ABSTRACT

A Distributed Database Management System

for Homogeneous Multicomputer Architectures

John Symeonidis

This thesis presents a model of a database management system suitable for distribution over a multicomputer system. The model is designed for architectures consisting of a set of similar computers which may be situated in one location or may be geographically dispersed. It is presented in the context of a comprehensive review of database architecture, covering software, specialized hardware, dedicated machines, and distributed systems.

ACKNOWLEDGEMENTS

I wish to express my sincere gratitude to my thesis advisor Dr. Terrill Fancott, for his expert guidance throughout the course of this research. He has always been accessible for discussions, suggestions and has played a major role in completing this thesis.

I am indebted to my wife, Fotini and my daughter Angela for their love and support during my long study hours throughout the course of this research.

I am thankful to my job superiors D. McDonough and R. J. McCreath for the financial support and the various conveniencies they have provided to me during my graduate studies.

I dedicate this thesis in memory of my beloved father Symeon Symeonidis.

TABLE OF CONTENTS

LIST OF FIGURES

0. INTRODUCTION

The object of this thesis is to develop a model of a database suitable for distribution over a multicomputer system, which may be local or geographically distributed. The model presumes the existence of a multicomputer interprocess communications system such as the one proposed by (Fancot81), (Lenah80), which is transparent to the processes resident on the different computers. While this aspect of system architecture relieves the model of the need for inter computer communication management, the distribution of the database itself over a number of nodes becomes the fundamental design parameter. This thesis addresses the problem on the level of system organization. Particular emphasis is made on the effects that distribution will have on the system design, file design and functions of the database command set. The problems of synchronization and deadlock have been studied elsewere, and are not addressed directly in this work. Reference is made, however, to contemporary research results which suggest solutions to these problems.

The logical design is presented in the context of an extensive review of work reported in database design, covering both single computer software as well as specialized systems such as database computers and back-end networks. This survey is reported in the first two parts of

this thesis. Part 1, including chapters 1 to 4, discusses aspects of conventional database design, including access methods and the different models of database architectures. While some parts of this review, such as access methods, are not used directly in the proposed model, their inclusion in the review provides a background of the software levels which support level described by the model structure. Part 2 reviews the different methods of distributing the tasks of database management. This section covers both the logical design and the principal implementations of database processors and networks, giving the technological context of the proposed model. Part 3 presents the logical design of a distributed database system.

The proposed model is designed for a system of similar, if not identical computers, rather than a set of specialized processors. As such, it is appropriate for architectures represented by a set of computers running similar database tasks as part of a large, distributed system, or, in the case of a local system, an array of identical minicomputers, each controlling part of the database. The cases of both partitioned and replicated databases are considered in this design.

1. BACKGROUND

The concept of generalized database management systems was first introduced in the early fifties, but the greater part of development has taken place in the past decade.

The database concept provides for a unique 'data storage pool' where all information records are stored together, as opposed to conventional data processing, where programs 'own' their data which are not available to other systems. The database concept is still young, and a variety of definitions exist to describe it. These non identical, but complementary definitions are as follows:

> "A Database is defined as a collection of interrelated data stored together with controlled redundancy to serve one or more applications in an optimal fashion." (Marti76)
>
> "A Database is a collection of physical records that are similarly defined and serve a single general application purpose." (Ross78)
>
> "A Database is a non redundant collection of data elements stored in logically related files." (Hp1000)

The common point of the above definitions is that a database system contains one or more files whose records are interrelated by relationships defined by the user. This technique allows the user to maintain his information under

one software package with a number of consequent advantages. These include:

a. Independence of data between application programs and file storage (Ross78),(Kroen78)
b. Controlled data redundancy
c. Standardization of data definition and documentation
d. Data security

Every system has, however, its weak points, and the following disadvantages of database technology have been noted:

a. More main memory is required
b. The CPU may be monopolized, forcing the user to upgrade more powerful computer (kroen78)
c. Backup and recovery are more elaborate
d. Integration and hence centralization increases vulnerability. A failure in one component of an integrated system can stop the entire system (Kroen78)
e. Multiuser systems are prone to deadlocks

Evolution of database management systems

Since the early fifties, a number of database families have been developed. The the most significant ones are the

following:

1. Hanford/RPG
2. MITRE/ Auerbach
3. Postley/MARK IV
4. Bachman/IDS
5. Formatted file/GIS
6. CODASYL
7. IMS
8. Inverted file

Hanford/RPG family

The RPG system was developed at the Hanford operations of the atomic energy commission which was then managed by the General Electric company. The first report generator (MARK I) was developed as a generalized sort routine for the IBM 702. The package was upgraded with the development of a report generator with file maintenance (MARK II). These routines provided the basis for the development of 709 package (9PAC) for the IBM 709/90. The 9PAC is the ancestor of most commercial report generators. RPG was developed for the IBM 1401 in 1961.

MITRE/Auerbach family

The US Air force sponsored DBMS development at the MITRE corporation. The prototype was called Experimental Transport Facility (ETF). It later became the Advanced Data Management System (ADAM), and was implemented in an IBM 7030

computer. ADAM did not implement all the features of the DBMS but its contribution to the technology was significant.

Postley/MARK IV family

The ancestor of the MARK IV family is the GIRLS (Generalized Information Retrieval and Listing System), developed for the IBM 7090 by Postley. Since MARK IV was first released for the IBM/360 in 1968, numerous subsequent releases have provided with more features, some of which have been implemented on the hardware level.

Bachman/IDS family

the Integrated Data Store (IDS) was developed by Bachman and his team at the General Electric company in 1964. The IDS system stems from the 9PAC package of the Hanford/RPG family. The IDS included random access storage technology and high level languages, thus providing a powerful network model. Since 1964 the IDS system has been used with different hardware systems, operating systems and host languages.

Formatted file/GIS family

The Information Retrieval (IR) experimental prototype developed in 1958 for the IBM 704 appears to be the ancestor of the family. The main sponsors for the development of this family were the US Air Force and the Navy. The Strategic Air Command (SAC) system was developed by the Air

Force branch, while the Information Processing System (IPS) was developed by the Navy branch. IPS contributed to database technology through the implementation of a multilevel hierarchical structured database on sequential media.

CODASYL family

Based on the IDS ideas, the CODASYL language committee started to work on a proposal for extending the COBOL to handle databases. This group, originally called the list processing task group, later became the Data Base Task Group (DBTG). In 1961 the DBTG issued recommendations for the syntax of a Data Description Language (DDL) in order to describe network structured databases, and to define a Data Manipulation Language (DML). User recommendations for improving the package led to major changes in 1971. It was decided to split the data description in two parts, a schema for defining the total database and a subschema for defining each of the various views of the database for different programming languages and applications.

IMS family

The IMS family is an outgrowth of the Apollo moon landing program. It originates from two developments at NASA. One is the Generalized Update Access Method (GUAM) and the other is the implementation of two teleprocessing applications EDICT (Engineering Document Information Collection Task) and LIMS (Logistics Inventory Management System). The software

package which supported the EDICT and LIMS, the Remote Access Terminal System (RATS), was developed by Rockwell International and IBM for the IBM 7010 computer with 1301 disk storage. In 1966, IBM, Caterpillar tractor corporation, and Rockwell International agreed to jointly develop a DBMS: the Information Management System (IMS) for the IBM/360.

Inverted file

LUCID, the ancestor of inverted files, was followed by the Time-shared Data Management System (TDMS), developed by the Advanced Research Project Agency (ARPA) of the System Development corporation. The system was designed to operate in a time-sharing environment on an IBM/360. It was the first DBMS to combine an inverted file implementation of the hierarchical data model with interactive processing.

2. DATABASE CONCEPTS AND COMPONENTS

Database technology has introduced a number of new terms into the vocabulary of computer science. Terminology specific to databases include well known concepts such as data item (field), data entry (record) and data set (file). Concepts which are specific to databases include schema, and subschema. The schema is a chart of the types of data used (Marti76) which specifies the names, attributes and relations between the data entries. A subschema is a user view of a subset of the schema.

Database software can be seen as two major software layers, the Data Description Language (DDL), and the Data Manipulation Language (DML). These two process the schema as follows.

DDL: The data description language describes the structure of the database application to the database software. The term refers to the software which specifies the structure of data items and the relations that exists between them. DDL itself can be subdivided into two layers. One software layer describes the structure of data as far as its physical layout is concerned, and specifies the allocation of resources, buffering, paging, addressing and searching techniques, dataset size etc. This physical description language is also known as Device Media Control Language (DMCL) in CODASYL and Physical Data Base Description (PDBD) in IBM's DL/1

(Marti76).

A second software layer describes the logical records in a manner independent of software and hardware. This layer corresponds to the view of the schema which is different from the physical organization of the data. In general the DDL can be seen as the data division in a COBOL program.

DML: The data manipulation language provides instructions that the application program can give to the database management system, and interprets status messages reporting results of the request. For this purpose the database management system provides a set of macroinstructions or callable subroutines for the application programmer. Further details on the DDL and the DML are given in Appendix 1.

3. ACCESS METHODS

Database records are stored in files along with information on record and file relationships. Although the physical organization of records appears to be simple (recorded in chronological sequence), retrieving those records according to some search value (key) or relationship, is a complex task.

In general, lists are employed to keep track of a set of relative records. In certain systems (Hp3000) the records of the detail data sets are organized as a double linked list structure. The address of the head of the list is located in the master data set, which itself must be accessed.

The complexity of a master data set is a function of the storage organization. Sequential searching is generally avoided because of its low efficiency. The different methods of physical organization and search are discussed in the following:

3.1 Hashing methods

One of the methods used to locate a key in a table or file is called the "hashing or scatter storage technique" (Knuthv.3) or "KAT- key to address transformation" (Ghosh79) where the location of the key is derived from its value after an algebraic or logical computation, and the computed address is called hash or home address (Hill78). The keys k1, k2 are said to be synonyms when h(k1)=h(k2) and their

occurrence is called a collision. Depending on the file organization, the synonym key might be forced to occupy a location other than the calculated one. When a key hashes to a location which is occupied by another non synonym key, the second one might be forced to move to another location. Such keys are referred to as "migrating secondaries" (Hp3000). It is difficult to find a function that does not give any synonyms, but is desirable to have as few synonyms as possible so that the housekeeping routines will keep their process time to minimum and avoid frequent table or file reorganizations.

Hashing functions

several types of hash functions are in use. The principal ones are the following:

DIVISION:

The division method is very simple and uses the remainder modulo (mod) operator. The function is defined as h(K)=KmodM. . The identifier K is divided by a number M and the remainder is used as the hash address of K. This function generates addresses in the range 0 to (M-1) and the hash table is at least of size b=M (b= table size) (Horro77). The choice of M is very critical. If M is an even number, h(K) will be even when K is even and odd when K is odd. If M is a power of the radix of the computer, the generated address will depend on the least significant digits of K. The situation will be worse if the identifiers

are stored left justified zero filled. These difficulties can be avoided by choosing M as a prime number. In practice, it has been observed that it is sufficient to choose M such that it has no prime number divisors less than 20 (Horro77).

MID-SQUARE

The h(K) is computed by squaring the identifier and then taking a certain number of bits from the middle of the square to obtain the bucket address. The middle bits usually depend upon all the characters in the identifier, and are more likely to generate different hash addresses. The identifier is assumed to fit into one computer word.

FOLDING

In this method the identifier K is partitioned into several parts, all but the last being of the same length. The parts then are added together to obtain the hash address. There are two ways to perform this addition (Horro77)

1. All but the last part are shifted so that the least significant bit of each part lines up with the corresponding bit of the last part (Figure 3.1).
2. The identifier is folded at the part boundaries and digits falling into the same position are added together (Figure3.2).

This method is implemented at the hardware level of some recent HONEYWELL large scale computers.

MULTIPLICATIVE

For P1= 123, P2= 203, P3= 241, P4= 112, P5= 20

P1= 123 +
p2= 203 +
P3= 241 +
P4= 112 +
P5= 20 +

h(K)= 699

Figure 3.1 Folding method.

P1= 123 +
P2= 302 +
P3= 241 +
P4= 211 +
P5= 20 +

h(K)= 897

Figure 3.2 Folding method

In this scheme the word size of the computer is used in calculations. In a computer with word size w, an integer A can be considered as a fraction A/w if the radix point is assumed to be at the left of the word. The method is to choose an integer constant A relatively prime to W and to have h(K)=[M((A/W*K)MOD 1)] (Knuthv.3).

In this case we usually choose M to be a power of 2 on a binary number so that h(K) consists of the leading bits of the least significant half of the product A*K. This method can be seen as generalization of the division method. Since it is possible to take A to be approximately W/M, multiplying by the reciprocal of a constant is often faster than dividing by that constant. In this method, h(K) takes values in the range 0 to W-1, so it is not very good as a hashing function, but it can be useful as a scrambling function. A feature of the multiplicative method is that it makes good use of nonrandomness. For names such as PART1, PART2, PART3, LINEA, LINEB this method will generate an arithmetic progression h(K), h(K+d), h(K+2d)...of distinct hash values, thus reducing the number of collisions. The division method has the same property.

DIVIDE BY POLYNOMIAL MODULO 2

In this method the identifier is divided by a polynomial modulo 2 instead of dividing by an integer (Knuthv.3). Here M is chosen to be power of 2 M=2**m and an mth degree polynomial P(x)=x**m+p(m-1)*x**(m-1)+...p0, is used.

An n-digit binary key can be expressed as a polynomial K(x)=k(m-1)*x(m-1)+...+k1x+k0 and the remainder is computed K(x)mod P(x)=h(m-1)*x(m-1)+....+h1*x+h0. If P(x) is chosen properly, the hash function h(K) can avoid collision between nearly equal keys. For example, if m=10, n=15 and P(x)=x**10+x**8+x**5+x**4+x**2+x+1 for keys K1, K2 different only in fewer than 7 bit positions the h(K1)≠h(K2). It should be noted that this algorithm is more suitable in a hardware or microprogrammed implementation than software.

POLYNOMIAL TRANSFORMATION

Suppose a key K consists of a string of symbols of length m1 and each symbol is represented by q bits. The total number of possible symbols is 2**q. If we assume q=8 (8-bit character) then a key is a binary string of length m1*q. The number of nonzero bits of the key is called the weight of the key. The number of positions in which two keys differ is called the Hamming distance between the two keys (Ghosh77). For example:

Given the keys,

k1=1110001101101111

k2=1010010111110011

then k1w=11 is the weight of k1

k2w=10 is the weight of k2

And the Hamming distance is d(k1,k2)=7

Clusters in the key space are defined as collections of keys

such that the distance between any two number keys is less than d where d is called the diameter of the cluster. The maximum distance of d is denoted by D. To avoid overflows in the file, no cluster should have a diameter greater than D. In this method, the key is represented as a polynomial whose coefficients are the symbols of the key:

$K(x)=\sum_{i=1}^{m_1} di\,(x)^{i-1}$ where d1,d2,d3.....dn are the symbols of the key.

The divisor polynomial is calculated as follows:

$g(x)=(x-a)(x-a^2)....(x-a^{D-1})$ where a is a primitive element, i.e. a**(2q-1)=1, a**i≠1 for i<2**q-1 D is the maximum diameter of a cluster in a key space.

The remainder polynomial is calculated from the division of K(x) by G(x)

$$K(x)=P(x)*G(x)+R(x)$$

The coefficients of R(x) give the address of the key.

Example:

Consider a key with symbols from the field of binary numbers, i.e. q=1, m1=5 where we want to break the key space into clusters of diameter 2 or less. Then D=3, a=1 and G(x)=(x-1)*(x-1)=x**2+1. Suppose the key to transfer is K=10111 then K(x)=x**4+0*x**3+x**2+1. After division the remainder polynomial R(x)=x+1, giving the address location h(K)=(011) binary, or 3 decimal.

RADIX TRANSFORMATION

The storage memory locations are addressed by a number

which is binary or decimal i.e a radix of 2 or 10 (Ghosh77). Let q be the radix of the address. The number of table entries is m. The key is treated as a binary string which in turn is divided into substrings of 3 or 4 binary bits to form digits. The digits obtained are treated as number with radix p. This number is converted to a number with radix q (radix of the address).

Example:

Assume the binary string is grouped in 3-bit digits 111100111111001==> 171771. Assume p=16, q=10: then 171771hex is expressed in radix 9. As p≠10 this number is converted to a number with radix 10 K=171771hex = 1513329 decimal. If m=10**4 then h(K)=Kmod m=(1513329)mod 10**4=3329, giving the address 3329 dec.

3.2 RECORD PROCESSING USING HASHING METHODS

It was observed that keys occasionally hash into the same address. Some methods have been developped to overcome this difficulty. Normally, direct access storage devices are used to store the records in blocks. The name bucket is used for such a block and the bucket size is the maximum number of records that that can be contained in a bucket (Hill78). The bucket can be considered as being subdivided into slots where each slot hods one data record. The load factor of the file is defined as the ratio of the active keys to the total slots available in all buckets.

3.2.1 Chaining with separate lists (Figure 3.2.1.1.)

Collisions may be resolved by maintaining linked lists for all synonyms of each possible hash code (Knuthv.3). In this organization the head of each list is contained in one slot of the bucket.

This method is quite fast, but it has been observed that if the file is too large, many of the lists will be empty. When records are small, however, the record storage can be overlapped with the list heads, thus providing more room (Knuthv.3).

The expected number of identifier comparisons is:

C=1+a/2 (a=load factor)

3.2.2 Chaining with coalescing lists (Figure 3.2.2.1)

In this method a key is hashed to an address and is inserted there if this address is empty, otherwise it is inserted in the next empty location and is linked to the home address. This method allows several lists to coalesce so that the records need not to be moved after they have been inserted into the table (Knuthv.3). The expected number of identifier comparisons is:

C=1+1/8*a(e**2a-1-2a)+(1/4)a (a=loading factor)

3.2.3 Linear probing

This method does not use links. Instead, various entries are inspected until either the key or an empty position is found. The concept is based on the computation of a "probe sequence" for each key to be followed. This general class of methods is called open addressing (Knuthv.3) and the

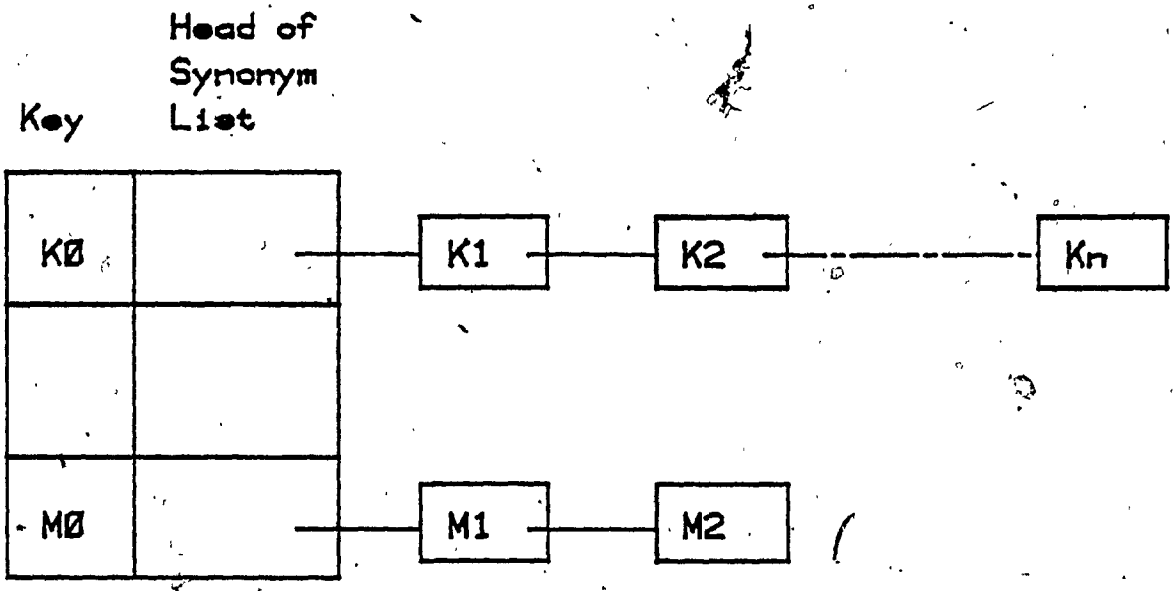


Figure 3.2.1.1 Chaining with separate lists.

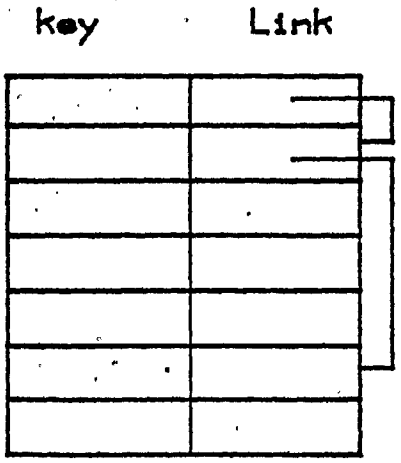


Figure 3.2.2.1 Chaining with coalescing lists.

simplest open addressing scheme is called linear probing. Linear probing uses the cyclic probe sequence h(k), h(k)-1, ...0, b-1, b-2, ... B(k)+1 (b=bucket size). When an open position is encountered while searching for a key using the appropriate probe sequence, it is considered that the key does not exist in the list, since the same sequence of probes is made every time the key is processed (Hill78). The expected number of identifier comparisons is:

$C=1/2+[1/(2-2a)]$

3.3 RECORD PROCESSING USING TREE METHODS

A tree is a special case of a directed graph. It is used as the basis of large proportion of recent work in file organization and is therefore presented in detail in the following. The basic structure of trees is summarized and different types applicable to databases are presented.

A directed graph is defined as a set of nodes N1,N2,N3,...NN and a set of arcs called branches with a specified orientation connecting various pairs of nodes. A sequence of branches such that the terminal node of each branch coincides with the initial node of the succeeding branch is called a path. The number of branches in a path is called the length or height of the tree. A path is a cycle when it is at least of length one and the terminal node coincides with the initial node N1N2.....NN. A tree is a directed graph that has no cycles and at most one branch entering each node (Hill78).

A root of a tree is the node which has no branches entering it.

A leaf or terminal node is a node which has no branches leaving it. The number of branches leaving a node is called the degree or branching ratio of that node.

A balanced tree (Figure 3.3.1) is a tree where the difference between the path lengths from the root to any two leaves is at most one (Hil178).

A binary search tree (Figure 3.3.2) is a directed graph with the following properties (Hil178).

1. There is only one node called Root such that for any node K there exists one and only one path which begins with the Root and ends with K
2. For each node K, the number of links beginning with K is either two or zero.
3. The set of links is partitioned in two sets L and R. Each link belonging to L is called a left link and each belonging to R is called a right link.
4. For each node K having two links there is exactly one left link beginning with K and exactly one beginning with P.

There are two types of nodes in a binary search tree. An internal node is one with at least one link that is not null. A node with a left link and the right null is called an external node. Insertion into a binary search tree is made as the value of the first blank node is found while

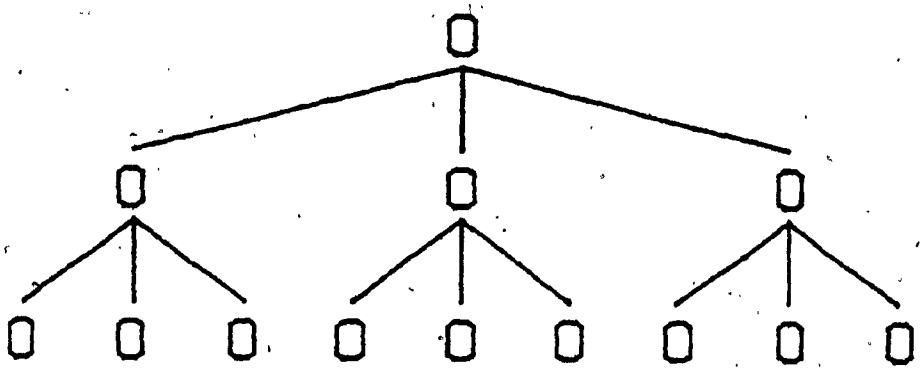


Figure 3.3.1 Balanced tree.

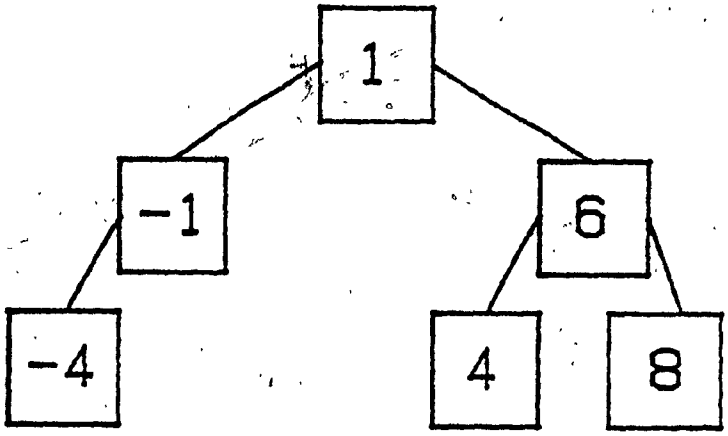


Figure 3.3.2 Binary search tree.

traversing the tree along some path from the root. While traversing the tree, the branches on a path are taken left if the key is less than node value or right if the key is greater than the node value. A blank node is used to indicate the end of a path. The search lengths of a binary tree are :

1. Binary search tree with only right or left links
   $(N+1)/2$
2. Balanced binary tree.
   $\log N$

There are three ways to traverse a binary tree (Knuthv.3).

Preorder traversal: visit the root, traverse the left subtree and then the right subtree.

Inorder traversal: traverse the left subtree, visit the root and traverse the right subtree.

Postorder traversal: traverse the left subtree, traverse the right subtree, and visit the root.

Trie (Figure 3.3.3)

An M-ary tree whose nodes are M-place vectors with components corresponding to digits or characters is called a TRIE (Hill78).

