PROCEEDINGS OF 1979 NATIONAL COMPUTER CONFERENCE, AFIPS PRESS.

Data management in distributed data bases*

by C. V. RAMAMOORTHY and BENJAMIN W. WAH
University of California
Berkeley, California

INTRODUCTION

The recent advances in large-scale integrated logic and memory technology, coupled with the explosion in size and complexity of the application areas, have led to the design of distributed architectures. Basically, a *Distributed Computer System* (*DCS*) is considered as an interconnection of digital systems called *Processing Elements* (*PEs*), each having certain processing capabilities and communicating with each other. This definition encompasses a wide range of configurations from an uniprocessor system with different functional units to multiplicity of general-purpose computers (e.g. ARPANET). In general, the notion of "distributed systems" varies in character and scope with different people.[30] So far, there is no accepted definition and basis for classifying these systems. In this paper, we limit our discussion to a class of DCSs which have an interconnection of dedicated/shared, programmable, functional PEs working on a set of jobs which may be related or unrelated.

Due to the information explosion and the need for more stringent requirements, the design of efficient coordination schemes for the management of data on a DCS is a very critical problem. Data on a DCS are managed through a data base. A *Data Base* is a collection of stored operational data used by the application systems of some particular enterprise,[6,12] and a *Distributed Data Base* (*DDB*) can be thought of as the data stored at different locations of a DCS. It can be considered to exist only when data elements at multiple locations are interrelated and/or there is a need to access data stored at some locations from another location. Due to the ever-increasing demand for on-line processing, there is a need for decomposing very large data bases into physically or geographically dispersed units and/or integrating existing data bases held in physically isolated nodes into a single, coherent data base that will be available to each of the distributed nodes.

In this paper, the design issues and solutions for resource management of data on a DDB are studied. The different aspects of resource management are categorized in the next section. These management issues are part of the issues related to the operational control of a DDB and are concerned with the management of data as resources. They can be divided into three related levels, namely, the query level, the file level and the task level. The query level is concerned with the processing of user queries and requests so that parallelism in processing can be maximized, and the amount of communications on the system can be minimized. On the file level, the related issues are the compression of data files for efficient storage and communication, as well as the placement and migration of files for efficient accesses. On the task level, the objective is to schedule the requests so that overlap in processing can be maximized. These issues and some of the corresponding solution algorithms are studied in detail in the third to sixth sections respectively. Finally, the seventh section provides some concluding remarks.

* Research supported by Ballistic Missile Defense Contract DASG60-77-C-0138.

RESOURCE MANAGEMENT OF DATA IN DDBS

There are many issues in the design of a data base, among which are the issues in logical organization, architectural designs, operational control and evolution. These issues have been discussed in Reference 31 and will not be repeated here. A summary of the issues in the design of a DDB are shown in Figure 1. In this paper, the resource management issues of data and files on a DDB are studied. The specific data management issues investigated are:

Query decomposition on DDBs

A *query* is an access request made by a user or a program in which one or more files have be accessed. When multiple files are accessed by the same query on a DDB, these files usually have to reside at a common location before the query can be processed. Substantial communication overhead may be involved if these files are geographically distributed and a copy of each file has to be transferred to a common location. It is therefore necessary to decompose the query into sub-queries so that each sub-query accesses a single file. These sub-queries may then be processed in parallel at any location which has a copy of the required file. The results after the processing are then sent back to the re-

