they access the same datasets, pass information among themselves and/or are activated by the same transaction. Formally, we use the following proximity relationship q(p,p1) between programs p & p1:

$$q(p,p1)=\frac{\Sigma_d(\min(Z(p,d),Z(p1,d))) + \Sigma_t(\min(T_e(t,p),T_e(t,p1)) + P_m(p,p1)}{\Sigma_d(Z(p,d) + Z(p1,d)) + \Sigma_t(T_e(t,p) + T_e(t,p1)) + P_m(p,p1)} \quad (2.10)$$

where Z(p,d) = no. of accesses from program p to dataset d

$$=(P_r(p,d) + P_w(p,d))$$

and P_r, P_w, P_m and T_e are application system parameters (table 2.1). This relationship is an extension of the following proximity relationship used by Buckles and Harding <BUCK79c> to cluster p1 and p2:

$$= \frac{\Sigma_d \ (\min \ (Z(p,d), \ Z(p1,d)))}{\Sigma_d \ (Z(p,d) + Z(p1,d))}$$

This relationship shows that the proximity between p1 and p2 is high if p1 and p2 frequently access the same dataset d (the numerator is high in such cases). We have extended this formulation to include the effect of interprogram relationships, by introducing the P_m term, and the effect of transaction to program relationships, by introducing the P_e terms, in equation (2.10).

Similarly, the proximity relationship q(d,d1) between datasets d & d1 can be expressed as:

$$q'(d,d1) = \frac{\Sigma_p \ (\min \ (Z(p,d), \ Z(p,d1)) + \Sigma_p \ P_c(p,d) \ D_c(d,d1)}{\Sigma_p \ (Z(p,d) + Z(p,d1)) + \Sigma_p \ P_c(p,d) \ D_c(d,d1)} \quad ..(2.11)$$

where $D_c(d,d1)$ shows the consistency relationships between datasets d & d1 and $P_c(p,d)$ is the number of consistency updates defined above. Note that $P_c(p,d)$ is used to represent the strength of the consistency relationship. Figure 2.8 shows the proximity relationships for a two program, 3 dataset application system.

STEP 2).The proximity relationships are extended to include the effect of model assumptions and restrictions and then the similarity matrices are calculated. For example, q(p,p1) =1 for all p and p1 if programs are to be replicated. Similarly if a program cannot access remote data, due to the query processing options to be discussed in chapter 4, then q'(d,d1)=1 for all d & d1 which are accessed by the same

FIG (2.8).: THE GROUPING EXAMPLE

Consider the student registeration example (two programs p1 and p2, three data-sets d1, d2, d3) presented in section 2.2.1. For simplicity, we assume $P_m(p,p1) = 0$ and $\min(T_a(p),T_a(p1))=0$ for all p, and build the following Z(p,d) matrix from the parameters shown in figure 2.1c:

	d1	d2	d3
p1	40	30	0
p2	50	60	10

i). THE PROXIMITY RELATIONSHIPS

Based on this Z(p,d), we calculate q and q' from equation (2.10).
q is the program proximity relationship and q' is the data proximity relationship:

$$q = \begin{array}{c|cc} & p1 & p2 \\ p1 & 1 & .77 \\ p2 & .77 & 1 \end{array} \qquad q' = \begin{array}{c|ccc} & d1 & d2 & d3 \\ d1 & 1 & .88 & .2 \\ d2 & .88 & 1 & .2 \\ d3 & .2 & .2 & 1 \end{array}$$

ii). MODIFY & DEVELOP THE SIMILARITY RELATIONS

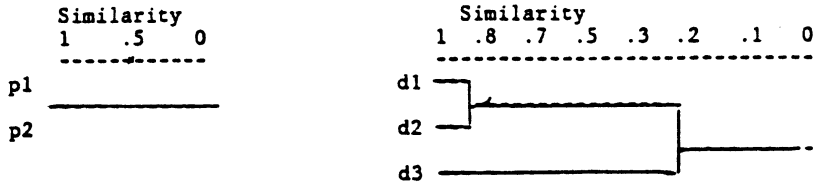
Since all the programs are replicated $q = \begin{array}{cc} 1 & 1 \\ 1 & 1 \end{array}$

The modified program proxmity relationship q is transitive but the proximity relationship q' is not transitive, however q' X q'= e' is transitive. Thus we get the following similarity relationsips:

$$e = \begin{array}{c|cc} & p1 & p2 \\ p1 & 1 & 1 \\ p2 & 1 & 1 \end{array} \qquad e' = \begin{array}{c|ccc} & d1 & d2 & d3 \\ d1 & 1 & .88 & .2 \\ d2 & .88 & 1 & .2 \\ d3 & .2 & .2 & 1 \end{array}$$

iii). BUILD THRESHOLD AND GROUP

TH, the group average threshold for programs, is 1 and TH', the group average threshold for data, is 0.677.
The dendograms for the two similarity relations are:



Clustering all the entities with similarity greater than the threshold, we get the following groups:

Process group= (p1, p2)
Data Groups = (d1,d2) and (d3)
(d1 and d2 are clustered since the similarity between these two elements is 0.88, which is greater than 0.67)

Note that due to this grouping algorithm, the 5 assignable units (2 program, 3 data records) are reduced to 3 assignable units (1 program group, 2 data groups). Thus grouping reduces the size of the allocation problem.

program. The computed proximity relationships are always reflexive (q(i,i)= 1) and symmetric (q(i,j)= q(j,i)) but not necessarily transitive (q(i,k)> Max (Min (q(i,j),q(j,k))). According to the "fuzzy set theory" <ZADE71f, WANG78f>, similarity relationships must be reflexive, symmetric and transitive (some authors, for example <DURA74f>, do not state transitivity to be a necessary condition). Zadeh <ZADE71f> has shown that similarity relationships e and e' can be constructed from any proximity relations q and q' by:

$$e(p,p1) = (q(p,p1)) + (q(p,p1))^2 + .. + (q(p,p1))^n \quad .....(2.12)$$

$$e'(d,d1) = (q'(d,d1))+ (q'(d,d1)^2+ ... + (q'(d,d1))^{n'} \quad .....(2.13)$$

Where n ≤ size of the proximity matrix q. Since each element of e and e' is between (0,1), hence special instructions for multiplication and addition are introduced (sum of two elements is the maximum and multiplication is the minimum). Figure 2.8 shows the similarity matrices for the sample q and q'.

STEP 3). Once a similarity matrix has been obtained, then a hierarchical clustering mechanism can be built which forms a cluster between two "nearest" objects based on similarity, then the next most similar object joins the cluster etc, until all the objects are grouped into a single cluster. A stopping rule, or threshold, is usually needed so that all the objects with similarity greater than the threshold are grouped. A wide variety of dendograms (trees) can result from this method <OHEU79f, DURA74f>. To avoid clustering between weakly interacting objects, we use the "group average" thresholds TH and TH' as a stopping rule:

$$TH = \Sigma_p \Sigma_{p1} e(p,p1)/(P_n \times P_n) \text{ and } TH' = \Sigma_d \Sigma_{d1} e(d,d1)/(D_n \times D_n) \quad .....(2.14)$$

where P_n and D_n are the maximum number of programs and datasets, respectively, in the application system. The grouping threshold can also be specified externally by a user to adjust the number of groups to be

formed. This is an important feature of the grouping scheme and will be used to evaluate the performance of the grouping approximation (see chapter 3). Figure 2.8 shows the dendogram, the calculated threshold and the resulting groups formed.

2.6.2 The Program Allocation Scheme

It is shown in appendix A that the program/data allocation problem stated in section 2.3 can be decomposed into two subproblems: a program allocation subproblem which determines the program allocations $\beta(p,i)$ to minimize the node to program and program to program remote cost given by $\Sigma_t\Sigma_k\Sigma_i\Sigma_j(R_{np}(t,k,i,j)+R_{pp}(t,k,i,j))C_R(i,j)$ subject to $\Sigma_i\ \beta(p,i) \geq 1$ for all $p \in P$, and a data allocation algorithm which determines the data allocations $\alpha(d,i)$ to minimize the storage plus local processing plus the program to data remote costs subject to the $\beta(p,i)$ determined by the program allocation problem.

This scheme solves the first subproblem and is trivial for the unconstrained problem because we can assign programs to all the nodes (set $\beta(p,i)=1$ for all p and i) due to the assumptions of homogeneous software, infinite processor and storage capacity and negligible disk storage of programs, thus minimizing the node to program and program to program cost ($R_{np} + R_{pp}$). The implemented scheme allows us to study the effect of given program allocations on total cost by allowing a user to externally specify program allocations to be evaluated. The scheme basically computes the program sites $\beta_s(k,p,i)$ for program p. In case of replicated programs, $\beta_s(k,p,i)=1$ for k=i. However, if program p is externally allocated by a user to, say, node 3, then $\beta_s(k,p,i)=1$ for i=3, 0 otherwise.

2.6.3 The Data Allocation Scheme

This algorithm attempts to solve the data allocation subproblem stated in appendix A, i.e. determine the data allocations $\alpha(d,i)$ which minimize the sum of storage, local and program to data cost:

$$OB2=\Sigma_t\Sigma_k\Sigma_i\Sigma_j C_R(i,j)T_a(t,k)R_{pd}(t,k,i,j)+\Sigma_t\Sigma_k\Sigma_i C_L(i)T_a(t,k)L(t,k,i)+\Sigma_i C_S(i)S(i$$

subject to:

. $\beta(p,i)$ determined by the program allocations

. $\Sigma_i\ \alpha(d,i) \geq 1$ for all $d \Xi D$

It was shown in section 2.6 that O2 is a nonlinear integer function of data allocation [1].Thus the problem can be theoretically solved by using standard linearizing techniques <PLAN71f, MODER78f>, however a prohibitively large number of constraints can result from this technique. Appendix B shows that this problem can be decomposed into D_g independent subproblems, where D_g is the total no. of data groups generated by the grouping scheme, if the consistency updates can be ignored ($D_c(d,d1) = 0$ and/or $P_c(p,d) = 0$). This approximation to achieve decomposition will be referred to as the "weak-consistency" approximation. If the weak-consistency does not apply, then we attempt to achieve decomposition by introducing a sequencing routine which arranges the data groups in such a fashion that the groups with no consistency relationships are allocated first and the groups with strong consistency interactions are allocated at the end. This routine labels the dataset groups (d,d',d",,,) as (d1,d2,d3,,,) where d1 is the first group to be allocated, d2 is the next and so on. We suggest the following algorithm for the sequencing routine:

1. To determine d1:

1.In this section, and later on, we will not explicitly differentiate between a dataset and a dataset group, unless necessary, because a dataset group has the same properties as a dataset.

. Choose a d such that $D_c(d,d') = 0$ for all d'.

. If none, then choose a d for which $P_c(p,d) = 0$ for all p.

. If none, then choose a d for which $D_c(d,d') \neq 0$ for all d' that have already been sequenced.

. If not possible, then choose d for which $P_c(p,d')$ is small for all d' with $D_c(d,d') \neq 0$.

2. Repeat the same procedure for d2, d3 etc.

Sequencing/labelling techniques like this have been used frequently in heuristics. For example, the greedy heuristic for the knapsack problem sorts the items by descending price/volume ratio for allocation <FISH80f>. If sequencing is 100 percent successful (group to be assigned at stage n is only related to groups assigned in the n-1, n-2,,,,1 stages) then the problem is separable and decomposable (see <NEMH66f>). If not, sequencing provides a good approximation. It should also be recalled that the proximity relationships developed in the grouping scheme tend to group data objects with consistency relationships, thus reducing these relationships between data groups generated. The effect of these approximations will be formally examined in chapter 3.

The data allocation scheme consists of two algorithms: the global allocation algorithm which subdivides the overall allocation problem into several stages where a dataset group is chosen for allocation in each stage, and the local allocation algorithm which determines the optimal allocation of the chosen dataset group to the nodes in each stage:

i).GLOBAL ALLOCATION: A dynamic programming based algorithm is chosen which allocates d1 in stage 1, d2 in stage 2, etc. The stages, states, decisions, returns and recursive equations of the dynamic programming framework are shown in figure 2.9a. Since the problem is unconstrained, the recursive equation for Y(n), the accumulated return at stage n (allocation of data group n) becomes:

$$Y(n) = OB[a(n), a^*(n-1),,,,,,a^*(1)]$$

Where OB[a(n),a(n-1),,,a(1)] is the objective function based on decisions a(n), a(n-1),,,, a(1) and a(n), the data allocation decision at stage n, is a vector of decision variables α:

a(n) = [α(dn,1),α(dn,2),α(dn,3),,,,]

The main feature of this algorithm is that the optimization of Y(n) is performed on [a(n), a^*(n-1),,,a^*(1)] rather than a(n). Thus in the case of decomposable objective functions, the algorithm exhibits standard dynamic programming features and yields exact solutions, while in the case of nondecomposable situations it becomes a hierarchical heuristic procedure in which the optimal decision made at stage n-1 becomes a constraint at stage n.

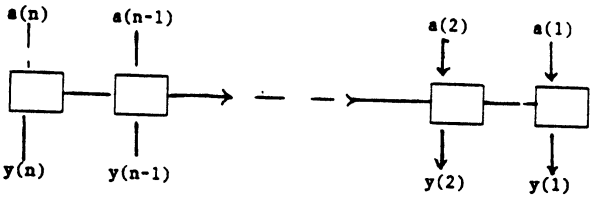
ii). LOCAL ALLOCATION: At stage n, Y(n) needs to be optimized. Optimization of Y(n) is very similar to the single file allocation problem addressed by Casey<CASE72c>. Appendix C shows that this problem can be formally transformed into Casey's formulation. This result is significant from two points of view a).Casey showed that the single file allocation algorithm can be represented as a cost access graph (a hypercube) which can be searched quickly for optimal points without having to search the entire graph. <MORG77c, SURI79c> have also used this technique. b). Eswaran <ESWA74c> has shown that Casey's formulation is polynomial complete and can be only solved by heuristics. Thus by showing that our allocation algorithm can be transformed to Casey's formulation, we indirectly prove that the problem is polynomial complete and can be solved by heuristics.

Because of the transformation shown in appendix C, we can directly use Casey's vertical search routine which can quickly find the exact optimal for Y(n). Figure 2.9b shows the cost access graph for a single dataset alllocation to a 3 node network and helps to define the following terms used in the search routine:

- The vertex 001 shows that the dataset is not allocated to nodes and 2, indicated by 00, but is allocated to node 3.

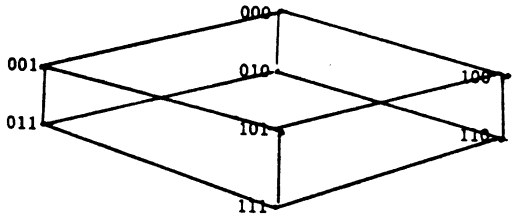
FIG (2.9) : DATA ALLOCATION SCHEME

a). DYNAMIC PROGRAMMING FRAMEWORK



x(n) = State variable at stage n
a(n) = Data allocation decision at stage n
= Allocation of data group n to nodes
y(n) = Return at stage n
x(n) = t(x(n), a(n))
= state transition function
$F_n(x(n))$ = Accumulated optimal return at stage n
= Min $(y(n) + F_{n-1}(x(n-1)))$
Y(n) = Accumulated return at at stage n = $y(n) + F_{n-1}(x(n-1))$

b). COST HYPERCUBE FOR SINGLE FILE ALLOCATION TO 3 NODES



• Successors J_n of vertex n contain vertices of n plus one more node. For example (101,011) are successors of (001).

• Antecedents H_n of vertex n contain vertices of n less 1 node. For example, (001,010) are antecedents of vertex 011.

• Admissable set K_r is the set of vertices at level r which may yield optimal return

• A vertex is "used" if it has been evaluated by a search routine.

The routine is now described.

.At stage n:

1. Start at level r=1 and store all vertices at level r=1 into the admissable set K_r. Initialize s, the level 1 counter, to 0.
2. Set s=s+1, where $s \leq N_n$,the number of nodes. (s sets an antecedent at level 1 for all the verical paths.)
3. Pick the sth vertex $v \in K_r$ and determine J_v , the

set of successors for vertex v at level r+1.

4. For the chosen v, select a vertex $v' \in J_v$ such that v' is admissable and usable.

5. Evaluate v'. If cost at v' ≥ cost at v, then:

   . Mark v' inadmissable.

   . Mark all successors of v' inadmissable.

   . Set r=1 and go to step 3.

   Otherwise (cost v' < cost v):

   . Mark v' used.

   . Set v = v'.

   . If r ≥ N then set r=1 and go to step 3,

     else set r=r+1 and go to step 3.

6. If $v' \in J_v$ cannot be found, this means all successors of v have been evaluated or are inadmissable:

   . If r≠1, then set r=1 and go to step 3.

   . If r=1, then find the optimal vertex v^*_s for all succesors of sth vertex and go to step 2.

7. When all vertices at r=1 have been evaluated (s ≥ N), then determine v^* (over all s) which minimizes cost. This is the optimal assignment at stage n, and minimizes Y(n).

Basically, this search routine travels the hypercube vertically along a path until the cost increases for a given vertex. At this point, the vertex and all its successors are marked inadmissable and a new vertical path is followed (backtracking). This routine can be modified to include greedy heuristics, and constraint checking. We will deal with these extensions later.

The algorithm discussed in this section and shown in figure 2.7 has been implemented in Fortran as a single program and has been tested on an IBM 3033 system under the MVS operating system. Details of this program will be given in chapter 6.

2.7 Chapter Summary and Significance

In this chapter, the effect of application system characteristics on the program/data allocations is studied in depth by assuming an idealized support system. Several parameters are introduced which can describe, existing or proposed, centralized or distribubed application systems. These parameters are a significant improvement over the traditional three parameters (node to file query traffic, node to file update traffic and file size) used in most allocation schemes, and thus enhance the applicability of this model to real life situations involving many to many relationships between transactions, programs and datasets. The main contribution of this chapter is that detailed expressions for storage occupied, local processing and remote processing are derived in terms of application system parameters and an unconstrained allocation algorithm is developed which can be easily extended later on. In addition, it is shown that for unconstrained program and data allocation, the program allocation problem is trivial and the total cost shows a covex behaviour with respect to data allocation.

CHAPTER 3

ANALYSIS OF THE UNCONSTRAINED ALLOCATION ALGORITHM

The unconstrained allocation algorithm presented in chapter 2 was based on the "structural" (modelling) assumptions and the optimization approximations of grouping and weak-consistency used to speed up the allocation algorithm. In this chapter we are not concerned with the structural assumptions, since any system that does not satisfy the explicit assumptions stated in section 2.4 cannot be adequately modelled by the unconstrained problem. Instead we focus on the optimization error introduced by the grouping and weak-consistency approximations which considerably reduce computation time but may produce suboptimal solutions. Two basic approaches are available for evaluation:

1. EMPIRICAL ANALYSIS: The algorithm is programmed and then applied to a "representative" sample of problems to demonstrate its performance of the algorithm in terms of optimal solutions reached and deviations from the optimal return.
2. ANALYTICAL MODELLING: The behaviour of the algorithm is evaluated based on analytical methods. Two such approaches are available at present: i) worst-case analysis which establishes the maximum deviation from optimality and ii) probabilistic analysis which establish the probabilistic properties of the algorithm based on an assumed density function for the problem data and produces average performance of the algorithms.

Most of the available distributed allocation algorithms and heuristics have been evaluated solely through empirical analysis. Analytical (worst-case, probabilistic) as well as empirical analysis of the unconstrained allocation algorithm are presented in this chapter.

3.1 Worst-Case Analysis

Let us examine the basic principles of worst cas analysis <GAREY79f, FISH80f>. Let U denote the set of problem instances, $I \Xi U$ a particular problem instance, Z(I) the optimal value of problem I and $Z_H(I)$ the value obtained when algorithm (heuristic) H is applied to the problem. It is assumed that algorithm H is well specified to uniquely define $Z_H(I)$. In terms of the unconstrained program and data allocation problem stated in section 2.3, I is defined by a particular instance of an application system and a DDP support system and we use H to denote the unconstrained allocation algorithm. Worst-case analysis establishes the maximum deviation of $Z_H(I)$ from Z(I). Let ε denote the deviation (error):

$$Z_H(I) - Z(I) = \varepsilon$$

where $\varepsilon \geq 0$ since the allocation problem is formulated as a minimization problem. The worst-case deviation, ε^*, is given by:

$$Z_H(I) - Z(I) \leq \varepsilon^*$$

A percentage measure of worst-case error is given by $\varepsilon^*/Z(I)$. The purpose of worst-case analysis is to determine ε^* for each H.

There are basically two heuristic schemes used: the grouping scheme and the data allocation scheme. The program allocation scheme is trivial since it automatically replicates programs. The grouping scheme, referred to as algorithm HG in this chapter, uses the grouping approximation which clusters datasets based on proximity relationships. The data allocation scheme, referred to as algorithm HD in this chapter, primarily uses the weak-consistency approximation to achieve global decomposition (assuming the effect of consistency relationships is small)

and to use Casey's results for the local search (Appendix B and C). Thus we need to determine ε^* for the weak-consistency and the grouping approximations. For notational simplicity, we will drop the argument I on $Z_H(I)$ and $Z(I)$ for the balance of this section.

3.1.1 Worst-Case Analysis of the Weak-Consistency Approximation

THEOREM: The worst-case deviation, ε_D^*, of the data allocation algorithm HD is given by:

$$Z_{HD}-Z \le \Sigma_k\, T_a(k)\, \Sigma_p T_e(p)\, P_b(p) \Sigma_d\, P_c(p,d)\, \Sigma_{d1} D_c(d,d1) \times \Big[\sum_{j=1}^{N_n-1} C_L(j) + \sum_{i=1}^{N_n-1} \delta(k,i) \sum_{j=1}^{N_n-1} C_R(i,j)\Big] \quad (3.1)$$

where $\delta(k,i) = 1$ if $k=1$, 0 otherwise and all other parameters specify an application system and have been defined in table 2.1. N_n is the number of nodes.

PROOF: Let $\lambda(d)$ = cost (local + remote) due to consistency updates generated whenever dataset d is updated. Then by examining equations 2.3 and 2.4c, we get:

$$\lambda(d)=\Sigma_k\, T_a(k)\Sigma_p\, T_e(p)P_b(p)[P_c(p,d)\Sigma_{d1}D_c(d,d1)\Sigma_i\, \alpha(d1,i)C_L(i) + P_c(p,d)\Sigma_{d1}\, D_c(d,d1)\, \Sigma_i\, \Sigma_j\, \beta_s(k,p,i)\alpha(d1,j)C_R(i,j)] \quad (3.2)$$

Let $OB(1,2,,D_n)$ denote the objective function due to the allocation of datasets $(1,2,,D_n) \Xi D$ with $\lambda(d) = 0$ for all $d \Xi D$. Then, OB can be decomposed (as shown in appendix B):

$$\text{Min } OB(1,2,,D_n) = y^*(1) + y^*(2) +....+ y^*(D_n)$$

where $y^*(d)$ denotes the optimal return due to allocation of d. Note that $y^*(d)$ reflects the exact optimal allocation when no consistency costs are involved ($\lambda(d)=0$ for all $d \Xi D$). Let $OB1(1,2,,D_n)$ denote the objective function when $\lambda(d) = 0$ for m datasets, but $\lambda(d) \neq 0$ for the remaining D_n - m datasets, where $m < D_n$ and $m \ge 0$. Then, since λ introduces additional cost due to consistency updates.

$$\text{Min } OB1(1,2,,D_n) = y^*(1) + y^*(2) +....+ y^*(D_n) + \theta \quad (3.3)$$

where $\theta > 0$. We need to determine θ in terms of HD. Since HD is a hierarchical heuristic which determines optimal allocation at each stage, we get:

$$\text{Min } OB1(1,2,,,D_n) = y'^*(1) + y'^*(2) + \ldots + y'^*(D_n)$$

where $y'^*(d)$ denotes the optimal return calculated at stage d by HD. Since m datasets do not have consistency relationships, the sequencing routine described above places these datasets in first m stages and consequently $y'^*(d) = y^*(d)$ for $d = (1,2,3,,m)$:

$$\text{Min } OB1(1,2,,,D_n) = y^*(1)+y^*(2)+ \ldots +y^*(m)+y'^*(m+1)+ \ldots + y'^*(D_n)$$

Now, $y'^*(d) = y^*(d) + \lambda(d)$, where $\lambda(d)$ is the consistency cost portion of $y'^*(d)$.

$$\text{Min } OB1(1,2,,,D_n) = y^*(1)+y^*(2)+ \ldots+y^*(D_n)+ \sum_{d=m+1}^{D_n} \lambda(d) \qquad (3.4)$$

Comparing equations 3.3 and 3.4, we get (note that $\sum_{d=1}^{D_n} \lambda(d) = \Sigma_d \lambda(d)$):

$$\theta = \sum_{d=m+1}^{D_n} \lambda(d) = \Sigma_d \lambda(d) \qquad (3.5)$$

Substituting θ in equation 3.3, we get:

$$\text{Min } OB1(1,2,,,D_n) = y^*(1) + y^*(2) +\ldots+ y^*(D_n) + \Sigma_d \lambda(d) \qquad (3.6)$$

This equation suggests the upper bound on Z_{HD}, the minimum value of OB1, generated by HD:

$$Z_{HD} \leq y^*(1) + y^*(2)+ \ldots +y^*(D_n) + \max \Sigma_d \lambda(d) \qquad (3.7)$$

where λ is given by equation 3.2. This expression is based on the observation that the highest value of θ, in equation 3.5, is $\max \Sigma_d \lambda(d)$.

Determination of Z, the exact minimum of OB1 for $\lambda(d) \neq 0$, is difficult to calculate as a general case. Instead of finding the exact value of Z, we determine the lower bound on Z, a technique used frequently in worst-case analysis (see <FISH80f>). Once again, equation 3.5 gives the lower bound on Z:

$$Z \geq y^*(1) + y^*(2) + \ldots\ldots + y^*(D_n)+ \min \Sigma_d \lambda(d) \qquad (3.8)$$

This expression is valid since the minimum value of $\theta = \min \Sigma_d \lambda(d)$. Using equations 3.7 and 3.8, we get the worst case performance of Z_{HD}:

$$Z_{HD} - Z \leq \max \Sigma_d \lambda(d) - \min \Sigma_d \lambda(d) \qquad (3.9)$$

It can be seen from equation 3.2 that λ is a monotonically increasing function of the consistency related data duplication $\alpha(d1,i)$. So λ is minimum when consistency related data d1 is allocated to only one node and maximum when d1 is fully replicated. Thus by substituting the values of min λ and max λ in equation 3.9 and simplifying, we get the final expression (equation 3.1) for ε_D*:

$$Z_{HD}-Z \leq \Sigma_k T_a(k) \Sigma_p T_e(p) P_b(p) \Sigma_d P_c(p,d) \Sigma_{d1} D_c(d,d1)$$
$$\times \left[\sum_{j=1}^{N_n-1} C_L(j) + \sum_{i=1}^{N_n-1} \delta(k,i) \sum_{j=1}^{N_n-1} C_R(i,j) \right]$$

We have replaced $\beta_s(k,p,i)$ with $\delta(k,i)$ because programs are assumed to be duplicated. (Q.E.D)

It can be seen from equation 3.1 that the worst-case performance of HD is a function of:

.The number of nodes N_n in the DDP network.

.Application system parameters like the consistency relationships $D_c(d,d1)$, the number of consistency updates $P_c(p,d1)$ and other application parameters shown in table 2.1.

.The costs C_L and C_R.

To illustrate this result, let us consider the allocation of the two program, three dataset student registeration system to a 5 node network (figure 3.1). The parameters used in table 2.2 will be used, however to make some calculations quicker, we will use the following simplifying information:

. The transaction arrival rate, $T_a(k)$, is 1 per minute at all the nodes, i.e. $T_a(k) = 1$ for all $k \Xi N$.

. The DROP transaction activates program p1 and p2, i.e. $T_e(p) = 1$ for all $p \Xi P$.

. The programs receive only one input record, i.e. $P_b(p)=1$ for all $p \Xi P$.

. A consistency relationship exists between datasets 2 and 3,

i.e. $D_c(d,d1)=1$ for d=2 and d1=3, 0 otherwise.

. All the local processing costs are 2 cents per service and remote costs are 5 cents per service, i.e. $C_L(i)=2$ for all $i \in N$, and $C_R(i,j)=5$ for all $i,j \in N$.

Substituting these values in equation 3.1 and simplifying, we get:

$$\varepsilon_D^* = Z_{HD} - Z \leq (N_n-1)(C_L+C_R) \Sigma_p P_c(p,2)$$

Noting that $C_L = 2$, $C_R = 5$ and $N_n=5$, and letting $P_c(p,2)=1$ for p=1, we get the worst-case difference ε_D^* of 28. To get a percentage ratio, we can calculate ε_D^*/Z. Using this numeric example, Z = 6340. Thus the worst-case deviation of HD is about 0.4 percent. However, if the consistency updates $P_c(p,2)$ are increased to 10, then $\varepsilon_D^* = 280$, Z=6700 and worst-case deviation is about 4%. Testing of this formula has shown that for 15 node networks with moderate consistency updates (less than 30% of the reads), the worst-case deviation due to the weak-consistency approximation ranges between 10 to 15 percent. However, for extremely high consistency relationships and large networks with more than 50 nodes, the worst-case deviation can reach 30%. The grouping approximation, discussed below, can be used in such cases to cluster consistency related datasets and thus reduce the weak-consistency error.

3.1.2 Worst-Case Analysis of Grouping Approximation

THEOREM: The worst-case deviation, ε_G^*, of the grouping algorithm HG presented in section 2.7.1 is given by:

$$Z_{HG} - Z \leq SD + LD + RD \tag{3.20}$$

where:

- SD= difference in storage cost due to grouping

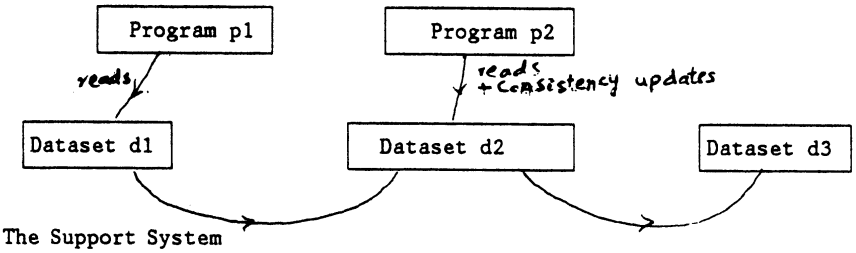
$$= \Sigma_d S_a(d) \sum_{i=1}^{N_n-1} C_S(i) \tag{3.21}$$

LD= difference in local processing cost due to grouping

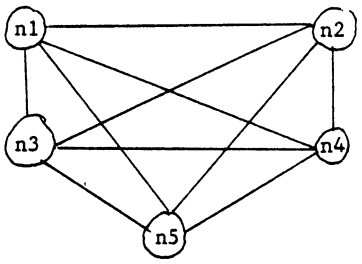
$$= \Sigma_d L_w(d) \sum_{i=1}^{N_n-1} C_L(i) + \Sigma_d L_r(d) C'_L \tag{3.22}$$

FIGURE 3.1: TWO PROGRAM, THREE DATASET ALLOCATION TO FIVE NODES

a). The Application System



b). The Support System



c). PARAMETERS DESCRIBING THE APPLICATION SYSTEM

i). DATABASE PARAMETERS:

$D_n = 3$

$D_c(d1,d2)$

d1 \ d2	1	2	3
1	0	1	0
2	0	0	1
3	0	0	0

$D_s(d,x)$

d \ x	1	2
1	10	10
2	10	10
3	10	10

ii). PROGRAM PARAMETERS:

$P_n = 2$

$P_r(p,d)$

p \ d	1	2	3
1	60	0	0
2	0	80	0

$P_w(p,d)$

p \ d	1	2	3
1	0	0	0
2	0	0	0

$P_c(p,d)$

p \ d	1	2	3
1	0	0	0
2	0	4	0

$P_l(p)$

p	1	2
	1	1

$P_o(p)$

p	1	2
	1	12

$P_m(p1,p2)$

p1 \ p2	1	2
1	0	1
2	0	0

iii). TRANSACTION PARAMETERS:

$T_n = 1$

$T_e(p)$

p	1	2
	1	1

$T_a(k)$

k	1	2	3	4	5
	3	2	3	4	1

RD= difference in the remote processing cost due to grouping

$$= \Sigma_d R_w(d) \sum_{i=1}^{N_n-1} \sum_{j=i}^{N_n-1} C_R(i,j) \Sigma_k \delta(k,i) + \Sigma_d R_r(d) C'_R \quad (3.23)$$

where $S_a(d)$ is the amount of storage occupied by dataset d, $L_r(d)$ and $L_w(d)$ are the local read and write activities, respectively, for dataset d, $R_r(d)$ and $R_w(d)$ are the remote read and write activities , respectively, for dataset d, and all other parameters have been defined in table 2.1. C'_L and C'_R designate the highest resource utilization costs for local and remote processing, respectively, for a given DDP network. The expressions for S_a, $L_r(d)$, $L_w(d)$, $R_r(d)$, $R_w(d)$ are:

$S_a(d)$ = storage occupied by dataset d

$$= \Sigma_x D_s(d,x) \quad \text{where } x=1 \text{ and } 2 \quad (3.24)$$

$L_r(d)$ = local service requests due to reads of dataset d

$$= \Sigma_k T_a(k) \Sigma_p T_e(p) \Sigma_i P_r(p,d) \quad (3.25)$$

$L_w(d)$ = local service requests due to writes (modify) of dataset d

$$= \Sigma_k T_a(k) \Sigma_p T_e(p) P_w(p,d) \quad (3.26)$$

$R_r(d)$ = remote service requests due to reads of dataset d

$$= \Sigma_i T_a(k) \Sigma_i T_e(p) P_r(p,d) \quad (3.27)$$

$R_w(d)$ = remote service requests due to writee of dataset d

$$= \Sigma_i T_a(k) \Sigma_i T_e(p) P_w(p,d) \quad (3.28)$$

PROOF: Every grouping can potentially produce an error in the objective function because the grouped datsets must be allocated to the same nodes. The worst-case for HG occurs when all the datasets $(1,2,,,D_n) \Xi D$, are grouped into a single dataset group G and is given by:

$$Z_{HG} - Z \leq \text{Min } OB2(g) - \text{Min } OB(1,2,,,,D_n)$$

where OB2(g) represents the objective function due to the allocation of group $g \Xi G$ and $OB(1,2,3,,,D_n)$ represents the objective function due to the allocation of datasets $(1,2,,,D_n) \Xi D$. Derivation of equation 3.20 is straightforward, since OB2 as well as OB are expressed as sums of storage, local processing and remote processing costs. The expressions for the differences in costs (SD, LD, RD) are derived below by using equations 2.2, 2.3 and 2.4. We have used $\alpha^*(d,i)$ to show the optimal

data allocation which produces Z and $\alpha''^*(g,i)$ to show the optimal allocation of group g which produces Z_{HG} ($g \Xi G$ is only one element). The variables $\alpha''_s(p,g,i)$ and $\alpha_s(p,d,i)$ are site selection matrices for g and d ($\alpha_s(p,d,i) = 1$ if node i is the site selected for dataset d when accessed from program p).

$$SD = \Sigma_i\, \alpha''^*(g,i) C_S(i)\, \Sigma_d\, S_a(d) - \Sigma_i\, \Sigma_d\, \alpha^*(d,i) C_S(i) S_a(d)$$

Simplifying this equation, we get:

$$= \Sigma_i\, \Sigma_d\, C_S(i) S_a(d) [\alpha''^*(g,i) - \alpha^*(d,i)] \,. \qquad (3.29)$$

$$LD = \Sigma_i C_L(i) [\alpha''^*(g,i) \Sigma_d L_w(d) + \Sigma_p\, \alpha''_s(p,g,i)\, \Sigma_d\, L_r(d)] - \Sigma_i\, \Sigma_d C_L(i) [\alpha^*(d,i) L_w(d) + \Sigma_p\, \alpha_s(p,d,i) L_r(d)]$$

Simplifying this equation, and letting $\alpha_s(p,d,i) = \alpha_s(d,i)$ for all $p \Xi P$, to reflect replication of program p, we get:

$$= \Sigma_i\, \Sigma_d\, C_L(i) [L_w(d)\{\alpha''^*(g,i) - \alpha^*(d,i)\} + L_r(d)\{\alpha''_s(g,i) - \alpha_s(d,i)\}] \qquad (3.30)$$

$$RD = \Sigma_i \Sigma_j C_R(i,j) \Sigma_k \delta(k,i) [\alpha''^*(g,j) \Sigma_d\, R_w(d) + \Sigma_p\, \alpha''_s(p,g,j) \Sigma_d\, R_r(d)] - \Sigma_i\, \Sigma_j\, \Sigma_d\, C_R(i,j) \Sigma_k \delta(k,i) [\alpha^*(d,j) R_w(d) + \Sigma_p\, \alpha_s(p,d,j)\, R_r(d)]$$

where $\delta(k,i)=1$ if $k=i$, 0 otherwise (this shows the arrival node).

Simplifying this equation, and letting $\alpha_s(p,d,i) = \alpha_s(d,i)$ for all $p \Xi P$, to reflect duplication of program p, we get:

$$= \Sigma_k\, \Sigma_i\, \Sigma_j C_R(i,j) \delta(k,i) [\Sigma_d\, R_w(d) \{\alpha''^*(g,j) - \alpha^*(d,j)\} + R_r(d)\{\alpha''_s(g,j) - \alpha_s(d,j)\}] \qquad (3.31)$$

Equations 3.29, 3.30 and 3.31 are not very useful for worst-case analysis since the parameters α^*, α''^*, α_s and α''_s cannot be measured directly. In addition, these equations produce SD = LD = RD = 0 for some cases. For example, SD = LD = RD = 0, if we substitute the following values in equations 3.29 through 3.31:

$\alpha''^*(g,i) = \alpha^*(d,i)$ for all $d \Xi D$, $i \Xi N$, $g \Xi G$

$\alpha''_s(p,g,i) = \alpha_s(p,d,i)$ for all $p \Xi P$, $d \Xi D$, $g \Xi G$, $i \Xi N$

This situation occurs in practice when the optimal allocation of datasets without grouping coincides with the optimal allocation after

grouping and yields $Z_{HG} - Z = 0$ from equation 3.20. Clearly, this does not represent a worst-case situation and must be eliminated from equations 3.29 through 3.31.

In equation 3.29, a measure of worst-case performance is achieved when $[\alpha''^*(g,i) - \alpha^*(d,i)]$ is maximized. Since these two variables are 0-1 variables, the maximum allowable difference is N_n-1, where N_n is the number of nodes. The maximum difference of N_n is not allowed since a given dataset, or group, must be allocated to at least one node. Thus, eliminating α^* and α''^* from equation 3.29 we get:

$SD = \Sigma_d\, S_a(d) \sum_{i=1}^{N_n-1} C_S(i)$ (same as equation 3.21)

In equation 3.30, we need to analyze the L_w and L_r terms separately. As can be seen, the L_w term is similar to the storage cost SD shown in equation 3.29 (the write activity L_w increases monotonically''' with data allocation in a manner similar to storage cost) and is maximum when $[\alpha''^*(g,i) - \alpha^*(d,i)] = N_n-1$. However, the read activity L_r is only conducted at one selected site, i.e. $\Sigma_i\, \alpha''_s(p,g,i) = 1$ for each g and p, and $\Sigma_i\, \alpha_s(p,d,i) = 1$ for each d and p. The L_r term is maximized when $[\alpha''_s(p,g,i) - \alpha_s(p,d,i)] = 1$. Thus the maximum contribution of $L_r(d)$ is $L_r(d)C'_L$ where C'_L indicates the maximum local storage utilization cost. Now, eliminating α''^*, α^*, α_s, α''_s from equation 3.30, we get the equation:

$LD = \Sigma_d\, L_w(d) \sum_{i=1}^{N_n-1} C_L(i) + \Sigma_d\, L_r(d)\, C'_L$ (same as eq. 3.22)

The analysis of equation 3.31 is similar to the analysis used for equation 3.30. R_w and R_r are analyzed separately and elimination of α''^*, α^*, α''_s and α_s from equation 3.31 gives:

$RD = \Sigma_d\, R_w(d) \sum_{i=1}^{N_n-1} \sum_{j=1}^{N_n-1} C_R(i,j) \Sigma_k\, \delta(k,i) + \Sigma_d\, R_r(d)\, C'_R$ ..(same as 3.23)

(Q.E.D)

It can be seen from equations 3.20 through 3.23 that the worst-case performance of HG depends on the number of nodes N_n, the costs C_S, C_L and C_R, and the storage, local processing and remote processing activities S_a, L_r, L_w, R_r and R_w given by equations 3.24 through 3.28. All

these variables are input parameters or functions of input parameters shown in table 2.1.

To illustrate these results let us reconsider the student registeration system shown in fig 2.1 to be allocated to a 5 node network. Then by using the parameters shown in figure 2.1c, with the simplifying values used above (worst-case of HD), we get the following values of SD, LD and RD:

. Let the storage cost be .06 at all the nodes,

  i.e. $C_S(i) = .06$ for all $i \in N$. Then using equations 3.24 and 3.21, we get SD = 1000 × .06 = 60

. By using equations 3.25, 3.26 and 3.22, we get LD = 10×2 = 20.

  Note that $L_w(d) = 0$ for all $d \in D$ since no updates are are performed ($P_w(p,d) = 0$ for all p and d in figure 2.1c).

. By using equations 3.27, 3.28 and 3.23, we get RD = 50×5=250.
  Note that $R_w(d) = 0$ for all $d \in D$ since $P_w(p,d) = 0$ for all p and d in figure 2.1c.

Then the worst-case deviation of HG, ε_G^*, is given by equation 3.20 as (SD + LD + RD) = 276. The percentage deviation in this case is approximately 5 percent.

3.1.3 Worst-Case Analysis of the Combined Grouping and Weak-Consistency Approximations

THEOREM: The worst-case deviation, ε^*, of the program and data allocation algorithm H (discussed in section 2.7) which combines the grouping and weak-consistency approximations HD and HG, is given by:

$$Z_H - Z \leq MAX(\varepsilon_D^*, \varepsilon_G^*) \qquad (3.35)$$

where ε_D^* is the worst-case indicator for HD given by equation 3.1 and ε_G^* is the worst-case indicator for HG given by equation 3.20.

PROOF: The performance of the algorithm HD is dependent on the performance of HG. This important observation needs to be analyzed. HG,

the grouping scheme, clusters datasets due to proximiity (similarity) functions. Equation 2.11 in section 2.7.1 shows that the proximity relationship q(d,dl) depends on the consistency relationship D_c(d,dl) and the consistency updates P_c(d,dl), among other factors. Thus if D_c(d,dl) × P_c(d,dl) ≠ 0, then datasets d and dl could be be grouped into a group g by the grouping scheme. This grouping affects the data allocation algorithm in the following manner:

- HD treats g as a single dataset for allocation purposes. This, of course, reduces the number of stages in HD.
- D_c(d,dl) is eliminated, assumed to be zero, by HD if d and dl belong to the same dataset group g. Thus HG helps in REDUCING the consistency relationships.
- A reduction in consistency relationships reduces the deviation ε_D of HD. For example, if D_c(d,dl) = 0 for all d and dl, then ε_D = 0 because the weak-consistency approximation is not needed. (See the discussion of worst-case analysis of HD above.)

To quantitavely measure the effect of grouping approximation on the weak-consistency approximation, let w(d,dl) show if d and dl are clustered together:

w(d,dl) = 1 if datasets d and dl are NOT clustered, 0 otherwise

We can define a modified consistency relationship D'_c(d,dl), in terms of the actual consistency relationship D_c(d,dl) and w(d,dl):

$$D'_c(d,dl) = D_c(d,dl)\ w(d,dl)$$

Note that D'_c is zero if d and dl are clustered (u(d,dl)=0). We now derive the expression for the consistency cost λ in terms of D'_c and obtain the worst-case deviation of H, ε*. Let g designate a dataset group formed by HG, i.e. w(d,dl)=0 if d and dl belong to g and α(d,i)=α(dl,i) for all i ∈ N. Let λ"'(g) denote the cost (local + remote) due to consistency updates generated whenever dataset group g is updated. The an-

alytical expression for $\lambda'''(g)$ can be obtained from equations 2.3 and 2.4 by replacing $D_c(d,d1)$ with $D'_c(d,d1)$:

$$\lambda'''(g)=\Sigma_k T_a(k)\Sigma_p T_e(p)P_b(p)\Sigma_d[P_c(p,d)\Sigma_{d1}D'_c(d,d1)\Sigma_i\ \alpha(d,i)C_L(i) + P_c(p,d)\Sigma_{d1}D'_c(d,d1)\Sigma_i\ \Sigma_j\ \beta_s(k,p,i)\alpha(d1,j)C_R(i,j)] \quad (3.36)$$

Based on the arguments presented in the worst-case analysis for HD, we can write the upper bound on Z_H:

$$Z_H \leq y'''^*(g) + \text{Max}\ \lambda'''(g)$$

where $y'''^*(g)$ is the optimal return from the allocation of group g when $\lambda'''(g) = 0$ and $\lambda'''(g)$ is given by equation 3.36. The lower bound on Z is given by (see equation 3.8):

$$Z \geq y^*(1) + y^*(2) + \ .... + y^*(D_n) + \text{Min}\ \Sigma_d\ \lambda(d)$$

Thus, the worst-case performance of H is given by:

$$Z_H - Z \leq y'''^*(g) - (y^*(1)+y^*(2)+ .. +y^*(D_n)) +\text{Max}\ \lambda'''(g)-\text{Min}\ \Sigma_d\ \lambda(d)$$

By using the arguments presented in the worst-case analysis of HG we get:

$$Z_H - Z \leq SD + LD + RD + \text{Max}\ \lambda'''(g) - \text{Min}\ \Sigma_d\ \lambda(d) \quad (3.37)$$

where SD, LD and RD are given by equations 3.21, 3.22 and 3.23. The sum (SD + LD + RD) = ε_G* from equation 3.20. Let WD denote the deviation due to weak-consistency:

$$WD = \text{Max}\ \lambda'''(g) - \text{Min}\ \Sigma_d\ \lambda(d) \quad (3.38)$$

$\lambda(d)$ is shown in equation 3.2 and $\lambda'''(g)$ is given by equation 3.36. An examination of these equations shows that λ and $\lambda 1$ are monotonically increasing functions of dataset (group) allocation. Thus $\lambda(d)$ is minimum at the unique allocation of dataset d and $\lambda'''(g)$ is maximum when g is replicated. Substituting these values in equation 3.38 and simplifying, we get:

$$WD = \Sigma_k\ T_a(k)\ \Sigma_p T_e(p)\ P_b(p)\Sigma_d\ P_c(p,d)\ \Sigma_{d1}\ D'_c(d,d1) \times [\ \Sigma_j\ C_L(j) + \delta(k,i)\ \Sigma_i C_R(i,j)] \quad (3.39)$$

A comparison of equations 3.39 and 3.1 show that $WD = \varepsilon_D$* if $D'_c(d,d1) = D_c(d,d1)$. Let us define ε_G'* to designate the worst-case deviation of

HD given the grouping w, then WD = $\varepsilon_G'^*$. Substituting this value of WD and the values of SD, LD and RD in equation 3.37, we get:

$$\varepsilon^* = Z_H - Z \leq \varepsilon_G^* + \varepsilon_G'^* \qquad (3.40)$$

Now let us look at the following two extreme cases of ε^*.

1. If all the datasets are grouped into a single group, then $w(d,dl)=0$ for all $(d,dl) \in D$, $D'_c(d,dl)$ is also zero for all $(d,dl) \in D$ and consequently $\varepsilon_G'^* = 0$. In other words, if all the datasets are grouped into a single group, then no error is introduced by HD.
2. If no groups are formed, i.e. $w(d,dl)=1$ for all $(d,dl) \in D$, then $\varepsilon_G^* = 0$ and $\varepsilon_G'^* = \varepsilon_D^*$.

Based on these observations, the worst-case deviation of H is either ε_G^* or ε_D^*. Thus:

$$Z_H - Z \leq MAX(\varepsilon_G^* , \varepsilon_D^*) \qquad (Q.E.D)$$

The result that the worst-case deviation of H is larger of the worst-case deviations from HD and HG is significant since it shows that the two schemes are interdependent. This is intuitively obvious since the grouping scheme tends to cluster the consistency related data, thus reducing the number of consistency relationships which in turn decreases the effect of weak-consistency approximation. (If there are no consistency relationships, then the error introduced by the weak-consistency approximation is zero.)

3.2 Probabilistic and Average Analysis

A major problem with the worst-case analysis is that it does not provide any insight into the average performance. Some algorithms are extremely poor in terms of worst-case performance but do quite well in practice due to the small probability of worst-case occurrence. For ex-

ample, the analysis performed above show the bounds on approximations but do not show us the frequency of worst-case occurrences. Probabilistic analysis provide a mechanism for the study of the probabilistic properties of algorithms based on assumed density function for the input data <KARP76f, KARP77f>.

The principles of probabilistic analysis are simple. Let U denote the set of problem instances, $I \in U$ a particular problem instance, $Z(I)$ the optimal value of problem I and $Z_H(I)$ the value obtained when algorithm (heuristic) H is applied to the problem. It is assumed that algorithm H is well specified to uniquely define $Z_H(I)$. In terms of the unconstrained allocation problem, I is defined by an instance of an application system and a DDP support system. We use ε to denote the deviation of $Z_H(I)$ from $Z(I)$, i.e:

$$\varepsilon = Z_H(I) - Z(I)$$

This deviation can assume values between 0 to ε^*, where ε^* is the maximum deviation achieved through worst-case analysis. Since H is deterministic, several instances I can generate the same ε. For example, if $\varepsilon = 0$, then two instances $(I1,I2) \in U$ may produce $Z_H(I1) = Z(I1)$ and $Z_H(I2) = Z(I2)$. Let $\varepsilon = n$ denote that ε assumes a value n, where $n \geq 0$, then let I(n) show the set of instances which produce $\varepsilon=n$ and let $Pr(\varepsilon=n)$ denote the probability that a given experiment produces $\varepsilon=n$, i.e. chooses an instance from I(n). Assuming t deviations (n1,n2,,,, nt), the expected deviation $EX(\varepsilon)$ is:

$$EX(\varepsilon) = \Sigma_n \, Pr(\varepsilon=n) \times n \qquad (3.50)$$

This equation gives a measure of average performance of H. The procedure used for this calculation is:

1. Choose a set of deviations to be analyzed. For the purpose of analysis, we choose three deviations: 0, $\varepsilon^*/2$ and ε^*, where ε^* is the maximum deviation given by the worst-case analysis.

2. Calculate, or estimate, the probabilities $Pr(\varepsilon=n)$. In most cases this estimation is difficult since it requires knowledge of the envi-

ronment in which the algorithm is used. We will assume some values for the purpose of analysis.

Now we perform the probabilistic analysis of the allocation algorithm described in section 2.7. Let ε_G denote the deviation due to the grouping scheme and let ε_D denote the deviation due to the data allocation scheme of the algorithm H. Then the average performance of algorithm H described in section 2.7 is given by $EX(\varepsilon) = EX(\varepsilon_D) + EX(\varepsilon_G)$ where EX(y) shows the expected value of variable y. The expected values of ε_G and ε_D are studied next.

i). EXPECTED VALUE OF GROUPING DEVIATION ε_G

From the discussion on worst-case analysis of grouping approximation, we get:

$$\varepsilon_G = SD + LD + RD \qquad (3.51)$$

where SD, LD and RD represnt deviations due to storage, local processing and remote processing , respectively, and are given by equations 3.29, 3.30 and 3.31. The expected value of ε_G is:

$$EX(\varepsilon_G) = \Sigma_n \Pr(\varepsilon_G=n) \times n$$

Where $\Pr(\varepsilon_G=n)$ is the probability that, for a given experiment, $\varepsilon_G = n$. For simplicity, we concentrate on three distinct vlaues of ε_G: 0, $\varepsilon_G^*/2$ and ε_G^*, where ε_G^* is the maximum deviation due to grouping and was derived using worst-case analysis (see equation 3.20). We need to calculate or estimate $\Pr(\varepsilon_G=n)$. Let us consider the situation when we can assume that the three values occur with equal probability, i.e. $\Pr(\varepsilon_G=n) = 1/3$ for n=1,2,3. Then $EX(\varepsilon_G)=\varepsilon_G^*/2$, i.e. the deviation is half the deviation produced by the worst-case analysis. For more general situations, we need to make the following observations:

1).The deviation introduced due to grouping depends upon the number of groups formed.If no groups are formed, then $\varepsilon_G=0$ for all instances I Ξ U. Two factors affect the number of groups formed: the proximity relationships and the grouping thresholds. The proximity relationships

exist due to sharing of programs and data in the application system and the grouping scheme clusters data and programs being shared. Thus $\varepsilon_G=0$ if an application system has no or little sharing of data and programs. A high grouping threshold, discussed in the grouping scheme above, reduces the number of groups formed. Fortunately, for the grouping scheme, the amount of grouping can be easily controlled externally by adjusting the value of the grouping threshold TH. The program described above which implements this scheme supports the external control and thus provides a flexible approach, where an analyst can choose, through a parameter, the optimization speed versus accuracy.

2). ε_G needs to be calculated for each group formed. Thus for each group we need to calculate ε_G = SD + LD + RD where SD, LD and RD are the grouping deviations due to storage, local processing and remote processing activities. SD, LD and RD are given by equations 3.29 through 3.31.

3). The situations that lead to $\varepsilon_G=0$: From the discussion on worst-case analysis of the grouping approximation, SD=LD=RD=0 if $\alpha^*(d,i) = \alpha''^*(g,i)$ for all g, d, i. Physically, this means that if the grouping error is zero if the optimal dataset allocation is equal to the optimal allocation of dataset groups generated by the grouping scheme. Another situation that yields $\varepsilon_G=0$ occurs when the network is homogeneous from cost point of view, i.e. all the nodes have same storage, local and remote processing costs. To illustrate this let us consider two datasets, d1 and d2, that are to be allocated to a two node network (nodes n1 and n2). Assume that a group g =(d1,d2) is formed and allocated to node n1 to generate Z_{HG} but d1 is allocated to n1 and d2 is allocated to n2 to generate Z. Then:

$\alpha''^*(1,1)=1$, $\alpha''^*(1,2)=0$

$\alpha^*(1,1)=1$, $\alpha^*(1,2)=0$, $\alpha^*(2,1)=0$, $\alpha^*(2,2)=1$

where $\alpha''^*(g,i)$ shows the optimal allocation of group g to node i and $\alpha^*(d,i)$ shows the optimal allocation of dataset d to node i. To compute

SD, LD and RD, let $L_t(d) = L_r(d) + L_w(d)$ and $R_t(d) = R_r(d) + R_w(d)$. Substituting the values of α^* and α''^* into equations 3.29, 3.30 and 3.31 and simplifying, we get:

$$SD = (S_a(d1)+S_a(d2))C_S(1) - S_a(d1)C_S(1) - S_a(d2)C_S(2)$$
$$= S_a(d2)\,[C_S(1)-C_S(2)]$$

$$LD = (L_t(d1) + L_t(d2))C_L(1) - L_t(d1)C_L(1) - L_t(d2)C_L(2)$$
$$= L_t(d2)\,[C_L(1) - C_L(2)]$$

$$RD = [R_t(d1)T_a(2)C_R(2,1) + R_t(d2)T_a(2)C_R(2,1)]$$
$$- R_t(d1)T_a(2)C_R(2,1) - R_t(d2)T_a(1)C_R(1,2)$$
$$= R_t(d2)\,[T_a(2)C_R(2,1) - T_a(1)C_R(1,2)]$$

The equations for SD, LD and RD show that SD=LD=RD=0 if the storage, local processing and remote processing costs (C_S, C_L, C_R) are same for all (i,j) Ξ N. In several practical situations, the costs are the same or differ slightly among systems. Thus according to these results the grouping algorithm will produce exact results even if the datasets and groups are allocated to different nodes.

To summarize, $Pr(\varepsilon_G=0)$, the probability that the grouping algorithm will produce zero deviation (exact result) is a function of proximity relationships, the grouping threshold and the costs of the network resources. Thus on the average, the grouping scheme produces exact results for high grouping thresholds, homogeneous networks and weakly interacting program and data application systems.

ii). EXPECTED VALUE OF WEAK-CONSISTENCY DEVIATION ε_D

From the discussion on the worst-case analysis of the algorithm H, we recall that the deviation ε_D is dependent on the grouping w(d,d1) formed by the grouping scheme HD and is given by:

$$\varepsilon_D = \Sigma_g\, \lambda'''(g) - \min \Sigma_d\, \lambda(d) \qquad (3.52)$$

where $\lambda(d)$ is the consistency update cost given by equation 3.2 and $\lambda'''(g)$ is the consistency update cost due to the allocation of group g (equation 3.36). The expected value of ε_D is given by:

$$EX(\varepsilon_D) = \Sigma_n \Pr(\varepsilon_D=n) \times n$$

where $\Pr(\varepsilon_D=n)$ is the probability that, for a given experiment, $\varepsilon_D = n$. Once again, we concentrate on three distinct vlaues of ε_D: 0, $\varepsilon_D^*/2$ and ε_D^* where ε_D^* is the maximum deviation due to weak-consistency that was derived using worst-case analysis (see equation 3.1). We need to calculate or estimate $\Pr(\varepsilon_D=n)$.

The probability that $\varepsilon_D = n$ is a function of the deviation ε_G from HG. Thus we need to express $\Pr(\varepsilon_D=n)$ in terms of conditional probabilities. Using the standard conditional probability multiplication formula:

$$\Pr(\varepsilon_D=n) = \Sigma_m \Pr(\varepsilon_G=m) \Pr(\varepsilon_D=n|\varepsilon_G=m)$$

where $\Pr(\varepsilon_D=n|\varepsilon_G=m)$ shows the conditional probability of $\varepsilon_D=n$ given $\varepsilon_G=m$ and $\Pr(\varepsilon_G=m)$. We need to calculate, or estimate, $\Pr(\varepsilon_D=n|\varepsilon_G=m)$. Let us reconsider the situation when ε_G assumes the three values equal probability, i.e. $\Pr(\varepsilon_G=n) = 1/3$ for n=1,2,3. Then:

$$\Pr(\varepsilon_D=0) = 1/3[\Pr(\varepsilon_D=0|\varepsilon_G=0)+\Pr(\varepsilon_D=0|\varepsilon_G=\varepsilon_G^*/2)+\Pr(\varepsilon_D=0|\varepsilon_G=\varepsilon_G^*)]$$

But, according to the worst-case analysis of H, $\Pr(\varepsilon_D=0|\varepsilon_G=\varepsilon_G^*) = 1$, and all other conditional probabilities are zero. Thus $\Pr(\varepsilon_D=0)=1/3$. Similarly $\Pr(\varepsilon_D=\varepsilon_D^*) = 1/3$ and $\Pr(\varepsilon_D=\varepsilon_D^*/2) = 1/3$. This shows that if ε_G has a uniform distribution, so does ε_D. Then $EX(\varepsilon_D)=\varepsilon_D^*/2$, i.e. the expected deviation is half the deviation produced by the worst-case analysis. For more general situations, we need to make the following observations about $\Pr(\varepsilon_D=n|\varepsilon_G=m)$:

1). $\Pr(\varepsilon_D=0|\varepsilon_G=m)$ is 1 if no consistency updates are applied or if there are no consistency relationsips. This is clear from an examination of equation 3.39 which shows that $\varepsilon_D=0$ if $P_c(p,d) = 0$ for all $p \in P$ and all $d \in D$ and/or if $D'_c(d,d1) = 0$ for all $d,d1 \in D$. Recall that $D'_c(d,d1) = D_c(d,d1)\ w(d,d1)$ where $D_c(d,d1)$ is the consistency relationship defined in table 2.1 and $w(d,d1) = 0$ if datasets d and d1 are grouped together. Thus $D'_c(d,d1)=0$ if $w(d,d1)=0$, i.e. if datasets are

grouped. In other words, the higher the grouping, the lower is the error introduced due to weak-consistency approximation.

2). $\varepsilon_D = 0$ or very small if optimal data allocations are unique

Equation 3.38 shows that at any given time:

$$\varepsilon_D = \Sigma_g \lambda'''(g) + \min \Sigma_d \lambda(d)$$

Since λ''' and λ are monotonically increasing functions of data assignment, the minimum of λ occurs when datasets are uniquely assigned. If the algorithm H generates unique optimal allocation, i.e. $\Sigma_i \alpha(d,i) = 1$ for all $d \in D$ then $\Sigma_g \lambda'''(g) = \min \Sigma_d \lambda(d)$ and consequently $\varepsilon_D = 0$. This is an important result since we wish to show that the generalized allocation algorithm does indeed generate unique allocations for most practical cases. The unconstrained model only generates unique optimal data allocations when the transaction update/read ratio is high (greater than 50%). We will treat this subject in detail later.

To summarize, the probability $Pr(\varepsilon_D=0 | \varepsilon=m)$ depends on the number of consistency relationships and updates and also on the update/read ratio of the transactions.

3.3 Computational Experience and Empirical Analysis

To illustrate the computational aspects of the algorithm, consider the allocation of the two program/three dataset application system, to a 5 node network. The input (an instance I discussed above), has been shown in figure 3.1 in terms of the application parameters described in table 2.1. Figure 3.2 shows the output of the allocation algorithm. Through explicit enumeration, the problem would require 2^{25} evaluations and about 45 hours of cpu time. The allocation algorithm finds the optimal value within 67 evaluations in 3 seconds on an IBM 3033 system.

Several observations can be made about the results of this example problem. First, the allocation algorithm did not cluster the consistency related datasets d2 and d3 together mainly because the consistency

FIG(3.2) THE ALGORITHM OUTPUT
(ALLOCATION OF 2 PROGRAMS, 3 DATASETS TO 5 NODES)

STORAGE COST	LOCAL PROC. COST	REMOTE PROC. COST	TOTAL COST	DATA ALLOCATION D1 * D2 * D3 123451234512345	DATASET NODES
*STAGE 1 .. ALLOCATION OF D1					
600	1560	3000	5160	100000000000000	
1200	1560	2400	5160	110000000000000	
1800	1560	1500	4860	111000000000000	
2400	1560	300	4260	111100000000000	
3000	1560	0	4560	111110000000000	
1200	1560	2100	4860	101000000000000	
1800	1560	900	4260*	101100000000000	
2400	1560	600	4560	101110000000000	
1200	1560	1800	4560	100100000000000	
1800	1560	1500	4860	100110000000000	
1200	1560	2700	5460	100010000000000	
600	1560	3300	5460	010000000000000	
1200	1560	2400	5160	011000000000000	
1800	1560	1200	4560	011100000000000	
2400	1560	900	4860	011110000000000	
1200	1560	2100	4860	010100000000000	
1800	1560	1800	5160	010110000000000	
1200	1560	3000	5760	010010000000000	
600	1560	3000	5160	001000000000000	
1200	1560	1800	4560	001100000000000	
1800	1560	1500	4860	001110000000000	
1200	1560	2700	5460	001010000000000	
600	1560	2700	4860	000100000000000	
1200	1560	2400	5160	000110000000000	
1800	1560	1200	4560	110100000000000	
600	1560	3600	5760	000010000000000	
*STAGE 2 .. ALLOCATION OF D3					
2400	1664	1100	5164	101100000010000	
3000	1768	1320	6088	101100000011000	
3000	1768	1300	6068	101100000010100	
3000	1768	1280	6048	101100000010010	
3000	1768	1340	6108	101100000010001	
2400	1664	1120	5184	101100000001000	
3000	1768	1320	6088	101100000001100	
3000	1768	1300	6068	101100000001010	
3000	1768	1360	6128	101100000001001	
2400	1664	1100	5164	101100000000100	
3000	1768	1280	6048	101100000000110	
3000	1768	1340	6108	101100000000101	
2400	1664	1080	5144*	101100000000010	
3000	1768	1320	6088	101100000000011	
2400	1664	1140	5204	101100000000001	
*STAGE 3 .. ALLOCATION OF D2					
3000	3848	5080	11928	101101000000010	
3600	3952	4280	11832	101101100000010	
4200	4056	3080	11336	101101110000010	
4800	4160	1480	10440*	101101111000010	
5400	4264	1080	10744	101101111100010	
3600	3952	3880	11432	101101010000010	
4200	4056	2280	10536	101101011000010	
4800	4160	1880	10840	101101011100010	
3600	3952	3480	11032	101101001000010	
4200	4056	3080	11336	101101001100010	
3600	3952	4680	12232	101101000100010	
3000	3848	5480	12328	101100100000010	
3600	3952	4280	11832	101100110000010	
4200	4056	2680	10936	101100111000010	
4800	4160	2280	11240	101100111100010	
3600	3952	3880	11432	101100101000010	
4200	4056	3480	11736	101100101100010	
3600	3952	5080	12632	101100100100010	
3000	3848	5080	11928	101100010000010	
3600	3952	3480	11032	101100011000010	
4200	4056	3080	11336	101100011100010	
3600	3952	4680	12232	101100010100010	
3000	3848	4680	11528	101100001000010	
3600	3952	4280	11832	101100001100010	
4200	4056	2680	10936	101101101000010	
3000	3848	5880	12728	101100000100010	

NOTES:
1. THE DATA ALLOCATION IS REPRESENTED BY A BINARY VECTOR WHICH SHOWS A 1 TO INDICATE ALLOCATION. FOR EXAMPLE, THE VECTOR 11000 FOR D1 INDICATES D1 IS ONLY ALLOCATED TO NODES 1 AND 2.THE ENTIRE ROW OF 101100000101010 SHOWS THAT D1 IS ALLOCATED TO NODES 1,3,4 (10110), D2 IS ALLOCATED TO NODE 5 (00001) AND D3 IS ALLOCATED TO NODES 2 AND 4 (01010).
2. THE UNITS FOR COST ARE ARBITRARY
3. TOTAL COST = STORAGE COST + LOCAL PROCESSING COST + REMOTE PROCESSING COST.

OPTIMIZATION RESULTS:
- THE OPTIMAL TOTAL COST IS 10440.
- THE OPTIMAL PROGRAM ALLOCATION IS:
  - ALLOCATE P1 TO ALL NODES
  - ALLOCATE P2 TO ALL NODES
- THE OPTIMAL DATA ALLOCATION IS
  - ALLOCATE D1 TO NODES 1,3,4.
  - ALLOCATE D2 TO NODES 1,2,3,4.
  - ALLOCATE D3 TO NODE 4.

updates were much smaller than the dataset reads (see parameters P_c, P_w, P_r in figure 3.1). Due to this, no dataset groups were formed and thus no error was introduced due to the grouping approximation. Second, the algorithm sequences datasets d1, d2 and d3 so that datasets are allocated in the order: d1, d3, d2. This sequencing makes the problem decomposable (d3 and d2 are related) by rearranging dataset allocations so that consistency related data is allocated at later stages in the algorithm. Third, the allocation algorithm follows a hierarchical heuristic in which d1 is allocated in stage 1, d3 is allocated in stage 2 subject to the stage 1 optimal allocation and d2 is allocated in stage 3 subject to the optimal allocations of stages 1 and 2. The allocation algorithm yields, in the final (third) stage, the exact global optimal cost of 10440, in arbitrary cost units, and the global optimal allocation for the datasets d1, d2, d3 (see figure 3.2). This truly optimal result is due to the non-existence of grouping and due to the success of the sequencing routine. Empirically, the global optimal value was verified through a separate experiment which explicitly evaluated about 30,000 data allocations (the two programs were replicated).

To study the effect of grouping, dataset grouping was "forced" by lowering the grouping threshold. As a result two dataset groups g1,g2 were formed: g1=d1, g2=(d2,d3). This grouping reduced the number of evaluations to 40 and took about 3 seconds to reach the optimal cost of 11528, which is not the exact global optimal but within 10% error. The solution of same problem for a 10 node network took 1.3 minutes and for a 15 node network took 5.2 minutes.

Additional experimentation has shown that for most practical situations, the sequencing is 100% successful; thus the data allocation problem is almost always decomposable and the hierarchical heuristics provide exact or close to exact results. The grouping scheme reduces the optimization time but introduces minor deviations from optimal return. However, these errors are usually compensated by the data allo-

cation scheme. Most deviations have been found to be in the 10 to 15% range. A detailed account of computational experience with the generalized allocation algorithm will be given later.

We need to emphasize that the optimal solution is found only if the stated assumptions, in addition to the grouping and weak-consistency approximations analyzed above, are satisfied. As shown in the worst-case and average analysis, the optimization results are dependent upon the input parameters, i.e. the systems being considered.

3.4 Summary of Results

The following major results have been established in this chapter:

1). WORST-CASE ANALYSIS: It has been shown that the worst-case deviation ε^* of the allocation algorithm presented in section 2.7 is given by:

$$\varepsilon^* = \text{MAX}\ (\varepsilon_D^*\ ,\ \varepsilon_G^*)$$

where ε_G^* represents the worst-case deviation due to the grouping approximation and ε_D^* is the worst-case deviation due to the weak-consistency approximation. ε_G^* shows that the worst-case performance is a function of the following input parameters: the number of nodes in the network, the resource utilization costs for storage, local processing & remote processing, the file sizes and program to data activity. ε_D^* is a function of the following input parameters: the number of nodes in the network, the consistency relationships, the consistency updates and transaction frequency (arrival rate).

2). PROBABILISTIC AND AVERAGE ANALYSIS: The expected deviation (error) of the allocation algorithm is given by:

$$EX(\varepsilon_G) + EX(\varepsilon_D)$$

where $EX(\varepsilon_G)$ shows the expected deviation due to grouping approximation and $EX(\varepsilon_D)$ is the expected deviation due to the weak-consistency approximation. $EX(\varepsilon_G)= 0$ if no groups are formed due to lack of interactions

between data and programs, high grouping threshold (an externally controllable parameter) and/or cost-homogeneous networks. $EX(\varepsilon_D) = 0$ if the application system does not contain consistency related data/updates and also if the update/read ratio is large (greater than 50 percent). An important result of this analysis is that the error ε_G is compensated by ε_D due to the reduction of consistency relationships by the amount of grouping. (More grouping increases ε_G but reduces consistency relationships which reduces ε_D because ε_D is a monotonically increasing function of consistency relationships.) For example, when all datasets are clustered into one group then the error ε_G is maximum but $\varepsilon_D = 0$.

3). EMPIRICAL ANALYSIS: A three dataset, two program application system is allocated to a 5 node network by using the unconstrained algorithm. This experiment yields exact optimal results because datasets are not clustered and the sequencing routine orders the datasets to produce exact decomposition.

The main limitation of the unconstrained algorithm is that the support system factors are ignored through numerous explicit simplifying assumptions. In the next chapters, we will introduce the effect of support system considerations by relaxing these assumptions.

CHAPTER 4

TRANSACTION PROCESSING AND UPDATE SYNCHRONIZATION EXTENSION

The unconstrained model presented in chapter 2 can be significantly extended by relaxing the following data distribution, transaction processing and update synchronization assumptions stated in section 2.4: a) there were no restrictions on data distribution, i.e. the data could be allocated uniqely, replicated or factored/partitioned, b) the transaction processing allowed direct access of remotely located data based on minimum cost, i.e. the data was accessed from a node to minimize communicaion cost, and c) the duplicate data was updated without employing any update synchronization algorithm. Removal of these assumptions introduces several key data distribution, access and update issues like the routing graphs, site selection, query decomposition, query optimization, centralization/decentralization of lock management and update synchronization mechanisms. This chapter focusses on the analysis, representation and measurement of the effect on program/data allocations of these factors.

4.1: The Data Distribution Strategies

In a DDP system consisting of N_n nodes, a dataset d can be allocated in one of the following manners: a) centralized, or unique, where d is allocated at one node, b) partially replicated, where d is allocated to n nodes where $n < N_n$, and c) fully replicated, where d is allocated to all the nodes, i.e. N_n copies of d. These strategies can be generalized to study entire database allocations. For example, consisder a database Y consisting of datasets (d1,d2,,,dm). Y can be centralized, partially replicated or fully replicated. However, a "factored" distribution can be achieved if portions of Y are allocated to different nodes (d1 is al-

located to node 1 and d2, d3 are allocated to, say, node 2). Fortunately, the application system model described in chapter 2 allows factoring since the units of allocation are the datasets and not databases. Thus, given the datasets (d1,d2,,,,dm) of Y, the allocation algorithm can centralize, replicate or factor Y. Consequently we need to concentrate on dataset allocation only and will not discuss database allocation explicitly.

Each distribution strategy can be evaluated in terms of storage costs, response time, availability and concomittant support system overhead (Teorey and Fry <TEOR82e>). In the unconstrained model it was assumed that there are no restrictions on data distribution, thus data could be centralized or replicated to minimize the total cost. However, several practical support systems restrict distribution of data in some manner. For example, the IDER system reported by Lien <LIEN78b> is restricted to unique data allocation, the system reported by Aschim <ASCH78a> allows only unique or fully replicated data assignments and SDD1 <ROTH77b> allows unique, fully replicated and partially replicated data distributions. The reason for these restrictions is that management of duplicate data introduces some difficult support system problems in the areas of data access and maintenance.

Due to the removal of the data distribution assumption, we need to introduce the following COPY parameter, a scalar, to include the data distribution restrictions of DDP support systems:

COPY = maximum number of copies supported in the network

where $1 \geq COPY \leq N_n$, and N_n is the no. of nodes. If COPY = 1, then the data can only be allocated uniquely (only one copy is allowed). The data distribution restriction can be easily introduced in the program/data allocation problem by generating the following constraint which restricts the data allocation decision variable α:

$$\Sigma_i \, \alpha(d,i) \leq COPY \qquad \text{for all } d$$

This constraint implies that if COPY = N_n, then data can be assigned uniquely, partially replicated or fully replicated.

4.2 Transaction Processing in DDP

A transaction t produces result r by performing operations of programs p Ξ P_t on datasets d Ξ D_t, where P_t is the set of programs activated by transation t and D_t are the set of datasets accessed by the transaction t. Before proceeding, it is important to distinguish between an operation and a program. An operation is essentially the access and/or manipulation of data and can be represented by relational operators like selection, projection and join. A program consists of one or more operations and, for the purpose of this research, is the basic unit of allocation. Example of a program is a COBOL module with, say, SEQUEL or IMS calls, where each IMS call or SEQUEL statement can be viewed as an operation.

Transaction processing refers to the execution of a transaction in a support system and is generally referred to in literature as query processing. Transactions may involve data retrieval as well as modification. A given transaction can exist in one of the following forms: a) single dataset, single program b) single dataset, multiple programs c) multiple datasets, single program and d) multiple dataset, multiple programs. In addition, the operation(s) performed by a transaction can be read-only, write-only or read-write. We consider general read-write transactions, unless otherwise specified, since the other two type of transactions are special cases of the read-write transactions.