A trie stores records in its leaves or external nodes. Any node on a given level h represents the set of all keys that begin with a certain sequence of h characters. The node indicates an M-ary branch depending on the (h+1)st character. In Figure 3.3.3, Node(1) is the root. If the

NODES = [A, BE, ABC, CAN]

	1	2	3	4	5	6	7	8	9	10
		A								
A	2				6					
B	3	4								
C	5			ABC						
D										
E			BE							
F										
G										
H										
I										
J										
K										
L										
M										
N						CAN				

Figure 3.3.3 A Trie.

first character is 'B' then we are directed to node 3. If the second character is 'E' then we have the key 'BE'

Addressing a trie is analogous to the addressing of multidimensional arrays. It resembles the transition table used in compiler construction.

The organization called a B-tree (Figure 3.3.4) uses pages as nodes of a tree.

Let $h>0$ to be the height of a tree, and let K denote the least numbers of records in any node of the tree.

A B-tree is defined as a tree having the following properties (Hill78).

1. The length of any path from the root to the leaf is h.
2. Each node except the root and the leaves has at least K+1 sons. The root is empty or it has at least two sons.
3. Each node has at most 2K+1 sons.

The nodes (pages) of a B-tree must be allowed to grow to the maximum, after which it is necessary to divide pages in a way that the B-tree will continue to grow. B-trees allow retrieval, insertion and deletion of records in a time proportional to K Log (N) where N is the number of records and k is a device dependent parameter. It is difficult to obtain average performance measures for B-trees (Hill78). A variation of the B-tree is a tree where all data are stored in leaf nodes. Any B-tree in which the non leaf nodes contain only pointers and keys and the data are stored in

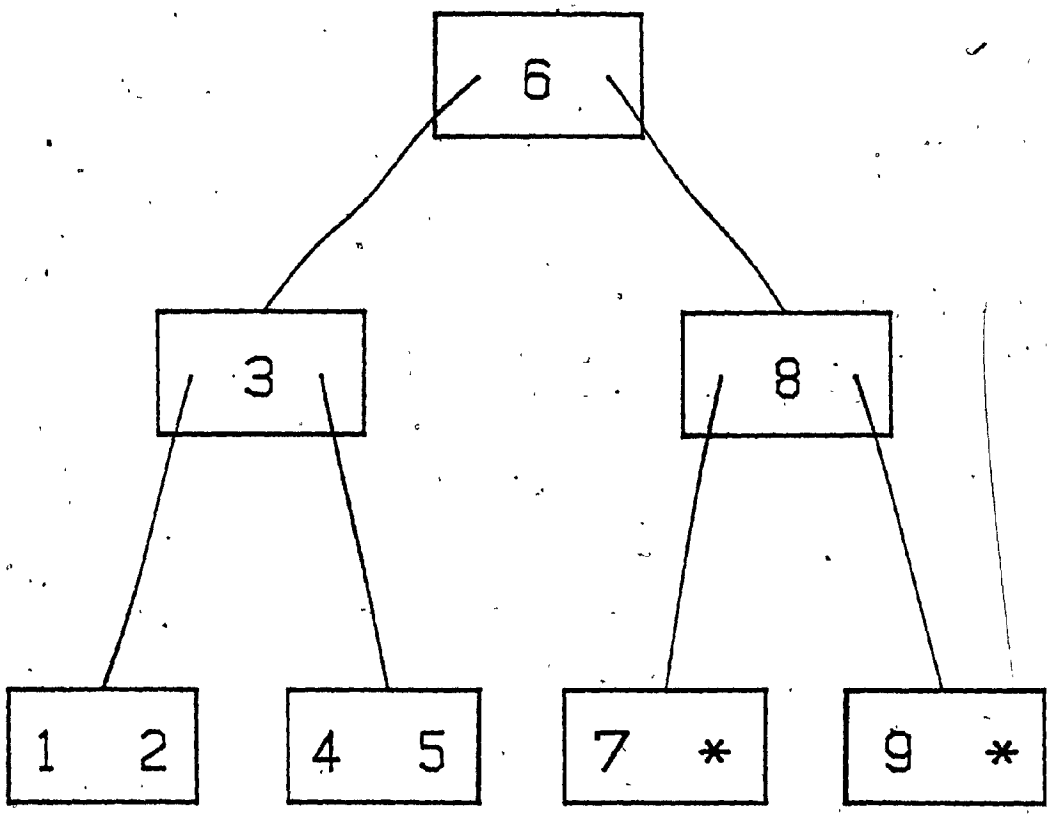


Figure 3.3.4 B-Tree.

the leaves is called B*-tree.

3.4 RECORD PROCESSING USING LINKED FILES

A set of related records are stored in the DASD as linked lists. In this structure, pointers are used to link each record with the next having the same relation (key). Pointers are disk addresses. A directory (Figure 3.4.1) contains each available keyword with its associated pointer. This pointer indicates the beginning of the list. In this scheme, a particular search consists of a directory search to locate the keyword and then a list search. The number of entries in the directory equals the number of keywords and the length of the list is equal the number of records per keyword (Hill78).

Every record corresponds to one node in the list. If a record is characterized by several directory keywords then the record corresponds to a node that is the intersection of the list corresponding to the several keywords. Directory search is done by linear probing.

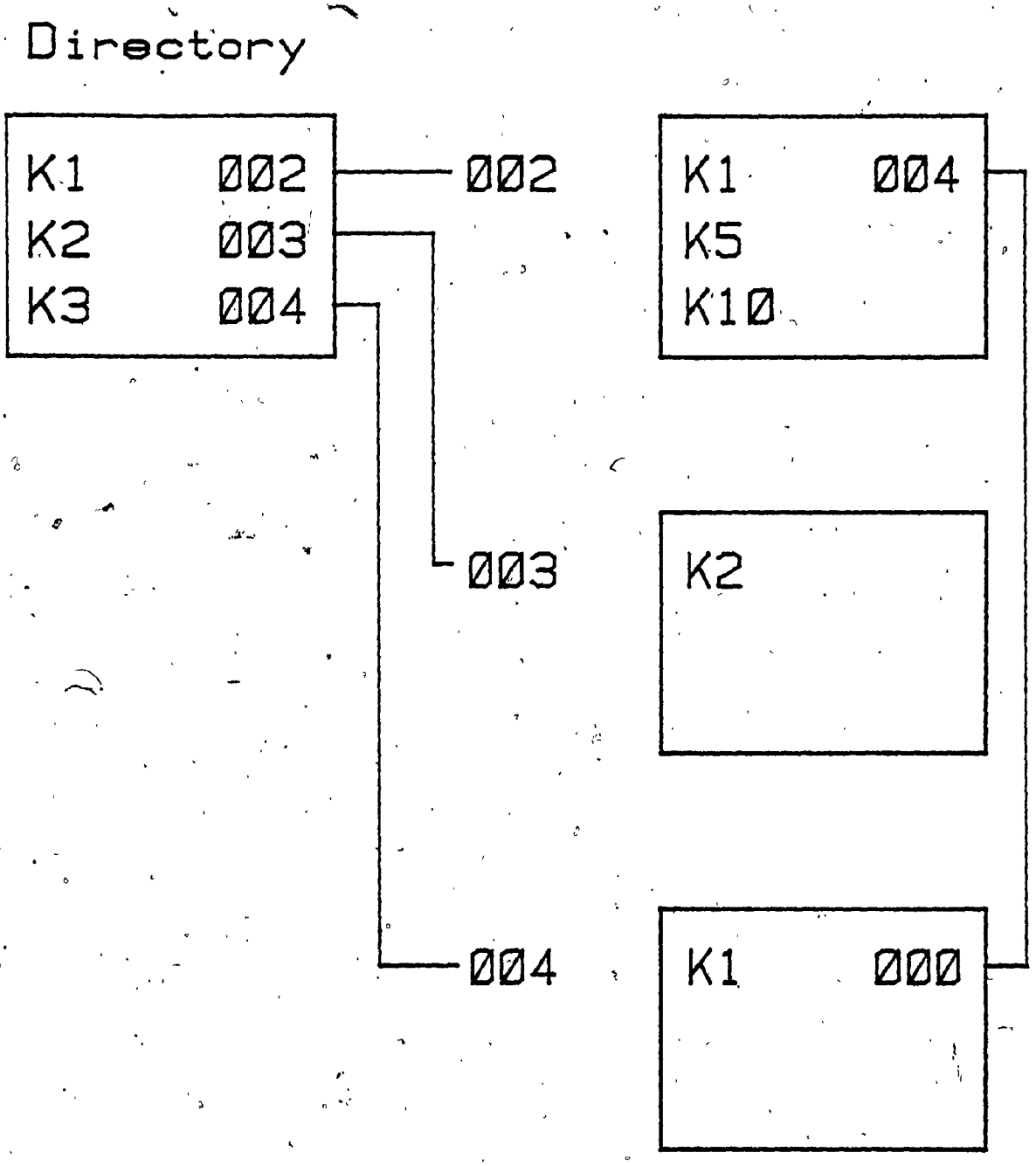


Figure 3.4.1 Directory structure for linked files (Hill78).

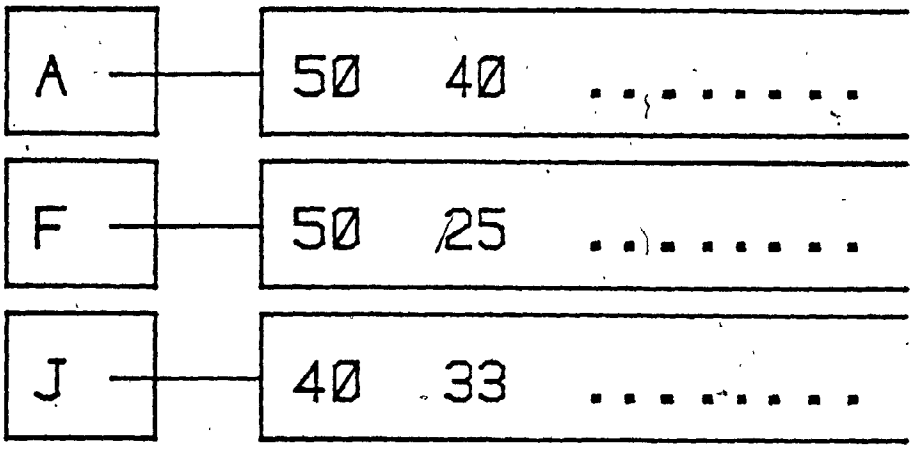
3.5 RECORD PROCESSING USING INVERTED FILES

Occasionally it is necessary to associate a key with one or more records or many keys with the same record (Figure 3.5.1).

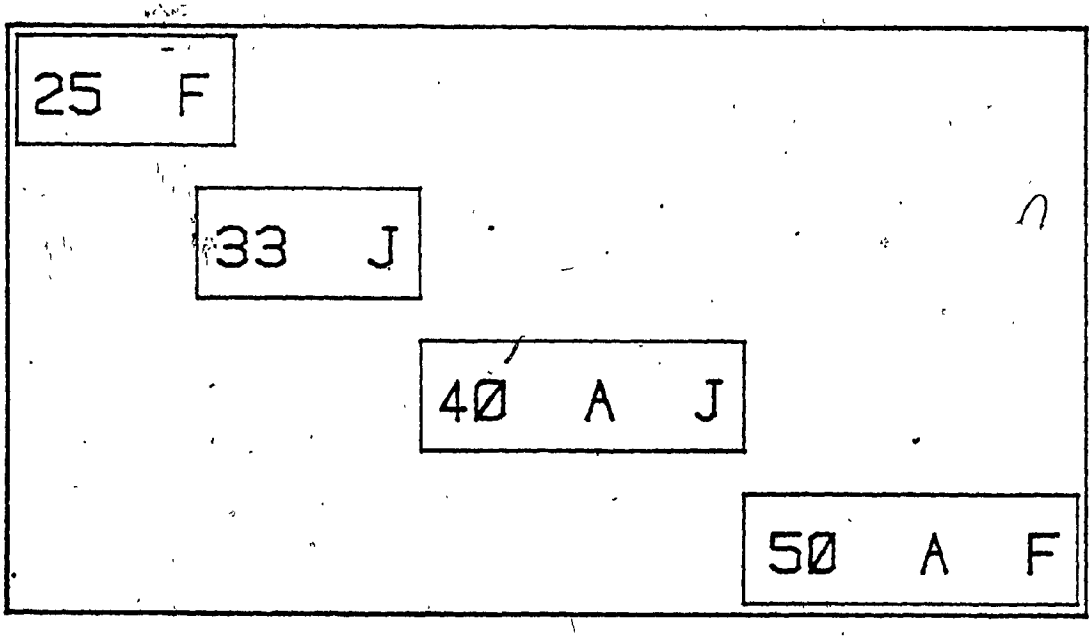
This organization lends itself to a representation by a structure called an inverted file. The time required to load an inverted file is called the load time. This process involves inverting the file. For large files this operation requires a large amount of computer time.

When the inverted file already exists the load time is only the time required by the DASD to get ready for processing. Assume that the items are randomly distributed in the file J to be inverted. The file is read sequentially until EOF is reached. Assumed there are F records in the file J with length n. Each READ of J produces one buffer full of information which is the size of one record. Define As to be the access time of the DASD the total time spent accessing J is Aj=As*F. The load factor is a=m/b where m=(n*F)/d N=n*F (Hill78).

Databases using inverted files have simpler DDL design and changes in data structures are less elaborate. On the other hand, indices represent significant overhead especially were many fields are inverted (Ross78).



a. Inverted list structure



b. Main file

Figure 3.5.1 Inverted file structure.

4. DATABASE MODELS

The different database families have different data structures and relationships among the records. There are three principal models: the hierarchical, the network, and the relational. The first two belong to the category of physically linked DBMS (Ross78), where related records are linked together by 'paths' (pointers). In contrast, the relational model does not use 'paths' at all and relations are evaluated at execution time.

4.1 Hierarchical model

The hierarchical model uses files with a tree structure relationship between the records. It is suitable for some applications, but many data structures are not in tree form with the result that, in order to be implemented, they must be converted if possible (Marti76). The hierarchical structure and the possible schemas supported by the hierarchical model are given in Figure 4.1.1

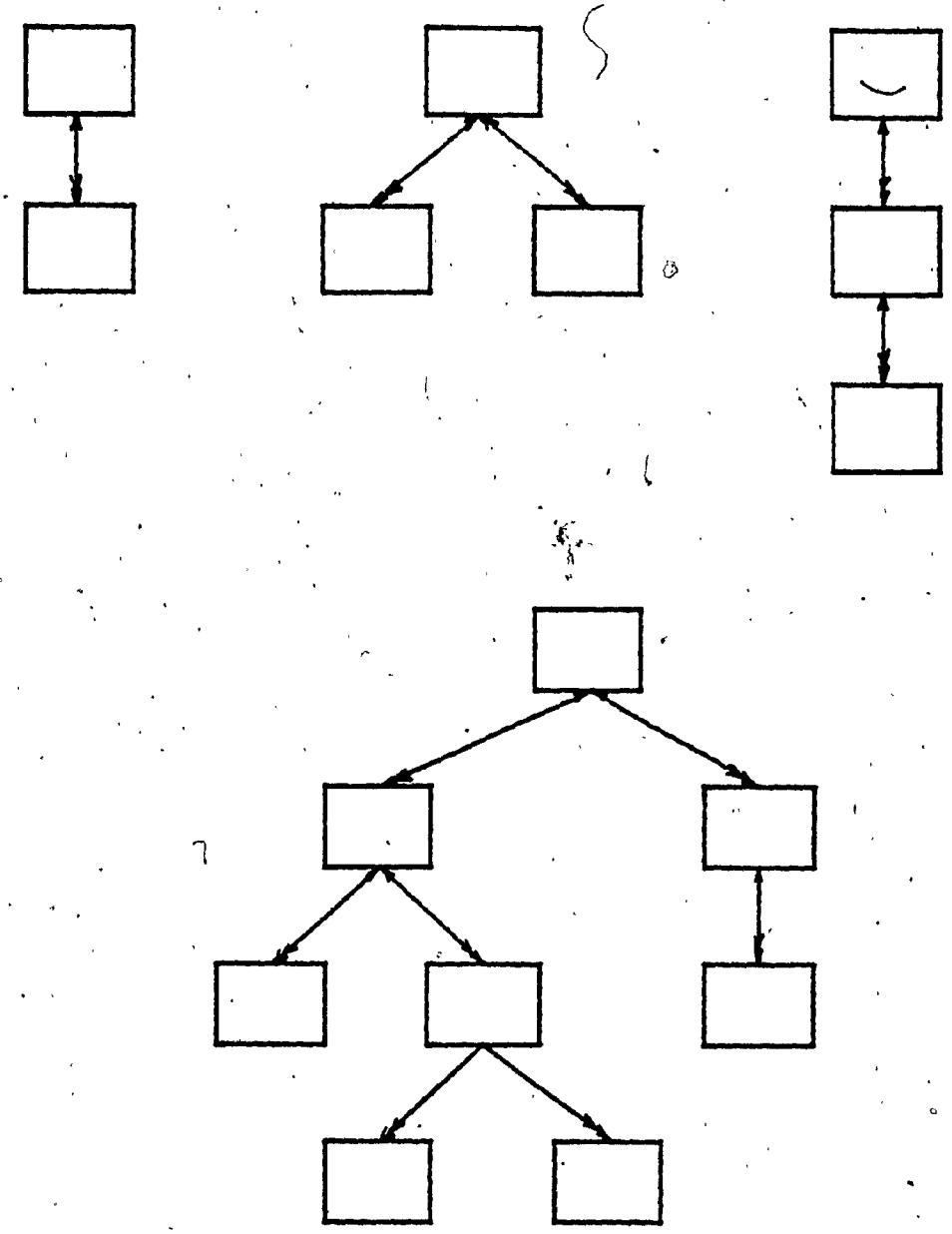


Figure 4.1.1 Hierarchical structures (Marti76).

4.2 Network model

This model is based on directed graph structures where the child in a relationship may have more than one parent. Because of this characteristic, this structure is not hierarchical and cannot be regarded as a tree. It is referred to as a "network" or "plex structure" (Marti76). In some networks a situation called a cycle occurs, where a node has as child its parent. A special kind of cycle, where the child is the same as parent, is called loop (Marti76). The network structure is divided into two major categories.

1. Simple network (plex structure) (Figure 4.2.1)
2. Complex network (complex plex structure) (Figure 4.2.2)

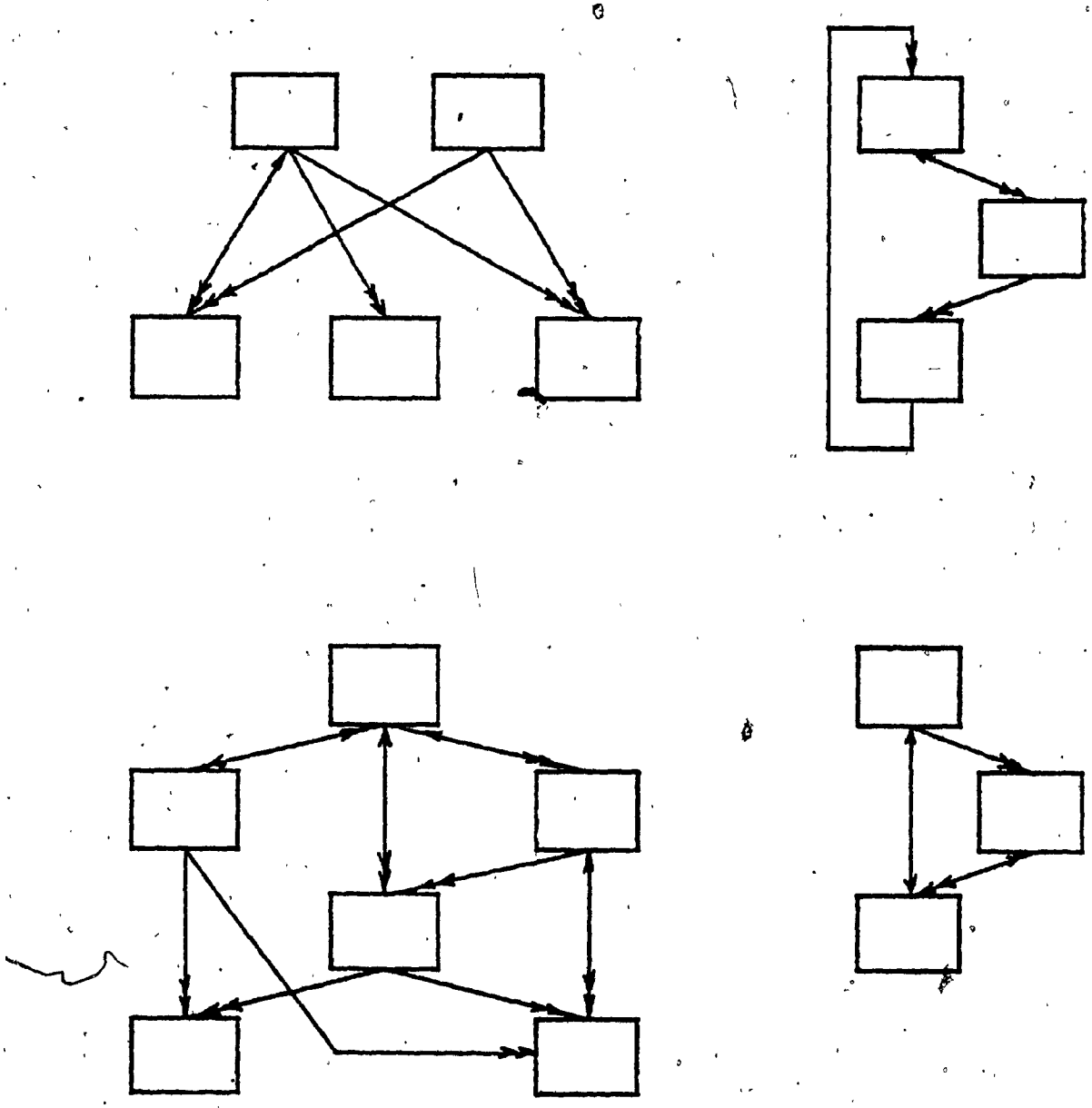


Figure 4.2.1 Simple network (Marti76).

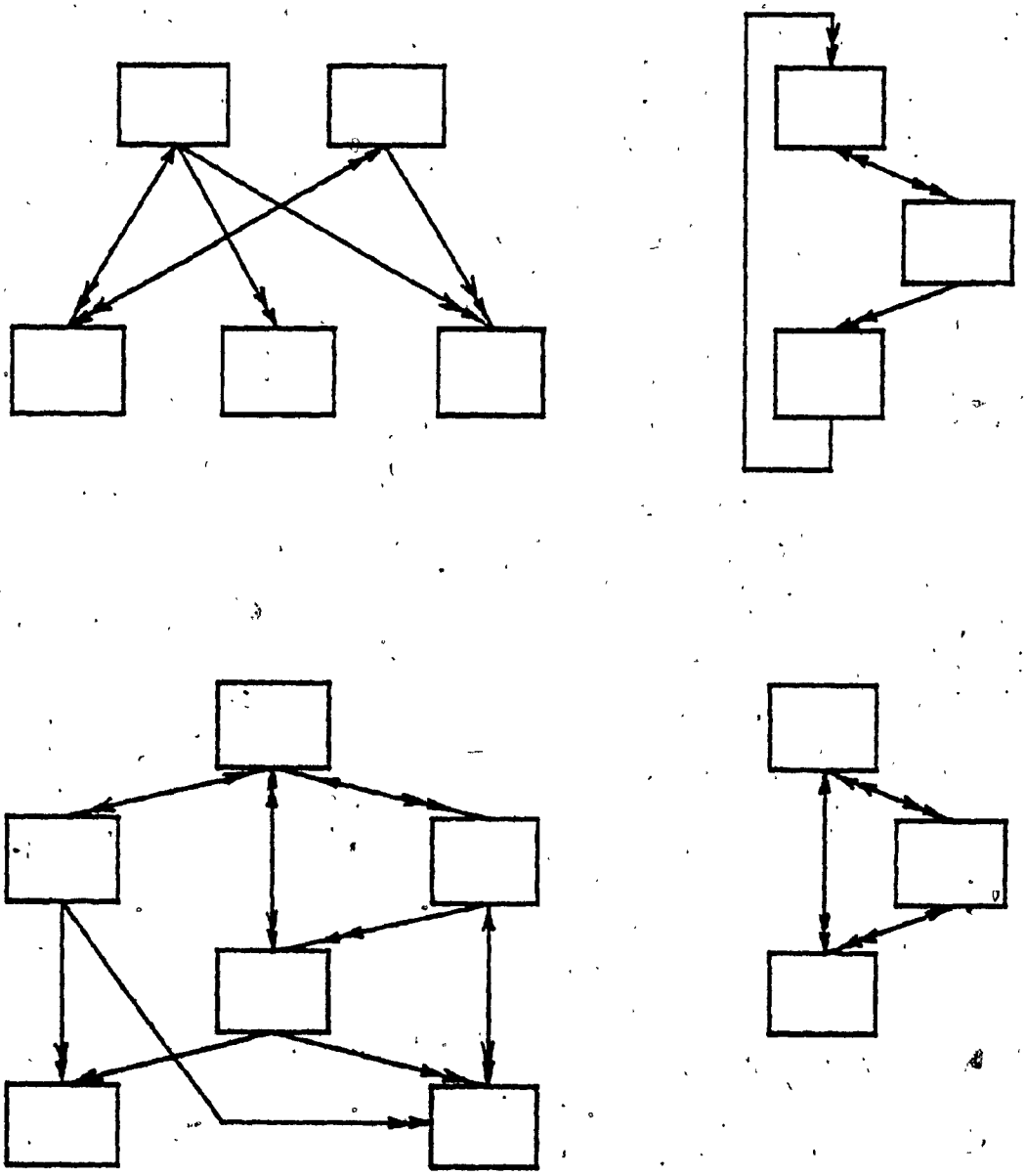


Figure 4.2.2 Complex network (Martí76).