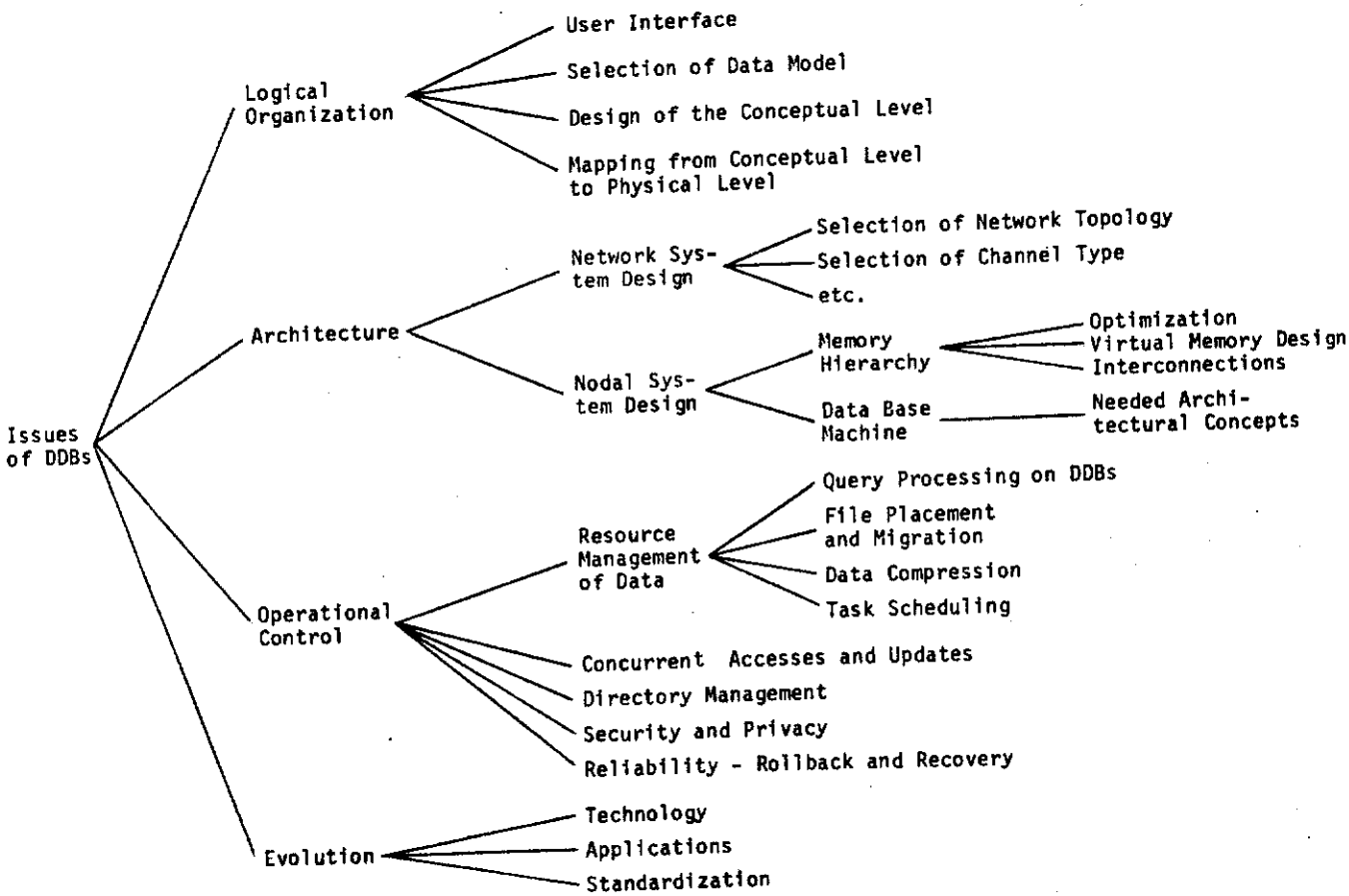

Figure 1—Classification of issues in distributed data base systems.

questing location. It is generally true that the amount of communications needed to transmit the results is much smaller than the amount needed to transmit the files. This approach has been proposed in the design of the centralized version of INGRES[41] and is extended to the design of SDD-1,[42] and distributed INGRES.[8] However, in some cases decomposition is impossible and some file transfers are still necessary. In order to avoid these extra transfers, a technique is proposed in the third section so that redundant information is added to the files and non-decomposable queries can still be processed without any file movements.

Data compression

Data compression is any reversible encoding technique that produces a measurable reduction in the size of the data encoded. By reversible, it is meant that the original data is recoverable from the compressed form. Due to the growth in the size of information processing, it is necessary to develop good data compression techniques which reduce the size of the stored information and the amount of internode communications. This issue is discussed in the fourth section.

File placement and migration

This issue relates to the distribution and migration of data base components, namely, files and control programs, on the DDB with the objective of minimizing the overall storage, migration, updating and access costs on the system. A file assignment algorithm is proposed in the fifth section.

Task scheduling

Requests on the DDB must be scheduled so that high parallelism and overlap can be achieved. The request may be a single word fetch or it may be a page or file access. The parallelism on the DDB is important because in order to attain high throughput, the parallel hardware and resources must be efficiently utilized. The control of task scheduling can be distributed or centralized. In distributed control, each node may act independently and coordinate with each other. In centralized control, there is a primary node in which all scheduling control will be performed there. The decision of which is the better control mechanism depends very heavily on the interconnection structure and the

communication overhead involved. This issue is discussed in the sixth section.