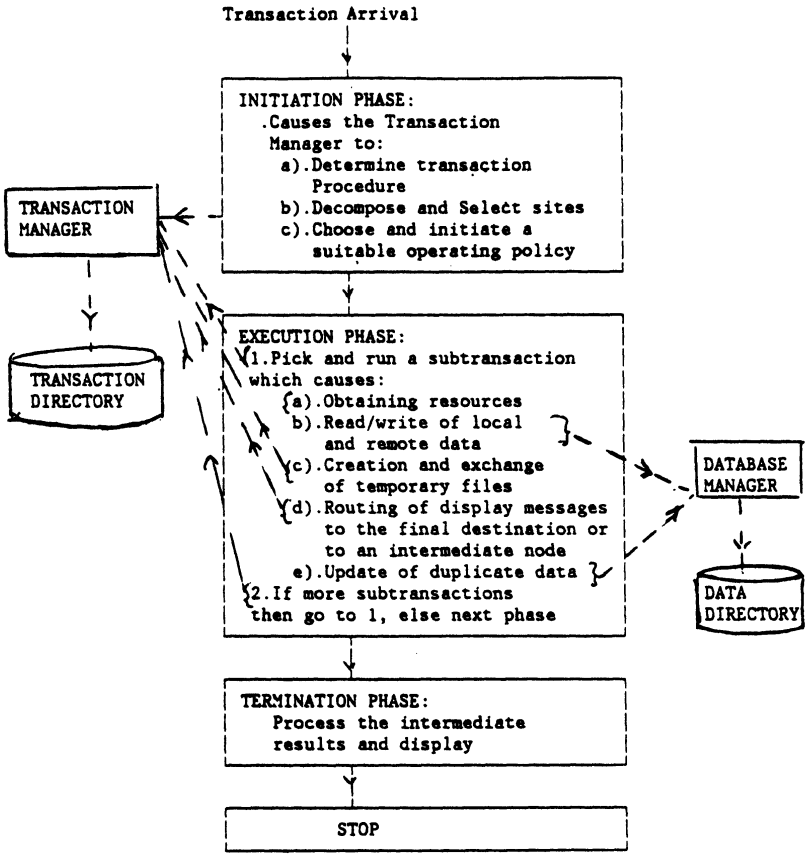
In a centralized system, all the datasets and operations are restricted to one node. The major issue in these systems is to sequence the operations in such a manner that a given objective function (processing cost or response time) is minimized <SACC81d>. The task of processing transactions in DDP is known as *distributed transaction*

processing and is complicated due to the existence of duplicate data/programs, the possibility of parallel processing and the large number of routing possibilities. The problem of determining an optimal distributed transaction processing algorithm which minimizes a given objective function was first addressed by Wong <WONG77d>. Since then several algorithms have been proposed and implemented (see <HEVN81d> and <APER83d> for an overview). A brief analysis of the various transaction processing models is given in appendix D. It is not our objective to propose yet another distributed transaction processing algorithm. Instead, we are interested in the formal representation of available algorithms so that the effect of these algorithms on program and data allocations can be measured. To this end, we introduce a generalized transaction processing model, define the parameters which represent transaction processing in DDP and show how these parameters can be used to represent available transaction processing algorithms.

4.2.1 A Model of Distributed Transaction Processing

Figure 4.1 shows a generalized distributed transaction model which represents flow of read/write transactions in DDP. The arriving transaction is analyzed, and a proper algorithm (policy) is chosen, in the inititiation phase. In the execution phase, the subtransactions are routed to various nodes where the subtransactions are executed and the final results are processed and displayed at the resulting node in the termination phase. We first describe the initiation phase activities since these activities help us parameterize distributed transaction processing. The other phases basically carry out the decisions made in the transaction initialization phase and will be discussed when the total cost is calculated. The control subsystem components (transaction manager, transaction directory, data manager, data directory) are also defined.

Figure 4.1: Model of Distributed Transaction Processing



i). TRANSACTION PROCEDURE: Let P_t designate the set of programs activated by transaction t and let D_t show the set of datasets manipulated by t. Then the transaction procedure of a given transaction t, which produces result r, shows: a) the programs $p \Xi P_t$ activated by transaction t, b) the datasets $d \Xi D_t$ manipulated by $p \Xi P_t$ and c) the precedence relationships between $p \Xi P_t$. A transaction procedure which shows the programs $p \Xi P_t$ and the sequence between all $p \Xi P_t$ as shown by the precedence relationship is termed a "basic" transaction procedure. As can be expected, several "equivalent" transaction procedures can be created for a given basic transaction procedure. Figure 4.2a illustrates this point by considering a transaction "TRAN" which activates four programs (p1, p2, p3, p4) ΞP_{TRAN} on dataset $d \Xi D_{TRAN}$, where p1 generates a temporary file f1, p2 and p3 receive f1 and operate on d to

generate files f2 and f3, and p4 recevies f2, f3 and generates result r.

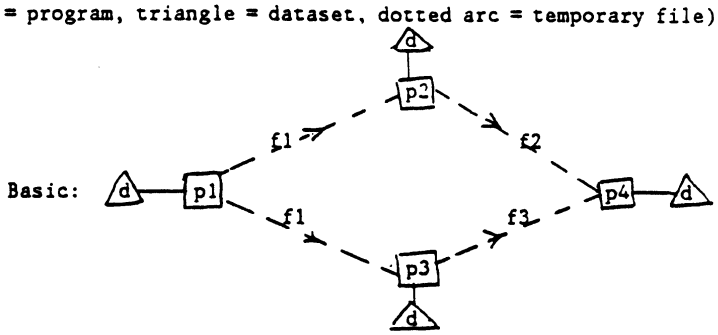
ii). SITE SELECTION: Site selection consists of: a) grouping of the programs to form subtransactions, where each subtransaction can execute at one node and b) determination of the sites where the programs and the datasets will be accessed. Formation of subtransactions is generally referred to as "query decomposition" and will be addressed in this research by using the grouping scheme which clusters programs based on proximity relationships. Thus each program group generated by the grouping scheme can be treated as a subtransaction because all the programs in a program group are allocated together (execute at one node). The subject of site selection is of vital importance in distributed transaction processing and is discussed as *materialization* by several researchers <HEVN81d>.

iii). OPERATING POLICIES AND COST ESTIMATION/MINIMIZATION: An operating policy of a given transaction shows the programs activated by the transaction and the flow (sequence and the sites) of the programs in the network. It thus shows the transaction procedure plus the sites where the operations are to be performed. In case of unique program/data allocations, there is one operating policy per transaction procedure since the site selection is straightforward (the node where a dataset or a program resides is the site selected). However, for duplicate programs and data, a given transaction procedure can generate several operating policies because different sites can be selected based on the "site selection strategies" employed by the support system. In order to illustrate this point, let us consider a 5 node network with the following program and dataset allocations: a) the programs p1, p2 and p3 are assigned to nodes 1, 2 and 3 respectively and p4 is assigned to nodes 4 and 5 (duplicate assignment) and b) the dataset d is assigned to nodes 1 and 5. Let us assume that the transaction TRAN, illustrated in figure 4.2a, arrives at node 1; then node 1 is selected for p1, node 2

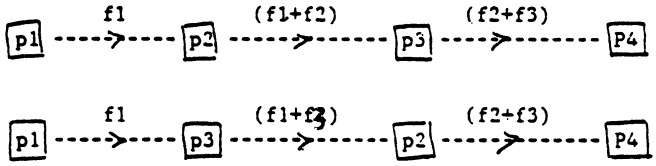
Figure (4.2): TRANSACTION PROCEDURES AND OPERATING POLICIES

A). TRANSACTION PROCEDURE FOR "TRAN":

(Box = program, triangle = dataset, dotted arc = temporary file)

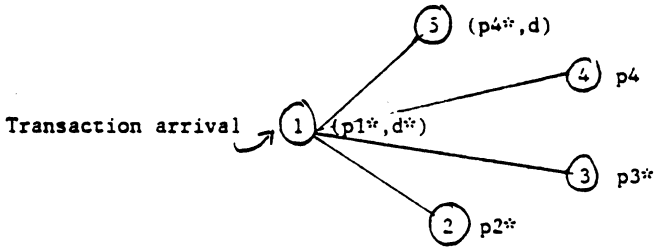


Equivalent transaction procedures (dataset d not shown for simplicity):



B). SITE SELECTION FOR "TRAN":

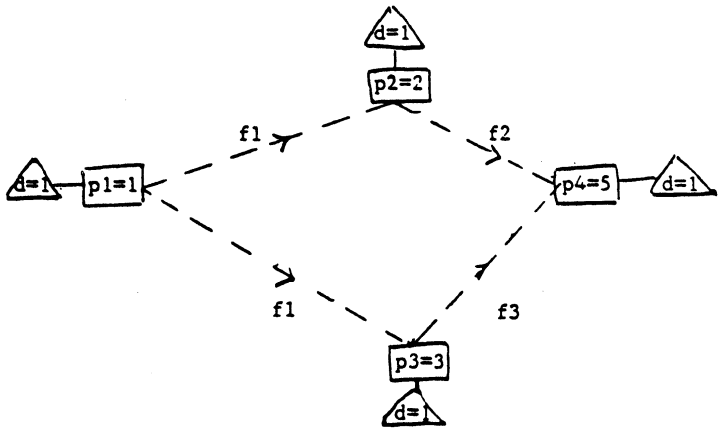
(* indicates the progam/dataset selected at that node)



NOTE: The site selection is based on minimum distance and the diagram above shows the nodes in scale of distance.

C) OPERATING POLICY FOR "TRAN":

(pn=i indicates that program pn is selected at node i)



NOTE: This operating policy corresponds to the basic transaction procedure, the program/data allocations shown above and the arrival of "TRAN" at node 1.

is selected for p2, node 3 is selected for p3 and node 4 or node 5 is selected for p4 depending on minimum cost or minimum distance site selection strategy being employed. However, the dataset d is selected at node 1 since the transaction arrives at node 1 (see figure 4.2b). This implies that the programs p1, p2, p3 and p4 will always access d at the arriving node, independent of the sites selected for programs, as long as the data site is selected based on minimum cost from the arrival node. For each program and data site selected, a new operating policy can be generated. For example, figure 4.2c shows the operating policy when program p4 is selected at node 5 and the dataset d is selected at node 1. If the program p4 is selected at node 4 instead of node 5, then a different operating policy is generated. Similarly if the the dataset d is selected at node 5, then still another operating policy is generated.

iv). IMPLEMENTATION OF DISTRIBUTED TRANSACTION PROCESSING: Transaction processing is implemented through a transaction manager control program and a transaction control database. The *Transaction Manager* is responsible for i) determining the transaction procedure, site selection, determination of optimal operating policy and controlling the flow of transactions in the system. The *Transaction Directory* contains all the information needed by the transaction manager. This database consists of i) data directory which shows the structure, access rights and the location of the data and ii) program directory which shows the characteristics, data accessed and location of the programs. In addition, a *Data Manager* and a *Data Directory* is used for the retrieval and manipulation of data. We will discuss these two components in section 4.3.

As can be expected, the transaction manager (TM) will read the transaction directory (TD) frequently. The allocation of TM and TD can itself be viewed as a program/data allocation problem. For a centralized transaction manager, the TM and the TD are located at the central site,

while TD and TM are distributed to all the nodes for distributed (decentralized) transaction management. The tradeoff between centralized and distributed transaction management is the cost of updating the TD. It should be noted that TM can be centralized, while TD is distributed and vice versa, but practical applications of such systems have not been reported. The distributed transaction manager is referred to as DTMS.

The TD itself can be subdivided into several physical datasets which can reside at different nodes <CHU77d>. We will consider distributed transaction management with fully replicated and core resident TD due to assumption 3, section 2.4. This assumption can be relaxed later.

4.2.2 The Distributed Transaction Processing Parameters

Three parameters are proposed to represent the site selection (the SRCH parameter), routing graphs (the Q_{rt} parameter) and the data access strategies (Q_{da}). These three parameters can be used to represent distributed transaction processing.

The SRCH parameter defines the site selection strategy used by the transaction manager. The matrix SRCH(k,x) shows the node which will be searched, to choose a program or data site, at "attempt" x, where x = $(1,2,,,,N_n)$, when a transaction arrives at node k . In other words, this matrix shows the order in which the nodes will be searched for site selection, the node at x=1 is searched first, then the node at x=2, and so on. Note that this matrix can be built based on any strategy: minimum cost, minimum distance, minimum access time, minimum spanning tree etc. To illustrate the SRCH matrix, let us consider a network of 3 nodes 1,2,3 ∈ N with the following SRCH matrix:

SRCH =

K \ X	1	2	3
1	1	3	2
2	2	3	1
3	3	1	2

This matrix shows that if a transaction arrives at node 1, then it first searches node 1 (the local storage), then nodes 3 and 2 for program and data site selection. Let us concentrate on data site selection for a transaction which accesses a dataset d. If d is replicated, then the SRCH matrix shown above will cause the arrival node to be the data site selected. For example, if a transaction arrives at node 2, then node 2 is the data site selected. If d is uniquely assigned to, say, node 3, then node 3 is the site selected for dataset d. From an implementation point of view, this matrix (or some form of it), resides on the transaction directory and is used by the transaction manager for site selection.

The Q_{rt} parameter is used to show the transaction flow and routing options implemented by a given support system. This parameter, in conjunction with the SRCH parameter and the application system parameters, can be used to construct a variety of operating policies. Q_{rt} is a 0-1 variable and shows if the *routing graph* is parallel (1) or serial (0). Parallel routing graph means that the arrival node controls the flow of query, i.e. sends / receives direct messages from program sites. This represents Wong's greedy heuristics and the minimum spanning distance algorithm presented by Li<LI81d>. Serial routing graph means that the transaction is circulated through the program sites. This represents the minimum spanning tree.

To illustrate the parallel and serial routing graphs, consider once again a 3 node network. Figure 4.3a shows the parallel and serial routing graphs for a two program (p1 and p2) and two dataset (d1 and d2) transaction. We are assuming that program p1 accesses dataset d1 and the program p2 accesses dataset d2. In addition, we are assuming that

each dataset and accessing program are assigned to the same node. As can be seen, in case of parallel routing graph, the transaction is decomposed into two parallel subtransactions, which are initiated from node 3. Subtransaction t1 is routed to node 1 and the subtransaction t2 is routed to node 2. The results from node 1 and node 2 are routed back to node 3. In case of serial routing, however, the transaction is routed from node 3 to node 1, where p1 accesses d1. Then the transaction is routed to node 2, where p2 accesses d2. A major feature of the serial routing graph is that the transaction first starts with all the inputs ($P_i(p1) + P_i(p2)$), where $P_i(p)$ represents the input of program p. At node 1, $P_i(p1)$ is used to produce $P_o(p1)$, where $P_o(p)$ represents the output generated by program p. Thus, when the transaction is routed from node 1 to node 2 it transfers ($P_o(p1) + P_o(p2)$). Notice that $P_i(p1)$ has been substituted by $P_o(p1)$ (see Figure 4.3a). Thus, a transaction may "grow" or "shrink" depending on the input to output ratio $P_i(p1)/P_o(p1)$. Also, note that parallel routing graph represents the minimum spanning distance query processing strategy, while serial routing graph represents the minimum spanning tree query processing strategy <LI81d>.

The *Q_{da} parameter* is used to show if the data can be accessed from a remote program or must be accessed only from local programs. This parameter can be used to show the support system restrictions or a user requirement which restricts access of sensitive data only by local programs so that proper trailer records can be kept. Q_{da} is a 0-1 parameter which shows if the data access mode is remote (1) or local (0). Remote data access implies that a program at node i can access data from node j, even if $i \neq j$. Local data access implies that program at node i can only access data at node i, i.e. programs and data exist at the same node. Local data access imposes the implicit restriction on the program and data allocation that the data and accessing programs must be allocated to the same nodes. In order to illustrate the data access modes, consider a program p which receives 1 keyboard record, reads 90 records

from dataset d and generates 500 display messages. Assume that d is assigned to node 2 and the request arrives at node 1. The program is assigned to nodes 1 & 2. Then in case of local data access:

. The request arrives at node 1

. The request is routed to node 2
(1 remote message exchanged).

. The program p at node 2 reads data at node 2
(no remote messages generated).

. The display messages are routed back to node 1
(500 remote messages generated).

. Total number of remote messages = 501

In case of remote data access:

. The request arrives at node 1.

. The program p at node 1 issues 90
reads from dataset d at node 2
(180 remote messages genearted - 90 for
requests and 90 for responses).

. No remote display messages between nodes 1 and 2
since the 500 display messages are now
generated at arrival node (local messages).

. Total number of messages = 181.

It can be seen that the traffic generated is significantly different for the local and remote data access. In order to understand the combined effect of routing graphs and data access modes, let us revisit figure 4.3a. In this figure local data access was assumed since program at node i accessed the dataset at node i. Let us now assume a 5 node network when p1 is assigned to node 1, p2 is assigned to node 2, d1 is assigned to node 5 and d2 is assigned to node 4. Figure 4.3b shows the parallel and serial routing graphs of this network for remote data access. Note that the remote calls from nodes 1 and 2 to access the data

located at nodes 5 and 4 would not be allowed in case of local data access.

The Q_{rt} and Q_{da} parameters can be used to perform optimization of distributed transaction processing. If the Q_{rt} and the Q_{da} parameters are specified, then no optimization is performed and the effect of these parameters is measured on the program/data allocations. However, if these two parameters are not specified, then the model shown in figure 4.1 assumes that the parameters can be chosen which minimize the total cost. This transaction optimization can be performed over both the parameters (Q_{rt} and Q_{da}) or the one not specified.

Thus the Q_{rt} and Q_{da} parameters can be used to represent, or optimize between, the following important extreme cases: a) local data access and parallel routing graph, b) local data access and serial routing graph, c) remote data access and parallel routing graph and d) remote data accesse and serial routing graph. It is interesting to note that almost all of the file allocation research implicitly assumes case c, while most of the query optimization algorithms assume cases a and b.

4.2.3 Evaluation of Transaction Processing Parameters

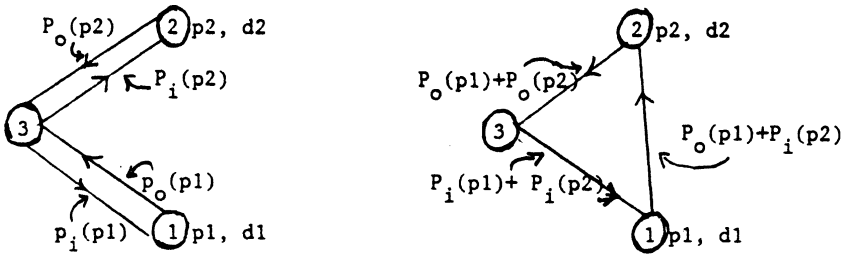
We demonstrate the effectiveness and limitations of the parameters presented above, and summarized in table 4.1, at three levels: i) representation of transaction procedures, ii) representation of operating policies and iii) representation of distributed query optimization algorithms.

i) REPRESENTATION OF TRANSACTION PROCEDURES: In order to include complex transactions in the program and data allocation problem, we must be able to represent the transaction procedures of multiple program/multiple dataset transactions. The approaches to represent transaction procedures range from flow charts to system requirement definition and analysis systems like PSL/PSA <ISDOS77e>. In this re-

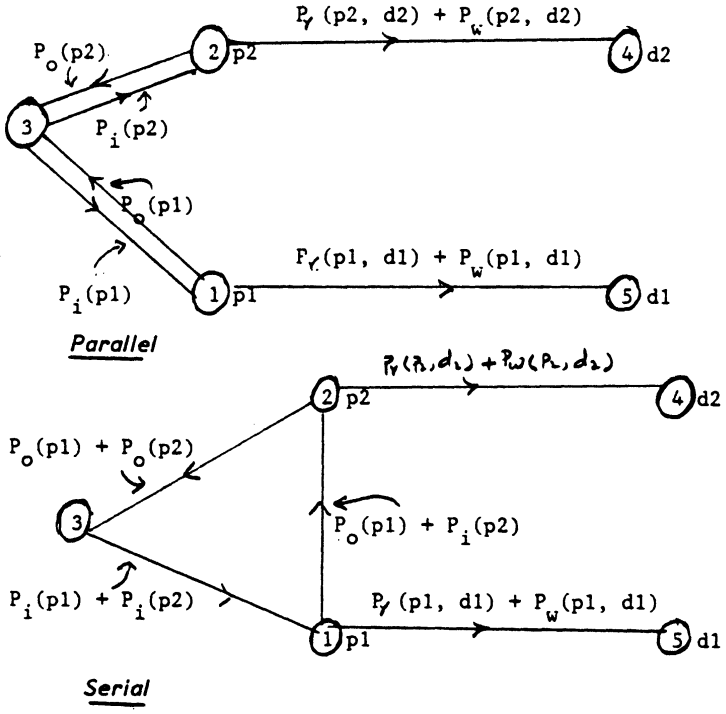
Figure 4.3: ROUTING GRAPHS AND DATA ACCESS MODES

NOTES:
- $P_i(p)$ and $P_o(p)$ show the input and output of program p.
- $P_r(p,d)$ and $P_w(p,d)$ show the number of dataset d tuples read and written by program p, respectively.
- Transaction arrives at node 3.

a). Parallel and Serial Routing Graphs



b). Remote Data Access: Parallel and Serial Routing Graphs:



search, the application system parameters defined in chapter 2, and repeated later in table 4.4 for convenience, will be used to specify the transaction procedures of complex transactions. Our approach consists of: a) each program p Ξ P is viewed as a basic operation (primitive), b) the primitives of each transaction are specified through the application system parameter $T_e(t,p)$ which shows if program p is invoked through a given transaction, c) the datasets manipulated are identified through the program read/write parameters P_r and P_w, and d) the precedence relationships between the various primitives are represented through the application system parameter $P_m(p1,p2)$. These parameters are sufficient to represent multiple program/ multiple dataset transactions which may be read, read/write or write only transactions.

ii) REPRESENTATION OF OPERATING POLICIES: The operating policies of a transaction show the processing options that can exist, in a DDP environment, for a given transaction with a given transaction procedure. All the operating policies of a single program/single dataset transaction can be completely represented by using the Q_{da} parameter since in this case only two options exist: remotely access dataset d from program p or access d locally from p if d and p coexist at the same node. The major issue in this type of transaction is selection of sites for p and d, which can be easily handled by using the SRCH parameter. Consider now multiple program/single dataset transaction which activates programs p1 and p2 to access dataset d. If p1 and p2 are assigned to the same node i, then this situation is the same as the single program/single dataset case and will not be discussed further. Let us designate the transaction arrival nodes as (k1,k2,,) Ξ K, program nodes as (i1,i2,,) Ξ I and dataset nodes as (j1,j2,,) Ξ J. Now consider the situation when p1 is assigned to i1, p2 to i2 where i1 $\neq$ i2, and let d be assigned to node j Ξ J. In this case there are two program sites, and basically two operating policies: i) send the requests directly from k to i1 and i2 and receive the output directly at k and ii) circulate the

TABLE 4.1: THE TRANSACTION PROCESSING PARAMETERS:

SRCH(k,x) = the search strategy for the queries arriving at node k.
x=1 shows the first node searched, x=2 is the second node etc.
Q_{rt} = the routing graph: parallel (1) or ring (0).
Q_{da} = the data access mode: remote access (1) or local access(0)
COPY = the maximum number of dataset copies allowed

transaction through i1 and i2. It can be seen that these two policies can be easily represented by the Q_{rt} parameter. If the number of programs per transaction increases much beyond two programs, then the transaction processing parameters cannot represent all the possible operating policies. In such cases, the transaction can be decomposed into subtransactions For a single program/multiple dataset transaction let us consider a program p which joins two datasets d1 and d2. Then the following operating policies are possible: i) access d1 and d2 remotely, ii) perform a semijoin <BERN79d>, iii) transfer d1 to d2 and perform the join at the d2 site, iv) transfer d2 to d1 site and perform the join at d1 site, and v) transfer d1 and d2 to the resulting node and perform the join there. All these policies can be easily represented by using the application parameters and the Q_{rt} and Q_{da} parameters. However, if the number of datasets are large, say more than 9, then transaction partitioning techniques, discussed below, can be employed to decompose complex transactions into simpler queries. The operating policies of multiple program/multiple dataset transactions can be discussed in terms of the single program/multiple dataset and multiple program/single dataset transactions.

Basically, we can correctly represent ALL the operating policies of two program/two dataset transactions completely. The transactions with upto four program/four dataset transactions can be represented reasonably well by adjusting the SRCH parameter. However, very complex transactions involving ten or more programs and datasets cannot be represented accurately. For such transactions, there are at least three

options: a) use partitioning techniques, like the ones employed by Hevner and Yao <HEVN78d>, in which a complex transaction, involving joins on several attributes, is decomposed into simpler transactions which involve joins on only one attribute, b) define "new" transactions to handle special situations, and c) use the grouping scheme to increase the clustering of datasets and programs by raising the grouping threshold. All these situations can be easily handled by the parameters shown above because we can define partitioned or new transactions by using the application parameters $T_e(t,p)$, $P_r(p,d)$, $P_w(p,d)$, etc and the grouping threshold is specified as an input parameter to the allocation algorithm to be described in chapter 6.

iii). REPRESENTATION OF CURRENT QUERY OPTIMIZATION ALGORITHMS: We show how the existing query optimization algorithms can be represented by the distributed transaction parameters defined above so that the ability of our model to represent real life implemented systems can be demonstrated. The existing distributed query processing algorithms fall into four categories: a) Wong's algorithms <WONG77d> that have been implemented on SDD1, b) Hevner and Yao's algorithms <HEVN78d, HEVN81d, APER83d>, c) the general algorihhm presented by Chu and Hurley <CHU79d> and d) the decomposition agorithms used in INGRES <STONE77d>.

Wong's Algorithm: This algorithm uses a "hill-climbing" strategy and utilizes a general cost function (total time or reponse time) where the results can be produced at any arbitrary node. The algorithm first starts with an initial feasible solution (IFS) which moves all the data to the resulting node. The IFS is replaced with a strategy where the data is moved in two moves, with intermediate local processing. This two move policy is further replaced with a three move strategy etc until there is no cost decrease. This algorithm does not guarentee global optima and effectively starts with a parallel routing graph and eventually leads to a serial routing graph. Thus, the IFS and the final strategy can be represented by using the following parameters: Q_{rt} for parallel

and serial routing graphs, Q_{da} for local data access and SRCH parameter to reflect the site selection based on cost function.

Hevner & Yao Algorithms: Three algorithms have been proposed by these researchers. Serial algorithm is used to minimize the total time of simple queries (queries in which all relations have the same joining attribute), Parallel algorithm is used to minimize the response time and General algorithm which decomposes complex transactions into simple transactions and then uses the algorithm parallel or serial to minimize the response time or total time. As can be expected, the serial and parallel algorithms can be represented straightforwardly by choosing the appropriate routing graph, with the additional restriction that site selection is done in such a fashion so that nodes with smaller datasets are selected before the nodes with larger datasets. The algorithm 'general' can be adequately represented in our model because a complex transaction t can be decomposed into simpler transactions t1,t2,,,tn, where each simpler transaction can now be represented by using the application system parameters $T_e(t,p)$, $P_r(p,d)$, $P_w(p,d)$, $P_m(p1,p2)$, etc. Note that the parameter $P_m(p1,p2)$, which shows the number of temporary records exchanged between programs p1 and p2, can be used to implicitly describe relationships between transactions t1 and t2 because program p1 may be activated by transaction t1 while program p2 may be activated by transaction t2.

Chu & Hurley Algorithm: This algorithm basically allows evaluation of different operating policies for minimizing the communication plus the local processing cost. The six operating policies used to illustrate the algorithm can be represented by using the SRCH, Q_{rt}, Q_{da}, and application system parameters because these strategies use a three program/three dataset transaction. As stated earlier, we can represent all the operating policies of transactions with small number of datasets and programs.

INGRES Algorithm: This algorithm employs decomposition to achieve query optimization and is a generalization of the centralized query optimization approach used in the Centralized INGRES. The main idea is to fragment (subdivide) one relation and replicate others so that each site, containing a fragment and the replicated relations, can perform joins in parallel. This algorithm cannot be adequetely represented by us since we do not allow subdivision of datasets.

iv) CAPABILITIES AND LIMITATIONS OF THE PARAMETERS: The parameters can represent the restrictions and capabilities of support systems. For example, the SRCH matrix can be used to show minimum cost access and the Q_{rt} and Q_{da} parameters show if parallel/serial or local/rem ote data accesses are allowed for a given support system. The parameters can also describe complex transactions and can correctly represent ALL the operating policies of two program/two dataset transactions. In addition, the parameters can specify most of the currently available query optimization algorithms and can represent some situations that cannot be represented by current algorithms. For example, these parameters allow update as well as retrieval and facilitate remote data access. However, only a few operating policies of the transactions with large number of programs can be represented correctly. Although, the model can be easily extended to introduce new parameters to handle more operating policies. It should be noted that the problem of determining optimal transaction processing policy is NP-hard for complex transactions <HEVN81d> and thus choosing and analyzing few "reasonable" policies is a practical approach.

4.3 Update Synchronization and Concurrency Control

The subject of update syhchronization in distributed systems, also referred to as the concurrency control algorithms, is concerned with coordinating (syhchronizing) the multiple user access to a distributed

database, while maintaining concurrency and consistency. Update synchronization can effect the local and inter-node traffic plus the response time. In this chapter we only concentrate on calculating the traffic in terms of local and remote service requests (LSR and RSR) and will discuss the response time in the next chapter. Deadlock detection and resolution are not treated in this analysis. It should also be noted that we do not propose yet another synchronization algorithm, instead an attempt is made to determine the minimum number of parameters which can describe currently available algorithms so that the effect of these algorithms on program/data allocations can be quantitatively measured. To this end, we review the update synchronization problem, present a generalized update synchronization algorithm and introduce parameters to describe available algorithms.

4.3.1 Overview of Update Synchronization and Concurrency Control

The problem of synchronizing multiple user access to common objects involves: a) specification and control of the joint activity and b) serialization of concurrent access. Traditionally, this problem has appeared in the multiprogramming operating systems where the common objects are the devices being accessed by the users. When the common object is a database to be accessed by multiple transactions being issued by the users, then this problem is referred to as database synchronization, commonly termed update synchronization or concurrency control problem. Algorithms to solve this problem must both preserve the consistency of the database, where database consistency reflects a correct user's view of data, and maximize the concurrency, ie. achieve maximum number of transactions executed per unit time.

In this context, a transaction consists of a sequence of operations on one or more database objects (files, records, etc.) that transforms a current consistent state into a new consistent state <ESWA76d>. Thus if

a transaction completes successfully, the new state of the database is automatically consistent. Theoretically, it is possible to define a set of *consistency constraints* which govern the database operations, however such constraints are usually difficult to define in practice. A simple example of consistency constraint for a bank database is:

Balance = Deposit - Withdrwal

Although a transaction produces a consistent database at normal termination, the database may be temporarily inconsistent *during* the execution of a transaction. For example, during the execution of a transaction that moves money from one bank account to another, the consistency constraint stated above will be violated when one account has been debited but the other account has not been credited yet. This presents the basic problem in concurrency control: How to prevent other transactions from accessing the inconsistent data and still maintain minimum delay? This problem has been studied extensively in centralized database systems and an accepted solution is available. However, the situation in case of distributed systems is far from settled.

CENTRALIZED CONCURRENCY CONTROL: The method used to solve the centralized concurrency control is called two phase locking mechanism and is primarily due to Eswaran <ESWA76d>. This method has been implemented in several database management systems and consists of the following features: a) *locks* are maintained for each data object (relation, tuple, attribute) where a lock is basically a sofware indicator which shows, in the simplest case, if given data object is being accessed (held) or is free, b) when a transaction attempts to access, i.e. read or write, a data object, it has to first obtain associated locks and a lock is granted only if it is not held by any other transaction, c) if a lock is granted, then the transaction proceeds, else it must wait for locks, and d) all the locks held by a transaction are released (unlocked) when a transaction terminates.

The two phase behavior of a transaction is due to the following theorem by Eswaran <ESWA76d>:

"To guarantee database consistency, a transaction must not issue a new lock request after it has released *any* of its locks."

A transaction, which follows this theorem, is called a "well formed transaction" and exhibits two non-interchangeable phases: growing phase in which the transaction issues all locks and shrinking phase in which the transaction releases its locks. The locking mechanism is implemented by the support system and is totally transparent to the user. The implementation consists of a lock manager, a control program, which manages the lock setting/resetting and a lock table, a control database, where all the locks are maintained.

A process which is sometimes confused with the two phase locking is "two phase commit." This process is used primarily to handle transaction failure and consists of two phases: pre-write phase in which the database updates are written to a secure storage (usually tape) and commit phase in which the updates are actually applied to the database. Most common centralized database management systems (DBMS) employ variants of two phase locking and two phase commit techniques.

DISTRIBUTED CONCURRENCY CONTROL: The problem of concurrency control in distributed systems is complicated because the data accessed by users may exist at several computers, which leads to problems of selecting and updating duplicate data; also, a concurrency control algorithm at one site cannot know about interactions at other sites instantaneously which creates multi-site coordination problems. Thus, in addition to the consistency/concurrency requirements for a centralized environment, the distributed concurrency control algorithms should be resilient to failures at the node and communication levels, incur minimal computational and storage overhead, perform satisfactorily in networks where the communication delay is significant and detect and resolve dead-

locks, where a *deadlock* occurs, for example, when a transaction t1 waits on locks held by transaction t2 while t2 is waiting on locks held by t1.

A wide variety of distributed concurrency control algorithms have been proposed and continue to appear regularly in computing journals. Generally speaking, these algorithms are variations of the centralized 2PL (two phase lock) approach. Examples of such algorithms are the centralized algorithms presented by Garcia <GARC79d>, the Alsberg algorithm <ALSB76d> and the Thomas Voting Algorithm <THOM79d>. Some algorithms use *timestamps*, which show the clock time of an activity, to serialize the concurrent operations, instead of the locks. These systems generate unique timestamps for each transaction request and also maintain the timestamps for the accessed data to serialize the transactions. Bernstein <BERN81d> and Hsiao <HSIA81d> give examples of several time stamp based algorithms. In addition, miscellaneous algorithms which utilize special techniques, for example conflict analysis, to achieve concurrency control, are also found. The algorithms used in SDD1 <BERN80d> are an example. We will briefly review and represent main algorithms later. For details, the reader is referred to the excellent surveys by Kohler <KOHL81d>, Bernstein and Goodman <BERN81d>, and Hsiao <HSIA81d>. Other classifications and analysis of these algorithms can be found in <ADIB79a, GARC79d, ROTH77a>.

The following points are worth noting about synchronization algorithms. First, despite the large number of approaches proposed, no universally accepted update synchronization/concurrency control algorithm exists for distributed systems. Also, very little is known at present about the relationship between the application systems and the update synchronization algorithms. In other words, the following question is difficult to answer: What type of update synchronization algorithm is suitable for a particular application? Finally, it is extremely difficult to compare one algorithm against the other due to the conflicting terminologies and difference in emphasis <BERN81d, HSIA81d>.

4.3.2 A Generalized Update/Synchronization Model

Figure 4.4 shows a generalized distributed transaction model which can be used to represent update synchronization in DDP. The transaction initiation and termination phases have been discussed in the last chapter. The update synchronization activities are related to the execution phase which consists of: a) *Open processing*: All the locks are acquired in this step. Thus, for the 2PL transactions, this starts the "growing phase." In case of timestamped algorithms, timestamps are generated for each transaction. b) *Primary processing*: The actual read/write operations are performed in this step. For locking based algorithms, the read/write operations are performed only if the locks have been granted. In case of timestamp based algorithms, the timestamps are attached to the read-write request so that the requests can be serialized. c) *Secondary (commit) processing*: All the duplicate and consistency related data is updated in this step. d) *Close processing*: All the locks are freed in this step. This represents the shrinking phase for locking based algorithms and corresponds to erasing the timestamps.

During these steps, several service requests are issued to a collection of control programs and databases, which can be classified into two major systems: a transaction management system and a database management system. The transaction management system is responsible for the flow of transactions in the system and consists of a transaction manager and transaction directory. The database management system (DBMS) is responsible for the access and manipulation of the user data and consists of a database manager and a database directory.

In distributed systems, a distributed database management system (DDBMS) is introduced so that the data existing at several nodes can be accessed and manipulated by transactions originating at arbitrary nodes. Several views and implementations of DDBMS have been reported in literature <PLEI77d, TOTH78d, ADIB78d, BRET79d, FARB74d, FARB78d,

GERM80d, HELL77d, MAGE80d, ROTH77d, SMAL79d, STON77d>. Our view of the DDBMS is given in figure 4.4. As can be seen, the DDBMS consists of three control programs (database controller, lock/stamp manager, commit processor) and three control databases (data dictionary, lock/stamp tables and logs). These DDBMS components can be centralized or duplicated. To simplify analysis, we will assume that the database controller is always distributed, the lock/stamp manager and the commit processor can be centralized or distributed, and all the control databases coexist with the control programs which primarily manage the control databases. For example, we assume that the lock/stamp table is centralized if the lock/stamp manager is centralized and vice versa. We will discuss the impact of this assumption later. To illustrate the interactions between these components and the program and database allocation, consider a single program-single dataset transaction t which activates a program p which issues r reads and w writes to the dataset d. Let us further assume that the dataset d is duplicated at three different nodes. Then the following processing takes place (see figure 4.5):

a. Open Processing: All the dataset copies are locked so (r + 3w) lock requests are sent from p to the lock manager.

b. Primary Processing:

Program p issues r reads to the primary copy of d.

Program p issues w writes to the primary copy of d.

These calls are handled by the database controller.

c. Secondary Processing (at expiration of consistency interval):

Program p sends duplicate updates (w) to the commit processor.

Commit processor performs duplicate updates (sends 2w updates).

d. Close Processing: The commit processor sends the unlock requests (r + 3w) to the lock manager.

These service requests can be local or remote depending upon the location of the database management components and will be calculated lat-

FIGURE 4.4 A GENERALIZED CONSISTENCY- CONCURRENCY ALGORITHM

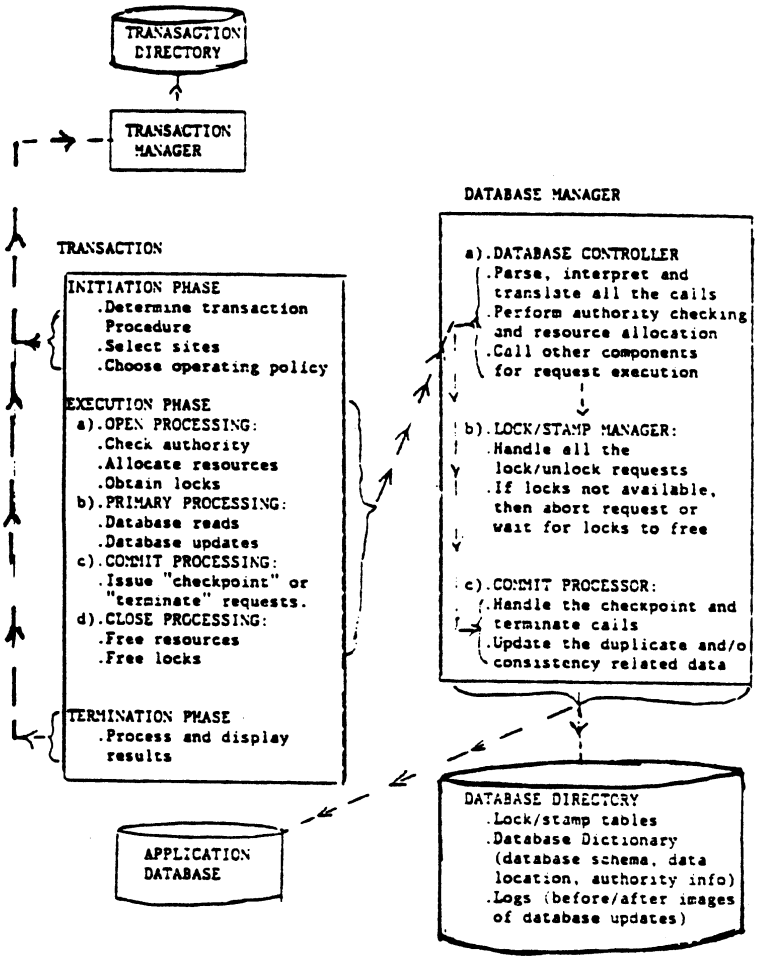
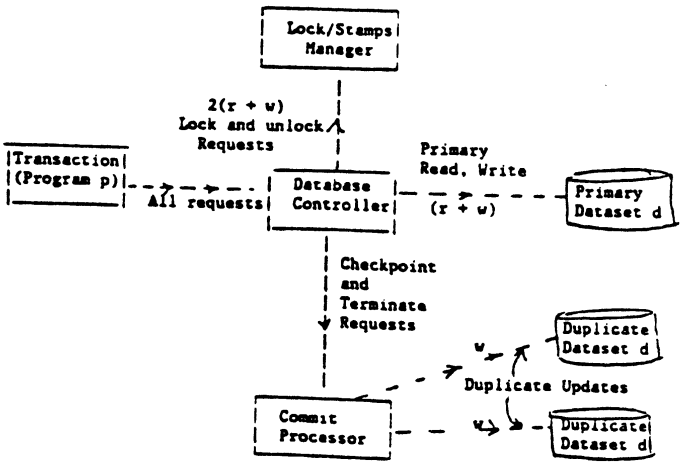


FIGURE 4.5 ILLUSTRATION OF UPDATE SYNCHRONIZATION MODEL



er on to estimate the local and remote processing costs. In order to use the model, we also need to define and discuss the following key concepts:

Timestamps versus Locks: A timestamp is a unique number which is assigned to a data object or a transaction and is usually chosen from a clock which yields monotonically increasing numbers. Basically, each arriving transaction is assigned a unique timestamp and the synchronization techniques serialize the transactions in a predetermined time-stamp order. In case of locks, a transaction can issue locks and unlocks any time and the synchronization techniques basically grant or deny the locks. Bernstein and Goodman <BERN81d> have given a detailed analysis of locking versus timestamp synchronization techniques. For the purpose of traffic calculation, the focal point of this chapter, the only difference between locking and timestamp algorithms is that in locking algorithms the lock/unlock requests can be routed separately and to nodes other than the accessed data nodes, while in case of time-stamped algorithms the timestamps are automatically concatenated with the transaction read/write requests. Thus timestamps do not generally increase the internode traffic. However, if the lock/unlock requests are "piggybacked" with the read/write requests, then the traffic generated in both the cases is same. For this reason, we will not explicitly differentiate between locking and timestamp synchronization techniques unless needed.

Consistency Granularity: This is the smallest portion of the database (relation, tuple, attribute) that can be accessed or updated by two or more concurrent transactions so that the database consistency is maintained. For example, in a locking system the consistency granularity is at a file level if the support system locks/unlocks at nothing lower than a file. Similarly, the consistency granularity of a timestamped system is at a tuple level, if timestamps are maintained for every tuple updated. The granularity at the relation level is referred to

as the "coarse" or "intent" level while the tuple/field granularity is referred to as "fine" or "operation" level granularity. Several tradeoffs exist between the coarse and fine granularity. First of all, coarse granularity is much easier and inexpensive to implement than fine granularity (this is the reason why most older flat file systems have implemented this type of granularity). Secondly, the size of the locks/stamp tables increases rapidly as the granularity level is reduced (made finer). Thirdly, for coarse granularity, the number of lock/unlock requests are much smaller than fine granularity, but the probability of conflict ("blocking" of one request by another) increases. Ries and Stonebraker <RIES79d> has done considerable research on locking granularities by using simulation models.

Consistency (synchronization) Interval: This shows the duration for which the database is locked. In most cases, this interval is the duration of the transaction since transaction is the usual unit of consistency in a 2PL (two phase locking) system. However, a transaction may decide to "commit", i.e. declare that the database is in a consistent state, at any time, called a checkpoint. At the checkpoint, the consistency interval expires, the database is consistent and thus all locks can be released. Basically, the consistency intervals can exist at several levels: transaction level when the consistency is maintained at the end of a transaction, subtransaction level when the consistency is maintained at the end of each subtransaction (checkpoint is issued at the end of each subtransaction) and operation level when the checkpoint is issued after every update operation. The concept of checkpoints has been discussed at length by Gray <GRAY78d> and has been implemented in several centralized database systems like IMS.

Consistency Related Data: As defined in Chapter 2, a consistency relationship exists between datasets d1 and d2 if d2 needs to be updated, whenever d1 is updated. Consistency relationships exist due to two reasons: *explicit* definitions through the parameter D_c(d1, d2) which

shows if d2 is updated whenever d1 is updated, and *implicit* definitions due to existence of duplicate data which must be updated after the primary data has been updated. It is the responsibility of the database manager to keep track of the explicit as well as implicit consistency relationships.

4.3.3 Update Synchronization Parameters

Five parameters (U_{cp}, U_{lm}, U_{pg}, U_{cr} and U_{sl}) are introduced to represent update synchronization algorithms. These parameters show the centralization/ decentralization of the DDBMS components and other synchronization options.

The U_{cp} parameter shows if the commit processor is centralized (0) or distributed (1). If the commit processor is centralized, then all the commit requests are sent to the central node and the central node performs all the commit processing by updating all the duplicate and consistency related data. This implies that all the duplicate update requests are routed, from a transaction arrival node k, to a central node z which then synchronizes and performs the updates, and then informs the arriving node k that the updates have been performed. In case of distributed commit processing, every arrival node k is responsible for synchronizing and performing all the duplicate updates. It should be recalled that commit processing can be performed after the transaction has terminated, as a standalone utility, or can be imbedded in the transaction so that when the transation terminates, all the duplicate and consistency realted data has been updated.

The U_{lm} parameter shows if the lock manager is centralized (0) or distributed (1). Once again, if the lock manager is centralized then all the lock/unlock requests are performed at a central node. This implies that all the lock/stamp information as well as the control program to access this information is restricted to a central node. Thus, in

the growing phase of a transaction, all the lock "grant" requests are routed to the central node and in the shrinking phase all the lock "free" requests are also routed to the central node. However, if the lock management is distributed, then each node will have a copy of the lock/stamp table, which may need to be updated whenever this table changes. This in turn generates inter-node traffic solely to synchronize the lock/stamp tables.

The U_{pg} parameter shows the amount of piggybacking used in update synchronization. A lock request can be issued as a separate request or can be concatenated (piggybacked) with a read/write request. Similarily, the lock grant request can be piggybacked with the messages coming from the database to the program. Full piggybacking implies that all the lock/unlock requests do not add extra overhead. If no piggybacking is used, then every lock/unlock request is treated as a separate service request. This parameter can have three values: 0 to indicate full piggybacking, 1 to indicate one-way piggybacking and 2 to indicate no piggybacking.

The U_{cr} parameter shows if the lock/stamp databases are core resident or not. If the locks/stamps are core resident, then effectively all local lock/unlocks can be ignored. However, if the locks/stamps are maintained on secondary storage, like disk devices, then each lock set/reset can be treated as a service request and adds to the local processing cost. Naturally, if the locks in the system are maintained at a high level, say file, then the size of the lock table is small enough to be loaded in main memory. However, for systems with large databases where the locking granularity is at a tuple or attribute level, the locks are usually kept on disks.

The U_{sl} parameter can be used to show the strategy employed for locking/unlocking. "Broadcast" (0) is used to indicate that the request is sent to ALL the nodes and "addressing" (1) is used to indicate that lock requests are sent only to those nodes where the data to be used by

the transaction is stored. For example, consider a 5 node network with dataset d stored at nodes 1 and 2. Then a transaction t which accesses d will cause all the lock/unlock requests to be sent to all the 5 nodes in case of broadcast and will send this request only to nodes 1 and 2 in case of addressing. The tradeoff between the broadcasting and addressing systems is that broadcast systems add considerable overhead but are simpler to implement, while the addressing systems are more efficient but require that the location of the primary as well as duplicate data is maintained in the directories at all times. Table 4.2 summarizes the update synchronization parameters.

The parameters defined above cannot describe all the available update synchronization algorithms, but these parameters plus the transaction processing parameters can be used effectively to represent the various classes of update synchronization algorithms. The algorithms that can be represented adequately are shown in Table 4.3. These algorithms are briefly described in appendix D.

4.4 The Extended Allocation Problem

The program and data allocation problem can now be extended to include the effect of data distribution, transaction processing and update synchronization options:

DETERMINE: the 0,1 decision variables:

. $\alpha(d,i)$ - Assignment of dataset d to node i

. $\beta(p,i)$ - Assignment of program p to node i.

TO MINIMIZE: Total cost =

Communication Cost + Node Processing Cost + Data Storage Cost

$$=\Sigma_t\Sigma_k\Sigma_i\Sigma_j T_a(t,k)R(t,k,i,j)C_R(i,j)+\Sigma_t\Sigma_k\Sigma_i T_a(t,k)L(t,k,i)C_L(i)+\Sigma_i S(i)C_S(i) \quad \ldots (4.1)$$

Where:

TABLE 4.2 THE UPDATE SYNCHRONIZATION PARAMETERS

U_{cp} = location of the commit-processor: distributed (1) or centralized (0).

U_{lm} = location of locks/stamps: distributed (1) or centralized (0).

U_{pg} = the amount of "piggybacking" in synchronization:
- = 0 complete piggyback
- = 1 partial piggyback (request or response)
- = 2 no piggybacking - all locks/unlocks sent separately

U_{cr} = the locks/stamps are core resident (0) or not (1)

U_{sl} = the lock/unlock requests are broadcast (0) or addressed (1).

TABLE 4.3: REPRESENTATION OF COMMON ALGORITHMS IN TERMS OF THE SYNCHRONIZATION AND QUERY PROCESSING PARAMETERS

	UPDATE SYNCHRONIZATION PARAMETERS					TRANSACTION PROCESSING PARAMETERS	
	U_{cp} = 0 if commit processor is centralized	U_{lm} = 0 if Lock Manager is centralized	U_{pg} = (0,1,2) shows the piggybacking	U_{cr} = 1 if Locks/Stamps are core resident	U_{sl} = 1 if locks broadcastd	Q_{da} = 1 remote date access is allowed	Q_{rt}=1 if routing graph is parallel
Basic 2PL Algorithm	1	1	2	0,1	1	0,1	0,1
Basic Time-stamped Alg	1	1	1	1	1	0,1	0,1
Garcia's CCAA & CCA	0	0	2 for CCAA 1 for CCA	0,1	0	0	1
Garcia's CLAA & CLA	0	1	2 for CLAA, 1 for CLA	0,1	0	0	1
Ellis Ring Algorithm	1	1	2	0,1	0	0	0
Thomas Voting Algorithm	1	1	1.5	0,1	0	0,1	0,1
Pessimistic & all site locking	1	1	2	0,1	1	0,1	0,1
Optimistic Algorithms	1	1	0	0,1	0	0,1	0,1
SDD1 Algorithms	1	1	2	0,1	0	1	1

- $R(t,k,i,j)$ = no. of remote service requests (interactions) exchanged between nodes i and j, when transaction t arrives at node k.
- $L(t,k,i)$ = no. of local service requests (interactions) executed at node i due to arrival of transaction t at node k.
- $S(i)$ = storage occupied at node i (in characters) .
- $C_R(i,j)$, $C_L(i)$ & $C_S(i)$ are the remote, local processing and storage costs.
- $T_a(t,k)$ is the arrival rate of a transaction at node k.

GIVEN:

(1). The SUPPORT SYSTEM INFORMATION :

. N_n - the number of nodes in the network

. COPY - the number of maximum copies allowed.

. The transaction processing options SRCH, Q_{rt} and Q_{da}.

. The update synchronization parameters U_{lm}, U_{cp}, U_{pg}, U_{sl} and U_{cr}.

2). The APPLICATION SYSTEM INFORMATION:

. The database, program and transaction parameters defined in section 2.2 and shown in table 2.1.

SUBJECT TO:

$\Sigma_i \alpha(d,i) \leq$ COPY (Number of dataset copies must not exceed COPY)

$\Sigma_i \alpha(d,i) > 1$ (each dataset must be assigned to at least one node)

$\Sigma_i \beta(p,i) > 1$ (each program must be assigned to at least one node)

ASSUMPTIONS: The support system assumptions 1 to 5 stated in section 2.4, are applicable to this formulation. This problem differs from the unconstrained problem described in section 2.3 in two respects: i) the assumptions 7 and 8 which ignore the data distribution and query processing options have been removed to introduce the parameters COPY, SRCH, Q_{rt} and Q_{da}, and ii) the assumption 6 which ignores the update synchro-

nization cost has been removed to introduce the update synchronization parameters U_{lm}, U_{cp}, U_{pg} and U_{cr}. In addition:

- Each lock/unlock is considered as a service request. Thus the local and remote processing cost calculations give the same weight to a lock/unlock request as a dataset access since cost = no. of service requests × cost per service request. This may exaggerate effect of locking/unlocking.
- An implicit assumption used in total cost calculation is that the transaction arrival node is also the transaction result node (the node where the results are displayed).
- Locking granularity is assumed to be at a tuple level, so that the number of locks requested by a transaction (subtransaction) is equal to the total number of reads/writes issued by the transaction. We will remove this assumption in the next chapter and study the effect of locking granularity on the locking conflicts and transaction turnaround delays.
- Consistency interval is assumed to be at a subtransation level, i.e. every subtransaction is treated as a unit of consistency and thus "commit" processing is performed at the end of every subtransaction.
- The logging activity due to two phase commit is assumed negligible. This may be due to fast secure storage logging or main memory logging (buffered logs). This assumption can be easily removed by introducing a new parameter which takes into account the effect of logging.
- The addressing scheme for sending locks will be assumed, unless the broadcast scheme is explicitly specified.
- The amount of storage occupied by locks/stamps is negligible.

4.5 Calculation of Total Cost

In order to solve the problem presented above, we need the analytical expressions for the dependent variables S, L and R. The expression for S is unaffected by the introduction of the transaction processing and update synchronization algorithm because we have assumed that locks/stamps do not occupy additional storage:

$$S(i) = \Sigma_i \ \alpha(d,i)\ D_s(d,1)\ D_s(d,2)$$

where $D_s(d,1)$ and $D_s(d,2)$ show the tuple size and occurrence in dataset d, respectively, and α is the data allocation decision variable. Please note that S can be easily modified to reflect size increases due to locks/stamps, by adding the following factor to S:

$$K \times D_s(d,2)\ [\ U_{lm} + (1-U_{lm})\delta(i,z)\ ]$$

where K = size of a lock, $U_{lm} = 1$ shows that the lock manager is distributed and $\delta(i,z) = 1$ if i is the central node z, 0 otherwise. For convenience, table 4.4 shows all the parameters used by the extended model (the decision variables, the application parameters, the support system parameters introduced so far and the cost variables). The predictive model presented in chapter 2 is now extended to recalculate L and R to include the effect of transaction processing options and update synchronization (section 4.5.1). The final expressions for S, L and R are shown in section 4.5.2.

4.5.1 Calculation of Service Requests

The site selection, transaction decomposition, transaction routing and display routing steps of the predictive model are affected by the transaction processing options and the open processing, primary processing, secondary (commit) processing and the close processing steps are affected by the update synchronization algorithms. The predictive model is now described and the analytical expressions for L and R are

TABLE 4.4 : THE MODEL PARAMETERS

1). ALLOCATION VARIABLES (INPUT)
- $\alpha(d,i)$ = 1 if dataset d is allocated to node i, 0 otherwise
- $\beta(p,i)$ = 1 if program p is allocated to node i, 0 otherwise

2). APPLICATION SYSTEM PARAMETERS (INPUT)

a). Data Description
- D_n = number of datasets in the application system.

a). DATA DESCRIPTION
(D = set of all the datasets in the application system)
- D_n = number of datasets in the application system.
- $D_c(d1,d2)$= consistency relationship between dataset d1 and d2; implies that d2 must be updated whenever d1 is updated .
- $D_s(d,x)$ = size (x=1) and occurrence (x=2) of tuples in dataset d.

b). PROGRAM DESCRIPTION
(P = set of all the programs in the application system)
- P_n = number of programs in the application system
- $P_r(p,d)$ = number of reads issued from program p to dataset d.
- $P_w(p,d)$ = number of updates issued from program p to dataset d.
- $P_c(p,d)$ = number of consistency updates issued from program p to dataset d.
- $P_s(p)$ = average number of input records read by program p
- $P_o(p)$ = display messages written by program p.
- $P_m(p,p1)$= No. of temperory records written by program p that are to be received by by program p1. Note that p and p1 can execute in parallel if $P_m(p,p1)=0$.

c). TRANSACTION DESCRIPTION
(T = set of all the transactions in the application system)
- T_n = total number of transaction types in the application system.
- $T_e(t,p)$= no. of times program p is executed per invokation of a transaction t
- $T_a(t,k)$= No. of times a transaction t arrives at node k for a given observation period.

3). SUPPORT SYSTEM PARAMETERS (INPUT)

a). Basic Parameters:
- N_n = No. of nodes in the network
- $C_S(i)$ = cost of storing at node i
- $C_L(i)$ = cost per local service request at node i
- $C_R(i,j)$ = cost per remote service request exchanged between nodes i and i

b). Transaction Processing Parameters
- Q_{rt} = the routing graph: parallel (1) or ring (0).
- Q_{da} = the data access mode: remote (1) or local (0).
- SRCH(k,x) = the search strategy for the queries arriving at node k.
- COPY = the maximum number of dataset copies allowed

c). Update synchronization options:

U_{cp} = location of the commit-processor: distributed (1) or centralized (0).

U_{lm} = location of locks/stamps: distributed (1) or centralized (0).

U_{pg} = the amount of "piggybacking" in synchronization:
- = 0 complete piggyback
- = 1 partial piggyback (request or response)
- = 2 no piggybacking

U_{cr} = the locks/stamps are core resident (0) or not (1)

U_{sl} = the lock/unlock requests are broadcast (0) or addressed (1).

4). OUTPUT PARAMETERS (OBSERVABLES)
a). ST(x) = storage occupied at node x
b). L(k,i) = local service requests executed at node i due to arrival of a transaction at node k
c). R(k,i,j) = remote service requests exchanged between nodes i and j due to arrival of a transaction at node k

derived at each step. The final expressions for L and R will be obtained in the next section by adding the L and R obtained in each step below. Throughout these calculations, R(t,k,i,j) = 0 if i=j.

STEP 1 - TRANSACTION INITIATION

This is the first step of a distributed transaction, after it arrives at a node. In this step, the transaction procedure is determined and the transaction directory is accessed to establish, or retrieve the pre-stored, transaction procedure. Since we are still assuming replicated and core resident directories (assumptions 3 and 5 stated in section 2.4), these accesses generate no local or remote service requests:

$$L(t,k,i) = 0 \quad \text{and} \quad R(t,k,i,j) = 0 \quad \text{for all } (k, i, j) \in N \qquad \ldots\ldots(4.10)$$

If the directories are not replicated and core resident, then the expressions for L and R will need to be rederived.

STEP 2: SITE SELECTION

In this step, the transaction t is decomposed into t' subtransactions, where each subtransaction is a collection of programs which execute at the same node and the sites are selected for all the programs (subtransactions) and datasets referenced by the transaction t, i.e. the site selection matrices α_s and β_s are determined, where $\beta_s(k,p,i) = 1$ if i is the site selected for program p when the transaction arrives at node k and $\alpha_s(k,p,d,i) = 1$ if i is the site selected for dataset d when accessed from the program (subtransation) p. It was shown above that, in this research, each program group can be viewed as a separate subtransaction.

It can be expected that the site selection is based on the SRCH matrix. Basically, $\beta_s(k,p,i) = \beta(p,i)$ where i = SRCH(k,x) for minimum x, for x=1 to N_n, and also $i \in I$, where I is the set of nodes that contain program p. Similarly, $\alpha_s(k,p,d,i)=1$ if $\alpha(d,i)=1$ and i = SRCH(k,x) for minimum x, 0 otherwise. The SRCH matrix is part of the transaction di-

rectory, consequently there is no local or remote activity in this step also due to reasons stated in step 1:

$L(t,k,i) = 0$ and $R(t,k,i,j) = 0$ for all $(k, i, j) \in N$ .....(4.11)

It should be noted that the β_s and α_s are dependent on the data access strategies represented by the Q_{da} parameter. For example, if data access is restricted to local, then $\alpha_s(k,p,d,i) = \beta_s(k,p,i)$ because the programs and data must exist at the same node for local data access.

STEP 3: OPERATING POLICIES AND (SUB)TRANSACTION ROUTING

In this step, the best operating policy is determined and then the subtransactions are routed according to the chosen operating policy. The service requests in this step depend upon the size of input data, the size of the temporary records exchanged and also on the routing graph Q_{rt}. The expression for $L(t,k,i)$ is straightforward:

$$L(t,k,i) = 0 \qquad \text{.....(4.12)}$$

However, considerable calculations are needed to compute $R(t,k,i,j)$ which, in this case, consists of two parts:

. The node to program traffic $R_{np}(t,k,i,j)$.

. The program to program traffic $R_{pp}(t,k,i,j)$.

The node to program traffic depends on the routing graph (parallel versus serial). For a parallel routing graph, the node to program traffic consists of all the input messages sent from the originating node to the program sites selected (see figure 4.3):

$$= \Sigma_p T_e(t,p) P_I(p) \beta_s(i,p,j) \qquad \text{.....(4.13)}$$

Recall that $\delta(k,i) = 1$ if $k=i$, 0 otherwise. The parameters $T_e(t,p)$, the number of times the program p is activated by a transaction, and $P_I(p)$, the number of input messages received by program p, are used frequently as multipliers in these calculations. Please note that this expression only takes into account the node to program input routing. The output message routing is shown in the display routing step. For a serial routing graph, the node to program traffic consists of all the messages sent from the originating node to the site selected for the first pro-

gram (operation) since in this type of routing graph, all the input messages are circulated through all the program sites (see figure 4.3):

$$= \Sigma_p T_e(t,p) P_I(p) \beta_s(i,p0,j) \quad \text{.....(4.14)}$$

where p0 shows the first program in the ring. Combining and rearranging equations 4.13 and 4.14, and recalling that Q_{rt}=1 represents parallel routing graph and Q_{rt}=0 represents a serial routing graph, we get the analytical expression for $R_{np}(t,k,i,j)$:

$$R_{np}(t,k,i,j)=\Sigma_p T_e(t,p) P_I(p) \delta(k,i) [Q_{rt} \beta_s(k,p,j)+\{1-Q_{rt}\}\beta_s(k,p0,j)] \quad \text{..(4.15)}$$

Now returning to calculation of the program to program traffic $R_{pp}(t,k,i,j)$. In case of parallel routing graph, this traffic consists of the temporary records exchanged between two program sites and is given by:

$$= \Sigma_p T_e(t,p) \Sigma_{p1} P_m(p,p1) \beta_s(k,p,i) \beta_s(k,p1,j) \quad \text{.....(4.16)}$$

The calculation of program to program traffic in serial routing graphs is difficult because the input messages plus the output messages, in addition to the temporary records, are sent from one program site to the next. This is due to the nature of the serial routing: the entire transaction (all input messages $P_i(1),P_i(2),,,P_i(n)$) are sent to the first site where the first operation is performed which eliminates the first input $P_i(1)$, and generates the output message $P_o(1)$. Now the transaction is routed to the second site (input messages $P_i(2),P_i(3),,,P_i(n)$ plus output message $P_o(1)$) where the input message $P_i(2)$ is eliminated and ouput $P_o(2)$ is generated. Thus when the transaction is routed from the second site to third site, it carries the input messages $P_i(3),P_i(4),,P_i(n)$ and output messages $P_o(1),P_o(2)$. In this fashion, the output messages $P_o(1),P_o(2),,,,P_o(n)$ are sent from the final site to the destination node. We can now write the analytical expression for the program to program traffic in a serial routing graph:

$$\Sigma_p T_e(t,p) \beta_s(k,p,i) \beta_s(k,p+1,j) [\sum_{p_2=p+1}^{P_m} P_m(p,p2) + \sum_{p_2=p+1}^{P_m} P_I(p2) + \sum_{p_2=1}^{p} P_o(p2)] \quad \text{..(4.17)}$$

where p+1 indicates that p2 varies from the program after p to P_n which where P_n is the last program in the system. Now combining equations 4.16 and 4.17 and using Q_{rt} to represent the type of routing graph (parallel or serial), we get the following expression for $R_{pp}(t,k,i,j)$:

$$R_{pp}(t,k,i,j) = \Sigma_p T_e(t,p)\beta_s(k,p,i)[Q_{rt}\Sigma_{p1} \beta_s(k,p1,j) P_m(p,p1) + \{1-Q_{rt}\beta_s(k,p+1,j)\}\{\Sigma_{p2=p+1} P_m(p,p2) + \Sigma_{p2=p+1} P_I(p2) + \Sigma_{p2=1} P_o(p2)\}] \quad \ldots\ldots\ldots (4.18)$$

The final value of R(t,k,i,j) in this step is:

$$R(t,k,i,j) = R_{np}(t,k,i,j) + R_{pp}(t,k,i,j) \quad \ldots\ldots\ldots\ldots\ldots\ldots (4.19)$$

where $R_{np}(t,k,i,j)$ is given by equation 4.15 and $R_{pp}(t,k,i,j)$ is given by equation 4.18.

STEP 4: OPEN PROCESSING

In this step, the (sub)transaction actually starts execution by checking authority and allocating resources (locks, cpu, memory etc.). The authority checking involves reading the external schema and the security tables to verify if the current transaction is authorized to perform the intended opeartions. The subject of security is beyond the scope of this research, however the net effect on cost can be included by counting the security table and schema (control database) accesses. The main issue of interest to us in resource allocation is the obtaining of locks/stamps for database access and is discussed in great detail here. In this context, the open processing step consists of: a) Interpretation of the open request from the transaction. In some programming languages, say Cobol and PL1, the open processing is initiated explicitly through an OPEN statement, while in other languages, like Fortran, the open processing is initiated through the first read/write instruction. b) The datasets/tuples to be read/updated are determined. c) If locking mechanism is being used, then requests to obtain locks are sent to the lock manager. The lock manager determines, through predetermined rules discussed above, if a lock request can be granted or denied. If the

locks are granted, the transaction proceeds to the next step or waits if locks are denied.

The locking activity can be described in terms of read locks, update locks and consistency related locks. Let us first calculate the local processing activities L(t,k,i), which can be expressed as:

$$L(t,k,i) = U_{cr} \{L1(t,k,i) + L2(t,k,i) + L3(t,k,i)\} \quad \ldots\ldots(4.20)$$

where L1(t,k,i) is the local open activity due to read, L2 is the local open activity due to updates and L3 designates the local open activity due to consistency related updates. The L1 occurs at the nodes which have been selected for dataset read. Thus, in case of distributed lock management (U_{lm}=1), the locking activity takes place at the nodes where the dataset is read and in case of centralized lock management (U_{lm}=0), the locking activity takes place at the central node z. Thus L1 is given by:

$$L1(t,k,i) = \Sigma_p T_e(t,p) P_I(p) \Sigma_d P_r(p,d) \{U_{lm}\alpha_s(k,p,d,i) + (1-U_{lm})\delta(i,z)\}$$

where P_r shows the dataset reads, α_s shows the dataset site selection matrix, $\delta(i,z)$=1 if i=z, 0 otherwise, and all other parameters have been defined in table 4.4. The lock activity due to updates (L2) and due to consistency related data (L3) causes activities where the duplicated and consistency related data resides. Thus:

$$L2(t,k,i) = \Sigma_p T_e(t,p) P_I(p) \Sigma_d P_w(p,d) \{U_{lm}\alpha(d,i) + (1-U_{lm})\delta(i,z)\}$$

$$L3(t,k,i) = \Sigma_p T_e(t,p) P_I(p) \Sigma_d P_c(p,d) \Sigma_{d1} D_c(d,d1) \{U_{lm}\alpha(d1,i) + (1-U_{lm})\delta(i,z)\}$$

The total L(t,k,i) is obtained by summing the expressions for L1, L2 and L3, and simplifying the final expression:

$$\begin{aligned} L(t,k,i) = U_{cr} \Sigma_p T_e(t,p) P_I(p) \Sigma_d U_{lm} [& \{P_r(p,d)\alpha_s(k,p,d,i) + P_w(p,d)\alpha(d,i) \\ & + P_c(p,d)\Sigma_{d1} D_c(d,d1)\alpha(d1,i)\} \\ & + (1-U_{lm})\delta(i,z)\{P_r(p,d) + P_w(p,d) + P_c(p,d)\Sigma_{d1} D_c(d,d1)\}] \end{aligned}$$

The computation of remote processing activity R(t,k,i,j) is, in principle, similar to the calculation of L(t,k,i), and is given by:

$$R(t,k,i,j) = U_{pg} \{R1(t,k,i,j) + R2(t,k,i,j) + R3(t,k,i,j)\} \quad \ldots\ldots(4.22)$$

where U_{pg} shows the amount of piggybacking involved in locking, and R1, R2 and R3 are respectively the remote services at open step due to read, write or consistency updates. The expressions for R1, R2 and R3 are similar to their counterparts in local processing:

$$R1(t,k,i,j)=\Sigma_p T_e(t,p)P_I(p)\Sigma_d P_r(p,d)\beta_s(k,p,i)$$
$$\times \{U_{lm}\alpha_s(k,p,d,j)+(1-U_{lm})\delta(j,z)\}$$

$$R2(t,k,i,j)=\Sigma_p T_e(t,p)P_I(p)\Sigma_d P_w(p,d)\beta_s(k,p,i)\{U_{lm}\alpha(d,j)+(1-U_{lm})\delta(j,z)\}$$

$$R3(t,k,i,j)=\Sigma_p T_e(t,p)P_I(p)\Sigma_d P_c(p,d)\beta_s(k,p,i)\Sigma_{d1} D_c(d,d1)$$
$$\{U_{lm}\alpha(d1,j)+(1-U_{lm})\delta(j,z)\}$$

The total R(t,k,i,j) is thus given by:

$$R(t,k,i,j)=U_{pg}\Sigma_p T_e(t,p)P_I(p)\beta_s(k,p,i)$$
$$[U_{lm}\Sigma_d\{P_r(p,d)\alpha_s(k,p,d,j)+P_w(p,d)\alpha(d,j)$$
$$+ P_c(p,d)\Sigma_{d1}D_c(d,d1)\alpha(d1,j)\}$$
$$+ (1-U_{lm})\delta(j,z)\{P_r(p,d) + P_w(p,d) + P_c(p,d)\Sigma_{d1}D_c(d,d1)\}]$$

STEP 5 - PRIMARY PROCESSING

In this step, the application programs actually read/update the application database. This step is complex and can be subdivided into the following substeps: a) request preparation which involves request parsing and query translation, b) request routing which consists of sending the request to a proper node for processing, c) request execution which involves the read/write of database and logging for recovery, d) response preparation which consists of preparing/converting the output data for the orginating node, and e) response routing which sends the results back to the originator. In case of a local access, the request routing and response routing steps are null while in case of homogeneous systems, the request preparation and response preparation steps are null since there is no need for query/data translation. For the purpose of this chapter, we can disregard the effect of request preparation and response preparation since the support systems are being assumed to be homogeneous and the control databases are also being assumed

to be core resident. The effect of response and request routing is actually taken into account in calculating RSR.

Now, let us calculate LSR and RSR. The LSR in this step is due to the read/update of data at the data sites selected:

$$L(t,k,i) = \Sigma_p T_e(t,p)P_I(p) \Sigma_d[P_r(p,d)+P_w(p,d)]\alpha_s(k,p,d,i)$$

The RSR in this step is due to the exchange of remote messages between program sites and data sites:

$$R(t,k,i,j) = \Sigma_p T_e(t,p)P_I(p) \Sigma_d [P_r(p,d)+P_w(p,d)] \beta_s(k,p,i) \times \alpha_s(k,p,d,j)$$

STEP 6: SECONDARY (COMMIT) PROCESSING

In this step, all the copies of data updated in step 5 are updated (synchronized). This step takes place only if the updated data is duplicated and/or has consistency relationships. This step consists of the following substeps: a).The transaction issues a commit by executing a "terminate" or "checkpoint" instruction. b).The database controller interprets the instruction and routes the request to a commit processor, which may be centralized or distributed. c).The commit processor updates all the duplicate and consistency related data. d).When all the updates are done, then the entire network has a consistent copy of the database. The commit processor sends the message 'committed' to the transaction.

The local service requests are executed at the nodes where the consistency related data exists, independent of the location of the commit processor:

$$L(t,k,i) = \Sigma_p T_e(t,p) P_I(p) \Sigma_d [\{ \Sigma_{d1} P_c(p,d) D_c(d,d1)\alpha(d1,i) \} + P_w(p,d)\{\alpha(d,i)-\alpha_s(k,p,d,i)\}]$$

The remote service requests are dependent on the location of the commit-processor. The updates are sent to the commit-processor which in turn applies the updates. The remote service requests can be expressed as:

$$R(t,k,i,j) = \Sigma_p T_e(t,p)P_I(p)[R_d(t,k,i,j)+R_c(t,k,i,j)] \quad \ldots\ldots (4.25)$$

where $T_e(t,p)$ and $P_I(p)$ are the program parameters shown in table 4.4 and:

$R_d(t,k,i,j)$ = remote service reqests due to duplicate data

$$= \Sigma_d P_w(p,d)[U_{cp}\beta_s(k,p,i)\{\alpha(d,j)-\alpha_s(k,p,d,j)\} + (1-U_{cp})\{\beta_s(k,p,i)\delta(z,j)+\delta(i,z)(\alpha(d,j)-\alpha_s(k,p,d,j))\}]$$

$R_c(t,k,i,j)$ = remote service requests due to consistency updates

$$= \Sigma_d P_c(p,d)\Sigma_i D_c(d,d1)[U_{cp}\beta_s(k,p,i)\alpha(d1,j) + (1-U_{cp})\{\beta_s(k,p,i)\delta(z,j)+\delta(i,z)\alpha(d1,j)\}]$$

All the parameters used in these equations are shown in table 4.4. The expressions for centralized commit-processor (U_{cp}=0) involve two terms. The first term shows the routing from the orginating node to the central node z and the second term shows the updates from the centralized node to the data nodes.

STEP 7: CLOSE PROCESSING

In this step, the resources and the locks acquired in step 4 are freed. The service requests are thus primarily due to freeing the locks acquired by the (sub)transaction in the open processing step and are equal to the number of requests issued in the open processing:

$$L(t,k,i) = U_{cr}\Sigma_p T_e(t,p)P_I(p)\Sigma_d U_{lm}[\{P_r(p,d)\alpha_s(k,p,d,i)+P_w(p,d)\alpha(d,i) + P_c(p,d)\Sigma_{d1}D_c(d,d1)\alpha(d1,i)\} + (1-U_{lm})\delta(i,z)\{P_r(p,d) + P_w(p,d) + P_c(p,d)\Sigma_{d1}D_c(d,d1)\}]$$

$$R(t,k,i,j)=U_{pg}\Sigma_p\beta_s(k,p,i)T_e(t,p)P_I(p) [U_{lm}\Sigma_d\{P_r(p,d)\alpha_s(k,p,d,j)+P_w(p,d)\alpha(d,j) + P_c(p,d)\Sigma_{d1}D_c(d,d1)\alpha(d1,j)\} + (1-U_{lm})\delta(j,z)\ \Sigma_p\Sigma_d\{P_r(p,d) + P_w(p,d) + P_c(p,d)\Sigma_{d1}D_c(d,d1)\}]$$

STEP 8 - DISPLAY ROUTING

In this step, the results computed in steps 5 and 6 are routed back to the originating node. The service requests depend upon number of display messages generated by each program and the routing graphs. The local service requests are insignificant:

$$L(t,k,i) = 0$$

The R(t,k,i,j) depends on the routing graph value Q_{rt}. Let pn represent the last program in the program list. Then:

$$R(t,k,i,j)=\Sigma_p T_e(t,p)P_I(p)P_o(p)\delta(k,j) \times [Q_{rt}\beta_s(k,p,i)+\{1-Q_{rt}\}\beta_s(k,pn,i)]. \quad \ldots\ldots\ldots\ldots(4.26)$$

This routing traffic is primarily due to the node to program messages and does not include the program to program messages.

STEP 9 - TERMINATION

This step terminates the processing of the entire transaction and displays the messages back to the user. We assume no activity in this step:

$$L(t,k,i) = 0 \qquad R(t,k,i,j) = 0 \quad \text{for all } (t,k,i,j) \in N$$

4.5.2 The Analytical Expressions for S, L and R

Let us now show the final expressions for storage occupied (S), local service requests (L) and remote service requests (R). We need these expressions to compute the objective function shown in equation 4.1.

The expression for S is straightforwrad:

$$S(i) = \Sigma_i\ \alpha(d,i)\ D_s(d,1)\ D_s(d,2) \quad \ldots\ldots\ldots\ldots\ldots\ldots(4.28)$$

where D_s shows the data size and α is the data allocation decision variable.