4.3 Relational model

The relational model differs in several aspects from the hierarchical and network models. First, the relational model is based on the theory of relational mathematics. Second, it consists of a group of concepts that are not related to any programming language in particular. Third, this model tends to present the data as it exists. Fourth, the relational model reduces data relationships to simpler components.

A relation is a two dimensional table having the following properties.

1. Entries in the table are single valued with no repeating groups.
2. Entries are column homogeneous: items of a column are of the same kind (Marti76).
3. Each column is assigned a distinct name and is referred to as attribute
4. Duplicate rows are not allowed.
5. The order of the rows or the columns is not significant.

Each row of a relation is called a tuple. If a relation has n columns, then each row is referred to as an n-tuple and the relation is said to be of degree n. Each attribute has a domain which is the set of values that can appear in the attribute. The data manipulation language is based on relational algebra with operators such as projection, join,

restriction, and set theoretic .

The projection operator returns only the specified columns of a given relation and eliminates duplicates from the result (Champ76).

The join operator takes two relations as arguments and forms a new relation by concatenating a tuple from the first and a tuple from the second wherever a given condition holds between them (Champ76).

The restriction operator selects only those tuples of a relation which satisfy a given condition(Champ76).

The set theoretic operators are union, intersection, and set difference (Champ76).

The implementation of a relational model at the initial stage may contain repeating groups. The relation in this case is said to be in the unormalized form, which contradicts the first property. In order to overcome this, the process of normalization is employed where all the repeating groups are removed to form smaller relations. The relation is now said to be in first normal form (Robin79). Second and third normal forms are obtained by dividing the relation into successively smaller ones. Normalization is applied so that updating will not affect existing programs. Normalized data structures have important advantages over other data structures, especially when they are in third normal form. Some of these advantages are:

1. Ease of use: simple representation of data (Marti76)

2. Flexibility: in accessing the desired data (Marti76)
3. Security: is more easily implemented (Marti76)
4. Ease of implementation: the physical storage of a flat file is less complex than the physical storage of a hierarchical or network structure (Marti76).
5. Data manipulation language: can be based on the relational algebra or relational calculus (Marti76).

5. DISTRIBUTED DATABASES

A single system with database files in more than one dinstinct location is referred to as a distributed database. The motivations for distributing a database are as follows.

1. Cost reduction

   A. Data can be stored close to its location of origin

   B. Less transmission of data is required, reducing telecommunication costs (Marti76).

   C. Economies can result from minimizing the usage of large central computer facilities (Marti76).

2. Load considerations

   In some systems the traffic load is greater than the capacity of today's database software. This load can be handled by a number of separate systems with identical data structures, which serve different areas exhanging update information when appropriate (Marti76).

3. Localized management

   Some computer sites in a large corporation maintain their own staff, and produce their own reports but consolidate some of the items at the corporate level (Marti76).

4. Small computer hardware

   The rapid fall in cost of small computer hardware (Marti76) has made local storage on mini or microcomputer systems an attractive option.

5. Separate information systems

Operating system and information system databases can be separated. An application system may be linked to local databases in different geographical locations (Marti76).

6. Availability

Local computer systems provide much more system availability to the user than a centralized one. Power or other types of failures in one site do not affect the whole network.

7. Security

The database application can be protected against fire, sabotage, earthquake, e.t.c. (Booth77).

8. Fast response to local problems

9. Modular growth for new appications

DATABASE DISTRIBUTION

Distributed databases can be classified in the following categories.

The partitioned database (Booth77), or separate information and operation databases (Marti76), refers to a database which is decomposed into separate units that reside at more than one location (Figure 5.1). Each of these units is called database partition and contains a segment of data structured according to the specifications of the overall logical database. Partitioning a database may be necessary in cases when the database is very large and is accessed frequently. In such cases, the contention is great because many programs may need to access the database. One

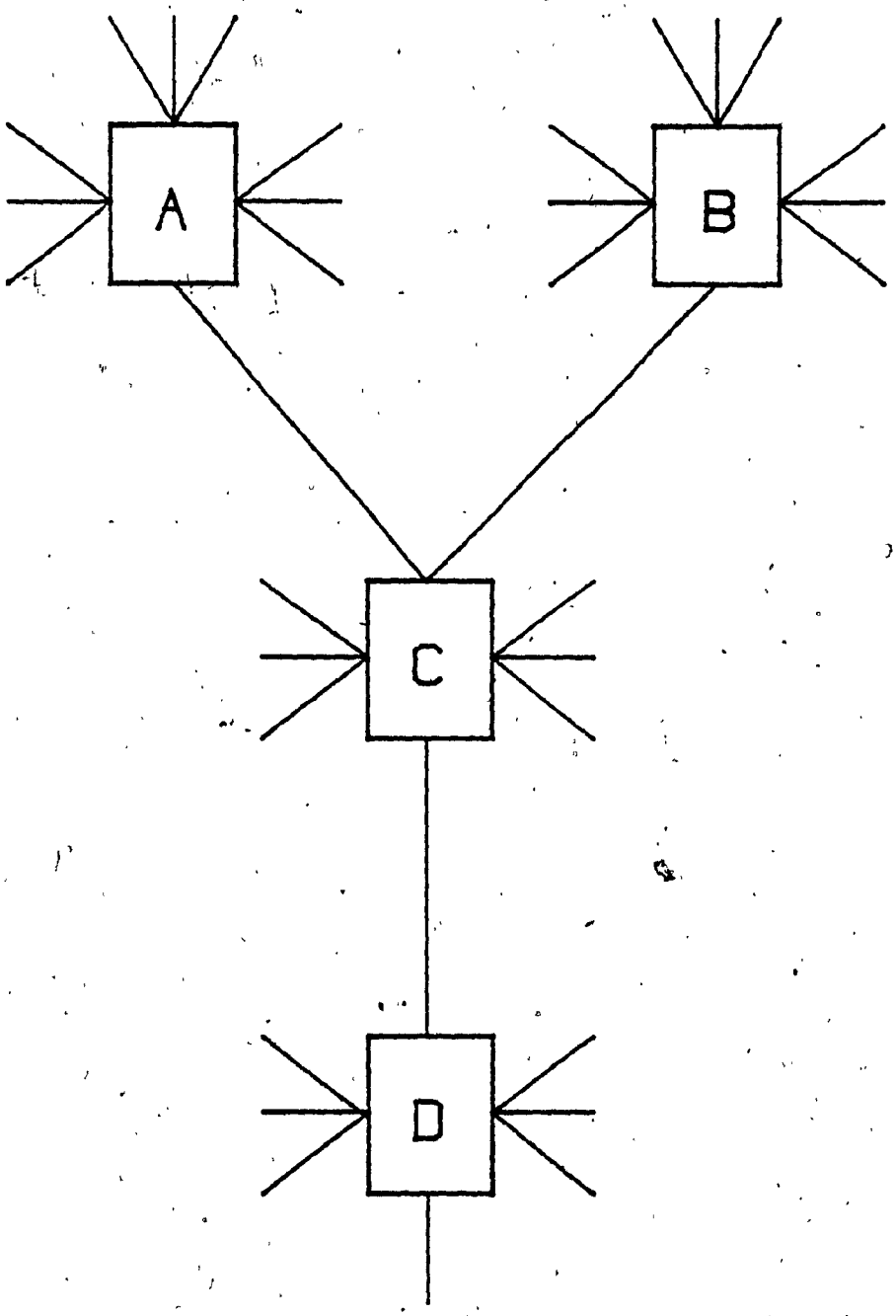


Figure 5.1 Partitioned Database

processing configuration may not therefore be adequate for the load. A second reason for partitioning a database is to reduce the telecommunication cost. If each segment of the database is physically close to its most frequent users, most communications with it will be local, and cosequently low cost.

The replicated database (Booth77) or split file system (Marti76) is the category where duplicate databases exist in different locations (Figure 5.2). The purpose of this scheme is to move elements closer to the point of origin and at the same time reduce the telecommunications cost. An additional advantage is that data replication decreases the threat of loss or damage. A replication of a database can serve as a backup copy. Updating the various copies of the databases, however is more elaborate. One approach is to update all copies simultaneously, but this implies a heavy overhead. It should be noted that replication may exist within a partitioned database (Booth77).

DESIGN CONSIDERATIONS (Ramam77)

A distributed database is a logical integration of several databases residing on different processing elements (PE's). The success of such an integration depends on the functional relationships between various processes residing on the PE's. In order to design a distributed database, some factors must be taken into consideration.

-The size of the database residing on the PE's depends on

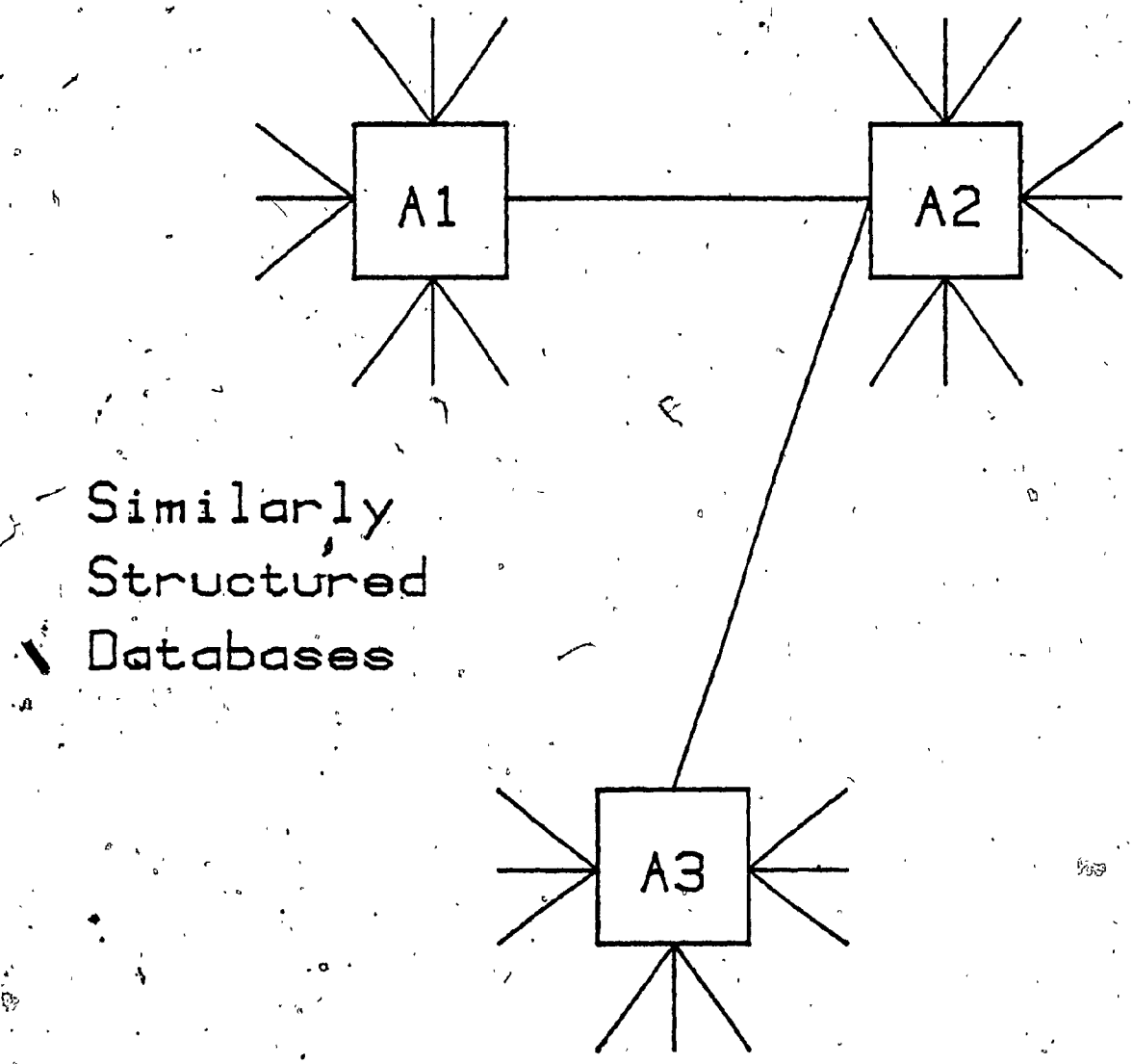


Figure 5.2 Replicate Databases

the application and access time requirements.

-Access time may be very critical and may not allow sufficient time for transfer of files between locations. Access time should be considered in conjunction with available data paths (pointers), the nature and size of files, and the cost associated with replication and storage.

-Access conflicts may occur when a database is shared between many PE's.

-File location and relocation will affect communications requirements and costs.

-Integrity must be maintained on all copies of the replicated database.

-Precedence relationships are important in producing correct results.

-The boundaries of interrelated databases must be protected.

-Recovery is more complex than in a centralized database.

-Security is required to protect sensitive data files from being destroyed due to malfunction in the system or due to a malicious user.

-Deadlock detection and prevention is very important.

6. BACK-END STORAGE NETWORKS

Back-end storage networks are dedicated systems whose design is optimized for high performance information storage and retrieval. They are a relatively recent concept in database technology, and owe their existence largely to the increased requirements of modern database systems, together with the falling cost of high performance computer hardware. The term 'back-end' is derived directly from the adjective 'front-end' used for communications processors. Back-end systems differ from the front-end systems in both data transfer speed and functional emphasis (Thorn80) (Figure 6.1).

Front-end machines evolved from hard-wired multiplexers which connected remote terminals to host computers by means of telephone lines. These multiplexers provided a primitive connection function, leaving most of the communications processing to the host computer. Modern front-end processors perform most of communications processing leaving the host computer free to perform its other tasks. In contrast, back-end systems perform functions associated with file transfer and high speed communications with the host computer. The objective of a back-end network system is to provide high speed access to peripheral subsystems. The first back-end network design was started in 1964 at Lawrence Livermore Laboratory with the development of a subnetwork for the high performance local computer network

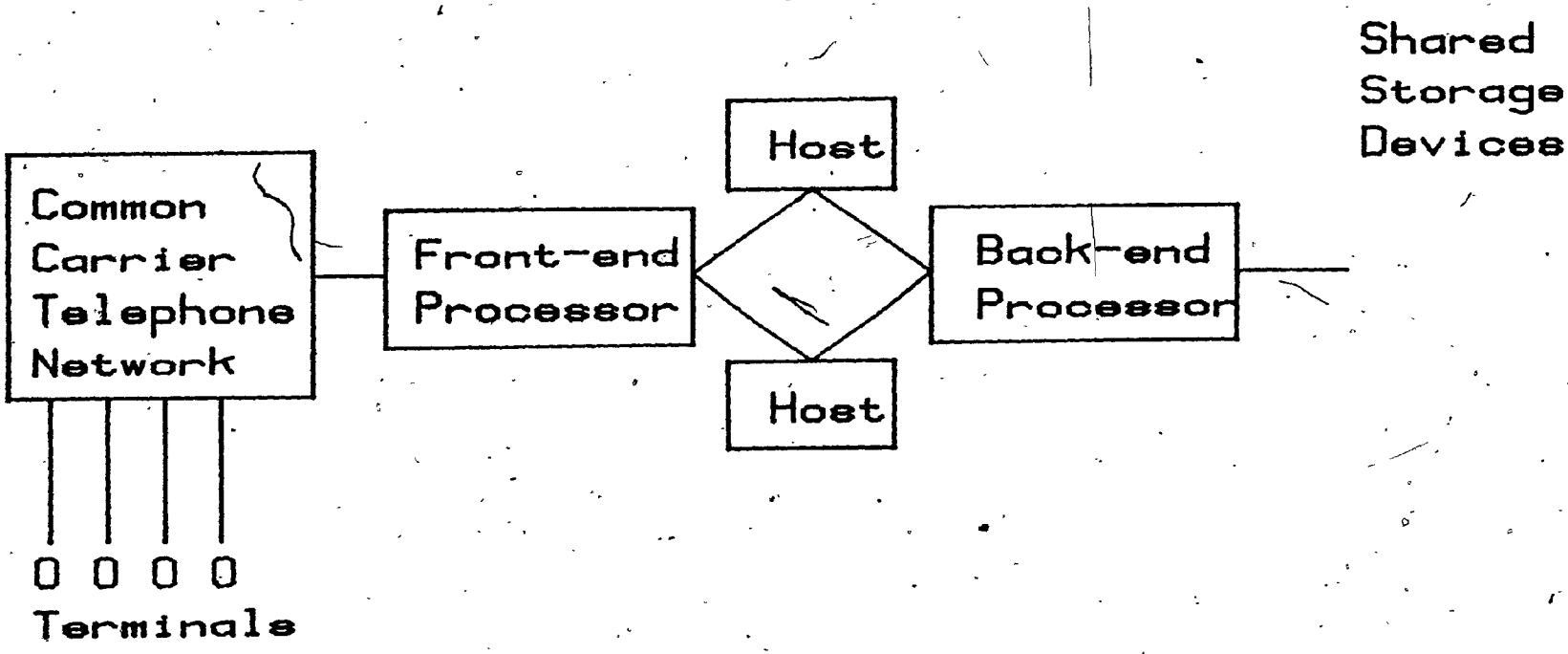


Figure 6.1 A Back-end Network (Thorn80).

called 'octopus'. The octopus network supports several thousand users in an interactive computing environment and utilizes four CDC 7600's, two CDC STAR's and a PDP11-20 operating as the back-end processor. Another back-end storage network was built by NASA to support the skylab project. Data transmitted from the orbiting laboratory were received at a remote tracking site and then relayed to mission control center. At Houston center, a UNIVAC 494 front-end processor distributed the data to the five IBM 360/75 computers having a CDC CYBER 73 operating as the back-end processor (Thorn80). Recent technology developments are expected to make back-end systems more attractive in the future. Large storage modules built from disks, bubble memory or charge-coupled devices are becoming feasible as the costs per bit decrease. Database processors, specialized hardware/software subsystems optimized for storage update and retrieval, are a logical choice. Database processors solve several problems in back-end storage systems. In conventional systems, large portions of the file are transferred to the requester for searching, while in a database processor subsystem, data is accessed and searched within the subsystem, avoiding wide bandwidth transfer (Champ80). One of the technologies used in database processors is associative processing, where the information is accessed by the value of the key instead of the physical location. Early systems used conventional minicomputers as front-ends to the main storage subsystem

and interfaced host computers over a conventional communication network. Early architectures were designed as back-end storage subsystems, and are considered as intelligent-controller approaches. Some notable examples are CAFS (content addressable file system) based on a pipelined processor accessing conventional disks, and RAP, an array of processing elements, each working on a portion of the total database (Champ80). The database processor goes beyond the intelligent controller approach by using a moving head disk with a microprocessor associated with each head, and featuring parallel data transfer from the disk.

Technology development in future years will have a determining effect on back-end network architecture. Microprocessors will play a vital role in back-end networks with applications such as protocol handling, storage management, disk controllers, etc. Storage technology is progressing rapidly and new types of memories are gaining importance in the market. Dynamic MOS storage is generally used in main storage due to low cost. With the introduction of 64K-bit chip a 0.5 megabyte memory will be contained on a board 15cm x 30cm and the 256K-bit in the next few years will reduce the size further. In direct access storage, the magnetic disk remains the least expensive device. Although the cost of direct access storage is declining, the lowest cost per stored byte pertains only to the largest units. If we consider a 'back-end network system requiring 600

megabytes of storage, and if the storage is centralized an a single unit, the cost per byte would be five millicents. If the same data is distributed in 60 nodes of 10 megabytes each, the cost per byte would be 40 millicents, or eight times as much. Solid state mass storage technologies such as bubble memory and CCD's have random access characteristics, and bubbles are expected to replace fixed head disks in future. Interconnecting the various modules of a back-end storage network involves four technologies: direct wire, coaxial cable, twisted wire and fiber optics (Champ80).

Direct wire, with transmission rates 10 to 100 megabit per second, can be used only for short distances (tens of meters).

Coaxial cable, with transmission rates up to megabits per second, can be used for distances 1000 to 10000 meters.

Twisted pair, with transmission rate up to 100.000 bits per second, is adequate for distances over 10.000 meters.

Fiber optics, with transmission rate up to 50 megabits per second, can be used for distances of several hundreds of thousands of meters. Experiments show that one gigabit per second systems are possible. In addition to performance advantages, fiber optics are immune to electromagnetic interference. In the next few years, continued progress is likely to produce gigabyte disks, and microcomputers with power of today's main frames. These developments, combined with fiber optic communications, will completely change the

design parameters for back-end networks, resulting in new and different architectural concepts.

7. BACK-END PROCESSORS

Back-end processors are specialized computers designed to off load database tasks from the main computer system. They are specially designed to perform database functions, and thus have appropriate functions implemented in hardware. This raises both their speed and efficiency for database tasks well above that of conventional computers. Their architectures are described in the following:

7.1 Associative processors

An associative processor can generally be described as a processor having the following two properties.

1. Stored data items can be retrieved using their contents or part of it instead of their addresses.
2. Data transformation operations can be performed over many sets of arguments with a single instruction.

These parallel processing characteristics enable associative processors to process data at a faster rate than conventional sequential computers. They are very efficient at handling information storage and retrieval, search functions, arithmetic and logical operations on large sets of data, and control and executive functions in large-scale computer systems. Because of their high implementation costs, associative processors are usually used in conjunction with standard sequential computer systems. It is anticipated that the rapid development of large-scale integrated circuits (LSI) and very large-scale integrated

circuits (VLSI) will reduce their implementation cost, and associative processors will be used more extensively for enhancing the performance of various special purpose and general purpose computer systems. The major difference between an associative processor and a standard sequential computer is the presence of the associative memory. (Figure 7.1.1). Because of this difference, the other blocks are different from those of the standard sequential computer. Associative memories, in order to retrieve stored data items by their content, must be able to access the memory words by matching their content with a given search keyword. The basic element of an associative memory is the bit-cell which can be written, read, or compared to interrogating information. A search keyword is compared to all the words in the memory through the interrogating bit drives and the comparison logic circuitry. To be able to match multiple search keywords, associative memories have methods of tagging all matched records. Matched words can then be accessed with a single instruction. Associative memories may perform either parallel by bit comparison (word-parallel or word serial) or serial by bit comparison (bit-serial). The following comparisons can be carried out.

equal	not equal
less than	greater than
not greater than	not less than
maximum value	minimum value
between limits	not between limits

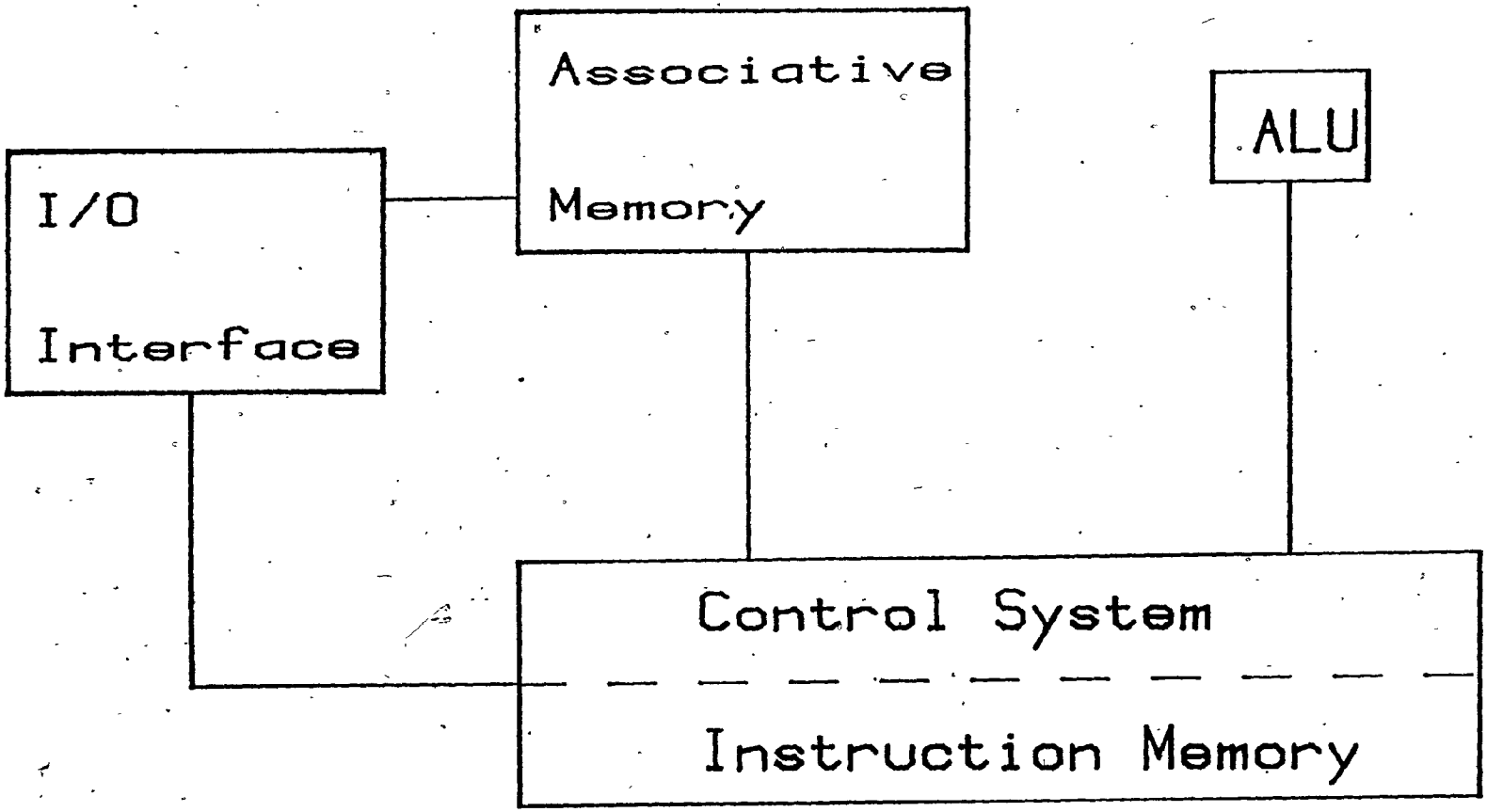


Figure 7.1.1 Associative processor architecture.

next higher next lower

The operation of an associative memory can be illustrated by the following example of a file search (Figure 7.1.2). The indicator bit is used to indicate the results of the search. When it is set to 1 a match is indicated while a 0 signifies a negative result.