The relationships among the various data management issues are shown in Figure 2 where a relation → is said to exist between two design issues â, b̂, i.e. â→b̂, if the solution of b̂ is transparent to the solution of â. That is, the solution of â is not affected by the solution to b̂, but not vice versa. The solution to â can therefore be developed independent of b̂. In Figure 2, it is seen that generally, task scheduling is transparent to file placement and migration which in turn could be transparent to data compression and query decomposition. Algorithms for data compression and query decomposition can therefore be developed independently. In developing algorithms for file placements and migrations, the solutions for data compression and query decomposition should be taken into account. However, in most cases, assumptions can be made about their solutions and the file placement and migration problem can be solved independently. For example, it may be assumed that all queries which access multiple files may be decomposed into sub-queries that access single files. The file placement and migration problem for multiple files is therefore decomposed into many single file optimization sub-problems. It must be noted that other operational control requirements may also impose restrictions on the solutions to the data management issues. For instance, different reliability requirements may demand different lower bounds on the number of copies of a file on the DDB; different concurrency control mechanisms may have different costs on the file placement problem; etc. Reasonable assumptions must therefore be made about these operational control requirements in order to determine their effects on the resource management issues and to solve these issues independently.

QUERY DECOMPOSITION ON DDBS

The approach using query decomposition is geared towards relational data bases.[4] In a relational data base, data

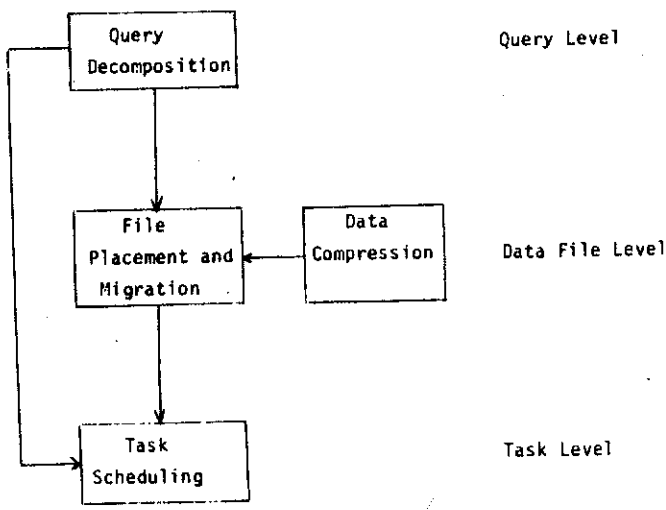

Figure 2—Relationships among various data management issues.

is viewed as relations of varying degree, the degree being the number of distinct domains participating in the relation. Each instance of a relation is known as a tuple, which has a value for each domain of the relation. Thus a relation can be simply represented in tabular form with columns as domains and rows as tuples. In query decomposition, optimization is performed on the processing of a single query originated at a node. The objective is to decompose a multiple relation query into as many single relation sub-queries as possible so that data (relation) movements from one node to another can be minimized.[41,42,8,13] However, there exists non-decomposable queries which require all the relations that they access to be present at a common location. A large number of relation transfers may be needed if these relations are geographically distributed. In order to avoid these extra relation transfers, a technique utilizing redundant information is proposed here. Instead of decomposing queries that access multiple files, it may be sufficient to provide redundant information in each relation so that multiple relations do not need to reside at a single location before the query can be processed. This will be illustrated later in this section. We begin by first examining the different types of queries on a relational data base.

A query on a relational data base consists of two parts: the part specifying the domains of the relation to be retrieved and the part specifying the predicate which is a quantification representing the defining properties of the set to be accessed. Let S be a relation of domain s#, sname, status, city; and SP be a relation of domains s#, p#, qty. The queries on a relational data base can be classified into the following categories:[6]

- *Retrieval Operations*
  a. Single Relation Retrieval—The predicate representing the defining property of the set to be retrieved is defined on the same relation as the set.
     E.g. GET (S.sname): S.city="Paris" AND S.status>10
  b. Multiple Relation Retrieval—The predicate, as well as the set to be retrieved, may be defined over multiple relations.
     E.g. GET (S.sname): (SP.s#=S.s# AND SP.p#="P_2")
     Relation SP and S must be available simultaneously before the query can be processed.
- *Storage Operations*
  a. Single Relation Update
  b. Multiple Relation Update
  c. Insertion
  d. Deletion
- *Library Functions*
  These represent more complicated operations on the predicate than the equality operations, e.g. counting the number of occurrences, selecting the maximum/minimum etc.

A query which is defined over multiple relations is not decomposable into single relation sub-queries when it has a

logical relation defined over a common domain of these multiple relations. For example, the query:

GET (S.sname): (SP.s#=S.s# AND SP.p#="P_2")

is not decomposable into single relation retrievals because there is a logical relation "=" which is defined over a common domain s# of the relations S and SP. These relations must be available simultaneously at a common location before the retrieval or update operations can be performed. It is noted that the common domains of these multiple relations actually represent multiple copies of the same domain on these relations (although the information they contain may not be identical). A lot of transfers can be eliminated if their common information is represented in both relations. For example, in processing the query:

GET (S.sname): (SP.s#=S.s# AND SP.p#="P_2")

on two geographically separated relations, S and SP (Figure 3a), it may be necessary to transfer relation S to the node where SP resides and then process the query there or vice versa. However, if the information SP.s#=S.s# are compiled beforehand into the two relations (Figure 3b), then it is only necessary to send the query to the location where S or SP resides and the query can be processed there.