The expression for L(t,k,i), the number of local service requests issued at node i due to the arrival of transaction t at node k, is obtained by calculating the read activity at the primary nodes, the update activity at all the nodes where the duplicate, and/or the consistency related, data resides and all the locking/unlocking activities. Thus:

$$L(t,k,i) = L_{pd}(t,k,i) + L_{lk}(t,k,i) \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots(4.29)$$

where L_{pd} and L_{lk} are given by:

$L_{pd}(t,k,i)$ = local service requests issued at node i due to arrival of a tranasction at node k for primary and secondary read and writes. This activity is obtained by summing

the LSR at the primary and secondary processing steps.

$$= \Sigma_p T_e(t,p) P_I(p) \Sigma_d [\, P_r(p,d)\alpha_s(k,p,d,i) + P_w(p,d)\alpha(d,i) + P_c(p,d)\Sigma_{d1} D_c(d,d1)\alpha(d1,i) \,] \quad \ldots\ldots\ldots\ldots\ldots\ldots (4.30)$$

$L_{lk}(t,k,i)$ = local service requests, due to locking/unlocking, executed at node i due to arrival of a transaction at node i. This activity is obtained by summing the LSR in the open and close processing steps:

$$= 2U_{cr}\Sigma_p T_e(t,p) P_I(p) \Sigma_d [\, U_{lm} \{P_r(p,d)\alpha_s(k,p,d,i) + P_w(p,d)\alpha(d,i) + P_c(p,d)\Sigma_{d1} D_c(d,d1)\alpha(d1,i)\} + (1-U_{lm})\delta(i,z)\{P_r(p,d)+P_w(p,d)+P_c(p,d)\Sigma_{d1} D_c(d,d1)\}] \ldots\ldots (4.31)$$

In the expressions for L_{pd} and L_{lk}, U_{lm} shows the location of the lock manager, and $P_r(p,d)$, $P_w(p,d)$, $P_c(p,d)$ are the data manipulation parameters, $D_c(d,d1)$ is the consistency relationship, $T_e(t,p)$ represents the number of times the program p is activated by a transaction, and $P_I(p)$ is the number of input messages received by program p.

The expression for R(t,k,i,j), the number of remote service requests exchanged between nodes i and j when a transaction t arrives at node k, is obtained by calculating the node to program, program to program, program to data traffic and the lock/unlock traffic:

$R(t,k,i,i) = 0$ and $R(t,k,i,j) = R(k,j,i)$

$$R(t,k,i,j) = R_{np}(t,k,i,j) + R_{pp}(t,k,i,j) + R_{pd}(t,k,i,j) + R_{lk}(t,k,i,j) \quad \ldots\ldots\ldots (4.32)$$

Where R_{np}, R_{pp} and R_{pd} are given by:

$R_{np}(t,k,i,j)$ = node to program remote services exchanged due to reading of input data, from nodes, and sending of display messages, to nodes, by programs. This traffic is obtained by summing equations 4.15 and 4.20 (transaction routing + display):

$$= \Sigma_p T_e(t,p) P_I(p)\, \delta(k,i)[\{1+P_o(p)\}Q_{rt}\beta_s(k,p,j) + \{1-Q_{rt}\}\{\beta_s(k,p0,j)+P_o(p)\beta_s(k,pn,j)\}] \quad \ldots\ldots (4.32a)$$

$R_{pp}(t,k,i,j)$ = program to program remote services exchanged to transfer

temporary data between programs at different sites:

This traffic is given directly by equation 4.18:

$$= \Sigma_p T_e(t,p)\beta_s(k,p,i)\,[Q_{rt}\Sigma_{p1}\,\beta_s(k,p1,j)\,P_m(p,p+1)$$

$$+ \{1-Q_{rt}\beta_s(k,p+1,j)\}\{\Sigma_{p=p_n}^{P_n} P_m(p,p2) + \Sigma_{p=p_{n+1}}^{P_n} P_I(p2) + \Sigma_{p=1}^{P} P_o(p2)\}] \quad \ldots\ldots\ldots(4.32b)$$

where P_n = number of programs in the application system.

$R_{pd}(t,k,i,j)$ = program to data traffic due to the read/update of remotely located data by programs. Note that the update traffic modifies the actual data plus the consistency related data. This traffic is obtained mainly from the primary and secondary processing steps and are dependent on the location of the commit-processor:

$$R_{pd}(t,k,i,j)=\Sigma_p T_e(t,p)P_I(p)[U_{cp}R'_{pd}(t,k,i,j)+(1-U_{cp})R''_{pd}(t,k,i,j)]$$

where $R'_{pd}(t,k,i,j)$ is given by:

$$= \Sigma_d P_r(p,d)\,\beta_s(k,p,i)\,\alpha_s(k,p,d,j) + \Sigma_d P_w(p,d)\,\beta_s(k,p,i)\,\alpha(d,j)$$

$$+ \Sigma_d P_c(p,d)\beta_s(k,p,i)\,\Sigma_{d1}D_c(d,d1)\alpha(d1,j) \quad \ldots\ldots\ldots(4.32c)$$

and $R''_{pd}(t,k,i,j)$ is given by:

$$= \Sigma_d [P_r(p,d)+P_w(p,d)+P_c(p,d)\Sigma_{d1}D_c(d,d1)]\beta_s(k,p,i)\delta(z,j) + \delta(z,i)$$

$$\times[P_r(p,d)\alpha_s(k,p,d,j)+P_w(p,d)\alpha(d,j)+P_c(p,d)\Sigma_{d1}D_c(d,d1)\alpha(d1,j)] \ldots(4.32d)$$

$R_{lk}(t,k,i,j)$ = lock/unlock remote traffic due to the arrival of a transaction at node k. This traffic is obtained by adding the RSR in the open ad close processing steps:

$$=2U_{pg}\Sigma_p T_e(t,p)P_I(p)[\beta_s(k,p,i)U_{lm}\Sigma_d\{P_r(p,d)\alpha_s(k,p,d,j)+P_w(p,d)\alpha(d,j)$$

$$+ P_c(p,d)\Sigma_{d1}D_c(d,d1)\alpha(d1,j)\}$$

$$+(1-U_{lm})\delta(j,z)\{P_r(p,d)+P_w(p,d)+P_c(p,d)\Sigma_{d1}D_c(d,d1)\}] \ldots(4.32e)$$

Where $P_I(p)$ and $P_o(p)$ show the number of input and output messages received/generated by program p, $P_m(p,p1)$ are the number of temporary records generated by program p for p1 and $\delta(k,i)$ is a 0-1 variable such that $\delta(k,i) = 1$ if i=k, zero otherwise. Note that the all the parame-

ters introduced in this chapter appear explicitly in equations for S, L and R. This is because some variables are included implicitly in the calculations. For example, the Q_{da} parameter is not included in the equations because it is used to calculate the site selection matrices α_s and β_s.

4.5.3 Comments on the Predictive Model

Figure 4.6 shows how the predictive model introduced in chapter 2 has been extended to include the effect of transaction processing and update synchronization algorithms on the local and remote service requests. So far the output of this model (the service requests) have been used primarily to calculate the objective function, however, as we shall see in the next chapter, the service requests play key role in computation of transaction turnaround, transaction availability and other dependent variables like disk activity, cpu activity, transport volumes and the like.

The major strength of the predictive model is that the flow of transactions in DDP is subdivided into several well defined steps in which well defined activities take place. In order to extend this model, one needs to know which step(s) are affected, modify the expressions for LSR and RSR at the affected steps and then recompute the total expressions. For example, we can easily include the following factors in this model:

Dirctory Distribution: So far, we have been assuming that the directories are replicated and core resident. This assumption can be relaxed by recomputing the L(t,k,i) and R(t,k,i,j) in the transaction decomposition, open processing and close processing steps for non-replicated and disk resident directories.

Logging: The logs are created for two-phase commit processing so that the effects of a failing transaction can be removed. Thus all the updates of a transaction are logged on a secure storage like tape or disk.

The basic effect of logging is that for every update, the L(t,k,i) are increased. Although the logs can be theoretically centralized, practical applications of such systems are not clear. The effect of logging can be easily included by simply increasing the L(t,k,i) for each update at the primary and secondary steps.

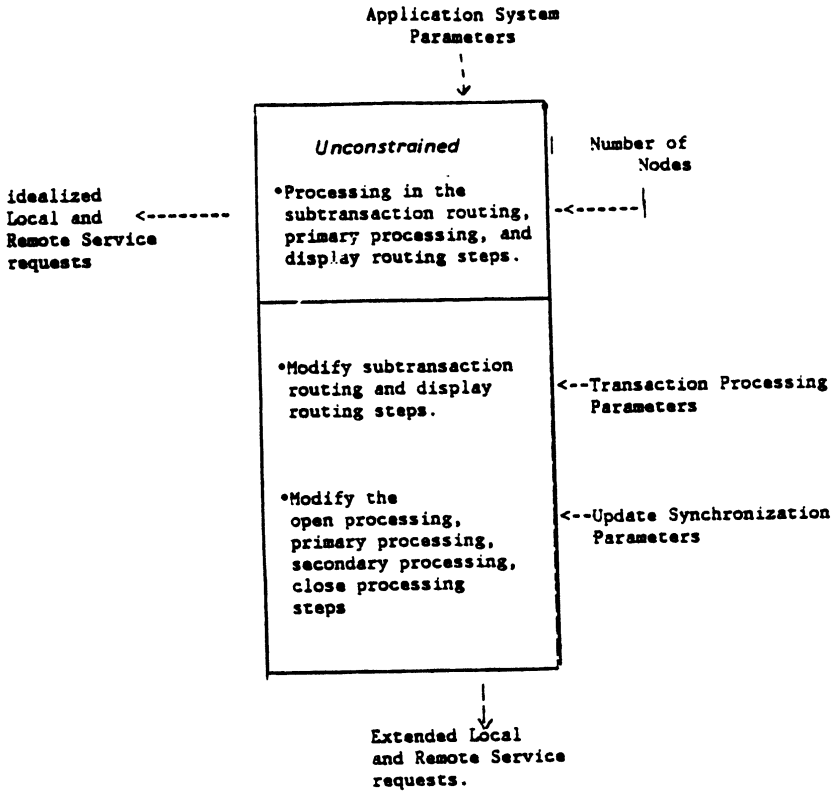
Query/Data Translation and Security: These issues basically involve reading of the control databases which may be centralized or distributed. The predictive model can be esaily extended to handle these issues by simply increasing the L(t,k,i) in the open processing and primary processing steps.

Consistency Intervals and Granularities: The locking granularity can be adjusted to a caorse (relation) level or fine (tuple) level by simply changing the expressions for L(t,k,i) and R(t,k,i,j) in the open and close processing steps. The consistency interval can also be adjusted by allowing the locking/unlocking at each read/write in the primary processing instead of the open and close processing steps.

Variation in Communication (Remote Processing) Cost: So far we have considered a communication cost based on remote services (messages) multiplied by the cost per remote service. The remote services are given by equations 4.32a through 4.32e. This cost can be modified to represent the volume transported by multiplying the remote services with the average message size. We will study this topic in next chapter.

In addition, the number of steps in this model are deliberately large so that the effect of modifications can be isolated. The analytical expressions are also not manipulated and reduced so that modifications and extensions can be made easily. The model is implemented as a well documented Fortran subroutine which can also be easily modified to reflect the changes. Thus an interested user can easily utilize this model to include the effect of logging, data/query translation, security checking and directory allocation.

FIGURE 4.6: EXTENSION OF THE PREDICTIVE MODEL



4.6 Discussion and Analysis of Objective Function

The objective function given by equation 4.1 can now be written as:

$$=\Sigma_t\Sigma_k\Sigma_i\Sigma_j T_a(t,k)[R_{np}(t,k,i,j)+R_{pp}(t,k,i,j)+R_{pd}(t,k,i,j)+R_{lk}(t,k,i,j)]\times C_R(i,j)+\Sigma_t\Sigma_k\Sigma_i T_a(t,k)[L_{pd}(t,k,i)+L_{lk}(t,k,i)]C_L(i)+\Sigma_i S(i)C_S(i)\ldots(4.40)$$

where R_{np}, R_{pp}, R_{pd} and R_{lk} are given by equations 4.32a through 5.32e, $L_{pd}(t,k,i)$ and $L_{lk}(t,k,i)$ are given by equations 4.30 and 4.31, S(i) is given by equation 4.25, $C_R(i,j)$, $C_L(i)$ and $C_S(i)$ are given costs and $T_a(t,k)$ is given arrival rate. The detailed analytical expressions for the local and remote service requests derived in the previous section allow us to measure the individual costs (storage, local processing, remote processing) and the total cost so that we can analyze the effect of transaction processing and update synchronization on program and data allocations. In addition, a comparison of the expressions for S, L and R obtained above with corresponding equations obtained from the uncon-

strained model (section 2.5) gives us additional insight into the combined effect of application system, transaction processing and update synchronization factors on program and data distribution. We first review the individual costs and then analyze the effect of transaction processing and update synchronization on total cost and program and data distributions.

i). STORAGE COST: The storage cost S is unaffected by the introduction of transaction processing and update synchronization since equation 2.2 and 4.30 are the same. This is because we have ignored the storage requirements of lock and stamps which can, in practice, increase the dataset size.

ii). LOCAL PROCESSING COST: The local processing activity (L) is unaffected by transaction processing options, however the update synchronization increases the local activity due to locking/unlocking. Recall that:

$$L(t,k,i) = L_{pd}(t,k,i) + L_{lk}(t,k,i).$$

Note that $L_{pd}(t,k,i)$, the local activity due to read/write shown by equation 4.30, is the same as L(t,k,i) given by equation 2.3 in chapter 2. This means that the local read/write activity is unaffected by the update synchronization, only an additional overhead, indicated by L_{lk} (equation 4.31), is introduced. Let us analyze L_{lk}. An examination of equation 4.31 shows that: a) $L_{lk}(t,k,i) = 0$ if $U_{cr} = 0$, i.e. if locks are core resident then the effect of locking on local processing can be ignored. The locks can reside in core if database is small, locking granularity is large or special locks are used (for example the "hole" lists <GARC79d>). b) The centralized lock management is cost efficient for cases when duplicate data exists. This is because in case of distributed lock management, the locks are set/reset at every node where the copy of the dataset resides, howewver in case of centralized lock management locks are set/reset at the central node, independent of the number of copies of dataset. Equation 5.31 can be used for quantitative

analysis. c) The overhead due to lock/unlock increases with the number of read/write activities.

iii) REMOTE PROCESSING COST: Recall that:

$$R(t,k,i,j)=R_{np}(t,k,i,j)+R_{pp}(t,k,i,j)+R_{pd}(t,k,i,j)+R_{lk}(t,k,i,j)$$

The node to program traffic (R_{np}) and the program to program traffic (R_{pp}) is influenced by transaction processing but are not affected by update synchronization. The program to dataset traffic (R_{pd}) and the lock/unlock remote traffic (R_{lk}) is not affected by the transaction processing extension, but is modified due to the update synchronization algorithms. Let us study the R_{np}, R_{pp}, R_{pd} and R_{lk}.

R_{np} - The Node To Program Remote Traffic: Equation 4.32a shows this traffic in terms of the routing graph Q_{rt}, the site selection matrix β_s and the application system parameters. It can be noted that the other transaction processing parameters (SRCH and Q_{da}) are not directly used in this expression. Instead these parameters are used to determine β_s and are thus implicitly included. Let us study the special case of R_{np} for a parallel routing graph by substituiting Q_{rt} = 1 in equation 4.32a, which yields:

$$R_{np}(t,k,i,j) = \delta(k,i)\{1+P_o(p)\}\beta_s(k,p,j)$$

Examination shows that this equation is same as equation 2.4a, the expression for R_{np} obtained from the unconstrained model. It is intuitively obvious since the unconstarined model assumed a parallel routing graph (see assumption 7, section 2.4). Now consider the situation for the serial routing graph by substituting Q_{rt}=0 in equation 4.32a:

$$R_{np}(t,k,i,j)=\delta(k,i)\{\beta_s(k,p0,j)+P_o(p)\beta_s(k,pn,j)\}$$

This equation shows clearly that in case of a serial routing graph all the input data is sent to the first site (p0) and all the output messages are sent from the final site (pn) to the arriving node k. It is being assumed that the originating node is also the destination

node, a generally true situation. However, this assumption can be relaxed easily.

The node to program traffic in the serial and parallel routing graph is the same if there is only one program site selected (p0=pn, i.e. single operation queries) or if the following equation holds:

$$\beta_s(k,p0,j)+P_o(p)\beta_s(k,pn,j) = \{1+P_o(p)\}\beta_s(k,p,j)$$

This equation can be used to minimize the node to program traffic by choosing the serial routing graph if the left hand side is greater than the right hand side and vice versa.

R_{pp} - The Program To Program Traffic: Equation 4.32b shows this traffic in terms of the routing graph Q_{rt}, the site selection matarix β_s and the application system parameters. This expression reduces to the unconstrained expression for R_{pp} (equation 2.4b) for parallel routing graphs, i.e. Q_{rt}=1. However, the expression is different in case of Q_{rt}=0, the serial routing graphs. Once again, the traffic due to the ring and parallel routing graphs is the same for single operation queries or if the following equation holds for each program p:

$$\beta_s(k,p+1,j)\}\{\sum_{p_2=p_{t1}}^{p_n} P_m(p,p2)+\sum_{p_2=p_{t1}}^{p_n} P_I(p2)+\sum_{p_2=1}^{p_n} P_o(p2)\}=\Sigma_{p1}\beta_s(k,p1,j)P_m(p,p1)$$

Once again, this equation can be used to minimize the program to program traffic.

R_{pd} - The Program to Dataset Traffic. An examination of equations 4.32c and 4.32d reveal that R_{pd} is influenced primarily by the location of the commit-processor: a) if a transaction is read only (P_w and P_c = 0), then the R_{pd} is not affected by the location of the commit-processor and b) the centralization of the commit-processor generally increases the remote traffic, unless a star network topology is used.

R_{lk} - The Remote Lock/Unlock Traffic: It can be seen from equation 4.40 that R_{pd} and R_{lk} are assigned same weight, i.e. a lock/unlock request is considered as a service request just like a program read/write, even though a program read/write transfers con-

siderably more data than a lock/unlock request. In next chapter we will calculate the remote processing cost based on information transported. Let us now consider the effect of update synchronization on R_{lk}: a) The remote traffic is greatly affected by the amount of piggybacking used to send lock/unlock requests along with the database read/write messages. If all the messages are piggybacked, then no additional remote processing cost is incurred. Thus, time stamped synchronization algorithms which usually send the time stamps along with the read/write messages are efficient. b) The cost of maintaining duplicate data is much greater for distributed lock management versus centralized lock management, for the reasons stated above (discussion of L_{lk}). c) The database read/write activities increase the remote traffic due to locking/unlocking.

iv). EFFECT OF TRANSACTION PROCESSING ON PROGRAM AND DATA ALLOCATION: Figure 4.7a shows how the storage, local processing, remote processing and total cost are affected by the increase in program duplication, when different transaction processing options are used. This diagram is based on a Fortran program which implements the predictive model. To avoid unnecessary complications, we assume that the data is completely replicated. We also assume that the programs occupy negligible amount of storage (a practical assumption in view of cheap storage and small amount of program storage needed as compared to data storage). Thus the storage and local processing costs are not affected by the program duplication, however, the communication cost decreases monotonically with increases in program duplication. The storage size is constant because we assume programs do not occupy any disk storage, the local processing is fixed because at one time only one copy of the program is in use and the communication costs drop due to the elimination of node to program traffic (there is no program to data traffic since every node has a copy of data). These observations are true for any transaction processing option (remote /local data access, direct/serial routing graphs). In

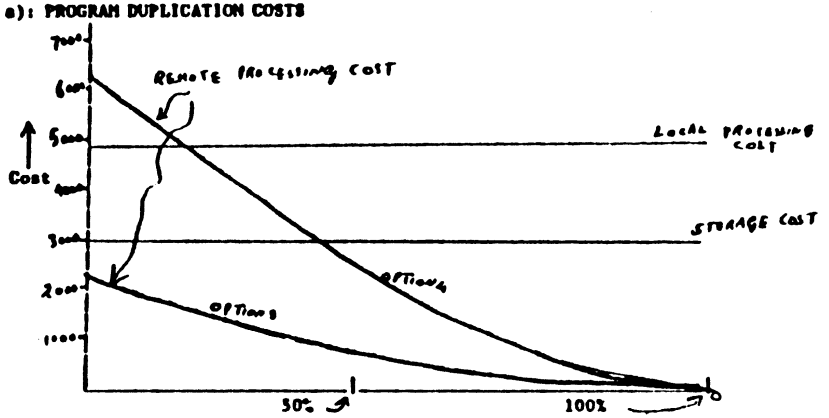
other words, the optimal program allocation is program replication and is indepedent of transaction processing options. Thus, the program allocation is trivial.

Now consider the influence of transaction processing on data duplication by assuming that programs are replicated and that there are no consistency relationships (see figure 4.7b). It can be seen that the storage cost is a monotonically increasing function of data duplication, the node processing cost is also a monotonically increasing function of data duplication and the remote processing cost is dependent on the following transaction processing configurations: a). Remote Data Access, Parallel Routing Graph: The data is accessed directly from the arriving node. The remote traffic consists of program to data traffic (R_{pd}), the other components (R_{np} and R_{pp}) are zero since all the programs are selected at the arriving node. b). Remote Data Access, Serial Routing Graph: The data is accessed directly from the arriving node and the remote traffic is same as in the Remote Data Access, Parallel routing algorithm case since all the programs exist at the arriving node, thus the serial and parallel routing graphs are same. c). Local Data Access, Parallel Routing Graph: The data is processed at the sites selected and the remote traffic primarily consists of R_{np} and R_{pp}. d). Local Data Access, Serial Operation Graph: This situation is the same as case 2 with the difference that the transaction is circulated as a ring. The remote traffic consists of R_{np} and R_{pp}.

Figure 4.7c illustrates the effect of transaction processing on optimal data assignments and costs by considering the allocation of 2 program, 3 dataset registeration system to 5 nodes. Case 1 basically illustrates the effect of data access mode (LDA versus RDA) on the program/data allocations by assuming replicated programs. The effect of routing graph is not clear since the two programs are performed at the same node. Case 2 shows the effect of routing graphs by assuming that the two programs are uniquely assigned to nodes 1 and 2 respectively.

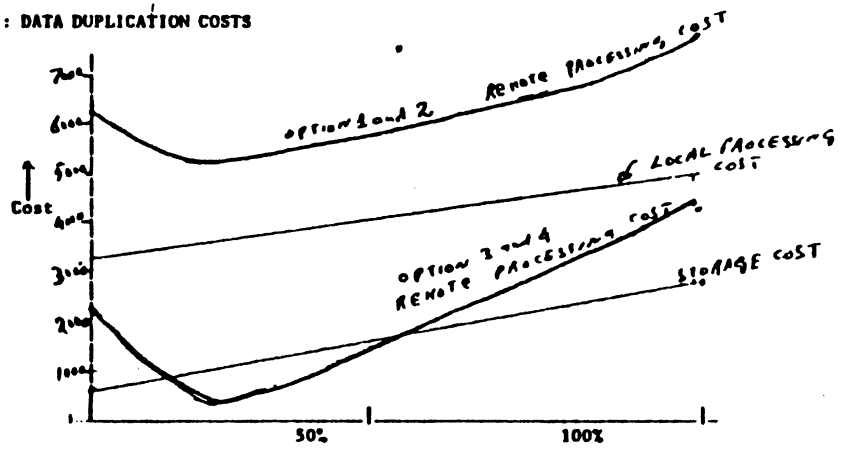
Figure 4.7 : EFFECT OF TRANSACTION PROCESSING ON TOTAL COST

a): PROGRAM DUPLICATION COSTS



Program Duplication ----->

b): DATA DUPLICATION COSTS



Data Duplication

NOTES:
Option 1 = Remote Data Access, parallel routing graph
Option 2 = Remote Data Access, serial routing graph
Option 3 = Local Data Access, serial routing graph
Option 4 = Local Data Access, parallel routing graph

c) EFFECT ON OPTIMAL DATA ALLOCATION

Case 1: Programs p1 and p2 are replicated

	Storage Cost	Local Proc. Cost	Remote Proc. Cost	Total Cost	Optimal Data Assignment $d_1 \mid d_2 \mid d_3$
Unconstrained	4800	4160	1480	10440	1011011111000010
LDA, Parallel	1800	3848	935	6583	000100000100001
LDA, Serial	1800	3848	935	6583	000100001000001
RDA, Parallel	4800	4160	1480	10440	101101111000010
RDA, Serial	4800	4160	1480	10440	101101111000010

Case 2: Program p1 assigned to node 1 and p2 assigned to node 2

	Storage Cost	Local Proc. Cost	Remote Proc. Cost	Total Cost	Optimal Assignment $d_1 \mid d_2 \mid d_3$
Unconstrained	1800	3848	880	6528	100000100001000
LDA, Parallel	1800	1768	100	3668	100001000000001
LDA, Serial	2400	1872	100	4372	100001010000001
RDA, Parallel	1800	3848	880	6528	100000100001000
RDA, Serial	1800	3848	1010	6658	100000100001000

NOTES:
- Two programs (p1 and p2) access three datasets (d1, d2, d3)
- p1 reads 1 input message, reads d1 60 times, generates 1 message and transfers a tuple to p2.
- p2 reads 1 input message, receives the tuple generated by p1, reads d2 80 times and updates d2 4 times. All the updates of d2 are consistency updates.
- The transaction arrival rates at nodes 1, 2, 3, 4 and 5 are 3, 2, 3, 4 and 1 respectively.
- The local storage cost is 6 cost units per 1000 bytes, the local processing cost is 2 cost units per local service and the remote processing cost is 5 cost units per remote service.
- The assignment 1 indicates allocated, 0 means not allocated.

In practice, this may happen when two nodes are best suited (fastest) to perform p1 and p2 or if the software duplication costs are large due to heterogenous nodes. It can be seen from figure 4.7c that the results are different for the two cases and that the optimal allocations as well as the costs are affected by the routing graphs.

Figure 4.7b shows that although the actual cost may differ for the four cases, the overall behviour of the remote processing (inter-node communication) cost is essentially the same for all these cases. This cost first decreases and then starts increasing with data duplication. The initial drop is due to the increase in the availability of local data, which makes internode communication unnecessary. However, as the data duplication keeps on increasing, more internode communication is needed for updating duplicate data, eventually resulting in a net increase in the remote processing cost. The objective function, as a result of the individual behaviours of the storage, local processing and remote processing costs, is a concave function of the data duplication.

v) EFFECT OF UPDATE SYNCHRONIZATION ON PROGRAM AND DATA ALLOCATION: Update synchronization algorithms do not directly influence program allocations, however the effect on data allocations is significant. The net effect of introducing update synchronization is that the cost of maintaining duplicate data increases dramatically due to the various synchronization activities. The actual cost depends on the read/write ratio and the update synchronization algorithm being used. Figure 4.8 shows the total cost (storage + local processing + remote processing) of a single program/single dataset transaction with 3% updates for the update synchronization algorithms discussed above. It can be seen from this figure that the optimal number of dataset copies decreases due to update synchronization (several algorithms yield single copy optimal allocations). However, the overall behaviour of the objective function is still concave.

Table 4.5 shows the combined effect of read/write ratios and update synchronization on optimal dataset allocations. To avoid unnecessary exaggeration of the remote processing cost due to locking/unlocking activity, the cost per remote lock/unlock has been set to 0.05 of the read/write cost by using a multiplier in the cost function. (The basis of this adjustment is the practical observation that in most systems a lock/unlock request ranges between 4 to 8 bytes and the read/write instructions transfer between 80 to 150 bytes.) It can be seen from table 4.6 that a) the optimal copies are the same for 0% and 3% update ratios, b) almost all the algorithms yield single copy allocations after 5% update ratio and c) all allocations, including the uconstrained, yield single dataset allocations for this example. We need to clarify the pessimistic and optimistic algorithm results. In case of pessimistic algorithms, ALL sites are locked so the effect of data duplication on locking/unlocking is minimal. The optimistic algorithms do not use any locking/unlocking technique so this algorithm is the same as the unconstrained algorithm (note that the results for optimistic and unconstrained are same in table 4.5). It should also be noted that in some cases, single copy allocations for same algorithm, yield different total costs because the single copy can be potentially allocated to any of the 5 nodes.

The results shown in figure 4.8 and table 4.5 are based on the analytical expressions derived in section 4.5 which show, quantitatively, the increase in cost due to data duplication, update synchronization algorithms, read/write ratio etc. Analytical expressions of this nature have not been reported in literature. Instead, results have been found through detailed simulation studies (see <MITS82d, BARB82d>). The cost increment due to synchronization of duplicate data is an important result since it affects the efficiency of the allocation algorithms. Simply stated, if data duplication is too expensive then only single copies

FIGURE 4.8 EFFECT OF UPDATE SYNCHRONIZATION ON TOTAL COST

NOTES:
.The transaction consists of 3% updates, 97% reads issued by program p to dataset d.
.The network consists of 5 fully connected homogeneous nodes.
.The cost per local service request is 2 (arbitrary cost units), 5 per remote service request and 6 per 1K of disk storage. The network is homogeneous from cost point of view.
.The cost per lock/unlock remote service request is .25

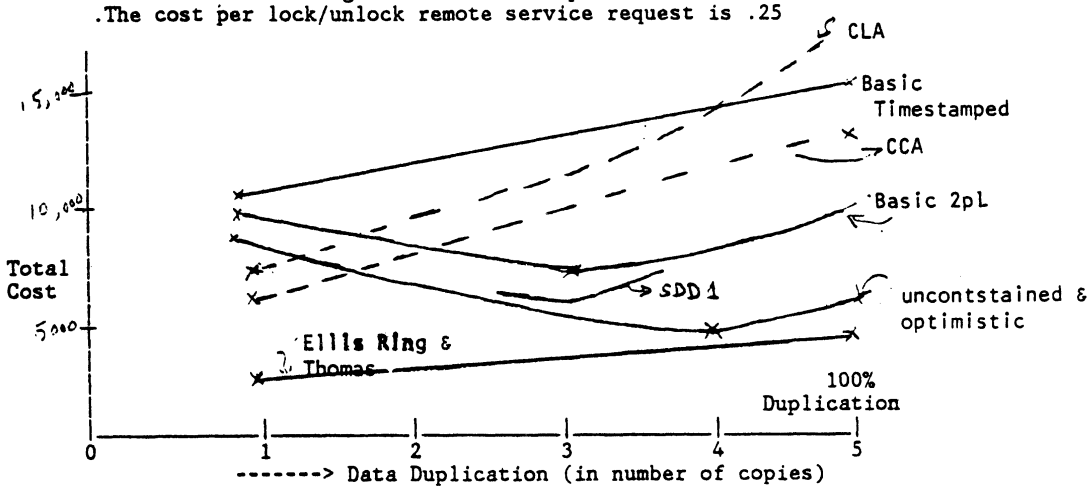


TABLE 4.5 EFFECT OF UPDATE SYNCHRONIZATION AND READ/WRITE RATIOS ON DATA ALLOCATIONS (NO. OF COPIES)

NOTE: The transaction and network are the same as in figure 5.4

	0% UPDATE		3% UPDATE		5% UPDATE		10% UPDATE	
	Optimal Copies	Optimal Cost	Optimal Copies	Optimal Cost	Optimal Copies	Optimal Cost	Optimal Copies	Optimal Cost
Uncon. traind	4	5,469	4	6,381	3	6,810	1	7,700
Basic 2PL	3	7,164	3	7,978	1	8,150	1	8,150
Time-stamp	1	10,383	1	10,719	1	10,525	1	10,525
Ellis	1	3,344	1	3,522	1	3,470	1	3,470
Thomas	1	3,354	1	3,422	1	3,380	1	3,380
CCA	1	5,848	1	6,004	1	5,900	1	5,900
CLA	1	7,164	1	7,978	1	8,150	1	8,150
SDD1	3	6,200	3	6,521	1	7,232	1	7,232
Pessi-mistic	4	7,809	4	8,981	3	9,410	1	10,330
Opti-mistic	4	5,469	4	6,381	3	6,810	1	7,700

of data can be allocated, which reduces the number of decisions to be made by the allocation algorithm.

4.7 Chapter Summary and Significance

In this chapter, the idealized support system is extended to introduce the data distribution strategies, the transaction processing options which include the transaction routing graphs and data access modes, and the update synchronization algorithms which include various centralized/distributed lock management and synchronization controls. Generalized transaction processing and update synchronization models are presented and parameters are introduced to so that a variety of real life systems can be represented. The predictive model is extended to show the effect of these parameters on the storage, local processing, remote processing and total cost through detailed analytical expressions. An analysis of the objective function yields two significant results: a) the actual total cost is affected by the transaction processing and update synchronization, however the total cost displays an overall concave behaviour, and b) update synchronization significantly increases the cost of data duplication which tends to decrease the number of dataset copies for optimal cost. These two results play key role in developing the generalized allocation algorithm.

CHAPTER 5
THE PHYSICAL RESOURCE EXTENSION

So far we have concentrated on calculating the total cost which consists of the storage cost, local processing cost and remote processing cost, in terms of the application systems, transaction processing and update synchronization parameters. This has been accomplished by first using an idealized support system based on explicitly stated assumptions (chapter2) which were removed later in chapter 4 to introduce the effect of transaction processing and update synchronization. In this chapter, the balance of the assumptions stated in chapter 2 are removed to calculate the communication traffic in terms of network topology, network control and routing (section 5.1), transaction availability in terms of program and dataset availability (section 5.2), cpu and disk activity (section 5.3), program duplication costs in distributed systems (section 5.4), and the transaction response time with queuing delays at local and network levels, locking conflicts and background workloads (section 5.5).

5.1 The Communication Traffic Calculations

A rough estimate of communication traffic is given by summing the remote service requests, i.e. $\Sigma_k \Sigma_i \Sigma_j R(t,k,i,j)$. However, R is independent of physical paths between nodes and was derived by assuming fully connected network with autonomous nodes (assumption 2, section 2.4). We wish to calculate the communication traffic by relaxing this assumption, which introduces the important issues of network toplogies,

network control and routing (section 5.1.1). Section 5.1.2 shows the main parameters introduced to represent the communication systems and the communication traffic is calculated in terms of the parameters in section 5.1.3.

5.1.1 Communication Systems in DDP - Background Information

The objective of a communication system is to allow exchange of information (messages) between remotely located devices. In distributed systems, the remotely located devices are computer systems (nodes) which are interconnected through communication *links*, also called channels. Exchange of messages in a DDP environment involves a large number of issues (see for example <KIMB75a> for a detailed discussion). Several views and classifications of communication systems in DDP exist. Probably the best known effort is the Distributed Reference Model proposed by the ISO/ANSI SPARC subcommittee-15 <BACH78d, DES78d, DES80d>. This model, shown in figure (5.1), views DDP systems as interconnections of "workstations" (nodes) where each workstation consists of layers and each layer performs well defined functions. Although the number of layers and the exact functions performed in layers is open to some debate (the original Reference Model had six layers), layered views of DDP systems has become an industrial standard. For example, Thurber <THUR80d>, Piatkrowski <PIAT80d> and Wecker <WECK79d> have used layered views to show the interrelationships between the Reference Model and available DDP support systems like IBM's SNA, Digital's DNA and Univac's DCA. The primary advantage of layered views is that the following two key concepts can be clearly defined:

.A *Protocol* is defined as the set of rules between two peer layers. For example, network protocols are the rules between the network control layers of the network.

. An *Interface* is defined as the set of rules between two adjacent layers at the same node.

Figure (5.2) illustrates the layers of the Reference Model by showing how a program p1 at node 1 can send a message 'HELLO' to a program p2 at node 2. As can be seen the higher level layers are concerned with the management of the interactions between data and programs. The lower level layers (transport, network, link and physical control) manage the transmission of information (messages) between nodes. Thus as we proceed from higher level layers to lower level layers, we go from more conceptual and application oriented issues to more physical and support system oriented issues. Specifically, the issues of network protocols, routing algorithms and network topologies, are concerned with the following layers of the Reference Model:

. TRANSPORT CONTROL LAYER which provides the transfer and management of message exchanges between application nodes by providing the addressing and flow control mechanisms.

. NETWORK CONTROL LAYER which provides the transfer and management of message exchanges between switching nodes by providing path selection and routing mechanisms.

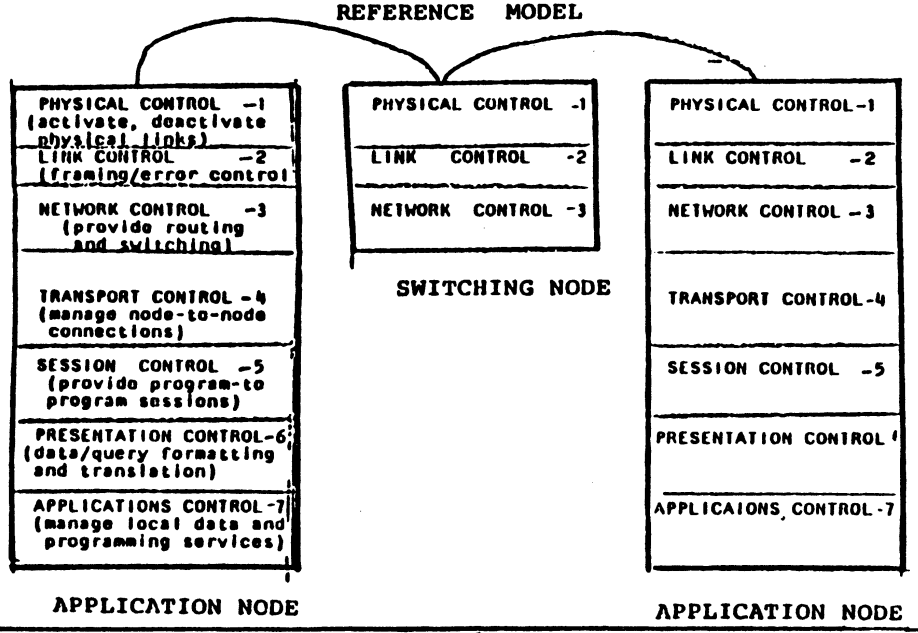
. LINK CONTROL LAYER which provides the transfer and management of message exchanges over single physical links by facilitating error control and checking.

. PHYSICAL CONTROL LAYER which provides for the physical, mechanical and electrical signal transfer.

For the purpose of this research we will only concentrate on the transport control and network control layers. The main issues in these layers are:

a). Addressing and Communications Control: Addressing provides the "mailboxes" for exchange of information between the nodes where applications resides and the intermediate communication controllers (switching nodes) which pass the messages between nodes. The network

FIG (5.1): THE ISO/ANSI DISTRIBUTED REFERENCE MODEL



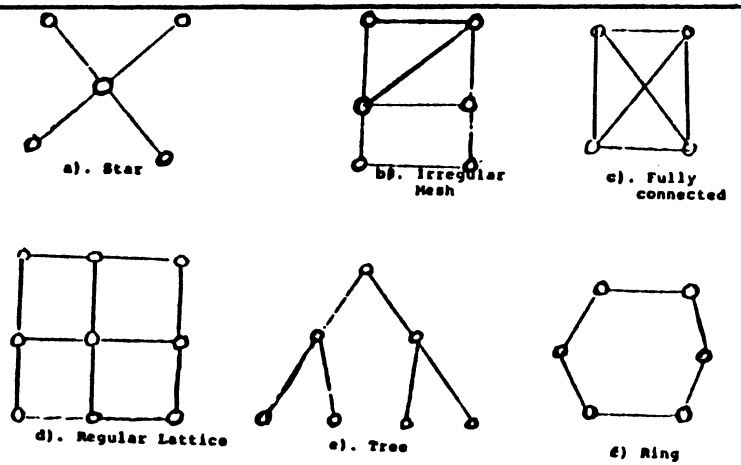
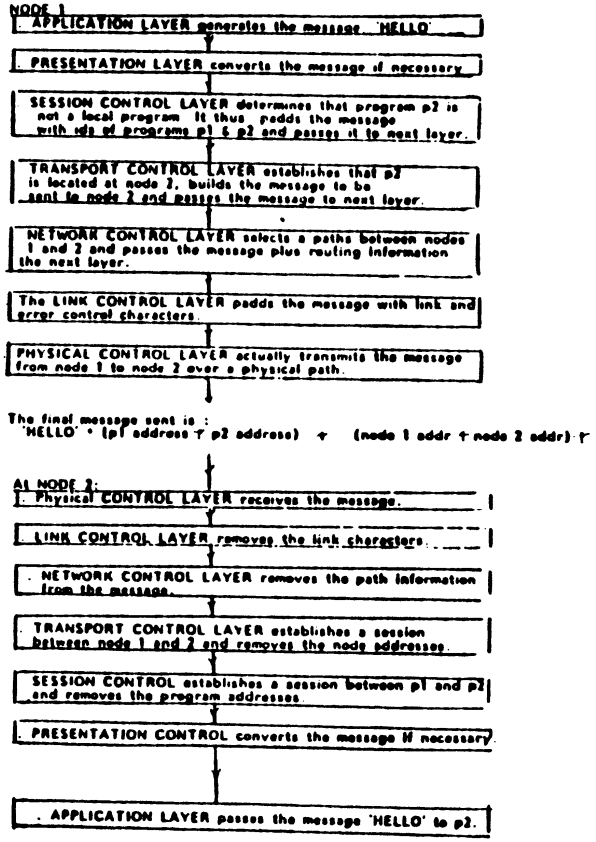


FIG (5.3) NETWORK TOPOLOGIES

FIG (5.2) EXAMPLE OF ANSI/REFERENCE MODEL

Send a message 'HELLO' from program p1 at node 1 to program p2 at node 2



addresses are kept in "network directories". Assignment of network addresses and allocation of network directories is a major issue in data communication design <HEBA77c, MART73a>. Different support systems have implemented different options. For example, IBM's SNA favours centralized network directories and 8 character addressing schemes. Communications control establishes the sessions between nodes and throttles the traffic. This activity is performed by transport as well as network control layers. Communication control can be centralized or distributed. As can be expected, different support systems choose different options <DAVI79a, DOLL78a, ABRA76a, KIMB78a>.

b). Routing & Switching: After the remote sessions have been established, the messages are routed from the sources to the sinks. The message can always follow a fixed path (fixed routing) or can dynamically select paths (adaptive routing) between sources and sinks to balance traffic or bypass unavailable paths. Different communication systems support different routing options: simpler communication systems are usually restricted to fixed routing while more sophisticated systems employ adaptive routing techniques. The path selection and routing is usually discussed in terms of circuit, message and packet switching. A circuit switching system basically allows a fixed path, a message switching system routes messages over different paths and a packet switching system decomposes the messages into "packets" which are then routed independently through various paths. These issues are primarily concerned with the network control layers. Several options and tradeoffs of routing, path selection and switching have been discussed in literature <WOO78d, WECK79d, WECK76d, THUR80d, SCHN79d>.

c). Network Topologies: Network topology is the arrangement and interconnection of network components. For the purpose of this research, we concentrate on two type of network components: the nodes and the communication links. Note that terminals, modems, communication controllers and switching nodes are not considered. Figure (5.3) shows a

variety of network topologies <COTT77a, DAVI79d, ITKO78a, CYPS78d>. These topologies are used to interconnect nodes over several hundred miles or in the same building (local area networks). For our purposes, we need to concentrate on the following major definitions. A network is viewed as a graph which consists of a set of 'nodes' or 'vertices', interconnected by 'arcs' or 'links'. The route over which messages are passed from one node to another is called a 'chain', and the number of links on a chain is called 'size' of chain. The chain which comes back to its starting node is called 'cycle' and the process is called 'folding'. A network is 'connected' if a chain exists between each pair of nodes. Network connectivity is calculated by using an 'adjacency' matrix which shows the number of lines from each node to all others (see <DAVI79d> for discussion).

5.1.2 The Communication System Parameters

The basic communication information can be provided in terms of three input parameters: $B_{tp}(i,j)$, B_{nc}, and $B_{rt}(x)$. The *$B_{tp}(i,j)$ parameter* shows the network topology of a distributed system: $B_{tp}(i,j)= z$, where z is the actual physical link that connects nodes i & j and $B_{tp}(i,j) = 0$ if no direct link exists between nodes i and j. Note that the B_{tp} matrix is similar to the adjacency matrix AD used very frequently in communication systems, where AD(i,j)=1 if a link exists between nodes i and j, 0 otherwise. The *B_{nc} parameter* shows if the network control is distributed(1) or centralized (0). The *$B_{rt}(x)$ parameter* shows the routing technique being used by the communication system: x = 1 shows the routing probability at the primary paths and x = 2 shows the routing probability at the secondary paths, where a routing probability is the probability that a message will traverse a given route. A primary path is the shortest path between two nodes. All other (alternate) paths are considered secondary. The probabilities of routing can be

usually measured from the network statistics that are kept by most communication systems. Note that this parameter also implicitly defines the fixed or adaptive routing. For example, in a fixed routing system, $B_{rt}(1)=1$ and $B_{rt}(2)=0$ while in adaptive routing systems, $B_{rt}(1) < 1$ and $B_{rt}(2) > 0$. For the purpose of analysis, we will stay with only two routing probabilities. As can be seen, this parameter can be easily extended to generalized routing systems. For example, we can let $B_{rt}(n)$ to show the routing probability over chain size n. It is expected that B_{rt} is measured by gathering statistics on a routing algorithm.

From these three parameters, the following "path" matrix can be constructed:

PATH(i,j,z)= the probability that a message exchanged between nodes i and j traverses link z.

The path matrix establishes a relationship between the logical internode path and the physical paths (links) that are actually traversed by the internode messages. This matrix can represent a variety of topologies, centralized/distributed network controls and adaptive as well as fixed routing and will be used heavily to compute the communication traffic. Fig (5.4) illustrates the relationships between an actual network and the parameters introduced here by showing a 4 node, 5 link network (fig 5.4a), the parameters B_{tp}, B_{nc} and B_{rt} showing the topology, network control and routing frequencies of the network (fig 5.4b) and the resulting PATH matrix (fig 5.4c).

The following algorithm is used to compute the PATH matrix:

- Compute the adjacency matrix AD from the network topology matrix B_{tp} by using the formula: AD(i,j) =1 if $B_{tp}(i,j) \neq 0$, 0 otherwise.
- Compute the chains of size n of the network by using the formula:

  AD^n = the matrix showing the n chain connections

  We employ a matrix multiplication routine to compute AD^n. However, this matrix multiplication must eliminate folding, which shows the

edges of sizes < n in the AD^n matrix. In practice, the edges are calculated by using "labelling algorithms" (see <DAVI79a>).

- Compute the matrix PAR(i,j,e,n,m) which shows the mth link on the nth path of chain length e, between nodes i and j. Several parallel paths (r1, r2,...rN) of chain size e can exist between nodes i and j of an N node network. Each path ri of chain size e consists of e links. Heuristics are employed to produce PAR.
- Compute the PATH matrix from the PAR matrix and the B_{nc} and B_{rt} input parameters. This computation involves the following steps:

  .Eliminate the paths from the PAR matrix which do not conform to the network control specification B_{nc}. For example, if B_{nc} indicates central control, then no paths of chain size e=1 exist between nodes i and j for i or j $\neq$ z, where z is the central node. This means that in centrally controlled networks like IBM's SNA, all the messages go through the central node z; thus a "logical" star network topology is enforced in which all the primary paths (paths with chain size e=1) must involve z.

  .Determine the primary path between nodes i and j which is the path ri with chain size e=1. All other paths are secondary. Thus in a star network topology, all the primary paths involve the central node z and all secondary paths have a chain size e = 2.

  .Compute the PATH(i,j,z) by using PAR(i,j,e,n,m) matrix and routing frequency, $B_{rt}(1)$ and $B_{rt}(2)$, for primary and secondary paths. For example, if z = PAR(i,j,1,1,1) then PATH(i,j,z) = $B_{rt}(1)$, and for all other links z between i and j, PATH(i,j,z) = $B_{rt}(2)$.

It should be noted that calculation of the PATH matrix is done only once in the program and data allocation algorithm and can be performed in the initialization step of the algorithm.

5.1.3 Calculation of Communication Traffic and Cost

The path matrix can be used to calculate M(t,k,z), the no. of messages handled by link z when transaction t arrives at node k, by using the following equation:

$$M(t,k,z) = \Sigma_i \Sigma_j R(t,k,i,j)\ PATH(i,j,z) \quad \ldots\ldots\ldots (5.1)$$

where R(t,k,i,j) is the number of remote service requests exchanged between nodes i and j when transaction t arrives at node k. The analytical expression for R have been derived in chapter 4 (see equation 4.32). Also, $\Sigma_t \Sigma_k M(t,k,z)$ gives the total number of messages handled by link z and $\Sigma_t \Sigma_k \Sigma_z M(t,k,z)$ gives the total communication traffic. From this communication traffic, we can calculate the communication cost, which exists in the following three forms:

i) PER MESSAGE COMMUNICATIONS COST: This cost, used sometimes in message and packet switching systems, is based on a per message (packet) charge serviced by the network. This cost can be computed straightforwardly by using the formula:

$$\text{communication cost} = \Sigma_t \Sigma_z C(z) \Sigma_k M(t,k,z) \quad \ldots\ldots\ldots\ldots (5.5)$$

where C(z) is the cost per communication message serviced by link z. In case of packet switching systems, where the messages are broken or blocked into packets, we can use the above formula by using the "packetting factor" F1:

$$\text{communication cost} = F1 \times \Sigma_t \Sigma_z C(z) \Sigma_k M(t,k,z) \quad \ldots\ldots\ldots (5.6)$$

ii) COMMUNICATION VOLUME COST: This cost is based on the volume of data transported over the communication links. In order to compute this cost, we need to calculate the communication transport volume CTV(i,j) exchanged between nodes i and j:

$$CTV(i,j) = \Sigma_t \Sigma_k \{R_{np}(t,k,i,j)V_{np} + R_{pp}(t,k,i,j)V_{pp} + R_{pd}(t,k,i,j)V_{pd} + R_{lk}(t,k,i,j)V_{lk}\}$$

where R_{np}, R_{pp}, R_{pd} and R_{lk} are the number of remote messages exchanged between nodes i and j when transaction t arrives at k, and are given by equations 4.32a through 4.32d. V_{np}, V_{pp}, V_{pd} and V_{lk} are the corresponding average transport volumes, which can be estimated from the application type and the communication system being used. We allow V_{np}, V_{pp}, V_{pd} and V_{lk} as input parameters. Once CTV has been determined, we can now calculate the communication cost for the volume transported between nodes:

$$\text{communication cost} = \Sigma_z\ C2(z)\ \Sigma_i\ \Sigma_j\ CTV(i,j)\ PATH(i,j,z) \quad \ldots\ldots (5.7)$$

where C2(z) is the cost per unit of data transported over link z and PATH has been calculated above.

iii) FLAT COMMUNICATION COSTS: In some systems, the communication cost is based on a monthly flat basis or a connect-time basis (see <MAHM77c>). In these cases, the service requests R and the PATH matrix are not used.

$$\text{Communication Cost} = C3 \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots (5.8)$$

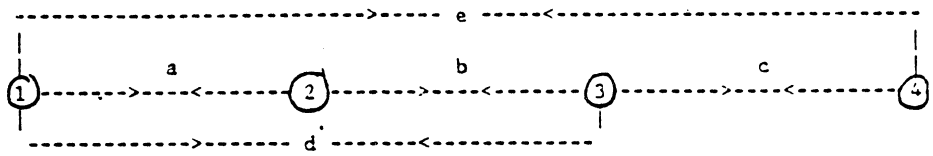
where C3 is the flat communication cost.

Our model allows a user to select one of the three cost functions and utilizes the following algorithm to calculate the total communication traffic in a network:

1. In case of flat communication cost, compute the cost directly from equation 5.8 and quit, else proceed.
2. Calculate remote service requests, R, based on the data/program allocations, application description and synchronization and query processing options.
3. Calculate the PATH matrix from the input parameters B_{tp}, B_{nc}, B_{rt}.
4. If the transport volume is to be used, then recieve V_{np}, V_{pp}, V_{pd} and V_{lk}, or use the default values.
5. Compute the communication cost by using equations 5.5, 5.6

Figure 5.4: Network Communications and Availability

a. The Network = Nodes 1, 2, 3, 4; Links a, b, c, d, e



b. The link matrices, routing frequencies and Network Control

B_{EP} =

i \ j	1	2	3	4
1	0	a	d	e
2	a	0	b	0
3	d	b	0	c
4	e	0	c	0

$B_{rt}(1)$ ↓ primary frequency

B_{nc} | secondary frequency → $B_{rt}(2)$

q	.9	.1

c. The path matrix

PATH (i, j, z) =

z \ ij	11	12	13	14	21	22	23	24	31	32	33	34	41	42	43	44
a	0	.9	.1	.1	.9	0	.2	.2	.1	.2	0	.1	.1	.2	.1	0
b	0	.2	.1	.1	.2	0	.9	.9	.1	.9	0	.1	.1	.9	.1	0
c	0	.1	.1	.2	.1	0	.1	1.0	.1	.1	0	.9	.1	1.0	.9	0
d	0	.1	.9	.1	.1	0	.1	.1	.9	.1	0	.1	.9	.1	.1	0
e	0	.1	.1	.9	.1	0	.1	.1	.1	.1	0	.1	.1	.1	0	0

d. Parallel Paths; link & path availabilities

- Paths between nodes 1 & 2 = a, (d, b), (e, c, b)
- Let availability of all links = .9
- $\xi 1$ = availability of path 1 = reliability of link a = .9
  $\xi 2$ = availability of path 2 = reliability of links (d, b) = .9 x .9 = .81
  $\xi 3$ = availability of path 3 = reliability of links (e, c, b) = .9 x .9 x .9 = .729

$A(1, 2)$ = availability of path between nodes 1 and 2
= 1-(1 - $\xi 1$) (1 - $\xi 2$) (1 - $\xi 3$) = 1-(1 - .9) (1 - .81) (1 - .729)
= 1 - .005149 = .9948

or 5.7 depending on the communication system being employed.

5.2 Transaction Availability

The availability of a transaction, given by the percentage of time a transaction executes properly, is an important performance measure in distributed systems, since data duplication can increase the availability of transactions which access the duplicated data. Availability of a system can be expressed as:

MTBF / (MTBF + MTTR)

where MTBF = mean time between failures and MTTR is the mean time to repair the system. For a transaction t which accesses dataset d through program p, a "failure" can be due to any one of the following reasons: a) the program p may not execute properly due to sofware bugs, b) the node where p resides may be unavailable, c) all the paths to the nodes where p resides may be unavailable, d) the nodes where d resides may be unavailable and e) all the paths from program p to dataset d may be unavailable. So far we have been assuming that all the software, nodes and links are 100% available. If we remove the assumption of 100% link availability, then we need to calculate the availability of a transaction in terms of the network availability and program and data allocations. (Note that the assumptions of "bug free" software and 100% node availability have not been relaxed.) Thus we need to calculate:

$$A_t(t,k) = \Pi_p \Pi_d A_p(k,p) A_d(p,d) \quad \ldots\ldots\ldots\ldots\ldots\ldots \quad (5.10)$$

where:

$A_t(t,k)$ = availabilty of transaction t when it arrives at node k

$A_p(k,p)$= availability of program p, utilized by t, from node k

$A_d(p,d)$ = availabilty of dataset d from program p

In order to calculate A_p and A_d, we need to know the allocation of program p and dataset d and the availability of paths between affected nodes. Mahmood & Riordon <MAHM77c> have given the expressions for file

availability from a given node. We generalize their expressions to determine A_p:

$$A_p(k,p) = 1 - \Pi_i[1-r(k,i)\beta(p,i)] \quad \ldots\ldots\ldots\ldots\ldots\ldots (5.11)$$

where r(k,i)= availability of the paths between nodes k and i and β is the program allocation variable. To calculate A_d, let j be the node where p is selected. Then:

$$A_d(p,d) = A_d(j,d) = 1 - \Pi_i[1-r(j,i)\alpha(d,i)] \quad \ldots\ldots\ldots (5.12)$$

where α is the dataset allocation variable. The major assumption in these calculations is that the distributed transaction manager has the "intelligence" to select the locations for programs and data with valid paths. To calculate r(k,i), we assume that n parallel paths exist between k & i. Thus we can use the following standard availability theory expression (see for example <MODER78f>):

$$r(k,i) = 1 - (1-\xi 1)(1-\xi 2),,,,,,,(1-\xi n) \quad \ldots\ldots\ldots (5.13)$$

where ξn = availability of parallel path n between nodes k and i. However, each parallel path can have e links, where e is the edge size of the path. To compute ξn, we need to compute the parallel paths between k and i and compute ξ for each path by using the following formula:

$$\xi n = A_a(z1) \times A_a(z2) \ ,,,,, \ A_a(z_e) \quad \ldots\ldots\ldots\ldots\ldots (5.14)$$

where $A_a(z)$ is the availaibility of link z, an input parameter. Equation 5.14 assumes that the parallel path n is of edge size e. Fig (5.4d) illustrates the parallel paths, the ξs and the corresponding r(k,i) for the 4 node, 5 link network shown in fig (5.4a). The overall procedure for calculating transaction availability is:

1. Compute the matrix PAR(i,j,e,n,m) as described in the PATH matrix calculations. Note that PAR does not have to be recalculated.
2. Compute the availabilities ξ of each parallel path by using the input parameter $A_a(z)$in equation 5.14.
3. Compute r(k,i) from equation (5.13)
4. Compute A_p and A_d from equation (5.12) and (5.11).

5. Calculate A_t by using equation (5.10).

The assumption of reliable nodes can be easily relaxed by using the following expression for equation 5.14:

$$\xi n = A_L(k) \times A_L(i) \times A_a(z1) \times A_a(z2),,,,A_a(z_e)$$

where $A_L(k)$ shows the availability of node k. Note that this expression only includes the availabilities of the source and sink nodes i and k. This is a realistic situation when the communication controllers house all the communication software and are independent of nodes where data resides.

The availability of transaction t when it arrives at node k, A(t,k), will be used in next chapter to study the availability constraints imposed by the user:

$$\Sigma_k \, T_a(t,k) \, A(t,k) \, / \Sigma_k \, T_a(t,k) \geq AV(t)$$

where $T_a(t,k)$ is the arrival rate of transaction t at node k and AV(t) is a user specified limit.

5.3 Disk, CPU Utilization and Local Database Design

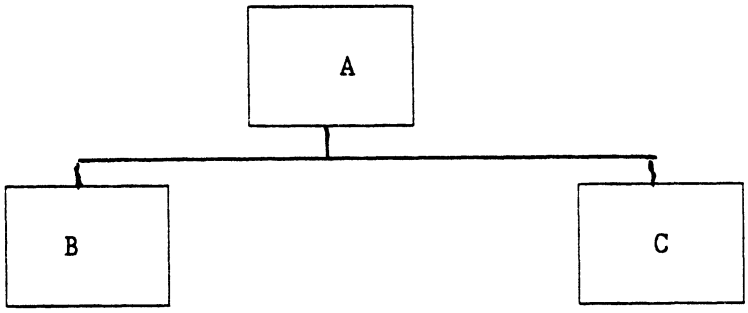
From the dependent variables calculated so far (local service requests L, remote service requests R, communication messages M etc), disk and cpu utlization can be approximated. For example, L(t,k,i) roughly represents the number of disk i/o at node i due to arrival of a transaction at node k, for a simple sequential disk file at node i. For more general cases the disk i/o at node i, $X_{disk}(i)$, can be estimated by:

$$X_{disk}(i) = \Sigma_t \, \Sigma_k f1(i) L(t,k,i) \quad \ldots\ldots\ldots\ldots\ldots\ldots (5.15)$$

where f1(i) is a conversion factor which shows the average no. of disk i/o issued at node i per local service request at node i (read/write). For simple sequential files the buffering and blocking can decrease the number of physical i/o issued per service request, thus f1(i)=1 for all i Ξ N represents a worst-case estimate in such situations. Note that f1(i) takes into account the local database design at node i and thus

provides the basic linkage between global database design, which consists of database partitioning and allocation, and local database design. For general database design problems, estimation of f1 and X_{disk} is difficult because it involves the following steps: a) develop a global schema of the database which shows a node-independent logical view of the data, b) estimate $P_r(p,d)$ and $P_w(p,d)$, the number of reads and writes issued from program p to dataset d, for the given global schema, c) compute the local service requests L(t,k,i) for each data allocation and perform the optimal data allocations to minimize total cost, d) develop the local schemas at each node based on the optimal data allocations, e) recompute P_r and P_w, to obtain the revised values P'_r and P'_w, based on local schema and compute the revised local service requests L'(t,k,i) at each node and f) now convert L'(t,k,i) into disk i/o by using equation 5.15. The following simple example illustrates this procedure.

Consider the example of records A, B and C that are arranged in the following hierarchical global schema (this example is from <TEOR82e>):



Let us assume that there are 100 occurrences of record type A, 200 occurrences of record type B per A, and 1000 occurrences of record type C per A. We assume that each record type can be viewed as a separate dataset. Thus d1= all occurrences of A, d2= all occurrences of B, and d3 = all occurrences of C. Consider the following programs of an application system:

. p1 finds random occurrence of record A

. p2 finds random occurrence of record B

. p3 finds random occurrence of record C

. p4 finds all A's

. p5 finds all B's associated with a given A

. p6 finds all C's associated with a given B

. p7 finds all A's associated with a given C

We assume that all the programs are activated by a single transaction t. Table 5.1a describes this schema and the relationships between records. Table 5.1b shows the $P_r(p,d)$, the number of reads issued from program p to dataset d for the global schema, assuming that record type A is the main entry point and only sequential accesses are allowed between record occurrences. Now consider a two node network with no locking/unlocking. For each allocation of d1, d2 and d3 to the nodes, we can calculate L(t,k,i) in terms of P_r, and other parameters, by using equations 4.29 and 4.30. Let us assume that d1 and d2 are allocated to node n1 and d3 is allocated to node n2 for minimal cost. Table 5.1c shows the optimal local service requests L and table 5.1d shows the optimal local schema. Since d1, d2 and d3 are allocated uniquely, the local service requests at any node are independent of the arrival node because processing for d1 and d2 will always take place at node n1 and processing for d3 will always be performed at n2. Thus the values shown in table 5.1c are independent of the arrival node. For the simple read-only transaction being considered, the LSR issued at node i due to program pn, Ln(i), is given by the following simplified version of equation (4.30):

$$Ln(i) = \Sigma_d P_r(pn,d)\alpha(d,i)$$

where $\alpha(d,i)=1$ if dataset d is allocated to node i, 0 otherwise. Let us now turn to the local database design at nodes n1 and n2. We assume that at node n1, record A is the only entry point and at node n2, every record of C can be directly accessed. Due to this, the revised values of P'_r and L' are obtained and shown in table 5.1e. Note that only $P'_r(p3,d1)$ differs from $P_r(p3,d1)$ because now all records of d3 can be randomly ac-

TABLE (5.1) : EXAMPLE OF GLOBAL TO LOCAL DATABASE CONVERSION

a). Global Schema Information:
Record type A is the parent and record type B and C are the children. Each record type can be viewed as a separate dataset. Thus d1= all occurrences of A, d2= all occurrences of B, and d3 = all occurrences of C. Record type A is the main entry point and only sequential accesses are allowed between record occurrences.

b). The Program reads, $P_r(p,d)$ for Global Schema:

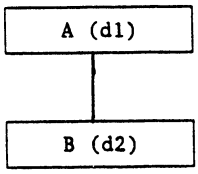
	d1	d2	d3
p1	50	0	0
p2	50	100	0
p3	50	0	500
p4	100	0	0
p5	0	200	0
p6	1	0	1000
p7	1	0	0

c). Local Service Requests for the Optimal Allocation:
Let Ln(i) = the number of LSR executed at node i due to program pn:

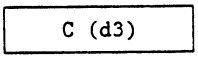
i →	1	2
L1	50	0
L2	150	0
L3	50	500
L4	100	0
L5	200	0
L6	1	1000
L7	1	0

Total Local service requests:
L(k,1) = L1(1) + L2(1) + L3(1) + . . . + L7(1) = 552 for all k
L(k,2) = L1(2) + L2(2) + L3(2) + . . . + L7(2) = 1500 for all k

d). The Optimal Local Schema:
Node n1



Node n2



e). The Revised Reads $P'_r(p,d)$=

p \ d	d1	d2	d3
p1	50	0	0
p2	50	100	0
p3	0 *	0	500
p4	100	0	0
p5	0	200	0
p6	1	0	1000
p7	1	0	0

The revised local service requests are:
L'(k,1) = 502
L'(k,2) = 1500

f). The Disk I/O:
Given f1(1) = 1, f1(2) = 0.8
$X_{disk}(1)$ = 502
$X_{disk}(2)$ = 1200

cessed. For purpose of analysis, we assume that f1(1) =1 and f1(2) = 0.8. Then equation 5.15 yields the number of disk i/o shown in table 5.1f, based on the values of L' shown in table 5.1e.

A variety of techniques can be used to measure or estimate f1(i). In some cases f1(i) can be estimated directly. For example, Lavenberg and Schneider <LAVEN76c> give some examples of the design factors that can be used in modelling IMS databases.

The number of cpu instructions executed at node i, $X_c(i)$ can be roughly estimated given by:

$$X_{cpu}(i) = \Sigma_t \Sigma_k \, f2 \, L(t,k,i) + \Sigma_t \Sigma_k \Sigma_j \, f3 \, R(t,k,i,j) \quad .....(5.16)$$

where f2 and f3 are the cpu instructions executed per local and remote service requests, respectively. The expression for X_{cpu} is based on the observation that in most business applications, the number of cpu instructions executed by a transaction are primarily due to local and remote i/o. Roughly, f2= 3000 in IBM's MVS operating system and f3= 5000 in IBM's SNA. This observation is utilized in several computer system evaluation tools for example the BEST1 system marketed by BGS Corportation.

5.4 Program Duplication Cost

In the generalized program and data allocation problem, the program duplication cost is included as a constraint which limits the number of program copies in the network. It is important to include the program duplication constraint for several reasons. First, a tradeoff exists between program duplication and communication costs. For example, if all the programs are duplicated, then the node to program communication cost is minimal but program duplication cost is high. Second, for several real life situations where data can be only accessed by local programs, program duplication cost can dictate the data allocation decision. In such systems, program duplication cost is the primary

binding constraint on data allocation. Finally, from a management point of view, the decision of duplicating programs is much more costly than duplication of data since the storage costs continue to decrease but the program duplication costs continue to rise in heterogeneous systems.

So far we have assumed that all the nodes are homogeneous from the software point of view, i.e. the software can be duplicated at various nodes without any significant additional cost. However, in practice, software written for node i needs to be converted for node j because of: a) different versions and releases of software at various nodes, b) different programming languages supported, c) differences due to database management systems for example programs written for IMS virtually need to be rewritten for CICS and d) special features and restrictions of nodes. The total cost of developing P programs on an N node network can be represented as:

$$SCOST = \Sigma_p \Sigma_i \; C_{ap}(p,i) \; \beta(p,i) \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots (5.20)$$

where $C_{ap}(p,i)$ is the cost of assigning program p to node i and $\beta(p,i)$ =1 if program p is assigned to node i. $C_{ap}(p,i)$ is given by:

$$C_{ap}(p,i) = \Sigma_s \; C_{dp}(p,s) \; h(s,i) \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots (5.21)$$

where $C_{dp}(p,s)$ is the cost of developing program in language s and h(s,i) = 1 if s is the primary, or desirable, language at node i, 0 otherwise. $C_{dp}(p,s) = 0$ if program p already exists in language s. We need to estimate the software development cost C_{dp}.

It is natural to think that cost of software development would be proportional to effort exerted in building the software. Thus if effort was known, then cost can be determined. Most of the work in software cost estimation has been, consequently, directed towards estimation of effort. In the beginning, the effort was estimated by dividing the size of software by the productivity rate (instructions generated per man month). Brooks <BROOK71e> showed that this was too simplistic and usually wrong. Since then several studies have been conducted to establish relationships between the cost of software development and software at-

tributes (size of software, complexity, language used, programmer experience, etc.). Some of these studies have suggested as many as 91 contributing factors to software costs. Basically the following two type of approaches are adopted: 1). Empirical studies involving study of empirical data and development of cost expressions. Sometime, theoretical justification is sought <WALST77e, BELA76e, PUTN76e, PUTN78e>. 2). Theoretical investigations involving the theoretical considerations in software development. The pioneer in this field is Halstead <HALS77e> who has introduced software science. At present no universally acceptable software cost estimation procedure exists.

In this thesis, the macro estimation procedure proposed and refined later by Putnam (PUTN76e, PUTN78e>, is used. This procedure is chosen since it considers the total life cycle cost of developing and maintaining software, and is based on very few parameters. The estimation is based on the work by <NORD70e>, who derived expressions for life cycle costs of R & D projects. Putnam has shown that the software life cycle surprisingly well fits the expressions derived by Nordon. Based on this observation, the following expression is derived:

$$ss = c^{(1/3)} \times e^{(4/3)} \times td$$

where:

ss= Size of software in instructions

c = Productivity constant depending on difficulty level etc. Putnam describes this constant in detail.

e = Total effort spent in man years

td = Development time allowed

The size of system (ss) and the constant c can be estimated, in terms of language s, and the development time is a given management constraint. Thus k can be calculated in terms of s. Once the total effort in man years is known, then the total cost of development and maintenance can be calculated by multiplying k with cost per man year. Thus the cost of developing and maintaining program p in language s, is given by:

$$C_{dp}(p,s) = y \left[\frac{ss(p,s)^3}{c(s)^3\, td^3} \right]^{1/4} \qquad \text{.....(5.25)}$$

where y = cost per manyear, ss(p,s)= Size of program p in language s, c(s) = productivity in language s, and td = Development time allowed.

5.5 Response Time Calculations

The response time of a transaction when it arrives at node k, RT(t,k), is the total time elapsed between the submission of the transaction at node k and the display of the results at k. The average response time RT'(t) of transaction t is given by:

$$RT'(t) = \{\Sigma_k\, T_a(t,k) \times RT(t,k)\} / \Sigma_k\, T_a(t,k)$$

where $T_a(t,k)$ is the mean arrival rate of transaction t at node k. Since $T_a(t,k)$ is an input parameter which can be specified for each incoming transaction t, we focus on the calculation of RT(t,k) for each transaction which, in a realistic DDP network, is an extremely complex task due to the combined effect of program and data allocations, application characteristics, network topologies, query processing options, update synchronization algorithms, queuing at several levels and synchronization lockouts. By using the predictive model developed in previous chapters, we can derive the expression for RT by using the following expression (see figure 5.5 for illustration):

$$RT(t,k) = t1(t,k)+t2(t,k)+t3(t,k)+t4(t,k)+t5(t,k)+t6(t,k)+t7(t,k)$$

where:

- t1(t,k) = time spent in interpreting and analyzing the arriving transaction. This time is usually spent on the arriving node and consists of reading the transaction and network directories to determine the transaction graphs and operating policies.

- t2(t,k) = time spent in routing the subtransaction to the node where the transaction will execute. This time may involve communication transmission time and queuing delays at the communication links if the transaction execution node is not same as the transaction arrival node.
- t3(t,k) = time spent during the open processing which involves resource allocation and acquisition of locks. As explained in chapter 4, this time may consist of requests to the lock/stamp manager which grants or rejects the lock request. Thus this time may consist of the communication time between the program execution node and the lock manager node, the local processing time at the lock manager node and queuing at the lock tables due to device busy or lock conflicts.
- t4(t,k) = time spent during the primary processing of the request which consists of request translation, data access, device busy queuing and data translation. If the data accessed resides at a node other than the program execution node, then the communication delays are added to this time.
- t5(t,k) = time spent during the secondary (commit) processing in which all the duplicate and consistency related data is updated. Since this step involves sending the commit processing request to a commit processor which performs the actual updates, t5(t,k) may involve communication delays and node processing delays.
- t6(t,k) = time spent during the close processing which frees all the locks aquired in the open processing step. Once again, this step may involve local as well as remote processing, depending on the location of the lock manager and lock tables.
- t7(t,k) = time spent during the display (results) routing step in which the final results are routed back to the resultant node. This time usually involves communication delay.

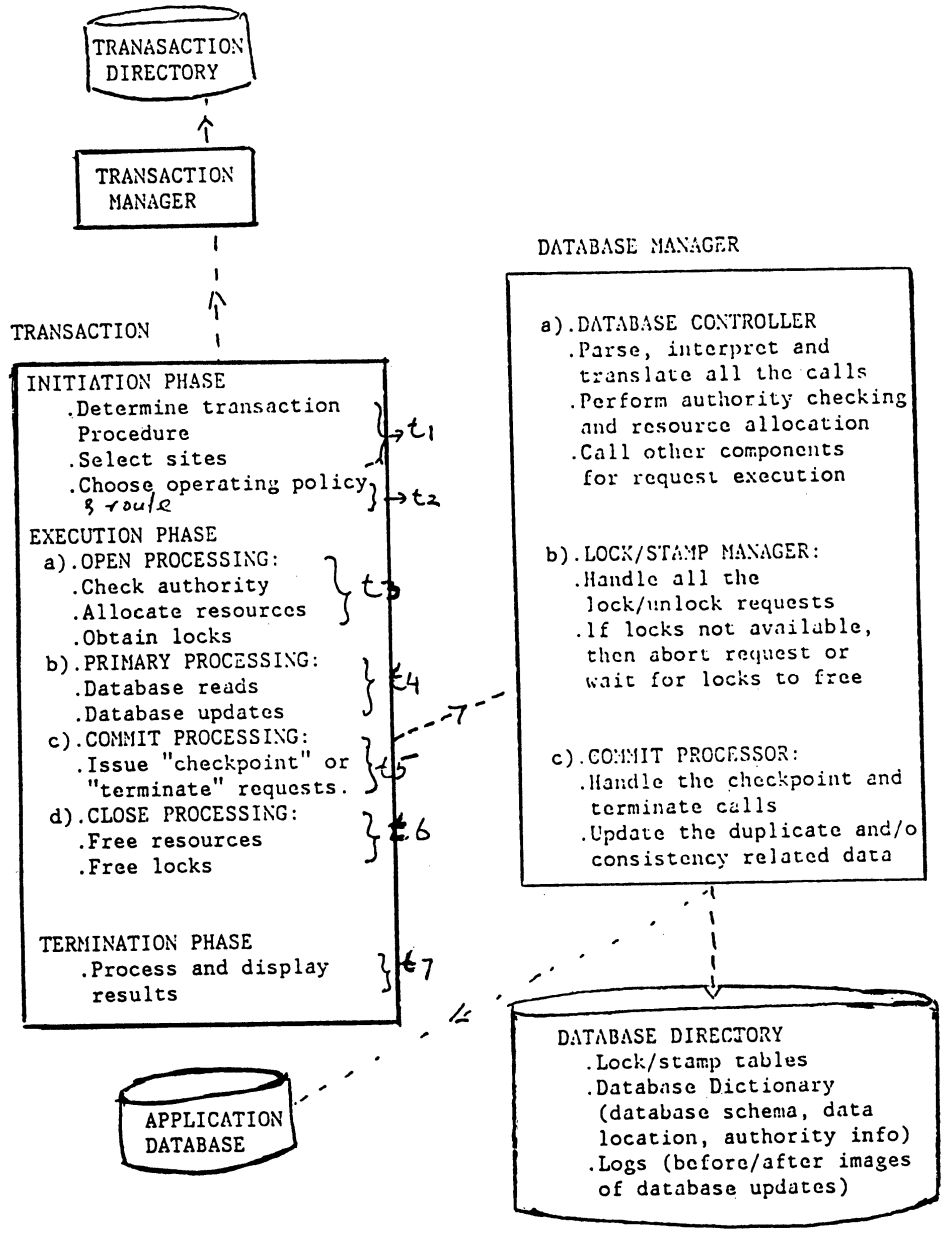
Note that we emphasize the importance of arrival node k for each transaction. These times can be grouped and viewed differently. A simple view separates all the queuing delays from the procesing delays, i.e:

$$RT(t,k) = PD(t,k) + QD(t,k) \quad \text{.....(5.30)}$$

where PD(t,k) is the total processing delay given by sum of all the local processing and communication processing delays, without queuing, and QD(t,k) is the sum of all the queuing/locking delays at the local devices, communication links and lock tables, for transaction t when it arrives at node k. Our approach to calculate RT is iterative and consists of the following steps:

1. The processing delay PD is calculated by using the predictive model introduced in chapter 2 and extended in chapter 4. This model ignores all queuing/locking delays and is modified to include the effect of parallelism. Details are presented in section 5.5.1.
2. The predictive model is extended in section 5.5.2 to include queuing at the the node level. The queuing calculations are based on operational analysis instead of the traditional stochastic approach. This model yields an initial value of queuing delay QD.
3. The response time model is further extended to include effect of synchronization traffic in section 5.5.3. The queuing delay QD is recomputed, however we assume that the synchronization (lock/unlock) requests do not encounter conflicts.
4. The complex issue of conflict analysis in synchronization requests is addressesd in section 5.5.4. The queuing delay QD is calculated by including probabilities of lock conflicts.
5. The queuing delays at the communication links are introduced and the effect of background workload is introduced to compute a final value of QD.