As mentioned earlier, associative processors can perform other complicated functions in addition to the comparison operation. This places them in the general category of SIMD (single instruction stream multiple data stream) parallel processors. A SIMD machine is a computer in which a single instruction instructs more than one processing element, which in turn can either execute or ignore the current instruction as a function of its own status.

The architecture of associative processors can be classified into four categories according to the comparison process of their associative memory.

Fully parallel (divided into two sub-categories).

A. Word organized where the comparison logic is associated with each bit cell of every word and the logical decision is available at the output of every word (PEPE).

B. Distributed-logic where the comparison logic is associated with each character cell or with a group of character cells.

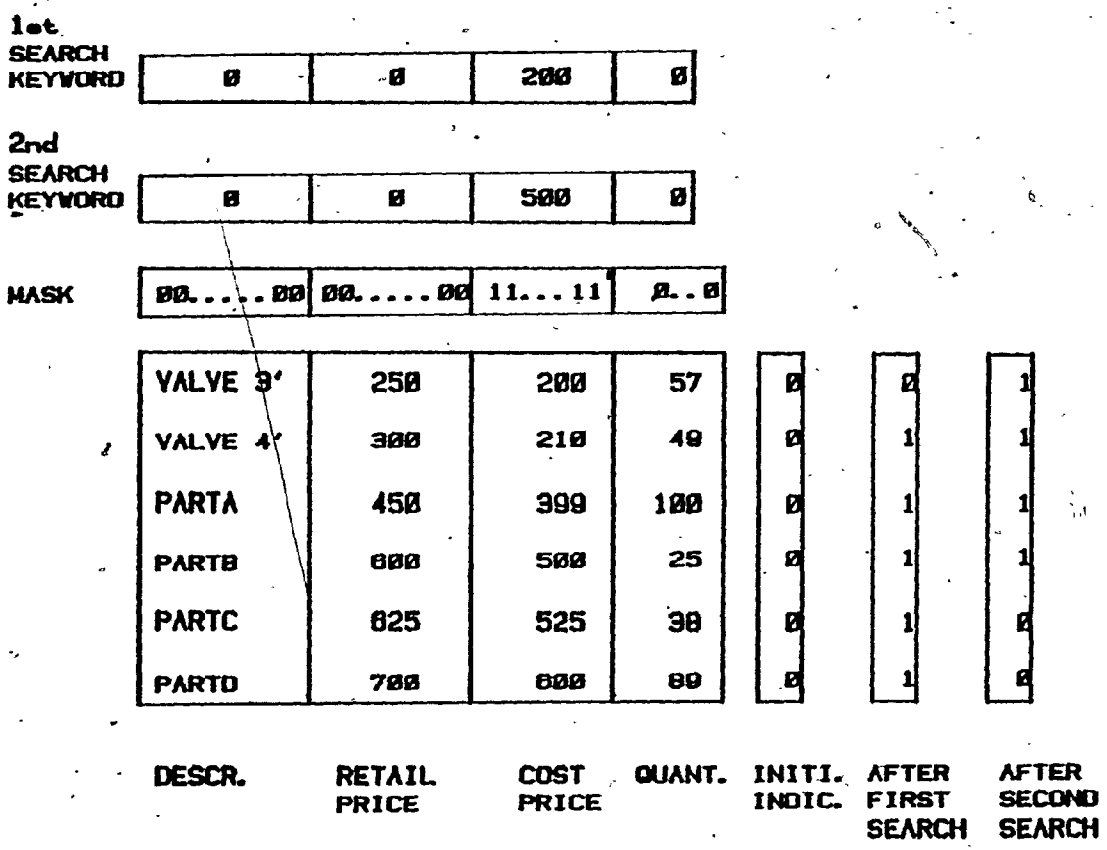


Figure 7.1.2 Associative search (Yau77).

Bit-serial. In this architecture only one bit column (also called bit-slice) for all the words is operated one at a time. For this reason a bit-serial associative processor is also called bit-serial word-parallel associative processor (STARAN).

Word-serial This architecture represents a hardware implementation of a simple program loop for search. The only factor contributing to increased efficiency is a reduction in instruction decoding time since only a single instruction is required to perform a search.

Block-oriented This architecture can be implemented by using logic-per-track rotating memory which is based on a head per track disk unit with some logic associated with each track.

Fully parallel word organized associative processors

The major characteristic of a fully parallel word-organized associative processor (Figure 7.1.3) is that the comparison logic is associated with each bit-cell of every word of the associative memory. The comparison is performed in parallel-by-word and parallel-by-bit. In this architecture, every crosspoint represents a bit cell of the associative memory. The operations in this associative processor are simple and fast compared to others. On the other hand the hardware is complex because every bit has to contain the comparison logic.

Fully parallel distributed logic associative processors

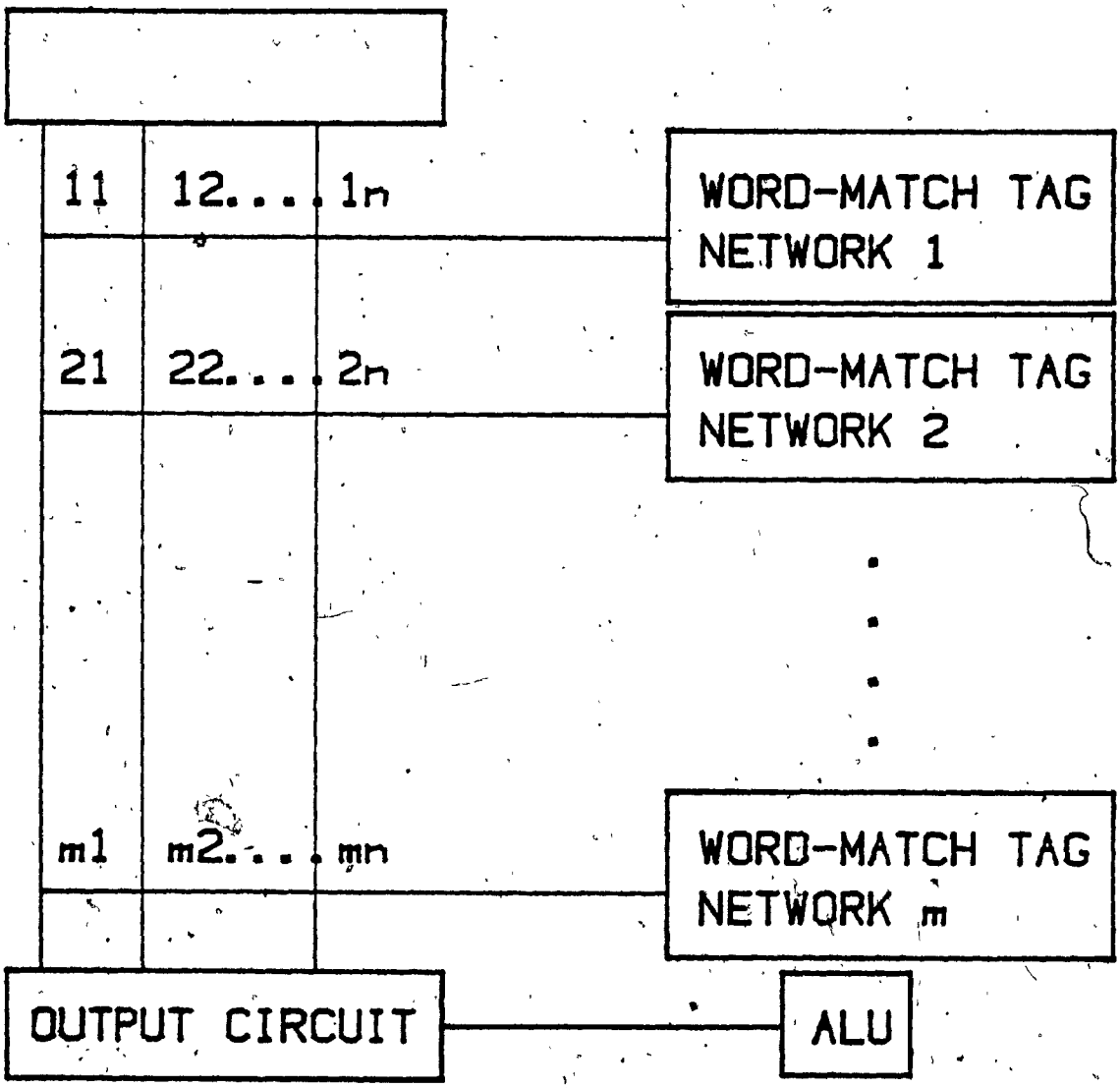


Figure 7.1:3 (Yau77)

Fully parallel word organized associative processors

This type of associative processor is character-oriented: its memory has its comparison logic associated with each character cell or group of character cells. The best known associative processor of this type is the PEPE developed by Bell laboratories for the US Army Advance Ballistic Missile Defence Agency (Figure 7.1.4).

In this type of architecture each character cell has a single state cell element which may be either in an active state or quiescent state. Each character cell also has a number of cell symbol elements E1,E2,E3,... depending on the size of the symbol alphabet. The state cell is a bistable device such a flip-flop. Each character cell stores one character symbol and can communicate with its two neighbour character cells and the control system . Under this scheme the distributed logic memory must have enough cell logic circuitry so that it can produce a yes or no answer to a simple question such as a symbol comparison. If we want to receive all strings whose name is AF, we have to ask each character cell whether its character symbol is A. In the case of a yes answer, the cell should have the necessary logic circuitry to request the next cell to check if its character symbol is F. When the cells answer yes they output their content.

Bit-serial associative processors

This architecture eliminates the need for expensive logic in each memory bit by using only one bit column for a large word being processed (Figure 7.1.5). In this memory

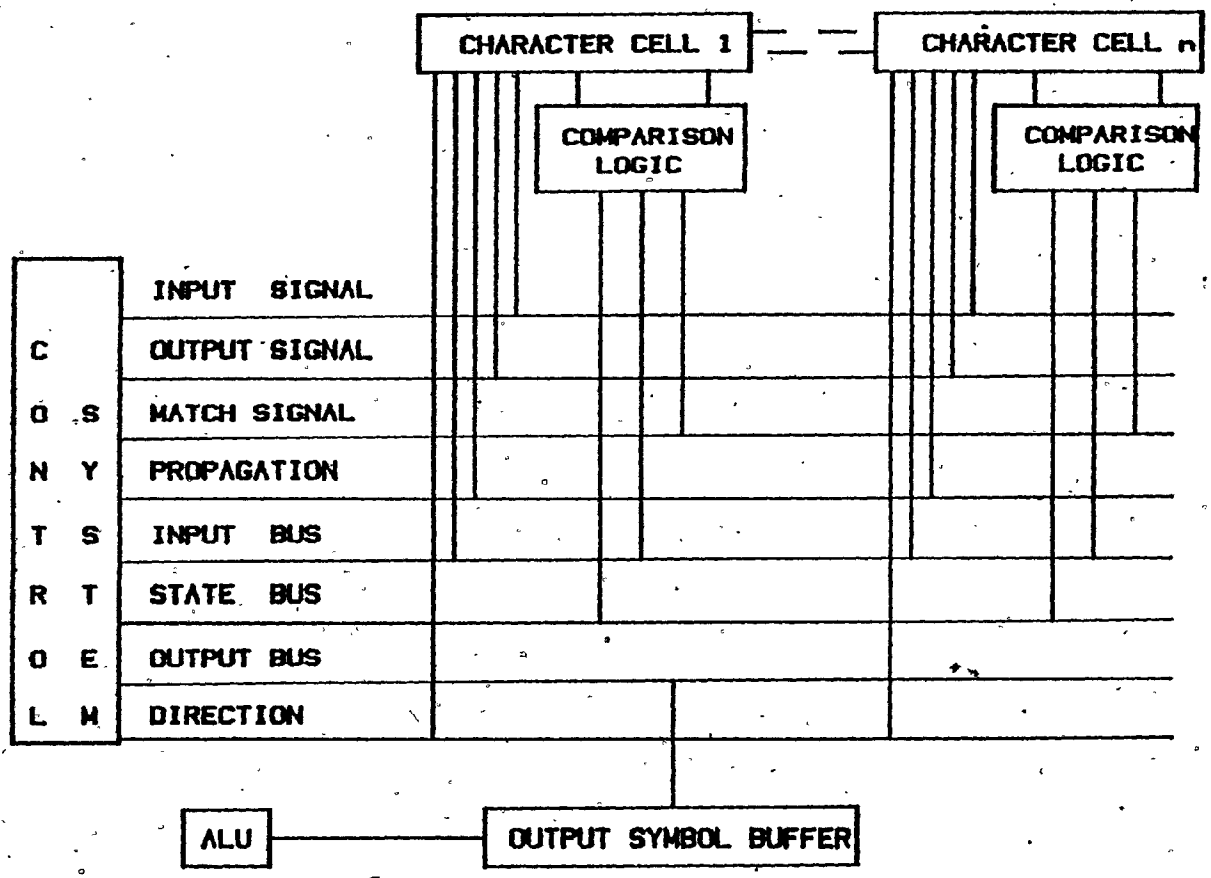


Figure 7.1.4 (Yau77)
Fully parallel distributed logic associative processor

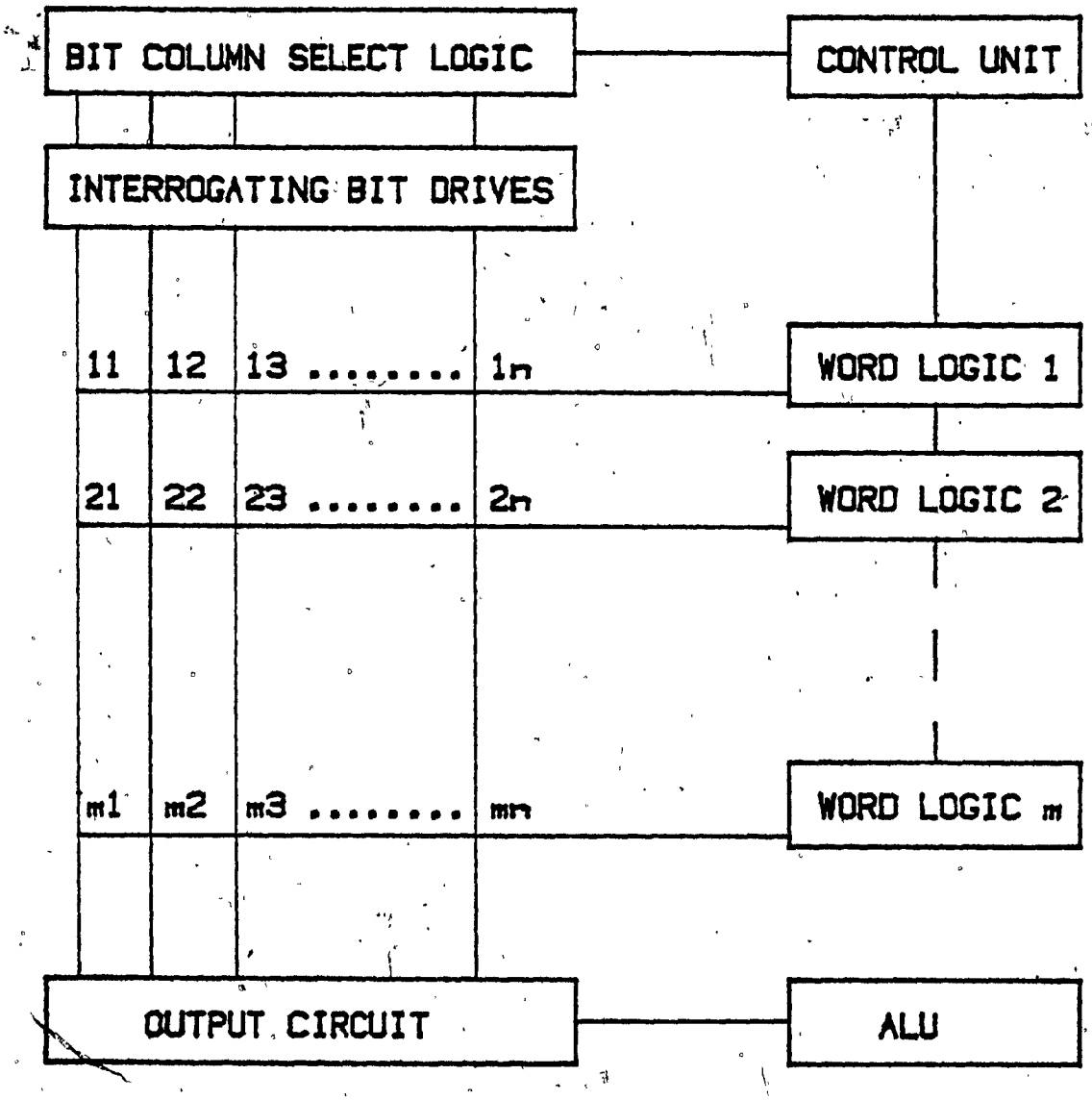


Figure 7.1.5 Bit-serial associative memory (Yau77)

organization only one bit column is operated on at any given time. Bit columns are selected by the bit column select logic circuitry. The word logic associated with each word line provides the ability to perform associative processing. The word logic is identical for all words and consists of a sense amplifier, storage flip flops, write amplifier, and control logic. The storage remembers the match state from one bit to the next. The capability of the storage to act as a shift register provides the communication link between adjacent words. STARAN, developed by the Goodyear Aerospace corporation, is a bit-serial associative processor installed in a number of locations, including the Rome air development center Defence Mapping agency, the US Army engineering topographic laboratories, and the NASA Johnson space center in Houston.

Word-serial associative processor (Figure 7.1.6)

It was mentioned above that the word serial associative processor represents a hardware implementation of a simple program loop for search. The reduced instruction decoding time due to the fact that only one instruction is required for a search contributes to the efficiency of this architecture. An earlier proposal suggested the use of circulating associative memories to allow many memory words to time-share a single set of content addressing logic. Another model was a word-serial associative processor based on a word serial associative memory using N ultrasonic

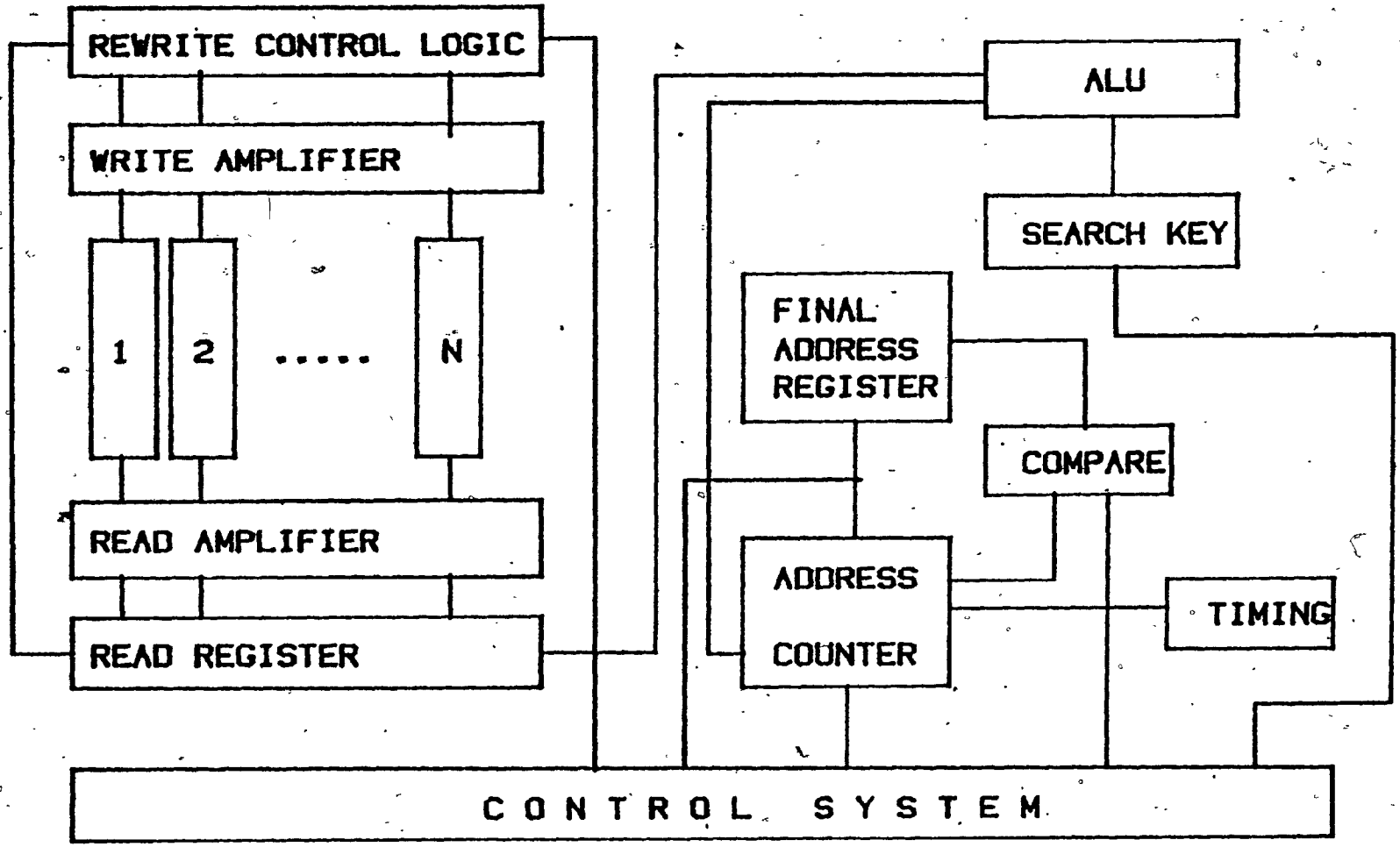


Figure 7.1.8 Word-serial associative processor (Yau77).

digital delay lines where N is the number of bits of a word, operating at 100 MHZ with 10 usec delay time. Because of the slow speed of word serial associative memories, only experimental models have been developed.

Block-oriented associative processors

Block-oriented associative processors represent a design compromise between the expensive bit serial and the low speed word serial associative processors. They utilize a rotating mass storage device such as a disk to provide a limited degree of associative capabilities. The processing logic is implemented on a per track basis: a head per track disk memory has some logic associated with each head. The concept of a fully parallel distributed logic associative processor, was implemented in the RAPID (Rotating Associative). In this processor, the data transfer rates between head per track disks and the distributed logic memory is high. The system is suitable for applications requiring large storage capacity.

7.2 DATABASE MACHINES

7.2.1 Philosophy of database machines

A database machine consists of specialized hardware supporting various database management functions, as found in most software databases. The principal motivations for the hardware implementation of the functions are reliability and performance. Large and complex software database systems tend to be failure prone and practical verification methods for software database systems are not yet available. Methods for verifying hardware functionality, design, and performance are however available. Technological advances have overcome some of the problems of logic complexity, capacity requirements and space. By implementing the database management functions in hardware, the basic functions are more reliable, and software is less complex and smaller in size. Conventional computer systems are not designed for database applications but rather to carry out numerical computations. Database functions, on the other hand, are concerned with storage and retrieval of data. A conventional computer equipped with a database management software spends most of its time interpreting calls between the software and the operating systems. Software database management systems therefore often become bound with heavy I/O traffic to or from secondary storage, with a resulting degradation of response time. Specialized hardware to perform database functions can thus improve performance by relieving the CPU, main memory, and channels from the heavy

I/O load.

Database machines require large on-line storage with fast access time. Although associative memories meet this requirement, their use is limited by their high cost and limited capacity. Researchers have therefore tended to focus on staging devices such as disks, or to use cellular logic design to provide content addressability. Moving head disks specially configured for parallel readout of all the tracks of the cylinder are used. Content addressability for a cylinder can be achieved by use of appropriate logic for all read heads. Several technologies such as bubble memory, charged coupled devices, and electron beam addressed memories are becoming available, promising larger memory capacities and faster access times.

7.2.2 EXISTING DATABASE MACHINES

In this chapter a few of the existing database machines will be presented

7.2.2.1 RAP - The relational Associative Processor (Figure 7.2.2.1.1) is designed to operate as peripheral or back-end processor in order to provide fast response time. The basic architecture of RAP consists of identical components called cells, a statistical arithmetic unit, and a central controller.