This technique poses several problems. First, it is necessary to take one extra bit for each tuple in order to compile this piece of information. If the amount of information to be added is large, (e.g. when the number of different predicates defined on a common domain of two relations is large), the size of the extra storage space may be significant. Second, when the common domain of one relation is modified, it is necessary to "multiple update" all the common domains of the other relations in the data base. Referring to Figure 3b, if an extra tuple with s#=2 and sname="Boston" is added to relation S, then it is necessary to update the SP.s#=S.s# information in relation SP because relation SP contains a tuple with s#=2. If updating activity is frequent, then the "multiple update" cost is large. Third, this technique requires that the data base designer to be able to estimate the amount of additional information to be compiled into the relations. A possible technique is to pre-analyze the type of predicates used in retrievals and updates and to determine what are the essential information to be compiled into the relations. A compromise should be made between introducing extra information with additional storage space and higher cost in multiple updates and reducing the amount of relation transfers. It would be advantageous for the more frequently used predicates and less advantageous for others.

S	s#	sname
	1	New York
	3	San Francisco
	5	Chicago

SP	s#	p#
	1	A1
	2	A1
	3	A2
	4	A2
	5	P2

Figure 3a—Relations S and SP.

S	s#	S.s#= SP.s#	sname
	1	1	New York
	3	1	San Francisco
	5	1	Chicago

SP	s#	S.s#= SP.s#	p#
	1	1	A1
	2		A1
	3	1	A2
	4		A2
	5	1	P2

Figure 3b—Relations S and SP with (S.s#=SP.s#) information compiled into the relations.

DATA COMPRESSION

With the increase in the amount of information processing, it is important to keep the utilization of the memory high. The information content of data stored in large alphanumeric data bases is usually low. Further, as the processing becomes distributed, the communication overhead of transferring data from one location to another is usually substantial. In order to keep the utilization of the storage sub-system high, and to keep the amount of data transferred over communication links low, data compression is a natural solution to the problem. However, the use of compression codes which remove the redundancy of data seems to be in direct conflict with the use of redundant coding, e.g. parity check codes, which increase the reliability. What is needed then is an exploration of efficient error limiting codes which can be applied to compressed data and an analysis of the error rate of various compression schemes.

Desirable properties of compression codes

In designing a compression code, it should possess to some degree each of the following properties:

1. The technique should be reversible, i.e. the original data should be fully recoverable from the compressed form. This property can be relaxed in certain situations when the data is repeated elsewhere, e.g. the keys in a directory structure are usually repeated across levels.
2. The coding scheme should cause a measurable reduction in the size of the stored data. In comparing compression codes, a standard measure called percent compression is generally used.

$$\text{percent compression} = \frac{[\text{size of input data}] - [\text{size of output data}]}{[\text{size of input data}]} \times 100\%$$

3. The technique should be reasonably efficient to implement.

4. The technique should be general enough to be equally applicable to all alphanumeric data files.

Two other properties which are often desirable in compression schemes are:

5. The prefix property, i.e. no code is the prefix of another code. This assures that the decoder never has to backup on any portion of the text.
6. Lexicographic ordering property, i.e. if the input data is in a sorted order, then after encoding, the output data is still in sorted order. This property is useful for indexes.

Existing compression techniques, which possess part or all of the above properties, can be classified into the following board categories: (1) Run length encoding; (2) Differencing; (3) Statistical encoding; (4) Value set schemes.

Run Length Encoding—In a data base, there are frequent occurrences in which the data occur in a continuous sequence of identical characters, e.g. sequence of zeroes. This sequence can be replaced by the character followed by a count. Run length encoding is a technique by which a string of continuous characters or a "clump" are replaced by a repeat flag for the character followed by the size of the "clump" or run length. In practice, however, since very long clumps are highly improbable, one can limit the run length encoded and combine the flag and length in a single byte. This is the technique used in WYLBUR.[10] Run length encoding of a single character type is potentially the most successful, with diminishing returns for more characters. Huang has discovered an upper bound for the entropy of run length encoding.[19]