FIGURE 5.5 THE PREDICTIVE MODEL



5.5.1 Processing Delays and Parallelism

Fig (5.6) shows a simple network model which can be used to calculate PD. In this network, transactions arrive at node k with rates $T_a(t,k)$. The arriving transaction is processed at the originating node where it performs initial processing and is then routed to other nodes depending upon the location of the affected programs and datasets, query processing options, update synchronization algorithms and network topologies. Thus the transactions enter a given node in two basic manners: external arrivals and requests generated during execution of transactions (see fig 5.6). The transactions also leave a node in two manners: after completion and to get service from other nodes. In the simplest case which involves no queuing, locking or parallelism, the processing delay PD(t,k) is given by:

$$PD(t,k) = \Sigma_y\ L(t,k,y)E(y) + \Sigma_z\ M(t,k,z)E(z) \quad \ldots\ldots\ldots\ldots\ldots (5.41)$$

where L(t,k,y) and M(t,k,z) are the local and remote service requests given by equation 4.29 in chapter 4, and equation 5.1 in this chapter. E(y) and E(z) are the mean service requests of device types y and z. Note that a "device" in this context can be a cpu, a disk, a communication link, a node or a subnetwork. The mean service times E can be externally measured and provided as input parameters.

To include parallelism, we first note that a single transaction can activate several programs and some of these programs could execute in parallel, depending on the operating policies discussed in distributed transaction processing. Then, pd(t,k,p), the actual processing time spent for execution of program p due to the arrival of a transaction at node k is given by:

$$pd(t,k,p) = \Sigma_y\ L_p(t,k,y)E(y) + \Sigma_z\ M_p(t,k,z)E(z) \quad \ldots\ldots\ldots\ldots (5.42)$$

where $L_p(t,k,y)$ and $M_p(t,k,z)$ are, respectively, the local service requests generated at local device y and the remote messages exchanged over link z, due to the execution of program p due to arrival of transaction t at node k. These variables can be derived from equations 4.29

Figure 5.6: The Response Time Model

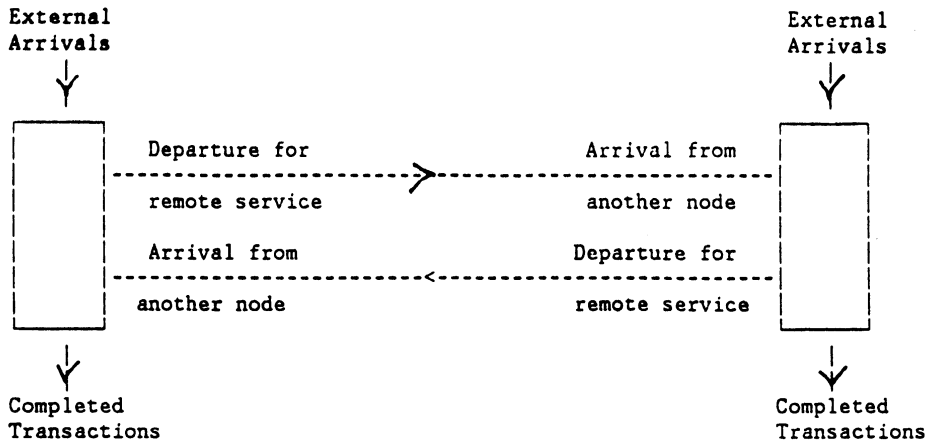
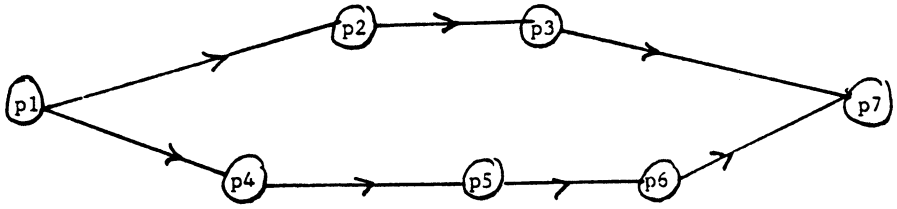


Figure 5.7 Parallelism of Programs in Transactions



and 4.31.

A transaction can be decomposed into a set of *serial and parallel program groups*, where a serial program group consists of a set of programs that must be executed in a predetermined sequence and a parallel program group is a set of programs that can execute in parallel to each other. To illustrate the concept of program groups, consider a transaction which utilizes programs (p1, p2, p3, p4, p5, p6, p7) such that programs p2 and p3 must run in a predetermined sequence, p4, p5 and p6 must also run in a sequence, however p2 and p4 can execute in parallel (fig 5.7). We can form the following serial groups:

g1=p1, g2=(p2, p3), g3=(p4,p5,p6), g4=p7

and the following parallel groups:

g1' = (g2, g3)

Then:

$$PD(t,k) = ws(t,k,g1) + wp(t,k,g1') + ws(t,k,g4) \quad \ldots\ldots\ldots\ldots (5.44)$$

where

$ws(t,k,g)$ = processing delay for serial group g

$$= \Sigma_p \, pd(t,k,p) \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots (5.45)$$

$wp(t,k,g)$ = processing delay due to parallel group g

$$= \max [pd(t,k,p)] \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots (5.46)$$

Thus, $PD(t,k)$ can be computed by using the following algorithm:

1. Compute $pd(t,k,p)$ for all the programs activated by transaction t, by using equation (5.42).
2. Decompose the transaction into serial program groups (g1,g2,,,gn). Compute $ws(t,k,g)$ for each serial group g by using equation 5.45. The transaction graphs introduced in chapter 4 can be used for this purpose.
3. Group the serial groups into parallel groups (g1',g2',,,gn'), if possible. The parallelism depends on the transaction graph and also on the transaction processing options. Compute $wp(t,k,g)$ for each parallel group by using equation 5.45.
4. Compute $PD(t,k)$ by using equation 5.44. Note that if no parallel groups are found, then this procedure yields the same $PD(t,k)$ as given by equation 5.41.

It should be noted that the query processing options significantly influence parallelism of programs in transactions. For example, in ring operation graphs, virtually no parallelism is used since the transaction travels from one node to another in a serial manner. Thus the actual procedure to build the parallel and serial groups must take into account the transaction processing options and the program sites selected.

5.5.2 The Device-Busy Queuing

Fig (5.8) generalizes the response time model to include device-busy queuing. A transaction enters the network at an originating node, where it waits at the cpu for execution. Once the transaction starts execution, it executes a burst of cpu instructions, i/o instructions and communication i/o instructions in a predetermined order. Thus a transaction circulates between the cpus, the i/o facilities and the communication links, before termination. Queues can be built at the links, the cpus and the disks, thus forming a network of queues. It is our objective to calculate QD(t,k), the average queuing delay encounterd by transaction t when it arrives at node k:

$$QD(t,k) = \Sigma_y L(t,k,y)qd(y) + \Sigma_z M(t,k,z)qd(z)$$

where qd(y) and qd(z) represent the queuing delays encountered at a local device y and a link z. Note that in this expression the queuing delays at the various devices are multiplied with the number of services requested on that device from transaction t. We will need to determine qd(y) and qd(z) in terms of the workload generated by transaction t plus all the other transactions in the system.

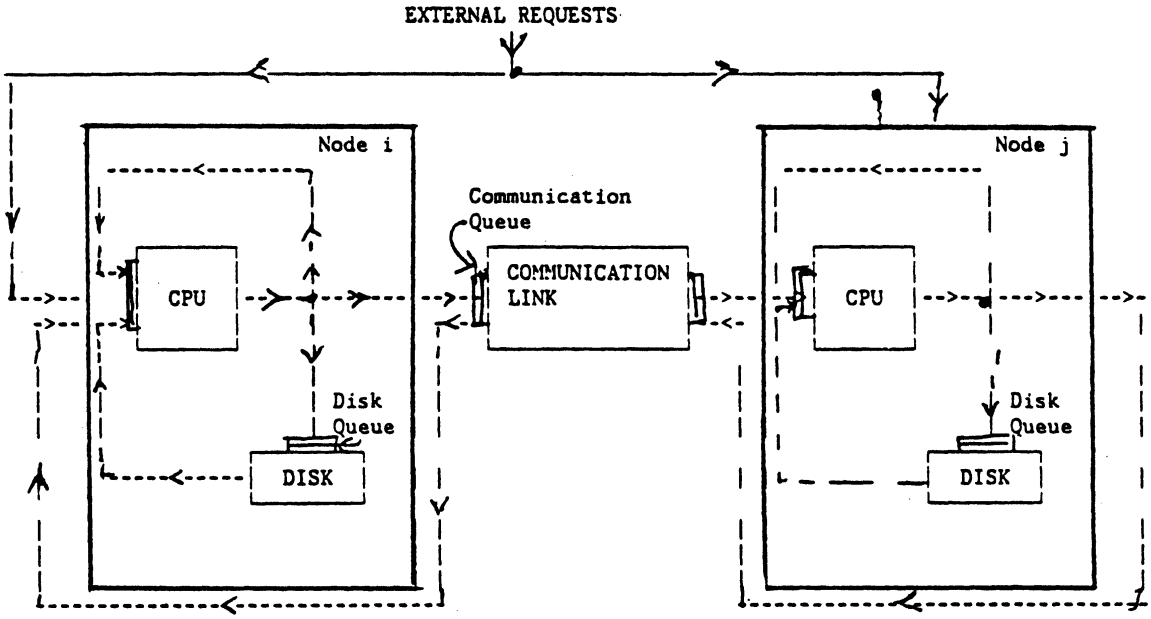
Calculation of the queuing delays qd in queuing network models is an extremely complex task. Traditionally, calculations for queuing network models have been based on stochastic assumptions (stationarity, independence, ergodicity, poisson arrivals, exponential service times etc.). A major source of puzzlement has been the accuracy of queuing network models because the stochastic assumptions can be disproved empirically. Buzen <BUZEN76f, BUZEN76f2, BUZEN78f> has made significant contribution by proving that the traditional equations of stochastic queuing network theory are relationships between operational (directly measured) variables and can be derived directly by using testable assumptions. In addition, he has also shown that several classical queuing theory relationships do not necessarily require Poisson arrivals and exponential service times. For example, he has shown that the gen-

eral birth-death Markovian queuing model is independent of the poisson arrivals and service times represented by independent, exponentially distributed random variables. Buzen's approach to queuing network calculations is termed *operational analysis* and has gained significance among practitioners and researchers <DENN79f>.

The calculations shown here are based entirely on operational analysis for four reasons: the model assumptions can be directly tested on a real life system, the model can be extended easily, the bottleneck analysis can be easily used to study the limiting behaviour, and the model is an interesting application of operational analysis that has not been reported previously. Denning and Buzen <DENN79f> have given an expert comparison of stochastic and operational approaches to queuing network models. The calculations in this section are based on <DENN79f, BUZEN76f, BUZEN76f2>.

Figure (5.9) shows the basic parameters and relationships of an operational system. The single server shown in figure (5.9) can be a cpu, an i/o facility like a disk or a communication link. The two node computing system shown in figure (5.8) is modelled as a queuing network, where each cpu, disk and link is considered as a server. For a large system with several nodes, where each node consists of several servers, the queuing analysis is complex and computationally expensive. We simplify the analysis by assuming that each node, i.e. all the servers in that node, can be analyzed independently. This process, called "decomposition", is based on the work done by Jackson <JACKS57f>, who showed that in a network of queues consisting of exponential servers and poisson arrival rates, each server can be treated independently with an arrival rate equal to the sum of all the arrival rates at that server. Decomposition can be applied at several levels: a large network of servers (devices) can be decomposed into "subnetworks", which in turn can be decomposed. In operational analysis, the decomposition holds if the servers satisfy the following two operational assumptions:

Figure 5.8 The Queuing Network Model



NOTES:

- When a transaction leaves the cpu, then it either goes back to the cpu or queues on the disk, the communication link, or the locks.
- After completing a disk or communication request, the transaction ALWAYS goes back to cpu.
- External requests are routed to nodes where they ALWAYS queue on cpu.

Fig. 5.9 Operational Parameters

BASIC PARAMETERS

τ = observation period
a = numbers of arrivals in time τ
b = busy time of the server. $b \leq \tau$
c = numbers of service completions in time τ

BASIC RELATIONSHIPS

a/τ = arrival rate
c/τ = departure rate - throughput
b/τ = server utilization
b/c = service time per request

1. The arrival rates are homogeneous, i.e. the arrival rates are not affected by the number of jobs in the system (state of the system).
2. The service times are homogeneous, i.e. the service times are not affected by the state of the system.

The assumption of homogeneous arrival rates is the operational counterpart of Poisson arrivals and the assumption of homgeneous service times is the operational counterpart of exponential service times. However, these assumptions can be easily tested in "offline" experiments. The subject of decomposition has been treated by <CHAND78f, CHAND75f, COURT77f, DENN78f>.

In addition to the homogeneous service time and arrival rate assumptions, we will postpone link level queuing for later discussion by assuming fixed communication times. At present, we treat each node in the network shown in figure (5.8) as a "closed central server" type model, in which the cpus are the central facilities and the number of jobs in the system is constant (a new job enters the system as soon as an old job terminates). Within the node, the request leaving a device i is routed to device j with a routing probability pr(i,j). If the number of arrivals in time τ are equal to the number of departures in the observation period τ (the principle of flow balance holds), then an important variable V, called the *visit ratio*, is given by:

V(y) = visit ratio of device y = c(y) / CO

where c(y) is the number of services completed by device y and CO is the total number of services completed by the entire system in the observation period τ. Basically V(y) expresses the mean number of requests per job for device y and can also be viewed as the job flow through device y relative to the systems output flow. For example, if each job completed in the system issues about 10 requests for disk1 and 15 for disk2, then the visit ratio of disk1 is 10 and the visit ratio of disk2 is 15. We

will show later that V(y) can be calculated in terms of the local and remote service requests. Another operational quantity used throughout is E(y), the mean service time of device y. We will show that all the queuing calculations of an operational queuing network are expressed in terms of V(y) and E(y), where E(y) is an input parameter and V(y) can be always computed from workload.

In order to determine the response time of each device with queuing delay, we define "state" of the system: a vector

Z = n(1),n(2),n(3),,,,,,n(NY)

where n(i) is the number of requests in queue at device i, including the request being processed, and NT = n(1)+ n(2) + ...n(NY) is the total no. of active requests in the system . NY is the maximum no. of devices in the system.

To calculate the queuing delays, PT(Z), the proportion of time the state of system is Z, needs to be calculated. In order to calculate PT, we derive the standard product form solutions by using operational analysis. The calculations presented here are based on the following major assumptions:

i). All devices exhibit homogeneous service times and arrival rates. Homogeneity is a reasonable assumption in most cases when one device does not block others. This assumption considerably simplifies analysis of the DDP network because we can assume that the transaction arrival rates at any node k, $T_a(t,k)$, and the local and remote service requests generated due to transaction arrivals, L and K, exhibit homogeneous arrival rates.

ii). The transactions do not overlap the use of devices. In other words a transaction relinquishes the control of the cpu whenever it issues a local i/o or a communication request. This assumption is valid for most commercial application systems, since the higher level languages (Cobol, Fortran) do not facilitate overlaps.

iii). The number of arrivals equal the number of departures at each device, for the observation period. This is called the principle of flow balance and resembles the concept of regeneration points in simulation. In practice, flow balance assumption is satisfied by several systems if the observation period is long, say a day.

iv). The observable state changes result from single jobs, not several jobs, either entering, exiting or moving between devices of the system. This assumption is called "one-step behaviour" assumption. One step behaviour is a property of many real life systems and in others, the number of job transitions occur simultaneously on rare occasions.

Once again, all these assumptions can be easily tested by setting up suitable experiments and are generally true for most computer systems. Denning and Buzen <DENN79f> have derived the following product form solution for a closed queuing network which satisfies the above stated assumptions:

$$PT(Z) = F(n(1)).\ F(n(2)).\ \ldots.\ F(n(NY))\ /\ G'$$

where: $F(n(y))$ = factor for device y $= (V(y).E(y))$

G' = normalizing constant

The mean no. of transactions waiting at device y is given by:

$$nw(y) = \sum_{n=0}^{N} n(y) \times PT(Zn(y)) \qquad \ldots\ldots\ldots\ldots (5.47)$$

where $Zn(y)$ = state of the system when there are n transactions on device y.

$PT(Zn(y))$ = Proportion of time the state of the system is $Zn(y)$

Thus by knowing PT(Z), the queued transactions and the queuing delays can be estimated. Unfortunately, PT(Z) is cumbersome to calculate. A mathematically efficient formula for PT(Z) has been given by Buzen <BUZE73f>, which yields the following expression for n'(y):

$$nw(y) = \sum_{n=0}^{NT} \{V(y)E(y)\}^{n} GG(NT-n,\ NY)/GG(NT,\ NY) \qquad \ldots\ldots\ldots (5.48)$$

where E(y) = mean time to service a request by device y (input parameter

V(y) = visit ratio of device y

= average number of requests issued for device y per transaction execution (see below).

GG = computation matrix, depends on V and E, (see below).

NT = the maximum no. of concurrent transactions in the system.

NY = the maximum number of device (processors, i/o facilities, communication links) in the network

Since nw(y) includes the transaction being processed, the mean queue length at device y is given by:

$$qw(y) = nw(y) - 1 \qquad \text{.....(5.49)}$$

If $qw(y) \leq 1$, then there is no queuing at device y and hence the average queuing delay qd(y) = 0, else qd(y) = E(y) × qw(y). Recall that it is our objective to calculate qd(y). Thus the *final expression* for the mean queuing delay at device y is given by:

$$qd(y) = E(y)\left[\frac{\sum_{n=0}^{NT}\{V(y)E(y)\}^{n} GG(NT-n, NY)}{GG(NT,NY)} - 1\right] \qquad \text{....(5.50)}$$

If qd(y) from this equation is negative, then we use qd(y) = 0. Thus the queuing delay for any resource can be computed if the visit ratios V(y) and service rates E(y) are known. The formula presented by equation (5.50) is used very heavily in the next two sections and can be used for different number and type of devices. It also yields the total queuing delay QD(t,k) when transaction t arrives at node k, through a summation of qd(y) for all y used by t when it arrives at node k, i.e:

$$QD(t,k) = \Sigma_{y}\, L(t,k,y)qd(y) \qquad \text{..... (5.51)}$$

The visit ratio, V(y), for node y is given by:

$$V(y) = \Sigma_{t\in T}\, \Sigma_{k\in N}\, T_a(t,k)\, L(t,k,y) \qquad \text{.....(5.52)}$$

where T is the set of all the transactions in the application system and N is the set of all the nodes. The visit ratio for the cpus and disks can be obtained by using equations 5.15 and 5.16. The matrix GG is a

two dimensional matrix with columns representing the devices and rows the load (degree of multiprogramming). The first row of GG is 1s, the first column is set to zeroes and row 1, column 1 = 0. An interior element is given by:

$$GG(n,y) = GG(n,y-1) + V(y)\ E(y)\ GG(n-1, k)$$

where n = (2,3,,,,NT) and y = (2,3,,,NY). Thus, this matrix is a function of E(y) and V(y), which have been discussed above. An example will illustrate these calculations later.

5.5.3 Locking Delays Without Conflicts

As indicated previously, a lock is a software indicator which allows/disallows transactions to access data. In this research we are primarily concerned with locking as it pertains to database consistency and update synchronization. Use of locks for security purposes is not discussed here.

Each lock/unlock request can be treated as a service request. These requests consist of:

- $L_{lk}(t,k,i)$, the local lock/unlock service requests issued at node i due to the arrival of transaction t at node k. The analytical expression for $L_{lk}(t,k,i)$ have been derived previously in terms of the data allocations, read/write operations, consistency relationships, location of the lock/stamp manager and core residency of the lock/stamp tables (chapter 4, equation 4.31). We assume that L_{lk} exhibit homogeneous arrival rates.
- $R_{lk}(t,k,i)$, the remote lock/unlock requests exchanged between nodes i and j, when transaction t arrives at node k. The analytical expressions for $R_{lk}(t,k,i)$ have been derived previously in terms of the data allocations, read/write operations, consistency relationships, location of the lock/stamp manager, and the ammount of piggy-

backing (chapter 4, equation 4.32e). We assume that R_{lk} exhibit homogeneous arrival rates.

The R_{lk} cause congestion at the communication link level. However, we first concentrate on $L_{lk}(t,k,i)$ which yields queuing at the nodes due to locking/unlocking by ignoring conflicts due to lockouts. We model the existence of locking/unlocking delays by introducing a "lock" device at each node. Thus at each node, the requests leaving the cpu are either queued at the "data" disk, at the "lock" disk , at the cpu or at a remote node. Fig (5.10) shows the new node model and shows the routing frequencies between the new devices.

We like to use the generalized queuing network formula given in equation (5.50) to compute the response time including the locking delays. To use this formula, the following two variables are needed:

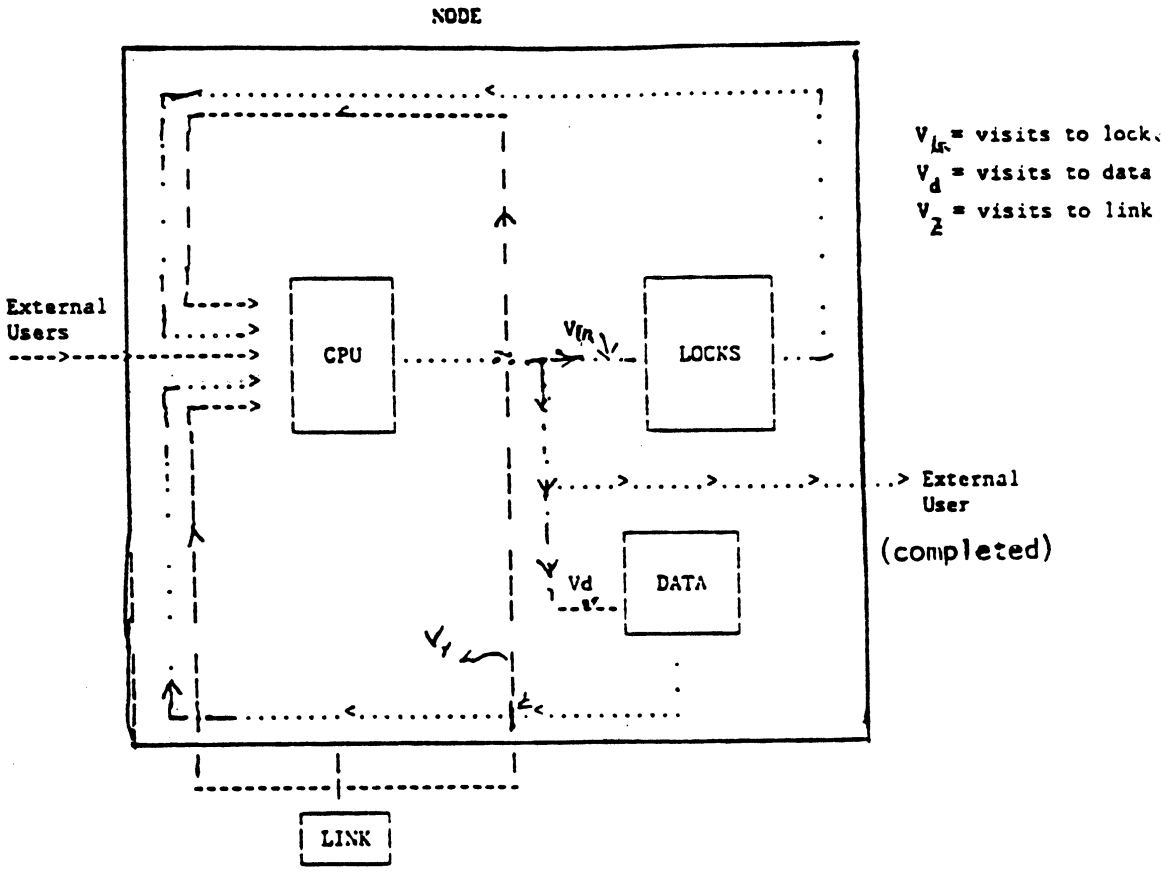
1. The service time E(i) of the lock device at node i.. This value depends upon the physical characteristics of the device where the lock tables are kept. If, for example, the locks are kept on a disk, then E(i) = the mean service time of the disk at node i.
2. The visit ratio V(i) of the lock device at node i. This value is given by the following equation:

$$V(i) = \Sigma_t \Sigma_k T_a(t,k) L_{lk}(t,k,i) \quad \text{.....(5.55)}$$

where $T_a(t,k)$ is the mean arrival rate at node k.

Thus we know the two basic quantities, and can calculate the queuing delays by using equation (5.50). Note that by introducing the "lock" table device, one device is added at each node. If the locks are core resident, then the queuing due to locking is insignificant. As a simplification, we can assume that the lock device is the same as the data device with a visit ratio which is the lock device plus the data device visit ratios.

Figure 5.10: Queuing Network with locks (no conflict)



5.5.4 Queuing Delays With Lock Conflicts

Queuing delays occur due to two reasons: device busy conflicts and the "intent" conflicts, referred to here as the locking conflicts. In the last section we determined device busy queues due to lock/unlock requests without taking into account the effect of locking conflicts. A locking "conflict" occurs whenever a transaction t attempts to lock an item that has been already locked by another transaction. Thus the transaction t will have to wait until the locked item is "freed". We must distinguish between locking conflicts and device "blocking" , which occurs frequently in computer systems where a control unit of an

i/o processor blocks all the devices on the control unit, because device blocking violates the homogeneous service times assumption.

We model the locking conflicts by introducing a new "conflict" device at each node (see fig 5.11). Note that in case of centrally controlled synchronization algorithms like CCAA, CCA, CLAA and CLA, the conflict device only exists at the central node since all the locking/unlocking activity is performed at the central node. Let $\rho_w(i)$ show the probability that a transaction will have to wait, at node i, for locks and let $V_{1k}(i)$ denote the average number of lock/unlock requests issued at node i per transaction. $V_{1k}(i)$ represents the visit ratio of lock/unlock requests and has been given by equation 5.55. In case of centrally controlled algorithms, $V_{1k}(i) > 0$ for i=z, 0 otherwise; where z is the central node. As shown in fig (5.11) the number of requests serviced by lock device are $[1-\rho_w(i)]V_{1k}(i)$ and the requests serviced by a conflict device is given by $\rho_w(i)V_{1k}(i)$.

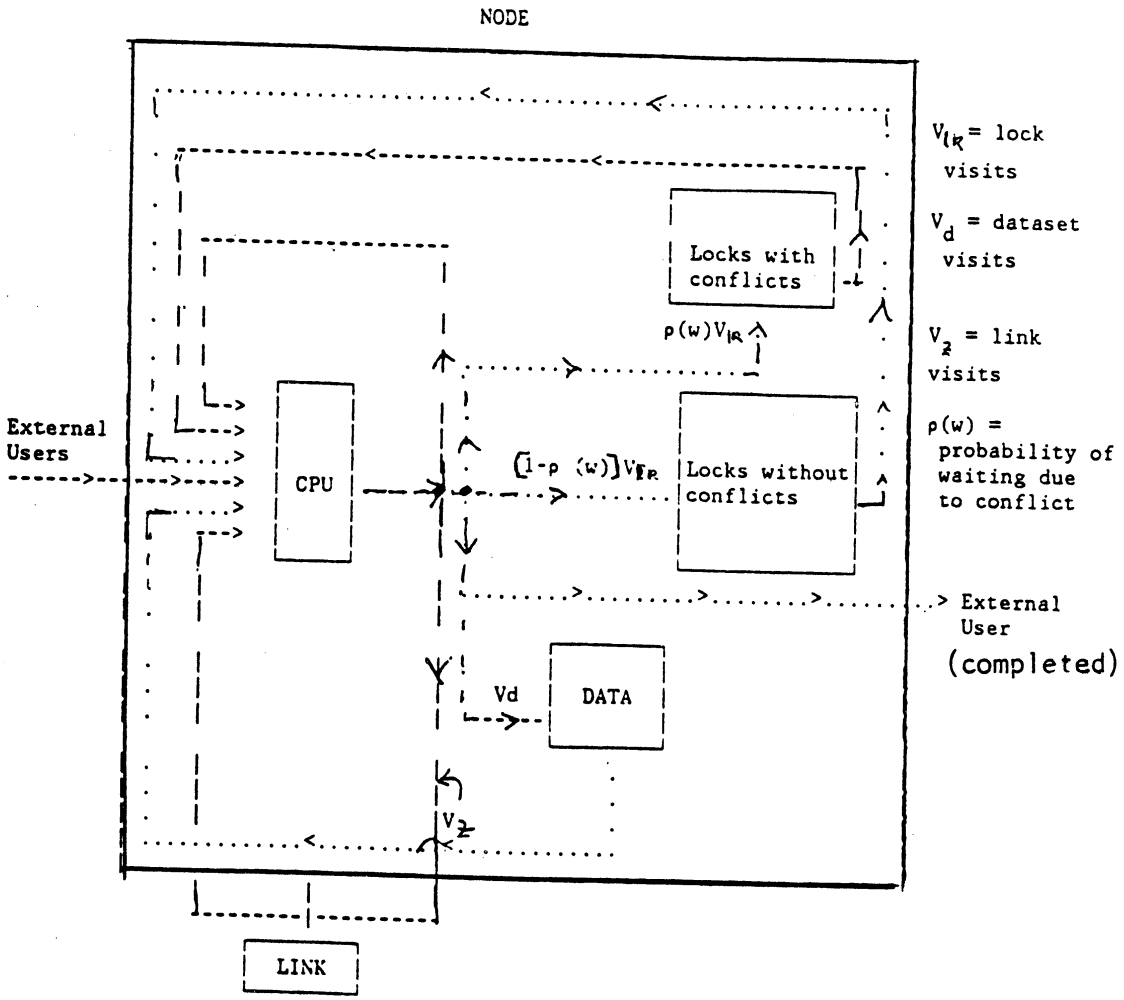
In order to use equation (5.50) to include the locking conflicts, we need to determine the service time and the visit ratio of the conflict devices.

a) Service Time: The service time of the conflict device i is given by:

$$[E(i) + \gamma/2] \quad \text{.......................................} \quad (5.60)$$

where E(i) is the actual service time of the lock device and γ is the average lock time for any transaction. $\gamma/2$ shows the average time a transaction must wait until the locks are released. This is based on the assumption that the conflicting lock request arrives at a random point in time with respect to the transaction that holds the locks. We are also assuming that at a given point, a transaction does not conflict with more than one transaction. If too many conflicts occur, then the system becomes unstable, leading to extremely large queuing delays <GARC79d>.

Figure 5.11: Queuing Network Model with Lock Conflicts



The average lock time γ for any transaction depends upon several factors: type of operation (read or write), exclusive or shared access, consistency (synchronization) intervals, locking granularity, the service discipline being used by the lock/stamp manager and the use of locks versus time stamps. Recently, David Chen <CHEN83c> has studied this problem by defining different classes of transactions (single operation/ read or write only, single operation/read and write, multiple operation/read and write), two type of read operations (current versus response mode), shared and exclusive read/write operations and several service disciplines (FCFS, LCFS, read preemptive priority, write

preemptive priorty). We suggest that γ is given by one of the following values:

1. $\gamma = 0$. This relationship holds if the transactions are read only (shared read) and also when optimistic synchronization algorithms are used.
2. γ = rt, where rt is the average response time for a read/write of a transaction. This relationship holds if the transactions are single operation, if exclusive write/read are used at fine granularity (tuple level locking), or if the synchronization interval is equal to an operation (at the end of every read/write, and the locks are freed). An estimate of rt is given by:

$$rt=\Sigma_t\Sigma_k\Sigma_i L(t,k,i)E(i)/\Sigma_t\Sigma_k\Sigma_i L(t,k,i) + \Sigma_t\Sigma_k\Sigma_z M(t,k,z)E(z)/\Sigma_t\Sigma_k\ \Sigma_z M(t,k,z)$$

   If needed, a best-case or worst-case approximation of rt can be used. For best-case, rt= E1, where E1 is the average disk read time. This assumes that all the data is locally available and no remote requests are needed for a read/write operation. For worst-case, rt= 2E2 + E1, where E2 is the longest remote service time in the network. The E2 has been multiplied by 2 to indicate the request plus response routing.
3. γ = RT, where RT is the average response time of a transaction, or a class of transactions. This relationship holds if synchronization interval is equal to transaction duration (usual implementation of 2PL) or if granularity of locking is coarse (say the dataset level). Note that the following iterative procedure can be used to derive γ:i) calculate RT by ignoring all queuing, ii) set γ = RT, iii) use this value of γ in equation (5.60) to compute the service time of the conflict device, iv) use the new service time to compute a revised RT, v) set γ = RT just obtained and re-

iterate. For most practical purposes, this procedure converges after three iterations.

The two major assumptions used in estimating γ are: a transaction does not conflict with more than one transaction at a given time, and the lock/stamp manager uses a FCFS discipline. These assumptions are valid for most practical cases <GARC79d>. In case of pessimistic synchronization algorithms, $\gamma = MM \times RT$, where $MM > 1$. For worst case analysis, MM = the number of transactions in the system = degree of multiprogramming. Thus, the average locking time γ can be given by:

$\gamma = 0$ for read only shared systems.

$= rt$ for short consistency intervals .....(5.61)

$= RT$ for long consistency intervals

It should be noted that these values of γ cover several cases studied by Chen <CHEN83c>. For example, $\gamma = 0$ represents the shared read systems, $\gamma = rt$ symbolizes the shared read/exclusive write system and $\gamma = RT$ describes the exclusive read/exclusive write systems. It should be further noted that we concentrate on "currency" mode transactions of Chen which produce the results after all commit processing. The "response-time" transactions can be easily represented by ignoring the commit processing step of the predictive model. In addition, our model is restricted to FCFS queuing discipline of the lock/stamp manager.

b) Visit Ratio: The visit ratio of conflict device at node i is given by $\rho_w(i) \times V_{lk}(i)$, where $V_{lk}(i)$ is the number of lock requests issued at node i and is given by:

$$V_{lk}(i) = \Sigma_t \Sigma_k T_a(t,k) L_{lk}(t,k,i)$$

$\rho_w(i)$, the probability that transaction t will have to wait for locks, at node i, is given by:

$$\rho_w(i) = \rho_c(i) J(i) \quad \text{.....(5.62)}$$

where $\rho_c(i)$= probability that a transaction will conflict at node i and J(i) = fraction of transactions holding locks at node i. To find ρ_c,

let u(t',i) represent the no. of items locked by transaction type t' at node i, and let NL(i) be the total no. of "lockable" items of the given application system at node i. In the discussion that follows we are assuming that data is not shared among applications. This is a practical assumption since in most of the cases data sharing among applications is rare. For example, a payroll application system does not share the information with inventory control or manufacturing applications. We shall show below that u can be represented in terms of L_{lk} and NL is a function of database size. Assuming that NL(i) >> u(t',i) for all t' Ξ T, the probability that the first lock request from transaction t will conflict with t' at node i, is:

$$\rho_c(t,t',i) = u(t',i) / NL(i)$$

The probability that the second lock request from t will also conflict is also u(t',i) / NL(i). Thus the probability of conflict for two lock requests will be 2u(t',i)/NL(i). Since transaction t locks u(t,i) items hence the total probability is given by:

$$= u(t,i)\ [u(t',i)/NL(i)]$$

The average value of $\rho_c(i)$ is:

$$\rho_c(i) = EXP(u(t,i)u(t',i)/NL(i)) = \{u(i)\}^2/NL(i)$$

where u(i) = mean value of u(t,i) for all transactions t Ξ T and EXP indicates expected value. Note that u(i) represents the mean value of lock requests generated at node i due to all transactions in the system Thus the expression used for ρ_c is:

$$\rho_c(i) = \{u(i)\}^2/NL(i) \qquad \text{.....(5.63)}$$

where u(i) = average no. of items locked by a transaction of the given application system at node i and NL(i) = total no. of "lockable" items of the application system at node i. Note that equation 5.63 holds only if NL(i) >> u(i). The values of u(i) and NL(i) depend on the synchronization algorithm being employed and the locking granularity. Due to the calculations of chapter 4, u is a function of the lock requests received at node i:

$$u(i) = \Sigma_t \Sigma_k \, L_{lk}(t,k,i)/N_n \, T_n \quad \ldots\ldots\ldots\ldots\ldots\ldots (5.64)$$

where N_n is the number of nodes, T_n is the number of transactions, and $L_{lk}(t,k,i)$ is the number of lock requests received at node i due to arrival of a transaction at node k.

In order to estimate NL(i), we need to recall that for distributed lock management systems, the lock tables are replicated, so NL(i) = ML for all i Ξ N, where ML represents the number of entries in the lock table. In case of centralized lock management systems, NL(i) = 0 for all non-central nodes, and is equal to ML at the central node z. ML is a function of the database size and locking granularity. Thus:

NL(i) = 0 when i is non-central node for centralized locks.

Else NL(i) = ML, where ML is given by:

ML = D_n for coarse granularity (D_n = no. of datasets)

= $D_n \, \Sigma_d \, D_s(d,2)$ for tuple granularity $\ldots\ldots\ldots (5.65)$

($D_s(d,2)$ = no. of tuples in dataset d)

The average no. of transactions holding the locks, J, is given by:

J(i) = (arrival rate of locks at node i).(average lock time)

arrival rate of locks = $V_{lk}(i) = \Sigma_t \Sigma_k \, T_a(t,k) \, L_{lk}(t,k,i)$ $\ldots\ldots (5.66)$

average lock time = γ (given by equation 5.61)

Thus the final expression for the probability of wait is given by:

$$\rho_w(i) = \{u(i)\}^2 \times V_{lk}(i) \times \gamma \, / \, NL(i) \quad \ldots\ldots\ldots (5.67)$$

where u(i), V_{lk} and NL(i) are given by equations 5.64 through 5.66. Now we can calculate visit ratio V(y) and the service time E(y) of conflict devices y. Next $w_{qd}(y)$ can be calculated from equation (5.50) to give the average queuing delay due to lock conflicts. Note that in these calculations we do not need to explicitly consider the location of the lock tables and the centralization versus decentralization of the update synchronization. These considerations were included in calculating the lock/unlock activities L_{lk} in chapter 4.

5.5.5 Communication Queuing, Background and Bottleneck Analysis

So far we have ignored queuing at the communication links. The average response time of the communication network for transaction t is given by:

$$\Sigma_k T_a(t,k) \Sigma_z M(t,k,z) E(z) / \Sigma_k T_a(t,k)$$

where $T_a(t,k)$ is the mean arrival rate of t at node k, M(t,k,z) is the number of messages processed by link z due to the arrival of transaction t at node k, and E(z) is the mean service time of link z. Note that M(t,k,z) has been used instead of R(t,k,i,j) to denote the remote service requests because M shows the messages travelled by physical links, and is thus dependent on the network topology and routing (section 5.1). Researchers like Garcia <GARC79d>, Chen <CHEN83d> and Kwon-Ki <KWON81d>, who have included the effect of update synchronization, have chosen to use a constant communication average response time to simplify analysis. This assumption holds well for lightly loaded communication systems and/or point-to-point networks with fixed routing.

We can easily extend our model to include queuing at network level, by calculating queuing delays at each link. In order to calculate qd(z), the queuing delay at link z, we simply need to: a) consider each link z to be an additional device at which services are performed in the network, b) calculate the visit ratios to link z , V(z), from the equation:

$$V(z) = \Sigma_t \Sigma_k T_a(t,k) M(t,k,z)$$

and c) use the formula shown in equation 5.50 to calculate qd(z). Now the total queuing delay encountered by transaction t when it arrives at node k, QD(t,k), is given by:

$$QD(t,k) = \Sigma_y L(t,k,y)qd(y) + \Sigma_z M(t,k,z)qd(z)$$

where y and z show the local devices and the communication links, respectively. Let us now turn our attention to including the effect of background workload on the response time. So far we have focussed on the foreground workload which consists of calculating the response time

due to the traffic generated of a chosen application, since the parameters L(t,k,i) and M(t,k,z) have been calculated in terms of given application system parameters. In practice, the foreground calculations are used for initial analysis, since the actual system workload consists also of the existing applications, the background workload. In order to include the effect of background workload, we need to reevaluate the service times E and visit ratios V of all the devices in the network, and use the adjusted values in equation 5.50.

The service rates E of the devices are usually not affected by the background workload since these parameters reflect the physical characteristics of the devices. Moreover, we will continue to assume the homogeneous service times. In order to compute the visit ratios, let the following parameters describe the background workload:

WL(i) - the total number of background local services completed at node i, per unit time

WM(z) - the total number of background communication messages handled by link z, per unit time

The adjusted visit ratios are:

V(y) = adjusted visit ratio at local device y

$$= \Sigma_t \Sigma_k T_a(t,k)L(t,k,y) + WL(y)$$

V(z) = adjusted visit ratio at link z

$$= \Sigma_t \Sigma_k T_a(t,k)M(t,k,z) + WM(z)$$

Substituting these values in equation 5.50, yield the queuing delays which include the background workload.

A benefit of the operational queuing network model is that the bottleneck analysis can be performed easily. Denning and Buzen <DENN79f> have proved that a device b is the bottleneck device, a device capable of saturating when the number of transactions in system becomes large, if:

$$V(b)E(b) = \max \{V(1)E(1),,,,,,,,,V(NY)E(NY)\}$$

where NY is the total number of devices in the network. The *saturation point* M*, the number of jobs in the system which results in queuing somewhere in the system, is given by:

$$M^* = \Sigma_y \, V(y)E(y) \; / \; V(b)E(b)$$

To estimate worst-case performance, the number of jobs in the system M can be set to $M > M^*$ in equation 5.50. Thus we can directly get the best case response time, RT_{pd}, by ignoring all queuing and locking delays, and worst case response time, by using the saturation point, from the operational queuing network model.

In addition, an estimate of system throughput and device utilization can be obtained from the following equations <Denn78f>:

System throughput = THR = GG(NT-1,NY)/GG(NT,NY)

Utilization of device i = V(i)E(i) × THR

Where NT = total number of jobs in the system, NY is the total number of devices in the system and GG is the computation matrix used to calculate the queuing delays qd(y) in equation 5.50.

5.5.6 Analysis and Illustration of Response Time Calculations

The overall procedure to calculate the total response time which includes the processing delays, parallelism, queuing, locking conflicts and background workload is:

1. Calculate the transaction reponse time PD due to processing delays, by using the algorithm described in section 5.5.1.
2. Compute the queuing delays qd(y) encountered at each device y by using equation 5.50 and then calculate the total queuing delay QD(t,k) by using equation 5.51. The average queuing delay, QD', is given by:

$$QD' = \Sigma_t \, \Sigma_k \, T_a(t,k) \; QD(t,k) \; / \; \Sigma_t \, \Sigma_k \, T_a(t,k)$$

3. Calculate the average lock time, γ , and the probability of wait due to conflict, ρ_w, by using equations (5.61) and (5.62) and then calculate visit ratio V(c) and the service time E(c) of conflict devices c. Now calculate qd(c) from equation (5.50) for the conflict devices. The total locking delay encounterd by transaction t arriving at node k, LD(t,k), is given by:

$$LD(t,k) = \Sigma_c \rho_w(c) L_{lk}(t,k,c) qd(c)$$

The average locking delay, LD', is given by:

$$LD' = \Sigma_t \Sigma_k T_a(t,k) LD(t,k) / \Sigma_t \Sigma_k T_a(t,k)$$

4. If needed, include the queuing delays due to the communication traffic and background workload. Recompute QD(t,k) as discussed in section 5.5.5.

5. Compute the total response time of transaction t when it arrives at node k:

$$RT(t,k) = PD(t,k) + QD(t,k) + LD(t,k)$$

Also compute the following average response time:

$$RT = \Sigma_t \Sigma_k T_a(t,k)RT(t,k)/ \Sigma_t \Sigma_k T_a(t,k)$$

6. Perform bottleneck analysis if needed and study the system throughput.

The main strength of this algorithm is that it can be easily used at several levels of detail. For example, the first step can be used in an initial analysis stage to determine the lower bound (best case) for the response time RT. This calculation can be used to eliminate clearly infeasible allocations in an early stage. More detailed, and consequently more expensive, computations can be performed at later stages when the feasible choices have been considerably narrowed. This is the cornerstone of the iterative program and data allocation methodology to be presented in the next chapter.

We need to observe that the main equation for computing queuing delays is equation 5.50, which is based on the assumptions of job flow ba-

lance, non-overlapping transaction and i/o, homogeneous service time, homogeneous arrival rates and one step behaviour. All these assumptions are weak, can be easily tested and any device that satisfies these assumptions can be included in the network model without major recalculations. In addition, these assumptions are usually satisfied by computing systems, and if not can be used for approximate analysis. We have also assumed that a given transaction conflicts with at most one other transaction. This assumption can be violated in systems where large number of lock conflicts occur. However, for systems with small locking granularity and short transactions, the probability of lock conflicts is small (section 5.5.4). Due to this reason our model can represent the pessimistic algorithms only approximately.

To demonstrate the usability of the response time model, consider a 3 node network, consisting of nodes n1,n2 and n3 to which a one program (p), one dataset (d) application system is to be allocated. Let us assume following properties of the network:

.The network consists of a star topology with node n1 as the central node. The two communication links are indicated as z1 and z2 (see figure 5.12a).

.The network control is distributed and the routing is fixed.

.The average service time of local service at device y, E(y), is 10 millisecond for all y.

.The average service time of remote service requests at link z, E(z), is 50 milliseconds for all z.

.The site selection is based on minimal cost.

.2PL update synchronization algorithm is used.

The application system consists of only one transaction t which accesses dataset d through program p, and consists of the following additional properties:

.The transaction arrival rate, $T_a(t,k)$, is 10 per minute, for all k.

.The program p issues 35 reads and 5 writes to dataset d.

.The program reads 30 input messages and generates 80 display messages.

For analysis, let us assume that program p is replicated and dataset d is allocated to nodes n1 and n2 (figure 5.12a). Due to this allocation, local service requests, L, and remote messages M are calculated based on the equations derived in chapter 4 and are shown in figure 5.12b. The L and M matrix show the data processing plus the locking/unlocking activity. Let us first determine PD(t,k), the processing delay encountered when transaction t arrives at node k, which is given by:

$$PD(t,k) = \Sigma_y L(t,k,y)E(y) + \Sigma_z M(t,k,z)E(z)$$

Using the values of L, M and E, this equation yields the following value of PD(t,k), for k= n1, n2, n3, in seconds:

PD(t,k) = 5.500 11.000 11.000

Note that the processing delay is minimal when the transaction arrives at node n1 because n1 is the central node and the data exists at this node. Next, we calculate the queuing delay without any locking conflicts. The first step in calculating the queuing delays is calculation of the visit ratio V(y) for local devices y and V(z) for links z, given by:

$$V(y) = \Sigma_t \Sigma_k T_a(t,k)\ L(t,k,y)$$

$$V(z) = \Sigma_t \Sigma_k T_a(t,k)\ M(t,k,z)$$

Since we have three nodes and two links, we get:

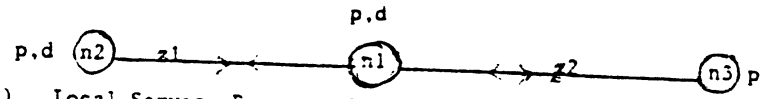
V(y) = 240.0 135.0 0.0

V(z) = 365.0 110.0

Note that the visit ratios of nodes n1 and n2 are high because dataset d is allocated to nodes n1 and n2 and is thus processed there. The difference between the visit ratios of n1 and n2 is due to the more frequent site selection of n1. Also note that the visit ratio of link z1 is high because all messages between nodes n2 and n3 are routed through n1 (figure 5.12a).

FIGURE 5.12: RESPONSE TIME CALCULATIONS

a) The Network



b). Local Service Requests L(k,i) and Remote Messages M(k,z):

L(k,i) =

k \ i	1	2	3
1	80	45	0
2	80	45	0
3	80	45	0

M(k,z) =

k \ z	1	2
1	85	0
2	195	0
3	85	110

c). Computation Matrix GG:

load \ Devices →	0	1	2	3	4	5
0	0	1	1	1	1	1
1	0	2	4	4	22	27
2	0	6	11	11	412	564
3	0	14	28	28	7553	10653
4	0	33	72	72	137919	196310
5	0	80	176	176	2517194	3597998
6	0	191	429	429	45939168	65728144
7	0	459	1038	1038	838390016	1199894528
8	0	1101	2502	2502	15300603904	21900021760
9	0	2642	6019	6019	279235722264	399e85779456
10	0	6340	14466	14466	5096046854144	7294317232128

d). Lock/unlock requests $L_{lk}(k,i)$

k \ i	1	2	3
1	40	40	0
2	40	40	0
3	40	40	0

e). Probability of Conflict and Wait Due to Lockout

$P_c(c)$ = 0.000008 0.000008 0.0

$P_w(c)$ = 0.006600 0.006600 0.0

f). Visit Ratios After Locking Conflicts:

V(y) = 239.2 134.2 0.0

V(z) = 365.0 110.0 0.8

V(c) = 0.8 0.0 0.0

g). Response Time For Various Synchronization Algorithms:

NOTE: The network and Data Allocation is the same as Shown in Fig 5.12a

Synchronization Algorithm	Average PDEL	Average QDEL	Average LDEL	Average TDEL
Basic 2PL	6.87	36.43	0.24	43.55
Basic Timestamp	5.37	22.99	0.13	28.49
Ellis Ring	5.25	34.99	0.09	40.32
Thomas Voting	3.87	22.94	0.04	26.86
CCAA and CCA	3.20	9.51	0.02	12.73
CLAA and CLA	6.87	36.43	0.42	43.55
Pessimistic	10.17	28.97	11.88	51.03
Optimistic	3.27	11.61	0.0	14.88

Next the GG matrix is calculated and the queuing delays at all 5 devices are computed by using equation 5.50. For the purpose of illustration, we assume a closed network where all the 5 devices are interconnected. Figure 5.12c shows the GG matrix for the 5 device closed queuing network, with a workload of 10 transactions. Using this value of the GG matrix, the visit ratios V and service rates E, equation 5.50 gives the following values of queuing delays qd(y) and qd(z):

qd(y)= 0.0001 0.0 0.0 qd(z) = 0.417 0.0

Note that a small queuing delay is encountered at node n1 and some delay is encountered at link z1. The total queuing delay encountered by the transaction, QD(t,k), when it arrives at node k, is given by:

$$QD(t,k) = \Sigma_y L(t,k,y)qd(y) + \Sigma_z M(t,k,z)qd(z)$$

In order to calculate the locking delays, we introduce one locking conflict device at each node and calculate the probability of conflict and then compute the probability of wait $\rho_w(c)$ at the conflict devices c=c1, c2 and c3, where c1 exists at n1, c2 at n2 etc. First, we need to calculate the service time of the lock conflict devices c, given by:

$$E(c) = E(y) + \gamma/2$$

Assuming a fine granularity system with 2PL, the value of γ, the average lock time, is approximately the average processing delay, PD', given by:

$$PD' = \Sigma_t\Sigma_k T_a(t,k)\, PD(t,k) / \Sigma_t\Sigma_k T_a(t,k) = 6.87$$

The service time of the conflict device is then given by:

$$E(c) = E(y) + \gamma/2 = 3.447 \text{ second for c=c1, c2 and c3}$$

and y = n1, n2, n3.

Next we need to calculate the new visit ratios based on the lock/unlock requests and conflicts. Figure 5.12d shows the lock requests, $L_{lk}(i)$, received at node i and figure 5.12e shows the probabilities of conflict ρ_c and the probability of wait ρ_w for the conflict devices. The visit ratios of the conflict devices are given by:

$$V(c) = \Sigma_t \Sigma_k T_a(t,k)\, \rho_w(c)\, L_{lk}(t,k,c)$$

Figure 5.12f shows the new visit ratios due to locking conflicts. Note that now the number of devices has increased to 8. Also note that the visit ratios of n1 and n2 have decreased because the locking conflict requests are being handled by devices c1 and c2, the conflict devices at nodes n1 and n2. Since the visit ratios and service requests have been modified, we need to recalculate the GG matrix and compute the locking delays qd(c) by using equation 5.50, given by:

qd(c)= 0.607 0.607 0.0

The total locking delay, LD(t,k), for the transaction arriving at node k, is given by

$$LD(t,k) = \Sigma_c \rho_w(c) L_{lk}(t,k,c) qd(c)$$

Now, we can calculate the average processing, queuing, locking and total delay for this configuration:

PD' = 6.875 QD' = 36.436 LD' = 0.240 TD' = 43.549

Note that the queuing delay is the major component of the response time. This analysis also gives the bottleneck device which is the device with largest V(y)E(y). It can be seen that link z1 is the bottleneck device, since V(z1)E(z1) = 365 × 5 = 1725. Intuitively this is obvious since link z2 has high traffic because all the accesses to data at node n2 must travel link z2, a slow link. Replacement of this link with a faster link will remove the bottleneck. We can even estimate how fast z1 should be, by using the following simple equation:

$$E(z1) \leq KK / V(z1)$$

where K is the next highest product V(i)E(i). Of course, we can change the network topology by adding a direct link between n2 and n3 which totally changes the visit ratios and thus introduces new bottlenecks. Let us now explore, the effect of other synchronization algorithms. Figure 5.12g shows the average processing, queuing, locking and total delay for the same application system, support system and program and data allocation; but by using other synchronization algorithms (the example above was based on 2PL algorithm). It can be seen from figure 5.12g

that for this particular case, the centrally controlled synchronization algorithms minimize the total response time. This is intuitively obvious since we are using a star network topology and the data is allocated to the central node. Note the extremely long delays encountered due to the pessimistic synchronization algorithm.

The calculations shown above have been performed by a Fortran program which implements the response time model. This program can be used to study the effect of transaction arrival rates, read/write ratios, the number of lockable data items, transaction duration, data duplication and various update synchronization algorithms on the processing delays, queuing delays, locking delays and the total response time. Extensive analysis of these issues have been given by Chen <CHEN83c>. The relationship between his work and this research is formally discussed in appendix J.

5.6 Chapter Summary and Significant Results

In this chapter, several important dependent variables have been calculated. The communication messages are calculated by introducing the path matrix, the transaction availability is calculated by extending the Mahmood and Riordon <MAHM77c> formulation, the software duplication cost is calculated by using the Putnam formulation and the transaction response time is calculated by using an operational queuing network model that can be used to calculate processing delays, queuing delays without locking conflicts and queuing delays due to locking conflicts. These dependent variables will be used in the allocation algorithm in the next chapter.

CHAPTER 6

THE GENERALIZED PROGRAM AND DATA ALLOCATION ALGORITHM

The program and data allocation problem can now be stated in its most general form and a generalized allocation algorithm can be defined which synthesizes all the extensions of the idealized model presented in chapter 2. Recall that the extensions of the idealized model in chapters 4 and 5 primarily focussed on the problem of calculating the objective function and dependent variables in terms of the transaction processing, update synchronization and physical resource parameters; without addressing the problem of optimal allocation. In this chapter, we shift our attention to the problem of optimal allocation of programs and data by integrating all the analytical expressions developed in chapters 4 and 5 into a single allocation algorithm.

6.1 The Generalized Program and Data Allocation Problem

The generalized program and data allocation problem includes application system parameters shown in table 6.1, support system parameters shown in table 6.2, and the dependent variables calculated in chapters 4 and 5. Table (6.3) shows the analytical expressions of the four major dependent variables: the local service requests (L), the remote service requests (R), the response times and the availabilities. All the variables used in the expressions have been described in tables 6.1 and 6.2. In addition, the following variables are used:

$\alpha(d,i)$ - the data allocation decision variable

$\beta(p,i)$ - the program allocation decision variable

$\beta_s(k,p,i)$ - an intermediate variable showing the program sites

= 1 if node i is the site selected for program p when a transaction arrives at node k, 0 otherwise

$\alpha_s(k,p,d,i)$ - an intermediate variable showing the data sites

= 1 if node i is the site selected for dataset d through program p, 0 otherwise

$\delta(i,j)$ - a delta function

= 1 if i=j, 0 otherwise

The parameters and expressions shown in tables 6.1, 6.2 and 6.3, are the main outputs of chapters 2, 4 and 5, and can represent a wide variety of applications and support system configurations. The generalized allocation problem is:

GIVEN:

1. The application system information which describes the following three components:
   - The database, represented by the parameters D_n, D_c and D_s.
   - The database consists of a collection of "datasets", where each dataset may be a tuple, or a relation (file).
   - The programs (operations) which manipulate the database and are represented by the parameters P_n, P_r, P_w, P_c, P_I, P_O and P_m.
   - The transactions (user requests) which activate programs and are represented by the parameters T_n, T_e and T_a.
2. The support system information which describes the following three components:
   - The query processing options, represented by the parameters Q_{da}, Q_{rt} and SRCH which show how the queries will access data, route in the network and select sites.
   - The update synchronization options, represented by U_{cp}, U_{lm} etc., which show how the database locks are set/reset and managed in the network.
   - The network characteristics like the network topology,

network control, and the availiability and service time of physical devices (links, cpus etc). This information is supplied through the A_a, B_{nc}, B_{tp} and E parameters.

DETERMINE: the 0,1 decision variables α and β where:

. $\alpha(d,i)$= 1 if dataset d is assigned to node i, 0 otherwise

. $\beta(p,i)$= 1 if program p is assigned to node i, 0 otherwise

TO MINIMIZE: the total cost

= storage cost + local processing cost + Remote Processing Cost

Where:

a.Storage cost= $\Sigma_d\Sigma_i\ \alpha(d,i)D_s(d,1)D_s(d,2)C_S(i)$ ....(6.1)

Where:

.$\alpha(d,i)$ = the data allocation decision variable

.$D_s(d,x)$ = size (x=1), in bytes, and occurrence (x=2), of tuples in dataset d

.$C_S(i)$ = Cost of storing a character at node i

b.Local processing cost= $\Sigma_t\Sigma_k\Sigma_i T_a(t,k)L(t,k,i)f1(i)C_L(i)$ .....(6.2)

Where:

.$T_a(t,k)$ = arrival rate of transaction t at node k

.L(t,k,i) = no. of local service requests (interactions between program and data) executed at node i due to the arrival of transaction t at node k.

L(t,k,i) can be expressed as:

$L(t,k,i) = L_{pd}(t,k,i) + L_{lk}(t,k,i)$

where:

$L_{pd}(t,k,i)$ = the local service requests due to the program to dataset accesses (data processing)

$L_{lk}(t,k,i)$ = the local service requests due to the lock/unlock activity for update synchronization.

See table 6.3 for analytical expressions of L

.f1(i) = conversion factor at node i which converts the L to local physical i/o. Assumed to be 1 if not specified.

TABLE (6.1) : APPLICATION SYSTEM PARAMETERS

a). DATA DESCRIPTION
(D = set of all the datasets in the application system)
. D_n = number of datasets in the application system.
. D_c(d1,d2)= consistency relationship between dataset d1 and d2; implies that d2 must be updated whenever d1 is updated
. D_s(d,x) = size (x=1) and occurrence (x=2) of tuples in dataset d.

b). PROGRAM DESCRIPTION
(P = set of all the programs in the application system)
. P_n = number of programs in the application system
. P_r(p,d) = number of reads issued from program p to dataset d.
. P_w(p,d) = number of updates issued from program p to dataset d.
. P_c(p,d) = number of consistency updates issued from program p to dataset d.
. P_b(p) = average number of input records read by program p
. P_o(p) = display messages written by program p.
. P_m(p,p1)= No. of temperory records written by program p that are to be received by by program p1. Note that p and p1 can execute in parallel if P_m(p,p1)=0.

c). TRANSACTION DESCRIPTION
(T = set of all the transactions in the application system)
. T_n = total number of transaction types in the application system.
. T_e(t,p)= no. of times program p is executed per invoc.ation of transaction t.
. T_a(t,k) = No. of times transaction t arrives at node k for a given observation period.

TABLE 6.2: SUPPORT SYSTEM PARAMETERS

a). BASIC PARAMETERS:
. N_n = No. of nodes in the network
. C_S(i) = cost of storing at node i
. C_L(i) = cost per local service request at node i
. C_R(i,j) = cost per remote service request exchanged between nodes i and j

b). TRANSACTION PROCESSING PARAMETERS
. Q_{rt} = the routing graph: parallel (1) or ring (0).
. Q_{da} = the data access mode: remote (1) or local (0).
. SRCH(k,x) = the search strategy for the queries arriving at node k.
. COPY = the maximum number of dataset copies allowed

c). UPDATE SYNCHRONIZATION OPTIONS:
. U_{cp} = location of the commit-processor: distributed (1) or centralized (0).
. U_{lm} = location of locks/stamps: distributed (1) or centralized (0).
. U_{pg} = the amount of "piggybacking" in synchronization:
= 0 complete piggyback
= 1 partial piggyback (request or response)
= 2 no piggybacking
. U_{cr} = the locks/stamps are core resident (0) or not (1)
. U_{sl} = the lock/unlock requests are broadcast (0) or addressed (1).

d). NETWORK PARAMETERS:
. A_a(z) = the availability of link z
. B_{tp}(i,j) = the network topology
= z, the communication link between nodes i and j
= 0 otherwise
. B_{nc} = the network control = 1 (distributed), 0 (centralized)
. B_{rt}(x) = routing frequencies:
x=1 - primary path
x=2 - secondary path
. E(x) = the mean service time of device x

.$C_L(i)$ = cost of processing one local physical i/o at node i

c. Remote Processing Cost =

$$\Sigma_t \Sigma_k \Sigma_i \Sigma_j T_a(t,k) R(t,k,i,j) \Sigma_z PATH(i,j,z) f2(z) C_R(z)$$

Where[1]:

.R(t,k,i,j) = no. of remote service requests (messages) exchanged between nodes i & j, when transaction t arrives at node k. To avoid additional parameters, we use R to also represent the remote transport volume, when transport volume is chosen for objective function. Also R(t,k,i,j) = 1 for flat cost. In general:

$R=(R_{np} + R_{pp} + R_{pd} + R_{lk})$ where:

R_{np} = node to program traffic - messages between the arrival node and the node where program resides

R_{pp} = program to program traffic - messages between two programs at two different nodes

R_{pd} = program to dataset traffic - messages between a program and dataset at two different nodes

R_{lk} = lock set/reset traffic - messages exchanged between nodes due to synchronization activities

See table 6.3 for analytical expressions of R

.PATH(i,j,z) = the probability that a message between nodes i and j traverses link z. PATH is a function of the support system parameters B_{tp}, B_{rt} and B_{nc}.

.f2(z) = conversion factor at link z which converts the R to a communication message. Assumed 1 if not specified.

.$C_R(z)$ = cost of exchanging one message on link z

SUBJECT TO:

1.We will use the per message remote processing cost for purpose of analysis since the transport volume cost is proportional to the per message cost and the flat cost is independent of communication traffic.

TABLE 6.3: ANALYTICAL EXPRESSIONS

1. LOCAL SERVICE REQUESTS

$$L(t,k,i) = \Sigma_p P_I(p) T_e(t,p) [L_{pd}(t,k,i) + L_{lk}(t,k,i)]$$

$L_{pd}(t,k,i)$ = local service requests due to primary read/writes

$$= P_r(p,d) a_s(p,d,i) + P_w(p,d) a(d,i) + P_c(p,d) \Sigma_{d1} D_c(d,d1) a(d1,i)$$

$L_{lk}(t,k,i)$ = local service requests, due to locking/unlocking

$$= 2U_{cr} U_{lm} (P_r(p,d) a_s(p,d,i) + P_w(p,d) a(d,i) + P_c(p,d) \Sigma_{d1} D_c(d,d1) a(d1,i))$$
$$+ (1-U_{lm}) \delta(i,z) (P_r(p,d) + P_w(p,d) + P_c(p,d) \Sigma_{d1} D_c(d,d1))$$

2. REMOTE SERVICE REQUESTS

$R(t,k,i,i) = 0$ and $R(t,k,i,j) = R(t,k,j,i)$

$$R(t,k,i,j) = \Sigma_p P_I(p) T_e(t,p) (R_{np}(t,k,i,j) + R_{pp}(t,k,i,j) + R_{pd}(t,k,i,j) + R_{lk}(t,k,i,j))$$

$R_{np}(t,k,i,j)$ = node to program remote services exchanged

$$= \delta(t,k,i) [(1+P_o(p)) Q_{rt} \beta_s(t,k,p,j) + (1-Q_{rt})(\beta_s(t,k,p0,j) + P_o(p) \beta_s(t,k,pn,j))]$$

$R_{pp}(t,k,i,j)$ = program to program remote services exchanged

$$= \Sigma_p T_e(t,p) \beta_s(k,p,i) [Q_{rt} \Sigma_{p1} \beta_s(k,p1,j) \Sigma_{k=p+1}^{p1} P_m(p,p+1) + (1-Q_{rt} \beta_s(k,p+1,j)) (\Sigma_{k=p+1}^{p2} P_m(p,p2) + \Sigma_{p=p+1}^{p1} P_I(p2) + \Sigma_{p=p+1}^{p} P_o(p2))]$$

$R_{pd}(t,k,i,j)$ = program to data traffic due to primary read/write

$$R_{pd}(t,k,i,j) = U_{cp} R'_{pd}(t,k,i,j) + (1-U_{cp}) R''_{pd}(t,k,i,j)$$

where $R'_{pd}(t,k,i,j)$ is given by:

$$= \Sigma_p \Sigma_d P_r(p,d) \beta_s(t,k,p,i) a_s(p,d,j) + \Sigma_p \Sigma_d P_w(p,d) \beta_s(t,k,p,i) a(d,j) + \Sigma_p \Sigma_d P_c(p,d) \beta_s(t,k,p,i) \Sigma_{d1} D_c(d,d1) a(d1,j)$$

and $R''_{pd}(t,k,i,j)$ is given by:

$$= [P_r(p,d) + P_w(p,d) + P_c(p,d) \Sigma_I D_c(d,d1)] \beta_s(t,k,p,i) \delta(z,j) + \delta(z,i) [P_r(p,d) a_s(p,d,j) + P_w(p,d) a(d,j) + P_c(p,d) \Sigma_I D_c(d,d1) a(d1,j)]$$

$R_{lk}(t,k,i,j)$ = lock/unlock remote traffic

$$= \Sigma_p \Sigma_d (2U_{pg} \beta_s(t,k,p,i) U_{lm} (P_r(p,d) a_s(p,d,j) + P_w(p,d) a(d,j) + P_c(p,d) \Sigma_{d1} D_c(d,d1) a(d1,j)) + (1-U_{lm}) \delta(j,z) (P_r(p,d) + P_w(p,d) + P_c(p,d) \Sigma_{d1} D_c(d,d1)))$$

3. RESPONSE TIME CALCULATIONS

$RT(t,k)$ = Response time of transaction t when it arrives at node k

$$= PD(t,k) + QD(t,k)$$

Where PD, processing delay, and QD, queidng delay, are given below.

a). $PD(t,k)$ = processing delay of transaction t when it arrives at node k

$$= \Sigma_g ws(t,k,g) + \Sigma_{g'} wp(t,k,g')$$

$ws(t,k,g)$ = processing delay for serial group g

$$= \Sigma_p pd(t,k,p)$$

$wp(t,k,g')$ = processing delay due to parallel group g

$$= \max [pd(t,k,p)]$$

b). $QD(t,k)$ = queuing delay encountered by transaction t arriving at k

$$= \Sigma_y L(t,k,y)\, qd(y)$$

$$qd(y) = E(y) \left[\left\{ \sum_{n=0}^{NT} (V(y)E(y))^n GG(NT-n, NY)/GG(NT, NY) \right\} - 1 \right]$$

where $E(y)$ = mean time to service a request by device y (input parameter)

$V(y)$ = visit ratio of device y $= \Sigma_t \Sigma_k \lambda(k) L(t,k,y)$

GG = computation matrix, depends on V and E,

NT = the maximum no. of concurrent transactions in the system

NY = the maximum number of device

c). Locking Conflicts : need to know $E_c(i)$ and $V_c(i)$ to compute the queuing delays $qd(i)$ at the conflict device i

$E_c(i)$ = service time of the conflict device i

$$= [E(i) + E/2]$$

$E(i)$ = service time of the conflict device

E = given in chapter 5.

$V_c(i)$ = visit ratio of conflict device i (given in chapter 5)

4. TRANSACTION AVAILABILITY

$A(t,k)$ = availability of transaction t when it arrives at node k

$$= \Pi_p \Pi_d A_p(k,p) A_d(p,d)$$

where A_p, the program availability, and A_d, the data availability, are:

$$A_p(t,k,p) = 1 - \Pi_i [1 - r(t,k,i) \beta(p,i)]$$

$$A_d(p,d) = A_d(j,d) = 1 - \Pi_i [1 - r(j,i) a(d,i)]$$

1). The storage constraints - the storage occupied at node i must not exceed the storage capacity of node i, STOR(i):

$\Sigma_d \alpha(d,i)\, D_s(d,1)\, D_s(d,2) \leq STOR(i)$ for all i .....(6.4)

2). Transaction availability constraints - the availability of transaction t must not be less than AV(t), a user specified limit:

$\Sigma_k T_a(t,k)\, A(t,k) / \Sigma_k T_a(t,k) \geq AV(t)$ for transaction t ...(6.5)

where:

$T_a(t,k)$ = arrival rate of transaction t at node k

A(t,k) = availability of transaction t when it arrives at node k. This dependent variable is a function of the decision variables α and β and the support system variables A_a, B_{tp}, B_{nc} and B_{rt}. See table 6.3 for analytical expression of A.

3). Transaction response time constraint- the average response time of transaction t must not exceed DEL(t), a user limit:

$\Sigma_k T_a(t,k)\, RT(t,k) / \Sigma_k T_a(t,k) \leq DEL(t)$ for transaction t (6.6)

where:

RT(t,k) = total response time of transaction t when it arrives at node k. This dependent variable includes the effect of processing delays, device-busy queuing delays and locking conflicts. Thus RT(t,k) is a function of the dependent variables L & R and the support system parameter E. See table 6.3 for the analytical expression RT(t,k).

4). Data allocation constraints - Each dataset must be allocated to at least one node, but may be restricted to a few nodes due to policy and/or security considerations:

$\Sigma_i \alpha(d,i) \geq 1$ for all datasets d .....(6.7)

$F_d(d,i) - \alpha(d,i) \geq 0$ for all d & i .....(6.8)

Where $F_d(d,i) = 1$ if dataset d can be allocated to node i, 0 otherwise. This matrix reflects the policies etc.

5). Program allocation constraints - Each program must be allocated to at least one node, but may be restricted to a few nodes due to the technical (processor not suitable for running the program) and/or program duplication cost constraints:

$\Sigma_i \beta(p,i) \geq 1$ for all programs p .....(6.9)

$F_p(p,i) - \beta(p,i) \geq 0$ for all p & i .....(6.10)

$\Sigma_p C_{ap}(p,i) \beta(p,i) \leq PCOST$ .....(6.11)

where:

$F_p(p,i)$ = 1 if program p can be assigned to node i, 0 otherwise. This matrix shows technical constraints

$C_{ap}(p,i)$ = cost of assigning program p at node i. This parameter can be an input parameter or can be computed by using equations 5.21 and 5.25, chap 5.

PCOST = program duplication cost limit

6.2 The Problem Analysis and Solution Approach

Substitution of the analytical expressions for dependent variables like the local service requests L(t,k,i), remote service requests R(t,k,i,j), the transaction availaibility A(t,k) and the transaction total response time RT(t,k) in the generalized allocation problem yields an integer programming problem with nonlinear objective function and nonlinear constraints with the following additional properties:

1. For x programs, y datasets and n nodes, explicit enumeration requires $2^{(x+y)n}$ evaluations to determine optimal allocation.
2. Interrelationships exist between data assignments and program assignments which make objective function non-separable. The data relationships exist due to the consistency relationships, the D_c terms in the L_{pd}, L_{lk}, R_{pd} and R_{lk} variables, and the program relationships are due to the information exchange between programs

represented by the parameter P_m in the R_{pp} expression in table 6.3. Fortunately, for most practical applications, these relationships are sparse, i.e. only few datasets and programs are interrelated. We utilize this observation to achieve approximate decomposition by sequencing the datasets and programs so that related objects are handled at the last stage (see section 6.3.2 for discussion).

3. The behavior of the objective function and the constraints is heavily dependent upon the support system and application system variables. In other words, different application systems and support system configurations can lead to different characteristics of the problem. For example, simple application systems in idealized support systems yield unconstrained nonlinear integer programming problems with convex and separable objective function (section 2.6, chapter 2). However, for more complex situations the behaviour of the objective function changes (section 4.6, chapter 4).
4. Despite variations, the objective function displays convexity in terms of data replication for a large number of cases (see sections 2.6 and 4.6 for discussion).
5. The problem is NP complete, since simplified versions (reductions) of this problem have been shown to be NP complete <ESWA74c, FISH80c>.

This problem differs from the unconstrained allocation problem, presented in chapter 2, in several respects. First, the effect of transaction processing and update synchronization is included in the objective function which can affect the behaviour as well as the cost of allocating data and programs. Second, several constraints have been introduced. Some of the constraints are linear, for example the storage constraint, while others are nonlinear. Third, the decompositions of

the unconstrained model may or may not be necessarily true. Finally, calculation of the objective function, the response time and the availaibility are expensive for each allocation decision. Thus any technique which minimizes the number of decisions before reaching an optimum has significant impact on the optimization algorithm.

Nonlinear integer programming problems of this nature fall into the category where most traditional optimization techniques fail (see for example <HOLM81f, GARE79f>). The solution approach selected must be flexible enough so that as the nature of the problem changes due to different input parameters, new models and procedures are not needed. It is also desirable to solve this problem in polynomial time. The algorithm presented in this chapter attempts to satisfy these requirements and is an extension of the unconstrained allocation algorithm presented in chapter 2. An overview of the algorithm is presented here and the details are discussed in next section.

The problem is solved in a hierarchical manner in three steps. First, the programs and datasets are clustered into disjoint groups by using a grouping algorithm. The objective of this algorithm is to reduce the size of the allocation task by reducing the number of datasets and programs to be assigned. The grouping scheme is a straightforward extension of the grouping scheme presented in chapter 2. The resulting program and data allocation problem is decomposed into program and dataset subproblems which are solved in the next two steps. The proof of this decomposition is given in appendix E and is based on two assumptions: the queuing delays are ignored and the query processing first chooses program sites and then data sites based on the program sites just selected. Hevner <HEVN81d> shows that a large number of existing systems use this technique. If this assumption is not true, then the decomposition is not possible. However, in such cases, we assume that programs can only access local data. Due to this assumption, the programs must be, at least, allocated to the nodes where the data is allo-

cated; thus the data allocation algorithm automatically performs the program allocation.

The program allocation problem is a nonlinear integer programming problem with a non-separable objective function but separable constraints (see 6.3.3 for details). It is important to solve this problem since program allocation affects software duplication and communication costs and is subject to considerations of processor capabilities like floating point arithmetic, graphics etc. A spatial dynamic programming algorithm is employed to solve this problem. This technique is an extension of the non-serial dynamic programming introduced by Nemhauser <NEMH66f> and has been recently pioneered by Larson & McIntyre <LARS82f, McENT81f, DOTY82f>.

The data allocation problem is also an integer programming problem with nonlinear objective function and nonlinear constraints. However, the objective function and the constraints are non-separable. Most existing techniques fail to solve such problems. However, we employ an ordering technique to achieve separability and use other approximations for decomposition. Appendix F and G show the proof of decomposing the data allocation problem. A hierarchical heuristic technique is employed to allocate each dataset.

As can be seen, several approximations are used by the allocation algorithm. The effect of these approximations is analyzed by using theoretical and empirical methods in section 6.5.