Each cell is composed of a processor and block addressable memory. The processor is specially designed for database

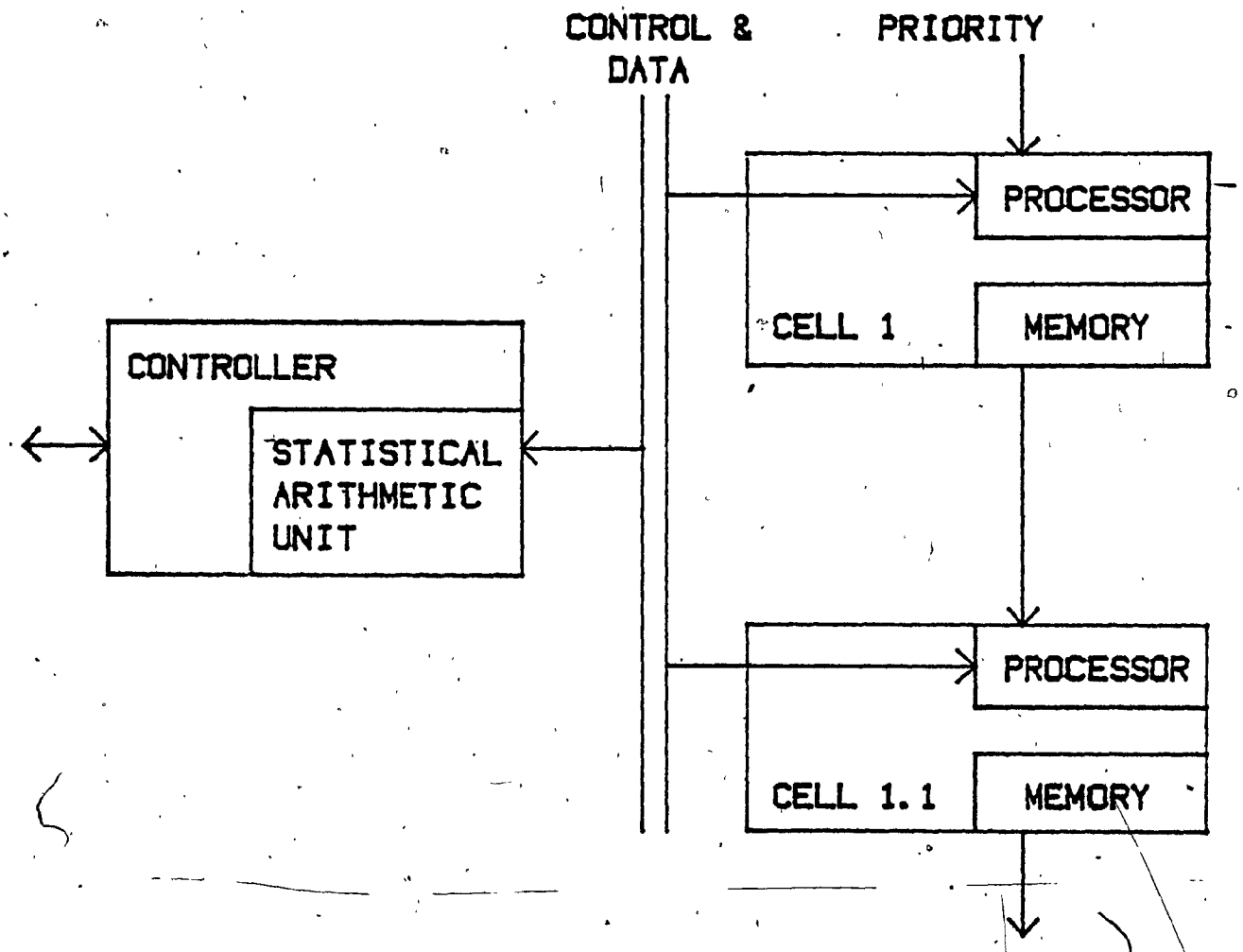


Figure 7.2.2.1.1 (Sohu79)
RAP.2- Relational Associative Processor

insertion, deletion, update, and retrieval functions. The processor has been designed using large scale integration (LSI) circuit technology. Memory can be implemented by a rotating magnetic device such a disk or a drum, charge coupled devices (CCD), random access memory (RAM) or bubble memory. The statistical arithmetic unit is part of the controller. It generates statistics (totals, averages) over the combined contents of the cell memories. The controller is designed to receive instructions in RAP machine format from a general purpose front-end host computer, decode them, broadcast control sequences to initiate cell execution, and pass retrieved or inserted items between the front end and RAP. Cells operate in parallel directly on data, and each instruction is executed within the cells. Cell inter-communication is provided for by priority polling. Each cell contains several logic units which perform tasks such as searching and comparing. The comparison units can test the contents of one item in the database against several search key items. The most important feature of RAP is its parallel processing capability which eliminates the need of inverted lists, B-trees or any other method for fast retrieval (Schus79). Another advantage is the use of a hardware mechanism to translate memory formats into user defined database files and records. This minimizes the need for specialized software. The front-end computer supports high level user functions, maintains communications via interactive terminals and translates various queries into

RAP instructions which in turn invoke several cell microcode instructions.

Relations can be represented by a table where the rows of the table represent a set of record occurrences (tuples) (Figure 7.2.2.1.2). Each relation and its occurrences are augmented by several special one bit items Mi called mark bits which can be set to 0 or 1 under user control or by the intermediate operations of other instructions (Schus79). These bits are used as a work area to indicate subsets of record occurrences so that the results of one instruction can be used in subsequent instructions. An extra mark bit called the delete flag is used to indicate deleted tuples to be ignored during instruction execution. The records of a relation can expand over cell memories but each call is allowed to store records from one relation only. A RAP device can therefore contain one large relation over N cells or N relations, one for each of N cells. A RAP relation is not quite relational as is defined by the relational model of a database. It allows duplicate records and their existence is not automatically detected.

Instruction format (Schus79)

The general format of a RAP instruction is:

<label><op-code><mark option>

[<object>:<qualification>][<parameters>]

label: optional symbolic instruction address.

op-code specifies the data manipulation operation e.g

	CELL NO.							
FORMAT & IDENTIFICATION	RELATION NAME							
	M1	M2	....	Mb	ITEM NAME	ITEM NAME	.....	ITEM NAME
RECORD (TUPLE) OCCURRENCIES UNORDERED	1	Ø	....	1	XXXX	YYYY	.....	9.99

Figure 7.2.2.1.2 RAP.2 logical data types (Schue79)

Select, Read-all, Save etc.

mark option: has the following significance and formats

1. Numm no marking is done.
2. Mark (bit specification) sets to '1' the mark bit specified in the bit specificatin of the qualified tuples for example inmark(m1m2m3).
3. Reset resets bits to 0. for example RESET(M1M2M3).

Object: has the following significance and format.

1. Rn (D1,D2,D3.. Ds) where Rn is a relation name and D1,D2,Ds is a list of data item names associated with relation Rn.
2. Cell(i) where i is the address of the ith cell

Qualification: has the following significance and format.

1. Null implies every tuple of the relation qualifies.
2. Q1Q2Q3Qn is the conjunction of a simple conditions.
3. Q1Q2Q3...Qn is the disjunction of simple conditions. Simple conditions may have the following format.
   1. <data item name><comparator><operand> where:
   A. Comparator is =, #, <, >, ≤, ≥.
   B. Operand is one of REG(i), <integer>, "<literal>" where REG(i) refers to the contents of the i-th controller register.
2. MKED (Mi): mark bit test Mi=1.

3. UNMKED (Mi): mark bit test Mi=0

4. CELL(i): call address is tested.

Parameters: depending on the instruction the parameter may have the following significance and formats.
<register list>,<work area>,<add,subtract,replace>, <cell list>, <format>.

Cell structure (Schus79)

Each cell consists of the elements described below:

1. The cell interface: interfaces the cell to the control and data buses and has a 16-bit relation name register. There are status states reflecting the results of the last pass in delete flags, M1,M2,M3,... Etc. The most significant bit of the status is always a '1' indicating the presence of a cell.

2. The synchronizer provides all the timing signals to the cell. It generates the read phase and write phase signals for the CCD memories. The cell logic operates at 10 MHZ.

3. The query analyzer determines whether a tuple satisfies the search qualification. It utilizes an 8-bit register to store the item number to be tested, a 32-bit shift register for an externally supplied constant, a 4-bit register indicating the selection of the unit, and the symbols <,=,> of the comparison and a serial comparator.

4. The i/o buffer provides temporary storage for retrieval and insertion. It consists of a 1KX16 RAM.

5. The arithmetic unit is independent and can be removed if not used. It is only required to perform arithmetic instructions such as Add, Sub, etc.

6. The update control executes the mask and reset instructions, writes new data supplied by the I/O buffer, and erases the track in order to create or delete selected tuples.

7. The output multiplexer assures that only one call at a time is in the READ state.

8. THE CCD memories provide the mass storage for a cell. The memory capacity is 1 megabit per cell.

A commercially available RAP would have capacity limitation due to the cost relative to the total database storage requirements. A cost effective system would consist of the following components.

- a front-end general purpose computer to interface with users to provide the operating systems functions, and high level language capability.
- one or more RAP processors to act as a file 'cache'.
- one or more conventional secondary memories.

The database file can be partitioned horizontally by storing certain records on RAP and others on disk, or vertically, storing clusters of data items on one device or the other. RAP is classified as a SIMD - single instruction multiple data stream machine. If we consider a RAP processor with 100 cells executing a query requiring only 10 cells, the other 90 cells will remain idle and therefore only the 10%

of its potential is used.

7.2.2.2 DIRECT (Figure 7.2.2.2.1) is a multiprocessor organization for supporting relational database systems and can be classified as a MIMD - multiple instruction multiple data stream architecture. DIRECT is a virtual machine and therefore there is no limit to the relation size. It is implemented using LSI-11/03 microprocessors using CCD memories which are searched in an associative manner.

System architecture. DIRECT consists of six main components:

1. Host processor.
2. Back-end controller.
3. Query processor.
4. CCD memory modules.
5. Interconnection matrix.
6. Mass storage.

The Back-end controller is a microprogrammable PDP 11/40 responsible for communicating with the host processor and controlling the query processors. When the back-end controller receives a query from the host, it determines the number of query processors required to execute this query (Dewit79.1).

Query processor: Each query processor is a PDP 11/03 with 28k words of memory. The function of each processor is to execute query packets assigned to them by the back end controller. Each processor associatively searches a subset

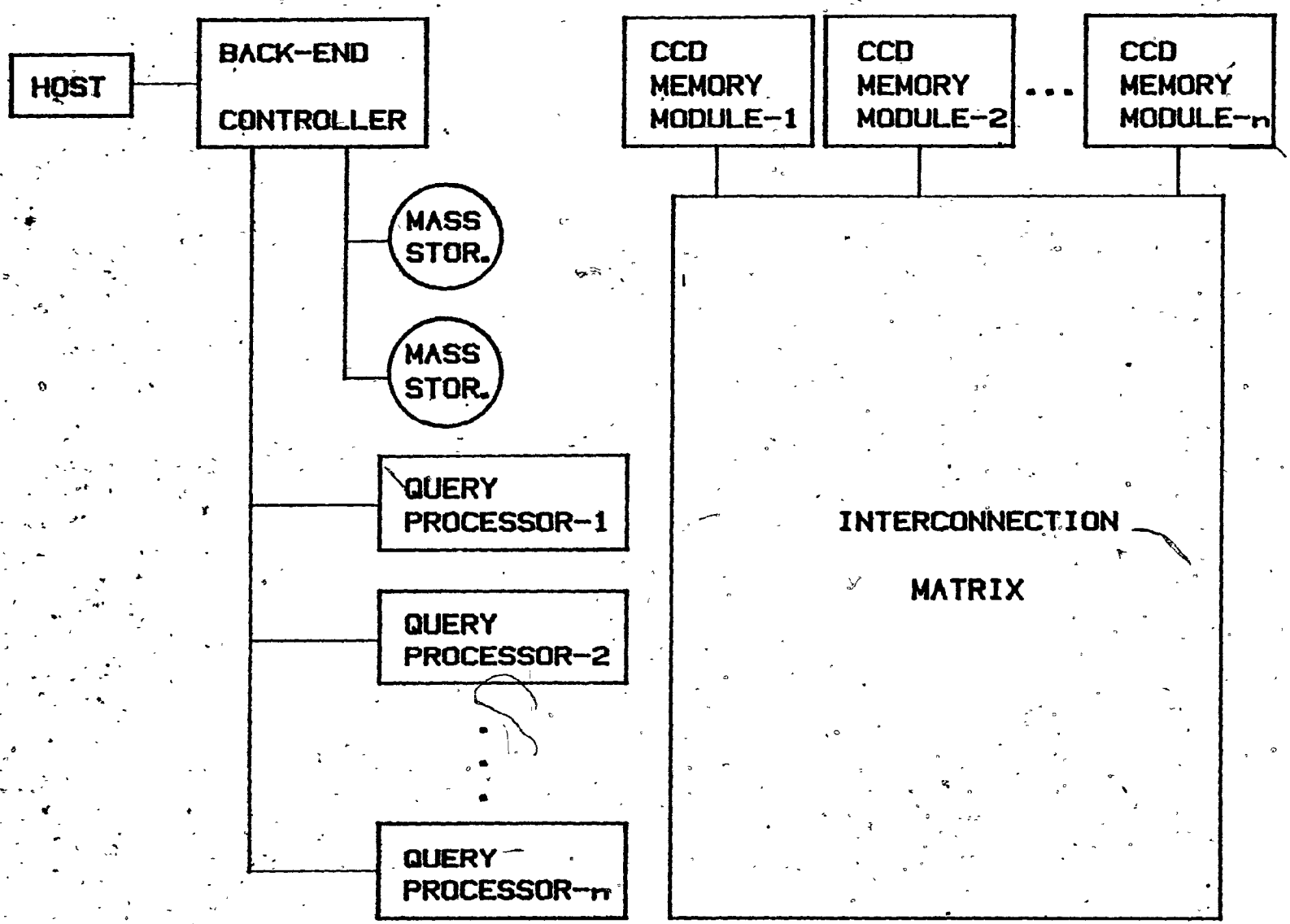


Figure 7.2.2.2.1 DIRECT system architecture (DeWit79.1).

of each relation referenced in the packet. When it finishes one page of a relation, it makes a request to the back-end controller for the next page (Dewit79.1).

CCD memory modules. Each associative memory is divided into fixed size pages of 16K bytes. The reasons choosing this size are the following:

1. Financial.
2. Size. If size is large then relations may fit on just one page. It also minimizes the amount of internal fragmentation when a relation does not fill all of the pages it occupies.

The interconnection matrix permits the following.

1. Query processors can rapidly switch between page frames containing pages of the same or different relations.
2. Two or more query processors can simultaneously search the same page of a relation.
3. All query processors can simultaneously access a page frame.

DIRECT does not use mask bits to indicate the tuples satisfying the search criterion of a query. Rather, tuples which satisfy the search criterion are written into a temporary relation in a page frame of the associative memory. Using mark bits may cause problems for DIRECT where two or more query processors executing different queries can access the same page of the same relation simultaneously. As far as the performance is concerned, DIRECT requires one disk revolution to extract the tuples satisfying the search

criterion unless the query processor's buffer is full, in which case a second revolution is necessary (Dewit79.1).

Instruction set: Each query processor has the following instructions.

RESTRICT:SELECT tuples from a relation based on boolean search condition.

PROJECT:ELIMINATE the specified columns of the relation.

JOIN:COMBINE two relations to form a third relation.

UNION:FORM the union of two relations.

DIFFERENCE:CALCULATE the set difference of two union compatible relations.

INTERSECTION:INTERSECT two union compatible relations.

CROSS-PRODUCT:Form the CROSS-PRODUCT of two relations.

MODIFY:MODIFY all tuples of a relation satisfying a specified boolean condition.

INSERT:INSERT a tuple into relation.

COMPRESS:COMPRESS a relation by removing tuples marked for deletion.

AGGREGATE OPERATORS:perform MAX, MIN, COUNT, and AVERAGE.

7.2.2.3 CAFS (Figure 7.2.2.3.1) CAFS (content addressable file store) consists of specialized hardware inserted between the disks and the host computer. The system is designed so that disk track heads can be multiplexed to the CAFS system. Data can be transferred to the host by bypassing the CAFS system. The system operates on twelve different data streams to allow only qualified tuples to

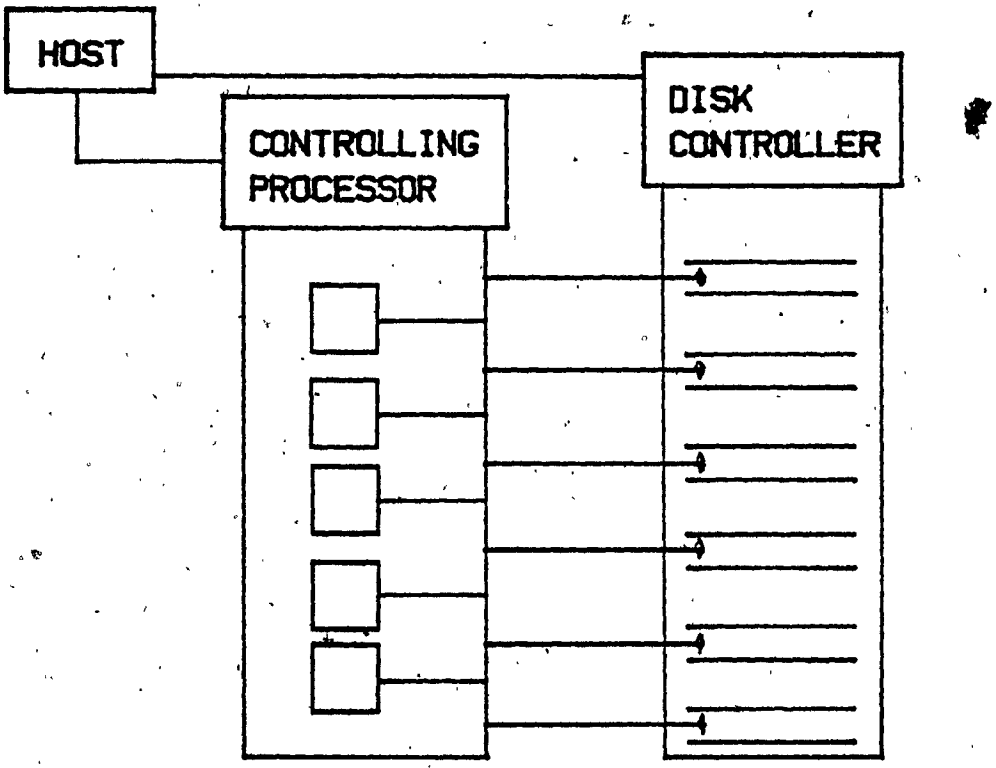


Figure 7.2.2.3.1 CAFS system architecture (DeWit80).

pass through to the host. In query processing, CAFS gets the search key from the host, and each track containing the relation is connected. Tuples qualifying the search value are passed to the host (Dewit80).

7.2.2.4 DBC (Figure 7.2.2.4.1)

DBC (Data Base Computer) was developped by the Ohio state University. It consists of several specialized processors which perform the database management function. The DBC uses moveable head disks and provides content addressability on a cylinder basis. The directory is kept in a separate memory for fast processing. DBC is a back-end machine capable of communicating with one or more hosts and supporting hierarchical, network, and relational models of database. It consists of 7 components and two loops of processors and memories.

The data loop (Hsiao79.2) consists of the database command processor (DBCCP), mass memory (MM), and security filter processor (SFP), and is used for storing and accessing the database and enforcing field level security.

The structure loop (Hsiao79.2) consists of the database command and control processor (DBCCP), the keyword transformation unit (KXU), the structure memory (SM), the structure memory information processor (SMIP) and the index transformation unit (IXU). It is used for limiting the mass memory search space, for determining the authorized records for access, and for clustering record received for insertion

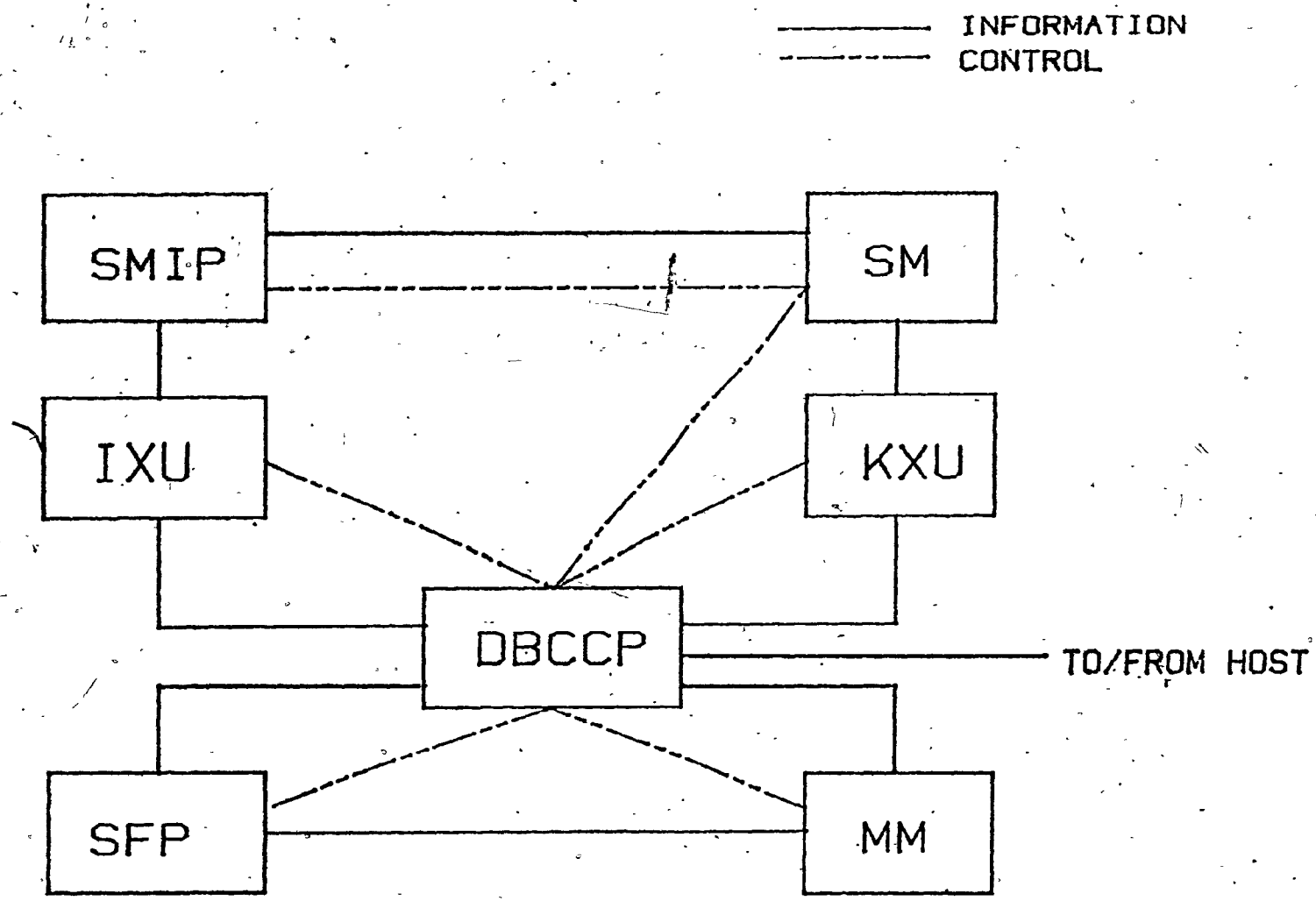


Figure 7.2.2.4.1 DBC system architecture (Heiao79.2)

into the database.

The mass memory (MM) (Hsiao79.2) consists of moving head disks modified to allow parallel track readout of an entire cylinder at each revolution. This feature allows rapid access to large blocks of data. The received data can be content addressed simultaneously by the track information processors (TIP's) during the readout revolution. As user requests for data transfer are seldom beyond one megabyte, this feature appears to be adequate. The mass memory information processor (MMIP) processes the information contained in the entire cylinder in one disk revolution. Every track of the cylinder has its own unit called track information processor (TIP) having some amount of buffer space. If we consider a disk with 40 tracks per cylinder, then the MMIP will contain 40 TIP's. The content addressable capability is achieved by transforming a query into a record having all the search identifiers, and comparing the track data against it. The query record is stored in sequential memory. Each track information processor (TIP) reads a record from the track, and the query record from the sequential memory carries out a bit by bit comparison of the two records. Records satisfying the search identifiers are sent to the security filter processor (SFP) for security check on these records. This processor in turn returns to the control processor (DBCCP) only those records allowed to be accessed by the inquiring user. Input requests are detected by the mass memory controller (MMC),

which directs the disk drive controller to position the read/write heads to the appropriate cylinder. When the cylinder is accessed, the MMC sends the requests one at a time to the mass memory information processor (MMIP). While the track information processors (TIP's) are executing a request, the MMC can request the drive controllers to position the read/write heads to another cylinder. Reading and writing is done in one revolution. Deletions require two revolutions and updates of records being increased in size require more than two revolutions.

The structure memory (SM) (Hsiao79.2) is the repository of the directories of the files in the database. For every keyword K designated for indexing there is an entry consisting of the keyword itself and a list of index terms. Index terms are composed of a cylinder number and a security atom. For each predicate, the SM determines all those keywords which satisfy the predicate. For each satisfying keyword, a set of indices is retrieved. These indices in turn are intersected for all predicates. The result of the intersection is a list of index terms for a given query conjunction. The list of index terms is compared by the DBCCP to the user's access privilege. Finally, a list of index terms with permitted access are forwarded to the mass memory (MM). In conventional databases, indices range from 1 to 10% of the size of the database. In DBC, indices are required for the cylinder level and are expected to constitute about 1% of the database size. Another important

requirement is that the SM should provide high search retrieval speed. A survey has indicated that CCD's with a random block access time of 100μs will cost about 50 mcents/bit. Bubble memories with 1 ms access time cost 10-20 mcents/bit. Electron beam addressable memories (EBAM) at a cost of 10-20 mcents/bit have been studied but their reliability is uncertain. Designers of the DBC have decided to employ either bubble memories or CCD's for the structure memory design.

The keyword transformation unit (KXU) (Hsiao79.2) encodes the keywords before they are sent to the structure memory. Each attribute in the database has a unique identifier. Information about them is stored in a table called the attribute information table. Each attribute contains information such as minimum, maximum, and values (numeric, floating point, alphanumeric, etc). Depending on the attributes, different hashing algorithms are used. They reside in the hash algorithm library.

The structure memory information processor (SMIP) performs intersections on the sets of index terms delivered by the structure memory SM. For example, consider a query conjunction q=P1@P2@P3... where each Pi is a predicate. Starting with P1, each index term satisfying the predicate is stored in the SMIP. The process is repeated until Pn is processed, and then the list of index terms is forwarded to the database command and control processor to be checked before being transmitted to the mass memory MM.