Differencing—Differencing refers to techniques which compare a current record to a pattern record and retain only the differences between them. It is particularly successful with large files of records with fixed length alphanumeric fields where most corresponding fields are the same or are blanks and zeroes. This is the approach normally used for sequential files, where the pattern record is taken by the previous record in the file. When differencing is applied to direct access files, however, the first record of each block is left uncompressed and used as a pattern for the remaining records in the block. The unit of information on which differencing is performed can be the bit, the byte, the field or some logical data in the record. Byte-level differencing is the most common case since byte access is convenient and cheap. In field-level differencing, bit maps are often used to indicate the presence or absence of a field when identical to the previous. Two examples of the use of differencing in relational data base systems are Titman's experimental system[38] and the Peterlee Relational Test Vehicle.[39]

Statistical Encoding—Statistical encoding is a transformation of an input alphabet so that it is assigned a code bit string whose length is inversely proportional to the frequency of its occurrences in the text. Since different characters occur with different frequencies, a statistical encoding scheme will usually compress the text. Huffman coding scheme[20] is an optimum, elegant and simple algorithm to assign variable length bit codes with the prefix property to characters, given their frequencies of occurrence in a text. There are other techniques such as the Hu-Tucker Algorithm,[18] which has both the prefix and the lexicographic ordering property. The major drawbacks of statistical encoding are that it does not exploit the natural radix of the computer (e.g. byte, word, etc.), and it does not take into account some special characteristics of the data, e.g. strings of repeating characters, and the distinction between numeric and character data. A solution to this is the use of fixed length encoding which manipulate data in units of byte.[32,2] Further, the fact that the size of each character is variable also causes problems when the data are modified and the reliability of the data is difficult to assure because the character stream would not be recognizable once a bit is destroyed.

Value Set Schemes—A value set scheme in a data base system is a coding scheme in which repeated storage of data elements in their full character representation is avoided. Instead, each data element is stored once in the system and all subsequent occurrences of the same data element are referred to the first stored occurrence. An example of this technique is shown in the MacAIMs Data Management (MADAM) System[14] in which a reference number is assigned to a new entering data element and all subsequent operations on the data element use the reference number. However, the fact that reference numbers are unique only within a relation could lead to problems in the reliability of the data management system and the integrity of the data. The MADAM System also uses a binary tree scheme for maintaining reference numbers which is inefficient for insertion and costly in storage space for large sets of data. There are other schemes which represent a better tradeoff between storage efficiency and processing efficiency.[24]

The decision of which code to use is highly dependent on the applications. For example, in a data base where the order of data is not important, the lexicographic ordering property is not important. The required properties of the applications must therefore be identified by the designers before the code is selected.

Future directions of research

While there are many reported results on data compression, the future directions of research are seen to be concentrated in the following areas:

Identify and characterize data redundancy—In a data base, there are many levels of data. For example, there are the file level, the record level, the field level and the byte level. The type of data redundancy at each level must be identified. This would aid in selecting data compression schemes best suited to the particular type of redundancy. Further, it leads us to the possibility of multi-level compression schemes, wherein data is compressed through a set of cascaded stages. Each level of the data is possibly compressed using a different technique. The compression code must be selected so that it minimizes the effects on other levels of the data.

Develop a comparison model for various compression schemes—The comparison model must be able to measure the amount of storage reduction and the computation cost for encoding and decoding. A simple measure is the percent compression defined earlier. The computation cost can be broken down into the CPU cost, the memory usage cost and the input/output cost. In order to calculate the storage reduction for a given compression scheme, the number of encodable units of tokens in a record or file must be predicted. This can be obtained from an assumed input distribution such as uniform distribution, normal distribution or Zipf's distribution at the given level of data.

Study adaptive Huffman coding techniques which respond to update activity—As the data base gets updated, the initial Huffman code assignment based on the a priori character frequency distribution may no longer be optimal. A threshold for the expected compression ratio has to be determined which can dynamically reassign the variable length codes for the new frequency distribution. Further, the threshold selected should not cause excessive re-coding. The problem of updates which change the size of the data, and the reliability problems should also be studied.

Investigate the feasibility of implementing, in a microprocessor, a simple self-measuring self-adjusting encoder/decoder—Experience has shown that the current implementation of data base systems are I/O bound within a node and communication bound on the DCS. A microprocessor encoder/decoder, by performing compression and decompression, would cause communications to be done more efficiently, at the same time distributing or relieving this function from the processor sub-systems. Such a device would perform the following functions: (a) encode and decode data; (b) measure and adjust the code assignments; (c) detect errors and automatically re-initiate the operation; and (d) control concurrent accesses. The advantage of this design is that it would make data compression transparent to the rest of the system.

In conclusion, the use of data compression allows data to be stored more efficiently and data communication to be done with shorter messages. However, many issues relating to the feasibility, the design of coding techniques, the reliability of the resultant codes, the implementation issues, etc., must be solved. It is contended that such solutions do not exist now and future study is necessary.