6.3 The Generalized Allocation Algorithm

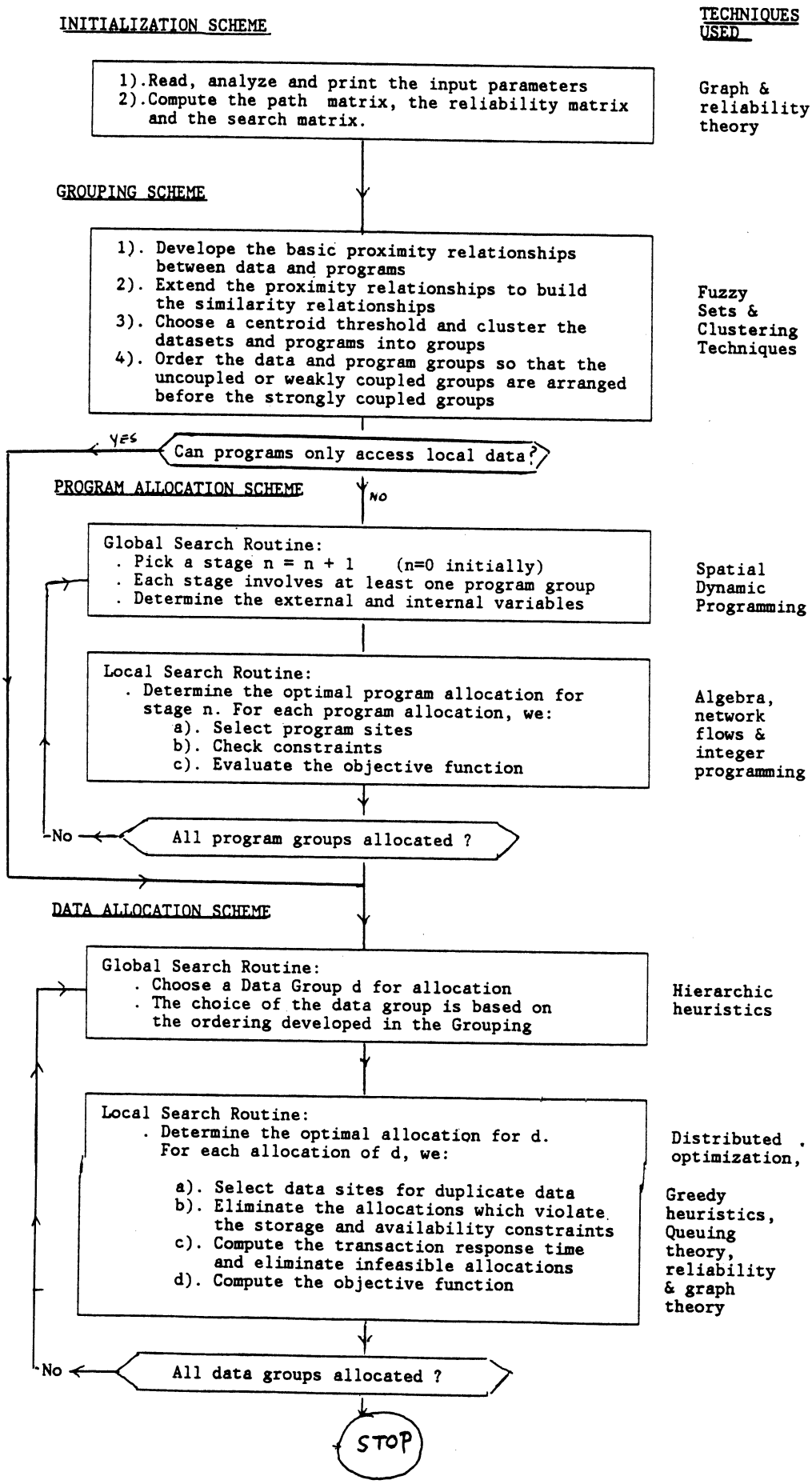
Fig (6.1) shows the overall flow of the allocation algorithm and the various techniques used in the algorithm. As can be seen, the algorithm consists of the initialization, grouping, program allocation and data allocation schemes. This algorithm differs from the unconstrained allocation algorithm in several respects. First, the initialization

scheme is added to compute the matrices which are used later on in the algorithm. Second, the grouping scheme is extended to include the local data access and update synchronization. Third, the program allocation scheme is introduced because program allocation was a trivial problem for the unconstrained model. Finally, the data allocation scheme is extended to handle constraints.

6.3.1 The Initialization Scheme

This subroutine reads the input parameters, initializes several variables and computes the following matrices for subsequent routines: 1). The "path" matrix PATH(i,j,z) which shows the probability that a message exchanged between nodes i and j traverses the link z. This matrix is computed from the input parameters showing the network topology (B_{tp}), routing frequencies (B_{rt}) and network control (B_{nc}). The algorithm employs the standard labelling techniques <DAVI79d> to compute the path matrix (see section 5.1.2 for details). 2). The reliability matrix r(k,i) which shows the reliability of the path between nodes k and i. This matrix is computed from the PATH matrix and the input parameter $A_a(z)$ which shows the availability of the link z. Computation of this matrix utilizes the reliability theory <MEYE78f> and is discussed in section 5.2. 3). The "search" matrix SRCH(i,x) which shows the order in which the nodes will be accessed from node i (x=1 is the first node, x=2 is the second node etc). This matrix can be given as an input parameter or can be computed based on the PATH matrix and other parameters which shows search criteria (minimum cost, minimum distance, minimum spanning tree), and is used very heavily in the site selection for programs and datasets. The calculation can utilize the network flow theory <JOHN78f>. 4). The cost of developing program p in language s, C_{dp}, can be calculted by using equation 5.25 in chapter 5. This value can be also provided as an input parameter.

FIG (6.1): OPTIMAL PROGRAM/DATA ALLOCATION ALGORITHM



6.3.2 The Grouping Scheme

The grouping scheme clusters programs and datasets to reduce the size of the optimization and is based on clustering techniques (see for example <EVER74f, DURA74f>). This scheme was introduced in chapter 2 and is now extended to integrate all the support system factors introduced in chapters 4 and 5.

The basic procedure consists of choosing and computing an appropriate similarity function based on fuzzy sets and then utilizing hierarchical clustering techniques for "fusing" similar objects. The grouping scheme consists of the following four steps (see fig 6.1).

STEP 1).A proximity relationship is developed which shows the commonality between various programs and datasets. The following proximity relationship q(p,p1) between programs p & p1 was introduced in chapter 2:

$$q(p,p1) = \frac{\Sigma_d(\min(Z(p,d),Z(p1,d)))+\min_t(T_e(t,p),T_e(t,p1)) + P_m(p,p1)}{\Sigma_d (Z(p,d) + Z(p1,d)) +\Sigma_t T_e(t,p)+T_e(t,p1) + P_m(p,p1)}$$

where $Z(p,d)$ = no. of accesses from program p to dataset d $=(P_r(p,d) + P_w(p,d))$ and P_r, P_w, P_m and T_e are application system parameters (table 6.1). The proximity relationship q'(d,d1) between datasets d & d1 can be expressed as:

$$q'(d,d1) = \frac{\Sigma_p (\min (Z(p,d), Z(p,d1)) + K \Sigma_p P_c(p,d)D_c(d,d1)}{\Sigma_p (Z(p,d) + Z(p,d1)) + K \Sigma_p P_c(p,d)D_c(d,d1)}$$

where $D_c(d,d1)$ shows the consistency relationships between datasets d & d1. Note that this q' differs from the q' presented in chapter 2 due to the introduction of the factor K which increases clustering for consistency relationships and update synchronization, where $K \geq 1$. If $K = 1$, then the data proximity relationships q' of the unconstrained and extended algorithm are the same.

STEP 2).The proximity relationships are extended to include the effect of model assumptions and restrictions and then the similarity matrices are calculated. This step is affected by the transaction processing options of remote and local data access strategies.

In case of a local data access, the datasets d1 and d2, accessed by a program p can be clustered togeteher since p and d1 must be at the same node and also p and d2 must be at the same node; thus d1 and d2 must exist at the same node. It is possible that the local data access rule can result in clustering of all the datasets together due to cascading (if d1 and d2 are accessed by p1 and d2 and d3 are accessed by p2, then d1, d2 and d3 are clustered together). This rule however cannot be used for remote data access. In addition, all programs can be clustered together if programs are to be replicated. These modifications of the proximity relationships can be easily incorporated into the grouping scheme by inserting the following simple algorithm in calculating the proximity relationships:

. If programs are replicated, then q(p,p1) = 1 for
  all p, p1 Ξ P, else compute q(p,p1) from equation 2.9.
. If local data access, then q'(d,d1) = 1 for d,d1 accessed
  by same program, else compute q'(d,d1) from equation 2.11.

The balance of this step (the computation of similarity relationships) is unaffected and is the same as described in chapter 2.

STEP 3). Once a similarity matrix has been obtained, then a hierarchical clustering mechanism can be built which forms a cluster between two "nearest" objects based on similarity, then the next most similar object joins the cluster etc, till all the objects are grouped into a single cluster. A stopping rule, threshold, is usually needed so that all the objects with similarity greater than the threshold are grouped. This step is essentially the same as described in the unconstrained allocation algorithm.

STEP 4). This scheme also orders/sequences the programs and datasets to achieve approximate separability of objective functions when program and dataset relationships exist. This step employs greedy heuristics and consists of two routines: data sequencing and program sequencing. The purpose of the data sequencing routine is to order the datasets in such a fashion so that the datasets with no consistency relationships are allocated first. This routine is used by the data allocation scheme to be described below (section 6.3.4) and basically "labels" ND dataset groups (d, d', d", ,,) as (d1, d2, d3,), where d1 is the first group to be allocated, d2 is the next and so on. The following algorithm is used for dataset ordering:

1.To determine d1:

. Choose a data group d such that $D_c(d,d')=0$ for all d'.

. If none, then choose a d for which $P_c(p,d) = 0$ for all p.

. If none, then choose a d for which $D_c(d,d') \neq 0$ for all d' that have been already ordered.

. If not possible, then choose d for which $P_c(p,d')$ is small for all d' with $D_c(d,d') \neq 0$.

. Weight the groups by workload, i.e. the groups with large number of accesses are ordered before the ones with less activity.

2. Repeat the same procedure for d2, d3, ,,, etc.

The purpose of the program sequencing routine is to order the programs so that the program with no inter-program relationships is allocated first. This sequencing routine is similar to the data sequencing routine and is used in program allocation scheme (section 6.4.3) to obtain decomposition when the weak-interprogram approximation is not applicable. The routine labels the program groups (p,p',p",,,) as (p1,p2,p3,,,) where p1 is the first group to be allocated, p2 is the next and so on:

1. To determine p1:

. Choose a p such that $P_m(p,p') = 0$ for all p1.

. If none, then choose a p for which $P_m(p,p') \neq 0$ for p' in stages n-1, n-2,,,,,2,1.

(We can assume that program allocation is also performed in several stages. We will discuss it in more detail later).

2. Repeat the same procepure for p2, p3 etc.

6.3.3 The Program Allocation Scheme

This scheme attempts to solve the optimal program allocation subproblem which is concerned with the allocation of programs to nodes to minimize the cost of running programs subject to the software duplication costs, availability and response time constraints. This scheme is employed by the allocation algorithm when the combined program and data allocation problem can be decomposed into program allocation and data allocation subproblems. This is true when transaction processing chooses program sites first and then data sites based on the program sites selected (see appendix E). Else, the program allocation is conducted autcmatically by the data allocation scheme (see the data allocation scheme below).

Most of the literature on resource allocations in DDP ignores the program allocation problem by assuming that the programs are replicated. We feel that program replication is not a suitable approach due to three reasons. First, some processors are more suited for certain type of applications due to speed and special hardware features. Second, cost of replicating programs in real life systems is prohibitively large due to the software development and maintenance required for many copies and versions of programs and finally, it is better to allocate programs to the nodes so that workload is balanced. Due to these reasons, several researchers have exclusively studied the problem of optimal program allocations <GYLY76c, BRYA81c, McENT81c, DOTY82c>.

6.3.3.1 The Problem Statement

GIVEN: the application system and support system parameters in tables (6.1) and (6.2).

DETERMINE: the program allocations $\beta(p,i)$

TO MINIMIZE: the (node to program+ program to program) communication cost

$$\Sigma_t\Sigma_k\Sigma_i\Sigma_j T_a(t,k)[R_{np}(t,k,i,j)+R_{pp}(t,k,i,j)]\Sigma_z PATH(i,j,z)f2(z)C_R(z). \quad (6.41)$$

where:

$R_{np}(t,k,i,j)$ and $R_{pp}(t,k,i,j)$ = the no. of remote node-to-program and program-to-program messages, respectively, exchanged between nodes i & j due to arrival of a trans. at node k (see figure 6.2 for the analysis of R_{np} & R_{pp}).

$T_a(t,k)$, PATH and C_R have been defined previously (tables 6.1 & 6.2)

SUBJECT TO:

1). Program allocation constraints defined in section 2:

a). $\Sigma_i\ \beta(p,i) \geq 1$ for all programs p ..... (6.42)

b). $F_p(p,i) - \beta(p,i) \geq 0$ for all p & i ..... (6.43)

c). $\Sigma_p\ \Sigma_i\ C_{ap}(p,i)\ \beta(p,i) \leq PCOST$ ..... (6.44)

where:

$F_p(p,i)$ = 1 if program p can be assigned to node i.
0 otherwise.

$C_{ap}(p,i)$ = cost of assigning program p at node i.

2). The "surrogate" constraints of response time and availability:

a). The processing delay, for transaction t, due to node to program, program to program and program execution must not exceed the transaction response time, DEL(t):

$$\Sigma_k\ \Sigma_i\ \Sigma_j\ T_a(t,k)(R_{np}(t,k,i,j)+R_{pp}(t,k,i,j))\ \Sigma_z PATH(i,j,z)E(z) + \Sigma_p\ \Sigma_d\ (P_r(p,d)+P_w(p,d))\Sigma_i\beta_s(k,p,i)E(i) \leq DEL(t) \quad ...(6.45)$$

where:

$E(z)$ = expected service time at device z

$\beta_s(k,p,i)$= 1 if i is the site selected for program p when transaction t arrives at node k

NOTE: The first two terms of this expression are similar to the objective function. The third term shows the program execution time expressed in terms of the read/write activity of each program.

b). The availability of each program must not be less than the availability of utilizing transaction, AV:

$$1-\Pi_{k,p,i}\ (1-r(k,i)\beta(p,i)) \geq AV(t) \quad \ldots\ldots\ldots \quad (6.46)$$

where $r(k,i)$ = availability of path between nodes i & j

6.3.3.2 Problem Analysis

The dependent variables R_{np} and R_{pp} play key role in the program allocation problem. The analytical expressions for these variables are given and discussed in section 6.2. Figure (6.2) illustrates R_{np} and R_{pp} through an example and shows how these variables are affected by program allocations and query processing options. Substitution of the analytical expressions for the dependent variables R_{np} and R_{pp} in the optimal program allocation problem stated above, yields an integer programming problem with the following characteristics. 1). The objective function is nonlinear. The nonlinearity is due to the program-to-program traffic R_{pp}. 2). The objective function is minimized if programs are 100% replicated, i.e. if all programs are allocated to all the nodes. This is intuitively obvious since in such cases no remote messages are sent from nodes to programs and from programs to programs. 3). The objective function is not separable in terms of the program allocations β and the program sites β_s. 4). All the program allocation constraints are separable for each program allocation. 5). The delay constraint is very similar to the objective function. This similarity suggests a Lagrangian type approach. 6). The availability constraint is

separable in terms of program allocation. We build the following Lagrangian objective function:

$$OB' = OB[R_{np},R_{pp},C_R] + M1[PD] + M2[PR] \quad \ldots\ldots\ldots\ldots\ldots (6.47)$$

where OB is the original objective function shown above in eq. 6.41 and M1 and M2 are the Lagrange multipliers for the delay and replication constraints:

PD = processing delay used in equation 6.45

$$= \Sigma_k \Sigma_i \Sigma_j T_a(t,k)(R_{np}(t,k,i,j)+R_{pp}(t,k,i,j)) \Sigma_z PATH(i,j,z)E(z)$$
$$+ \Sigma_p \Sigma_p (P_r(p,d)+P_w(p,d)) \Sigma_k \Sigma_i \beta_s(k,p,i)E(i)$$

RP = program replication cost given by equation 6.44

$$= \Sigma_p \Sigma_d C_{ap}(p,i)\ \beta(p,i)$$

The reason for the inclusion of the processing delay into the objective function is the obvious similarity. The replication cost is included in the objective function to make the objective function roughly convex (the replication cost increases with program replication).

6.3.3.3 The Program Allocation Algorithm

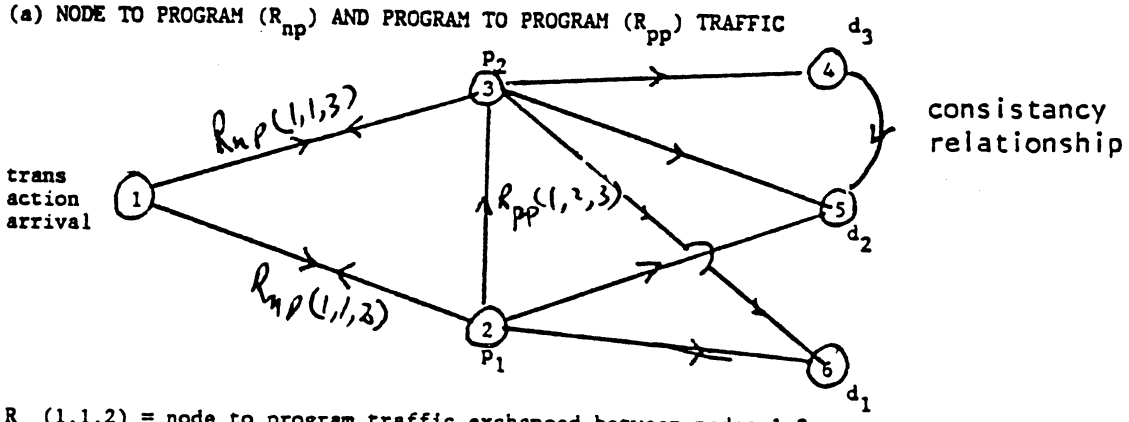
As shown in fig (6.1), this algorithm consists of two major subroutines: a global subroutine which chooses the programs to be allocated and a local subroutine which attempts to optimally assign the chosen programs. Recall that the program allocation algorithm is called only if remote data access is assumed. In case of local data access, the program allocation algorithm is bypassed and the data llocation scheme discussed below automatically allocates the programs to the same nodes where the data has been allocated.

The GLOBAL SUBROUTINE is based on spatial dynamic programming. This generalization of standard dynamic programming utilizes spatial decomposition of sparse networks and has been pioneered by Larson & McEntyre <LARS82f, McENT81c>. The basis of the spatial dynamic programming is that a mathematical programming problem with nonseparable objective function but with separable constraints can be solved rigorously by us-

FIG (6-2): AN ILLUSTRATION OF PROGRAM ALLOCATION

Consider the example discussed previously.
Assume that the two programs p1 & p2 and the three datasets d1, d2, d3 are allocated to a 5 node network as shown in fig (7.3a).
In fig (7.3a) the node to program & the program to program traffic is illustrated.
In figures (7.3b & c) the variation of these activities due to replication of p1 (restricting p2 to node 3) is shown.

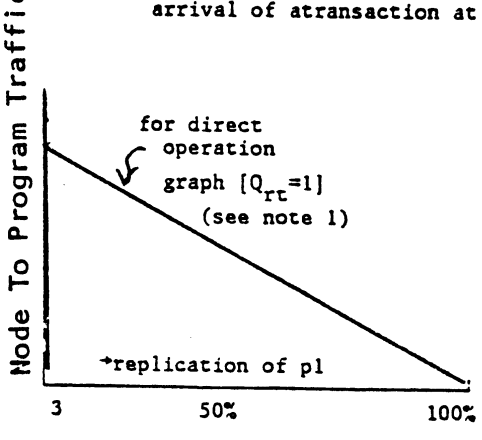
(a) NODE TO PROGRAM (R_{np}) AND PROGRAM TO PROGRAM (R_{pp}) TRAFFIC



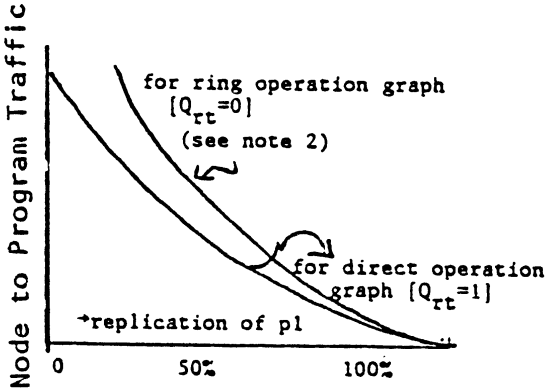
$R_{np}(1,1,2)$ = node to program traffic exchanged between nodes 1,2 due to arrival of a transaction at node 1.

$R_{np}(1,1,3)$= node to program traffic exchanged between node 1,3 due to arrival of a transaction at node 1.

$R_{pp}(1,2,3)$= program to program traffic between node 2,3 due to arrival of atransaction at node 1.



(b) Node to program traffic due to replication of p1



(c) Program to program traffic due to replication of p1

NOTE 1 = In a direct operation graph, the arrival node directly sends/receives the program input/output. Thus, in fig (4.3a) p1 at node 2, sends and receives i/o directly from node 1 and p2 at node 2 sends and receives i/o directly from node 1. Thus

$R_{np}(1,1,2)$= input to p1 + output from p1 = $P_i(1)+P_o(1)$

$R_{np}(1,1,3)$= input to p2 + output from p2 = $P_i(2) + P_o(2)$

$R_{pp}(1,2,3)$= temporary messages from p1 to p2=$P_m(1,2)$

NOTE 2 = In a ring operation, the input/output of the program is circulated. Thus program p1 receives the input for p1 + p2, sends, from node 2 to 3, the output from p1 etc. Thus

$R_{np}(1,1,2)$= $P_i(1) + P_i(2)$

$R_{np}(1,1,3)$= $P_o(1) + P_o(2)$

$R_{pp}(1,2,3)$= $P_m(1,2) + P_i(2) + P_o(1)$

ing a dynamic programming framework. In this method a problem is decomposed into several subsystems (units) for which local objective functions are computed. Variables are used to represent interactions between the units. The technique calls for the units to be considered sequentially, and, as each of the units is considered, it becomes part of the "internal" set of units whereas those remaining consititue "external" units. In each stage, only variables relating a given unit to the associated external units are applied. The optimal values of variables relating the unit to other internal units are found.

Fig (6.3) illustrates the spatial dynamic programming formulation for a problem of 2 program allocation to 3 nodes. As can be seen, the problem is subdivided into 5 units (stages). The first unit consists of program p1, the second unit consists of node n1, the third unit consists of program p2 etc. The external, internal and decision variables are formally defined and illustrated below:

X_{pi} = the decision variable - shows allocation of program p to node i.

E_n = the external variables at stage n - the decision variables at stage n which cross over to stages (n+1, n+2, ,,, N)

I_n = the internal variables at stage n - the decision variables at stage n which terminate at stage n

$f(E_n, I_n)$ = objective function at stage n

$J(E_n)$ = optimal return at stage n

The basic recursive equation (assuming additive return) is:

$$J(E_n) = \text{Min } (f(E_n, I_n) + J(E_{n-1})) \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots (6.48)$$

The following relationships illustrate the variables and recursive equations:

Stage 1: $E_1 = (X_{11}, X_{12}, X_{13})$, $I_1 = (0)$, $J(E_1) = \text{Min } (f(E_1, I_1))$

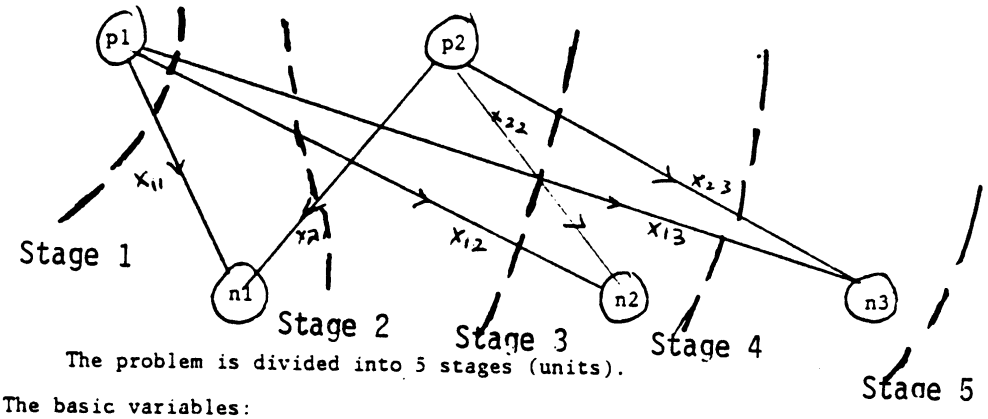
Stage 2: $E_2 = (X_{12}, X_{13}, X_{21})$, $I_2 = (X_{11})$, $J(E_2) = \text{Min } (f(E_2, I_2) + J(E_1))$

Stage 3: $E_3 = (X_{12}, X_{13}, X_{22}, X_{23})$, $I_3 = (X_{21})$, $J(E_3) = \text{Min } (f(E_3, I_3) + J(E_2))$

Stage 4: $E_4 = (X_{13}, X_{23})$, $I_4 = (X_{22}, X_{12})$, $J(E_4) = \text{Min } (f(E_4, I_4) + J(E_3))$

FIGURE (6.3): SPATIAL DYNAMIC PROGRAMMING

Consider allocation of two programs (p1, p2) to 3 nodes (n1,n2,n3).
Let X_{ij} show the allocation of program i to node j.



The problem is divided into 5 stages (units).

The basic variables:

E_n = the external variables at stage n - the decision variables at stage n which cross over to stages (n+1, n+2, ,,, N)

I_n = the internal variables at stage n - the decision variables at stage n which terminate at stage n

$f(E_n, I_n)$ = objective function at stage n

$J(E_n)$ = optimal return at stage n

Stage 5: $E_5 = (0)$, $I_5=(X_{13}, X_{23})$, $J(E_5) = \text{Min } (f(E_5,I_5)+J(E_5))$

A given problem can be formulated in a wide variety of ways by using spatial dynamic programming. For example, we have chosen the decision variable X_{pi} to denote the allocation of each program p to node i. Alternatively, we could have chosen the decision variable Y_{pi}, which shows the allocation of program p to the successors of node i, where a successor of node i is a node j which is on a vertical path from i, in a cost hypercube. Similarly, we could have chosen to consider more than one program or node to be a single stage. However, the scheme implemented in this research uses X_{pi} as the decision variable and each program and node is considered as a stage. The following algorithm is used by the global program allocation routine:

1. Arrange the stages in the following manner: p1, n1, p2, n2, p3, n3, p4, n4, n5, n6 etc. where p1 is the first program and n1 is the

first node. The programs have been sequenced by using the program sequencing routine described above (section 6.3.2). In a problem with NP programs and NN nodes, the number of stages NS = NP + NN.

2. Each stage is classified into a 'program' or 'node' stage by using a program indicator.

3. The set of internal variables I_1, I_2,,,I_{NS} at each stage are determined by using two vectors: PLIST and NLIST which show the programs and nodes which constitue the internal variables at that stage, respectively. For example, for the two program, three node example considered above, the NLIST and PLIST vectors consist of the following values at stage 4: PLIST = (p2, p1) and NLIST = (n2).

4. Build the internal variables at each stage from PLIST and NLIST. For example, for the above example, the internal variables at stage 4 are given by: $I_4 = (X_{22}, X_{12})$.

5. Similar procedure is used to determine the external variables at each stage. Basically, at each stage, one or more external variables are moved to the internal variable set. The PLIST, NLIST, the stage type and the calculated internal variables are used to calculate the external variables.

The global scheme basically determines the internal and external variables at each stage. These variables are used in the local subroutine for allocation since these variables determine $J(E_n)$ as shown by the basic recursive equation (6.48).

The LOCAL SUBROUTINE performs the local optimization at each stage of the spatial dynamic programming problem. Specifically, this subroutine determines $J(E_n)$ at stage n and consists of the following steps:

1. Start with the variables E_n and I_n at stage n generated by the global routine. Each I_n and E_n consists of a set of X_{pi}. Note

that basically $X_{pi} = \beta(p,i)$, the program allocation decision variable. However we use X_{pi} in the algorithm instead of the decision variable $\beta(p,i)$ for sake of generality.

2. Select the site $\beta_s(k,p,i)$ for program p for all k & i. This site selection is based on the search matrix SRCH and the program allocations $\beta(p,i)$. For example, if program p is allocated to node i, and node i is the nearest node to the transaction arrival node k, then $\beta_s(k,p,i)=1$.

3. Check the program allocation (equations 6.42 and 6.43) and availability (equation 6.46). If violated, go back for next X_{pi}.

4. Evaluate the objective function $f(E_n, I_n)$ based on the external and internal variables E_n and I_n. Recall that f actually represents the Lagrangian OB' shown in equation 6.47. This Lagrangian includes the delay (equation 6.45) and program duplication (equation 6.44) constraints.

5. Determine the optimal return $J(E_n)$ at stage n by optimizing over the internal variables I_n. The values for I_n and E_n are generated by travelling the cost hypercube in a reverse direction, i.e. we start with all $X_{pi} = 1$ and get next value of X_{pi} by removing p, i.e. consider $X_{pi} = 0$. This allows an unconstrained program allocation to terminate quickly because program replication is optimal for unconstrained problems.

6. Determine $J(E_n)$ at all stages and then stop.

Note that we also need to choose proper values of the Lagrangians. The local subroutine uses the following theorem proposed by Everret <EVER63f> to choose and adjust the values of the Lagrange multipliers M1 and M2.

THEOREM: Consider the problem:

minimize OB(β) where β is the set of program allocations

subject to: $PD(t,\beta) \leq DEL(t)$ for all transactions $t \in T$

$RP(\beta) \leq PCOST$

Then β^* which minimizes Lagrangian:

$OB'(\beta) = OB(\beta) + M1 \times PD(t,\beta) + M2 \times RP(\beta)$

where M1 and M2 are real nonnegative Lagrange multipliers, is a GLOBAL minimum to the original problem if:

$PD(t,\beta^*) = DEL(t)$

$RP(\beta^*) = PCOST$ (END THEOREM)

In order to use this theorem, we first choose initial values of M1 and M2 and minimize OB' to determine β^*. Now if $PD(t,\beta^*) = DEL(t)$ and $RP(\beta^*) = PCOST$, then stop, else choose other values of M1 and M2 and reiterate. The main strength of this procedure is that it is applicable to nondifferentiable objective functions and arbitrary constraints. We use the following heuristic to find the values of M1 and M2:

1. Set M1=M2=0.
2. Determine β^* which minimizes OB subject to the program allocation constraints of policy restrictions and at least one copy of program per node.
3. Evaluate $PD(t,\beta^*)$ and $RP(\beta^*)$.
4. Check if $PD(t,\beta^*) \leq DEL(t)$ and $RP(\beta^*) \leq PCOST$. If so, then stop, else increase M1 and M2 and reiterate. Since every iteration is expensive, we use the following increasing values of M1 and M2: 0, 1, 10, 100, 1000, 10000 etc. These values, or variants thereof, are adequate for approximate analysis. However, a more rigorous technique may be needed if a more accurate solution is desired.

Note that this heuristic terminates if the processing delays and program costs are less than DEL(t) and PCOST. Thus this heuristic produces a pessimistic solution.

6.3.4 The Data Allocation Scheme

This scheme attempts to determine the optimal data allocations $\alpha(d,i)$ subject to the program allocations $\beta(p,i)$ determined by the program allocation scheme described above. If the allocation problem is not decomposable into program and data allocation subproblems as discussed above, then this algorithm also allocates programs to the same nodes where the datasets are allocated. This is because we assume in such cases that programs can only access local data, so the accessing programs must be housed in the same node where the data exists.

6.3.4.1 Problem Statement

GIVEN: the application and support system parameters shown in tables 6.1 & 6.2.

DETERMINE: the data allocations $\alpha(d,i)$

TO MINIMIZE: Storage cost + Local processing cost + Communication cost

$$= \Sigma_d \Sigma_i \alpha(d,i) D_s(d) C_S(i) + \Sigma_t \Sigma_k \Sigma_i T_a(t,k) L(t,k,i) C_L(i) f1(i)$$
$$+\Sigma_t \Sigma_k \Sigma_i \Sigma_j T_a(t,k)\{R_{pd}(t,k,i,j)+R_{lk}(t,k,i,j)\}\Sigma_z PATH(i,j,z) C_R(z) f2(z)$$

Where $R_{pd}(t,k,i,j)$ and $R_{lk}(t,k,i,j)$ =

the no. of remote program-to-dataset messages and the lock-unlock synchronization requests exchanged between nodes i & j when transaction t arrives at node k (see section 6.2 for the analytical expression of R_{pd} and R_{lk}).

All other parameters have been defined in tables 6.1 and 6.2.

SUBJECT TO:

1). The program allocations $\beta(p,i)$ determined by the algorithm described in section 6.4.3.3.

2). The storage constraints defined in section 2:

$$\Sigma_d \alpha(d,i) D_s(d) \leq STOR(i) \quad \text{for all } i \quad \ldots\ldots\ldots\ldots \quad (6.51)$$

3). Transaction availability constraints defined in section 2:

$$\Sigma_k T_a(t,k) A(t,k) / \Sigma_k T_a(t,k) \geq AV(t) \text{ for transaction } t \ldots (6.52)$$

where:

$T_a(t,k)$ = arrival rate of transaction t at node k

$A(t,k)$ = availability of a transaction when it arrives at node k.

4). Transaction response time constraint defined in section 2:

$\Sigma_k \; T_a(t,k) \; RT(t,k) / \; \Sigma_i T_a(t,k) \leq DEL(t)$ for transaction t ..(6.53)

where:

$RT(t,k)$ = total response time of transaction t when it arrives at node k.

5). Data allocation constraints defined in section 2:

$\Sigma_i \; \alpha(d,i) \geq 1$ for all datasets d ..... (6.54)

$F_d(d,i) - \alpha(d,i) \geq 0$ for all d & i ..... (6.55)

Where $F_d(d,i) = 1$ if dataset d can be allocated to node i, 0 otherwise. This matrix reflects the policies etc.

6.3.4.2 Problem Analysis

For the purpose of analysis, let us first ignore all the constraints, i.e. consider an unconstrained problem. The objective function is a nonlinear integer function. Thus the unconstrained problem can be theoretically solved by using standard linearizing techniques <PLAN71f, MODER78f>, however a prohibitively large number of constraints can result from this technique. It has been shown in chapter 2 that, for an idealized support system, this problem can be decomposed into g independent subproblems, where g is the total no. of data groups generated by the grouping scheme, if the consistency updates can be ignored (the weak-consistency approximation). It is shown in appendix F that, due to the weak-consistency approximation, this problem is still decomposable even after the transaction processing and update synchronization costs are included in the objective function. A special case of the problem exists when programs can only access local data because in such cases the program and data allocation problems are not separable and are

solved jointly. Appendix G shows that even this problem is decomposable into individual data allocation subproblems. Due to these decompositions, the dynamic programming based algorithm introduced in chapter 2, to solve the idealized allocation problem, can be easily extended to include the effect of transaction processing and update synchronization. Although dynamic programming is usually non-polynomial, this algorithm generally yields polynomial results because each stage can be solved in polynomial time due to the effect of update synchronization costs discussed in chapter 4.

Consistency updates can make the objective function non-decomposable. The sequencing/ordering routine described previously arranges the data groups in such a fashion so that the groups with no consistency relationships are allocated first and the groups with strong consistency interactions are allocated at the end. If sequencing is completely successful (group to be assigned at stage n is only related to groups assigned in the n-1, n-2,,,,1 stages) then the problem is separable and decomposable (see <NEMH66>). If not, sequencing provides a good approximation. The effect of the grouping, weak-consistency and sequencing approximations was formally discussed in chapter 3. The main result of this analysis is that the grouping and the weak-consistency approximations complement each other because the proximity relationships developed in the grouping scheme tend to group data objects with consistency relationships, thus reducing the effect of consistency relationships. These issues will be reviewed again in next section.

Now, let us analyze the constraints and the constrained optimization problem.

The storage and data replication constraints are linear and decomposable in terms of the data allocations. However,the availability constraints are nonlinear and non-decomposable in terms of data allocations for transactions accessing more than one dataset but are linear and decomposable for single dataset transactions. This is because

the availability of a transaction when it arrives at node k, A(t,k), is given by:

$$A(t,k) = \Pi_p A_p(p) \Pi_d A_d(d) \quad \text{for all p,d accessed by transaction t}$$

where $A_p(p)$ is the availability of program p from node k and $A_d(d)$ is the availability of dataset d from program p. If $A_p(p) = 1$, then we can transform the transaction availability to dataset (file) availability:

$$A(t,k) = \Pi_d A_d(d) \quad \text{for all d accessed by the transaction}$$

This shows that availability of transaction t is decomposable into dataset availability which can be calculated at each data allocation stage. Let us analyze this decomposition. In case of RDA query processing strategy, we can easily assume that the arrival node is the program selection site and since we are assuming 100% node availability, $A_p(p) = 1$ for all $p \in P$. In case of LDA, $A_p(p) \neq 1$, but in this case the programs must run on the same nodes where the data resides. Thus $A_p(p) = A_d(d)$ for all $d \in D$ and all $p \in P$ for LDA. This decomposition does not hold in case of RDA when the arrival node is not the program site selected, mainly due to the unique or sparse allocation of programs. Once decomposed, we treat these constraints separately in a fashion similar to Mahmood & Riordon <MAHM76c>. Thus the availability constraint can be decomposed if we assume that programs are always available or if the program availability is given as a constant. We can easily make this assumption since the data allocation is conducted after the program allocation scheme which determines the program allocations and program availability.

The response time constraints are also nonlinear and non-separable. The response time calculations, however, can be decomposed into the processing and queuing/locking delays. The processing delays are separable for transactions not involving consistency updates. Generally speaking, the queuing/locking delays are nonseparable even for single dataset transactions due to the interactions between devices. However,

the queuing delays QD can be expressed in terms of qd(i), the queuing delays at node i:

$$QD = \Sigma_i \, qd(i)$$

if the following assumptions are true: i) the queuing at network links is ignored and ii) queuing at each node is calculated separately by using the Jackson Decomposition Theorem <JACK63f>. Fortunately, the queuing calculations shown in chapter 5 satisfy these conditions because the homogeneous arrival rate and homogeneous service time assumptions are satisfied by most computing environments. Recall that homogeneous arrival rate is an operational analysis equivalent of Poisson arrivals and homogeneous service time is an operational equivalent of exponential service time. Now, qd(i) is a function of L(t,k,i), the local service requests executed at node i due to arrival of transaction t at node k. This variable is dependent primarily on the data allocations. Thus, if qd(i) is decomposable into the queuing delays q(d1), q(d2),,,q(dn) due to allocation of datasets d1,d2,,,dn to node i, then the total queuing delay QD is decompoable into dataset allocations, else not. Unfortunately, this is true only if a single copy of each dataset is allocated at each node and the background workload is ignored. We achieve approximate decomposition by sequencing the most heavily used datasets first and also by restricting the queuing/locking calculations to fewer cases (the first pass of the algorithm only uses processing delays). The basic argument is that if a portion of the transaction does not satisfy the response time constraint, then the overall transaction will also be infeasible. At the dataset allocation stage n, the response time is calculated for all the transactions, or subtransactions, which utilize the datasets allocated in stages n, n-1, n-2,,,1. The effect of this approximation, called the workload approximation, will be discussed later in this chapter (section 6.5).

Thus, the overall allocation problem is a integer programming problem with nonlinear and non-separable objective function and con-

straints. By using the weak-consistency and workload approximations, the objective function as well as the constraints can be separated and the problem can be decomposed into several subproblems, where each subproblem optimizes the allocation of a single dataset or datagroup.

6.3.4.3. The Data Allocation Algorithm

The data allocation exists in two forms: DATA1 which allocates data after program allocation and DATA2 which allocates data before program allocation. The allocation algorithm chooses DATA1 or DATA2 depending on the value of data access mode, RDA or LDA, respectively which is represented by the transaction processing parameter Q_{da}. Basically, DATA1 assumes that programs have been already allocated and DATA2 allocates data and programs at the same nodes. Besides this difference, the rest of the discussion that follows applies to both DATA1 and DATA2. The main data allocation algorithm can be viewed in terms of a global subroutine which selects datasets to be allocated and a local subroutine which optimally allocates a given dataset.

The GLOBAL SUBROUTINE is a dynamic programming based hierarchic heuristic procedure. The stages, states, decisions, returns and recursive equations of the dynamic programming framework are:

x(n) = State variable at stage n

a(n) = Data allocation decision at stage n

= Allocation of data group n to nodes

y(n) = Return at stage n

x(n) = t(x(n), a(n))

= state transition function

$F_n(x(n))$ = Accumulated optimal return at stage n

$= \text{Min} \ (y(n) + F_{n-1}(x(n-1)))$

Y(n) = Accumulated return at at stage n = $y(n) + F_{n-1}(x(n-1))$

The recursive equation for Y(n), the accumulated return at stage n (allocation of data group n) can also be expressed as:

$$Y(n) = O[a(n), a^{*}(n-1),,,,,,a^{*}(1)]$$

Where $O[a(n), a(n-1),,,a(1)]$ is the objective function based on decisions $a(n)$, $a(n-1),,,,$ $a(1)$ and $a(n)$, the data allocation decision at stage n, is a vector of decision variables α:

$$a(n) = [\alpha(dn,1),\alpha(dn,2),\alpha(dn,3),,,,]$$

The main feature of this algorithm is that the optimization of $Y(n)$ is performed on $[a(n), a^{*}(n-1),,,a^{*}(1)]$ rather than $a(n)$. Thus in case of decomposable objective functions, the algorithm exhibits standard dynamic programming features and yields exact solutions, while in case of nondecomposable situations it becomes a hierarchical heuristic procedure in which the optimal decision made at stage n-1 becomes a constraint at stage n. This routine basically picks a dataset group at each stage for evaluation by the local search routine. The dataset groups have been already ordered by the sequencing routine described above.

The LOCAL SEARCH ROUTINE attempts to determine the optimal value of the return at stage n, Y(n) and also calculates and evaluates the constraints. Optimization of Y(n), in the decomposed form, is very similar to the single file allocation problem addressed by Casey<CASE72> (see chapter 2 for a translation/reduction of this problem to Casey's formulation). This subroutine basically evaluates the allocation of dataset group dg generated for stage g, by the global routine and utilizes the following algorithm:

1. Pick the dataset group dg for stage g.

2. Construct a cost hypercube for the dataset dg. For an Nn node network, the hypercube consists of 2^{Nn} vertices. Eliminate all the vertices which violate the availabilty, storage and policy constraints (see figure 6.4). Note that avilability constraints eliminate higher vertices while the storage and policy constraints eliminate lower vertices.

3. Start at level r=1, which satisfies the availability constraints and store all vertices at level r into the admissable set K_r. Initialize s, the level counter, to 0.

4. Set s=s+1, where s ≤ NN ,the number of nodes. (s sets an antecedent at level 1 for all the verical paths.)

5. Pick the sth vertex v Ξ K_r and determine J_v , the set of successors for vertex v at level r+1.

6. For the chosen v, select a vertex v' Ξ J_v such that v' is admissable (feasible) and usable (not evaluated). Note that the availability, storage and policy constraints significantly reduce the number of admissable vertices, i.e. the constraints improve the efficiency of the algorithm.

6. The cost of allocating dataset dg to v' is evaluated. If cost at v' ≥ cost at v, then:

. Mark v' inadmissable.

. Mark all successors of v' inadmissable.

. Reset r=1 and go to step 5.

Otherwise (cost v' < cost v):

. Mark v' used.

. go to next step to evaluate response time constraints.

Note that the scheme always traverses the hypercube vertically.

8. The transaction response time is determined for v' and the transaction response time constraint is checked. The processing delays are used in the first few stages to represent response time. This helps us to quickly eliminate infeasible allocations and perform the detailed queuing/locking calculations at the last stages. If the response time constraint is violated, then:

. Mark v' inadmissable.

. Mark all successors of v' inadmissable.

. Reset r=1 and go to step 5.

Otherwise:

. Set $v = v'$.

. If $r \geq NN$ then set reset r=1 and go to step 5,

else set r=r+1 and go to step 5.

Thus as the algorithm proceeds, only feasible alternatives are evaluated.

9. If $v' \in J_v$ cannot be found, this means all successors of v have been evaluated or are inadmissable:

. If $r \neq 1$, then set r=1 and go to step 4.

. If r=1, then find the optimal vertex v^*_s for

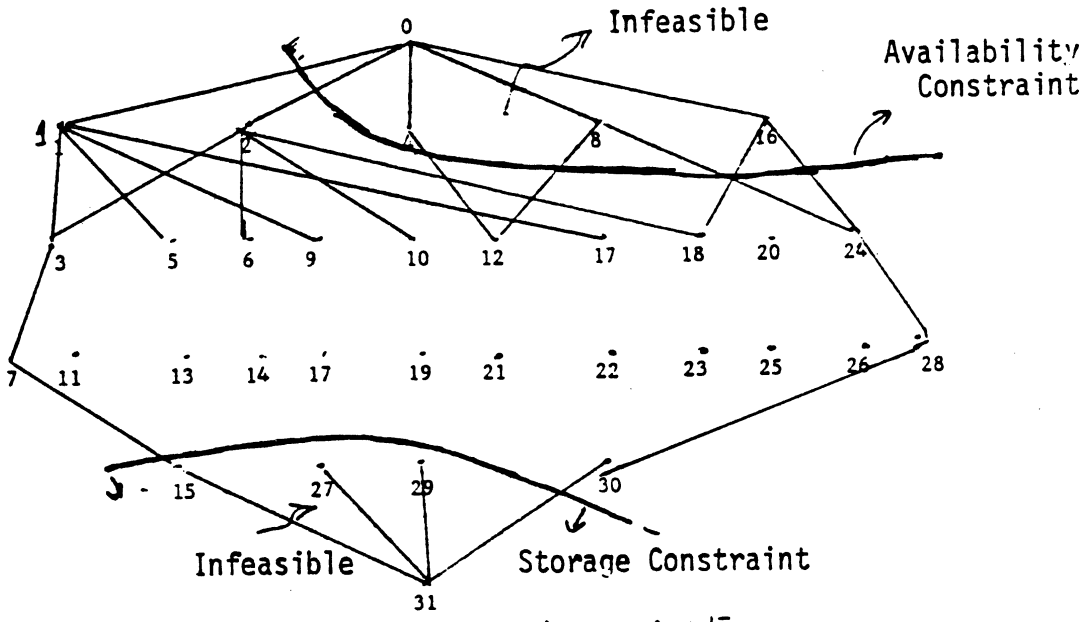
all succesors of sth vertex and go to step 3.

10. When all vertices at r=1 have been evaluated ($s \geq NN$), then determine v^* (over all s) which minimizes cost. This is the optimal assignment at stage n, and minimizes Y(n).

We need to pay special attention to the effect of update synchronization costs and the use of "distributed" optimization algorithms in the data allocation algorithm. Essentially, the objective function (storage cost + local processing cost + remote processing cost) increases rapidly with multiple dataset copies due to update synchronization. This result was discussed in detail in chapter 4. Recent simulation studies <BARB82c, MATS82c> have also shown that in most cases, more than one dataset copy is costly. For example, <MATS82c> has shown that more than one dataset copy is desirable only if the read to update ratio of the dataset is less than 0.2. The local search routine uses a "vertical" search of the hypercube and thus takes advantage of these results by limiting the search of the hypercube to only the first few levels of the hypercube (see fig 6.4). This reduces the size of the problem considerably.

The data allocation algorithm presented here can be easily converted to a distributed optimization algorithm. To illustrate the use of distributed optimization, consider the allocation of 3 datasets (d1,d2,d3)

FIGURE (6.4): DATA ALLOCATION COST HYPERCUBE



. All vertices are not connected for clarity
. Each vertex represents a data allocation.
. The vertex number is a decimal equivalent of binary allocation matrix. For example, 4 represents 00100, 14 represents 01110 and 29 represents 11011.

to 5 nodes (n1,n2,n3,n4,n5). Let us also assume that each of the 5 nodes (computers) is available for running the optimization algorithm. Then we can "distribute" the data allocation algorithmm in one of the following manners:

1. Each stage (dataset allocation) can be run simultaneously on the available systems and the results can be combined at the end. Notice that only three computers will be used in this algorithm.
2. For each stage, portions of the local search can be run in parallel. For example, all the vertices of the hypercube involving node 1 could be evaluated on computer 1 while the vertices involving node 2 are being evaluated on computer 2. Notice that we can use all 5 computers simultaneously.

3. A mixture between the above two cases can be developed.

We are investigating implementation of the second (local search) approach due to two reasons: a) as the size of the problem grows with the number of nodes, we can use the nodes itself to reduce the computation time and b) the vertical search routine can be "distributed" in a very straightforward manner. A simple distributed optimization algorithm is given below. This algorithm consists of a main algorithm which coordinates the execution of several parallel subalgorithms.

a). The Main Algorithm

. Divide the optimization problem into Q subalgorithms,
In most cases, Q=NN, where NN = no. of nodes in the network.

. The first subalgorithm evaluates all the successors of node 1,
the second subalgorithm evaluates successors of node 2, etc.

. Send messages to computers to run one or more subalgorithms.
It is assumed that the programs to run the subalgorithm
exist at the respective computers.

. Receive and store the optimal return M(q) for the subalgorithm q.

. When all M(q) have been received, then determine the least
M(q), for q=1 to Q. This is the final value.

b). The Subalgorithms

. Receive the node no. n, the maximum no. of nodes
NN and the subalgorithm no. q.

. Construct the successors of node n.

. Determine the optimal return M(q) by using the local search
routine shown above.

. Return M(q).

The subject of distributed optimization is frequently referred to as parallel optimization and is an area of considerable research activity (see for example Cook <COOK83f>). This research is motivated by the advances in VLSI technology which allow several processors to perform com-

putations in parallel. The application of parallel (distributed) optimization to solving DDP resource allocation is natural. However, very little attention has been paid to this important area so far.

6.4 Implementation and Synthesis

All the schemes, algorithms and routines which solve the generalized program and data allocation problem stated in section 6.1, and shown in figure 6.1, have been implemented in Fortran as a single program. This program, for convenience called DADP (Distributed Application Design Program), consists of about 2500 lines of source code including comments (about 800 lines) and I/O. The program has been tested on an IBM 3033 system under the MVS operating system and consists of a main program which calls several subroutines. The flow of DADP is:

1. Read the input parameters, initialize varaibles and calculate the PATH matrix by calling the CALCTP subroutine.
2. Call the subroutine GROUP to:
    . Form the dataset and program groups.
    . Sequence the dataset groups
3. If local data access (LDA), then goto step 4, else call the PGMOPT subroutine to determine the optimal allocation of programs. This subroutine:
    a. Picks each stage in a predetermined order, identifies each stage as a program or node stage and then calculates the PLIST and NLIST at each stage.
    b. Determines the internal variables for spatial dynamic programming. These internal variables, as discussed in the program allocation scheme above, constitute the decision variables at each stage. The subroutine always starts with a full allocation (replication) and deletes allocations if feasible.

c. For each program allocation decision genererated from the internal variables, the subroutine:

- . determines if the allocation violates the program allocation constraints of program duplication and policies.
- . calculates the communication cost for node to program and program to program traffic by calling subroutine CALPGM.
- . Calculates the response time RT for each transaction and checks the feasibility.

d. When all the internal variables for a given stage have been evaluated, then go to step 3a for the next stage.

4. Start the data allocation scheme to:

a. Choose a dataset group d from the set D.

b. Eliminate the infeasible vertices which violate the availabilty, storage and policy constraints. Now, call the subroutine SEARCH to:

- . Determine the next lower vertex (each vertex shows the data allocation $\alpha(d,i)$).
- . Cause backtracking (go back to level 1 node) if needed.

c. For the given vertex, call the subroutine CALC to:

- . Calculate dataset site, $\alpha_s(k,p,d,i)$.
- . Calculate the LSR and RSR.
- . Calculate the total cost (objective function).

d. If cost decreases go back to next step for evaluation, else go to 4b for backtracking.

e. Call CALDEL to compute the response time. The response time can be calculated at several levels: processing delays only, queuing without conflicts, queuing with locking conflicts and

background workload. If response time constraint is violated, then go back to b for backtracking, else get lower vertex.

f. When all vertices for 4d and 4e evaluated, go to a.

g. When all datasets evalauted, determine the optimal value.

5. Print the main results and terminate.

DADP receives an options card, application parameters and support system parameters, and produces a vector of 10 values for each program or data allocation. The program is heavily documented (800 lines out of 2500 are comments) and can be easily modified to fit analyst needs. The source code and listings are available upon request. The following additional features of DADP are worth noting:

1. The algorithm can be used in two modes: predictive mode to evaluate given configurations for "what if" answers and optimization mode to determine optimal program and data allocations. The predictive mode allows explicit enumeration and table input for data and program allocations, thus enabling the analyst to explore and investigate the most attractive choices, without having to run the optimization algorithm.
2. Defaults are used heavily by the input routine to eliminate unnecessary input data. For example, if the problem is to be solved for an idealized support system, then N_n (the number of nodes) is the only input parameter needed. In this case a fully connected reliable network with fast cpus and links is assumed. This allows an analyst to quickly eliminate infeasible options without having to describe the support system fully.
3. The algorithm allows evalation of different objective functions. For example, the remote processing cost can be chosen on a per message, transported volume or flat cost bases. The optional con-

version factors f1 and f2 can be used to translate the local and remote service requests to physical i/o and messages.

4. Several options, specified through input parameters, control the level of detailed calculations performed by various subroutines. For example, i) the grouping threshold TH can be adjusted to achieve different levels of clustering in the grouping scheme, ii) the grouping scheme can be bypassed altogether, iii) the unconstrained calculations can be performed by bypassing all constraints and iv) the response time calculations can be performed at three levels: processing delays only, processing + device-busy delays and processing + device-busy + lock-conflict delays.

6.5 Evaluation of the Algorithm

The generalized allocation algorithm and the implemented program DADP is a hierarchical algorithm which involves several approximations and a series of decisions and heuristics. A variety of analytical and empirical techniques are employed to evaluate this algorithm, henceforth referred to as DADP. Within the life cycle of an application system, the DADP operates at the distributed design (global) level which focusses on allocation of data and programs to nodes. After this design is completed, the analyst then conducts the local physical design at each node which consists of local database and program design. Thus the calculations at the global design step are, at best, approximate because some variables cannot be correctly determined unless the local physical design is conducted. Since different data and program allocations yield different local database and program design problems, the exact storage size, which includes the physical pointers, and the local processing activity, which includes the cpu and disk i/o, can only be found after the data and programs have been allocated. Thus in the formulation of the generalized program and data allocation algorithm we can introduce the

following changes: storage occupied and local processing activity are represented by random variables. Then the storage and local processing costs become expected values and the program and data allocation algorithm is translated into a stochastic programming problem. Problems of this nature are frequently encountered in operations research, where a series of decisions have to be made over time with increasing levels of detail. These methods are termed "hierarchical planning systems" (see Dempster et al <DEMP80f, DEMP81f> for general discussion and examples). While exact solution of stochastic programming problems is almost impossible, without special structure, these models can be used for evaluation purposes by using the following procedure:

1. Formulate an initial model which defines the relationships between variables that are known precisely plus the variables that will be determined later at a detailed level. This was accomplished in section 6.1.
2. Use approximate numbers to reflect expected values. The values from worst-case, best-case or average estimates can be used. In our formulation, we have used local service requests to reflect the local processing activity (cpu + disk i/o) and datasize to estimate the storage requirements, while the effect of pointers and blocking is ignored.
3. Optimize the initial model for a given objective function and a set of constraints. The DADP program accomplishes this.
4. Conduct the detailed design and run experiments to determine actual values. The local physical design is used in our case to determine the actual values of local processing and storage costs.
5. Use the actual values now to rerun the high level algorithm. This is called "feedback analysis" and will be discussed later.

Analytical model can be used to predict the performance of the high level model and empirical analysis can be employed for verifying the actual performance.

6.5.1 Analytical Model

DADP uses several approximations: i) the grouping approximation which clusters the programs and datasets, ii) the weak-consistency approximation which achieves decomposition of the data allocation problem, iii) the weak interprogram approximation which allows the use of program and data allocation for LDA case and iv) the workload and availability approximations which allow decomposition of the data allocation in case of response time and availability constraints. The grouping and weak-consistency approximations were evaluated in chapter 3 and the weak-interprogram approximation is evaluated in appendix H. The combined effect of all the approximations on the correctness of this algorithm can be discussed in terms of worst-case and probabilistic analysis.

WORST-CASE ANALYSIS: Let U denote the set of problem instances, $I \in U$ a particular problem instance, Z(I) the optimal value of problem I, and $Z_H(I)$ the optimal value obtained when algorithm (heuristic) H is applied to the problem. It is assumed that algorithm H is well specified to uniquely define $Z_H(I)$. In terms of DADP, I is defined by a particular instance of an application system and a DDP support system, and H denotes DADP. Worst-case analysis establishes the maximum deviation of $Z_H(I)$ from Z(I), and different type of measures can be used to show the deviation. Let ε denote the deviation (error):

$$Z_H(I) - Z(I) = \varepsilon$$

where $\varepsilon \geq 0$, since the allocation problem is formulated as a minimization problem. The worst-case deviation, ε^*, is given by:

$$Z_H(I) - Z(I) \leq \varepsilon^*$$

A percentage measure of worst-case error is given by $\varepsilon^*/Z(I)$. The purpose of worst-case analysis is to determine ε^* for each H. For notational simplicity, we will drop the argument I on $Z_H(I)$ and $Z(I)$ for the balance of this section. The worst-case performance of the generalized program and data allocation algorithm DADP is given by (see appendix H for proof):

$$Z_{DADP} - Z \leq MAX\{\varepsilon_g, \varepsilon_t\} \quad \text{.....(6.70)}$$

where:

Z_{DADP} = optimal cost obtained by running DADP

Z = the actual optimal cost

ε_g = worst-case deviation due to grouping approximation

ε_t = total worst-case deviation of the data allocation scheme

$= \varepsilon_c + \varepsilon_p + \varepsilon_w$

where:

ε_c = worst-case deviation due to weak-consistency approximation

ε_p = worst-case deviation due to weak-interprogram approximation

ε_w = worst-case deviation due to workload/availability approxim.

(see appendix H for analytical expressions of ε_g, ε_c, ε_p and ε_w)

We observe from this expression that ε_g plays a key role in worst-case deviation. This is primarily due to the observation that if all the datasets and programs are clustered together, then ε_g is maximized; however $\varepsilon_c = \varepsilon_p = \varepsilon_w = 0$ because there is no need for decomposition. Other observations about the worst-case performance of DADP are:

- ε_g, the worst-case deviation due to grouping approximation, is a function of the number of nodes NN, the storage size, local processing activities, remote processing activities and costs. However, the amount of grouping can be easily adjusted through an input parameter, the grouping threshold TH. For a centroid threshold, the error introduced by ε_g ranges from 5 to 10%, for a 15 node network.

- ε_c, the worst-case deviation due to the weak-consistency approximation, is a function of the number of nodes NN, application system parameters (such as the consistency relationships), and updates $D_c(d,dl)$ and $P_c(p,dl)$, and local and remote resource utilization costs. For application systems with few consistency relationships and updates (less than 30% of reads), the error introduced due to ε_c for 15 nodes ranges from 10 to 15%. However, this error can be reduced by increasing the grouping threshold TH.
- ε_p, the worst-case deviation due to weak-interprogram approximation, is small if transactions exhibit parallel behaviour and if dataset clustering is large.
- ε_w, the worst-case deviation due to decomposition of the availability and response-time constraints, is also a function of the number of nodes NN, the storage size and read/write activity of the application system. For a 15 node network, with moderate read/write activity, the error introduced by ε_w is about 10%.

In short, for low level of grouping, the total worst-case deviation of DADP due to ε_c, ε_p and ε_w can add upto 30%, corresponding to a grouping error of about 5%. However, we can increase the level of grouping in such cases which increases the grouping error but decreases the other errors. We have found that, for most cases, the total worst-case deviation of DADP can be lowered to about 10 to 15% by properly adjusting the grouping threshold (see empirical analysis). It should be empasized that the actual worst-case performance of DADP is dependent upon the type of input parameters (e.g., application and support systems).

PROBABILISTIC ANALYSIS: Probabilistic analysis provides a mechanism for the study of the probabilistic properties of algorithms based on assumed density functions and thus gives us insights into the average performance of algorithms. Karp <KARP76f> has surveyed this technique and has shown the application of probabilistic analysis to a partitoning al-

gorithm for large scale trevelling salesman problems <KARP77f>. Fisher and Hochbaum <FISH80f> have shown application of this technique to the planar K-Median problem. To illustrate the principles of probabilistic analysis, let ε denote the deviation of Z_H from Z, i.e:

$$\varepsilon = Z_H - Z$$

This deviation can assume values between 0 to ε^*, where ε^* is the maximum deviation achieved through worst-case analysis. Since H is deterministic, several instances I can generate the same ε. For example, if $\varepsilon = 0$, then two instances $(I1,I2) \in U$ may produce $Z_H(I1) = Z(I1)$ and $Z_H(I2) = Z(I2)$. Let $\varepsilon = n$ denote that ε assumes a value n, where $n \geq 0$, then let I(n) show the set of instances which produce $\varepsilon=n$ and let $Pr(\varepsilon=n)$ denote the probability that a given experiment produces $\varepsilon=n$, i.e., it chooses an instance from I(n). Assuming t deviations (n1,n2,,,, nt), the expected deviation EXP(ε) is:

$$EXP(\varepsilon) = \Sigma\ Pr(\varepsilon=n) \times n \qquad .....(6.71)$$

This equation gives a measure of average performance of H. The procedure used for this calculation is:

1. Choose a set of deviations to be analyzed. For the purpose of analysis, we choose three deviations: 0, $\varepsilon^*/2$ and ε^*, where ε^* is the maximum deviation given by the worst-case analysis.
2. Calculate, or estimate, the probabilities $Pr(\varepsilon=n)$. In most cases this estimation is difficult since it requires knowledge of the environment in which the algorithm is used. We will assume some values for the purpose of analysis.

For the purpose of analysis, we use equation 6.70 to study $Pr(\varepsilon=0)$, the probability that the error is zero. $Pr(\varepsilon_g = 0)$, the probability that the grouping approximation produces zero deviation is a function of proximity relationships, the grouping threshold and the costs of the network resources. This probability can be adjusted, as stated above, by changing the groupung threshold TH, an input parameter to DADP.

$Pr(\varepsilon_c = 0)$, the probability that the weak-consistency approximation depends on the number of consistency updates, relationships and on read/write ratios. These values cannot be adjusted since these reflect the properties of the application system being studied. $Pr(\varepsilon_p=0)$, the probability that the weak-interprogram approximation is zero for applications with few program-to-program relationships,is not an adjustable parameter. $Pr(\varepsilon_w=0)$, the probability that the weak workload approximation is zero, if the first stages of the data allocation scheme eliminate the infeasible allocations which violate the response-time and availability constraints. This value can be adjusted by using the following techniques: the processing delays and file availability calculations are used in early stages and also the processing delays can be adjusted to reflect potential queuing delays by increasing the service times of the various devices. This can only be achieved, however, as part of the feedback analysis.

6.5.2 Empirical Analysis

The computational efficiency of DADP can be analyzed in terms of three factors: i) the number of computations needed to evaluate the objective function and the constraints for each allocation, ii) the number of allocations needed to determine optimal return and iii) the deviations observed from the optimal. In order to analyze the first factor, let us consider the input size given by the following parameters:

- NR, the number of transactions being analyzed.
- NP, the number of programs in the application system,
- ND, the number of datasets in the application system
- NN, the number of nodes in the network
- NL, the number of links in the network

DADP requires the order of $NN^3 \times (NR \times NP^2 \times ND^2)$ computations for each computation of the objective function (total cost). The availability and

response-time constraints are most expensive to evaluate. $NN^2 \times NR \times (NP+ND)$ computations are needed per evaluation of the availability constraint and $NN^2 \times NL$ calculations are needed per evaluation of the response time constraint. Thus the heuristics employed in DADP to bypass evaluation of total cost, response-time and availability calculations play an important role.

DADP has been programmed and tested extensively on an IBM 3033 system running under MVS and the model has been validated in an existing DDP environment (see appendix J for model validation). To investigate the computational aspects of DADP in terms of the evaluations needed to reach the optimal, and the error introduced, and then to compare the results with the worst-case and probabilistic analysis, consider the test case allocation of the two programs/three datasets to a 5 node network. It was shown in chapter 2, that the unconstrained allocation algorithm finds the optimal value within 67 evaluations in 3 seconds on an IBM 3033 system. The same problem is solved by DADP in about 5 seconds of cpu time. The increase in time is due to the program allocation scheme and the introduction of availability and response-time calculations. Solution of the same problem with 10 nodes requires approximately one minute of cpu time.

Table 6.4 summarizes the results of running about forty experiments on an IBM 3033 system for the allocation of two program, three dataset student registration system to 5 nodes. It is assumed that the transaction response time must not exceed ten seconds and transaction availability must not be less than 70%. The support system environments included four transaction processing options (parallel and serial routing graphs, local and remote data access) and nine update synchronization algorithms (basic 2PL, SDD1, pessimistic, optimistic etc) for fully connected unconstrained, fully connected constrained and star toplogy networks. The table shows the number of allocations (iterations) needed to reach the optimal solution, time required, the optimal

cost obtained from DADP, Z_{DADP}, the actual optimal cost, Z, and the percentage error $(Z_{DADP}-Z)/Z$. The deviations from the optimal were determined by using explicit enumerations in portions of the allocation algorithm so that all the single, two and three file allocations were searched exhaustively. Each enumeration experiment was allowed to run for 3 hours of CPU time which was enough to evaluate over 100,000 allocations.

We can make several observations from table 6.4: the algorithm performs very well for unconstrained and fully connected networks, errors are introduced in star network topologies, the update synchronization algorithms tend to decrease the number of evaluations needed for determining optimal or infeasible situations (for the pessimistic algorithm, the optimality and infeasibility are determined within 10 iterations), the maximum error is about 6%, a large number of cases yield exact optimal allocations and the errors introduced are within the range indicated by the worst-case and probabilistic analysis. Although it would be interesting to run the same experiments for larger problems, the computational time required to determine exact optimal by using explicit enumeration is prohibitive. (It took about 2 hour of cpu time to find the exact optimal for each experiment.) Table 6.5 shows the run time results of DADP for larger problems.

The sequencing and the grouping schemes are very effective in reducing the computation time and introduce minimal errors. Application of mathematical programming in the grouping scheme <RA071f, VIN069f> can be investigated to further improve the effectiveness of the grouping scheme.

Program allocation schemes, have the property that when there are no program allocation constraints, all the programs can be allocated to all the nodes since this yields minimum cost. We have used this observation in designing the local search routine which allocates a given program to nodes. Since in most practical cases, few constraints apply to program

TABLE 6.4: PERFORMANCE OF THE GENERALIZED ALLOCATION ALGORITHM

a). FULLY CONNECTED, UNCONSTRAINED NETWORK

	Allocations to Reach Optimal	Time(sec) to Reach Optimal	DADP Optimal Cost	Actual Optimal Cost	Error %
Parallel, RDA	45	3.5	17,528	17,528	0
Parallel, LDA	46	3.5	6,583	6,323	4.1
Serial, LDA	45	3.5	6,583	6,323	4.1
Serial, RDA	45	3.5	6,473	6,473	0
Basic 2PL	40	3.4	53,064	53,064	0
Basic Timestamp	41	3.4	42,264	42,264	0
ELLIS Ring	40	3.4	24,569	24,309	1.1
THOMAS Voting	40	3.4	22,544	22,284	1.1
GARCIA's CCA	45	3.5	20,334	20,334	0
GARCIA's CLA	32	2.9	53,464	53,464	0
Pessimistic	10	1.1	211,008	211,008	0
Optimistic	45	3.5	17,528	17,528	0
SDD1	40	3.4	53,064	53,064	0

b). FULLY CONNECTED, CONSTRAINED

	Allocations to Reach Optimal	Time(sec) to Reach Optimal	DADP Optimal Cost	Actual Optimal Cost	Error %
Parallel, RDA	45	5.1	17,028	17,028	0
Parallel, LDA	45	5.1	6,583	6,323	4.1
Serial, LDA	45	5.1	6,583	6,323	4.1
Serial, RDA	49	5.2	6,473	6,473	0
Basic 2PL	22	2.5	56,064	55,901	0
Basic Timestamp	21	2.5	45,724	44,444	2.2
ELLIS Ring	31	3.4	24,569	24,569	0
THOMAS Voting	33	3.5	22,544	22,284	1.1
GARCIA's CCA	40	4.8	20,334	20,334	0
GARCIA's CLA	22	2.5	56,012	56,004	0
Pessimistic	8	1.0	INFEAS.	INFEAS.	0
Optimistic	45	5.1	18,028	18,028	0
SDD1	21	2.5	53,064	53,064	0

c). STAR NETWORK TOPOLOGY, CONSTRAINED: Node 1 is the central node

	Allocations to Reach Optimal	Time(sec) to Reach Optimal	DADP Optimal Cost	Actual Optimal Cost	Error %
Parallel, RDA	34	3.9	30,332	28,508	6.3
Parallel, LDA	52	5.2	7,282	7,208	0
Serial, LDA	22	2.3	2,389	2,250	6.1
Serial, RDA	29	3.1	7,302	7,222	1.1
Basic 2PL	21	2.3	34,064	32,250	5.6
Basic Timestamp	24	2.4	27,772	27,014	2.8
ELLIS Ring	22	2.1	31,746	31,486	0
THOMAS Voting	25	2.4	28,896	28,636	0.9
GARCIA's CCA	50	5.1	20,334	20,334	0
GARCIA's CLA	19	2.0	22,846	22,232	2.7
Pessimistic	7	1.0	INFEAS.	INFEAS.	0
Optimistic	34	3.4	30,332	28,508	6.3
SDD1	22	2.3	34,064	32,250	5.6

allocations, this algorithm completes in polynomial time by replicating programs to most of the nodes.

Data allocation schemes have the property that the decision space is considerably reduced due to availability and storage constraints. These constraints are inexpensive to check and are thus very useful in quickly eliminating infeasable allocations. Recall that the availability constraints eliminate high order vertices while the storage constraints eliminate low order vertices of a cost hypercube. For most cases, the additional cost due to update synchronization makes it very costly to allocate more than one copy in the network, thus drastically reducing the size of the problem. (It is interesting to note that the program allocations tend to favour complete replication while data allocations, due to the constraints and additional costs, favour single copies.) Also, some of the allocations can be evaluated in parallel by using distributed optimization algorithms, where several computers are employed to independently solve portions of the optimization problem and then coordinate the results. This technique can significantly reduce the computation time and is particularly attractive for solving computing network problems since for large networks, the size of the problem is large, but then the number of available computers is also large.

Due to these reasons, the optimal program and data allocations can be determined in polynomial time. The main strength of this approach is that it automatically yields exact solutions for simple application systems and support systems, and near optimal solutions for complex situations. For example, if an application system consists of single program and single dataset transactions, or if no relationships exist between the datasets and programs of an application system, then the algorithm produces exact results because the allocation problem is decomposable without any approximations and the grouping scheme is not needed. However, complex application systems with many relationships between datasets and programs need the approximations for decompos-

Table (6.5): Run Time Analysis of DADP

No. of Datasets	No. of Programs	No. of Nodes	Cpu time (IBM 3033 System)
2	2	5	2 to 5 seconds
10	10	5	10 to 15 seconds
15	15	5	20 to 30 seconds
3	2	10	2 to 5 minutes
3	2	15	15 to 25 minutes

NOTE: The range of cpu time is due to the effect of support system configurations on optimization (see for example table (6.4))

ition. Note that an analyst does not have to know if a particular application is decomposable or not. This allows an analyst to apply the same model to a wide range of application systems of an enterprise.

Let us now briefly review the feedback analysis. The feedback analysis is obtained by using the following procedure: 1). Run the algorithm with given costs, application parameters and support system parameters. 2). After obtaining the optimal results, conduct the local design for the optimal allocation. In the absence of local design, we "simulate" this activity by using some predetermined values. 3). The adjusted values are introduced in the model by changing the costs. For example, if the storage size increases by 10%, then we simply multiply the storage cost by 1.1. 4). Go back to step 1 and repeat till the values converge. Feedback analysis can be also used within the optimal allocation algorithm between stages. We have found that in most cases, the algorithm

converges after three iterations. This procedure, however, does not work if the data models at various nodes are so different that an estimate of local design factors is difficult. For example, we have assumed at the global design level that one local service request roughly results in one physical i/o. However, if the number of physical i/o issued per local service request varies very widely beween nodes, then the feedback analysis will require more iterations to converge.

6.6 Chapter Summary

In this chapter, the generalized program and data allocation algorithm is presented which attempts to find optimal program and data allocations to minimize the total cost (storage + local processing + communication processing) subject to transaction response time, availability, storage, policy and program duplication cost constraints. This algorithm synthesizes and integrates all the tools and techniques presented in previous chapters into a single model. The algorithm has been coded and tested and is the main output of this research. The algorithm is evaluated by using thoretical as well as empirical analysis.

CHAPTER 7

SIGNIFICANCE OF RESEARCH AND CONCLUSIONS

The generalized allocation algorithm integrates a diverse set of techniques to near-optimally allocate programs and data for complex application systems in real life support system environments. The algorithm is applicable in a variety of ways. First, an analyst can model real life situations by using the defined parameters and can then study the effects and tradeoffs of program and data allocations of given application systems for various support system configurations. Second, the program and data allocation process can be automated for a given application system environment. Third, many of the currently available allocation schemes can be shown to be special cases of DADP, the program which implements the generalized allocation algorithm; thus this program can be used as a means to synthesize and analyze existing algorithms. Fourth, a complete distributed application system design methodology can be developed which utilizes DADP at several levels as a design aid. Finally, the algorithm can be used in practical situations to specify how existing centralized applications can be distributed, to evaluate the effects of support system changes on application design and to distribute control programs and directories. The objective of this chapter is to illustrate the significance of this algorithm, point out limitations of this research, and suggest future areas of investigation.

7.1 Representation of Real Life Distributed Systems

Program and data allocation in real life situations requires specification of two major systems: application systems and support systems. Application systems may consist of simple queries or large scale applications involving complex relationships among data, programs and

transactions. Support systems may consist of a few homogeneous nodes or a large number of heterogeneous nodes, interconnected in a variety of ways, controlled by different operating systems, database managers and transaction managers. It was shown in chapter 1 that most existing allocation schemes are limited to simple, one program/one dataset application systems, and idealized support systems without update synchronization, transaction processing and communication system considerations.

Table 7.1 shows all the application and support system parameters, and other parameters, used by the allocation algorithm presented in chapter 6. The application and support system parameters were defined and discussed in chapters 2, 4 and 5, and were used as input to derive the analytical expressions for the output variables like the local processing costs, remote processing costs, storage costs, transaction availability and transaction response time. Let us briefly review how these parameters can be used to represent realistic systems.

APPLICATION SYSTEM DESCRIPTION: An application system consists of three components: the database, the programs which manipulate the database and the transactions which activate the programs. One to one, one to many, many to one or many to many relationships may exist between any two components.

The 13 application system parameters shown in table 7.1 can define the application database, the application programs, the user transactions, and the interrelationships between these components. The purpose of these parameters is to describe application systems at a logical (conceptual) schema level; thus physical properties of blocksizes, pointer types, programming languages, and the like, have been excluded. The database parameters can describe database size and consistency relationships between data records. The program parameters show the program input, output, database accesses and interactions with other program (parameter $P_m(p,p1)$). The transaction parameters show the

TABLE (7.1) : MAIN DADP PARAMETERS

1). MAJOR OPTIONS VARIABLES:
- MODE = 0 to optimize program and daatset allocations,
  = 1 to 5 for predictive analysis
- TH = Grouping threshold
- UNCONS = 1 for unconstrained evaluations, 0 otherwise

2). ALLOCATION VARIABLES (INPUT)
- $\alpha(d,i)$ = 1 if dataset d is allocated to node i, 0 otherwise
- $\beta(p,i)$ = 1 if program p is allocated to node i, 0 otherwise

3). APPLICATION SYSTEM PARAMETERS (INPUT)

a). Data Description
- D_n = number of datasets in the application system.
- $D_c(d,d1)$= consistency relationship between dataset d and d1; implies that d1 must be updated whenever d is updated .
- $D_s(d,x)$ = size (x=1) and occurrence (x=2) of tuples in dataset d.

b). Program Description
- P_n = number of programs in the application system
- $P_r(p,d)$ = number of reads issued from program p to dataset d.
- $P_w(p,d)$ = number of updates issued from program p to dataset d.
- $P_c(p,d)$ = number of consistency updates issued from program p to dataset d.
- $P_I(p)$ = average number of input records read by program p
- $P_o(p)$ = display messages written by program p.
- $P_m(p,p1)$= No. of temperory records written by program p that are to be received by by program p1. Note that p and p1 can execute in parallel if $P_m(p,p1)=0$.

c). Transaction Description
- T_n = total number of transaction types in the application system.
- $T_e(p)$= no. of times program p is activated per invokation of transaction type t
- $T_a(k)$ = No. of times transaction type t arrives at node k for a given observation period.