PARAMETER	DESCRIPTION	VALUE
A. DISK		
BSIZE	BLOCK SIZE	512 BYTES
DROT	DISK ROTATION TIME	.0167 SEC.
DAVAC	AVERAGE ACCESS TIME	.0300 SEC.
DRATE	DATA RATE TO HOST	.0012 SEC/BLK.
DREAD	CELL READ TIME	.0008 SEC/BLK.
DCYL	# OF 512-BYTE BLOCKS/CYL.	418 BLOCKS
B. CCD		
CSIZE	SIZE OF CCD PAGE	16K BYTES
CSCAN	PAGE SCAN TIME	.012 SEC.
C. HOST TIME		**BEST CASE - WORST CASE**
HOVIO	HOST OVERHEAD I/O TIME	- .0300 SEC.
HOVCPU	HOST OVERHEAD CPU TIME	.006 - .1600 SEC
HBCOM	HOST CPU TIME TO COMMUNICATE WITH BACK-END MACHINE	.006 - .0300 SEC
HDP	HOST TIME FOR PRINTING E.T.C	QUERY AND DATABASE MACHINE DEPENDENT

INGRES: the time required to execute the query is defined as follows:

QTIME=HOVIO+HOVCPP+HDPIO+HDPCPU

The worst case will be when the three pages reside on separate cylinders, requiring one access per reference. The best case will be when all three references reside on the same track. Therefore HDPIO=.0336 for the best case and .080 for the worst case. The host data processing cpu time is defined as HDPCPU=.06 seconds. The total time required is QTIME=.13-.34 seconds.

Associative disks and CAFS

the time required to execute the query is defined as:

QTIME=HOVIO+HOVCPU+BHCOM+DAVAC+N*DROT.

The system requires one disk access to position the read/write heads at the appropriate cylinder. Since data resides on a single cylinder, and by assumption cell processors operate at the rotational speed of the storage media, it will need one revolution of the disk (n=1) to process the query. Therefore QTIME=.089-267 seconds.

CASSM

The time required to execute the query is defined as:

QTIME=HOVIO+HOFCPU+BHCOM+DAVAC+N*DROT

CASSM will require the following number of rotations to execute the query.

1 rotation to mark all tuples for the relation.

1 rotation to find the pointer to the query qualifier (JS)

1 rotation to transfer all marked tuples satisfying the 'day' attribute.

1 rotation to transfer all marked tuples satisfying the 'hour' attribute .

Five rotations are therefore required to evaluate the query and QTIME=.156-.334 seconds.

DBC

The time required to execute the query is defined as QTIME=HOVIO+HOVCPU+HBCOM+DCYL+DAVAC+DROT.

In DBC, cylinder indices reside in RAM memory, thus requiring time to search and retrieve them. The time for this analysis is assumed to be DCYL=.006 seconds and therefore QTIME=.095-.272

DIRECT

The time required to execute the query is defined as QTIME=HOVIO+HOVCPU+HBCOM+DPIO+N*CSCAN

where CSCAN is the CCD page scan time. DPIO is one disk seek plus the transfer rate of all pages.

DPIO=(0-(DAVAC+274*DREAD))=(0-.249) seconds

Nine data cells are required to store the relation and there are only 8 cell processors available. Two to three scan times are sufficient, giving

QTIME=HOVIO+HOVBPU+HBCOM+(2*CSCAN-3*CSCAN)+HDPIO+HDPIN =.066-.490 seconds.

RAP The time required to execute the query is defined as QTIME=HOVIO+HOVCPU+HBCOM+DPIO+N*CSCAN.

The relation requires only Q data cells to be stored. If the relation is already in the cache, then DPIO=0, otherwise it is the same as DIRECT. The number n of cell scans is determined as follows:

1 scan to mark qualifying tuples in 9 processor cells.

1 scan to transmit marked tuples to the host.

Thus:

QTIME=HOVIO+HOVCPU+HBCOM+DPIO+2*CSCAN=.066 - .493

Conclusions: Single relation queries

RAP and DIRECT have the slowest worst-case times because they must serially search the entire relation, while INGRES maintains it as Q hashed relation (Figure 7.3.1).

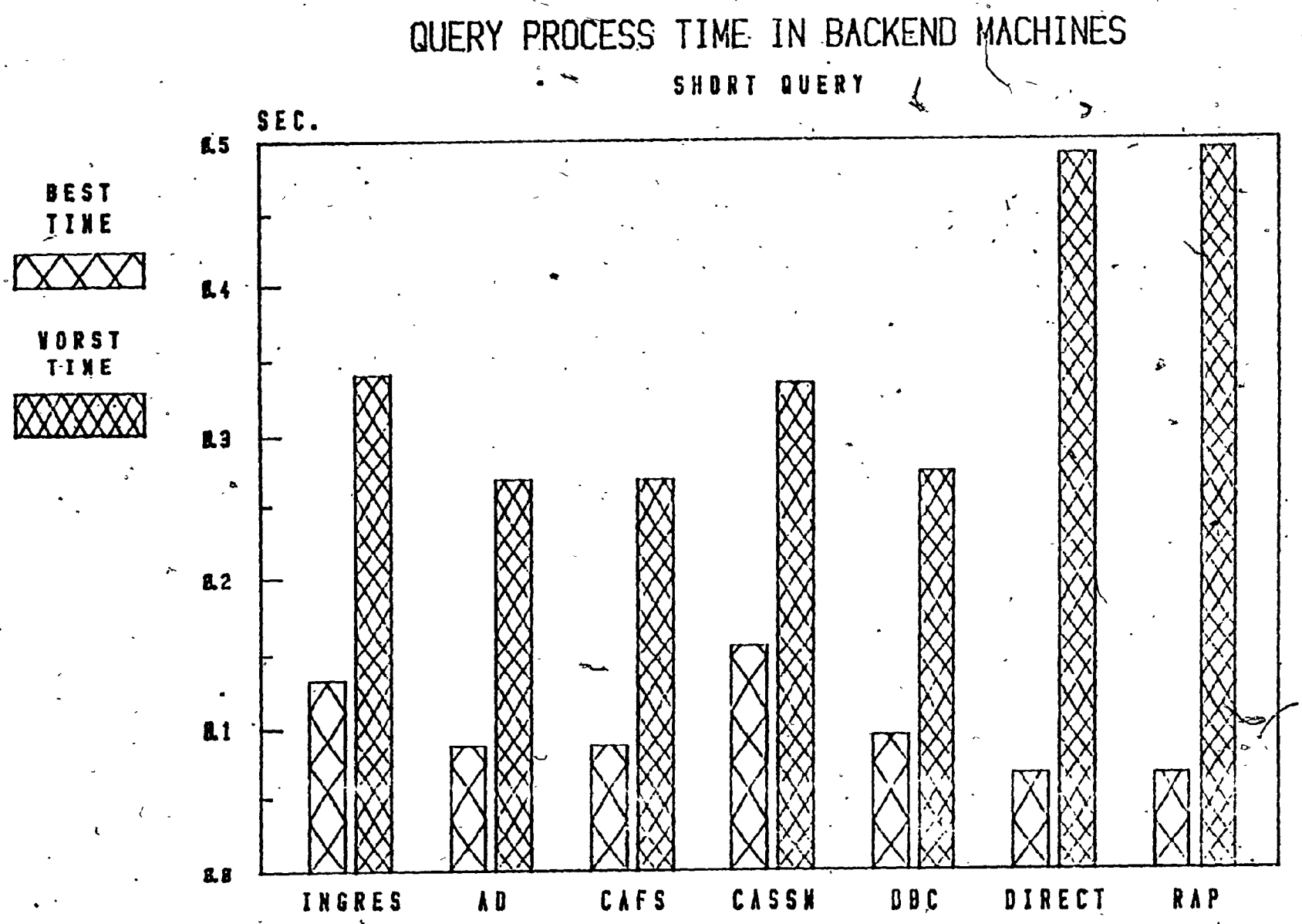


Figure 7.3.1 Short Query in Backend machines.

Following the same steps and constructing the formulae appropriately, QTIME was calculated for multi relation queries giving the following results:

INGRES QTIME = 29.74 - 37.55 sec.

ASSOCIATIVE DISKS QTIME = 5.04 - 5.90 sec

DBC QTIME = 5.04 - 5.90 sec

CASSM QTIME = 15.50 - 15.83 sec.

DIRECT QTIME = 3.29 - 4.07 sec.

RAP QTIME = 14.48 - 14.81 sec.

Conclusions: Multi relation queries

INGRES shows the worst time of all because it is a software package. CAFS and CASSM are relatively slow because some of the logical operations such as the join, must be performed at the host. RAP's performance depends on whether the relation can be completely stored in the CCD cells (Figure 7.3.2).

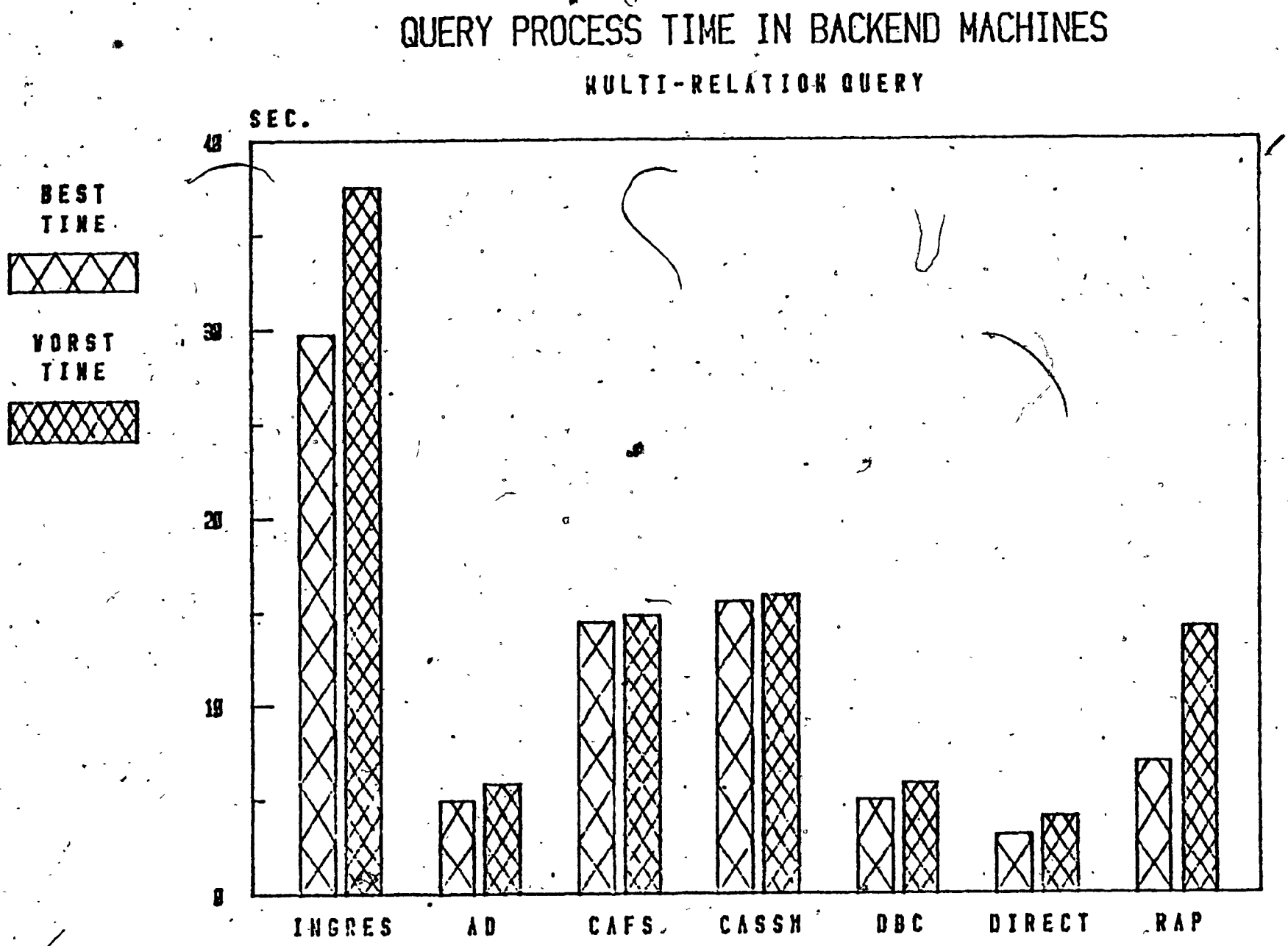


Figure 7.3.2 Multi-Relation Query in Backend machines.

8. PROPOSED MODEL

Distribution of a database may be made according to function or according to data. The back-end networks and database processors discussed in the previous sections are representatives of distribution by function, and are particularly effective for large, centralized systems. Distribution by data implies that all the essential functions of the database are present on each of the computers of the system. The organization is appropriate to geographically distributed systems, and to local systems composed of arrays of identical computers. The general problem of distribution by data should include both horizontally partitioned as well as replicated data sets. These two cases cover the problem both of large databases and high reliability systems, both of which are fundamental motivations for the implementation of this type of distribution. The model proposed in this section presents a structure and file organization for this type of system.

Assume the network of computers [N=1,2,3,4] of Figure 8.1 Assume a relational DDBMS consisting of the following relations [R-DDBMS=A,B,C,D,E,F]
Assume that relations E and F originate only at nodes 2, and 3 respectively while the others are common. The R-DDBMS will be distributed across the network by partitioning the common relations so that each node retains only those tuples

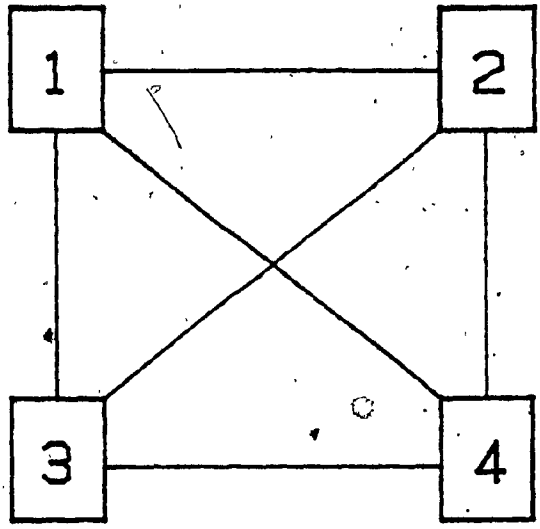


Figure 8.1 Distributed network N.

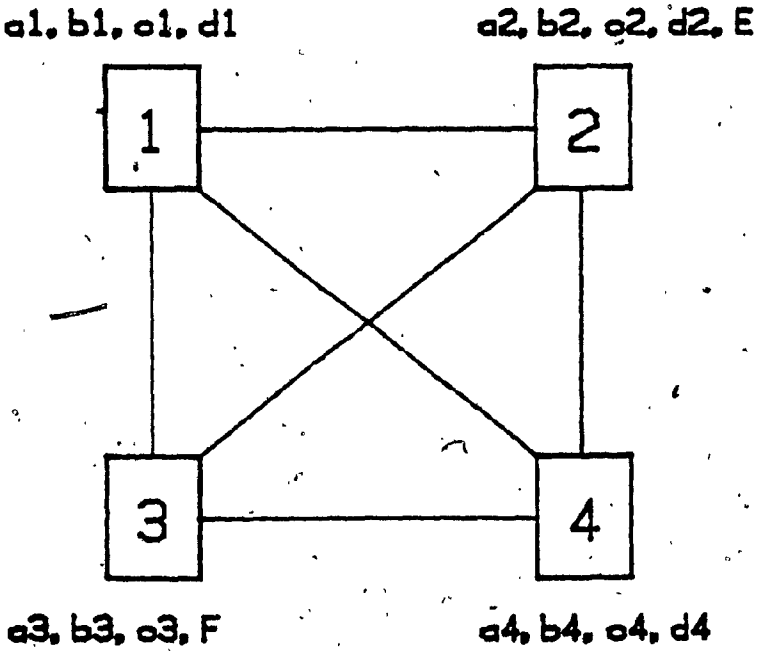


Figure 8.2 The R=DDBMS partitioned across the network N.

that belong to it. All other relations that are unique to any node remain at the point of origin (or where they are most frequently accessed), and are not partitioned (Figure 8.2). The R-DDBMS is distributed over the 4 nodes, and from the global point of view appears as:

R-DDBMS = [A,B,C,D,E,F] where

A = [a1a2a3a4]

B = [b1b2b3b4]

C = [c1c2c3c4]

D = [d1d2d3d4]

One may observe that updating is relatively simple when no replicates exist. The system backup of each node is simpler and communications traffic is at a minimum since most of the operations are performed locally. Finally, should a node become inoperative, it will not affect the network except that its information will not be available to others.

8.1 MODEL ARCHITECTURE

In this architecture the backup is the most important operation. The system should be able to achieve a backup synchronization so that all nodes operate under the same generation number (#) of the R-DDBMS. For this reason the processes should be able to detect that the information transferred between nodes is of the same generation #. Locally, the generation # is almost transparent, and the frequency of backup is determined by the database administrator (DBA). The system allows three types of relations: partitioned, unique, and replicates. The type of the relation is determined by the DBA at the time of the application design. A relation is partitioned if is common to more than one node and each node represents a portion of it. An example of this organization is a corporation's inventory relation where each node represents a part of the inventory local to the operations of its corresponding site. Unique relations are those belonging to one only node. Replicate relations are ones that are common to all nodes, and where exact copies may reside in more than one node. In each node the location of the appropriate relation is obtained by the use of the system directories.

The system directories are the LOCAL RELATION DIRECTORY (LRD), the PARTITIONED RELATION DIRECTORY (PRD), and the REPLICATED RELATION DIRECTORY (RRD) (Figure 8.1.1). The LRD maintains the names of all local files (relations) relating

LRD

RELATION NAME	RELATION ADDRESS

PRD

RELATION NAME	LOCAL ADDRESS	REMOTE NODE ID
		NODE-1 .... NODE-n

RRD

RELATION NAME	LOCAL ADDRESS	REMOTE NODE ID
		NODE-1 .... NODE-n

Figure 8.1.1 R-DDBMS System Directories.

to this node. The PRD maintains the names of partitioned relations and where they reside. These relations may be partitioned among several nodes of the network or among all of them. The RRD maintains the names of replicated relations and the associated node-id's. In this case a relation may be replicated among several nodes or among all of them. When a relation is needed, a relation fault occurs and the appropriate directory is searched to locate the required relation. When a relation resides in another node a request is sent to that node and the process should fall into a wait state (with a possible timeout). Each node maintains a set of these files that describe the schema of R-DDBMS.

- the root file
- the relation file
- the index file

These are described in the following:

Root file:

The root file contains the information about relation names, primary key names and passwords. It also contains additional information pertaining to the relation partitioning.

The Root file is illustrated by Figure 8.1.2.

File name : ROOT.<DBNAME>.<NODE-ID>.

Levels indicate the password associated to each level.

Global security clearance is the password applied globally

$$$. PASSWORD. INFO

LEVEL #	PASSWORD

$$$. DISTRIBUTED. PASSWORD. INFO

GLOBAL SECURITY CLEARANCE	LOCAL SECURITY CLEARANCE

$$$. RELATION. INFO

RELATION NAME	PRIMARY KEY NAME	PARTITIONED FLAG

$$$. KEY. DESCRIPTION

KEY NAME	SIZE	TYPE	RELATION NAME	PARTITIONED FLAG

$$$$. EOD

Figure 8.1.2 'ROOT' file.

for distributed access.

Local security clearance is the password applied locally to access local relations only.

Relation name is the name of the relation that belongs to this rdbms.

Relation primary key is the primary key associated with this relation.

Partitioned flag is a flag which indicates whether the relation is partitioned among other nodes, is unique, or is replicated.

Key name

Size indicates the number of bytes that the key occupies. It is applicable only to alphanumeric types.

Type indicates whether the item is Integer, Real, Alpha. (I,R,A).

Relation name is the relation name to which this key belongs.

Note: This design allows two or more relations to have keys with the same key-name.

Relation file:

the Relation file describes the individual relation in detail. Each column is described as to which security level is allowed to read or write it, its size, type, name, and value.

The Relation file is illustrated by Figure 8.1.3.

File name : REL.<DBNAME>.<REL-NAME>.

$$$. GENERAL INFO

DBNAME	NEXT AVAIL. REC.	REL. DEGREE	BACKUP GENER. #

$$$. COLUMN. INFO

SIZE	TYPE	R-LEVEL	W-LEVEL	COL. NAME	COL. SUB-NAME	CHAIN	DISL.

$$$. TUPLES

DELETE	LOCKED	TO COL.	VALUE	FWRD	BKWRD	COL.	VALUE	FWRD	BKWRD
.....									
.....									
.....									
.....									
.....									
.....									

$$$$. EOD

Figure 8.1.3 The 'RELATION' file.

Dbname is the database name.

Next rec. avail is the address of the next available record in the relation.Rel.degree is the number of columns in this relation.

Backup gener.# is the latest backup number.

Size is the size in bytes applicable only to alphanumeric items.

Type indicates whether the item is integer, real, alphanumeric (I,R,A).

R-level indicates the security level that can read this column.

W-level indicates the security level that can update the column.

Col.name is the name of the column.

Col.sub-name is the column sub-name.

Chain indicates that chaining of similar tuples is desired (for secondary) keys.

Displ. is the position of the column relative to the begining of the tuple.

Delete is the delete flag which is set to indicate deletion of the tuple.

Locked to is the process id that has locked the tuple.

Col.value is the actual value of a particular column.

Fwrd is the pointer to the next tuple in chain if chaining was specified.

Bkwrd is the pointer to the previous tuple in chain if chaining was specified.

Note: pointers are present only if chaining option was selected.

Index file:

File name : INDEX.<DBNAME>.<REL-NAME>.<COL-NAME>.

The index file contains addresses of primary keys.

Program run table (PRT): (FIGURE 8.1.4)

The PRT table holds information about the user program and the database relations. Each relation is represented by an entry in the PRT. When a "locate" command is issued (see 8.2), depending on the search condition, the column number and the address of the first record that satisfies the search value is entered in relation's entry. Subsequent "get" commands will use the record address to retrieve all the records satisfying the search condition. The PRT can be seen as the interface between the user program and the database system. It reflects all error codes occurring in the database system during the execution of a command by posting the error code to the status flag. The PRT contains the program-id and node-id so that in certain emergency cases the PRT can be saved and the process resumes at later time.

System modules

The organization of the proposed system is illustrated by Figure 8.1.5. The system modules shown are resident on each

PRT

STATUS	PROGRAM ID	NODE ID	GLOBAL SECUR.	LOCAL SECUR.	READ LEVEL	WRITE LEVEL

RELATION NAME	NODE ID	GENER. #	REL. STATUS	COLUMN #	RECORD ADDR.	COLUMN #	RECORD .. ADDR.
....	..	...	...	....	....	....	....
....	..	...	...	....	....	....	....
....	..	...	...	....	....	....	....

Figure 8.1.4 The Program Run Table.

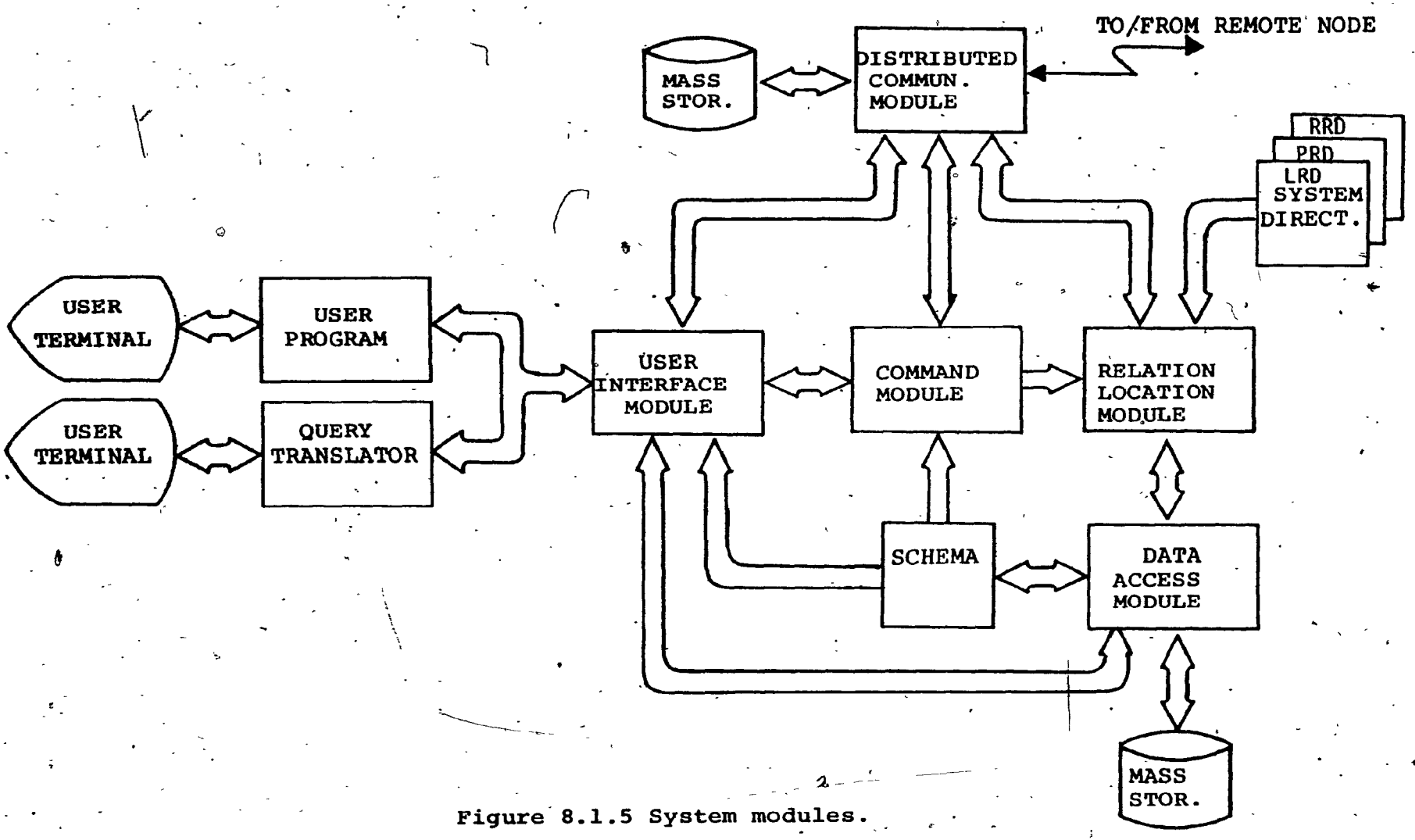


Figure 8.1.5 System modules.

nodes of the network. Intercommunication with the nodes is maintained by the distributed communication module. Their operation is described as follows:

User program (up) is a program written in high level language such as Cobol, Fortran, or Assembler. User programs issue instructions explicity to the database system (see 8.2 R-DDBMS commands).