PROGRAM/DATA PLACEMENT AND MIGRATION

Definition of the problem

The problem is defined as follows: given a number of computers that process common information files, how can one allocate the files so that the allocation yields minimum overall operating costs. This problem has been called the *File Allocation Problem* (*FAP*).[9] A more general problem is the *Dynamic File Allocation Problem* (*DFAP*) in which the files are allowed to migrate over time so as to adapt to changing access requirements. The solution to this problem is affected directly by the query decomposition strategies and rather lightly by the data compression techniques. If the query is always decomposable into single file sub-queries, then the placement of each file may be optimized independently. Otherwise, the distribution of the files on the DCS must be optimized jointly and this increases the complexity of the problem significantly. On the other hand, data compression techniques generally affect the amount of data requested at a node and therefore the cost of an access is governed by the type of compression techniques used. By making certain assumptions on the query decomposition strategy and the compression technique, the FAP can be studied independently.

Motivations for file placement and migration

The major reason for allocating multiple copies of a file to certain parts of the system at certain times and the unnecessariness of keeping a copy of every file at every node all the time is because users have localities of access in any time interval. At any particular time, a file may be used by a group of users and it will continue to be used by the same group for a certain length of time. For a particular user, the file that he wants to access may be available locally, in which case, he can access the file with very little cost. If the file is not available locally, he would have to pay a cost in terms of delay in accessing the file and also introducing traffic in the network before he can make the access. It is under this situation that we should consider moving a copy of the file to his node. Introducing a new copy would also increase the cost in terms of storage space and the additional overhead in locking and concurrency control. Therefore, the decision of whether to introduce a new copy of a file involves a balance of the cost between the two cases. The costs, e.g. communication costs, are a function of the topology of the system, the type of communication protocols used and most importantly, the extensiveness of usage at a particular node. Further, as the request frequencies change, the file allocation on the system must also change accordingly. However, in this case, the cost in migrating the file from one node to another must also be taken into account in the file placement algorithm.

Previous work

Most of the previous studies on optimization are based on static distribution, that is, the allocation does not change with time. Some variation of dynamic distribution involves the application of static algorithms whenever need arises. A summary of the previous researches in this area is shown in Table I. These algorithms are very expensive to run in real time. A particular solution to this problem involving a 30 site network required about an hour on an IBM 360/91 computer.[16] The difficulty in optimization is also exemplified in Reference 33. Moreover, most of the algorithms are shown

TABLE 1.—A Survey of Previous Researches in File Placement/Migration

	Network Flow Techniques		Mathematical Programming & Exhaustive Searches					Heuristic	
	Stone[36-37]	Jenny[31,17]	Chu[3]	Casey[1]	Levin & Morgan[33,69]	Ghosh[18]	Foster et. al.[11]	Loomis & Popek[36,37]	Mahmoud & Riordon[38]
Assumption	Complete relations among objects; No redundant copies of objects.	Complete relations among objects	Complete relations among objects; File access is poisson.	All objects independent.	Only Program-data relation exists between objects.	All objects independent.	Star network: All objects independent.	Complete probabilistic relation among objects.	Indep. obj's; Query & ret'n traffic divided equally among alloc. nodes.
Parameters	Avg. amount of comm. traffic among obj.; Execution cost on a computer; Overhead in migration.	Functional equations represents constraints on which process placements depend; Communication demands between processes.	Storage cost; Transmission cost; File length; Request rate between files; Update rate between files; Maximum allowable access time; Storage capacity.	Storage cost; Query trans. cost; Update trans. cost; Query rate between nodes; Update rate between nodes.	Communication cost for query; Communication cost for update; Traffic rate for query/update from a node to a file via a program.	Data base with multiple target segment types; Queries with multiple target segment types	Queuing time & service time for transactions; Storage capacity; Avg. no. of messages in network; Avg. local processing; Average file length; Access frequency; H/W, S/W characteristics.	Inter-node trans. cost; Node capability; File length; Processing needs of file; Prob. of a request acc. an object; Prob. of a request/update is incident on a node; Prob. of 2 objects processed in parallel.	Communication cost; File storage cost; Query/update traffic & corresponding return traffic for each file at each node; Availability requirements.
Algorithm used	Network flow	Network flow & predicate calculus.	Integer programming	Path search on cost graph	Path search on cost graph	Combinatorial search thru. possible sol.	Queueing network alg.; Integer prog.	Clustering	Int. prog. or add-drop heuristic
Remarks	Static; Optimal for 2 processors; Sub-optimal for 3 processor system; Can calculate critical load factor for 2 processor system.	Do not consider multiple copy allocation; Min-cut alg. produces optimal subprocess groupings; Minimize communication overhead.	Algorithm very complex; Consider delay from network queuing approach.	Algorithm efficient; Independence of objects reduces allocation of multiple file to single file.	Algorithm efficient; Definite access relations among objects reduces the allocation of multiple file to single file.	Maximize no. of segments that query can retrieve in parallel from different nodes; Do not model communication delays.	Minimize difference from optimal branching probabilities; Algorithm complex.	Dynamic network behavior ignored; Maximize potential for parallelism.	Obtain both capacity assignment for links & file placements; Should consider query to be routed to nearest node & not distributed equally among all nodes.