4) SUPPORT SYSTEM PARAMETERS (INPUT)

a). Basic Parameters:
- N_n = No. of nodes in the network
- $C_S(i)$ = cost of storing at node i
- $C_L(i)$ = cost per local service request at node i
- $C_R(i,j)$ = cost per remote service request exchanged between nodes i and j

b). Transaction Processing Parameters
- Q_{rt} = the routing graph: parallel (1) or ring (0)
- Q_{da} = the data access mode: remote (1) or local (0).
- SRCH(k,x) = the search strategy for the queries arriving at node k.
- COPY = the maximum number of dataset copies allowed

c). Update Synchronization Parameters
- U_{cp} = location of the commit-processor: distributed (1) or centralized (0).
- U_{lm} = location of locks/stamps: distributed (1) or centralized (0).
- U_{pg} = the amount of "piggybacking" in synchronization:
  = 0 complete piggyback
  = 1 partial piggyback (request or response)
  = 2 no piggybacking
- U_{cr} = the locks/stamps are core resident (0) or not (1)
- U_{sl} = the lock/unlock requests are broadcast (0) or addressed (1).

d). Network and Miscelaneous Parameters:
- $A_a(z)$ = the availability of link z
- $B_{tp}(i,j)$ = the network topology
  = z, the communication link between nodes i and j
  = 0 otherwise
- B_{nc} = the network control = 1 (distributed), 0 (centralized)
- $B_{rt}(x)$ = routing frequencies:
  x=1 - primary path
  x=2 - secondary path
- E(x) = the mean service time of device x
- STOR(i) = storage capacity of node i
- $F_D(d,i)$ = 1 if dataset d can be allocated to node i, 0 otherwi
- $F_p(p,i)$ = 1 if program p can be allocated to node i, 0 otherwi

5). OUTPUT PARAMETERS (OBSERVABLES)
a). ST(x) = storage occupied at node x
b). L(k,i) = local service requests executed at node i due to arrival of a transaction at node k
c). R(k,i,j) = remote service requests exchanged between nodes i and j due to arrival of a transaction at node k
d). A(k) = availability of a transaction when it arrives at node k
e). W(k) = total response time of a transaction when it arrives at node k.

transaction arrival rates at various nodes and the programs activated by the transactions. These parameters, defined and discussed in chapter 2, can represent a wide range of distributed and centralized application systems, and should be compared with the following three traditional parameters used by most data allocation schemes:

. size of file d - S(d)

. number of reads from node k to file d - R(k,d)

. number of updates from node k to file d - W(k,d)

These three parameters cannot represent data relationships, cannot distinguish between program input, output and data accesses, cannot show exchange of information between programs, and are thus limited to simple one program/one dataset transactions. For example, a transaction which performs a join between two relations, d1 and d2, at two different nodes cannot be represented by these three parameters because the transfer of information between the two relations cannot be modelled. However, we can easily represent this situation by introducing a program p1 to transfer the relation d1 to program p2 which actually performs the join between d1 and d2.

DDP SUPPORT SYSTEM DESCRIPTION: A support system consists of a machine subsystem and a control subsystem. In a DDP environment, these two subsystems can be viewed in terms of local and global components (see figure 7.1a). Our objective is to describe and analyze the global components of the DDP support systems.

The support system parameters shown in table 7.1 were introduced and discussed in chapters 4 and 5, and describe the machine (hardware) subsystem, the distributed transaction management, the distributed database management (update synchronization), and the network management parameters, at global level. The machine subsystem parameters allow us to define the processing speeds, availabilities, capacities, and the interconnections (topologies) of the network. The distributed transaction management parameters can be used to define the site selection

strategies, the transaction routing graphs, and the data distribution and access modes. The distributed database management parameters focus on the update synchronization aspects of DDBMS and allow us to specify the centralization/distribution of lock management which shows the location of lock tables and the lock manager, and the commit processor which is responsible for updating the duplicate data. In addition, the core residence of locks, the piggybacking of locks with the actual read/writes and the broadcast (sending to all nodes) versus addressing (sending to specific nodes) mechanisms for lock requests can be defined. The network management parameters specify the centralization/distribution of network control and the routing frequencies. The miscellaneous parameters allow us to specify the resource utilization costs of storage, local processing and remote processing. Note that the communication (remote processing) cost can be specified in terms of number of messages transported, volume of information transported, or a flat monthly rate.

It is important to note that our objective is to define diverse DDP support system configurations by using a minimum number of parameters, rather than defining one configuration, or one component, in great detail. This is not an easy task since a large number of views, implementations and classifications of DDP support systems exist (see section d of the bibliography). We have reviewed the literature extensively and classified the DDP support systems in terms of transaction processing, distributed database and network management components. Figure 7.1b lists the main functions performed by these components and shows which functions can be represented by our parameters. The main features of these components were categorized into various classes, and an attempt was made to represent these classes by using 0-1 parameters, so that the effect of these features can be represented as analytical expressions for the local and remote service requests. In some cases, the 0-1 parameters can only represent extreme cases, but in several cases binary

choices are quite realistic. For example, the lock management is either centralized where all locking/unlocking activities are limited to one node, or distributed where every node has a lock manager. Partially replicated lock management where a few nodes manage all locks have not been reported.

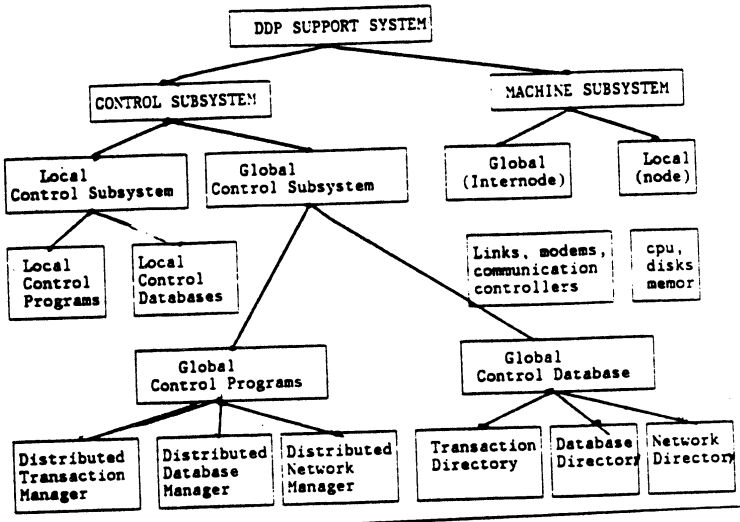
Chapter 4 showed how the transaction management and update synchronization parameters can represent existing transaction processing and update synchronization algorithms, and chapter 5 discussed the description of network topologies and management. In short, the support system parameters shown in table 7.1 can represent, albeit approximately in some cases, a large number of real life DDP support systems with different communication features, transaction processing options and centralized or distributed synchronization algorithms. As can be seen, these parameters do not represent query/data translation, security checking and separate directory location (it has been assumed that the directory distribution is identical to the control program distribution). Moreover, the situations that do not fall into binary classes cannot be represented adequately. However, we can use these parameters to effectively study the combined effect of several support system factors on program and data allocation. Appendix J shows how an existing application system and support system of an actual organization are represented and by the parameters shown in table 7.1.

7.2 Modelling Results

The input variables of program and data allocation in table 7.1 are categorized as the decision variables (controllables) and the output variables of storage cost, local processing cost and communication processing cost are used to form the objective function which represents the total cost of running the application system. Table 7.2 shows the conceptual model which illustrates the interdependencies between the

FIGURE 7.1: SUPPORT SYSTEM DESCRIPTIONS

a). DDP COMPONENTS



b). FUNCTIONS OF DISTRIBUTED CONTROL PROGRAMS

	REPRESENTATION
DISTRIBUTED TRANSACTION MANAGER:	
. Determine the transaction procedure of an arriving transaction	. Application description
. Decompose the transaction into subquerries	. Each program group = subquery
. Construct an operating policy	. SRCH parameter
. Select program and data sites and construct a routing graph	
. Route the transaction through the network and control the flow of the transaction	. Q_{rt} and Q_{da} parameteter
. Accumulate the results of the transaction and display the results	. Assume arriving node is result
DISTRIBUTED DATABASE MANAGER:	
Database Controller	
. Parse, interpret and translate all the database requests	. Transalations not handled
. Check authority and allocate resources	. Not handled
. Log the before/after data images for recovery	. Implicit
Lock/stamp Manager:	
. Generate, receive, parse and interpret all the lock/unlock requests	. U_{lm}, U_{pg}, U_{sl} parameter
. Read and update the lock/stamp tables	. U_{cr} parameter
. If the locks cannot be granted, then either put the transaction in wait state, or abort.	. Wait assumed
Commit Processor:	
. Handle all the transaction terminate and/or checkpoint calls.	. U_{cp} parameter
. Update the consistency related and duplicate data in the network at checkpoint, transaction termination or as offline "store forward"	. D_c, P_c parameters and a dec. var.
DISTRIBUTED NETWORK MANAGER:	
. Parse and translate all the internode requests	. Implicit
. Initiate/terminate all the sessions between nodes and control all the inter-node traffic	. B_{nc} parameter
. Select paths between nodes and route the traffic between nodes on selected paths	. B_{rt} parameter
. Padd the messages with the link control data and perform all the error recoveries	. Not handled

input and output variables. The rows of table 7.2 show the independent (input) variables to DADP, when it is used in predictive mode, and the dependent (output) variables are shown as columns. The effect of an input variable on an ouput variable is shown as X, a blank indicates no effect. The predictive model, conceptually represented by table 7.2, allows us to answer the following questions:

1)Application system questions:

- What is the effect of allocating programs and datasets to to a given node, on the output variables?
- What is the effect of database and program size on output variables like response time and cost of running an application?
- Should the related data be allocated together or not?
- What effect do the program to dataset access frequencies have on the various output variables?
- When should the data be replicated or restricted to one node?

2)Support system questions:

a). Transaction processing questions:

- Should the site selection be based on minimum cost, minimum distance or some other criteria?
- Is it better to circulate the query in the network or should one node, usually the arriving node, control the flow of the query by starting subqueries which process simultaneously and send results to the controlling node?
- Is it always better to run a program at the arriving node or is it better to run the program wherever the data resides?

b). Update Synchronization questions:

- Should updates be synchronized immediately or conducted as a store forward application at a later time?
- If updates are to be synchronized, what algorithm(s) should be employed? Should the synchronization algorithms be

centralized or distributed?

c). Communication system and physical resource factors:

. What is the effect of network topologies and network control on the output variables?

. What is the effect of node processing and communication speeds on the response time?

A major result of this study is that most questions can be answered quantitatively by using the following procedure:

1. Define the application system by using the DADP parameters. This defines the database, the programs and the transactions of the application system that needs to be studied.
2. Define the support system configuraurations to be studied. This defines the machine subsystem, transaction processing, update synchronization, and network management features to be studied.
3. Define the resource utilization costs. Unimportant costs can be ignored by using zero values and the flat, per message or per character values can be used to compute the communication costs by using an input parameter "COMM" to DADP (not shown in figure 7.1). This parameter has three possible values: flat (0), per message (1) and per character (2).
4. Choose the predictive mode of DADP, and allocate data and programs to specific nodes for analysis. If needed, use the table input feature of DADP to read a set of program and data allocations from a table.
5. Run the DADP in predictive mode. For each run, DADP prints a vector of ouput variables (storage cost, local processing cost, remote processing cost, transaction availability, transaction response time and software costs). By changing some options, detailed printouts which show local and remote service requests,

processing delays, locking delays, probabilities of conflict etc can be printed.

6. Vary the application system and/or the support system and reiterate to answer the questions stated above.

This procedure can be used to answer specific questions about specific situations. An application design methodology and a case study will illustrate this point later. However, it is very difficult to make general comments about the behaviour of the output variables mainly because of the number of interdependencies and the effect of special cases. The discussion that follows lists only the significant observations regarding the dependence of the output variables on the input variables.

7.2.1 Storage Cost

The storage cost primarily depends on the dataset size and the dataset allocations. It is also affected by program size and program allocations in a similar manner; however, this effect can be usually ignored due to the relative small size of programs versus datasets. The storage cost always increases monotonically as a function of data duplication.

7.2.2 Local Processing Cost

The local processing cost consists of the cpu cost plus the i/o cost. We have measured these costs in terms of a logical entity, a local service request L which counts the interactions between programs and datasets at the same node, and is given by:

$$L(k,i) = L_{pd}(k,i) + L_{lk}(k,i)$$

where L_{pd} represents the data access and manipulation activity, and L_{lk} is the local activity due to lock/unlock. An examination of equation 4.31 shows that: a) $L_{lk}(k,i) = 0$ if $U_{cr} = 0$, i.e. if locks are core res-

TABLE (7.2) = RELATIONSHIPS BETWEEN THE INPUT AND OUTPUT VARIABLES

	INPUT \ OUTPUT	Storage Cost	Local Proc. Cost	Communication Costs			Resp. Time			Trans. Avail.	Soft. Devel
				NP	PP	PD	PDEL	QDEL	LDEL		
APPLICATION SYSTEM FACTORS	Data Alloc.	X	X			X	X	X	X	X	
	Program Allocation			X	X	X	X	X		X	X
	Transaction Arrival Mode			X				X		X	
	Program to Data Access		X			X	X	X	X	X	
	Program I/O & Relationships			X	X		X			X	X
	Program Size	X									X
	Data Size	X									
	Data Relationships	X				X	X				
SUPPORT SYSTEM FACTORS	Site Selection		X	X	X	X					
	Routing Graphs			X	X		X				
	Data Access Strategies					X					X
	Commit Proc.		X					X	X		
	Lock Mgt.		X			X		X	X		
	Piggy Backing		X			X	X		X		
	Core Res. & Others			X	X	X	X	X		X	
	Topology & Routing			X	X	X	X	X		X	
	Device Reliability									X	
	Device Speed						X	X	X		
	Dir. Location										
	Database Design	X	X			X	X				
	Prog. Design	X	X			X	X				X
	EQUATIONS	4.27	4.29 4.31	4.32a through 4.32e			5.41	5.50	5.61	5.10	5.21

ident then the effect of locking on local processing can be ignored. The major contributing factors to L_{pd} activity are the data allocations and program-to-data access frequencies. This variable is independent of the transaction processing and update synchronization, and is a monotonically increasing function of data duplication, if the data is updated. L_{lk} is dependent on number of lockable data items, centralization/decentralization of lock tables, and the number of reads and writes. This variable is significant especially if the lock/stamp tables are not core resident. The locks can reside in core if database is small, locking granularity is large or special locks are used (for example the "hole" lists <GARC79d>). The centralized lock management minimizes L_{lk} for cases when duplicate data exists. This is because in distributed lock management, the locks are set/reset at every node where the copy of the dataset resides; howewver in centralized lock management, locks are set/reset at the central node, independent of the number of copies of dataset. The overhead due to lock/unlock increases with the number of read/write activities.

The total local processing cost does not increase due to data duplication for read only transactions, since data is only read from a single node. However, the local processing cost due to data duplication increases for update transactions due to duplicate activities. It is also shown in chapter 4 that local processing activity is generally independent of the arriving node. Once LSR is found, then the cpu and disk i/o time, CPU(i) and DIO(i), at node i can be found by using local performance measures like the "record access" of Teorey and Fry <TEOR80e>.

7.2.3 Communication (Remote) Processing Cost

The communication processing cost, also referred to as the remote processing cost, is dependent on the transaction arrival node, data allocation, program allocation, program to dataset access frequencies,

program i/o, update synchronization and transaction processing options. Although it is very difficult to make general remarks about the relative importance of these factors, we consider the remote processing cost in terms of the node to program (NP), program to program (PP) and program to dataset (PD) traffic:

$$R(k,i,j) = R_{np}(k,i,j) + R_{pp}(k,i,j) + R_{pd}(k,i,j) + R_{lk}(k,i,j)$$

The node to program traffic (R_{np}) and the program to program traffic (R_{pp}) is influenced by transaction processing but are not affected by update synchronization. The program to dataset traffic (R_{pd}) and the lock/unlock remote traffic (R_{lk}) is not affected by the transaction processing extension, but is modified due to the update synchronization algorithms. The analytical expressions for these costs have been developed and analyzed in chapter 4.

In short, the NP and PP traffic depends on the program allocations, program i/o, site selection and routing graphs, and is a monotonically decreasing function of program duplication (see chapter 4). The PD traffic includes the data processing plus the lock/unlock activities and are dependent on the data allocation, program allocation, program to data access frequencies and various other factors shown in table 7.2 and is a convex function of data duplication, even when update synchronization cost is included. Equations (4.32a) through (4.32e) show the dependence of PP, NP and PD traffic in terms of the input variables.

The network topologies, routing and controls also influence communication cost (see equation 5.1 for analytical expressions). In addition, most of the communication costs are calculated based on a per message cost basis. However, a variety of communication cost functions can be defined: flat rate, time rate and transport volume. The behaviour of the transport volume is similar to the per message cost and the constant costs are independent of the program/data allocations and update synchronization algorithms.

7.2.4 Transaction Response time

This variable, RT, is dependent on a large number of variables and can be represented in terms of the processing delays (PDEL), the queuing delays (QDEL) and the locking delays (LDEL):

$$RT = PDEL + QDEL + LDEL$$

In table 7.2, the processing delays are indicated to be dependent on the program/data allocations, program i/o, program to dataset access frequencies and the transaction processing options which determine the parallelism. Experimentation has shown that PDEL behaves very similar to the local plus remote processing cost when no parallelism is used. This is because the processing delay is a function of local and remote processing activities. Thus the processing delays are a convex function of the data duplication.

The queuing delays QDEL are dependent on service rates of various devices and the device arrival rates which are a function of data allocation, program allocation, program to dataset accesses and other factors shown in table 7.2. The locking conflict delays are a function of update synchronization algorithms and transaction characteristics (equations 5.61 and 5.62). Due to the intricate interactions, it is difficult to make general comments about the behaviour of queuing and locking delays; however we can make the following observations about the dependence of these variables on the update synchronization algorithms.

The 2PL algorithm and the basic time-stamped algorithm roughly yield the same queuing and locking delays, although the locking delays of the time-stamped algorithm are less than 2PL. This is because time-stamped algorithms concatenate the time-stamp with the read/write request, thus avoiding separate lock/unlock requests. The locking delays due to the Ellis Ring and Thomas Voting are usually large because the transaction goes through two passes in such cases: the circulation/voting pass and the actual processing pass. The centrally controlled algorithms (CCAA

and CCA) significantly increase the queuing delays at the central node and introduce negligible locking delays because the locking conflicts are eliminated by serially running the transactions at the central node. However, the centralized locking algorithms (CLAA and CLA) decrease the locking as well as the queuing delays because in such cases the transactions execute at the arrival node; only the locks are centralized. The pessimistic algorithm eliminates queuing delays but leads to extremely large locking delays because of the extensive locking activity. Our experiments have shown that the pessimistic locking delays are greater than 15 to 20 times the average locking delays encountered due to other algorithms. The optimistic algorithms on the other hand eliminate the locking delays altogether.

It should be noted that the problem of response time calculations is generally so complex that detailed simulations are needed for prediction. We once again recall that we are not considering the local design factors, which can also significantly influence the response times.

7.2.5 Transaction Availability

The transaction availability is dependent on the program allocations, data allocations, program to dataset access frequencies and the network topologies, routing and component reliabilities. Basically, transaction availability increases if the datasets and programs utilized by the transaction are duplicated. However, it should be emphasized that the transaction manager must have the intelligence to be able to access duplicate copies in case of failures.

7.2.6 Combined Effect and Practical Insights

In general, if data is read-only and the storage and local processing costs are negligible, then it is better to replicate data. However, it was found that if data write/read ratio is greater than 10%, then it is

better to keep a single copy of the data. This result has been also reported by other researchers <MITS82d, BARB82d>. Perhaps the most practical solution is to keep data at two sites: the site where it is most frequently used and the "central" site where a master copy is kept. In addition, it is better to keep related data on the same node when the communication transmission rates are slower than local data transmission rates. In local area networks, however, several low speed processors can be interconnected through high speed links. It should be emphasized that it is not always better to allocate programs and data at the most frequently referenced node, especially when the arriving node is slow or congested and a high speed link exists to a faster and lightly loaded system. The exact answer depends on detailed analysis obtained by using DADP. It is important to note the effect of data duplication on transaction availability. Although complete data replication theoretically gives 100% availability, for most practical cases, availability of data at another node that can be accessed through alternate leased lines or dial-up facilities gives approximately 100% availability since the probability that two nodes will fail simultaneously is extremely small with current computer systems consisting of dyadic processors and the like.

It appears that direct access and parallel transaction processing algorithms perform better than the local access and serial routing transaction processing algorithms for transactions that can be decomposed into parallel subtransactions. For short transactions, the optimistic algorithms work very well because for such transactions the probability of conflicts is negligible, and thus all the locking/unlocking overhead is unnecessary. For very long update transactions involving thousands of updates, the pessimistic algorithms perform better because the probability of conflicts is large. For systems with moderate size transactions, the centrally controlled synchronization algorithms perform quite well <GARC79d>. It is difficult to gain general insights into the

network topologies because their performance is dependent on the network control techniques being employed. For example, the star topologies work quite well for centrally controlled networks and centrally controlled update synchronization.

7.3 Optimization Results and Computational Complexity

The computational efficiency of any algorithm can be discussed in terms of two factors:

K = no. of evaluations needed to reach the optimal

J = no. of computations per evaluation

Thus the total number of computations = J × K. For an NN node, PP programs and DD datasets, $K = 2^{NN(PP+DD)}$ in case of explicit enumeration and for DADP $J = NN^3 \times NR \times NP^2 \times DD^2$, where NR is the number of transactions. Note that J is a polynomial but K is a non-polynomial. For this reason, we attempt to minimize K. This is achieved by using the following techniques. First, the problem is decomposed into several subproblems: the program allocation subproblem and the data allocation subproblem. The latter can be further decomposed into single dataset allocation subproblems. These decompositions give the following value of K:

$$K = PP \times 2^{NN} + DD \times 2^{NN}$$

The proof of these decompositions is given in appendices E, F and G. The approximations needed to achieve decompositions introduce minimal errors (about 15% error) but significantly reduce the computational requirements of the problem, even though K is still non-polynomial. Second, K is reduced to a polynomial by using the properties of the objective function for data and program duplication because the objective function, which represents the total cost of running an application system, increases rapidly with data duplication, thus limiting the number of copies in the network, and decreases rapidly for program dupli-

cation, thus keeping the number of program copies close to maximum. Thus, K is now reduced to:

$$K = PP \times NN^{x} + DD \times NN^{y}$$

where x and y vary between 0 to 5 for 10 to 15 node networks. For example, if only single copies of data are allocated and all programs are replicated, then x=0 and y=1. Third, the grouping scheme is employed to significantly reduce the optimization run-time by clustering datasets and programs together. Thus, K is reduced to the final value:

$$K = PP' \times NN^{x} + DD' \times NN^{y}$$

where $PP' < PP$ and $DD' < DD$. Our experiments with two program, three datasets in five node networks has shown that K varies between 20 to 60 (see table 6.4).

We can make the following additional observations about the optimization. First, the program allocation problem can be solved by using spatial dynamic programming while a hierarchical heuristic is most suitable for data allocations. Second, the behaviour of the total cost due to data allocations is convex even when complex factors like the update synchronization and network topologies are included. This is an important observation and is the basis of data allocation scheme. Third, the constraints help reduce the size of the problem. The storage constraints and policy constraints which restrict the allocation of data and programs to nodes, are of particular interest since these constraints can be quickly checked. Fourth, the local processing cost increases the total cost of duplicating data and thus tends to reduce the number of optimal copies. Finally, update synchronization also increases the total cost of data duplication and further tends to decrease the dataset copies for optimal allocations. We have found that for most practical case, single dataset allocations are optimal for transactions which involve more than 10% updates due to the additional costs of total processing and update synchronization. However, data duplication is efficient for read only data because it increases the

availability of data and also minimizes the response time because the data can be accessed at the arrival node.

7.4 Comparison with other Efforts - A Numerical Example

Over twenty resource allocation schemes have been published in the literature. These schemes, listed in section c of the bibliography and reviewed in chapter 1, can be categorized in terms of the data allocation, program allocation, data/program allocation and data/communication allocation schemes. Table 7.3 compares our model with existing approaches. A major contribution of this research is that DADP can be easily translated/transformed into most of the available schemes by choosing appropriate parameters. Appendix I shows formally how DADP can be reduced to Casey <CASE72c>, Mahmood & Riordon <MAHM76c>, Khabbaz <KHAB80c>, Morgan & Levin <MORG77c, LEVIN75c> and Chen <CHEN83c>. These schemes are selected because:

i). Morgan and Levin <MORG77c, LEVIN75c> were the first to consider program as well as data assignment, which is the focus of this research. Thus an analysis of their work is important. It is shown in appendix I that DADP can be reduced to this model by ignoring local processing cost, restricting transaction processing to parallel routing graph and local data access, ignoring update synchronization costs (assume optimistic algorithm) and ignoring the consistency relationships.

ii). Casey <CASE72c> was the first to show that the optimal file allocation problem can be viewed as well known plant location problems. His work has laid the foundation of several later attempts (see for example Suri <SURI79c>). His work is used as a prototype of the optimal file allocation problems. DADP can be reduced to this model by replicating programs to all nodes, ignoring local processing costs, ignoring consistency relationships, assuming parallel routing graph and remote

TABLE (7.3) = COMPARISON OF ALLOCATION SCHEMES

		OBJECTIVE FUNCTION			APPLICATION DESCRIPTION			SUPPORT SYSTEM DESCRIPTION			SUPPORT SYSTEM & MANAGMT RESTRICTIONS				USER PERFORMANCE REQUIREMENTS			
															TRANSACTION AVAILABILITY		TRANSACTION RESPONSE	
		STOR COST	LOCAL PROC COST	RMT PROC COST	DATA DESC	PROG DESC	TRAN-SACT. DESC	TRAN PROC MODE	UPDATE SYNCH. ALGOR.	NETWK TOPOL. & CNTL	STORAGE CAPACITY	LINK CAPCTY	DATA RESTR	PROGM RESTR	FILE	TOTAL	COMMUNIC	TIME TOTAL
Program & Data Allocation	UMAR83	*	*	*	13	13	13	14	14	*	*	*	*	*	*	*	*	*
	MORG77	*		*	1	2	2				*		*				9	
	BUCK79	*		*	1		3											
	FISH80	*		*	1	2	2											
	MARC81	*		*	1	*	3				*				12		10	
Data Allocation	CHU69	*		*	1		3				*						10	
	CASE72	*		*	1		3											
	LOOM76		*	*	*		3											
	CHAND76	*		*	1		3											
	TABA78			4			3											11
	SURI79	*		*	1		3				*	*			12		9	
	GOFF80	*		*	1		3		5									
	CHANG81	*		*	*	6	6		5									
	CHEN83	*		4			*		*									
Program Allo-cation	GYLY76			7		*					*						9	
	BRYA81			7		*												11
	McENT81		*	7		*											9	
	DOTY82			7		*											9	
	WHIT70	*		*	1		3			8								
Data & Comm. Allo-cation	CASE73	*		*	1		3			8					*		9	
	MAHM76	*		*	1		3			8					12		10	
	CHANG77	*		*	1		3			8					12		9	
	IRAN81	*		*	1		3			8		*			12		10	

LEGEND FOR TABLE ENTRIES:
- . * indicates that the factor is fully included by the researcher
- . n, where n is an integer, indiactes that the factor is partially considered by the researcher. See note n for explanation.
- . blank indicates that the factor is ignored by the researcher

NOTES:
1. Only describes data (file) size. Relationships between data are ignored.
2. Only program to data read/update operations are defined. The relationships between programs are ignored.
3. The transaction is described in terms of node to file read/update traffic. Cannot model complex transactions which involve operations on more than 1 file.
4. The objective function is not a dollar cost. It represents the transaction turnaround time (response time).
5. Only simple/specialized cases of update synchronization can be handled. For example, Ellis ring algorithm.
6. Can describe relational view of data.
7. The objective function is not dollar cost. Instead, processing time or some "benefit" function is used.
8. Network design is a decision variable. May include topology design or channel capacity assignment.
9. Communication delay is expressed in terms of the "maximum expected internode delay", an input parameter.
10. Node to node communication delay; includes queuing at the communication links (channels).
11. Communications + local + queuing delays. Excludes delays due to locking.
12. Availability of a file from a node. Does not consider the node to program and program to file availability (total transaction availability).
13. The application system description is adequate for a logical description of the application system. Physical description (blocksizes, pointer types, programming language etc) cannot be handled.
14. Most synchronization and query optimization algorithms can be represented.

data access, ignoring update synchronization costs and all the constraints (see appendix I).

iii). Mahmood & Riordon <MAHM76c> and Khabbaz <KHAB80c> combine the problem of optimal file allocation with communication system design problems like link capacity (speed) assignments and network topology. However, the program allocation is ignored. Their work is used as a prototype for the combined file/communication system design problem. DADP can be reduced to this model by assuming a flat communication cost, ignoring local processing cost and, except the constraints, making the same assumptions as used for Casey (see appendix I).

v). Chen <CHEN83d> has recently formulated and solved the problem of data allocation which includes the update synchronization cost. It is important to study this work because we also include update synchronization costs. Detailed discussion of representing Chen's model by DADP is given in appendix I.

Table 7.4 numerically compares and contrasts our results with these schemes. For comparison, an application system consisting of a single database and a single program that is to be allocated to a 5 node fully connected network with basic 2PL synchronization algorithm is considered. We assume that the transaction availability must not be less than 70% and transaction response time must not exceed 5 seconds. The table shows the data allocation and the objective function computed for each allocation. Let us illustrate this table by considering the first row. The data allocation 10100 shows that the dataset is allocated to nodes 1 and 3. This allocation yields a cost of 11200, 12950 and 1825 when the formulations of Casey, Morgan and Levin, and Mahmood and Riordon are used, respectively. The differences in the cost exists because Casey ignores all the local processing and node to program costs; Morgan and Levin ignore the local processing cost; and Mahmood and Riordon use a flat communication cost function. This allocation is marked infeasible

TABLE 7.4: COMPARISON WITH OTHER EFFORTS

NOTES AND EXPLANATIONS
- The application system consists of one program and one dataset.
- The dataset size is 100 K bytes, the program issues 35 reads and 5 writes to the dataset, receives 30 input mesaages and displays 80 messages.
- The communication cost is 5 units per remote message and the local cost, if applicable, is 2 units per local service.
- The support system consists of a fully connected, 5 node network, with basic 2PL synchronization algorithm. Node 4 does not have enough storage for dataset storage. All other nodes have infinite storage capacity.
- The data allocation is represented by a binary vector which shows a 1 to indicate allocation. For example, the vector 11000 idicateS that the dataset is only allocated to nodes 1 and 2.
- The units for cost are arbitrary
- Total cost = (storage + local processing + remote processing) cost
- The program allocation is optimized for Morgan & Levin and Umar (p is allocated to nodes 1 and 2). For all other approaches p is replicated.
- Total cost of 999999x indicates infeasible solution where:
  - x = 1 indixates response time violation
  - x = 2 indicates availability violation
  - x = 3 indicates storage violation

TOTAL COST

DATA ALLOCATION	CASEY	MORGAN & LEVIN	MAHMOOD & RIORDON	CHEN	UMAR
10000	7600	11600*	99999992	63600	99999992
01000	5600	17600	1225*	57600	13620
11000	7900	12950	1825	115950	20880
00100	8850	17600	1225	56850	12720*
10100	11200	12950	1825	99999991	99999991
01100	9200	18950	1825	99999991	99999991
11100	11550	14300	2425	99999991	99999991
00010	3350	12350	1225	47350*	99999993
10010	10950	12350	1825	110950	19080
01010	8950	12950	1825	99999991	99999993
11010	11300	18950	2425	99999991	99999993
00110	12200	14300	2425	99999991	99999993
10110	14550	18950	3025	99999991	99999993
01110	12550	14300	1225	99999991	99999993
11110	14900	20300	99999991	99999991	99999993
00001	3100*	15650	1225	48100	14520
10001	10700	12350	1825	106700	21780
01001	8700	12950	1825	99999991	99999991
11001	11050	18950	2425	99999991	99999991
00101	11950	14300	1825	99999991	99999991
10101	14300	18950	2425	99999991	99999991
01101	12300	20300	2425	99999991	99999991
11101	14650	15650	99999991	99999991	99999991
00011	6450	13700	1825	99999991	99999993
10011	14050	14300	2425	99999991	99999993
01011	12050	20300	2425	99999991	99999993
11011	14400	15650	99999991	99999991	99999993
00111	15300	21650	3025	99999991	99999993
10111	17650	15650	99999991	99999991	99999993
01111	15650	21650	99999991	99999991	99999993
11111	18000	17000	99999991	99999991	99999993

by Chen and us because the Casey, Morgan and Mahmood do not consider the response time constraints.

We make two major observations from table 7.4. First, the costs and the optimal decisions differ among the various approaches. For example, we note that the optimal allocation for Casey, Morgan and Levin, Mahmood and Riordon, Chen and us are 00001, 10000, 01000, 00010 and 00100, respectively, each producing a different cost. The difference is due to the elimination of local processing and update synchronization costs by most researchers. Second, several decisions, including the optimal choices, become infeasible as the physical resource limitations are introduced. For example, we note that the allocation 10000 is optimal for Morgan and Levin, but is marked infeasible by Mahmood, Chen and us. This result suggests that quick optimal search routines which do not take into account the real system constraints can lead to optimal but infeasible results. This result also illustrates the strength of generalized allocation schemes like ours which optimize total costs subject to real life constraints.

7.5 Distributed Application System Design Methodology

A methodology which uses DADP as an aid to conduct design of application systems in distributed systems is now presented. A generalization of the traditional centralized data processing (CDP) design approaches to take into account the DDP considerations was presented in chapter 1 and consists of the following major phases:

1. *Requirement formulation and analysis phase* which involves establishment, derivation and documentation of the application system requirements. This phase consists of several specific steps: determination of application objectives (short range and long range); specification of the reports to be generated; collecting

information about data usage (the organizational units which generate input, receive reports and modify data); definition of user constraints (response time, availability, integrity and security) and management restrictions (time and money to be spent on the application development, policies and standards); and documentation of the requirements by using manual techniques and/or requirement definition languages.

2. *Logical design phase* which is concerned with determining the database conceptual schema, number and sequence of programs and the interfaces between programs and data which include external schema and access frequencies. The specific steps of this phase are: selection of data entites and the relationships between the data entities; specification of programs which translate given data entities into required output data; and the sequence of programs needed to generate the required output.
3. *Distributed design phase* which is concerned with partitioning of the data and programs and then allocation of these partitions to the nodes of the DDP support system. Note that this phase does not exist in CDP.
4. *Local design phase* which consits of the physical database design (record clustering, access path design etc) and program design of the allocated components at each node. Basically, this pahse is physical design phase in CDP.

In essence, the requirements analysis and logical design phases are independent of the support system considerations and thus the same for CDP and DDP. Thus any of the available techniques can be used in these two phases <RAM78e, TEOR82e, TEICH77e>. We thus exclusively concentrate on the distributed design phase, since this phase represents the differential of activities between CDP and DDP.

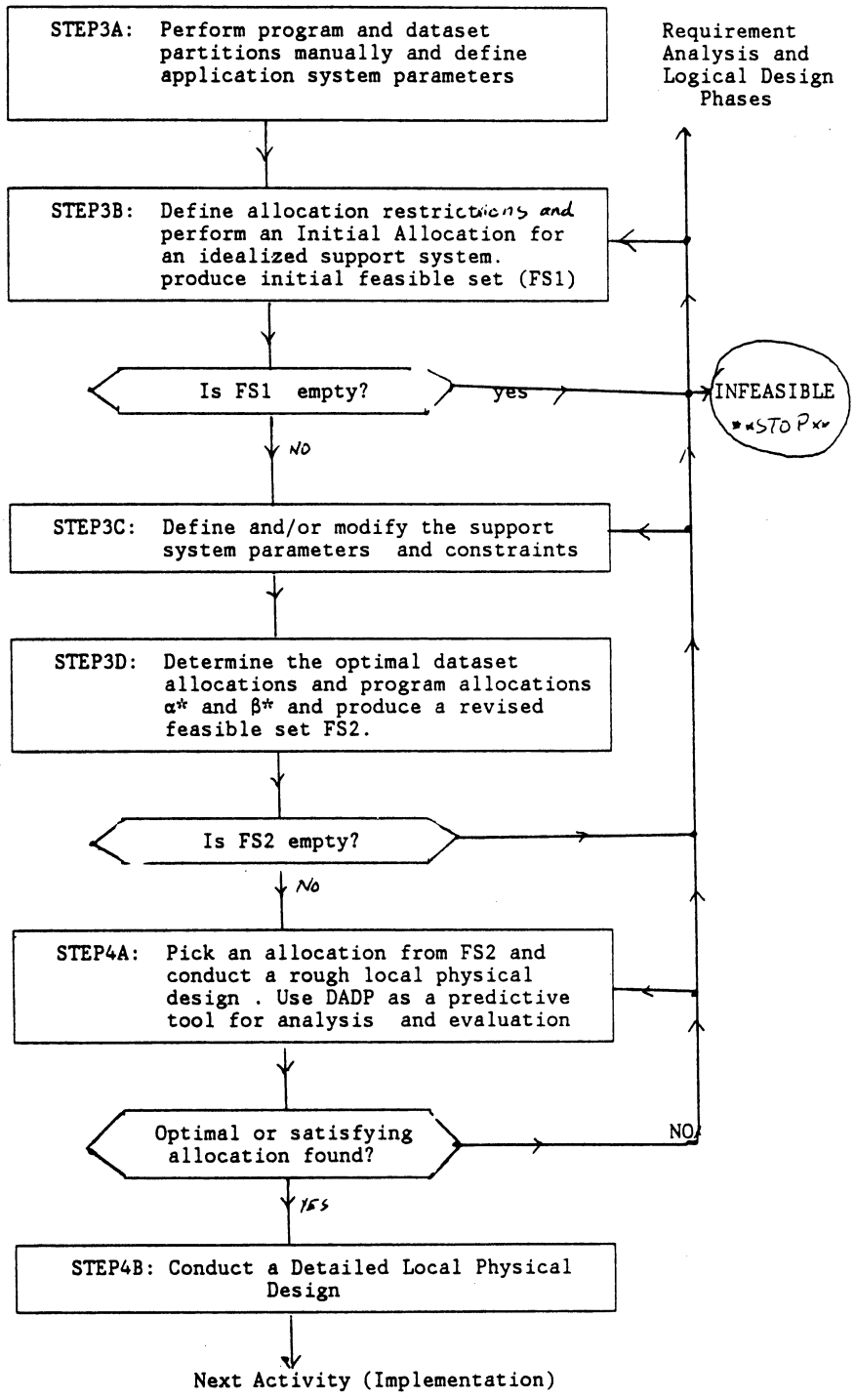
7.5.1 The Design Methodology

Figure 7.2 defines the design methodology that can be used in this phase. The main emphasis of this methodology is to eliminate infeasible solutions at an early stage by using approximate analysis and then study in detail the more promising alternatives by defining and including more detailed parameters. DADP is used in this methodology at several levels to define application and support system parameters, perform optimizations and conduct evaluations.

3A): PARTITIONING OF DATA AND PROGRAMS: The logical structure of programs and database produced by the logical design phase is decomposed into partitions. For example, a student database can be partitioned by schools, majors, faculty or campus. We are assuming that the logical database structure (schema) and user views (subschemas) are presented as entity-relationships. The partitioning step focusses on the problem of dimensionality in distributed systems which is concerned with instances of data and programs that can be allocated to different nodes. Different types of strategies can be used for partitioning: vertical partitioning, horizontal partitioning, axiomatic partitioning etc. We recommend that the database be partitioned in terms of data "specificity" which represents the access pattern from node to data <HEBA77b>. For example, a databsae is 100% specific if all the requests originate from the same node and is 0% specific if it can be accessed with equal frequency from any node. Then all the 100% specific partitions can be manually allocated to nodes where they are accessed, and eliminated from future analysis. See the example below for illustration.

After determining the partitions manually, the analyst defines the data and program partitions in terms of the application system parameters shown in table 7.1. These parameters allow definition of database partitions (size and relationships), program partitions (program to dataset access and modification frequencies, program i/o, interprogram exchanges and relationships) and transactions (arrival rates, programs

FIGURE (7.2): DISTRIBUTED APPLICATION DESIGN METHODOLOGY



activated). It was shown in chapter 2 that these parameters can be used to represent a variety of applications.

3B). INITIAL ALLOCATION AND ANALYSIS: In this step the user and management requirements that affect allocation are defined by using DADP parameters and an initial allocation which satisfies these requirements in an idealized support system environment is obtained. The objective of this step is to manually allocate the specific partitions to nodes, cluster the remaining partitions, if possible, by using the grouping scheme and eliminate clearly infeasible options. Once again, the motivation is to reduce the size of the problem as we proceed.

The following requirements can be defined by DADP parameters: the transaction response time, transaction availability and the management policiy constraints which may restrict the allocation of sensitive data and programs to specific nodes. The idealized support system is defined by specifing only one parameter: NN - the number of nodes in the network. DADP assumes the following default values: fully connected and reliable network, fast node and communication speeds (10 millisecond. per local service request and 100 millisecond per remote service request), no update synchronization costs and softwrae homogeneous nodes. The default values used by DADP can be easily overriden. DADP is executed to get an initial feasible set (FS1) consisting of all feasible allocations. Notice that if FS1 is empty, then the algorithm is terminated.

3C). SUPPORT SYSTEM DEFINITION: A support system is defined or modified in this step by using the transaction processing, update synchronization, network configurations, processing speeds and other support system parameters shown in table 7.1. The resource constraints of the support system (storage available, communication link capacity) are also defined. These parameters allow definition of a very wide variety of centralized as well as distributed support systems with different transaction processing options, update synchronization algorithms and network configuration with heterogeneous nodes. This step is used for every proposed support system configuration; however, DADP allows spec-

ification and analysis of several support system configuration in one run.

3D). DETAILED ALLOCATION: This step starts with FS1 and allocates programs and datasets defined in step 1 to minimize the total cost subject to the user and management constraints (step 3B) and support system considerations (step 3C). The infeasible solutions from step 3B are eliminated from the design by using the "policy" matrices F_D and F_P in the following constraints:

$$F_D(d,i) - \alpha(d,i) \geq 0 \quad \text{for all d and i}$$

$$F_P(p,i) - \beta(p,i) \geq 0 \quad \text{for all p and i}$$

where $\alpha(d,i)$ and $\beta(p,i)$ are the data and program allocation decision variables, respectively, and $F_D(d,i) = 1$ if dataset d can be allocated to node i, 0 otherwise. These two constraints can be used to exclude program and data allocations for management, political, technical or any other reasons. DADP is run extensively in this step in optimization mode to produce optimal allocations of programs and datasets and a revised feasible set (FS2) is produced. The allocations in FS2 are ranked according to cost. Note that the transaction availability and response time calculations are included in this step.

4A). INITIAL LOCAL DESIGN: An approximate local design is conducted to include the cpu, disk i/o and the effect of background workload on transaction response time. This step also involves generation of implementation schema which is the DBMS (data base mangement) processible structure (hierarchical, network, relational) at each node for the local DBMS. DADP does not allow definition of network or hierarchical schema at present (this is an interesting future extension), however the net performance effect is included by allowing approximate values which convert local service requests into cpu utilization and disk i/o (see chapter 5). The background workload can be defined by using the workload parameters described in chapter 5. DADP is used, in a predictive mode, to first determine the actual costs, response time and availabili-

ty of the optimal allocation so that any violation of feasibility constraints can be checked in detail. If there is no violation, then go to next step else pick a suitable allocation from FS2 for evaluation. If optimal allocations are highly desirable, then steps 3C, 3D and 4A can be repeated several times for feedback analysis which consists of using approximate values in steps 3C and 3D, modification of these parameters in step 4A, and reiterating till the optimal algorithm converges. Feedback analysis are used frequently in many hierarchical optimization algorithms like the generalized program and data allocation algorithm used by DADP <DEMP81f>.

After this step, the data and program distributions represent optimal or, acceptable, distributed application system design. Before proceeding with the next steps of detailed physical design and implementation, administrative approval should be sought. In order to proceed, parameters are needed for record clustering, access path definition, data storage design and program design activities. Definition of needed parameters and the design steps is beyond the scope of this research and is thus not handled by DADP. currently available techniques and the judgement of the analyst. Any of the avaialble techniques can be used in this step.

7.5.2 Illustration of the Methodology: A Case Study

In order to illustrate the methodology from a user point of view, let us reconsider the student registration system in a multicampus university environment. The database of the application system contains information about courses, students and departments. This information is used to register students, drop/add courses, and print class lists. The support system consists of three computing facilities, an IBM 3033 at main campus (node 1) and two IBM 8100 systems located at smaller campusses, interconnected through 9600 bits per second communication links

supported by the IBM System Network Architecture (SNA). Because of SNA, the 8100 systems are connected to the 3033 system in star network topology (see figure 7.3a). The 3033 system supports an IMS database manager, while the 8100 systems support a traditional flat file system. Let us now go through the steps of the application system design methodology.

STEPS 1 and 2: The user constraints specify that the response time for all the online transactions must not exceed 5 seconds and the availability of such transactions must be greater than 90%. The batch transactions may run slower, two to three minutes. The major management consideration is that the system has to be developed within a year under $100,000. The major support system consideration is that the available control subsystem (MVS, SNA, IMS) cannot be modified or replaced by another control subsystem.

Let us define the application system in more detail. The database consists of student records (STUDENT) the course records (COURSE), and the department records (DEPT), with the following information:

DEPT (Dno, Dname, Dinfo)

COURSE (Cno, Cname, Dno, Instructor, Cinfo)

STUDENT (Sno, Sname, Cno, Sinfo)

Note that a consistency relationship exists between the COURSE and STUDENT information because if a course is deleted then the student records must also be updated. Although a similar consistency relationship also exists between the DEPARTMENT and COURSE information, we will ignore it in this example because the departments are not deleted frequently. Three programs access this database:

. CLIST - for a given course, print the students enrolled.

. DLIST - for a given department, list the courses offered.

. DROPADD - drop or add a course to a student.

The following transactions activate the three programs:

. PRINT - for each department, print the courses offered

and/or the class roster for each course.

REGISTER - enroll, drop or add students

The PRINT transaction is batch and the REGISTER transaction is online; thus the 5 second response time and 90% availability restrictions apply only to the REGISTER transaction. The user demand is the number of transactions originating at the three nodes. We assume that the largest activity is on student registration during the enrollment periods.

STEP3A: We first build a global database schema and then partition the application system. Let us use relational model to describe the global schema. Figure 7.3b shows the global schema for the student database. To partition the database, we classify the database information into two categories: a) the campus specific information which shows the students, the departments and the courses which are offered by one campus only, and 2) the common information which shows the students, the courses and the departments that exist at different campusses. Thus we obtain the following partitions of COURSE, STUDENT and DEPT:

COURSE1, COURSE2, COURSE3, COURSE'

STUDENT1, STUDENT2, STUDENT3, STUDENT'

DEPT1, DEPT2, DEPT3, DEPT'

where STUDENT1 represents the partition of STUDENT that is specific to campus 1 and STUDENT' denotes the common partition. Figure 7.3c shows the partitions of the STUDENT database.

STEP3B: Let us assume that the database must satisfy the policy restriction that all the department information must be kept at the central node because department records contain financial data that must be kept under administrative control. In practice, the student records may also need to be kept at a central site for security, but we will ignore this restriction for the purpose of analysis. Note that the data restriction has two implications: allocation of department partitions can be ignored from the balance of this analysis, and access of department information from 8100 systems will cause a remote data request.

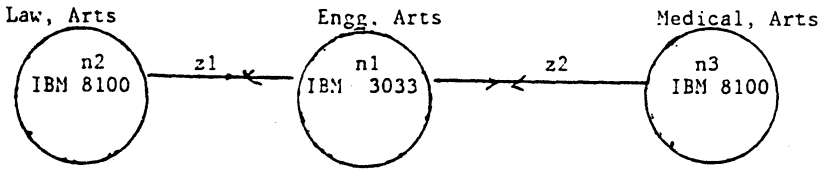
Now let us consider the other 8 partitions. We assume that all the campus specific data can fit on the accessing campus nodes and thus we initially allocate the dataset partitions STUDENT1, STUDENT2, STUDENT3, and COURSE1, COURSE2 and COURSE3 to nodes 1, 2 and 3, respectively. Figure 7.3d shows the allocation of the student partitions. We also assume that these explicit allocations satisfy the response time and availability constraints. In practice, we may need to run DADP in the predictive mode to verify that these allocations do not violate any constraints. If no constraints are violated, then these explicit allocations can be eliminated from future analysis. If not, then the datasets which violate the constraints are passed to the next step for evaluation.

After this explicit allocation, now only two dataset partitions, STUDENT' and COURSE', and three programs, CLIST, DLIST and DROPADD, need to be allocated for the balance of the design phase. Let us now run the DADP program for an idealized environment. TABLE 7.5a shows the input parameters defined in this step. The grouping scheme clusters STUDENT' and COURSE' partitions into a single dataset group, called DATAGRP, because the student and course information is related and accessed together frequently. In addition, no allocations are marked infeasible. Hence, FS1 represents all the potential allocations of DATAGRP to the 3 nodes uniquely or duplicted in some manner. The CLIST and DLIST programs are also clustered together into a program partition, called LIST, because the PRINT transaction activates these programs.

STEP3C: Now we can define and or modify the support system environment. In addition to the support system information provided above, we assume that this network allows 2PL and CCA update synchronization algorithms. As a first run, we choose the 2PL synchronization algorithm. Because of the management and technical considerations discussed above, we are restricted to the following support system configurations: a) star network topologies due to SNA restrictions, b) 8100 programming

FIG (7.3) : EXAMPLE OF LOGICAL, PARTITIONED & DISTRIBUTED VIEWS OF STUDENT DATA

a). Support System Network



b). Logical View

STUDENT ID	STUDENT NAME	SCHOOL
302500369	Umar, A.	Engg
300762728	Duke, P.	Engg
900540983	Harris, T.	Engg
789283830	Burke, F.	Law
376890323	Peter, H.	Law
547689000	Garw, Y.	Medical
653637208	Wilson, F.	Medical
376399783	Jenkin, C.	Arts
012389000	Mach, W.	Arts
834902208	Levy S.	Arts

c). Partitioned Views(partitioning by school)

STUDENT ID	STUDENT NAME	SCHOOL
302500369	Umar, A.	Engg
300762728	Duke, P.	Engg
900540983	Harris, T.	Engg

STUDENT ID	STUDENT NAME	SCHOOL
789283830	Burke, F.	Law
376890323	Peter, H.	Law

STUDENT ID	STUDENT NAME	SCHOOL
547689000	Garw, Y.	Medica
653637208	Wilson, F.	Medica

STUDENT ID	STUDENT NAME	SCHOOL
376399783	Jenkin, C.	Arts
012389000	Mach, W.	Arts
834902208	Levy, S.	Arts

d). Distributed (Allocated) View

Three nodes (campusses). Node 1 offers engineering, node 2 offers law, node 3 offers medicine courses. The arts courses are offered at all campusses.

NODE 1

STUDENT ID	STUDENT NAME	SCHOOL
302500369	Umar, A.	Engg
300762728	Duke, P.	Engg
900540983	Harris, T.	Engg

NODE 2

STUDENT ID	STUDENT NAME	SCHOOL
789283830	Burke, F.	Law
376890323	Peter, H.	Law

NODE 3

STUDENT ID	STUDENT NAME	SCHOOL
547689000	Garw, Y.	Medica
653637208	Wilson, F.	Medica

NEED TO BE ALLOCATED
(single node, two nodes, three nodes)

STUDENT ID	STUDENT NAME	SCHOOL
376399783	Jenkin, C.	Arts
012389000	Mach, W.	Arts
834902208	Levy, S.	Arts

languages which are not transferable to MVS environments, c) disk capacity limitations of the 8100 systems - a maximum of 100M, and d) centrally controlled communication systems. Table 7.5d shows the support system definition in terms of the DADP parameters.

STEP3D: Now we can run DADP in the optimization mode to allocate the DATAGRP and programs LIST and DROPADD for the support system configuration specified in step 3C and the availability, response time, policy and program duplication restrictions stated earlier. It should be noted here that we are mainly interested in the REGISTER transaction since this is an online transaction. Thus we can concentrate on determining the optimal allocation of the program (DROPADD) and dataset (DATAGRP) which are referenced by the REGISTER transaction. It should also be emphasized that the programs and datasets manually allocated in step 3B are included in this step by using "dummy" stages in the DADP allloca-tion algorithm in which the allocations are read directly as input from an external table. This is an important consideration because we must evaluate the cost, response time and availability of the entire system even if a very small portion of the programs and data are being allocated. For this example, the algorithm runs in the following manner:

Program allocation:

. Read the externally defined allocations to allocate the program REGISTER.

. Determine optimal allocation of CLIST and DROPADD.

Dataset allocation:

. Read the externally defined allocations to determine the allocation of DEPT (all partitions).

. Determine optimal allocation of DATAGRP.

STEP4A: Table 7.5c shows the results of running the DADP algorithm. There are two feasible solutions in FS2: 110 which shows that STUDENT' must be allocated to nodes 1 and 2, and 101 which shows that STUDENT' must be allocated to nodes 1 and 3. The allocations which did not in-

volve node 1 are infeasible because the arrival rate of REGISTER transaction at node 1, the central node, is very high. The program allocation allocates DROPADD to the central node due to the software duplication restrictions. The two allocations constitue FS2. We first pick the allocation from FS2 with minimum cost (110) and perform a rough local physical design of this allocation, coupled with the manual allocations determined above and evaluate this option in more detail. For example, the local implementation schema at node 1 consists of developing the implementation schema of the STUDENT, COURSE and DEPT databases for IMS, while the local implementation schema at the two 8100 nodes consists of the STUDENT and COURSE database. Let us assume that this schema leads to an infeasible solution when detailed queuing calculations are conducted which show that too many delays are being encountered due to synchronization. Although we can now study the other option in FS2, we note that this effort will be useless because the response time of 101 is greater than allocation 110. In this case, we need to go back to step3C to choose another synchronization algorithm (CCA), which produces the optimal solution of restricting DATAGRP to the central node (see table 7.5d). The rough physical design basically now consists of an IMS database design.

We can now conduct a detailed IMS physical database and program design for the central node. Let us assume that now this evaluation does not violate the constraints (note that the response time is now much less than 5 seconds). At this point an administrative acceptance of the distributed application system structure should be sought before a detailed local physical design is undertaken. If the application system is found unacceptable at this point because, maybe, the administration wants to show that distribution is better than centralization, then one of the following decisions can be made:

a). Declare the system infeasible and stop.

b). Go back to step3C to explore support system options with

TABLE 7.5: ILLUSTRATION OF DESIGN METHODOLOGY

a) APPLICATION SYSTEM DEFINITION

.No. of Datasets D_s=3, No. of Programs P_n=3, No. of Trans. T_n =2

.Size and occurrence of tuples $D_s(d,x)$, x=1 and 2, =

d1	80	50	d1 = DEPARTMENT'
d2	80	100	d2 = COURSE'
d3	80	2000	d3 = STUDENT'

.No. of reads $P_r(p,d)$, writes $P_w(p,d)$ and consistency updates $P_c(p,d)$

	d1	d2	d3
p1	0	0	25
p2	1	30	0
p3	0	0	10

	d1	d2	d3
p1	0	0	0
p2	0	0	0
p3	0	0	3

	d1	d2	d3
p1	0	0	0
p2	0	0	0
p2	0	0	0

Note: p1 = CLIST, p2 = DLIST, p3 = DROPADD

.Consistency Relationships $D_c(d,d')$ =

	d1	d2	d3
d1	0	0	0
d2	0	0	1
d3	0	0	0

.Interprogram messages $P_m(p1,p2)$ = 0 for all p1 and p2

.Program input $P_I(p)$ = 1 for all p, output $P_o(p)$ = 30

.Transaction arrival rate $T_a(k)$ = (5, 3, 2) for REGISTER

b) SUPPORT SYSTEM DEFINITION

.Number of nodes N_n = 3 Number of links = 2

.Availability $A_a(z)$ = 0.9 0.85

Topology $B_{tp}(i,j)$ =

i \ j	0	1	2
1	1	0	0
2	0	0	

Service rate of nodes E(x) = 1 3 3 → 1 = .1 second

Service rate of links E(z) = 5 5 → 5 = 1200 bps

Storage capacity STOR(x) = 1000 1 1

2PL Algorithm: U_{cp} = 1, U_{lm} = 1, U_{pg} = 2, U_{cr} = 1, U_{sl} = 0

Transaction processing: Q_{rt} = 1, Q_{da} = 1, COPY = 3

SRCH(k,x) =

k \ x	1	2	3
1	2	3	1
2	3	1	2
3	1	2	

Network definition : B_{nc} = 0, B_{rt}(x) = 1.0 0.0

c). FEASIBLE SET FOR 2PL:

Data Alloc	Stor. cost	Local Proc.	Remote Proc.	Total Proc.	Proc Delay	Queu Delay	Lock Delay	Total Delay	Avail	Prog. cost
110	1200	2500	7000	10700	1.2	2.8	0.5	4.5	0.92	25.0
101	1200	2500	7000	10700	1.3	2.8	0.6	4.7	0.92	25.0

d). FEASIBLE SET FOR CCA:

Data Alloc	Stor. cost	Local cost	Remote cost	Total cost	Proc Delay	Queu Delay	Lock Delay	Total Delay	Avail	Prog cost
100	600	1600	2750	4950	1.1	2.4	0.4	3.9	0.90	25.0

e). SENSITIVITY ANALYSIS:

INPUT						OUTPUT			
Node Speed			Com. Speed		Synch.	Optimal	Optimal Total		Total
N1	N2	N3	z1	z2	Algor.	Alloc.	Cost	Delay	Avail
1	3	3	3	3	2PL	110	10,700	3.5	0.92
1	3	3	3	3	CCA	100	4,950	3.1	0.90
1	2	3	2	2	2PL	110	10,700	2.4	0.92
1	2	3	2	2	CCA	100	4,950	2.2	0.90

faster links, disks and cpus. The 3033 as well as the 8100 systems can be speeded up by attached processors, microcode etc.

c). Go back to step3B relax and reconsider the restrictions and the effect of manual allocations.

d). Go back to step3A to reconsider the partitioning.

e). Go back to requirement analyis to relax restrictions.

Table 7.5e shows results of the sensitivity analysis when the communication speeds and the cpu speeds are varied. The table shows the optimal allocation and the cost, response time and availability for the optimal allocation for each configuration. Note that by increasing the communication speeds, the allocation 110 is optimal for several cases.

To summarize, we have shown through this informal example how an application system can be designed by using the design methodology described above. This example emphasizes the following important points: a) the large number of datasets and programs to be allocated are considerably reduced, in practice, by initial analysis of the problem and allowing manual and policy considerations, b) centralized allocation may be the best solution for some applications even if a distributed system exists,and c) hierarchical techniques which eliminate infeasible solutions at an early stage before detailed analysis are valuable design aids.

7.6: Other Practical Applications of the Algorithm

The design methodology discussed above shows how the algorithm can be used to design new application systems in DDP environments. In addition, this algorithm can be used in the following practical situations.

First, an existing centralized application system can be distributed, an existing distributed application system can be centralized or an existing distributed application system can be redistributed, by using this algorithm. The design methodology described above can be used

to accomplish this by defining the existing application systems in steps 3A and 3B, and defining the new support system environment (centralized or distributed) in step 3C.

Second, support system configurations can be chosen to best fit the application needs. Although, our model does not consider support system variables as decision choices, we can define proposed support systems in step 3C of the design methodology and evaluate the performance of optimal data and program allocations for each application system. This process can be very slow if a large number of support systems are to be evaluated for a large number of applications. In practice, this situation is handled by first narrowing the number of acceptable support system configurations based on compatability, cost, technical features, growth potential and other analysis and then evaluating the chosen support systems for the critical applications.

Third, this algorithm can be used to distribute the control programs like transaction processing modules, update synchronization programs and security checkin routines, and control databases like the data directories, network directories, and lock/stamp tables. This can be accomplished by viewing the support system as an application system which services the DDP requests. The transactions in this case represent requests like set/reset a lock, update directory, check authority etc, the programs are the control software modules and the datasets are the directories. This "application system" can be defined in step 3A. The support system definition in step 3C consist of defining those components of the support system that are already fixed.

Finally, the algorithm can also be used to analyze environments which include local area networks. In such systems, the main consideration is that all the costs must be included for analysis, because for local area networks, the communication cost is less important than the local processing cost, while for remote networks the cost considerations are usually reversed. Our model is applicable because we include

the local as well as communication costs. However, a major limitation of our algorithm, and most other allocation algorithms, is that networks with greater than 50 nodes are expensive to analyze. Systems consisting of local area networks can easily exceed 500 nodes, where most of the nodes are microcomputers. A possible solution to this problem is to decompose the overall network into subnetworks, where each subnetwork may be a local area network. Now the allocation can be first conducted so that each subnetwork is viewed as a node. After this, allocation at each subnetwork can be conducted. The design methodology can now be extended to decompose the distributed design phase into several subphases, where at each subphase one level of allocation is conducted.

Program and data allocation for large number of nodes need special attention. It appears that the only possible approach is decomposition into subnetworks mentioned above. After decomposition, two approaches are possible for optimization: a) consider each subnetwork as a local node, optimally allocate to each subnetwork, determine optimal allocation in each subnetwork, b) optimize in each subnetwork and then interconnect to form the overall network. The first approach has been used frequently in several cases <CHAND77a> and the second approach has been used to solve travelling salesman problems with more than 1000 cities <KARP77f>. We feel that the first approach is more suited to the program and data allocation problem.

7.7 Conclusions and Future Research Directions

The problem of allocating the programs and data of a generalized application system in a DDP environment is formulated and a solution algorithm is presented which employs a large number of techniques such as clustering, decomposition, hierarchical heuristics, spatial dynamic programming, distributed optimization, integer optimization and queuing theory. Analytical (worst-case, probabalistic) and empirical

(run-time, feedback) analysis techniques are used to evaluate the heuristics employed and bounds on approximations. The algorithm is implemented as a computer program, called DADP, which can be used to study the various tradeoffs in a predictive mode, automate program/data allocation and to design a distributed application system by using a design methodology. Our experience with DADP has demonstrated that the solution of realistically complex program/data allocation problem is computationally feasible.

Although there is an abundance of allocation schemes, this is the first attempt to analyze, formally represent and integrate the large number of technical as well as management factors which influence the allocation decisions. This research contributes to a quantitative understanding of the complex and practical tradeoffs involved in designing distributed application systems and leads to a methodology that can be used to design new applications, evaluate the designs of existing applications and measure the effect of support system variations on application system performance.

Further research can be pursued to eliminate the limitations of this research. First, several support system parameters can be added to allow more detailed description of distributed control subsystem features like security checking, data/query translation, deadlock detection and logging/recovery. Second, the predictive model which computes the local and remote service requests can be extended to include the effect of query/data translation, security checking, logging and directory placement. In addition, the response time calculations can be extended to include queuing disciplines like preemptive priorty scheduling and more extensive locking conflict analysis. Finally, the capability of DADP can be improved so that it can automatically read application system and support system information from a requirement definition database like PSL/PSA. This will enable DADP to be used as an integrated Decision Support System (DSS) in DDP system life cycles. The general area of

solving the allocation problems for large networks with greater than 50 nodes still needs further research. The decomposition technique presented above which decomposes the network into subnetworks needs special attention.

APPENDICES

APPENDIX A: DECOMPOSITION OF THE UNCONSTRAINED PROBLEM

THEOREM: The unconstrained problem (section 2.3) of determining $\alpha(d,i)$ and $\beta(p,i)$ which minimizes:

$$O = \Sigma_k\Sigma_i\Sigma_j T_a(t,k)(R_{np}(t,k,i,j)+R_{pp}(t,k,i,j)+R_{pd}(t,k,i,j))C_R(i,j)$$
$$+ \Sigma_k \Sigma_i T_a(t,k)\, L(t,k,i)\, C_L(i) + \Sigma_i S(i)\, C_S(i) \quad \ldots\ldots\ldots\ldots (A.1)$$

Subject to: $\Sigma_i \beta(p,i) \geq 1$ for all $p \in P$

$\Sigma_i \alpha(d,i) \geq 1$ for all $d \in D$

can be decomposed into the following two subproblems:

1). Determine the program allocations $\beta(p,i)$

To minimize:

$$O1=\Sigma_k\Sigma_i\Sigma_j T_a(t,k)(R_{np}(t,k,i,j)+R_{pp}(t,k,i,j))C_R(i,j) \quad \ldots\ldots\ldots (A.2)$$

. Subject to $\Sigma_i \beta(p,i) \geq 1$ for all $p \in P$.

2). Determine the data allocations $\alpha(d,i)$

. To minimize:

$$O2 = \Sigma_k \Sigma_i \Sigma_j T_a(t,k)\, R_{pd}(t,k,i,j)\, C_R(i,j)$$
$$+ \Sigma_k \Sigma_i T_a(t,k)\, L(t,k,i)\, C_L(i) + \Sigma_i S(i)\, C_S(i) \quad \ldots\ldots\ldots (A.3)$$

. Subject to:

a). $\beta(p,i)$ determined by the program allocation problem (1)

b). $\Sigma_i \alpha(d,i) \geq 1$ for all $d \in D$.

PROOF: The minimization problem is decomposable to the required subproblems if we can show that $O = O1 + O2$ where:

$$O1 = \Sigma_k \Sigma_i \Sigma_j T_a(t,k)\, (R_{np}(t,k,i,j)+R_{pp}(t,k,i,j))\, C_R(i,j)$$

$O2 =$

$$\Sigma_k\Sigma_i\Sigma_j T_a(t,k)R_{pd}(t,k,i,j)C_R(i,j)+\Sigma_k\Sigma_i T_a(t,k)L(t,k,i)C_L(i)+\Sigma_i S(i)C_S(i)$$

where R_{np}, R_{pp}, R_{pd} are given by equations (2.4a, 2.4b, 2.4c), L(t,k,i) is given by equation (2.3), S(i) is given by equation (2.2) and $C_R(i,j)$, $C_L(i)$ and $C_S(i)$ are resource utilization costs specified as input. It can be seen that the dependent variables S, L, R_{np}, R_{pp} and R_{pd} are functions of the decision variables α and β, the application system parameters shown in table 2.1 and the site selection matrices α_s and β_s. Let us represent all the application system parameters by a matrix AP, then:

$$S = f(\alpha, AP)$$

$$L = f(\alpha, \alpha_s, AP)$$

$$R_{np} = f(\beta_s, AP)$$

$$R_{pp} = f(\beta_s, AP)$$

$$R_{pd} = f(\alpha, \alpha_s, \beta_s, AP)$$

These expressions are based on the explicit assumptions stated in section 2.4. Assumptions 4 (program storage cost negligible) and 7 (program sites are selected before data sites) are of main importance for the decomposition. It was shown in section 2.5 that:

$$\beta_s(p,i)=\beta(p,i)\ \delta(t,k,i) \quad \text{for all p, k, i}$$

$$\alpha_s(k,p,d,i)=\alpha(d,j)\beta_s(k,p,i)\delta(i,j) \quad \text{for all p,d,i,j,k}$$

Thus α_s and β_s can be eliminated from equations 2.2, 2.3 an d 2.4. Let X represent the set of all the data allocation decisions and Y the set of all the program allocations and let X|Y denote X given Y. Then:

$$S = f(X, AP)$$

$$L = f(X|Y, AP)$$

$$R_{np} = f(Y, AP)$$

$$R_{pp} = f(Y, AP)$$

$$R_{pd} = f(X|Y, AP)$$

Combining and rearranging the objective function (A.1), we get:

$$O1 = \Sigma_k \Sigma_i \Sigma_j T_a(t,k)\ (R_{np}(t,k,i,j)+R_{pp}(t,k,i,j))C_R(i,j)$$

$$O2 = \Sigma_k\Sigma_i\Sigma_j T_a(t,k)\ R_{pd}(t,k,i,j)\ C_R(i,j) + \Sigma_k\Sigma_i T_a(t,k)L(t,k,i)C_L(i) + \Sigma_i S(i)C_S(i)$$

The constraint $\Sigma_i \beta(p,i) \geq 1$ is binding only on O1, and the constraint $\Sigma_i \alpha(d,i) \geq 1$ is binding on O2, thus we have decomposed the problem.

APPENDIX B: DECOMPOSITION OF DATA ALLOCATION (UNCONSTRAINED)

THEOREM: The problem of allocating D_g data groups, as stated in Appendix A (equation A.3), can be decomposed into D_g allocation independent subproblems as long as the consistency updates can be ignored.

PROOF: Since a summation function is decomposable, we intend to show that $O2 = \Sigma_d\ y(d)$, where O2 is the objective function shown in Appendix A and y(d) is the return due to allocation of data group d. The objective function O2, after substituting $D_c(d1,d2) = P_w(p,d) = 0$ and using equations 2.2, 2.3 and 2.4, can be written as:

$$O2 = \Sigma_d\ \Sigma_x\ D_s(d,x)\ \Sigma_i\ \alpha(d,i)C_S(i)$$
$$+ \quad \Sigma_k\Sigma_i\Sigma_p\Sigma_d P_b(p)T_a(t,k)T_n(p)C_L(i)(P_r(p,d)\alpha_s(k,p,d,i)+P_w(p,d)\alpha(d,i))$$
$$+ \quad \Sigma_k\Sigma_i\Sigma_j\Sigma_p\Sigma_d P_b(p)T_a(t,k)T_n(p)C_R(i,j)(P_r(p,d)\beta_s(k,p,i)\alpha_s(k,p,d,j)$$
$$+ \quad P_w(p,d)\ \beta_s(k,p,i)\ \alpha(d,j))$$

Since the summation for d is over all d, hence it can be factored out, leading to $O2 = \Sigma_d\ y(d)$, where:

$$y(d) = \Sigma_x\ D_s(d,x)\ \Sigma_i\ \alpha(d,i)\ C_S(i)$$
$$+ \Sigma_k\Sigma_i\Sigma_p\ P_b(p)T_a(t,k)T_n(p)C_S(i)(P_r(p,d)\alpha_s(k,p,d,i)+P_w(p,d)\alpha(d,i))$$
$$+ \Sigma_k\Sigma_i\Sigma_j\Sigma_p P_b(p)T_a(t,k)T_n(p)C_R(i,j)(P_r(p,d)\beta_s(k,p,i)\alpha_s(k,p,d,j)$$
$$+ P_w(p,d).\beta_s(k,p,i)\ \alpha(d,j)) \qquad \text{.....(B.1)}$$

APPENDIX C: LOCAL SEARCH ROUTINE

THEOREM: The minimization of y(d), the cost (return) given by equation (B.1), satisfies the cost access graph property that if a given vertex has a cost less than the cost of any vertex along the paths leading to it, then the sequence of costs encountered along any one of these paths decreases monotonically.

PROOF: Casey <CASE72> has shown that this property is satisfied by any cost function that can be represented as:

$$T = \Sigma_i X(i) + \Sigma_k Z(k) \Sigma_i q(k,i) + \Sigma_k Z'(k) \min_i(q'(k,i)) \quad \ldots\ldots(C.1)$$

where X(i) is the cost of storing data d at node i, Z(k) and Z'(k) are the update and query traffic respectively from node k to file d, and q(k,i) and q'(k,i) are the costs per update and query between nodes k and i. An examination of (B.1) shows that this equation can be transformed to (C.1) by using the following mapping:

$$X(i) = \Sigma_d \Sigma_x D_s(d,x)\ \alpha(d,i)\ C_S(i)$$

$$Z(k) = T_a(t,k)\ \Sigma_p T_n(p)\ P_b(p)\ \Sigma_d P_w(p,d)$$

$$q(k,i) = \Sigma_d\ \alpha(d,i) C_S(i) + \Sigma_j \Sigma_p \Sigma_d\ \beta_s(k,p,j)\ C_R(i,j)\ \alpha(d,i)$$

$$Z'(k) = T_a(t,k)\ \Sigma_p T_n(p)\ P_b(p)\ \Sigma_d P_r(p,d)$$

$$q'(k,i) = \Sigma_p \Sigma_d\ \alpha_s(k,p,d,i) C_L(i) + \Sigma_j \Sigma_p \Sigma_d\ \beta_s(k,p,j) C_R(i,j) \alpha_s(k,p,d,i)$$

Note that we have interchanged $\beta_s(k,p,i)\ \alpha(d,j)$ into $\beta_s(k,p,j)\ \alpha(d,i)$. This is based on the assumption that no loss of communication messages occurs, i.e. the traffic between nodes i and j is the same as between j and i. These expressions can be further simplified if needed. For example:

$\Sigma_p \Sigma_j\ \beta_s(k,p,j)\ C_R(i,j) = C_R(i,k)$ because

$\beta_s(k,p,j) = 1$ if $k=j$

Thus substituting k instead of j, we get:

$$\Sigma_p\ \beta_s(k,p,k)\ C_R(i,k) = C_R(i,k)$$

APPENDIX D: BACKGROUND INFORMATION

D.1: Analysis Of Transaction Processing Algorithms

The problems of query optimization and program/data allocation complement each other. In other words, query optimization assumes a program/data allocation while program/data allocations assume a query optimization algorithm. As stated in chapter 1, almost all of the available allocation algorithms assume a single program/single dataset transaction and minimum cost site selection strategy. We study program/data allocations subject to a variety of query optimization algorithms involving many program/many dataset transactions and generalized site selection. Secondly, the problem of determining optimal distributed transaction processing policy is NP-hard <HEVN81d>. Thus most availbale algorithms use heuristics and do not guarentee exact optimal return. It is also difficult to represent the various distributed transaction processing algorithms due to the following reasons:

1. The support system models vary. For example, some algorithms are based on specific support systems like SDD1, Distributed INGRES and POLYPHEME, while others, like Chu and Hurley <CHU79d>, are based on conceptual models. Different network topologies are also commonly assumed.
2. The cost functions vary widely. Some algorithms utilize arbitrary costs (SDD1), while others use communication costs and local processing costs in a variety of ways.
3. The objective function to be minimized varies. Some algorithms concentrate on minimizing response time, others on total time and still others use some combinations.
4. The role of the result site varies. In SDD1 and Distributed INGRES, the result site (the site where user receives the results) is insignificant, while Hevner and Yao allow a user specified result site (usually the transaction arrival node).

5. Some algorithms are restricted to uniquely assigned datasets, while others support duplicated data.
6. Several algorithms have been proposed, but only a few have been implemented. Very little actual experience with the implemented algorithms has been reported, making it difficult to identify the algorithms that need to be represented.

In addition, almost all of the existing transaction processing algorithms concentrate only on queries and ignore the update properties of transactions. Thus these algorithms may not be optimal for generalized transactions which involve data retrieval as well as modification.

D.2: Main Update Synchronization Algorithms

BASIC 2PL METHOD: This algorithm consists of the following steps: a) send lock 'set' requests to all the nodes, b) receive the lock grants, c) send the database read/writes, d) receive the read/write responses, e) update duplicate and consistency related data and f) send messages to all the nodes to release locks. This model assumes a distributed commit processor and lock manager and assumes no piggybacking.

BASIC TIME-STAMPED ALGORITHM: This algorith consists of the following steps: a) transaction arrives at node k, b) the time stamp TS is obtained for the transaction, c) the read/write request is concatenated with TS and is routed to the lock/timestamp manager, d) the lock/timestamp manager checks if TS < TD(d)where TD(d) is the largest timestamp of any read/write activity that has been processed against data object d; if yes, then wait or abort, else set TD to max(TS,TD) and return to k, e) node k now reads and updates the primary data, f) the consistency related data d' is "locked" by setting TD(d') high so that any transactions seeking to update d' are blocked, g) at synchronization time, the TD(d') are set to TS (this is similar to unlock). Most of the timestamp checking activity causes local service requests. This can be modelled by making the locks disk resident.