Query translator (QR) is the module for the non data processing user where an English like command is translated in one or more database commands. This module also may provide some reporting capability so that simple reports may be obtained without programming effort.

User interface module (UIM) is the interface between the database and the outside world. At log in time, it verifies passwords and passes the command to the command module. When the information returns to the user, it notifies the process, by updating its PRT table, that the request has been completed and the requested data have been stored in the indicated place (work buffer or file).

Command module (CM) receives the command in explicit form (see R-DDBMS commands) and verifies its correctness by accessing the schema. Once the request has been evaluated, the CM may request the UIM to provide some additional information from program's PRT (program run table). When the user command explicitly requests remote access, the CM will issue to DCM the request for the node that is needed.

This is achieved by supplying the node-id in the appropriate field of the R-DDBMS command. Finally when the node-id is blank, the CM will request the relation location module (RLM) to take over.

Relation location module (RLM) searches the local relation directory LRD to locate the required relation or index. When the search ends successfully, it passes control to the data access module (DAM) to finally access the data. When the search fails, the RLM will search the distributed system directories (PRD, RRD) to find the node where the requested relation or index is located. Having completed the search, it will request the distributed communications module (DCM) to establish communication with the requested node and request transfer of the data. The RLM may also perform a search due to a distributed call through the DCM.

Data access module (DAM) will read or write the data from the local database to the location indicated by the RLM and report the success of the operation to the UIM. In the case of a distributed call, it reports back to the RLM which in turn reports back to the DCM. One of its privileged functions is to update the schema when necessary, but this can only be done under a privileged password.

This module can use any one of the access methods described earlier in this study. The chaining feature of the model imposes the use of double linked lists for all related data entries. To locate the head of a list, the

following access methods can be used:

Hashing: Any hashing algorithm can be used. If it is implemented at the hardware level, the multiplicative method may improve operation by generating less synonyms and thus executing much faster.

Inverted file: As it was mentioned before inversion simplifies the DDL design but adds significant overhead in processing the record.

Tree structures: The addresses can be contained in the nodes of the tree which is allowed to expand. Tree structures require less maintainance compared to the inverted files but they may require more storage.

The implementation of access methods is beyond the scope of this study and depends on the available software and hardware. Implementation of a hashing algorithm in the hardware, however, would provide faster access time and would have less storage requirements.

Distributed command module (DCM) interfaces the node with the network. It routes the R-DDBMS commands to the network and to the RLM from the network. Incoming data from the network are stored in mass storage in the indicated place (work buffer or file) and the success of the operation is reported to the UIM. In response to a distributed call, the DAM will create work areas holding the data. When the RLM indicates the end of an operation the DCM will transfer

those work areas into the network. The DCM is expected to recognize when it is called and respond to such a call.

8.2 R-DDBMS COMMANDS

The R-DDBMS command set includes commands for searching, retrieving and updating tuples or relations. The commands can be issued by the application programs or directly by the Query subsystem. The details of these commands are as follows:

<ATTR.>:= Relation's attribute name.

<REL-OP>:= AND|OR|NOT|<|>|≠

<SEARCH COND.>:= <ATTR.><REL-OP><ATTR.>|

<SEARCH COND.><REL-OP><SEARCH COND.>

LOCATE (NODE-ID),(REL-NAME)((SEARCH CONDITION))[,(LOCK)]

This is used prior to an addition or an update. The LOCATE will find the tuple according to the search condition and will return its address in the relation entry of the program run table (PRT). When 'LOCK' is requested, the tuple(s) is locked to this program. This option is used prior to an update or delete and is described in detail in chapter 8.3.

GETF (NODE-ID),(REL-NAME),(WORKBUFFER)[,<ATTR.>]ⁿ

GETB (NODE-ID),(REL-NAME),(WORKBUFFER)[,<ATTR.>]ⁿ

The GET instruction will retrieve the tuple that has been located by the LOCATE instruction, and it will return into the 'workbuffer' the entire tuple or selected attributes. If the chaining option is specified for the attribute, the GETF will always access forward the next tuple in chain and the GETB will always access the preceeding tuple in chain.

GETR (NODE-ID),(REL-NAME),(FILENAME)

The GETR transfers the entire relation into the file designated by the 'filename'.

All the GET commands copy the required data into a user supplied space one tuple at a time. These commands should be used in local access only to avoid excessive telecommunications overhead.

SELECT (NODE-ID),(REL-NAME)(SEARCH COND.),
,(filename),[,<attr>]^{n}

The SELECT copies the tuples or selected attributes that satisfy the search condition and creates a new relation into the 'filename'. This command must be used instead of GET when a distributed call takes place. It will reduce the telecommunications traffic.

PROJECT (NODE-ID),(REL-NAME),(NEW-REL-NAME)[,<ATTR>]^{n}

the PROJECT will select the indicated attributes of a relation and produce a new relation.

JOIN (NODE-ID),(REL-NAME1),(JOINING CONDITION),(REL-NAME2)
(NEW-REL)

The joining condition is a boolean expression of comparisons (=, ≠, <, >, <=, >=) between columns in REL-NAME1 and REL-NAME2. The effect of this instruction is to create a

new relation consisting of tuples formed by concatenating tuples from REL-NAME1 with tuples from REL-NAME2 where the values in the tuples satisfy the joining condition.

UNION (NODE-ID),(REL-NAME1)(ATTR-PAIR-LIST),(REL-NAME2), (NEW-REL)

The ATTR-PAIR-LIST is a list of pairs consisting of an attribute name from REL-NAME1 and an attribute name from REL-NAME2. The effect of this instruction is a creation of a new relation NEW-REL consisting of all the tuples of both relations with any duplicates removed and the attributes ordered in a sequence given in the ATTR-PAIR-LIST.

INTERSECT (NODE-ID),(REL-NAME1),(ATTR-PAIR-LIST), (REL-NAME2),(NEW-REL)

The result of intersecting two relations is a new relation (NEW-REL) consisting of all those tuples appearing in both relations and the attributes ordered in a sequence given in the ATTR-PAIR-LIST.

DIFFERENCE (NODE-ID),(REL-NAME1),(ATTR-PAIR-LIST), (REL-NAME2),(NEW-REL)

The result of differencing two relations is a new relation (NEW-REL) consisting of tuples that are in the first relation (REL-NAME1) and not in the second relation (REL-NAME2) and attributes ordered in sequence given in the ATTR-PAIR-LIST.

ADD (NODE-ID),(REL-NAME),(WORKBUFFER)

The ADD command adds the tuple contained in 'workbuffer' into the relation indicated by the 'Rel-name'.

UPDATE (NODE-ID),(REL-NAME),(ATTR='VALUE')[,(ATTR='VALUE')]$_0^n$

This will update the attributes of the current tuple with the assigned values. The update will not allow modifications of key-attributes and will abort the command if this is attempted. The update, at the end of the operation will unlock the tuple that has been locked by the LOCATE command.

In case of a replicate relation all the copies must be updated virtually at the same time. The update synchronization is necessary to maintain the data integrity. In chapter 8.3 a few update synchronization methods are presented.

DELETE (NODE-ID),(REL-NAME),(TUPLE\ALL)

This command will delete the tuple(s) located by the LOCATE command. If the option TUPLE is used, only that particular one is deleted. If the option ALL is used, all the tuples in chain are deleted. This command will set the delete flag of the tuple to reflect the logical deletion. Deleted tuples are skipped by the GET and SELECT commands.

Deletion is a type of update and therefore is of the same importance with regard to replicate relations.

COLINS (REL-NAME)(COL-NAME='NAME',COL-SUBNAME='NAME' [,SIZE=XX][,TYPE=X][,CHAIN])

This command inserts one column to a relation. The size is optional because when TYPE=R (real number), the size is always fixed. The CHAIN is used only if this will be used as key item.

COLDEL (REL-NAME),(COL-NAME)

This command deletes one column from a relation. In case that the column is used as key item the operation is rejected and thus the request is aborted.

COLUPD (REL-NAME),[(OLDNAME='NAME'),(NEWNAME='NAME)]" [(OLDSUBNAME='NAME'),(NEWSUBNAME='NAME')]"

This command updates the literal information of a column.

The commands COLINS, COLDEL, COLUPD, affect directly the ROOT file and the header information of the relation. These commands operate only locally and must be used only when no other process uses the database resources. Each one generates a schema update and requests to all nodes of the network, and they are issued only by privileged users such as the DBA. The COLINS especially causes the relation file to be reformatted in order to accommodate the new column.

The COLDEL on the other hand, will delete the entry from the Column.info of the relation file for that column. In general, all the commands perform the housekeeping functions such as maintaining record pointers where applicable.

8.3 PROCESS SYNCHRONIZATION

The proposed model allows replicate relations as well as partitioned relations. A partition of a relation resides in the site where it belongs and is accessed the most. Replicate relations reside in more than one site of the network and at any given time they should be identical. The model allows locking to be affected at the lowest level of the structure, the TUPLE. Locking is used only in case of updating to ensure that only one process will update the tuple at any given time. Where locks are used there is a possibility of deadlocks (deadly embrace) which must be detected and corrected. The system may maintain a state graph (Menas79) consisting of all presently pending requests in order to detect a cycle (deadlock) (Stone79). Once the deadlock has been detected, recovery procedures must be initiated and resolve the situation. Another approach is to employ a centralized locking mechanism and every update request will be forwarded to this node. The major disadvantage is that all sites depend on one central node, and should this node go down, the entire system would halt. Partitioned relations, being unique to the site, do not represent any problem in process synchronization since the chance of a site updating a remote partitioned relation is minimal. The great concern in updating replicate relations is the update synchronization of the transactions so that the database consistency is preserved while excessive

overhead in propagating control information among the nodes of the DDBMS is avoided. The problem of DDBMS update synchronization has been studied by numerous authors. (Alsbe76) in multi-copy resiliency techniques, proposes a 'primary site' along with 'backup sites' to control all the update requests. Depending on the number of service hosts and the average host down time/day, two tables are presented for the 'Service down time' (figure 8.3.1).

This approach appears to be very good and very reliable for a simple database. In more complex situations, the method cannot avoid the need for global database locking mechanism (Rothn78). Another method, different from the preceeding, tries to avoid global locking whenever possible. This methodology has been implemented in the SDD-1 DBMS (Berns78). Each site applies the updates and then sends them to the other nodes having the same copy of data. The consistency of database copies is achieved by the use of 'timestamps'. When a data item is modified, it is stamped with the timestamp of the updating transaction. However the clocks at different sites run at different times and therefore must be synchronized. (Thoma78) introduced the 'majority concensus algorithm' where an update request is either broadcast to all participating nodes, or propagated as a daisy chain. Each node 'votes' to accept or reject the update request. If the majority of nodes accept the request, then the database update will be applied. The system uses a timestamp so that no two update requests

# OF SERVICE HOSTS	SERVICE DOWN TIME
2	3.8 HRS/DAY
3	28 MIN/DAY
4	3 MIN/DAY
5	19 SEC/DAY
6	2 SEC/DAY
7	6 SEC/MONTH
8	7 SEC/YEAR
9	1 MIN/CENTURY
10	6 SEC/CENTURY

HOST DOWN TIME 2 HRS/DAY

# OF SERVICE HOSTS	SERVICE DOWN TIME
2	2 HRS/DAY
3	7 MIN/DAY
4	24 SEC/DAY
5	1 SEC/DAY
6	23 SEC/YEAR
7	1 SEC/YEAR
8	5 SEC/CENTURY
9	2 SEC/THOUSAND YEARS
10	1 SEC/THOUSAND YEARS

HOST DOWN TIME 1 HRS/DAY

Figure 8.3.1 Service down time (Alebe76).

should have the same timestamp, and successively generated timestamp should increase. This method reduces communications volumes but it introduces lengthy communication delays (Rothn78).

The proposed model may adopt any of the above mentioned algorithms. However the algorithm proposed by (Menas79) appears to be the best candidate for this model. Implementation of such algorithm in INGRES (Stone79) proved to be successful. The algorithm may maintain state graphs at both the local level and the global level to detect deadlocks. Special procedures must be provided to resolve the deadlocks when they occur.

8.4 SYSTEM OPERATION

As mentioned earlier, the proposed model supports partitioned as well replicate relations. A banking system may be one of the applications requiring the features of this model. Transactions generated against the account relation originate at each node and they are unique to that node. The interest rates, in contrast, are common to all nodes and therefore may be replicated. Using sample data the relations of the Figure 8.4.1 may be required.

When the R-DDBMS has processed the information, the files pertaining to this application will be created. Using FILE-TYPE.DBNAME.REL-NAME.KEY-NAME to name the files, these would be:

- ROOT.DBBANK
- REL.DBBANK.CUSTOMER
- REL.DBBANK.ACCOUNTS
- REL.DBBANK.LOAN
- REL.DBBANK.TRANSACTIONS
- REL.DBBANK.INTEREST-TYPE
- INDEX.DBBANK.LOAN.CUSTOMER-NUMBER
- INDEX.DBBANK.LOAN.LOAN-AMOUNT
- INDEX.DBBANK.TRANSACTIONS.ACCOUNT-NUMBER
- INDEX.DBBANK.ACCOUNTS.ACCOUNT-NUMBER
- INDEX.DBBANK.ACCOUNTS.ACC-TYPE
- INDEX.DBBANK.CUSTOMER.CUSTOMER-NUMBER

ACCOUNTS RELATION

ACCOUNT NUMBER	CUSTOMER NUMBER	ACC-TYPE	BALANCE ($)
1-0001	1-100	SAV	1000.00
1-0002	1-100	SAV	5000.00
1-0003	1-200	CHK	300.57
1-0004	4-150	SAV	10000.00
1-0005	3-400	CHK	5700.00

CUSTOMER RELATION

CUSTOMER NUMBER	CUSTOMER NAME	ADDRESS	MISC INFO.
1-100	J. J SMITH	MONTREAL	.....
1-200	K. P FREY	MONTREAL	.....

Figure 8.4.1(a) Banking system's relations

TRANSACTIONS RELATION

ACCOUNT NUMBER	TYPE OF TRANSACTION	DATE	AMOUNT ($)
1-001	DEPOSIT (DP)	110880	500.00
1-004	DEPOSIT (DP)	110880	550.00
1-003	WITHDRAW (WD)	120880	100.00
1-001	WITHDRAW (WD)	150880	25.00

LOAN RELATION

CUSTOMER NUMBER	LOAN AMOUNT	INTEREST RATE	MISC INFO.
1-100	10000.00	.1375	HOME LOAN
1-200	20000.00	.1275	HOME LOAN
1-210	5000.00	.1375	CAR LOAN
1-100	5000.00	.1400	VAC. LOAN

Figure 8.4.1(b) Banking system's relations

INTEREST RELATION

INTEREST TYPE	INTEREST RATE
CHK	.0975
SAV	.1100
TRM	.1300

Figure 8.4.1(c) Banking system's relations

- INDEX.DBBANK.CUSTOMER.MISC-INFO

The files prefixed with "INDEX" always contain the address of the tuple that is the head of the list. The Root and relation file structure is presented in the Figures 8.4.2, 8.4.3, 8.4.4, 8.4.5, 8.4.6 and 8.4.7.

Considering the previously mentioned relations, transactions would generate the following instructions:
Assume that there is transaction to be added to the transactions relation. The user application program would issue first a command to verify the account existence, and then upon a positive reply would issue the addition request. These commands would be:

LOCATE NODE-ID,ACCOUNTS(ACCOUNT='1-002'
AND ACC-TYPE='SAV')

This command will post the number 10 in record address of the PRT and set both the status and rel-status to reflect the success of the operation. After the data are evaluated, the user program would proceed with the addition request as follows:

ADD NODE-ID,TRANSACTIONS,WORK

This command will add the tuple that is contained in the buffer 'work' and then housekeeping routines will record the pointers where applicable. In a banking system a frequent operation is to update the customer's account booklet. It is therefore essential to be able to retrieve all non posted transactions. Earlier in the R-DDBMS command description it

$$$. PASSWORD. INFO

LEVEL #	PASSWORD
1	TELLER.PASS
2	SUPERV.PASS
15	DBA.PASS

$$$. DISTRIBUTED. PASSWORD. INFO

GLOBAL SECURITY CLEARANCE	LOCAL SECURITY CLEARANCE
GLOBAL.DBMS.PASSWORD	LOCAL.DBMS.PASSWORD

$$$. RELATION. INFO

RELATION NAME	PRIMARY KEY NAME	PARTITIONED FLAG
ACCOUNTS	ACCOUNT	YES
CUSTOMER	CUSTOMER	YES
TRANSACTIONS		YES
INTEREST	TYPE	NO
LOAN	NUMBER	YES

$$$. KEY. DESCRIPTION

KEY NAME	SIZE	TYPE	RELATION NAME	PARTITIONED FLAG
ACCOUNT	6	A	ACCOUNTS	YES
CUSTOMER	6	A	CUSTOMER	YES
ACC-TYPE	4	A	ACCOUNTS	YES
MISC	30	A	CUSTOMER	YES
ACCOUNT	6	A	TRANSACTIONS	YES
TYPE	3	A	INTERESTS	NO
NUMBER	6	A	LOAN	YES

$$$$. EOD

Figure 8.4.2 'ROOT.DBBANK' file.

1 $$$. GENERAL INFO

DBNAME NEXT AVAIL. REC. REL. DEGREE BACKUP GENER. #

2	DBBANK	13	4	1

3 $$$. COLUMN INFO

SIZE TYPE R-LEVEL W-LEVEL COL. NAME COL. SUB-NAME CHAIN DISL.

4	6	A	1	1	CUSTOMER	NUMBER		0001
5	30	A	1	1	CUSTOMER	NAME		0007
6	30	A	1	1	ADDRESS			0037
7	30	A	2	2	MISC	INFO.		0067

8 $$$. TUPLES

DELETE LOCKED TO COL. VALUE FWRD BKWRD COL. VALUE FWRD BKWRD

9			1-100			J.J. SMITH		
	.....		MONTREAL			BUSINESSMAN		
10			1-200			K.P. FREY		
	.....		MONTREAL			PROGRAMMER		
11			4-150			T. SINCLAIR		
	.....		TORONTO			RETIRED	12	
12			3-400			T. BIRCH		
	.....		VANCOUVER			RETIRED		11
13								
	.....							
	.....							

$$$$: EOD

Figure 8.4.3 'REL. DBBANK. CUSTOMER' file.

1 $$$. GENERAL INFO

DBNAME NEXT AVAIL. REC. REL. DEGREE BACKUP GENER. #

2 DBBANK	14	4	1

3 $$$. COLUMN. INFO

SIZE TYPE R-LEVEL W-LEVEL COL. NAME COL. SUB-NAME CHAIN DISL.

4 6	A	1	1	ACCOUNT	NUMBER		0001
5 6	A	1	2	CUSTOMER	NUMBER		0007
6 4	A	1	1	ACC-TYPE			0013
7 *	R	1	1	BALANCE	($)		0017

8 $$$. TUPLES

DELETE LOCKED TO COL. VALUE FWRD BKWRD COL. VALUE FWRD BKWRD

9		1-0001			1-100		
.....		SAV	10		1000.00		
10		1-0002			1-100		
.....		SAV	12	9	5000.00		
11		1-0003			1-200		
.....		CHK	13		300.57		
12		1-0004			4-150		
.....		SAV		10	10000.00		
13		1-0006			3-400		
.....		CHK		11	534.55		
14							
.....							

$$$$. EOD

Figure 8.4.4 'REL.DBBANK.ACCOUNTS' file.

```
1  $$$. GENERAL INFO
   .DBNAME    NEXT AVAIL. REC.   REL. DEGREE  BACKUP GENER. #
2  DBBANK     13                 4            1

3  $$$. COLUMN INFO
   SIZE TYPE R-LEVEL W-LEVEL COL. NAME  COL. SUB-NAME  CHAIN DISL.
4  6    A    1       1       CUSTOMER   NUMBER        YES   0001
5  *    R    1       1       LOAN       AMOUNT        YES   0007
6  *    R    1       1       INTEREST   RATE                0011
7  30   A    2       2       MISC       INFO.               0015

8  $$$. TUPLES
   DELETE LOCKED TO COL. VALUE FWRD BKWRD COL. VALUE FWRD BKWRD
9                  1-100     12             10000.00
   .........       .1375                    HOME LOAN
10                 1-200                    20000.00
   .........       .1275                    HOME LOAN
11                 1-210                    5000.00   12
   .........       .1375                    CAR LOAN
12                 1-100           9        5000.00             11
   .........       .1400                    VAC. LOAN
13
   .........

   $$$$. EOD
```

Figure 8.4.5 'REL_DBBANK_LOAN' file.

1 $$$. GENERAL INFO

DBNAME NEXT AVAIL. REC. REL. DEGREE BACKUP GENER.#

2	DBBANK	13	4	1

3 $$$. COLUMN INFO

SIZE TYPE R-LEVEL W-LEVEL COL. NAME COL. SUB-NAME CHAIN DISL.

4	6	A	1	1	ACCOUNT	NUMBER	YES	0001
5	2	A	1	1	TYPE OF	TRANSACTION		0007
6	6	A	1	1	DATE			0009
7	*	R	1	1	AMOUNT	($)		0015

8 $$$. TUPLES

DELETE LOCKED TO COL. VALUE FWRD BKWRD COL. VALUE FWRD BKWRD

9	x		1-0001	12		DP		
	.....		110880			500.00		
10			1-0004			DP		
	.....		110880			550.00		
11			1-0003			WD		
	.....		120880			100.00		
12	x		1-0001		9	WD		
	.....		150880			25.00		
13								
	.....							
	.....							

$$$$. EOD

Figure 8.4.6 'REL. DBBANK. TRANSACTIONS' file.

1 $$$. GENERAL INFO
OBNAME NEXT AVAIL REC. REL. DEGREE BACKUP GENER. #

2	DBBANK	10	2	1

3 $$$. COLUMN INFO
SIZE TYPE R-LEVEL W-LEVEL COL NAME COL SUB-NAME CHAIN DISL

4	3	A	1	15	INTEREST	TYPE	0001
5	*	R	1	15	INTEREST	RATE	0004

6 $$$. TUPLES
DELETE LOCKED TO COL VALUE FWRD BKWRD COL VALUE FWRD BKWRD

7			SAV			.1100		
	.....							
8			CHK			.0975		
	.....							
9			TRM			.1300		
	.....							
10								
	.....							
	.....							
	.....							
	.....							

$$$$. EOD

Figure 8.4.7 'REL DBBANK INTEREST' file.

was mentioned that the GET and SELECT commands will always skip deleted tuples. This feature will allow retrieval of transactions, updating the booklet and subsequent deletion of the transactions so that they will not be available for the next time. This operation will require the following commands:

LOCATE NODE-ID,TRANSACTIONS(ACCOUNT='1-001')

Since the chaining feature is selected for the 'account', all transactions pertaining to this account will be in double linked list. The LOCATE command will locate the head of the list in the INDEX file and post the address (9) in the PRT table. In order to access all transactions the GETF command should be used.

GETF NODE-ID,TRANSACTIONS,WORKBUFFER,(ACCOUNT)

This command should be repeated until no more transactions are available and this condition would be posted to the PRT table in the relation status field. The last operation would be to find again the head of the list and delete all the tuples. These commands would be:

LOCATE NODE-ID,TRANSACTIONS(ACCOUNT='1-001')

DELETE NODE-ID,TRANSACTIONS,ALL

These transactions can be issued for local or distributed relations. The presence of the 'NODE-ID' keyword in all commands allows the user program to address any node of the network.

The system files are variable record length so that column insertion and deletion do not present any file structure problems. The system allows one primary key and more than one secondary key. Chaining option is not allowed on the primary key and therefore certain tuples can only occur once in a relation such as the account number of the relation 'ACCOUNTS'.

Another feature of the model indicated in figure 8.1.5 is the Query subsystem. This subsystem would consist of a number of programs to translate queries into commands and produce reports on request with a minimum programming effort.

Each of the previously issued instructions can address either local or distributed relations. The keyword NODE-ID determines whether the type of instruction is local or distributed. When the local node is addressed, the local relation directory (LRD) is searched to locate the relation referenced in the instruction. If the relation is not found, the request is aborted. In the case where the relation is found, the process is identical to a single computer system with a centralized database.

When a remote node is addressed, the root file is first searched to determine whether the relation is partitioned or replicate, and then the partitioned relation directory (PRD) or the replicate relation directory (RRD) is searched. At

this point the entire relation or selected tuples may be transferred to the node which originated the instruction. This operation appears to be simple but implications may result from the following:

a) If one or more tuples are locked to a process, transfer will be inhibited.

b) If the node is not available, transfer will be inhibited.