to be NP-complete.** Although polynomial algorithms could exist for some special cases of the problem, e.g. the allocation of files in a two-processor system,[34] their use in practical applications is very limited. This result suggests that the distributed system designer should focus his attention to efficient heuristics.

Heuristics for file distribution on a DDB are usually interactive algorithms. A feasible solution can be generated. Users of some decision algorithms then have to decide whether to improve the solution or not and how to improve it. The disadvantages of these types of algorithms are that they usually find a local optimum instead of a global optimum and the validation of the algorithm is very difficult. For most cases, the heuristics can be shown to perform satisfactorily for some example values, but the algorithm is so complex that its worst case behavior is very difficult to determine. We first classify the three most commonly used heuristics, then we will discuss the application of a file assignment algorithm on this problem.

Hierarchical designs

This is a heuristic procedure in which attention is first restricted to the more important features of a system. In a file allocation problem, attention can first be restricted to geographical regions. After analysis has been performed and the files have been distributed to different geographical regions, attention can be directed to the less important details such as allocating files within a geographical region. This stepwise refinement procedure can continue down many levels. At each level of optimization, it is hoped that the effects on the optimization of the current level from the levels above and the levels below are very small. Neverthe-

** NP-complete problems[23] is a class of problems for which there are no known optimal algorithms with a computation time which increases polynomially with the size of the problem. The computation times for all known optimal algorithms for this class of problem increase exponentially with problem size, i.e., if n represents the size of the problem, then the computation time goes up as k^n where $k>1$.

less, iterations and design cycles may exist to refine the solution.

Clustering algorithms

Clustering algorithms are horizontal design processes which have a similar objective as hierarchical algorithms, namely, to reduce the complexity of the analysis in a large system. In a DDB, clusters can be formed on geographical distribution of access frequencies. The files are then allocated to clusters. The file allocation within a cluster may further be refined as in hierarchical algorithms.[26,27]

Add-drop algorithms

In applying this algorithm, a feasible distribution of files is first found. The total cost of the system can be improved by successive addition or deletion of file copies. When a feasible solution with a lower cost is found, it is adopted as a new starting solution and the process continues. Eventually, a local optimum is reached in which addition or deletion does not reduce the cost. The whole procedure can be repeated with a different starting feasible solution and several local optima can be obtained. By taking the minimum of all the local minima obtained, it is hoped that we can get very close to the global optimum.[28]

The above techniques are by no means complete. A combination of these techniques may be chosen by the designer. In the next section, we introduce a file assignment heuristic which utilizes some of the principles of add-drop algorithms.

File assignment algorithm

In this section, we present an algorithm which can be used to optimize the file placements on a DCS. The assumptions that we use in developing the model are:

1. **File accesses are independent**—By this, it is meant that there are no interactions among the files and all the accesses on the system are single file accesses. The placements of each file can therefore be optimized independently.
2. **It is assumed that all the constraints on the system can be represented in the form of costs.** For instance, paths linking two nodes in the network which violate some constraints such as the response time constraint, have a high inter-communication cost induced on them.
3. **It is assumed that for a certain time interval considered, it is divided into periods.** The file access behavior for a period are assumed to be estimated at the beginning of the period and the access behavior for the subsequent periods cannot be estimated at that point. With this assumption, it is possible to optimize the file allocations of each period independently and is not necessary to use dynamic programming to optimize the allocations for all the periods as done in Reference 25.

No assumption is made on the length of each period. Their lengths need not be identical and may be determined dynamically. The algorithm described in this section determines the file placements for each period, but no provision is made for determining the length of each period.