COMPLETE CENTRALIZED ALGORITHMS (CCAA & CCA): These are "primary copy" algorithms and have been discussed in detail by Garcia <GARC79d>. The major assumptions of these algorithms are: data is fully replicated, transactions are update only, and transmission delay between different nodes is constant. The main steps of the completely centralized algorithm with acknowledgement (CCAA) are: a) update transaction t arrives at node k, b) node k routes t to central node Z, c) t is executed at node z, d) the updated data is routed from central node z to all the nodes, e) after a node gets an update request, it performs local updates and sends acknowledgements back to the central node, f) when the central node receives all the acknowledgements, then it 'finishes' the transaction t by routing the results back to node k and then picks the next transaction for execution.

A simple variation of CCAA is obtained by eliminating step f if a sequence number is attached to the update message. This technique allows the central node to start executing the next transaction after step e and the algorithm is called CCA (Centrally Controlled Algorithm). To represent these algorithms, we choose the following parameters: lock manager and commit processor are centralized, all data and programs are selected at central node, and piggybacking and core residence of locks depends on CCAA versus CCA.

CENTRALIZED LOCKING ALGORITHMS (CLAA & CLA): In this case, the transactions are processed at the arriving node and the central node only keeps the lock tables. The centralized locking algorithm with acknowledgements (CLAA) consists of the following steps (see <GARC79d> for details): a) transaction t arrives at k, b) k sends "request locks" for all the data referenced by t to the central node z, c) the central node z grants locks to t, d) k processes t, e) k sends "update t" messages to all the nodes (commit processing), f) when k gets acknowledgements of "update t" from all the nodes, it sends "release" locks to z, g) central node releases the locks for t, now other transactions can get these

locks. To represent this algorithm, we choose centralized lock manager, distributed commit processing and piggybacking. Several variations of this algorithm exist. For example, step f can be eliminated by attaching a sequence number to the update message (CLA algorithm), step g is eliminated by "piggybacking" the unlock messages with updates and "wait lists" can be created to reduce the size of lock tables (see <GARC79d>).

ELLIS RING ALGORITHM: This algorithm views the database in three states: Idle (free to be updated), Active (being updated), Passive (to be processed). The main steps of this algorithm are: a) transaction t arrives at node k, b) t is routed to nodes in a daisy chain to acquire locks. If database is idle, then the database is locked and the request is passed on. If not, t waits for the database to be idle and then proceeds. c) when k receives t, then it means that all the locks have been acquired. Now t is sent again for actual updates. The algorithm, proposed by <ELLIS77d>, is very costly in terms of time spent and the message traffic generated because the transaction is circulated twice in the network and all the updates are performed serially even if they do not conflict. Also, the algorithm assumes a ring structure and is not robust in case of link failures. The lock manager and commit processor are distributed, each node is visited twice, and a serial routing is used.

THOMAS VOTING ALGORITHM: This algorithm is similar to the Ellis Ring algorithm with the main difference that the decision of applying the updates is not based on getting an 'OK' from all the nodes. Instead, if the majority of the nodes say OK, then the updates are applied, else the transaction restarts. This algorithm, proposed by <THOM79d>, can be represented by reducing the amount of piggybacking used in the Ellis Ring Algorithm. However, it is not necessary for this algorithm to follow a ring; instead the voting process can be conducted in a parallel manner.

OPTIMISTIC AND PESSIMISTIC ALGORITHMS: The optimistic algorithm works on the premise that conflicts do not occur often, a situation true for short queries. The transaction is first executed at the arrival node and then the update messages, plus timestamps, are sent to other nodes. If there is no conflict at any nodes, then the update is made permanent, else the transaction is aborted. The algorithm presented by Kung et al <KUNG79d> is an example. This type of algorithm can be represented by assuming that all lock/unlocks are completely piggybacked (eliminated). The pessimistic algorithm assumes that conflicts occur often. The transaction first reserves all the data items at all the sites and then starts executing. Once it is done, all the locks are freed. Milne <MILN80d> shows some examples. This type of algorithm can be represented by reconizing that all sites are locked.

SDD1 ALGORITHMS: SDD1 implements four algorithms P1, P2, P3, and P4. These algorithms are based on the premise that some classes of transactions do not need any synchronization and others need varying degree of synchronization. The transactions are subdivided into classes at transaction definition time and the transactions within the same class are synchronized by serializing the conflicting transactions in time stamp order; the transactions in different classes are synchronized by using the protocols P1, P2, P3 and P4 (see <BERN80d> for discussion of the protocols). The protocols basically consist of rules for resolving read-read, read-write and write-write conflicts and deadlocks. The basic algorithm followed by the protocols consists of the follwoing steps: a) transaction t arrives at k, b) t sends lock requests (stamps), c) if locks granted, based on the protocols, then proceed, else wait, d) t performs the local update, e) t routes the duplicate updates, f) t requests unlocks.

APPENDIX E: DECOMPOSITION OF GENERALIZED PROGRAM/DATA ALLOCATION PROBLEM

THEOREM: The allocation problem (section 6.1) of determining $\alpha(d,i)$ and $\beta(p,i)$ which minimizes:

$$OB=\Sigma_k\Sigma_i\Sigma_j T_a(t,k)(R_{np}(t,k,i,j)+R_{pp}(t,k,i,j)+R_{pd}(t,k,i,j)+R_{lk}(t,k,i,j))$$
$$\times\ \Sigma_z PATH(i,j,z)C_R(z)$$
$$+\ \Sigma_k\Sigma_i T_a(t,k)(L_{pd}(t,k,i)+L_{lk}(t,k,i))C_L(i)+\Sigma_i S(i)C_S(i) \quad .....(E.1)$$

Subject to:

a). Data Allocation Constraints:

$\Sigma_i\ \alpha(d,i) \geq 1$ for all $d \Xi D$

$\Sigma_d\ \alpha(d,i)\ D_s(d,1)D_s(d,2) \leq STOR(i)$ for all i

$T_d(d,i) - \alpha(d,i) \geq 0$ for all d & i

b). Program Allocation Constraints:

$\Sigma_i\ \beta(p,i) \geq 1$ for all $p \Xi P$

$T_p(p,i) - \beta(p,i) \geq 0$ for all p & i

$\Sigma_i\ \Sigma_p\ C_{ap}(p,i)\ \beta(p,i) \leq PCOST$

c). Other Constraints:

$\Sigma_k\ T_a(t,k)\ A(t,k)\ /\Sigma_k T_a(t,k) \geq AV$

$\Sigma_k\ T_a(t,k)\ RT(t,k)/\Sigma_k\ T_a(t,k) \leq DEL$

can be decomposed into the following two subproblems if the program sites are selected before data sites and if the program storage requirements can be ignored:

1).Determine the program allocations $\beta(p,i)$

.To minimize:

$$OB1=\Sigma_k\Sigma_i\Sigma_j T_a(t,k)(R_{np}(t,k,i,j)+R_{pp}(t,k,i,j))\Sigma_z PATH(i,j,z)C_R(z)..(E.2)$$

.Subject to:

$\Sigma_i\ \beta(p,i) \geq 1$ for all $p \Xi P$

$T_p(p,i) - \beta(p,i) \geq 0$ for all p & i

$\Sigma_p\ \Sigma_i\ C_{ap}(p,i)\ \beta(p,i) \leq PCOST$

$\Sigma_k\ T_a(t,k)\ A1(k,p)\ /\Sigma_i T_a(t,k) \geq AV$

$\Sigma_i\ T_a(t,k)\ RT1(t,k)/\Sigma_i\ T_a(t,k) \leq DEL$

2).Determine the data allocations $\alpha(d,i)$

.To minimize:

$$OB2 = \Sigma_k \Sigma_i \Sigma_j T_a(t,k)(R_{pd}(t,k,i,j)+R_{lk}(t,k,i,j))\Sigma_z PATH(i,j,z)C_R(z)$$
$$+ \Sigma_k \Sigma_i T_a(t,k)(L_{pd}(t,k,i)+L_{lk}(t,k,i))C_L(i)+\Sigma_i S(i)C_S(i) \quad ..(E.3)$$

.Subject to:

$\beta(p,i)$ determined by the program allocation

$\Sigma_i \alpha(d,i) \geq 1$ for all $d \Xi D$

$\Sigma_d \alpha(d,i)\, D_s(d,1)D_s(d,2) \leq STOR(i)$ for all i

$T_d(d,i) - \alpha(d,i) \geq 0$ for all d & i

$\Sigma_k T_a(t,k)\, A2(p,d) / \Sigma_k T_a(t,k) \geq AV$

$\Sigma_k T_a(t,k)\, RT2(t,k) / \Sigma_k T_a(t,k) \leq DEL$

PROOF: The minimization problem is decomposable to the required sub-problems if we can show that OB = OB1 + OB2 and if the constraints are binding on either OB1 or OB2, where:

$$OB1 = \Sigma_k \Sigma_i \Sigma_j T_a(t,k)\,(R_{np}(t,k,i,j)+R_{pp}(t,k,i,j))\,\Sigma_z\, PATH(i,j,z)C_R(z)$$

$$OB2 = \Sigma_k \Sigma_i \Sigma_j T_a(t,k)(R_{pd}(t,k,i,j)+R_{lk}(t,k,i,j))\,\Sigma_z\, PATH(i,j,z)C_R(z)$$
$$+ \Sigma_k \Sigma_i T_a(t,k)(L_{pd}(t,k,i)+L_{lk}(t,k,i))C_L(i)+ \Sigma_i S(i)C_S(i)$$

where R_{np}, R_{pp}, R_{pd} and R_{lk} are given by equations 4.32a to 4.32e, $L_{pd}(t,k,i)$ and $L_{lk}(t,k,i)$ are given by equations 4.31a and 4.31b, $S(i)$ is given by equation (4.29) and $C_R(i,j)$, $C_L(i)$ and $C_S(i)$ are resource utilization costs specified as input. Let us first concentrate on decomposing the objective function. It can be seen that the dependent variables S, L_{pd}, L_{lk}, R_{np}, R_{pp}, R_{lk} and R_{pd} are functions of the decision variables α and β, the application system parameters shown in table 6.1, the support system parameters shown in table 6.2, and the site selction matrices α_s and β_s. Let AP and SYS represent the set of all the application system parameters and all the support system parameters, respectively. Then we can represent main dependent variables as follows:

$$S(i) = f(\alpha,AP) \quad .....(E.5)$$

$$L_{pd}(t,k,i) = f(\alpha,\alpha_s,AP,SY) \quad .....(E.6)$$

$$L_{lk}(t,k,i) = f(\alpha,\alpha_s,AP,SY) \quad \text{.....(E.6)}$$

$$R_{np}(t,k,i,j) = f(\beta_s,AP,SY) \quad \text{.....(E.7)}$$

$$R_{pp}(t,k,i,j) = f(\beta_s,AP,SY) \quad \text{.....(E.8)}$$

$$R_{pd}(t,k,i,j) = f(\alpha,\alpha_s,\beta_s,AP,SY) \quad \text{.....(E.9)}$$

$$R_{lk}(t,k,i,j) = f(\alpha,\alpha_s,\beta_s,AP,SY) \quad \text{.....(E.10)}$$

It was shown in section 2.5 that:

$\beta_s(p,i)=\beta(p,i)\ \delta(t,k,i)$ for all p, k, i

$\alpha_s(k,p,d,i)=\alpha(d,j)\beta_s(k,p,i)\delta(i,j)$ for all p,d,i,j,k

Thus α_s and β_s can be eliminated from equations E.5 through E.10. Let X represent the set of all the data allocation decisions and Y the set of all the program allocations and let X|Y denote X given Y. Then:

$S(i) = f(X,AP)$

$L_{pd}(t,k,i) = f(X|Y,AP,SY)$

$L_{lk}(t,k,i) = f(X|Y,AP,SY)$

$R_{np}(t,k,i,j) = f(Y,AP,SY)$

$R_{pp}(t,k,i,j) = f(Y,AP,SY)$

$R_{pd}(t,k,i,j) = f(X|Y,AP,SY)$

$R_{lk}(t,k,i,j) = f(X|Y,AP,SY)$

This can be achieved only if the two assumptions of program site selection prior to data site selection, and the negligible program storage requirements, are valid. These equations show that R_{np} and R_{pp} are dependent on program allocations and S, L_{pd}, L_{lk}, R_{lk} and R_{pd} are dependent on the data allocation. Combining and rearranging the objective function (E.1), we get:

$OB = OB1+OB2$

$$OB1 = \Sigma_k\ \Sigma_i\ \Sigma_j\ T_a(t,k)\ (R_{np}(t,k,i,j)+R_{pp}(t,k,i,j))\ \Sigma_z\ PATH(i,j,z)C_R(z)$$

$$OB2 = \Sigma_k\ \Sigma_i\ \Sigma_j\ T_a(t,k)\ (R_{pd}(t,k,i,j)+R_{lk}(t,k,i,j))\ \Sigma_z PATH(i,j,z)C_R(z)$$
$$+ \Sigma_k\ \Sigma_i\ T_a(t,k)(L_{pd}(t,k,i)+L_{lk}(t,k,i))C_L(i) + \Sigma_i\ S(i)\ C_S(i)$$

Now let us analyze the constraints. First of all, the storage and data allocation constraints are binding on OB2 and the program allocation constraints are binding on OB1. So we do not discuss these constraints

any further. However, the response time and availability constraints need special attention. The two dependent variables used in these two constraints are:

$$RT(t,k) = f(\alpha,\alpha_s,\beta_s,AP,SY) \quad \text{.....(E.11)}$$

$$A(t,k) = f(\alpha,\alpha_s,\beta_s,AP,SY) \quad \text{.....(E.12)}$$

We can subdivide the transaction response time into RT1 and RT2:

$$RT(t,k) = RT1(t,k) + RT2(t,k)$$

where:

RT1(t,k) = transaction response time due to node to program and program to program traffic

$= f(\beta_s, AP, SY)$

RT2(t,k) = trans. response time due to program to dataset traffic

$= f(\alpha_s, AP, SY)$

By using the variables X and Y, we get:

$$RT1(t,k) = f(Y, AP, SY) \quad \text{and} \quad RT2(t,k) = f(X|Y, AP, SY)$$

Thus RT1 is binding on OB1 and RT2 is binding on OB2. We can similarly decompse the transaction avaialbility A(t,k) into A1 and A2, where:

$$A(t,k) = \Pi\, A1(k,p)\, A2(p,d)$$

A1(k,p) = availability of programs from node k

$= f(Y, AP, SY)$

A2(p,d) = availability of dataset d from program p

$= f(X|Y, AP, SY)$

Thus A1(k,p) is binding on OB1 and A2(t,k) is binding on OB2. Thus the generalized program/data allocation problem can be decomposed into a program allocation and a data allocation subproblem.

APPENDIX F: DECOMPOSITION OF GENERALIZED DATA ALLOCATION

THEOREM: The objective function of the data allocation problem, extended to include the update synchronization, transaction processing, and communication systems can be decomposed into independent data allocation subproblems if the consistency constraints are ignored.

PROOF: Let OB2 represent the objective function of the extended problem. OB2 is a nonlinear 0-1 function and is given by (see equation 6.1):

$$OB2=\Sigma_k\Sigma_i\Sigma_j T_a(t,k)(R_{pd}(t,k,i,j)+R_{lk}(t,k,i,j))\Sigma_z PATH(i,j,z)C_R(z)$$
$$+ \Sigma_k \Sigma_i T_a(t,k) (L_{pd}(t,k,i)+L_{lk}(t,k,i))C_L(i) + \Sigma_i S(i)C_S(i)$$

Since a summation function is decomposable, we intend to show that the storage costs (SC), local processing costs (LC) and remote processing costs (RC) components of OB are separable by showing that these dependent variables can be individually represented as sums of returns due to allocation of data group d. In other words, we intend to show that:

$$OB2 = \Sigma_d [y_S(d) + y_L(d) + y_R(d)]$$

where:

$y_S(d)$ = return due to storage cost of dataset group d

$y_L(d)$ = return due to local processing of dataset group d

$y_R(d)$ = return due to remote processing of dataset group d

The storage cost SC is given by:

$$SC = \Sigma_i S(i) C_s(i)$$

The expression for S(i) is given by:

$$S(i) = \Sigma_d \Sigma_x \alpha(d,i) D_s(d,x)$$

where D_s is the data size and α is the data allocation decision variable. This equation is obviously separable, since we can write:

$$SC = \Sigma_d y_S(d)$$

where $y_S(d) = \Sigma_i \Sigma_x \alpha(d,i) D_s(d,x) C_S(i)$

Now turning to the local cost LC, we know that:

$$LC = \Sigma_k \Sigma_i T_a(t,k) [L_{pd}(t,k,i) + L_{lk}(t,k,i)]C_L(i)$$

The expressions for L_{pd} and L_{lk} are given by (see equations 4.30 and 4.31):

$L_{pd}(t,k,i)$ = local service requests issued at node i due to arrival of a tranasction at node k for primary and secondary read and writes:

$$= \Sigma_p \Sigma_d (P_r(p,d)\alpha_s(k,p,d,i)+P_w(p,d)\alpha(d,i) + P_c(p,d)\Sigma_{d1} D_c(d,d1)\alpha(d1,i))$$

$L_{lk}(t,k,i)$ = local service requests, due to locking/unlocking, executed at node i due to arrival of a transaction at node i:

$$= 2U_{st} U_{lm}\Sigma_p \Sigma_d\{P_r(p,d)\alpha_s(k,p,d,i) + P_w(p,d)\alpha(d,i) + P_c(p,d)\Sigma_{d1}D_c(d,d1)\alpha(d1,i)\}$$
$$+(1-U_{lm})\delta(i,z)\Sigma_p \Sigma_d\{P_r(p,d)+P_w(p,d)+P_c(p,d)\Sigma_{d1}D_c(d,d1)\}$$

L_{pd} is decomposable if the weak-consistency approximation is used, i.e. if $P_c(p,d) \times D_c(d,d1) = 0$. In order to show the separability of L_{lk}, we first apply the weak consistency approximation which simplifies L_{lk} to:

$$= 2U_{st} U_{lm}\Sigma_p \Sigma_d\{P_r(p,d)\alpha_s(k,p,d,i) + P_w(p,d)\alpha(d,i)\}$$
$$+ (1-U_{lm})\delta(i,z)\Sigma_p \Sigma_d\{P_r(p,d) + P_w(p,d) \}$$

which is also separable. Thus LC is separable and can be written as:

$$LC = \Sigma_d y_L(d)$$

where:

$$y_L(d) = \Sigma_k \Sigma_i \Sigma_p T_a(t,k)[P_r(p,d)\alpha_s(k,p,d,i)+P_w(p,d)\alpha(d,i) + P_c(p,d)\Sigma_{d1} D_c(d,d1)\alpha(d1,i) + 2U_{st} U_{lm} \{P_r(p,d)\alpha_s(k,p,d,i) + P_w(p,d)\alpha(d,i)\} + (1-U_{lm})\delta(i,z)\{P_r(p,d)+P_w(p,d)\}]C_L(t,k,i)$$

Let us now consider the remote processing cost component RC given by:

$$RC = \Sigma_k\Sigma_i\Sigma_j T_a(t,k)[R_{pd}(t,k,i,j)+R_{lk}(t,k,i,j)]\Sigma_z PATH(i,j,z)C_R(z)$$

The expressions for R_{pd} and R_{lk} are given by:

$R_{pd}(t,k,i,j)$ = program to data traffic due to the read/update of remotely located data by programs. Note that the update traffic modifies the actual data plus the

consistency related data. This traffic is obtained mainly from the primary and secondary processing steps and are dependent on the location of the commit-processor:

$$R_{pd}(t,k,i,j) = U_{cm}\, R_{pd}'(t,k,i,j) + (1-U_{cm})\, R_{pd}''(t,k,i,j)$$

where $R_{pd}'(t,k,i,j)$ is given by:

$$= \Sigma_p \Sigma_d P_r(p,d)\beta_s(k,p,i)\alpha_s(k,p,d,j) + \Sigma_p \Sigma_d P_w(p,d)\, \beta_s(k,p,i)\, \alpha(d,j) + \Sigma_p \Sigma_d P_c(p,d)\beta_s(k,p,i)\Sigma_{d1} D_c(d,d1)\alpha(d1,j)$$

and $R_{pd}''(t,k,i,j)$ is given by:

$$= \Sigma_p \Sigma_d [P_w(p,d)+P_c(p,d)\Sigma_i D_c(d,d1)]\, [\beta_s(k,p,i)\delta(z,j)] + \delta(z,i)[P_w(p,d)\alpha(d,j) + P_c(p,d)\Sigma_i D_c(d,d1)\alpha(d1,j)]$$

$R_{lk}(t,k,i,j)$ = lock/unlock remote traffic due to the arrival of a transaction at node k. This traffic is obtained by adding the RSR in the open and close processing steps:

$$= 2U_{pg}\beta_s(k,p,i)U_{lm}\Sigma_p\Sigma_d\{P_r(p,d)\alpha_s(k,p,d,j)+P_w(p,d)\alpha(d,j) + P_c(p,d)\Sigma_{d1}D_c(d,d1)\alpha(d1,j)\} + (1-U_{lm})\delta(j,z)\Sigma_p\Sigma_d\{P_r(p,d)+P_w(p,d)+P_c(p,d)\Sigma_i D_c(d,d1)\}$$

The expression for R_{pd} is separable if weak-consistency is applied (R_{pd}' and R_{pd}'' are both separable). Similarly, R_{lk} is separable if weak-consistency is applied. Thus, RC is separable and can be written as:

$$RC = \Sigma_d\, y_R(d)$$

The expression for $y_R(d)$ is relatively long and is not given here. It can be easily obtained by adding R_{pd} and R_{lk}. Thus the extended allocation objective function OB2 is separable.

APPENDIX G: DECOMPOSITION FOR LOCAL DATA ACCESS MODE

THEOREM: The problem of allocating P_g program groups and D_g dataset groups, when restricted to local data access (LDA) can be decomposed into D_g data allocation subproblems provided: a) the program to program traffic is ignored and b) the consistency relationships are ignored.

PROOF: Let OB represent the objective function of the extended problem to include the effect of transaction processing, update synchronization and communication system. OB is a nonlinear 0-1 function and is given by equation 6.1, chapter 6:

$$\Sigma_k\Sigma_i\Sigma_j T_a(t,k)(R_{np}(t,k,i,j)+R_{pp}(t,k,i,j)+R_{pd}(t,k,i,j)+R_{lk}(p,d))$$
$$\times \quad \Sigma_z PATH(i,j,z)C_R(z)+\Sigma_k\Sigma_i T_a(t,k)(L_{pd}(t,k,i)+L_{lk}(t,k,i))C_L(i)+$$
$$\Sigma_i S(i)C_S(i)$$

An examination of this equation shows that this objective function is similar to the objective function OB2 analyzed in appendix F. The difference is due to the R_{np} and R_{pp} terms. Thus we analyze these two terms because it has been already shown in appendix F that all other terms are decomposable into data allocations if the consistency relationships are ignored. The expressions for R_{np} and R_{pp} are given by:

$R_{np}(t,k,i,j)$ = node to program remote services exchanged due to reading of input data, from nodes, and sending of display messages, to nodes, by programs.

$$= \delta(t,k,i)\ \Sigma_p\ [\{1+P_o(p)\}Q_{rt}\beta_s(k,p,j)$$
$$+\{1-Q_{rt}\}\{\beta_s(k,p0,j)+P_o(p)\beta_s(k,pn,j)\}]$$

$R_{pp}(t,k,i,j)$ = program to program remote services exchanged to transfer temporary data between programs at different sites:

$$= \Sigma_p\ \Sigma_{p1}\ P_m(p,p1)\beta_s(k,p,i)\beta_s(k,p1,j)$$
$$\times\ [Q_{rt}+\{1-Q_{rt}\}\{\Sigma_{p2}P_b(p2)+\Sigma_{p2}P_o(p2)\}]$$

The weak-interprogram approximation implies that $P_m(p,p1) = 0$ for all p and p1 Ξ P. This approximation yields $R_{pp} = 0$. Thus we need to deter-

mine if $R_{np}(t,k,i,j)$, given by equation G.2, is decomposable because if R_{np} is decomposable, then we have proved the theorem. Since we are assuming local data access, the program sites selected $\beta_s(k,p,j)$ for program p is the same as the site selected for all the datasets accessed by program p. However, the grouping scheme for LDA clusters all the datasets accessed by a program into a single dataset group. Thus:

$$\beta_s(k,p,j) = \alpha_s(k,p,d,j)$$

where p and d represent dataset groups. Now, we can substitute this value of β_s in equation G.2. Note that now equation G.2 can be rewritten as:

$$R_{np}(t,k,i,j) = \Sigma_p\, \delta(t,k,i)[\{1+P_o(p)\}Q_{rt}\beta_s(k,p,j) + \{1-Q_{rt}\}\{\alpha_s(k,p0,d,j)+P_o(p)\alpha_s(k,pn,d,j)\}]$$

which shows that $R_{np}(t,k,i,j)$ is separable in terms of dataset allocations d. Thus we have shown that OB is decomposable into individual dataset allocations and can be represented as the sum of returns $y_S(d)$, $y_L(d)$ and $y_R(d)$ shown in appendix F.

APPENDIX H: WORST-CASE PERFORMANCE OF DADP

THEOREM: The worst-case performance of the generalized program/data allocation algorithm DADP is given by:

$$Z_{DADP} - Z \leq MAX\{\varepsilon_g, \varepsilon_t\} \quad \ldots\ldots\ldots\ldots\ldots (H.1)$$

where:

Z_{DADP} = optimal cost obtained by running DADP

Z = the actual optimal cost

ε_g = worst-case deviation due to grouping approximation (see equation 3.1, chapter 3, for expression).

ε_t = total worst-case deviation due to the data allocation scheme and is given by:

$$\varepsilon_t = \varepsilon_c + \varepsilon_p + \varepsilon_w \quad \ldots\ldots\ldots\ldots\ldots (H.2)$$

where:

ε_c = worst-case deviation due to weak-consistency approximation given by equation 3.20, chapter 3.

ε_p = worst-case deviation due to weak-interprogram approximation given by equation H.10 for LDA and is zero for RDA.

ε_w = worst-case deviation due to workload & availability approximation

= SD(f) + LD(f) + RD(f)

where SD(f), LD(f) and RD(f) are the storage cost, local cost and remote processing cost difference due to the final dataset f. The expressions for SD(f), LD(f) and RD(f) are given by equations (H.12) through (H.14).

PROOF: In order to prove this theorem, we need to: i) show that ε_g is the same as shown in equation 3.20, ii) derive the analytical expressions for ε_c, ε_p and ε_w, iii) derive equation H.2 and iv) prove equation H.1.

i) Derivation of ε_g: It was shown in chapter 3, section 3.1.2, that the worst-case deviation of the unconstrained grouping scheme occurs

when all the datasets are clustered in a single dataset group. The resulting expression for ε_g was derived and is given by equation 3.20. (In equation 3.20, ε_g is shown as ε"*.) Due to the transaction processing extension, the grouping scheme is modified to cluster the datasets which are accessed by the same program for LDA. However, this modification does not affect the worst-case error introduced due to grouping since the equations 3.29 through 3.31 are unaffected. Thus equation 3.20 still represents the worst-case error ε_g in case of the extended program/data allocation problem. (Q.E.D)

ii) Derivation of ε_c, ε_p and ε_w: The worst-case deviation due to weak-consistency approximation, ε_c, was derived in section 3.1.1 and is given by equation 3.2. This indicator is unaffected by the transaction processing extension because consistency relationships and updates are not affected by transaction processing options.

In order to derive the analytical expression for ε_p, the worst-case deviation due to the weak-interprogram approximation, we need to consider the objective function OB as sum of three variables:

$$OB = \Sigma_d \, (\lambda c(d) + \lambda p(d) + \lambda(d))$$

where $\lambda c(d)$ is the cost due to consistency updates generated whenever dataset d is updated, $\lambda p(d)$ is the cost due to interprogram traffic for dataset d and $\lambda(d)$ is the balance of the storage, local and remote processing cost. Let us analyze these three variables in more detail. The expression for $\lambda(d)$ was derived in section 3.1.2, equation 3.2 and $\lambda p(d)$ is given by:

$$\lambda p(d) = \Sigma_k \, \Sigma_i \, \Sigma_j \, R_{pp}(t,k,i,j) \, C_R(i,j)$$

where $R_{pp}(t,k,i,j)$ shows the program-to-program traffic given by equation 4.32b and $C_R(i,j)$ is the cost per remote service request exchanged betweeen nodes i and j. An examination of equation 4.38b shows that R_{pp} does not include any explicit reference to the datasets. This situation holds in case of RDA, however in case of LDA, the datasets and the accessing programs must be allocated to the same node, i.e $\beta_s(k,p,i)$

$= \alpha_s(k,p,d,i)$. Also in case of RDA, the weak-interprogram approximation is not needed, and thus $\varepsilon p = 0$ for RDA. For LDA, the weak interprogram approximation is utilized and we need to calculate εp for such cases. We can substitute $\alpha_s(k,p,d,i)$ instead of $\beta_s(k,p,i)$ in equation 4.32b for all $p \in P_d$, where P_d represents the set of programs which access dataset d. Then:

$$\lambda p(d) = \Sigma_k \Sigma_i \Sigma_j \Sigma_p \Sigma_{p1} \Sigma_{d1} \beta_s(k,p,i)\alpha_s(k,p,d1,j) \; [Q_{rt} \; P_m(p,p1) + \{1-Q_{rt}\}\{\Sigma_{p2}P_b(p2)+\Sigma_{p2}P_o(p2)\}]C_R(i,j) \quad \ldots\ldots\ldots(H.3)$$

where $p \in P_d$ and $p1 \in P_{d1}$. Note that $OB = \Sigma_d \lambda(d)$ when $\lambda c(d) = 0$ and $\lambda p(d) = 0$ for all $d \in D$. In such cases, OB can be decomposed (see the proof in appendix B):

$$\text{Min } OB = y^*(1) + y^*(2) + \ldots + y^*(D_n) \quad \ldots\ldots\ldots(H.4)$$

where $y^*(d)$ denotes the optimal return due to allocation of d and there are D_n datasets in the system. Let us now consider the situation when $\lambda c(d) = 0$ but $\lambda p(d) \neq 0$ for some $d \in D$. Then, since λp introduces additional cost:

$$\text{Min } OB = y^*(1) + y^*(2) + \ldots + y^*(D_n) + \Theta \quad \ldots\ldots\ldots(H.5)$$

where $\Theta > 0$. Consider the hierarchical allocation algorithm DADP which allocates the datasets d1, d2,,,D_n and programs p1, p2,,P_n. We need to determine Θ. Since DADP is a hierarchical heuristic which determines optimal allocation at each stage, we get the following expression by using the arguments presented in section 3.1.1:

$$\text{Min } OB = y^*(1) + y^*(2) + \ldots + y^*(D_n) + \Sigma_i \lambda p(d) \quad \ldots\ldots\ldots(H.6)$$

This equation suggests the upper bound on Z_{DADP}, the minimum value of OB, generated by ALDA:

$$Z_{DADP} \leq y^*(1) + y^*(2) + \ldots\ldots + y^*(D_n) + \max \Sigma_d \lambda p(d) \quad \ldots\ldots\ldots(H.7)$$

where λp is given by equation H.3. The lower bound of Z, the axact optimal return is also given by equation H.6:

$$Z \geq y^*(1) + y^*(2) + \ldots\ldots + y^*(D_n) + \min \Sigma_d \lambda p(d) \quad \ldots\ldots\ldots(H.8)$$

Using equations H.7 and H.8, we get the worst case performance of Z_{DADP}:

$$Z_{DADP} - Z \leq \max \Sigma_d \lambda p(d) - \min \Sigma_d \lambda p(d) \quad \ldots\ldots\ldots(H.9)$$

Now, λp is minimum when $\alpha_s(k,p,d,i) = \alpha_s(k,p,d1,j)$ if i=j and is maximum when $\alpha_s(k,p,d,i) = \alpha_s(k,p,d1,j) = 1$ if $i \neq j$ for all $(d,d1) \in D$, $(i,j) \in N$ and $p \in P$. Simply stated, $\lambda p(d)$ is minimum when all the datasets are replicated (no interprogram remote traffic) and maximum when the datasets are uniquely assigned. Using these observations, we get the final expression for ε_p for LDA, the worst-case deviation due to weak-interprogram approximation, from equation H.3 and H.9:

$$\varepsilon_p = \Sigma_k \Sigma_i \Sigma_j \Sigma_p \Sigma_{p1} C_R(i,j) [Q_{rt} P_m(p,p1) + \{1-Q_{rt}\}\{\Sigma_{p2} P_b(p2) + \Sigma_{p2} P_o(p2)\}] \quad \text{.....(H.10)}$$

This completes the derivation of ε_p. To derive the expression for ε_w, the worst-case deviation due to workload approximation, we recall that the deviation due to grouping approximation complements ε_w because if all the datasets are clustered together, then the workload approximation is ineffective since no decomposition is needed. To compute ε_w, let us assume:

$$\varepsilon_g = \varepsilon_c = \varepsilon_p = 0$$

This assumption can be relaxed later without lack of generality. Let a* denote the optimal data allocation produced by DADP which yields the cost Z. If a* does not violate the constraints, then no error is introduced in optimal cost and $Z_{DADP} = Z$, i.e. no error is introduced due to the workload approximation. However, if a* violates a constraint that has been simplified, then $Z_{DADP} - Z = \varepsilon_w$. Let us review this situation in more detail.

DADP employs a hierarchical data allocation algorithm allocates dataset d1 in stage 1, dataset d2 at stage 2, etc. At stage n, the objective function is computed due to the allocation of datasets (dn,,,,d2,d1). The constraints applicable to datasets (dn,,,d2,d1) are also checked at stage n and eliminated from further consideration if infeasible. By using this method, the clearly infeasible constarints are eliminated early and the final constraint checking is performed in the final stage. Let us assume that dataset d_f is to be allocated at the fi-

nal stage f and let a* = {a*(f),a*(f-1),,,,,a*(1)} where a*(n) shows the optimal data allocated at stage n and a(n) itself, * indicates optimal, consists of a vector of dataset allocation decision variables α used in DADP:

$$a(n) = \{\alpha(dn,1), \alpha(dn,2),,,,\alpha(dn,NN)\}$$

where $\alpha(dn,i) = 1$ if dataset dn is allocated to node i, 0 otherwise and NN is the number of nodes. Then if a* satisfies all the constraints in stage f, then $\varepsilon_w = 0$. However, if a* violates any constraint, then new value for a(f), say a'(f), is needed for dataset d_f. Now if all the constraints are satisfied, then the error introduced is given by:

$$\varepsilon_w = Z_{DADP}(a') - Z_{DADP}(a^*)$$

However, if a' does violate a constraint, then new assignment for final dataset d_f is needed. In the worst-case, the allocation a" which replicates d_f needs to be evaluated, since data replication yiels highest cost for DADP. Thus the worst-case deviation due to workload approximation is given by:

$$Z_{DADP}(a'') - Z_{DADP}(a^*) \quad \text{.....(H.11)}$$

We need to derive the actual expression for (H.3). This is straightforward since the objective function is sum of storage, local processing and remote processing cost. By following the computations employed in deriving the worst-case deviation due to the grouping approximation (section 3.1.2, chapter 3), we derive the following worst-case deviation for ε_w:

$$\varepsilon_w = SD(f) + LD(f) + RD(f)$$

Where:

SD(f)= difference in storage cost for final dataset d_f.

$$= \Sigma_x D_s(f,x) \Sigma_i C_S(i) \quad \text{.....(H.12)}$$

LD(f)= difference in local processing cost for final dataset d_f

$$= L_w(f) \Sigma_i C_L(i) + \Sigma_i L_r(f) C'_L(i) \quad \text{.....(H.13)}$$

RD(f)= difference in the remote processing cost due to dataset d_f

$$= R_w(f) \Sigma_i \Sigma_j C_R(i,j) \Sigma_k \delta(t,k,i) + R_r(f) C'_R \quad \text{.....(H.14)}$$

Where $L_r(f)$ and $L_w(f)$ are the local read and write activities, respectively, for dataset d_f, $R_r(f)$ and $R_w(f)$ are the remote read and write activities , respectively, for dataset d_f and all other parameters are the application system parameters. C'_L and C'_R designate the highest resource utilization costs for local and remote processing, respectively, for a given DDP network. The expressions for $L_r(f)$, $L_w(f)$, $R_r(f)$, $R_w(f)$ are:

$$L_r(f) = \Sigma_k T_a(t,k) \Sigma_p T_e(t,p) \Sigma_d P_r(p,d) \quad \text{.....(H.15)}$$

$$L_w(f) = \Sigma_k T_a(t,k) \Sigma_p T_e(t,p) \Sigma_d P_w(p,d) \quad \text{.....(H.16)}$$

$$R_r(f) = \Sigma_k T_a(t,k) \Sigma_p T_e(t,p) \Sigma_d P_r(p,d) \quad \text{.....(H.17)}$$

$$R_w(f) = \Sigma_k T_a(t,k) \Sigma_p T_e(t,p) \Sigma_d P_w(p,d) \quad \text{.....(H.18)}$$

(END PROOF)

iii)Derivation of equation H.2: Let us first consider the situation when $\lambda c(d) \neq 0$. It was shown in chapter 3, section 3.1.1, that for $\lambda c(d) \neq 0$, the unconstrained data allocation scheme produces the following result (see equation 3.6):

$$\text{Min } O = y^*(1) + y^*(2) + \ldots + y^*(D_n) + \Sigma_d \lambda c(d) \quad \text{.....(H.19)}$$

where O is the objective function of the unconstrained data allocation scheme, $y^*(d)$ denotes the optimal return due to allocation of dataset d. This equation holds if $\lambda c(d) \neq 0$ and $\lambda p(d) = 0$. Now consider the situation when $\lambda c(d) \neq 0$ and also $\lambda p(d) \neq 0$, then following the derivation given in section 3.1.1, we get:

$$\text{Min OB} = y^*(1)+y^*(2)+ \ldots y^*(D_n) + \Sigma_d \lambda c(d) + \Sigma_d \lambda p(d) \quad \text{.....(H.20)}$$

This equation suggests the upper bound on Z_{DADP}, the minimum value of OB, generated by DADP:

$$Z_{DADP} \leq y^*(1)+y^*(2)+ \ldots +y^*(D_n)+\max\Sigma_d\lambda c(d)+\max\Sigma_d\lambda p(d) \quad \text{.....(H.21)}$$

where λc is given by equation 3.2 and λp is given by equation H.3. Equation H.20 also helps to determine the lower bound on Z:

$$Z \geq y^*(1) + y^*(2) + \ldots + y^*(D_n)+ \min \Sigma_d \lambda c(d) + \min \Sigma_d \lambda p(d) \quad \text{...(H.22)}$$

Using equations H.21 and H.22, we get the worst case deviation ε_t*:

$$\varepsilon_t = \max \Sigma_d \lambda c(d) - \min \Sigma_d \lambda c(d) + \max \Sigma_d \lambda p(d) - \min \Sigma_d \lambda p(d) \quad ..(H.23)$$

It was shown in section 3.1.1 (see equation 3.2) that λc is a monotonically increasing function of the consistency related data duplication - $\alpha(d1,i)$. So λc is minimum when consistency related data d1 is allocated to only one node and maximum when d1 is fully replicated. The minimum and maximum values of $\lambda p(d)$ were discussed above (see the derivation of ε_p*). Thus by substituting the maximum and minumum values of λc and λp in equation H.23 and simplifying, we get:

$$\varepsilon_t = \varepsilon_c + \varepsilon_p$$

Now let us consider the effect of ε_w, the error due to workload approximation. Since this error is introduced only when the response time or availability constraints are violated, hence the ε_w adds to ε_t only when constraints are violated. This addition is strictly linear because $\varepsilon_t = \varepsilon_w$ when $\varepsilon_c = \varepsilon_p = 0$. Thus, we get equation H.2:

$$\varepsilon_t = \varepsilon_c + \varepsilon_p + \varepsilon_w$$

iv). Proof of Equation H.1: It was shown in section 3.1.3 that the performance of the data allocation scheme is dependent on the performance of the grouping scheme because the grouping scheme clusters the related datasets and programs which in turn reduce the deviation ε_1*. Let us define ε'_t to designate the total worst-case deviation of data allocation, after grouping. Then the worst-case deviation of algorithm DADP which utilizes the grouping scheme and then the data allocation scheme is given by:

$$Z_{DADP} - Z \leq \varepsilon_g + \varepsilon'_t{}^*$$

Let us look at the following two extreme cases:

1. If all the datasets are grouped into a single group, then $\varepsilon'_t{}^* = 0$ since the error due to consistency as well as the interprogram traffic is eliminated.

2. If no groups are formed then $\varepsilon_g = 0$ and $\varepsilon'_t = \varepsilon_t$, where ε_t is given by equation H.2.

Based on these observations, the worst-case deviation of DADP is either ε_g or ε_t. Thus:

$Z_{DADP} - Z \leq MAX(\varepsilon_g , \varepsilon_t^*)$ .... (END PROOF)

APPENDIX I: LITERATURE SURVEY AND ANALYSIS HEMES

I.1. Analysis of Past Efforts

The past efforts are classified and analyzed as program/data allocation schemes, data allocation schemes, program allocation schemes and the data/communication allocation schemes.

Program and Data Allocation Problems

The problem stated in section 1.4 is a program/data allocation problem which was first addressed by Morgan & Levin <MORG77c, LEVIN75c>. The original problem formulated was:

Determine:

. Allocation of data (files) to nodes

. Allocation of programs to nodes

To minimize:

storage cost + communication cost

Subject to:

. storage restrictions imposed by the node storage capacities

. internode communication delay constraints for messages

. policy/security restrictions on data allocations

Morgan and Levin showed that this problem can be decomposed into single dataset allocation subproblems and used a hypercube search technique to solve individual subproblems. Their formulation, so far the most significant work in the area of program and data allocation since it explicitly considered the relationships between data and programs, cannot be used to solve the program and data allocation problem stated in section 1.2 due to the following limitations:

1. Realistic support system configurations cannot be represented. For example, the effects of update synchronization algorithms,

query processing models, data distribution strategies, network topologies and network control are not included. For example, they assume fully connected network, no update synchronization costs, direct program to dataset accesses and no restrictions on data distributions.

2. The user requirements are overly simplified. For example, the internode communication delay is only a portion of the total transaction turnaround time the main area of user concern. The model also ignores some important constraints like availability. Thus this model may yield an "optimal" solution which may not be feasible in a practical situation. To illustrate this point, consider a two node network where node n1 is very slow and unreliable and node n2 is fast and reliable. Nevertheless, their model will always allocate data accessed from n1 to n1, even though the response time and availability is better if the data is allocated to n2.

3. The objective function does not represent total cost since the local processing cost (cpu cost, I/O cost) is ignored. The local processing cost plays important role in several type of networks, especially the local area networks.

4. Complex situations like database relationships and interprogram interactions are ignored. Thus the application of this model is restricted to simple cases.

The research since Morgan & Levin has not addressed these limitations. For example, Buckles & Harding <BUCK79c> have solved an unconstrained problem which allocates P "resources" (programs or data files), to N nodes to minimize the communication + storage costs. This model is even simpler than Morgan & Levin, however the main strength of this work is the use of grouping in allocation problems. Fisher and Hochbaum <FISH80c> have reconsidered the unconstrained version of the problem

formulated by Morgan & Levin and have developed more efficient optimization techniques to solve the problem. More recently, Marcogliese & Novarese <MARC81c> have addressed the problem of program/data allocation as two separate problems. The data allocation problem attempts to assign data which would minimize storage and communication costs subject to the storage, access time and availability constraints. The program allocation problem assigns programs to minimize the transmission and wait times subject to the processor capacity constraints. The data allocation algorithm is very similar to existing data allocation schemes (for example <SURI79c>) however, the program allocation model is quite sophesticated since it includes homogeneous as well as heterogeneous systems with multiple copies of programs. The major problem with this work is that clear interrelationships between the data and program allocations, which show the datasets accessed by programs, are not established. In addition, they do not present a solution algorithm for the program allocation problem.

Data Allocation Algorithms

These schemes are primarily concerned with optimal allocation of data files to nodes and can be viewed as special cases of the program/data allocation problems if the program allocation decisions are ignored. A significant number of different data allocation schemes have been published. Chu <CHU69c> was the first to formulate the allocation of D files to N nodes as a zero-one mathematical programming problem. Casey <CASE72c> studied an unconstrained problem of allocating a single file to N nodes, highlighted significant properties and showed that the number of file copies can be determined in terms of query and update traffic to the file. Casey's work has led to several other research efforts: Eswaran <ESWA76c> has formally shown that Casey's problem is polynomial complete, Chandy & Hewes <CHAND76c> have solved the same problem as a linear programming problem and later on Suri <SURI79c> used Casey's results for a constrained data allocation problem (see table

1.1). Segal <SEGA79> has considered a dynamic file allocation problem and Tabatabi & Arizoulla <TABA78c> formulate a unique file distribution algorithm which minimizes transaction response time. Loomis <LOOM76c> was the first to include the effect of data relationships on file allocations in a predictive model, and Coffman & Gelenbe <COFF80c> have studied data allocation for Ellis ring update synchronization algorithm. Chang <CHANG81c> has presented a model which includes data relationships and a simplified version of update synchronization costs. Chen <CHEN83c>, has developed a data allocation algorithm which minimizes the transaction response time for a variety of update synchronization algorithms. He gives a detailed calculation of the response time for different queuing disciplines like FCFS (first come first serve), preemtive priority scheduling and the like. However, he ignores transaction availability constraint and local processing costs.

More recent efforts in data allocation have been directed to discrete simulation models. For example, Barbara and Garcia-Molina <BARB82c> have studied the effect of update synchronization on data duplication and Matsushita <MATS82c> has reported results of detailed simulation for data duplication. The evaluation model presented by Bucci and Streeter <BUCC79c> allows a user to study tradeoffs between centralization versus distribution of data.

Thus the data allocation schemes also address only portions of the program and data allocation problem due to the following reasons: the local processing is ignored in most of the cases, the data relationships are only represented by <LOOM76c, CHANG81c>, complex transactions cannot be represented since most researchers describe transactions in terms of node to file read/update traffic which cannot correctly model transactions involving multiple operations (programs) on multiple files, and the support system and user performance requirements are overly simplified. For example total response time is considered only by Chen <CHEN83c>, file availability is used instead of the transaction

availability and transation processing and update synchronization options are not included in most schemes.

Program Allocation Algorithms

These algorithms attempt to determine optimal program (module) allocation to nodes (processors) and can be considered special cases of the program/data allocation problem if the data allocation decisions are ignored. Gylys <GYLY76> was the first to formulate a program allocation problem. Brya <BRYA81c> developed a predictive model which shows how modules can be allocated efficiently by using approximate queuing analysis. McEntyre <MCENT81c> and Doty <DOTY82c> have formulated this problem and have solved it using dynamic programming. These approaches tend to minimize the program processing time subject to the memory constraints and generally ignore a large number of considerations like update synchronization, query processing and network topologies. It should be also observed that the objective function for most program allocations is response time instead of the storage plus communication cost traditionally used in data allocations.

Data/Communication Algorithms

These algorithms fall into the category of "combined" application/support system problems (case 4 in section 1.1) and cannot be considered as special cases of the program/data allocation problem. However, we can transform these algorithms into the general problem since a comprehensive network description (topology, control, capacity, speed) is included in this problem. Whitney <WHIT70c> was the first to formulate the problem of allocating files, channel capacity and design of a tree network topology. Casey <CASE73c> also considered the problem of tree topology design. Mahmood and Riordon <MAHM76c> considered the combined file and channel capacity assignment subject to file availability and delay constraints for given network topologies. Khabbaz and Irani <KHABB80c, IRANI81c> have formulated and solved the combined file

and network topology design problem subject to availability and delay constraints. Chang <CHANG77c> has given a comprehensive design problem which assigns processors to nodes, links to processors, transactions to nodes and routing between processors. His algorithm, however only computes feasible solution. As indicated in table 1.1, this class of algorithms also ignore a large number of factors included in the general problem. Dinkel et al <DINK78c> developed allocation algorithms for resources between regional computing centers. Problems of this nature are used primarily for the overall planning of regional centers and are beyond the scope of this research.

1.2 Reduction/Transformation to other Resource Allocation Schemes

The generalized program/data allocation algorithm presented in chapter 7 can be reduced and/or transformed to the following currently available schemes.

Casey's Formulation <CASE72c>.

Casey addresses the following problem:

Given :

N = number of nodes

D = number of files

Y(k,d) = no. of queries directed from node k to file d

Y'(k,d) = no. of updates directed from node k to file d

C(t,k,i) = cost of one query between nodes i and j

C'(t,k,i) = cost of one update between nodes i and j

S(j) = cost of storing a file on node j

Choose the nodes I Ξ (n1,n2,n3,,,nN) where d should be located to minimize:

$$O(d) = \Sigma_k \ (\Sigma_j Y'(k,d)C(t,k,j) + Y(k,d).\min(C(t,k,j))) + \Sigma_j \ S(j)$$

We can make the following observations:

a). There are no constraints.

b). The objective function concentrates on communication and file storage costs, the local processing costs are ignored.

c). The only decision variable is the allocation of files to nodes.

It is our objective to reduce the problem formulated in section 7.1 to Casey's formulation. By ignoring the constraints, our problem reduces to:

DETERMINE: the 0,1 decision variables α and β where:

. $\alpha(d,i)$= 1 if dataset d is assigned to node i, 0 otherwise

. $\beta(p,i)$= 1 if program p is assigned to node i, 0 otherwise

TO MINIMIZE: the total cost

TC = storage cost + local processing cost + Remote Processing Cost

Where:

a.Storage cost= $\Sigma_d \Sigma_i \alpha(d,i) D_s(d,1)D_s(d,2)C_s(i)$

Where:

.$\alpha(d,i)$ = the data allocation decision variable

.$D_s(d,x)$ = size (x=1) and occurrence (x=2) of tuples in dataset d

.$C_s(i)$ = Cost of storing 1000 bits on node i

b.Local processing cost= $\Sigma_k \Sigma_i \lambda(t,k) L(t,k,i) C_l(i)$

Where:

.$\lambda(t,k)$ = arrival rate of a transaction at node k

.$L(t,k,i)$ = no. of local service requests (interactions between program and data) executed at node i due to the arrival of a transaction at node k. L is shown in table 7.2.

.$C_l(i)$ = cost of processing one local request at node i

c.Remote processing cost=$\Sigma_k \Sigma_i \Sigma_j \lambda(t,k) \Sigma_z R(t,k,i,j) PATH(i,j,z) C_r(z)$

Where:

.R(t,k,i,j) = no. of remote service requests (messages) exchanged between nodes i & j, when a transaction arrives at node k. R is shown in table 7.2.

.PATH(i,j,z) = the probability that a message between nodes i and j traverses link z.

.$C_r(z)$ = cost of exchanging one message on link z

Consider the following simple case:

a. $\beta(p,i)$ =1 for all p and i - programs assigned to every node.

b. $C_l(i)$= 0 - no local processing cost

c. Application system parameters:

.$D_c(d,d1)$ = 0 for all d and d1 - no consistency relationships

.$P_i(p) = P_o(p) = P_m(p,p1)$ = 0 for all p and p1 -one program per transaction and no program i/o.

d. Transaction processing options:

.SRCH(k,x) = based on minimum cost access

.Q_{da} = 1 - remote data access is allowed

.Q_{rt} = 1 - routing is parallel

e. Update synchronization algorithms

. U_{pg} = 0 - full piggybacking (optimistic algorithm)

. U_{cp} = 1 and U_{lm} = 1 - commit processor and lock manager are distributed

f. Communication system options:

. fully connected, fixed routing network, i.e. PATH(i,j,z) = 1 if z connects i and j, 0 otherwise.

Then $R(t,k,i,j) = R_{pd}(t,k,i,j)$ since:

$R_{np}(t,k,i,j) = R_{pp}(t,k,i,j) = R_{lk} = 0$ and

Now substituting these values in equations I.1, I.2 and I.3, we get:

total cost =

$$\Sigma_k \Sigma_i \Sigma_j \lambda(t,k) R_{pd}(t,k,i,j) \Sigma_z PATH(i,j,z) C_r(z)$$
$$+ \Sigma_d \Sigma_i \alpha(d,i)\, D_s(d,1)\, D_s(d,2)\, C_s(i)$$

In order to compute R_{pd}, we substitute the values of the above variables in equation 5.32d, and get:

$$R_{pd}(t,k,i,j)=$$

$$\Sigma_p\Sigma_d P_r(p,d)\beta_s(k,p,i)\alpha_s(k,p,d,j)+\Sigma_p\Sigma_d P_w(p,d)\beta_s(k,p,i)\alpha(d,j)\ldots(I.6)$$

Since the programs are replicated and the SRCH matrix indicates minimum cost site selection:

$\beta_s(k,p,i) = 1$ for k=i, 0 otherwise

$\alpha_s(k,p,d,j) = 1$ if j is the minimum cost node for d, 0 otherwise

Then substituting these values in equations I.6 andI.5, and after simplification, we get:

$$\text{total cost} = \Sigma_k\Sigma_i\lambda(t,k)\ \{\Sigma_i\Sigma_i P_w(p,d)\alpha(d,i)+\min(P_r(p,d))\}$$

$$\Sigma_z PATH(i,j,z)C_r(z) + \Sigma_d\ \Sigma_i\ \alpha(d,i)D_s(d,1)D_s(d,2)C_s(i)$$

Let : $S(j) = D_s(d,1)D_s(d,2)\ C_s(j)$

$C(t,k,j) = \Sigma_i\ \alpha(d,j).PATH(i,j,z).C_r(z)$

$C'(t,k,j) = C(t,k,j)$

$Y(k,d) = \Sigma_i\ \lambda(t,k)P_r(p,d)$

$Y'(k,d) = \Sigma_i\ \lambda(t,k)P_w(p,d)$

Now substituting in eq. (I.7), we get:

$$\text{total cost} = \Sigma_k\ (\ \Sigma_j\ Y'(k,d).C'(t,k,j) + Y(k,d).\text{Min}_i\ (C(t,k,j))) + \Sigma_j\ S(j)$$

which is Casey's formulation. Thus we have shown that the optimization problem stated in section 7.1 can be reduced to Casey's formulation by properly choosing the decision variables and other options.

Morgan & Levin <MORG77c, LEVIN75c>

The problem addressed by Morgan & Levin can be formulated as follows:

Given:

- N nodes
- D files, P programs
- L(k,p,d) = query traffic from node k to file d via progrm p
- L'(i,p,d) = update traffic from node i to file d via program p

. C(i,j) = communication cost per query from i to j.

. C'(i,j) = communication cost per update from i to j.

. X(j,k,d) = proportion of transactions arriving at node j to access file d that are routed to node k.

. X(i,j,p) = proportion of transations arriving at node i which access program p that are sent to node j.

. A = expansion factor for queries

. B = expansion factor for updates

Determine:

Y(k,d) = 1 if file d is assigned to node k, 0 otherwise

Y'(k,p) = 1 if program p is assigned to node k, 0 otherwise.

which minimizes :

$O = OA + OB + OC$

where:

OA = query processing cost

$$= \Sigma_i \Sigma_j \Sigma_p \Sigma_d L(i,p,d)X(i,j,p)(C(i,j)+A.Min(C(j,k).X(j,k,d))$$

OB = update processing cost

$$= \Sigma_i \Sigma_j \Sigma_p \Sigma_d L'(i,p,d).X(i,j,p).(C'(i,j) + B.C'(j,k).Y(k,d))$$

OC = storage cost

$$= \Sigma_k \Sigma_d S(k,d)$$

subject to the following constraints:

$T(i,j) + T(j,k) < T(d)$ .... access time constraint

Tij - node to program time

Tjk - program to file time

Td - maximum allowable delay to access file d

Policy constraints .. not specified formally

Note that the objective function is similar to Casey's formulation, the only difference is that program placement (node to program, program

to file) considerations are added. In order to reduce our formulation to this formulation, we make the following assumptions:

a. $\beta(p,i) = 1$ if program p is assigned to node i, 0 otherwise.

b. $C_l(i) = 0$ - no local processing cost

c. Application system parameters:

.$D_c(d,d1) = 0$ for all d and d1 - no consistency relationships

d. Transaction processing options:

.SRCH(k,x) = based on minimum cost access

.$Q_{da} = 1$ - remote data access is allowed

.$Q_{rt} = 1$ - routing is parallel

e. Update synchronization algorithms

. $U_{pg} = 0$ - full piggybacking (optimistic algorithm)

. $U_{cp} = 1$ and $U_{lm} = 1$ - commit processor and lock manager are distributed

f. Communication system options:

. fully connected, fixed routing network, i.e.

PATH(i,j,z) = 1 if z connects i and j, 0 otherwise.

After substituting these values in the objective function (eq 7.1) and performing the same type of manipulations as shown in Casey's case (section I.1.1), we get Morgan & Levin's objective function. The following expressions show the mapping between Morgan & Levin's variables and the variables used by us:

$$L(i,p,d) = \Sigma_k \lambda(t,k)(P_r(p,d) + P_I(p) + P_0(p))$$

$$L'(i,p,d) = \Sigma_k \lambda(t,k)(P_w(p,d) + P_I(p) + P_0(p))$$

$$C(i,j) = C'(i,j) = \Sigma_z PATH(i,j,z)C_r(z)$$

$$X(i,j,p) = \beta_s(t,k,i,p)$$

$$X(j,k,d) = \alpha_s(k,p,d,j)$$

$$Y(k,d) = \alpha(d,k)$$

$$Y'(k,p) = \beta(p,k)$$

A = B = MG - the expanson of communication requests (section 6.1.1).

The access time constraint is simply:

$DEL(t,k) \leq MDEL$

where DEL(t,k) is now calculated by ignoring:

. Parallelism

. Queuing

. Locking

The policy constraint is taken care of by the data and program policy matrices T_d and T_p defined in section 7.1 (equations 7.8 and 7.10). Thus, we have shown that the formulation of Morgan & Levin can be achieved from the formulation presented in section 7.1.

Mahmood & Riordon <MAHM76c>

The problem addressed by Mahmood & Riordon is:

Given:

. N nodes

. D files

. $Y_q(k,d)$ = query traffic from node k to file d

. $Y_u(k,d)$ = update traffic from node k to file d

. $Y'_q(k,d)$ = return traffic from node k as a result of $Y_q(k,d)$

. $Y'_u(k,d)$ = return traffic to node k as a result of $Y_u(k,d)$

. C(z,s) = cost of link x at speed s

. S(k,d) = cost of assigning dataset d to node k

Determine:

.$\alpha(k,d)$ = 1 if file d is assigned to node k, 0 otherwise.

.V(z,s) = 1 if link z is assigned speed s, 0 otherwise.

To minimize:

$$\Sigma_z \Sigma_s C(z,s).V(z,s) + \Sigma_k \Sigma_d S(k,d)$$

Subject to the constraints:

$T \leq Tm$ ... access time delay

where T = average delay

Tm = maximum delay

$A(d) \leq Am$ .... availability constraint

where A(d) = availability of file d

A = minimum availability

The objective function shown in section 7.1 can be transformed to Mahmood & Riordon's objective function by :

. Ignoring all local activity (L or C_1 is zero)

. Replacing the remote traffic by a constant, say GG.

. Choosing the following decision variable:

$\beta(p,i) = 1$ for all p and i.

. Choosing the same application and support system variables as Casey.

The analytical expressions which show the mapping between Mahmood & our formulation are:

$$C(z,s) = \Sigma_i \Sigma_j PATH(i,j,z).C_z(z).GG$$

The main difficulty encountered is in transforming the response time and availability constraints. The expression for response time by Mahmood & Riordon is:

$$T = w \cdot \Sigma_z l(z).a.\{E(z) - l(z).a\}$$

where w = total traffic in the network

l(z) = traffic over link z

a = average message length

E(z) = speed (bits per second) of link z

In order to reduce the response time constraint (section 7.1) to this format, we make the following additional assumptions:

. The transaction arrival rate, NRrk = poisson

. No parralel processing or loc(t,k,i)ng delays

The mapping analytical expressions are:

$w = \Sigma_t \Sigma_k \Sigma_z M(t,k,z)$ where M(t,k,z) = no. of messages exchanged over link z, due to arrival of a trans. at r

$$l(z) = \Sigma_t \Sigma_k M(t,k,z)$$

$$a = \Sigma_t \Sigma_k \Sigma_z CTV(t,k,z) / \Sigma_t \Sigma_k \Sigma_z M(t,k,z)$$

Mahmood derives the following expression for A(d), the availability of file d:

$$A(d) = \Sigma_k w1(k,d) \cdot a1(k,d) / \Sigma_k w1(k,d)$$

where w1(k,d) = total traffic from node k to file d

a1(k,d) = availability of file d from node k

The expression for availability constraint (section 7.1) can be transformed to Mahmood's expression by assuming that the availability of all nodes is $100\Sigma_i$. The mapping expressions are:

$$w1(k,d) = \Sigma_p \lambda(t,k)(P_r(p,d) + P_w(p,d))$$

$$a1(k,d) = \Sigma_i \Sigma_j \Sigma_z \nabla(t,k,i)\alpha(d,j).PATH(i,j,z).A_a(z)$$

Khabbaz <KHAB80c>

In principle, Khabbaz considers the same problem as the one considered by Mahmood & Riordon. The only difference is that Khabbaz adds another decision variable - the network topology. The decision variables considered by Khabbaz are:

. $\alpha(k,d)$ - allocation of file d to node k

. V(z,s) - allocation of speed s to link z

. TOP(n,c,h) - topology of n nodes, with connectivity c and network diameter h.

Since it has been already shown how Mahmmood's formulation can be obtained from our formulation, we mainly concentrate on addressing the topology consideration introduced by Khabbaz. The topologies considered by Khabbaz are restricted to "maximally reliable topologies" which minimize network partititioning and network diameters for given nodes and links (channels). Topologies of this nature are characterized by:

. Connectivity c of a network, which is a measure of the degree to which a graph resists disconnection when nodes are removed. Formally, connectivity is the minimum size of the node CUTSET, where a node cutset is the set of nodes which, if deleted, will disconnect the network.

. Diameter h of a network, which is the greatest distance between any pair of nodes.

In our formulation, network topology is given by the matrix $B_{tp}(i,j)$ where $B_{tp}(i,j)=z$, if link z directly connects nodes i and j, 0 otherwise. The main characteristics of this topology are: a) the only nodes considered are computer systems which house application systems. The switching nodes are ignored and b) two nodes are directly connected only through a unique link, i.e multidrop topologies are not considered.

We suggest that the topologies considered by Khabbaz can be represented by the $B_{tp}(i,j)$ matrix. In addition, we construct a path matrix which takes into account the network control and adaptive routing, an issue not addressed by Khabbaz. Thus we can reduce our problem formulation to Khabbaz's formulation by utilizing the Mahmood & Riordon's results discussed in section I.1.3.

Chen <CHEN83c>

The main emhasis of Chen's reserach is to calculate the response time of a transaction in terms of update synchronization algorithms. The main features of this calculation are:

1. Several components of transaction response time in DDP networks are identified and analyzed.
2. Transactions are categorized in terms of transaction length (no. of operations performed) and operation type (read or write). Thus the most general transaction type is N/2, which indicates a transaction of length N and read/write operations.
3. The read operations are further categorized into current mode and response mode transactions. Current mode transactions read the most current data from a "primary" site and the response mode transactions read non-current data from secondary nodes, to minimize response time.

4. The read/write operations are also classified into shared or exclusive (no sharing) operations.
5. The service disciplines of the data manager are identified as first come first serve, nonpreemtive priority and preemptive priority disciplines.
6. With these definitions, 10 combinations of service disciplines are identified and detailed queuing analysis are performed for the centrally controlled, Ellis Ring and secondary site synchronization algorithms.

Although the response time calculations are extensive, the allocation algorithm is restrictid to only centrally controlled synchronization algorithms and is formulated to minimize response time subject to storage and communication costs for read/write transactions. Specifically, he solves the following problem:

Given:

- N nodes
- D datasets
- λr, $\lambda r'$ and λu - the arrival rates of read-currency, read-response and update type transactions.
- CCu(i,j), CCr(i,j), CCm(i,j) and CCt(i,j) - the communication costs for sending an update, read, control message or results, respectively, from node i to j.
- CDu(i,j), CDr(i,j), CDm(i,j) and CDt(i,j) - the communication delays encountered for sending an update, read, control message or results, respectively, from node i to j.
- SP(i) and SS(j) - the costs of storing primary and secondary copies at node i, respectively.
- Er(i), Ew(i) and Ew'(i) - data processing times for read, write and update transactions at node i.

Determine:

. $\alpha(d,i) = 1$ if primary copy of dataset d is assigned to node i, 0 otherwise.

. $\alpha'(d,i) = 1$ secondary copy of dataset d is assigned to node i, 0 otherwise.

To minimze the response time =

$$\Sigma_d \ \Sigma_i \ \alpha(d,i) \ \{ \ \Sigma_k \ \lambda w(t,k).ZW(i,k)+\lambda r(t,k).ZRC(i,k)\}$$

$$+ \ \Sigma_d \ \Sigma_j \ \Sigma_k \ \alpha'(d,j)B(j,k)\lambda r'(t,k)ZRR(j,k)$$

where:

. $ZW(i,k)$ = response time for the write transaction
$= CDm(t,k,i) + CDt(i,k) + Gw(\lambda w, \lambda r, Ew, Er)$

. $ZRC(i,k)$ = response time for the read current transaction
$= CDm(t,k,i) + CDt(i,k) + Gr(\lambda w, \lambda r, Ew, Er)$

. $ZRR(i,k)$ = response time for the read current transaction
$= CDm(t,k,i) + CDt(i,k) + Gr(\lambda w, \lambda r', Ew, Er')$

. $B(j,k) = 1$ if read response mode transactions arriving at node k will access a secondary copy of data at j, 0 otherwise.

subject to:

. $\Sigma_d\Sigma_i \ \alpha(d,i).\ SP(i) \leq SCOSP$ .. storage constraint for primary copies

. $\Sigma_d\Sigma_i \ \alpha'(d,i).\ SS(i) \leq SCOSS$ .. storage constraint for secondary copies

. $\Sigma_d\Sigma_i \ \alpha(d,i).\ \{ \ \Sigma_i \ \lambda w(t,k).\ (CCm(t,k,i)+CCw(t,k,i))\} \leq$ comm. cost for write

. $\Sigma_d\Sigma_i \ \alpha(d,i).\ \{ \ \Sigma_i \ \lambda r(t,k).\ (CCm(t,k,i)+CCr(t,k,i))\} \leq$ comm. cost for read current

. $\Sigma_d\Sigma_i \ \alpha'(d,i).\ \{ \ \Sigma_i \ \lambda r'(t,k).\ B(j,k)\ (CCm(t,k,i)+CCr(t,k,i))\} \leq$ comm. cost for read resp

. $\Sigma_d\Sigma_i \ \alpha(d,i) = 1$ ... only one primary copy

. $\Sigma_d\Sigma_i \ \alpha'(d,i)\ .\ B(i,k) = 1$ ... for all k, .. secondary copy

It is difficult to translate this allocation problem to our formulation since we minimize costs (storage + local processing remote processing)

subject to response time and other (availability, storage, program duplication and policy) constraints, while Chen minimize response time subject to storage and communication costs. However, we can derive the expressions for the costs and response time. An inspection of Chen's formulation indicates that we mainly need to show how the G functions (GW, GRC and GRR) can be obtained from our formulation. (All other variables are input parameters that correspond to our parameters.) In order to derive the analytical expression for G, we need to use the following values:

a. $\beta(p,i) = 1$ for all p and i - programs assigned to every node.

b. $C_l(i) = 0$ - no local processing cost

c. Application system parameters:

.$D_c(d,d1) = 0$ for all d and d1 - no consistency relationships

.$P_i(p) = P_o(p) = P_m(p,p1) = 0$ for all p and p1 -one program per transaction and no program i/o.

d. Transaction processing options:

.SRCH(k,x) = based on minimum cost access

.$Q_{da} = 1$ - remote data access is allowed

.$Q_{rt} = 1$ - routing is parallel

e. Update synchronization algorithms

.$U_{pg} = 1$ - oneway piggybacking

.$U_{cp} = 0$ and $U_{lm} = 0$ - commit processor and lock manager are centralized

f. Communication system options:

.$B_{nc} = 0$ - the network control is centralized

.B_{rt} = fixed routing

By substituting these values, we compute the values of L and R by using equations 5.31 and 5.32 in chapter 5. These variables give local processing at various nodes and remote messages exchanged between pairs of nodes. The calculation of the response time in terms of these variables has been shown in chapter 6. The G functions used by Chen are equiv-

alent to the delays encountered at the nodes, which are given by the sum of all the processing delays, device-busy queuing delays and lock-conflict queuing dealys encountered at each node. These derivations were discussed at length in chapter 6. However, we should notice the following differences:

1. We only consider FCFS service disciplines
2. We do not explicitly differentiate between read-response and read-current mode transactions.
3. Chen doen not consider parallel processing of transactions.

In addition, Chen ignores the transaction availability, program duplication and policy restrictions. Chen also ignores the local processing costs and the complex dataset relationships.

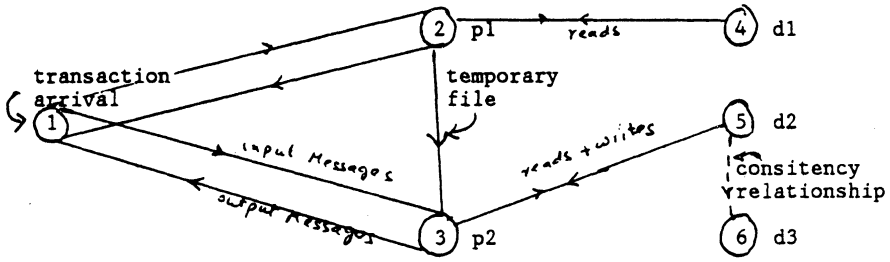
APPENDIX J: MODEL VALIDATION

The predictive model which forms the core of DADP is validated by using walkthroughs and "live" tests on an existing system. The output of the predictive model consists of six dependent variables: the local processing cost, the remote processing (communication) cost, storage cost, program duplication cost, transaction availability, and transaction response time. The walkthroughs verify the DADP estimates of these variables by using a paper and pencil method and the live tests validate the transaction response time computations of DADP on an existing distributed system. In addition, live tests validate the i/o time, cpu time, communication time and queuing delay estimations of DADP.

J.1. Walkthrough Validation

The walkthroughs verify that the analytical expressions of the dependent variables derived in chapters 4 and 5 correctly represent real life situations and that DADP correctly implemented the analytical expressions. We used a two program, three dataset transaction with inter-program and inter-dataset relationships in a six node DDP network for walkthroughs (see figure J.1). Since it is difficult to do numerous hand calculations, the paper and pencil analysis primarily consisted of calculating the dependent variables by studying the flow of local and remote activities when transactions arrive at node 1, programs p1 and p2 are assigned to nodes 2 and 3, respectively, and datasets d1, d2 and d3 are allocated at nodes 4, 5 and 6, respectively (see figure J.1). The tests were run to verify the results for fully connected network and a star topology network with several transaction processing and synchronization algorithms (see table J.1). It can be seen from table J.1 that the walkthrough results obtained by excercising the flow of transactions in DDP match 100% with the results produced by DADP. Additional

FIGURE J.1: TEST ENVIRONMENT FOR WALKTHROUGHS



NOTES:
1. The transaction arrival rate is 3 per unit time on node 1
2. Each dataset is 100 KB. A consistency relationship exists between d2 and d3.
3. Program p1 reads 1 input message, reads d1 60 times, generates 1 display message, and sends 1 temporary record to p2.
4. Program p2 reads 1 input message, reads d2 80 times, updates (consistency) d2 4 times, and generates 12 messages.
5. The cost of storing 1 KB is 6 cents on all the nodes, the cost per local service request is 2 cent on all the nodes and the cost per remote service is 5 cent between all nodes.

TABLE J.1: WALKTHROUGH VALIDATION

	PAPER AND PENCIL ESTIMATES					DADP ESTIMATES				
	Storg. cost	Loc. cost	Comm. cost	Resp. time	Avail.	Storg. cost	Loc. cost	Com. cost	Resp time	Avai
Fully Conn	1800	888	7263	6.1	0.85	1800	888	7263	6.1	0.85
LDA, Basic	1800	888	135	1.2	0.85	1800	888	30	1.2	0.85
LDA, Ring	1800	888	180	1.3	0.85	1800	888	180	1.3	0.85
RDA, Basic	1800	888	4545	4.6	0.85	1800	888	4565	4.6	0.85
2PL	1800	4104	13185	15.3	0.85	1800	4104	13185	15.3	0.85
Timestamp	1800	4104	8865	11.9	0.85	1800	4104	8865	11.9	0.85
Ellis Ring	1800	4104	900	5.6	0.85	1800	4104	900	5.6	0.85
Thomas	1800	4104	540	5.4	0.85	1800	4104	540	5.4	0.85
CCA	1800	4104	1035	5.7	0.85	1800	4104	1035	5.7	0.85
CLA	1800	4104	14769	11.9	0.85	1800	4104	14769	11.9	0.85
Optimistic	1800	888	4545	4.6	0.85	1800	888	4545	4.6	0.85
Pessimist.	1800	20184	42585	81.1	0.85	1800	20184	42585	81.1	0.85
SDD1	1800	4104	13185	15.2	0.85	1800	4104	13185	15.2	0.85

NOTES:
1. The first three tests are based on an idealized, fully connected support system with no update synchronization.
2. The transaction availability is unchanged because the data and program allocations are unique.

tests were conducted to study the effect of program and data duplication by replicating programs and datasets to all nodes and by employing adaptive routing.

J.2 Live Tests

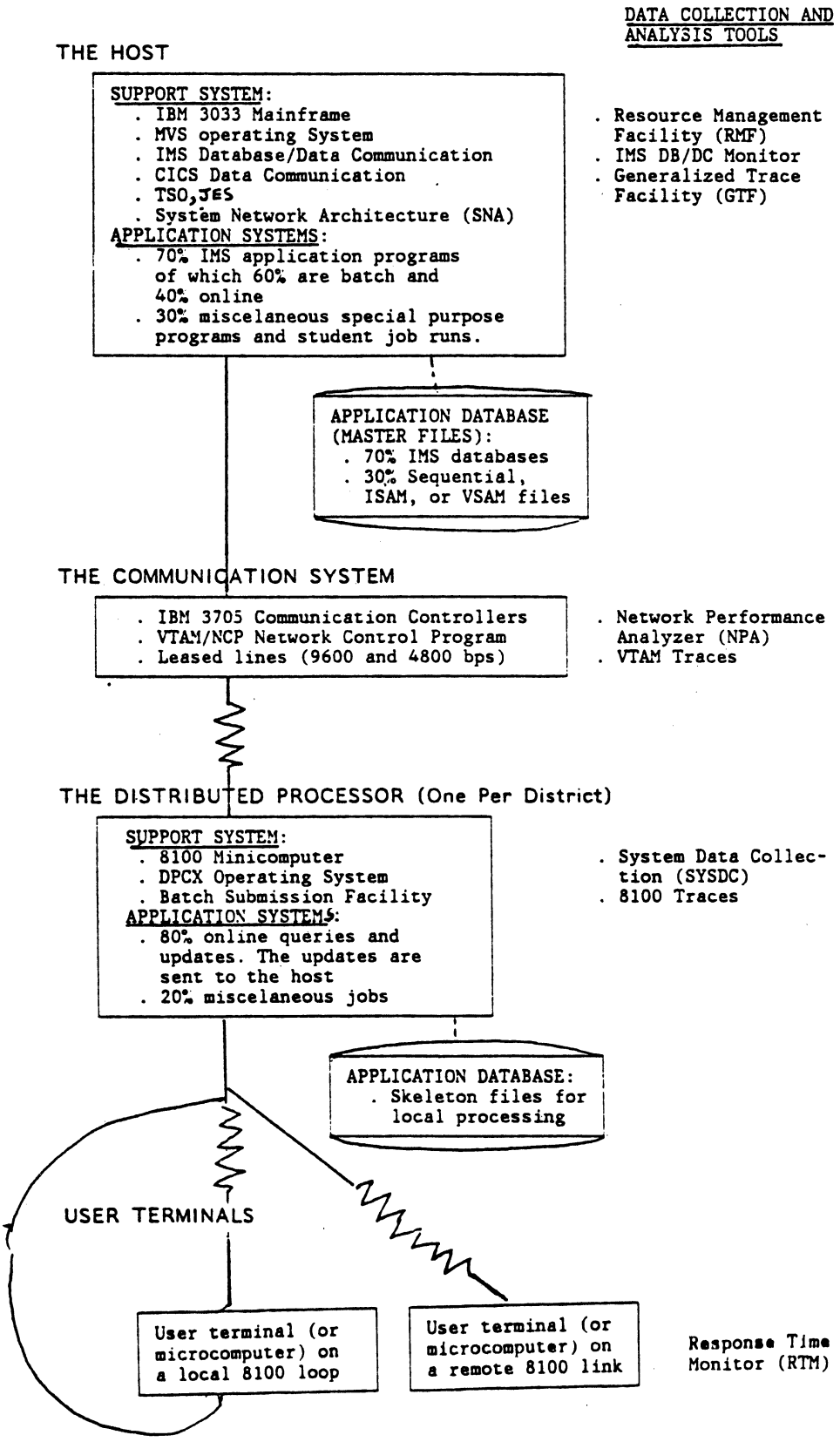
The live tests were run on an IBM 3033 computing system connected to several IBM 8100 minicomputers in a star network topology at the Wayne Intermediate Computing Center. Section J.2.1 describes this support system in detail. The applications system chosen for verification, described in section J.2.2, is a student information system which is distributed among the 8100 systems with a master copy at the 3033 (host) computer. As stated above, the purpose of the live tests is to study the response time (with and without queuing), the cpu time, the i/o time and the communication time estimates of DADP. A major problem in measuring these variables is the very wide range of tools needed for collecting and analyzing data at the host, 8100 and the communication system level (see section J.2.3). Figure J.2 shows the support system, the application system and the various tools.