In the first case, the delay will be minimal and the process of the requesting node may fall into a wait state with a possible time out. In the second case the request should be abandoned with a message to the process of the requesting node. Both cases however, are sources of a potential deadlock and should be treated carefully.

The model provides the Select, Project, Join, Union, Intersect and Difference instructions which may reduce the telecommunications costs in a distributed process. In the banking system example, a listing of all customer names and their account numbers from accross the network may be required by the bank headquarters. The customer relation (Figure 8.4.1(a)) contains much more information than is required. In this case the requesting process would issue the following instruction to each node of the network.

PROJECT (NODE-ID),(CUSTOMER),(NEW-CUST),
(CUSTOMER-NUMBER,CUSTOMER-NAME)

This instruction will create the new relation NEW-CUST consisting of two attributes, the customer number and the customer name (Figure 8.4.8). The new relation is just an ordinary "flat" file that may be processed sequentially since the tuple pointers do not exist any more.

The relational instructions can be combined so that a more powerful instruction is created. Assume that the bank's headquarters requires information for each customer number and associated loan amount. In this system, the required information is contained in both the customer relation and the loan relation. The operation to be performed in this case is first to join the two relations and then to project the required attributes. The requesting process would issue the following instruction to each node of the network.

```
PROJECT (NODE-ID)(JOIN (NODE-ID),(LOAN),
    (CUSTOMER-NUMBER = CUSTOMER-NUMBER),
    (CUSTOMER),(WORK-REL)),(NEW-LOAN),
    (CUSTOMER-NUMBER,LOAN-AMOUNT)
```

which will require two processing steps to create the new relation 'NEW-LOAN' (Figure 8.4.9)

In the previous two examples, the intent was to create a new relation consisting of the information of all nodes. This can be accomplished in two ways: First, by requesting

NEW-CUST RELATION

CUSTOMER NUMBER	CUSTOMER NAME
1-100	J. J SMITH
1-200	K. P FREY

Figure 8.4.8 Projection on customer relation.

WORK-REL

CUSTOMER NUMBER	LOAN AMOUNT	INTEREST RATE	MISC. INFO	CUSTOMER NUMBER	CUSTOMER NAME	ADDRESS
1-100	10000.00	.1375	HOME LOAN	1-100	J.J SMITH	MONTREAL
1-100	5000.00	.1275	VAC. LOAN	1-100	J.J SMITH	MONTREAL
1-200	20000.00	.1275	HOME LOAN	1-200	K.P FREY	MONTREAL
1-210	5000.00	.1375	CAR LOAN	1-210	J. BLOW	TORONTO

STEP-1: JOIN OF CUSTOMER AND LOAN RELATIONS.

NEW-LOAN

CUSTOMER NUMBER	LOAN-AMOUNT ($)
1-100	10000.00
1-100	5000.00
1-200	20000.00
1-210	5000.00

STEP-2: PROJECTION OF THE JOIN OF RELATION OF STEP-1.

Figure 8.4.9 Projection of the Join.

each node directly the returned information is appended to the information previously received, second, by broadcasting the request to all nodes. The new relation, propagating through them in a daisy-chain fashion, receives each node's information appended to it. In a similar way each instruction can be issued to one or more nodes. Instructions such as Select and Join are very complex operations, and a single computer system or a centralized database system may monopolize the resources and thus reduce the processing time significantly. In contrast, in a distributed system such as the proposed model, each node is autonomous operating in a portion of the overall information and therefore the processing time will be much smaller since nodes will process the request simultaneously. The Select and Join operators appear to have the most potential for hardware implementation since they require a large amount of cross checking between tuple and attributes of different relations.

Recent developments in hardware technology show particular promise for providing direct support for the relational features. The very large scale integration (VLSI) of logic on chips and the development of electronic rotating store based on CCD and magnetic bubble technology may be used to implement sophisticated and specialized logic on inexpensive chips. This suggests that logic that can be easily replicated and distributed over data may be available

in the near future. Functions such as searching and sorting on which the operators SELECT and JOIN are based can be moved from CPU to storage, thereby reducing the data transfer costs and decreasing processing time because of increased parallelism (Smith79).

The model provides the 'CHAIN' option which is a conventional record linking method. This feature tends to create a heavy overhead, and therefore the model will operate at a lower speed. Technological advances in associative storage hardware would, however, make an associative memory a practicable option, replacing the chain feature and making operation faster and more efficient. Another alternative to the chain feature could be the implementation of the systolic (VLSI) arrays (Kung79) that would process the relational database operations in a pipeline fashion.

9. CONCLUSIONS

The model presented here illustrates the organization of a database suitable for distribution over a set of non-specialized computers. Its application has been illustrated by the example of a small subset of a banking system. This example is typical of the type of applications which is appropriate to a distributed database architecture. In this case, both replication and partitioning are important. Replication is required because reliability is of the upmost importance, with every transaction ideally requiring backup the moment it is made. Partitioning is desirable because the greater part of transactions are always local, with only a small proportion of remote transactions being required. The organization presented here provides the logical capability of meeting these performance goals.

Apart from the pragmatic considerations illustrated by the application, there is an economic advantage to this type of database. Microcomputers, while dropping rapidly in cost, are increasing in power to the level of the mainframes of the late sixties. One of the most promising possible architectures for powerful, low cost computers is an array of identical, mass produced microcomputers. While the problem of software for such systems has yet to be solved, the model presented here affords an appropriate software for

database applications. Given a system without communications bottlenecks, the power of the database computer could be proportional to the number of component computers.

The fundamental advantages of the architecture of the model are parallelism and distribution of data. Parallelism is inherent in the architecture, since all nodes have the same software. All nodes may then process a distributed call simultaneously, giving faster execution in the case of partitioned files. Distribution of data allows localization of files physically close to their points of most frequent usage, reducing communications cost.

While the problems of deadlock and process synchronization have not been addressed specifically in this work, the research results discussed in chapter 8 have been demonstrated sufficient to cope with these problems. The model can therefore be considered as a practical solution for distributed database architecture.

10. APPENDICES

- Appendix 1 DDL and DML examples

- Appendix 2 existing relational databases.

APPENDIX 1

The DML is analogous to the procedure division in a COBOL program.

The main functions of both DDL and DML are the following:

DDL:

1. Describe name and type of data item.
2. Describe name and composition of each data entry.
3. Indicate primary and secondary keys.
4. Represent record relationships.
5. Describe schemas and subschemas.
6. Specify security restrictions.

DML:

1. Describe techniques for retrieval, replacement, insertions, and deletions of data entries.
2. Describe use of keys and data entry relationships.
3. Free the user from database structure maintenance.
4. Be independent of programming languages.

The following example illustrates these concepts. Consider a system that maintains the information on each Master's thesis in Canada, with access to the information through a number of different data items.

Assume that the information to be retained is as follows:

1. Master thesis title.

2. Short abstract.
3. Classification code (Management,Comp. Science etc).
4. Topic code (operating systems DBMS etc).
5. University code.
6. Completion date.
7. Author.
8. Degree inFormation.

for this example the Hewlett Packard IMAGE/1000 database system is used,Figure A1 illustrates the schema defined in IMAGE code.

FIGURE A1

```
<<*****************************************>>
<<                                         >>
<<  THIS IS THE SCHEMA FOR THE DATABASE    >>
<<  APPLICATION SYSTEM CALLED 'THSIS'      >>
<<*****************************************>>
$CONTROL LIST,ERRORS=0,ROOT,SET,TABLE
BEGIN DATA BASE THSIS: CR012:00010
<<                                         >>
<< FOLLOWING, THE PASSWORDS FOR VARIOUS>>
<< ACCESS LEVELS ARE DEFINED              >>
<<                                         >>
LEVELS:
       1   PASS1  << CLERK- LOWER LEVEL>>
       5   PASS2  << ASST. PROF.        >>
      10   PASS3  << CHAIRMAN OF FACUL.>>
      12   PASS4  << THE DEAN           >>
      15   PASS5  << THE RECTOR-HIGHEST>>
<<                                         >>
<<                                         >>
<< FOLLOWING, THE DATA ITEMS ARE          >>
<< DESCRIBED IN DETAIL, SUCH AS           >>
<< ITEM, ATTR SZ (RLVL,WLVL)              >>
<<                                         >>
<<   WHERE :  ITEM= ITEM NAME             >>
<<            ATTR= I: INTEGER            >>
<<                  R: REAL               >>
<<                  U: CHARACTER          >>
<<            SZ  = SIZE IN WORDS         >>
<<            RLVL= READ LEVEL            >>
<<            WLVL= WRITE LEVEL           >>
<<                                         >>
<<                                         >>
ITEMS
     TITLE  U 60( 1,12) <<60-CH.TITLE   >>
     ABSTR  U256( 1,12) <<256-CH.ABSTR.>>
     CLASS  U  8( 1,12) <<CLASSIF.CODE  >>
     TOPIC  U  4( 1,10) <<THESIS TOPIC  >>
     UNIV   U  6( 1,10) <<UNIVER. CODE  >>
     DATE   U  6( 1,12) <<COPLET. DATE  >>
     AUTH   U 30( 1,12) <<AUTHOR'S NAME>>
     DEGR   U 60(10,15) <<DEGREE INFO.  >>
     CLAS1  U  4( 1,12) <<MASTER-CLASS  >>
     TOPC1  U  4( 1,10) <<MASTER-TOPIC  >>
     UNIV1  U  6( 1,10) <<MASTER-UNIV.  >>
     DATE1  U  6( 1,12) <<MASTER-DATE   >>
<<                                         >>
<<                                         >>
<< ALL ITEMS OF THE DATABASE HAVE BEEN  >>
<< DESCRIBED. FOLLOWING, THE ITEMS WILL>>
<< ALLOCATED TO THE APPROPRIATE DATA    >>
<< SETS.                                >>
<< DATA SETS ARE DIVIDED INTO MASTER    >>
<< AND DETAIL DATA SETS.                >>
<< MASTER DATA SETS: HOLD THE LOCATION  >>
```

```
<< OF THE FIRST ENTRY OF A GROUP OF    >>
<< RELATED ENTRIES IN A DETAIL DATA SET>>
<< E.G  CLASS='COMP'                   >>
<<                                     >>
<< DETAIL DATA SETS: HOLD JUST DATA    >>
<< ENTRIES AS ENTERED IN CHRONOLOGICAL >>
<< SEQUENCE.                           >>
<<                                     >>
<<                                     >>
SETS:
     NAME: CLMST,A,CR012 <<CLASS MASTER >>
         ENTRY: CLAS1(1) <<INDICATES 1  >>
                         <<RELATIONSHIP >>
                         <<WITH THE CLASS>
                         <<ITEM IN THE  >>
                         <<DETAIL DATA  >>
                         <<SET 'THINF'  >>
         CAPACITY: 1000
                         <<             >>
     NAME: TOMST,A,CR012 <<TOPIC MASTER >>
         ENTRY: TOPC1
         CAPACITY: 200
                         <<             >>
     NAME: UVMST,A,CR012 <<UNIVER.MASTER>>
         ENTRY: UNIV1
         CAPACITY: 200
                         <<             >>
     NAME: DTMST,A,CR012 <<DATE MASTER  >>
         ENTRY: DATE1
         CAPACITY: 1000
                         <<             >>
     NAME: THINF,D,CR012 <<DETAIL THESIS>>
         ENTRY: TITLE,   <<INFO DATA SET>>
                ABSTR,
                CLASS(CLMST),
                TOPIC(TOMST),
                UNIV (UVMST),
                DATE (DTMST),
                AUTH,
                DEGR,
         CAPACITY: 1000
END.
```

This schema is represented in the IMAGE DDL language. All the data items are defined along with their attributes and security levels. Data sets have been defined as far as their names, capacity and disk unit are concerned.

User inquiries: Database software sometimes provide a

software package that interfaces with the user and enables him to retrieve and/or modify information stored in the database application system with the minimum programming effort. These packages are known as "Query" languages. IMAGE/1000 supplies a such query language package called QUERY/1000 allowing the user to

1. Retrieve data entries.
2. Modify data entries and/or data items.
3. Delete data entries.
4. Produce reports.

Queries usually use Englishlike commands so that the average non computer science user can access the system. For the example of Figure A1 a typical query is illustrated in the following:

Assume that we want to retrieve all the entries related to the TOPIC="DBMS". The following command should be issued.

FIND TOPIC IS "DBMS" END:

Query then will access the data set and will respond:

00005 ENTRIES QUALIFIED

This indicates that there are only 5 master thesis records under the "DBMS" topic.

For this example, the DML will have performed the following tasks:

- At log on time check the passwords and perform housekeeping tasks
- Search the database application system and locate the master entry having value "DBMS"

- Get the address of the first detail, retrieve all the remaining (if any) and pass them to Query.

Application programs can access the database application system via callable subroutines. IMAGE/1000 allows languages such as FORTRAN and BASIC to access its database system. In this case also the DML commands are issued to the program functions which resolve the necessary references and interface with the calling program. Application programs do not normally access the datasets directly. Some typical instructions are

DBOPN -Database open

DBINT -Database initialize

DBCLS -Close database

DBINF -Database information

DBGET -Retrieve a data entry

DBUPD -Update a data entry

DBDEL -Delete a data entry

DBPUT -Add a data entry.

NAME	YEAR	MACHINE	STATUS	IMPLEMENTORS
MADAM	1970	H6000	IMPL./INACT.	MIT PROJECT MAC
RDMS	1971	H6000	IMPL./ACTIVE	MIT EE DEPT.
IS/1 (PRTV)	1971	IBM360, 370	IMPL./ACTIVE	IBM UK.
RDMS (REGIS)	1972	IBM360, 370	IMPL./ACTIVE	GM RESEARCH
RM/XRM	1972	IBM370	IMPL./ACTIVE	IBM CAMB. SC
DAMAS	1972	—	DESG./INACT.	MIT CE DEPT.
GAMMA-0	1973	—	DESG./INACT.	IBM SJ
SEQUEL	1974	IBM370	IMPL./INACT.	IBM SJ
RISS	1974	PDP11	IMPL./ACTIVE	FOREST HOSPITAL
GMIS	1975	IBM370	IMPL./ACTIVE	MIT SSM & IBM CAMB.
ZETA/TORUS	1975	IBM360, 370	IMPL./INACT.	UNIV. TORONTO CAN.
OMEGA	1975	PDP11	DESG./INACT.	UNIV. TORONTO CAN.
PLANES	1975	PDP10	IMPL./ACTIVE	UNIV. ILLINOIS URB.
MAGNUM	1975	PDP10	IMPL./ACTIVE	TYMSHARE INC.
INGRES/CUPID	1975	PDP11	IMPL./ACTIVE	UNIV. CALIFORNIA
RARES	1975	—	DESG./INACT.	UNIV. UTAH
SQUIRAL	1975	—	DESG./INACT.	UNIV. UTAH
GXRAM	1975	IBM370	IMPL./ACTIVE	IBM SJ
RAP	1976	-	IMPL./ACTIVE	UNIV. TORONTO CAN.

Appendix 2(a). Existing relational databases (Kim79).

NAME	YEAR	MACHINE	STATUS	IMPLEMENTORS
RENDEVOUS	1976	IBM370	IMPL./ACTIVE	IBM SJ
QUERY BY EXAMPLE	1976	IBM370	IMPL./ACTIVE	IBM YORKTOWN N.Y
LEECH	1976	_	DESG./ACTIVE	GLASGOW, ENGLAND
CAFS	1976	_	IMPL./ACTIVE	ICL, ENGLAND
DBC	1976	_	DESG./ACTIVE	OHIO STATE UNIV.
SYSTEM R	1977	IBM370	IMPL./ACTIVE	IBM SJ
DB85	1977	INTERDATA	IMPL./ACTIVE	UNIV. KANSAS
SDD-1	1977	_	DESG./ACTIVE	CCA, CAMBRIDGE MASS.
CASSM	1978	_	IMPL./ACTIVE	UNIV. FLORIDA
DIRECT	1978	PDP11	DESG./ACTIVE	UNIV. WISCONSIN

THE MACHINE IS THE COMPUTER ON WHICH A SYSTEM HAS BEEN IMPLEMENTED. RAP, CASSM, RARES, LEECH, CAFS, AND DBC ARE DESIGNS FOR SPECIALIZED PROCESSORS. SDD-1 IS A DISTRIBUTED SYSTEM UNDER DEVELOPMENT AND DAMAS, SQUIRAL, AND GAMMA-0 REPRESENT PROPOSALS FOR IMPLEMENTING A COMPONENT OF A SYSTEM. THE STATUS IS DESIGNATED EITHER AS IMPLEMENTED (IMPL.) OR ONLY DESIGNED (DESG.), AND AS EITHER ACTIVE, THAT IS, CURRENTLY UNDER DEVELOPMENT OR IN USE, OR INACTIVE (INACT.)

Appendix 2(b). Existing relational databases (Kim79).

11. ANNOTATED BIBLIOGRAPHY

Alsbe76 P.A. Alsberg, G.G. Belford, J.D. Day, E. Grapa "Multi-copy resiliency techniques" Tutorial: Distributed database management. IEEE- Computer society 1978 pp 128-176.

Astra79 M.M. Astrahan et al. "System R: a relational database management system" IEEE computer society May 1979 pp. 42-48.

Baner78.1 J. Banerjee, D.K. Hsiao "A methodology for supporting existing CODASYL databases with new database machines" ACM 1978 pp. 925-936.

Baner78.2 j. Banerjee, D.K. Hsiao, R.I. Baum "Concepts and capabilities of a database computer" ACM transactions on database systems vol 3, No.4 December 1978 pp. 347-384.

Baner79 J. Banerjee "Performance analysis and design methodology for implementing database systems on new database machines" the Ohio state University. Ph. D 1979.

Berns78 P.A. Bernsten, J.B. Rothnie, JR.N. Goodman G.G. Papadimitriou. "The concurrency control mechanism of SDD-1 A system for distributed databases (the fully reduntant case)" IEEE

transactions on software engineering vol SE-4 No.3, May 1978 pp 113-126.

Baner80 J. Banerjee, D.K. Hsiao, F.K. Ng "Database transformation, Query translation and performance analysis of a new database computer in supporting Hierarchical database management" IEEE transactions on software engineering. Vol SE-6, No.1 January 1980 pp. 91-109.

Berra79 P.B. Berra, E. Oliver "The role of associative array processors in the database machine architecture" IEEE computer society March 1979 pp. 53-61.

Booth77 C.M. Booth "Distributed data bases" Infotech report vol:2 1977 pp 25-33.

Champ76 D.D. Champerlin "Reletional data-base management systems" ACM computing surveys Vol.8 #1 March 1976 pp. 43-66.

Champ79.1 G.A. Champine "Current trends in database systems" IEEE computer society May 1979 pp. 27-41.

Champ79.2 R.B. Chamberlain "The promise and problems of relational database design" computerworld September 1979 pp. 17-21.

Champ80 G.A. Champine. "Back-end technology trends" IEEE computer society February 1980 pp. 50-58.

Chang80 S-K. Chang, W.H. Cheng "A methodology for structured database decomposition" IEEE transactions on software engineering vol SE-6 no.2 March 1980 pp. 205-218.

Chlam80 I. Chlamtac, N.R. Franta, P.C. Patton. "Performance issues in Back-end storage networks" IEEE computer society February 1980 pp. 18-31.

DeWit79.1 D.J. De Witt "DIRECT- A multiprocessor organization for supporting relational database management systems" IEEE transactions on computers vol C-28, No-6,June 1979. Pp. 395-406.

DeWit79.2 D.J. De Witt "Query execution in DIRECT" ACM 1979 pp 13-22.

DeWit80 P.B. Hawthorn, D.J. DeWitt "Performance analysis of alternative database machine architectures" computer science technical report #383 March 1980.

Engle72 R.W. Engles "Tutorial in database organizations" Automatic programming. Pergamon press 1972.

Fancot81 T. Fancott "Complementary Interface Functions for Communication Between Distributed Processes", Proceedings, CIPS Conference 81, Waterloo, Ontario, June 8-10. In press.

Freem80 H.A. Freeman. "Back-end stoarge networks" IEEE computer society February 1980 pp. 7-8.

Fry76 J.P. Fry, E.H. Sibbley "Evolution of data-base management systems" ACM computing surveys vol.8 # 1 March 1976 pp. 7-42.

Gelen79 E. Gelenbe, K. Sevcik "Analysis of update synchronization for multiple copy data bases" IEEE transactions on computers . Vol C-28 no.10. 1979 pp. 737-747.

Ghosh77 S.P. Ghosh "Database organization for data management" IBM research laboratory. Academic press 1977.

Haerd78 T. Haerder "Implementing a generalized access path structure for a relational database system" ACM transactions on database systems vol:3, no:3, September 1978 pp. 285-298.

Henve79 A.R. Henver, S.B. Yao "Query processing in distributed database systems" IEEE transactions on software engineering vol SE-5 no.3 May 1979

pp. 177-187.

Hill78 E. Hill Jr. "A comparative study of very large databases" Springer-Verlag 1979.

Holla79 L.A. Hollaar. "Text retrieval computers" IEEE computer society March 1979 pp.40-50.

Horro77 Horrowitz and Sahni "Fundamendals of data structures" computer science press inc. 1977.

Hp1000 "IMAGE/1000" Hewlett-Packard.

Hp3000 "IMAGE/3000" Hewlett-Packard.

Hsiao76 R.I. Baum, D.K. Hsiao "Database computers- a step towards data utilities" IEEE transactions on computers vol C-25 no 12 December 1976. Pp. 1254-1259.

Hsiao79.1 D.K. Hsiao. "Data base machines are coming" IEEE computer society March 1979 pp. 7-9.

Hsiao79.2 J. Banerjee, D.K. Hsiao, K. Kannan "DBC- A database computer for very large databases" IEEE transactions on computers vol C-28 no.6 June 1979 pp.414-429.

Hsiao79.3 D.K. Hsiao, D. Kerr, G.J. Nee "Database access control in the presence of context dependent protection requirements" IEEE

transactions on software engineering vol SE-5 no.4 July 1979 pp. 349-358.

Kerr79 D.S. Kerr "Data base machines with large content addressable blocks and structural information processors" IEEE computer society March 1979 pp. 64-79.

Kim79 W. Kim "Reletional database systems" ACM computing surveys vol 11 no.3 September 1979 pp. 185-211.

Knuthv.3 D.E. Knuth "The art of computer programming" vol.3 'sorting and searching'. Addison-Wesley 1973.

Kroen78 D. Kroenke "Database- A professional's primer" SRA 1978.

Kung79 H.T. Kung P.L. Lehman "Systolic (VLSI) arrays for relational database operations" CARNEGIE-MELLON University October 1979.

Lenah80 J.J. Lenahan, F.K. Fung "Performance of cooperative loosely coupled microprocessor architectures in an interactive database task" IEEE transactions on computers, vol C-29, no.2 February 1980 pp. 161-180.

Levin74 K.D. Levin "Organising distributed databases in

computer networks" Wharton school of Finance and Commerce -Philadelphia- PHD 1974.

Marti76 J. Martin "Principles of data-base management" Prentice-hall 1976.

Menas79 D.D. Menasce, R.R. Muntz "Locking and deadlocking detection in distributed data bases" IEEE transactions on software engineering vol SE-5 no.3 May 1979 pp. 195-202.

Peebl78 R. Peebles, E. Manning "System architecture for distributed data management" IEEE computer society January 1978 pp. 40-46.

Ramam77 Ramamoorthy Krishnarao "Distributed database design: design considerations" Infotech report vol:2 1977 pp 157-158.

Robin79 S.L. Robinson "Relational databases: what, when, where, why" computerworld September 1979 pp. 13-16.

Rosen78 D.J. Rosenkrantz, R.E. Syerns, P.M. Lewis "System level concurrency control for distributed database systems" ACM transactions on database systems vol.3 no.3 September 1978 pp. 168-199.

Ross78 R.G. Ross "Database systems Design,

implementation, and management" amacom 1978.

Rothn78 J.B. Rothnie, N. Goodman "A survey of research and development in distributed database management" Tutorial: Distributed database management IEEE computer society 1978. Pp 30-44.

Schla78 G. Schlageter "Process synchronization in database systems" ACM transactions on database systems vol.3 no.3 September 1978 pp. 248-271.

Schus79 S.A. Schuster, H.B. Nguyen, E.A. Ozkarahan K.C smith "RAP.2- An associative processor for databases and applications" IEEE transactions on computers vol C-28 no.6 June 1979 pp. 446-458.

Sible76 E.H. Sibley "The development of database technology" ACM computing surveys vol.8, no.1 March 1976 pp. 1-5.

Smith79 D.C.P. Smith, J.M. Smith "Relational database machines" IEEE computer society March 1979 pp. 28-38.

Stone79 M. Stonebaker "Concurrency control and consistency of multiple copies of data in distributed INGRES" IEEE transactions on software engineering vol SE-25 no.4 July 1979 pp.188-194.

Su79 S.Y.W. Su "Cellular-logic devices, concepts and applications" IEEE computer society March 1979 pp. 11-25.

Taylo76 R.W. Taylor, R.L. Frank "CODASYL database management system" ACM computing surveys vol.8 no.1 March 1976 pp. 67-103.

Thoma78 R.H. Thomas "A solution to concurrency control problem for multiple copy data bases" Tutorial: Distributed database management IEEE computer society 1978 pp 88-94.

Thorn80 J.E. Thornton. "Back-end network approaches" IEEE computer society February 1980 pp. 10-17.

Tsich76 D.C. Tsichritizis, F.H. Lochovsky "Hierarchical data-base management" ACM computing surveys vol.8 no.1 March 1976 pp. 104-123.

Ullma80 J.D. Ullman "Principles of database systems" stanford University- Computer science press 1980.

Verho78 J.M. Verhofstad "Recovery techniques for database systems" ACM computing surveys vol.10 no.2 June 1978 pp. 167-195.

Watso80 R.W. Watson. "Network architecture. Design