The actual tests, discussed in sections J.2.4 and J.2.5, consist of response time analysis with and without queuing. It should be noted that DADP is actually designed for global program and data allocation and is not intended for detailed local physical design and analysis. However, we have found that an implemented distributed application system can be adequately represented and analyzed by using DADP if proper data, for example the no. of physical i/o per logical i/o, is available.

J.2.1 The Support System

The support system, shown in figure J.2, consists of a central host computer, a communication system, and several remotely located distributed processors. The host computer is an IBM 3033 system with 16 MB of storage and about thirty IBM 3350 disk drives. The average i/o time of

FIGURE J.2: THE TEST ENVIRONMENT



the disk drives is about 40 milliseconds and the main processor speed is about 5 MIP (million instructions per second). The host uses an MVS operating system and runs an IMS DB/DC system, along with several other subsystems (JES, TSO, CICS, VTAM etc). On a typical day, about 4000 batch jobs, 10,000 IMS transactions and 5,000 CICS transactions are processed on the host. At present no update synchronization algorithm is being used; instead the redundant data is synchronized by using store forward applications.

The host is connected to a network of thirty 8100 minicomputers located throughout Michigan, some in the Upper Peninsula. The communication system consists of 9600 bits per second (bps) or 4800 bps leased lines that are controlled by two IBM 3705 controllers, operating under the IBM System Network Architecture (SNA). The communication system is centrally controlled and employs a fixed routing algorithm.

Each 8100 system has at least one disk, with an average i/o time of 90 milliseconds, and processor speeds ranging from 0.3 MIP to 0.8 MIP. Terminals can be connected to the 8100 system on a 38.4 K bps local area network loop or as 4800 bps remote links. The 8100 system runs under a DPCX operating system and facilitates interactive sessions with 8100 system or the host, along with the batch job submission and printing/spooling facilities.

At present, several 8100 terminals are being replaced with microcomputers, which yields a three-tiered distributed system with the host at first level, the 8100 system at second level and the microcomputer at the third level. Thus a user of this system can access files located at the micro, 8100 or host locations.

J.2.2 The Application System

The application system chosen is a student information system which provides the information about students, the courses, the school buildings and the board offices. The information system also allows for the student scheduling, student registeration and course drop/add. Most of

the information is kept at the host mainframe under an IMS database system. Portions of the information, the "skeleton" files, are kept at the local IBM 8100 systems in each Board office for quick-queries and edit checking for the host batch jobs (see figure J.2). Any jobs which update the skeleton files generate batch jobs which are routed to the host so that the master files can be updated at the host.

We chose the following transactions of the student information system for evaluation:

- Transaction SINFO, which retrieves basic student information (name, phone, address, courses enrolled) from the skeleton files. This transaction activates program EXTRACT which issues, on the average , 20 calls to a Relative Sequential Dataset (RSDS) at the 8100 system. An RSDS file is similar to ISAM in operation. The program receives one input, about 80 characters, and generates an output of one screen with about 960 characters.
- Transaction DINFO which retrieves the district information directly from the host by activating an IMS program, which receives an 80 character input, reads IMS database 7 times and generates a 960 character display.
- Transaction FLEMNT which first updates the skeleton file (10 reads/writes) and then routes the updates to IMS for master file synchronization (14 read/writes).

It has been shown in chapters 2 and 7 that transactions of this nature can be easily represented by using the application system parameters.

J.2.3 The Data Collection and Analysis Tools

In order to measure the response time, the i/o time and the queuing time in test environment, tools are needed at the host, communication and the 8100 levels (see figure J.2). The host tools consist of resource management facility (RMF), IMS DB/DC monitors and generalized

trace facilities (GTF). An overview of these tools is given below and the main information used in validation is briefly discussed. Details of the pertinent information will be presented when the actual experiments are described in next sections.

RMF collects information at the host system running under MVS and generates the following reports: the cpu report which shows the cpu busy and wait time in the observation period; the channel activity report which shows the number of channel requests and the channel busy percentage per channel; i/o device report which shows the number of i/o requests issued per device, the device busy percentage, the average queue length per device etc; and the work load activity reports which shows, for each "performance group" the number of i/o issued, the cpu services used and other information. (A performance group consists of one or more programs that are grouped together for purpose of performance monitoring.) We have used the RMF reports primarily to measure the average queue lengths per device.

The IMS DB/DC monitor collects detailed information about IMS transactions and provides the following information: number of database (DL1) calls issued per transaction, the number of i/o issued per call, the average elapsed and cpu time per call, the elapsed time between transaction initiation and termination, number of DL1 calls per dataset, number of messages received and displayed per transaction, etc. The IMS DB/DC monitor has been used very heavily in this analysis.

The GTF traces each service (local i/o, communication i/o, cpu requests, program initiation and termination, and interrupts) in the system. Most of the GTF reports are used for diagnostic purposes, but in several cases these reports are used for measuring the flow of a single request in the system because GTF attaches a time stamp for each record. We have used the GTF traces to obtain an accurate measure of message flow in the system.

The communications system basically uses two types of data collection facilities: the VTAM traces and the network performance analyzer (NPA) reports. These traces and reports show the length of the communication messages, the destination and origin of the messages, the number of messages per link, the average time per message, the average queue length per link, the number of errors (retries) per link, and polling rates. We have used the NPA reports in our tests to determine the average time per message, the queue length per link and the average message length.

The 8100 system has limited data collection and analysis facilities. The main tool available on the 8100 system for data collection is SYSDC, which shows processor utilization, the storage pools, the number of disk accesses, the number of dataset references and the average number of i/o per dataset referenced. The major shortcoming of SYSDC is that it does not record activity by application, instead it shows the activity per device. Thus SYSDC can be used accurately when only one application is running in the system because then all the activity is due to this application. A trace facility is available on the 8100 system, but is very difficult to use.

The Response Time Monitor (RTM) is a hardware device which collects the information about the response time from a terminal. It can display accurately the actual time elapsed between pressing the enter key on a terminal and the message display. RTM also can display a histogram of response times and is used in these tests very heavily.

J.2.4 Experiments Without Queuing

The transactions SINFO, DINFO and FLEMNT were submitted from an 8100 local attached terminal when no backround workload existed. The Response Time Monitor was attached to the terminal for recording the transaction response time. In addition to the response time measurements, we measured the i/o time, the communication time and the cpu time for each transaction. These measuremnts can be used to compute the total

TABLE J.3 : SAMPLE MEASUREMENTS

a). IMS DC MONITOR REPORT (PROGRAM SUMMARY)

IMS MONITOR ****PROGRAM SUMMARY**** TRACE START 1984 025 18:45:05 TRACE STOP

PSBNAME	NO. SCHEDS.	TRANS. DEQ.	DL/I CALLS	DL/I CALLS /TRAN	I/O IWAITS	I/O IWAITS /CALL	TRAN. DEQN. /SCH.	CPU TIME /SCHED.	ELAPSED TIME /SCHED.
SECONLEO	17	17	70	4.1	0	0.0	1.0	7182	17669
XFORMAT	5	5	15	3.0	0	0.0	1.0	6713	4797
XATTENDA	1	1	56	56.0	0	0.0	1.0	138214	206674
XTRANSCP	1	1	10	10.0	0	0.0	1.0	16014	30679
XSRSTNTI	1	1	7	7.0	0	0.0	1.0	10927	10693
XMINIFMI (FLEMNT)	3	3	42	14.0	12	0.2	1.0	28392	249791
VFORMAT	1	1	3	3.0	0	0.0	1.0	6579	3743
VCLASLST	2	2	93	46.5	87	0.9	1.0	211459	2141693
XDISTINFI (DIATFO ←)	2	2	14	7.0	1	0.0	1.0	10991	109677
VOROPADD	6	6	106	17.6	24	0.2	1.0	67547	291531
XMASTSKE	1	1	6	6.0	0	0.0	1.0	10140	15831
XBUILDIN	1	1	8	8.0	0	0.0	1.0	12376	27111
XCALENDA	2	2	112	56.0	0	0.0	1.0	89140	159671
VGRADEPN	2	2	14	7.0	0	0.0	1.0	11414	37795
VGDEHIST	1	1					1.0	13989	36343
**TOTALS	46	46	956	12.0	124	0.2	1.0	91921	179019

→ I/O per call

b). 8100 SYSDC REPORT (DISK, CPU UTILIZATION)

DATA COLLECTED FOR ALL DISKS

No. of disk accesses

DISK	CYLINDER 1-4	5-8	9-12	13-16
1	00	11	76	11
2	00	00	00	00
3	00	00	00	00
4	00	00	00	00
5	00	00	0C	00
6	00	00	00	00
7	00	00	00	00
8	00	00	00	00

INTERRUPT LEVEL UTILIZATION

LEVEL 1	LEVEL 2	LEVEL 3	LEVEL 4
00	0C	17	0C

CPU utilization for applications

LEVEL 5	LEVEL 6	LEVEL 7	WAIT
00	01	00	80

VIRTUAL STORAGE REFERENCES

POOL SIZE = 216K
MINIMUM STORAGE AVAILABLE = 5K

ACTUAL	% OF FAULTS
3537	01

reflect 1% of I/O

c). NPA REPORT (LINE UTILIZATION)

DATA FOR LINE LNESD FROM 02/02 00:00:00 TO 02/02 23:59:59

TIME	TOT REC	INTVL MIN:SC	OUTQ LEN	MSGS /MIN	BYTES /SEC	LINE-UTZ PHY	LINE-UTZ SDRY	POLLS/MIN RATE	POLLS/MIN %NEG	ERRS	UNLOCKED RETRANSMIT PIUS	UNLOCKED RETRANSMIT BYTES
8:40:49		59:59	1	12	90	8	0	295	98	0	0	0
9:40:53		60:01	1	36	141	12	0	290	95	0	0	0
10:40:50		59:58	1	77	276	23	0	267	90	0	0	0
11:40:53		60:00	1	207	754	63	0	217	72	0	0	0
12:40:48		59:59	1	50	191	16	0	282	93	0	0	0
13:40:50		60:01	2	89	349	29	0	263	90	0	0	0
14:40:52		60:01	2	36	134	11	0	291	94	0	0	0
15:40:54		59:59	3	42	129	11	0	293	93	0	0	0
16:40:49		59:58	3	50	154	13	0	289	92	0	0	0
17:40:49		59:59	3	20	88	8	0	297	96	0	0	0
18:40:48		59:58	3	26	104	8	0	295	95	0	0	0
SUMMARY:		10:59:56	1	59	219	18	0	280	92	0	0	0

PAGE:

d). RMF DISK UTILIZATION REPORT

DEV ADR	VOLUME SERIAL	LOG CHN	DEVICE ACTIVITY COUNT	DEVICE ACTIVITY RATE	AVG SERV TIME	PERCENT QUEUE LENGTH DISTRIBUTION 0	1	2	3+	AVG Q LNGTH	%REQ DEFER DISTRIBUTION DEV BUSY	PATH BUSY	% DEV BUSY	% CU DELAY
250	WCISDM	2	1,379	0.766	0.015	99.3	0.7	0.0	0.0	0.01	0.00	100.00	1.22	0.00
251	WCISDC	2	4,058	2.254	0.044	98.8	1.2	0.0	0.0	0.01	31.82	68.18	10.44	0.00
252	TESTO1	2	0	0.000	0.000	100.0	0.0	0.0	0.0	0.00	0.00	0.00	0.00	0.00
253	SPOOL1	2	23,044	12.803	0.042	64.0	14.7	8.0	13.3	0.95	98.30	1.70	53.89	0.00
254	MVS13M	2	8,154	4.530	0.033	96.2	3.3	0.3	0.1	0.04	74.36	25.64	15.17	0.00
255	WCISDB	2	7,993	4.441	0.142	92.3	7.0	0.6	0.1	0.08	94.04	5.96	63.30	0.00
256	SYSLB2	2	10,340	5.745	0.036	93.9	4.9	0.9	0.3	0.08	85.00	15.00	20.94	0.00
257	WCISD4	2	200	0.111	0.059	99.9	0.1	0.0	0.0	0.00	0.00	100.00	3.67	0.00

cost of running a transaction which is usually a function of the i/o, cpu and communication activity generated for the transaction. Since the transactions originated at one 8100 (node 1) and were either processed at node 1 or the host (node 2) we can ignore the rest of the network and concentrate on a two node network interconnected through a 4800 bps leased line. Table J.2a shows the sample input data and table J.2b summarizes the results of these experiments and compares these results with the DADP computed values. Let us now analyze these results in detail.

I/O TIME: This quantity is computed by multiplying the number of i/o requests generated by the transaction with the average i/o time. DADP uses the following formula to compute the i/o time at node i for transaction t:

$$IO(t,i) = f1(i) \Sigma_k L(t,k,i) E(i)$$

where L(t,k,i) is the number of local service requests executed at node i due to arrival of transaction t at node k, E(i) is the average time per i/o and f1(i) is the conversion factor which converts L into physical i/o. L is computed by DADP based on the analytical expressions developed in chapter 4 and represents the dataset references (logical i/o). Figure J.2c shows that L is calculated correctly by DADP. E(i) is a function of device characteristics and is known for the host, about 40 millisecond per i/o, as well as the 8100 system, about 90 millisecond. Computation of f1 is not easy because this factor depends on a large number of physical database design factors, and is assumed to be 1 for most of the global design problems. However, fortunately, the IMS DC monitor at host and the SYSDC at the 8100 system both give accurate average values of f1 for given system configurations. For example, IMS DC monitor prints, for each transaction, the number of DL1 calls which represent the program to dataset accesses or L(t,k,2), the number of actual i/o issued and the i/o to DLI ratios (see table J.3a). The i/o per call actually represents f1 at the host. In case of the 8100 system, SYSDC gives statistics about the number of dataset calls, called the

RSDS references, and the percentage faults which show the number of times the information needed was not found in the buffers and thus caused an i/o (see table J.3b). These percentage faults represent f1 at the 8100 system. Based on a series of measurements extending over five working days, f1 is assigned a value of .2 at the host and a value of .5 at the 8100 system (see table J.3). These values are lower than one would expect in an actual situation because these measurements were taken when the system was not busy (no queuing) and thus the data is loaded and kept in buffers.

We should comment about the measured i/o values recorded in table J.2. The i/o time for IMS is given directly by the IMS DC monitor and the i/o time at the 8100 is computed by using the percentage of RSDS faults. Note that the i/o time computed by DADP and the actual i/o time measured for all the transactions match quite well.

COMMUNICATION TIME:

DADP uses the following formula to compute the communication time at link z for transaction t:

$$COMTIME(t,z) = f2(z) \sum_k M(t,k,z)\ E(z)$$

where M(t,k,z) is the expected number of messages sent over link z when transaction t arrives at node k, f2(z) is the average length of message at link z and E(z) is the speed of the link z. DADP computes M based on the following expression derived in chapter 5:

$$M(t,k,z) = ff \times \sum_i \sum_j R(t,k,i,j) \cdot PATH(i,j,z)$$

where R(t,k,i,j) is the number of remote service requests exchanged between nodes i and j, PATH(i,j,z) is the probability that a message between nodes i and j traverses link z, and ff is a conversion factor which converts R into a physical communication message. R is calculated by DADP very accurately (see table J.2c), PATH(i,j,z) = 1 when i = 1, j=2 and z=1, zero otherwise, and ff depends on the characteristcs of the communications manager. In our situation, each application reads about 100 bytes of input and generates a 960 byte (full screen) display. The

TABLE J.2: VALIDATION WITHOUT QUEUING

a). SUMMARY OF MAIN RESULTS

	DADP ESTIMATES (in seconds)				ACTUAL MEASUREMENTS			
	i/o time	comm. time	cpu time	total time	i/o time	comm. time	cpu time	total time (note1)
DINFO	0.9	0.0	0.2	1.1	1.1	note2	0.1	1.3
SINFO	0.08	1.6	0.01	1.69	0.06	1.6	0.05	1.91
FLEMNT	0.61	1.6	0.15	2.36	0.72	1.7	0.36	3.0

NOTES:
1. The total measured response time also includes some delays that cannot be directly measured, e.g. the program loading time and dataset open processing. Due to this reason, the total response time measured by the user is always slightly greater than the sum of the cpu, i/o and communication time even when no device queuing exists in the system.
2. There is no direct way of measuring the communication time for a terminal connected to the 8100 system on a local area loop.

b). DETAILS OF CALCULATIONS AND MEASUREMENTS

	DADP ESTIMATES			ACTUAL MEASUREMENTS		
	Transactions			Transactions		
	SINFO	DINFO	FILEMNT	SINFO	DINFO	FILEMNT
L(t,1,1)	20.0	0.0	10.0	20.0	0.0	10.0
L(t,1,2)	0.0	7.0	14.0	0.0	7.0	14.0
f1(1)	0.5	-	0.5	0.3	-	0.3
f1(2)	-	0.2	0.2	0.2	0.2	-
IO(t,1)	0.9	0.0	0.45	1.1	0.0	0.5
IO(t,2)	0.0	0.08	0.16	0.0	0.06	0.22
R(t,1,1,2)	0.0	2.0	2.0	0.0	2.0	2.0
M(t,1,1)	0.0	4.0	4.0	0.0	4.0	4.0
f2(1)	-	240.0	240.0	-	210.0	210.0
COM(t,1,1)	0.0	1.6	1.6	0.0	1.6	1.7

SNA communication system has a maximum packet size of 465. Thus, for input messages, ff=1 and for output messages, ff=3. Based on these calculations M(t,k,z) can be estimated by DADP (see table J.2c). f2(z), the average message size, based on NPA analysis (fig J.3c), is 220 bytes and E(z) is 4800 bits per second. The computed communication time COM(t,k,1) is shown in table J.2c.

It should be noted that our model does not include the terminal to 8100 communication delay, instead the communication time between the 8100 and the host is estimated. As can be seen from table J.2 the measured and computed values of the communication time are quite close. Please also note that we can obtain a rough estimte of communication time without considering the conversion factors ff and f2. We have included these calculations here to illustrate the flexibility of DADP.

CPU TIME: DADP uses the following formula to estimate the cpu time at node i due to transaction t:

$$CPTME(t,i)=APP(t,i)+f3(i)\Sigma_k L(t,k,i)E'(i)+f4(i)\Sigma_k\Sigma_j R(t,k,i,j)E'(i)$$

where f3(i) and f4(i) are the number of cpu instructions executed per local and remote service request, E'(i) is the cpu speed at node i and APP is the application logic cpu time. f3 and f4 are about 5000 for the host and about 1000 for the 8100 system, E' is about 4 million instructions per second for the host, 0.5 MIP for the 8100 system, and the APP logic time can be ignored because the three transactions are all i/o bound. It can be seen from table J.2a that DADP underestimates the cpu time. This is because we do not include the application logic time APP. Thus, depending on the type of application, DADP can understimate the cpu time between 5 to 30%.

TOTAL RESPONSE TIME: DADP computes the total response time by adding the i/o time, the communication time and the cpu time, to get a lower bound on response time. As can be expected, this computation does not include queuing and program/dataset load timings.

J.2.5 Queuing Analysis

TABLE J.4: VALIDATION WITH QUEUING

	DADP ESTIMATES			ACTUAL MEASUREMENTS		
	Transactions			Transactions		
TIME (secs)	SINFO	DINFO	FLEMNT	SINFO	DINFO	FLEMNT
i/o time	0.9	0.08	0.61	note1	0.12	0.79
i/o wait	0.5	0.20	0.2	note1	0.32	0.42
total i/o time	1.4	0.28	0.81	1.6	0.44	1.21
comm. time	0.0	1.6	1.9	note1	1.6	2.0
comm. queue time	0.0	0.3	0.3	note1	0.4	0.5
total comm. time	0.0	1.9	2.2	note1	2.0	2.5
cpu time	0.2	0.01	0.15	0.4	0.05	0.4
cpu queue time	0.1	0.09	0.11	note1	note2	note2
total cpu time	0.3	0.1	0.26	note1	0.05	0.40
total resp. time	1.7	2.28	3.27	2.1	3.1	4.5

NOTES:
0. The measurements reflect the mean values of the variables for ten experiments
1. These quantities cannot be measured in the 8100 system because SYSDC does not collect information by task. Thus when background wokload exists, then the i/o time, i/o queue time and cpu wait time for a given transaction cannot be measured directly.
2. The cpu queue time cannot be measured easily.

In order to verify the queuing calculations, we first need to validate the following assumptions of the queuing model described in chapter 5.

1. The devices exhibit homogeneous arrival rates and service times, i.e. the number of arrivals in the system and device service times are not affected by the state of the system.
2. The transactions do not overlap the use of devices.
3. For the observation period, the number of arrivals in the system equal the number of departures (the job flow balance assumption).

In the support system being studied the arrival rates are independent of the state of the system, except in case of extremely busy periods when the users are discouraged and choose to do something else. The service time of almost all the system devices is also homogeneous because service times are function of physical hardware characteristics. None of the transactions being studied use i/o overlap. The system being studied operates in such a manner that most of the requests start arriving at 8 A.M. and very few requests are received during the lunch hour. After lunch, the requests start arriving again and subside completely at 5:00 P.M. The average online transaction turnaround is about 5 seconds and average batch turnaround time is about 3 minutes. With such short turnaround times, and almost no jobs left overnight for next day execution, the observation period of 4 hours (8 to 12 and 1 to 5) satisfies the job flow balance assumption although shorter observation periods, say one hour, can be used effectively. Thus the support system does satisfy all the stated assumptions.

Now let us consider the queuing calculations. Please recall that we can only study device-busy queuing because the locking delays do not exist due to the absence of an update synchronization algorithm. In order to include device queuing, experiments were run at about 3 P.M. so that the effect of background workload could be estimated. Table J.4 summarizes the results of the experiments and shows the processing, queuing and total times for i/o, communication and cpu times.

The queuing delays at the host can be measured directly from the RMF data reports which show the average queue lengths at various devices (see table J.3d). The queuing delay at the 8100 cannot be measured directly but NPA gives the queuing delays at the network level (see table J.3c). DADP computes the queuing delays by using the expressions derived in chapter 5. As can be seen from table J.4, DADP underestimates the queuing delays and total response times because we do not consider delays due to program loading, directory searches, channel/control unit

queuing, buffer queuing and paging. Due to these factors, DADP can understimate the queuing delays by about 70% in busy systems with heavy paging and channel contention. However, in moderatley loaded systems, the queuing delays of DADP rarely deviated beyond 30% for the conducted experiments.

J.3 Summary

An attempt was made to validate and verify the DADP estimates. It has been demonstrated that DADP can represent real life situations and produces approximate measures of local i/o, communication traffic, cpu activity and total response times. However, a key factor in such measurements is the determination of the conversion factors which translate service requests, a measure of "logical" i/o that is computed precisely by DADP, into physical i/o. The success of this task is critically dependent on the availability of adequate data collection facilities. Fortunately, the test system used in this analysis provides such tools.

The model underestimates the queuing delays by about 30% because the node queuing model employed is simplistic and does not consider paging, buffering and priority scheduling. However, this model is intended for global design problems and is designed for approximate analysis at the detailed local design steps, which are beyond the scope of this research.

BIBLIOGRAPHY

a). GENERAL INFORMATION, OVERVIEWS

ABRA76a). Abrams, M.D., 'Network Hardware & Components', Computer Networks: A Tutorial, IEEE, 1976.

ADIB78a). Adiba,M. and Chupin,J.C; 'Issues in Distributed Data Base Management Systems: A Technical Overview', 4th Interntl. Conference on Very Large Data Bases, W. Berlin, Sept. 1978, pp.89-110.

ANDE75a). Anderson, G. & Jensen, E. D., 'Computer Interconnection Structures: Taxonomy, Characteristics & Examples', CACM, Dec. 1975

ASCH74a). Aschim, F., 'Data Base Networks-An Overview', Management Information, Vol.3.1, Feb. 1974, pp.12-28

BLAN76a). Bland, R.P., 'Network Components & Vocabulary', Computer Networks: A Tutorial, IEEE, 1976.

BAGL79a). Bagley, J.D., 'Some Definitions Related to Distributed Data Processing', SHARE draft, 03/15/79

BENN83a). Bennett, J.M., 'Computer Networks', Encyclopedia of Computer Science and Engineering', 2nd edition, Edited by Ralston, Van Nostrand Reinhold, 1983, pp.359-364.

CHAM79a). Champine, G.A.,'Trends in Database Processor Architectures', IEEE Compcon, Spring 1979, pp.69-71.

CHAN77a). Chandy, K.M.'Models of Distributed Systems', IEEE Conference on Very Large Data Bases, May 1977, Tokyo, pp.115-120

COTT77a). Cotton, I.W.,'Computer Network Interconnection', Proc. of 2nd Workshop on Distributed Data Mgmt & Computer Networks, May 1977, pp.3-18

DAVI79a). Davies, D.W. et al, 'Computer Networks & Their Protocols', John Wiley & Sons, 1979.

DOLL78a). Doll, D.R., 'Data Communications - Facilities, Networks and Systems Design', John Wiley & Sons, N.Y. 1978

DOWN76a). Down, P.J and Taylor,E.F., 'Why Distributed Computing', NCC Publications, Manchester 1976

FARB72a). Farber, D.J.,'Networks: An introduction', Datamation, April 1972

FRAT74a). Fratta, L., and Montarian,U., 'Analytical Techniques for Computer Networks Analysis and Design', Computer Architectures and Networks, Edited by E.Gelenbe and R.Mahl, North Holland Publishing Co., 1974, pp.155-187

FRY76a). Fry,J. and Deppe,M.,' Distributed Data Bases-A Survey of Research', Computing Networks, Vol.1, No.2, 1976, pp.1-12

GANZ76a). Ganzhorn,K.,' Evolution in Communication Systems', Proc. of 6th Infmatics Symposium, Bard Hamburg, Nov. 1976, pp. 361-380.

HELM76a). Helm, H.A., 'Communications & Computers', Encyclopedia of Computer Sciences, Petrocelli/Charter Books, N.Y. 1976

HEVN81a). Hevner, A.R., 'A Survey of Data Allocation and Retrieval Methods for Distributed Systems', School of Business and Management Working paper 81-036, Univ. of Maryland, Oct. 1981.

ISAC79a). Isacson,P., Editor, 'Microprocessors & Microcomputers', Selected papers from COMPUTER, IEEE catalog no.THP662-0, 1979

ITKO78a). Itkowtz, A. E., 'A Survey of Computer Communication Protocols: Vol. 1', Report. no. CERL-TR-0-1, U.S. Army Engineer, Construction Engineering Research Labs, Champigne, Illinois, 1978.

KALL78a). Kalley, N.D.,' Learning To Communicate With One Another', INFOSYSTEMS report, No.7, 1978, pp.43-47

KIMB75a). Kimbleton,S.R. and Schneider,G.M., 'Computer Communication Networks: Approaches, Objectives & Performance', ACM Computing Surveys, Vol.7, No.3, Sept. 1975

KLEN76a). Kleinrock,L., 'Queuing Systems', Vol.2, John Wiley & Sons, 1976

MARI79a). Mariani, M.P. and Palmer, D.P., 'Tutorial: Distributed System Design', IEEE Catalog No. EHO 151-1, 1979.

MART69a). Martin,J., 'Telecommunications and the Computer', Prentice Hall, Englewood Cliffs, N.J., 1969

MART73a). Martin, J., 'Systems Analysis for Data Transmission', Prentice Hall, Englewood Cliffs, N.J., 1973

MART81a). Martin, James,'Design and Strategy for Distributed Data Processing', Prentice Hall, 1981.

MCGL78a). Mcglynn,D,R.,'Distributed Processing & Data Communication', John Wiley & Sons, 1978

OKEE78a). O'Keefe,P.M.W., 'Distributed Processing and Corporate Committment', IEEE Compcon, Spring 1978, pp.348-350

PARG78a). Pargellis, A.M., Coffey, T.W., Moraski, R.M., 'In-Depth Assessment of the Status of Distributed Databases and Potential U.S. Navy Requirements', Data Solutions Corporations, Vienna, Virginia, June 1978.

PATR80a). Patrick, R.L., 'Application Design Handbook for Distributed Systems', CBI publishers, Boston, 1980.

ROME80a). Rome Air Development Center, 'Distributed Database Technology: State of the Art Review', RADC-TR-80-7, 1980.

ROTH77a). Rothnie,J.B. and Goodman,n.'A Survey of Research & Development in Distributed Data Base Management', 3rd Conference on Very Large Data Bases, Tokyo, Oct. 1977, pp.48-60

SALT78a). Saltzer, J.H., 'Research Problems of Decentralized Systems with Largely Autonomous Nodes', Operating Systems, Lecture Notes in

Computer Sciences, 60, Edited by G.Goos and J.Hartman, Springer Verlarg, N.Y. 1978, pp.584-593.

SOBO78a). Sobolewski,J.S., 'Data Communication Networks', Encyclopedia of Computer Sciences, Petrocelli/Charter Books, N.Y. 1978

TRIP80a). Tripathi, A.R. et al, 'An overview of Research Directions in Distributed Processing', COMPCON Fall 1980.

b). DISTRIBUTED APPLICATION SYSTEM DESIGN
(QUALITATIVE ANALYSIS)

BRAY76b). Bray,O.H., 'Distributed Data Base Design Considerations' IEEE Computing Networks Symposium, Gaithusburg, 1976, pp.121-127

FISH80b). Fisher, P.S. et al ,'A design Methodology for Distributed Databases', Compcon Fall, 1980.

HEBA77b). Hebalker, P.G. and Tung,C. 'Logical Design Considerations for Distributed Data Base Systems', IEEE COMPSOC, Nov. 1977, pp.562-580

HEBA78b). Hebalker, P.G., 'Application Specification for Distributed Data Base Systems', 4th Conf. on Very Large Data Base Systems, W.Berlin, Sept. 1978, pp.442-229.

HEIN80b). Heinselman, R.C., 'System Design Selection for Distributed Data Systems', COMPCON Fall, 1980.

KUNI77b). Kunii,T. and Kunii,H., 'Design Criteria for Distributed Database Systems', IEEE conference on very Large Data Bases, May 1977

LAWS78b). Lawson, T.J., and M.P. Mariani, 'Distributed Data Processing System Design - A Look at the Partitioning Problem', proc. COMPSAC 78.

OBER78b). Obermarch, R.L., 'Distributed Data Bases', IBM Technical Bulletin, G320-6019, Sept. 1978

OBER79b). Obermarch, R.L., 'Distributed Data Bases', Share 52, San Francisco, April 1979

PALM78b). Palmer, D.F, and W.M. Denny, 'Distributed Data Processing Requirements Engineering: High Level DDP design', COMPSAC 78.

RAMA77b). Ramamoorthy, C.V; Ho,G.S; Krishnarao,T and Wah,B.W., 'Architectural Issues in Distributed Data Base Systems', IEEE Conf. on Very Large Databases, May 1977, Tokyo, pp.122-130

ROCK77b). Rockart, J.F., 'The Management Issues in Centralization vs. Decentralization of Information Systems', SHARE 48, Houston, March 1977, pp.487-494

SAXT78b). Saxton,W.A. and Edwards,M., 'Decision Model for Distributed Processing', Infosystem, Sept. 1978 pp. 88-92

SMIT79b). Smith, J.L., 'Alternatives in the Architecture and Design of Distributed Databases', The Australian Computer Journal, Vol. 11, No. 1, Feb. 1979

SMOL79b). Smoliar, S.W., 'Using Applicative Techniques to Design of Distributed Systems', proc. of Specification of Reliable Software Conf., IEEE, April 1979

SMOL79b2). Smoliar, S.W. & Scalf, J.E., ' A Framework for Distributed Data Processing Requirements', IEEE COMPSAC, 1979.

c). DISTRIBUTED APPLICATION SYSTEM DESIGN (ALLOCATION PROBLEMS)

BARB82c). Barbara, D. & H. Garcia Molina; 'How expensive is Data Replication', IEEE 3rd Intl. Conf. on Distrib. Comp. Systems, Florida, Oct. 1982, pp.263-268

BRYA81c). Bryand, R.M., and Agre,J.R., 'A Queuing Network Approach to the Module Allocation Problem in Distributed Systems', ACM Conf. on Measurement and Modelling of Computer Systems, Sept. 1981, pp.181-190.

BUCC79c). Bucci,G., Streeter,D.N.,: 'A Methodology for the Design of Distributed Information Systems', CACM,v.22, No.4, April 1979, pp.233-245.

BUCK79c). Buckles, B.P. and Hardin, D.M., 'Partitioning and Allocation of Logical Resources in a Distributed Computing Environment', General Research Corporation Report, Huntsville, Alabama, 1979.

CASE72c). Casey, R.G., 'Allocation of Copies of a File in an Information Network', SJCC 1972, AFIPS Press, Vol. 40, 1972

CASE73c). Casey, R.G., 'Design of Tree Networks for Distributed Data', AFIPS Conf. Proceedings, Vol. 42, 1973, pp.251-257.

CHAND76c). Chandy, D.M. and J.E. Hewes, 'File Allocation in Distributed Systems', Proc. of the Intl Symp on Computer Performance Modelling, Measurement and Evaluation, March 1976, pp.10-13.

CHANG76c). Chang,S.K., 'A Model of Distributed Computer System Design', IEEE Tran. on Systems, Man, and Cybernetics, Vol.5, No.6, May 1976, pp.344-359.

CHANG81c). Chang, S.K. and Liu, A.C., 'A Database File Allocation Problem', COMPSAC, 1981, pp.18-22.

CHEN83c). Chen, David, 'Performance of Concurrency Control and Data Allocation in Distributed Database Systems', Ph.D. Dissertation, The University of Michigan, 1983.

CHU69c). Chu, W.W., 'Optimal file allocation in a multiple computer system', IEEE Tran. on Computers, Oct. 1969, pp.885-889.

COFF80c). Coffman, E.G., Gelenbe, E, et al, 'Optimization of the Number of Copies in Distributed Databases', Proc. of the 7th IFIP Symposium on Computer Performance Modelling, Measurement & Evaluation , May 1980, pp.257-263.

DINK78c). Dinkel, J.J. and Kochenberger, G.A, 'Resource Allocation in Computer Networks by a Regional Access Model', Computers and Operations Research, Vol. 5, pp.139-147, 1978.

DOTY82c). Doty, K.W., McEntyre, P.L. and O'Rielly, J.G., 'Task Allocation in a Distributed Computer System', Proc. of IEEE INFOCOM, 1982, pp.33-38.

ESWA74c). Eswaran, K.P; 'Allocation of Records to Files and Files to Computer Networks', IFIP, 1974, pp.304-307.

FISH80c). Fisher, M.L. and Hochbaum, D.S., 'Database Location in Computer Networks', ACM Journal, Vol.27, No.4, Oct. 1980.

GYLY76c). Gylys, V.B. and Edwards, J.A, 'Optimal Partitioning of Workload for Distributed Systems', COMPCON, Fall 1976.

IRANI81c). Irani, K.B. and Khabbaz, N.G., 'A Combined Communication Network Design and File Allocation for Distributed Databases', 2nd Intl Conf on Distributed Systems, Paris, April 1981.

KHAB80c). Khabbaz, G.N, 'A Combined Network Design and File allocation in Distributed Computer Networks', Ph.D dissertation, Univ. of Michigan, 1979.

KIMB77c). Kimbleton,S.R; 'A Fast Approach To Network Data Assignment', Proc. of 2nd Berkeley Workshop on Distributed Data Management & Computing Networks, May 1977, pp.245-255

LEVIN75c). Levin,K.D; and Morgan,L.H; 'Optimizing Distributed Data Bases- A Framework For Research', Proc. NCC, 1975, Vol.44, pp.473-478

LOOM76c). Loomis,M.S; and Popack,G.J; 'A Model for Database Distribution', IEEE Computing Network Symposium, Gaithusburg, 1976, pp.162-170

MAHM76c). Mahmood, S. and Riordan, J.S.; 'Optimal Allocation of Resources in Distributed Information Networks', ACM Trans. on Database systems, Vol.1, No.1, March 1976, pp.66-78

MARC81c). Marcogliese, R. and Novarese,R., 'Module and Data Allocation Methods in Distributed Systems', 2nd Intl. Conf on Distributed Computing Systems, Paris, April 1981.

MATS82c). Matsushita, Y et al; 'Allocation Schemes of Multiple Copies of Data in Distributed Database Systems', IEEE 3rd Intl. Conf. on Distributed Systems, Florida, Oct. 1982, pp.250-256

McENT81c). McEntyre, P.L. & R.E. Larson, 'Optimal Resource Allocation in Sparse Networks', Proc. of 8th Triennial World Congress of IFAC', Kyoto, Japan, August 1981.

MORG77c). Morgan,H.L; and Levin,K.D; 'Optimal Program and Data Locations in Computer Networks, CACM, Vol.20, No.5, May 1977, pp.345-353

SEGA79c). Segal, A. & Sandell, N.R., 'Dynamic File Assignment in a Computer Network-Part II: Decentralized Control', IEEE Trans. on Automatic Control, Vol. AC-24, No.5, Oct. 1979.

SURI79c). Suri, Rajan, 'A Decentralized Approach to Optimal File Allocation in Computer Networks', Ph.D. dissertation, MIT, 1979.

TABA78c). Tabatabi,V. and Arizoulla, M., 'A Queue Theoretic Model for File Assignment in a Distributed Data Base Network', IEEE Compcon, 78, pp.175-183.

UMAR81c). Umar, A., 'Allocation of Programs/Databases in Distributed Data Processing Environments', Dissertation Proposal, IOE Department, Univ. of Mich, 1981.

WAH84c). Wah, Benjamin; 'File Placement on Distributed Computer Systems', IEEE Computer, January 1984, pp.23-32.

WHIT70c). Whitney,K.M; 'A Study of Optimal File Assignments and Communication Network Configurations', Ph.D. dissertation, U. of Michigan, 1970

d). DDP SUPPORT SYSTEMS
TRANSACTION/QUERY PROCESSING, UPDATE SYNCHRONIZATION, COMMUNICATION SYSTEMS, IMPLEMENTATIONS

ADIB78d). Adiba,M; Caleca,J.Y. and Euzet, 'A distributed Database System Using Logical Relational Machines', 4th Interntl. Conf. on Very Large Data Bases, W. Germany, Sept. 1978, pp.450-461

ALSB76d). Alsberg, P.A et al, 'Multi-Copy Resilience Techniques', Document No. 202, Center for Advanced Computation, Univ. of Illinois, May 1976.

APER83d). Apers, P.G., Hevner, A.R., and S.B. Yao, 'Optimization of Algorithms for Distributed Queries', IEEE Transactions on Software Engg,. Vol. SE-9, NO.1, January 1983, pp.57-68.

APER83d). Apers, P. M., Hevner, A.G. and Yao, S. B., 'Optimization Algorithms for Distributed Queries', IEEE trans. on Software Engg., Vol. SE-9, NO.1, Jan. 1983, pp.57-68.

BACH78d). Bachman,C., 'Distributed Reference Model', SHARE 51, August 1978, Boston, pp.154-160

BACH78d2). Bachman, C. and Canepa,M., 'Session Control Layer of an Open System Interconnection', IEEE Compcon, Fall 1978, Washington, pp.150-156

BACH78d3). Bachman,C., ' Domestic and International Standards Activities for Distributed Systems', IEEE Compcon Fall 1978, Washington, pp.140-143

BAKE77d). Baker,J., 'Distributed Processing: Modeling and Building Blocks', SHARE 49, Houston, Aug. 1977, pp.65-70

BANE78d). Banerjee, J. et al, 'Concepts & Capabilities of a Database Computer', ACM Trans. on Database Systems, Dec. 1978, Reprinted in IEEE tutorial on Centralized & Distrib. Databases

BAUM76d). Baum, R.I. & Hsau,D.K, 'Database Computers-A step towards Data Utilities', IEEE Tran. on Computers, Dec. 1976.

BHAR82d). Bhargava, B., 'Performance Evaluation of the Optimistic Approach to Distributed Database Systems and Its Comparison to Locking', 3rd Intl. Conference on Distributed Systems, Florida, Oct. 1982, pp.508-517

BERN78d). Bernstein, P.A. et al, 'The Concurrency Control Mechanism of SDD1: A system for Distributed Database', IEEE Tran. on Software Engg., Vol. SE-4, NO.3, May 1978.

BERN81d1). Bernstein, P.A. and Chiu, D.W, 'Using Semi-joins to solve Relational Queries', J. Assn. of Computing Mach., Vol.28, Jan. 1981

BERN81d2). Bernstein, P. A. and Goodman, N., 'Concurrency Control in Distributed Database Systems', ACM Computing Surveys, Vol.13, No.2, June 1981, pp.185-222.

BERN81d3). Bernstein, P.A. et al., 'Query Processing in a System for Distributed Databases - SDD1', ACM Trans. Database Systems, Vol.6, Dec. 1981.

BERR79d). Berra, P.B.& Oliver,E., 'The Role of Associative Processors in Database Machine Architectures', IEEE Computer, 1979, Reprinted in IEEE tutorial on Centralized & Distrib. Databases

BJOR73d). Bjork,C.A., 'Recovery Semantics for DB/DC systems', ACM National Conf., 1973. pp.140-146

BRET79d). Bretweiser, H. & U.Kersten, 'Transaction & Catalog Management of Distributed File Mgmt System DISCO', VLDB conf., Brazil, Oct. 1979, pp.340-359

CHAMP77d). Champine, G.A., 'Six Approaches to Distributed Data Bases', Datamation, Vol.23,No.5, May 1977, pp.69-72

CHIU80d). Chiu, D. and Ho., Y., 'A Methodology for Interpreting Tree Queries into Optimal-Semijoin Expressions', Proc. ACM SIGMOD Conf. Santa Monica, CA, 1980.

COMP78d). Computerworld, 'Burroghs Unveils DDP Architecture', Oct. 16, 1978

CHU77d). Chu, W.W, 'Performance of File Directory Systems in a Star and Distributed Network', AFIP Conference Proceeding, Vol.45, 1976, Reprinted in IEEE Tutorial on Centralized & Distributed Databases.

CHU79d). Chu, W.W., and Hurley, P., 'A Model for Optimal Query Processing for Distributed Databases', IEEE Compcon, Spring 1979, pp.116-122.

CYPS78d). Cypser,R.J., 'Communications Architectures for Distributed Systems', Addison-Wesley, 1978.

DATE79d). Date, C.J., 'Locking & Recovery in a Shared Database System: An Application Programming Tutorial', VLDB Conf. Brazil, 1979, pp.1-10

DAVI73d). Davies, C.T., 'Recovery Semantics for a DB/DC System', ACM National Conf., 1973, pp.140-146.

DECN78d). DECNET, Digital Equipment Corporation Document, 1978.

DESJ80d). Des Jardinis, R.P, 'Reference Model for Distributed Systems', COMPCON, Fall 1980.

DESJ78d). Des Jardinis, R.R. and White, G.; 'ANSI Reference Model for Distributed Systems', IEEE Compcon Fall 78, Washington, pp.144-159

HONE78d). 'Distributed Data Processing Technology', Vol. 15W6, Honeywell Research Center, Minneapolis, 1978.

ELLI77d). Ellis, C.A., 'A Robust Algorithm for Updating Data in Duplicate Databases', Proc. 2nd Berkeley Workshop on Distributed Data Management & Computer Networks, 1977. Reprinted in IEEE tutorial on Centralized & Distributed Databases.

EPST78d). Epstein, R., Stonebraker, M. and Wong. E., 'Distributed Query Processing in a Relational Database System', Proc. ACM-SIGMOD, May 1978, pp.169-180.

ESWA76d). Eswaran, K.P; Gray,J.N; et al 'Notions of Consistency & Predicate Locks in a Database System', CACM, Vol.19, No.11, Nov. 1976

ESTR74d). Estrin,G., 'Computer Technology-State of the Art', Proc. of 2nd Jerusulam Conf. on Information Technology, Jerusulam, Aug. 1974, Vol.1, pp.37-63

EVEN79d). Evens, M.W., 'Computers for Distributed Systems', IEEE COMPSAC, 1979, pp.436-440

FARB74d). Farber, D.J., 'Considerations in Distributed Architectures', COMPUTER, March 1974, pp.31-35.

FARB78d). Farber,D.J.,'Structure of Distributed Computing System', SHARE 51, August 1978, Boston, pp.1419-1424

FELD79d). Feldman, J.A., 'High Level Programming Languages for Distributed Computing', CACM, June 1979, pp.353-370.

FORS78d). ForsdiFk, H.C. et al, 'Operating Systems for Computer Network s', Computer, Jan. 1978, Reprinted in IEEE tutorial on Distributed Processing.

FOST76d). Foster,J.,'Distributed Processing for Banking', Datamation, Vol.22, No.2, July 1976, pp.54-56

FRAN78d). Frank,H.,'SNA - An Outsider's View', SHARE 51, August 1978, Boston, pp.1415·

FULT76). Fulton, D.L. et al, 'The Design of a Minicomputer Network Operating System', Trends & Applications in Computer Networks, IEEE 1976, Reprinted in IEEE Tutorial on Distributed Processing.

GARC79d). Garcia-Molina, H., 'Performance of Update Synchronization Algorithms For Replicated Data in a Distributed Database', Ph.D. Dissertation, Stanford University, June 1979.

GARC80d2). Garcia-Molina, H., 'Reliability Issues for Completely Replicated Databases', COMPCON Fall 1980.

GARD77d). Gardarin, G.& Binam, J.L., 'An Approach Towards a Virtual Database Protocol for Computer Networks', AICA 77, Italy, Reprinted in IEEE tutorial on Centralized & Distributed Databases.

GARD79d). Gardarin, G. and Chu, W.W., 'A Reliable Distributed Control Algorithm for Updating Replicated Databases', Proceedings of the sixth Data Communication Symposium, pp.42-51, 1979.

GELE78d). Gelenbe, E. & Sevcik, K.,'Analysis of Update Synchronization for Multiple Copy Databases', Proc. of 3rd Berkeley Workshop on Distributed Data mangmt & Computer Networks, 1978. Reprinted in IEEE tutorial on Centralized & Distributed Databases.

GERM80d). Germano, F., 'The Multiple-Schema Architecture of DSEED: A Distributed Codosyl Prototype Systems', COMPCON Fall 1980.

GOUD76d). Gouda,G.M; and Manning,E.G., 'On The Modeling, Analysis and Design of Protocols - A Special Class of Software structures', Proc. of 2nd Intl. Conf. on Software Engg, Oct. 1976, San Francisco, pp.257-262.

GRAY78d). Gray,J., 'Notes on Data Base Operating Systems' Operating Systems, Lecture Notes in Computer Sciences, 60, edited by G.Goos and J.Hartman, Springer Verlarg, N.Y., 1978, pp.394-410

HARE78d). Hare,P., 'Basic Considerations in Network Design', INFOSYSTEMS, No.7, 1978, pp.54-58

HELL77d). Heller, J. and Osterrer,L., 'The Design and Data Model of BNL Archive and Dissemination System', Proceeding of 2nd Berkeley Workshop on Distributed Data Management and Computing Networks', May 1977, pp.68-86

RAHI79d). Rahimi, S.K. and Franta, W.R., 'A Posted Update Approach to Concurrency Control in Distributed Computing Systems', Proc. of the 1st Intl. Conf. on Distrib. Computing Systems, 1979, pp.632-641

RAM79d). Ramirez, R.J. and Santora, N., 'Distributed Control of Updates in Multiple Copy Databases: A Time Optimal Approach', Proceedings of 4th Berkely Workshop on Distrib. Data Managmt and Computing Networks, pp.191-206, 1979.

ROLL82d). Rolland, C. and Vittal, M., 'Transaction Modelling in Distrib. Environments', 3rd Intl. Conf. on Distributed Computer Systems, Oct. 1982, Miami, Florida, pp.2-7.

HEVN78d). Hevner, A.R. & Yao, S.B., 'Query Processing on a distributed database', EHO141-2/78/0000, IEEE, Reprinted in 'Distributed Database Management', IEEE catalog No. EHO 141-2, 1978.

HEVN81d). Hevner, A. R., 'A Survey of Data Allocation and Retrieval Methods for Distributed Sysems', Univ. of Maryland College of Business and Management, working paper MS/S 81-036, Oct. 1981.

HEVN79d). Hevner, A.R. and S.B. Yao, 'Processing in Distributed Database Systems', IEEE Trans. Software Engg., IEEE Transactions on Software Engineering, Vol.SE-5, pp.177-187, May 1979.

HEVN83d). Hevner, A.R., 'Data Allocation and Retrieval in Distributed Systems', in Advances in Database Management, Vol. II, Ed.by P. Fisher, Wiley, 1983.

HOLG78d). Holmgren, J.F., 'Resource Sharing Unix', Compcon, Fall 1978, pp.302-305

IBM75d). IBM Manual, 'System Network Architecture-General Information' Manual no. Ga27-3102-0, 1975

IBM78d). IBM Manual, 'An Introduction to IBM 8100 System', Manual No.Ga27-2875-0, 1978

ISLO79d). Isloor, S.S et al, 'System Recovery in Distributed Databases', IEEE COMPSAC 1979, pp.421-426

KAWA79d). Kawaza, S. et al, 'Two Phase Deadlock Detection Algorithm in Distributed Databases', VLDB Conf., Brazil 1979, pp.360-367

KLEI78d). Kleinrock,L., 'Packet Switching Networks', SHARE 51, August 1978, Boston

KOHL81d). Kohler, W. H., 'A Survey of Techniques for Synchronization and Recovery in Decentralized Computer Systems', ACM Computing Surveys, Vol.13, No.2, June 1981, pp.149-184.

KUNG79d). Kung, H.T. et al , 'On Optimistic Methods for Concurrency Control', Departmnt of Computer Sciences, Carnegie-Mellon Univ., Pittsburgh, October 1979.

LI81d). Li, V.O.C., 'Query Processing in Distributed Databases', NTIS AD-A103-826, University of Southern California, July 1981.

LIEN78d). Lien, Y.E.,& J.H. Yong, 'Design of a Distributed Entity-Relationship Database System', COMPSAC 1978, Reprinted in IEEE Tutorial on Centralized & Distributed Databases.

LIU82d). Liu, A.C. and Chang, S.K., 'Site Selection in Distributed Query Processing', 3rd Intl. Conf. on Distributed Computer Systems, Oct. 1982, Miami, Florida, pp.7-12.

LORI80d). Lorin, H. 'Key Decisions in Distributed Processing', COMPCON Fall 80.

MAGE80d). Mager, P.S. 'Alternative Architectures for Distributed Data Sharing', COMPCON Fall 1980.

MAHM77d). Mahmood,S.A. and Riordon, J.S., 'Design of a Distributed Data Base File Manager for a Minicomputer Network', IEEE COMPSAC, Nov. 1977, pp.822-825

MCQU79d). McQuillan, J.M, 'A Practical View of Computer Communication Protocols: A Tutorial', IEEE Catalog EHO-137-0, 1979.

MENA78d). Menasce, D.A., and Muntz, R.R., 'Locking and Deadlock Detection in Distributed Databases', printed in 'Distributed Database Management', IEEE catalog No. EHO 141-2, 1978.

MENA78d2). Menasce, D.A , 'Centralized and Hierarchical Locking in Distributed Databases', ACM/SIGMOD Intl. Conf. on Management of Data, Austin, Texas, May 1978.

NELS78d). Nelson, D.L. and Gordon, R.L., 'Computer Cells - A Network Architecture for Data Flow Computing', IEEE Compcon, Fall 1978, pp.296-301.

PIAT80d). PiatKrowski, T.F., 'The ISO-ANSI Open System Reference Model: A Proposal for a System Approach', pp.111-124, Computer Networks, June 1980.

PLEI77d). Pleiner,M; Mcgowan,L and Spalding,A., 'A Distributed Data Base Management System For Real Time Applications', Proc. of 2nd Berkeley Workshop on Distributed Data Management and Computing Networks, May 1977, pp.68-86

ROSE78d). Rosenkratz, D. et al, 'Concurrency Control for Distributed Database Systems', Tran. on Database Systems, June 1978. reprinted in IEEE tutorial on Centralized & Distributed Databases

ROTH77d). Rothnie, J.B. and Goodman,N.,'An Overview of Preliminary Design of SDD1: A System For Distributed Data Bases', Proc. of 2nd Berkeley Workshop on Distributed Data Management & Computing Networks, May 1977, pp.39-57

SACC81d). Sacco, G.M., and Yao, S.B., 'Query Optimization in Distributed Database Systems', Univ. of Maryland, College of Business and Management, working paper MS/S 81-029, August 1981.

SACCO82d). Sacco, G.M. and S.B. Yao, 'Query Optimization in a Distributed Database Systems', in Advances in Computers, Vol.21, New York: Academic, 1982.

SCHN79d). Schneider, G.M., 'Computer Network Protocols-A hierarchical Viewpoint', Computer, pp.8-11, Sept., 1979.

SHAP78d). Shapiro, R.M. & Millstein, R.C., 'Failure Recovery in a Distributed Database System', COMPCON Fall 1978.

SILB80d). Silberchatz, A., 'A Survey Note on Programming Languages For Distributed Computing', COMPCON Fall 1980, pp.719-723

SMAL79d). Small, D.L., 'A Distributed Database Architecture for Data Processing in a Dynamic Environment', IEEE Compcon, Spring 1979, pp.123-127.

SMIT80d). Smith, C. et al, 'Aspects of Software Design Analysis: Concurrency & Blocking', Proc. of IFIP symposium on Computer Performance Modelling, Measurement & Evaluation', pp.245-253, May 1980

SMIT80d2). Smith, L.W, 'Distributed Systems Management: Key Issues', COMPCON Fall 1980.

STON77d). Stonebroker,M. and Neuhold,E., 'A Distributed Database Version of INGRES', Proc. of 2nd Berkeley Workshop on Distributed Data Management & Computing Networks, Manual, Ga27-3102-0, 1977

SU78d). Su,S.Y; S.Lupkiewcz; C.J.Lee, D.H.Lo and K.L.Doty, 'Micronet: A Microcomputer Network System for Distributed Databases', 4th Intl Conf. on Very Large Data Bases, W. Germany, Sept. 1978, pp.288-298

SYKE80d). Sykes, D.J., 'The Economics of Distributed Systems', COMPCON Fall 1980.

TOAN80d). Toan, N.G., 'Decentralized Dynamic Query Decomposition for Distributed Database Systems', Proc. ACM Pacific Conf., San Francisco, CA, 1980.

THOM78d). Thomas,R.H., 'A Solution to the Concurrency Problem for Multiple Databases', IEEE COMPCON, Spring 1978, pp.56-63

TING78d). Ting,P.D; and Tsichritz,D.C; 'A Micro-DBMS for a Distributed Data Base', 4th Intl Conf. on Very Large Data Bases, Sept. 1978, W. Germany, pp.462-4673

TOTH78d). Toth,K.C; Mahmood, S.A; Riordon, J.S; and Ostref; 'An Architecture for Distributing Data Bases', 4th Intl. Conf. on Very Large Data Bases, Sept. 1978, W. Germany, pp.288-298

TRIV78d). Trivedi, K.S. and Leech, R., 'Design and Analysis of Distributed Computer Systems', Dep. of Computer Sciences, Duke Univ., Durham, N.C, Feb. 1978.

THUR80d). Thurber, K.J., 'An Assessment of the State of Network Architectures', COMCON Fall, 1980.

WECK76d). Wecker,S. and Conant,G., 'DNA: An Architecture for Heterogeneous Computer Networks', Proc. of 3rd International Conference on Computer Communications, Toronto, Aug. 1976

WECK79d). Wecker, S. 'Computer Network Protocols', IEEE COMPUTER, Vol.12, No.9, Sept. 1979, pp.58-81

WONG77d). Wong, E.,'Retreiving Dispersed Data from SDD1: A System for Distributed Data Bases', Proc. of 2nd Berkeley Workshop on Distributed Data Management and Computing Networks, May 1977, pp.217-235

WOO78d). Woo,J.W; 'Queuing Network Modeling of Computer Communication Networks', ACM Computing Surveys, Vol.10, No.3, Sept. 1978, pp.342-351

YU82d). Yu, C.T. and Lin, Y.C., 'Some Estimation Problems in Distributed Query Processing', 3rd Intl. Conf. on Distributed Computer Systems, Miami, FLA, Oct. 1982, pp.13-21.

YU82d). Yu, C.T. and Y.C. Lin., 'Some Estimation Problems in Distributed Query Processing', 3rd Intl. Conf. on Distributed Computer Systems, Oct. 1982, Miami, Florida, pp.13-21.

e). CONVENTIONAL APPLICATION SYSTEM DESIGN MANAGEMENT, SOFTWARE, DATABASE

ALFO79e). Alford, M.W., 'Software Requirements Engineering Methodology (SREM) at the age of Two', COMPSAC 78. Reprinted in Tutorial on Distributed Systems Design , IEEE Catalog No. EHO 151-1, 1979.

ARON69e). Aron,J.D., 'Estimating resources for large scale programming systems', NATO Science Conference, Rome, Oct. 1969

ARON74e). Aron, J.D, 'The Program Development Process', Part1, Addison Wesley, Philipines, 1974

BAKE75e). Baker,F.T., 'Software Management through Improved Programming Techniques', ACM Conference, OCT. 1975.

BELA76e). Belady and Lehman, 'A Model of Large Program Development', IBM Sys. Journal, Vol.15, No.3, 1976.

BELA77e). Belady and Lehman, 'Large Program Development', Conf. on Reliable Software, April 1977.

BOEH73e). Boehm, B.W.,et all, 'Characteristics of Software Quality', TRW Systems Group, Report No.25201, 6001-RO-00, Dec. 1973.

BOEH74e). Boehm, B.W., 'Cost of Software', paper published in in Practical Strategies for Developing Large Scale Software, edited by Horowitz, Edison Wesley, 1974.

BRAT75e). Bratman,H., and Court,T., 'Software Factory', Computer,Vol.8, No.7, May 1975.

BROO75e). Brooks,F., 'The Mythical Man Month',Addison Wesley,1975

BUBE77e). Bubenko, J., 'Information Modelling & Reliability', Conf. on Reliable Software, INFOTECH, April 1977.

CARL75e). Carlson,D., 'Producing an Organizational Plan for Computerized Infor mation System', Ph.D. Thesis, Univ. of Mich., 1975.

CHANG82e). Chang, W.P. & Teorey, T.J., 'A Method for Database Record Clustering', Conf. on Information Systems, Ann Arbor, Michigan, Dec. 1982.

CHEN77e). Chen, P.P; and Yao,S.B; 'Design and Performance Tools for Data Base Systems', IEEE Conf. on Very Large Data Bases, May 1977, Tokyo, pp.3-18

CHEN77e2). Chen, P.P., 'The Entity-Relationship Model - A Basis for the Enterprise View of Data', AFIP Conf., Vol.46, 1977 NCC

CHEN76e). Chen, P.P., 'The Entity-Relationship Model - Towards A Unified View of Data', Trans. on Database Systems, Vol.1, No.1, March 1976

CLEL74e). Cleland & King, ' Systems Analysis & Project Management', McGraw Hill, 1974.

COUG75e). Cougar,J.D., 'Computer Based Management Information Systems', John Wiley & Sons, 1975.

DALY77e). Daly, E.B, 'Management of Software Development',IEEE Trans. on Software Engg., May 1977.

DONO76e). Donovan,J.J, 'Database System Approach to Management Decision Support', ACM Tran. on Database Systems, December 1976.

DATE75e). Date, C.J., 'An Introduction to Database Systems' Addison Wesley, 1975.

ESTE76e). Estell,R.G., 'Software Life Cycle Management', ACM SIG on Install ation Management, Vol.5, No.2, Aug. 1976.

FAGA76e). Fagan,M.E.,'Design & Code Inspections',IBM system Journal, Vol.15, No.3, 1976.

FITS78e). Fitsimmons, A. and Love,T., 'A Review and Evaluation of Software Science', ACM Computing Surveys, March 1978, pp.5-30.

FARE77e). Farentino, A.B. and Mills, H.D., 'State Machines and their Semantics in Software Engineering', IEEE COMPSAC 1977, Nov. 1977, Chicago, pp.242-251.

FREE76e). Freeman,P.: 'The Context of Design', Tutorial on Software Design Techniques, IEEE Catalog No. 76CH145-2 C, Oct. 1976, San Francisco, Ca. pp.2-11.

FREE76e2). Freeman, P. 'The Nature of Design', Tutorial on Software Design Techniques, IEEE Catalog No. 76CH145-2 C, Oct. 1976, San Francisco, Ca. pp.35-63.

FRY77e). Fry, J.P., et al , 'Stored Data Description & Data Translation: A Model & Language', Information Systems, Vol.2, No.3, 1977, pp.95-147

FRY78e). Fry,J.P; and Teorey,T.J; 'Design & Performance Tools for Improving Database Usability and Responsiveness', Univ. of Michigan, Data Base Research Group Technical Report 78DB10, March 1978

FRY79e). Fry,J.P; and Teorey,T.J; 'The logical Record Access Approach to Database Design', Univ. of Michigan, Database Research Group Technical Report DSRG-79-DE16, August 1979.

GANS76e). Gansler,J.G, 'Keynote: Software Management', Symposium on Software Engg., April 1976.

GILD74e). Gildersleeve,T, 'Data Processing Project Management', Van Nostrand, 1974.

GUID73e2). Guide , 'Planning & Design of Information Systems', GUIDE GSD Publication, Chicago 1973.

HAMI76e). Hamilton,M., and Zeldin,S., 'Integerated Software Development System', R & D Technical Report, ECOM-76-0329-F, Nov. 1976.

HAMI76e2). Hamilton,M., and Zeldin,C., 'Higher Order Software-A Methodology for Defining Software' ,IEEE Trans. on Software Engg., March 1976.

HALS77e). Halstead, M.H, 'Elements of Software Science', Elsevier Computer Science Library, N.Y., 1977

HANS73e). Hansen, Per Brinch, 'Concurrent Programming Concepts', ACM Computing Surveys, December 1973, pp.224-250.

HOAR72e). Hoare,C.A.R., 'The Quality of Software', Guest Editorial, Software Practice and Experience, Vol.2, 1972, pp.102-105.

HOLT77e). Holton,J.B., 'Are the New Programming Techniques being used', Datamation, July 1977.

ISDOS77e). ISDOS Working paper 89,' Problem Statement Language'.

ISDOS78e). ISDOS Memo META-1, 'The Meta Facility'.

JONE77e). Jones,T.C.,'Program Quality & Programmer Productivity', IBM Technical Report TR 02.764, Jan. 1977.

JONE77e2). Jones,T.C.,'Optimizing Program Quality & Production', GUIDE, Oct. 1977.

KANT77e). Kanter,J., 'Management Oriented Management Information System', Prentice Hall, 1977, pp.149-160.

KEND77e). Kendal,R.c.,' Management Perspectives on Programs, Programming and Productivity', GUIDE Conference, Oct. 1977.

MCFAR73e). Mcfarlan,et al, 'Information Systems Administration', Holt, Rinehart, and Winston, 1973.

MERT78e). Merten,A.G., 'Computer & Information Systems', Handbook of Operations Research, Vol.2, Van Nostrand, 1978.

MILL76e). Mills,H.D.,'Software Development' IEEE Trans on Software Engg., July 1976.

MIXON76e). Mixon,S.R., 'Handbook of Data Processing, Administration, Operations and Procedures', AMACOM book, 1976.

MOHAN78e). Mohan, C., 'An overview of Recent Data Base Research', Database, Fall 1978, pp.3-24.

NOLAN73e). Nolan,R.L., 'Plight of EDP Manager', Harvard Business Review, (May-June,1973) 143-152.

NUNAM71e). Nunamaker, J.F., 'A methodology for the design and optimization of Information Processing Systems', AFIPS, Conference Proc., Vol.38, 1971, pp.283-293.

PARNA72e). Parnas, D.L, 'On the Criteria to be used in decomposing Systems into modules', CACM, Dec. 1972.

PARNA72e2). Parnas, D.L, 'A recent technique for software module specification with examples', CACM, May 1972.

PRES75e). Presser,Leon, 'Software Management', ACM annual Conference, Oct.1975

POON78e). Poonen,G., 'CLEAR - A Conceptual Language for Entity Relationships', Intntl. Conf. on Management of Data, Italy, Aug. 1978

PUTN78e). Putnam, L.H., 'A General Empirical Solution to the macro software sizing and Estimating Problem', IEEE Tran. on Software Engg., Vol. SE-4, No.4, July 1978, pp.345-360.

PUTN76e). Putnam, L.H.,'A Macro Estimation Methodology for Software Development', COMPCON, Fall 1976, pp.138-143.

RAMA78e). Ramamoorthy, C.V. and H.H.So, 'Software Requirements and Specifications: Status and Perspectives', Tutorial: Software methodology, 1978. Copyright IEEE.

ROBI76e). Robinson,L.,'SPECIAL- A Specification and assertion language', SRI report AD/A-038-25 , 1976.

SAYER71e). Sayers, A.P; editor; 'Operating Systems Survey', Auerbach, 1971.

SCHWA74e). Schwartz,J.L., 'Construction of Software: Problems and Practices', Publ. in Practical Strategies for Developing Large Scale Software, edited by Horowitz, Edison Wesley, 1974.

SENK77e). Senko, M.E., 'Data Structures and Data Accessing in Data Base Systems past, present, future', IBM sys. Journ., Vol.16, No.3 1977, pp.208-257.

STEEL77e). Steel,T.R., 'Development of very Large Programs', IFIP, Washington, 1965.

STRAS73e). Strassman, P.A., 'Managing the Evolution to Advanced Information System', Printed in Mcfarlan et. al.

TEICH74e). Teichroew, D., 'Improvements in System Life Cycle', IFIP Proc., 1974, pp.972-978.

TEICH77e). Teichroew,D., 'Computer Aided Software Development', Conf. on Reliable Software, April 1977.

TEICH70e). Teichroew, D., 'Problem Statement Languages in MIS', Proc. sysmposium on BIFOA, 'Management Information Systems-A challenge to Scientific Research', Cologne, July 1970, pp.253-270.

TEOR80e). Teorey, T.J. & Fry, J.P., 'The Logical Record Access Approach to Database Design', ACM Computing Surveys, 12,2 (June 1980), pp.179-211.

TEOR82e). Teorey, T.J. & Fry, J.P., 'Design of Database Structures', Prentice Hall, 1982.

TSCH78e). Tschritzis, D. and Klug, A. , 'The ANSI/X3/SPARC DBMS Framework', SHARE 51, Boston, Aug. 1978, pp.320-332.

WALS77e). Waltson & Felix, 'A Method of programming measurement and estimation', IBM System Journal, Vol.16, No.1, 1977.

WEIL72e). Weil,J.,'Industrial Dynamics & MIS', SMIS conf., 1972.

YAO75e). Yao, S.B., and A.G.Merten, 'Selection of File Organization Using an Analytic Model', Proc. of 1st intl. conf. on very large databases, ACM, N.Y., 1975, pp.255-267

YOUR75e).Yourdon, E., 'Techniques of Program Structure and Design', Englewood Cliffs, Prentice Hall, 1975.

f). MISCELANEOUS TOOLS, TECHNIQUES

BASK75f). Baskett, F. et al, 'Open, Closed and Mixed Networks with different Classes of customers', J. ACM 22, 2 April 1975, pp.248-260

BUZEN76f). Buzen, J.P, 'Operational Analysis: The Key to the new Generation of Performace Prediction Tools', IEEE Compcon, Fall, 1976

BUZEN76f2). Buzen, J.P., 'Fundamental Operational Laws of Computer System Performance', ACT. Informatica, Vol.7, 1976, pp.167-182

BUZEN78f). Buzen, J.P., 'Operational Analysis: An Alternative to Stochastic Modelling', Conf. on Performance of Computer Installations, D.Ferrari (editor), CILEA, North Holland Publising Company, 1978.

CHAND75f). Chandy, K.M., Herzog, U; and Woo, L., 'Approximate Analysis of General Queuing Networks', IBM J. Res. Dev., Vol. 19, No. 1, Jan. 1975, pp. 43-49.

CHAND78f). Chandy, K.M. and Sauer, C.M., 'Approximate Methods for Analyzing Queuing Network Models of Computing Systems', ACM Computing Surveys, Vol.10, No.3, Sept. 1978, pp.281-318.

COOK81f). Cook, Stephen. A., 'Towards a Complexity Theory of Synchronous Parallel Computation'. L'Enseignement Mathematique XXVII - 1981, pp. 99-124.

COOK83f). Cook, Stephen. A., 'An Overview of Computational Complexity', Turing Award Lecture, CACM, Vo. 26, No. 6, June 1983, pp. 400-408

COUR75f). Courtois, P.J., 'Decomposability, instabilities, and saturation in multiprtogramming systems', CACM, vol. 18, no. 7, July 1975, pp. 371-377.

DEMP80f). Dempster, M.A.H.; ' Introduction to Stochastic Programming', Academic Press, 1980.

DEMP81f). Dempster, M.A.H., Fisher, M.L. et al, 'Analytical Evaluation of Hierarchical Planning Systems', Operations Research, Vol.29, No.4, July-August 1981, pp.707-717.

DENN78f). Denning,P.J. and Buzen,J.P., 'The Operational Analysis of Queuing Network Models', ACM Computing Surveys, Vol.10, No.3, Sept. 1978, pp.225-261

DURA74f). Duran, B.S. and Ordell, P.C., 'Cluster Analysis - A Survey', Lecture notes in Economics & Mathematical Sys., Springer-Verlag,1974

EVER63f). Everett, H. 'Generalized Langrange Multiplier Method for Solving Problems Optimum Resource Allocation', Operations Research, 11, pp.399-417 (1963).

EVER74f). Everret, B., 'Cluster Analysis', Heinemann Educational Books, LTD. London, 1974.

FISH80f). Fisher, M.L. and Hochbaum, D.S., 'Probabilistic Analysis of the Planar K-Median Problem', Math. of Operations Research, Vol. 5, No. 1, pp.27-34.

FISH80f). Fisher, M.L., 'Worst-Case Analysis of Heuristic Algorithms', Management Science, Vol.26, No.1, January 1980, pp.1-17.

GARE79f). Garey, M.R. & Johnson, D.S.; 'Computers and Intractability: A Guide to the Theory of NP-Completeness', Freeman & Co., 1979.

HOLM81f). Holmes, R.B. and J.W. Tolleson, 'Practical Aspects of Nonlinear Programming', MIT Lincoln Laboratory Technical report 576, June 1981.

JACK63f). Jackson,J.R. 'Jobshop Like Queuing Systems', Management Sciences, 10, 1(1963), pp.131-142

JOHN78f). Johnson, E.L., 'Flows in Networks', Published in Handbook of Operations Research, edited by Moder & Elmagharby, Van Nostrand Reinhardt, 1978.

JENS69f), Jensen, R.E., "A dynamic Programming Algorithm for Cluster Analysis", Op. Res., 12, Nov. 1969, pp.622-626.

KARP76f). Karp, R.M., 'The Probabilistic Analysis of Some Combinatorial Search Algorithms', Proc. Symposium on New Directions and Recent Results in Algorithms and Complexity, Academic Press, New YOrk, 1976.

KARP77f). Karp, R.M., 'Probabilistic Analysis of Partitioning Algorithms for the Travelling Salesman Problem in the Plane', Math. Operations Research., Vol. 2, 1977, pp.209-224.

KOBA78f). Kobayishi,H; 'System Design and Performance Analysis Using Analytic Models', Current Trends in Programming Methodology, Vol. 3, Prentice Hall, Englewood Cliffs, N.J., 1978, pp.72-114

LARS79f). Larson, R.E, 'Distributed Control: Tutorial', IEEE Catalog No. EHO-153-7, 1979.

LARS82f). Larson, R.E et al, 'Distributed Control', IEEE Tutorial, IEEE Catalog No. EHO 199-0, October 1982.

LAVEN76f). Lavenberg,S.B; and Schedler,G.S; 'Stochastic Modeling of Processor Scheduling with Application to Data Base Management Systems', IBM Journal of R & D, Sept. 1976

LEWIS76f). Lewis, L.J., 'Servive Levels: A Concept for the user and the computer center', IBM System Journ., Vol.15, No.4, 1976, pp.328-357

MODER78f). Moder, J.J. and Elmaghary, S.H., 'Handbook of Operations Research', Van Nostrand Reinhardt, 1978

OHSU79f). Ohsumi, N., 'Evaluation Procedure of Agglomerative Hierarchical Clustering Methods by Fuzzy Relations', 2nd Intl. Symp. on Data Analysis & Informatics, Versailes, Oct. 1979, pp.509-522.

NEMH66f). Nemhauser, G.L, 'Introduction to Dynamic Programming', John Wiley & Sons, N.Y., 1966

PETER77). Peterson, J.L; ' Petrinets', ACM Computing Surveys, 1977

PLAN71f). Plane, D.P. and McMillan,C., 'Discrete Optimization', Prentice Hall, 1971

RAO71f). Rao, M.R., 'Cluster Analysis & Mathematical Programming', Journ of American Statistcs Ass., Vol.66, 1971, pp.622-626.

REIS78f). Reiser,M; and Saure,C.H; 'Queuing Network Models: Methods of Solution and Program Implementations', Current Trends in Program Methodology, Vol.3, Englewood Cliffs, N.J., 1978, pp.115-167

REIS76f).Reiser, M.,'Interactive Modelling of Computer Systems' IBM sys. Journ., Vol.15, No.4, 1976, pp.309-327

ROSE83f). Rosenkratz, D.J. and Stearns, R.E., 'NP-Complete Problems', Encyclopedia of Computer Science and Engineering', 2nd edition, Edited by Ralston, Van Nostrand Reinhold, 1983, pp.1026-1029.

SCHER67f). Scherr,A; 'Analysis of Time Shared Computer Systems', MIT Press, Cambridge, Mass, 1967.

TIVED78f). Trivedi, K.S., 'Analytic Modeling of Computer Systems', COMPUTER, 1978, IEEE.

TEOR75f). Teorey, T.J., 'Validation Criterian for Computer System Simulation', ACM Simuletter/VI14.

TRAUB83f). Traub, J.F., 'Computational Complexity', Encyclopedia of Computer Science and Engineering', 2nd edition, Edited by Ralston, Van Nostrand Reinhold, 1983, pp.258-261.

VINO69f). Vinod, H.D., 'Integer Programming & Theory of Grouping', JASA, June 1969, pp.506-519.

WANG78f). Wang, P.P. and Chang, S.K., 'Fuzzy Sets - Theory and Applications', Plenum Books, 1978.

YAO76f). Yao,S.B; Das,K; and Teorey,T.J; 'A Database Reorgination Algorithm', ACM Trans. on Data Base Systems, Vol.1, No.2, June 1976.

ZADE71f). Zadeh, L.A., 'Quantitative Fuzzy Semantics', Information Sciences 3 (1971), pp.159-176.

ZADE71f2). Zadeh, L.A., 'Similarity Relations and Fuzzy Orderings', Information Sciences 3 (1971), pp.177-200.