The symbols used in the model are:

n	number of nodes in the distributed system
a,b,c	indices for files
t	length of the current period of consideration, T
q_i^a	a random variable indicating the total amount of query accesses (including updates) at node i to file a (since we are optimizing each file independently, we will not write the superscript a in the remaining part of the discussion)
α_i	a random variable indicating the fraction of queries at node i that are updates to file a
$S_{i,j}$	per unit cost of accessing file a from node i to node j
$M_{i,j}$	per unit cost of multiple updating file a from node i to node j
$N_{i,j}$	per unit cost of moving file a from node i to node j
f_i	per unit cost of storing file a at node i
l_a	length of file a in bits
$Y_i=$	$\begin{cases} 0 & \text{if file a does not exist at node i during period} \\ 1 & \text{T otherwise} \end{cases}$
$X_i=$	$\begin{cases} 0 & \text{if file a does not exist at node i during period} \\ 1 & \text{T-1 otherwise} \end{cases}$
$K_0=$	$\{j: Y_j=0\}$=set of nodes without a copy assigned
$K_1=$	$\{j: Y_j=1\}$=set of nodes with a copy assigned
$K_2=$	$\{j: J_j=\text{unassigned}\}$=set of nodes unassigned
$K=$	$K_0 \cup K_1 \cup K_2$, $\lvert K\rvert=n$ (cardinality of K)

Consider the problem of allocating file a on the system at the beginning of period T of length t, the total amount of the retrievals and updates in this interval are estimated to be $q_i(1-\alpha_i)$ and $\alpha_i q_i$. The per unit cost of assessing, updating and transferring file a from node i to node j are $S_{i,j}$, $M_{i,j}$, and $N_{i,j}$ respectively. We assume that whenever a user at node i makes a request to a file not residing at node i, he will make the access at a node which has a copy of the file and with the lowest cost of access from node i. Our objective is to minimize the cost in the system. Our objective function is:

$$Z=\sum_{i=1}^{n} q_i(1-\alpha_i)\min_{j,Y_j=1} S_{i,j} + \sum_{j=1}^{n} Y_j(f_j+\min_{i,X_i=1} N_{i,j})l_a + \sum_{j=1}^{n}\sum_{i=1}^{n} Y_j\alpha_i q_i M_{i,j}$$

The first term in the above equation represents the query access cost; the second term represents the fixed cost of the period (cost of storage+cost of file transfers at beginning of

period); while the third term represents the multiple update cost of the system. We can rewrite the integer program as follows:

$$Z=\sum_{i=1}^{n} Q_i \min_{j,Y_j=1} S_{i,j}+\sum_{j=1}^{n} Y_j F_j \quad (1)$$

where

$$Q_i=q_i(1-\alpha_i) \quad (2)$$

$$F_j=(f_j+\min_{i,X_i=1} N_{i,j})l_a+\sum_{i=1}^{n} \alpha_i q_i M_{i,j} \quad (3)$$

subject to

$$Y_i=0,1 \quad (4)$$

The file assignment algorithm proposed here consists of the following basic parts:

1. Property or condition to assign or not to assign a copy of the file to a node.
2. Computation of a representative value for a candidate problem. (The state of a candidate problem is made up of the states of allocation to the n different nodes of the DCS. In general, the n nodes of the DCS can be partitioned into three sets, K_0, K_1 and K_2.) The function of the representative value is to illustrate the minimum of the candidate problem without actually enumerating all the allocations for the unassigned nodes.
3. Stopping the criterion.

The general steps of the algorithm are shown in Figure 4. We discuss each of these steps briefly here.

M-1. This is to initialize the candidate problem—all nodes are unassigned at this point. The candidate list, which is a list of states where an extra node from K_2 is added to K_0 or K_1 and its corresponding representative value, is assigned the empty set.

M-2 to M-5. These four steps essentially achieve the following: a node is selected from the un-assigned set, K_2, and is assigned a copy or not assigned a copy of the file. A representative value, which is chosen to be a lower bound estimated by solving the integer program (Equation 1) without the integrality constraints (Equation 4), is calculated for each of the corresponding candidate problems. The derivation of the linear programming lower bound for a candidate problem is shown in the appendix. The computed lower bound and the corresponding assignments are attached to the candidate list. These steps are then repeated for each of the nodes in K_2.

M-6. This step selects, from the candidate list, the candidate problem with the minimum lower bound and the corresponding assignment of nodes and use it for the next iteration. Steps M-2 to M-6 therefore have selected a node and have decided whether a copy should be placed at that node. This node is removed from the K_2 list.

M-7. The steps M-2 to M-6 are repeated until the K_2 list is empty. The overall computational complexity of the algorithm is $0(n^4)$. To further illustrate the steps of the algorithm, it is applied on the following example.

Suppose the following matrix represents the query cost $S_{i,j}$ for a five-node system.[1]

$$S=\begin{bmatrix} 0 & 6 & 12 & 9 & 6 \\ 6 & 0 & 6 & 12 & 9 \\ 12 & 6 & 0 & 6 & 12 \\ 9 & 12 & 6 & 0 & 6 \\ 6 & 9 & 12 & 6 & 0 \end{bmatrix}$$

Let

$$Q=[Q_i]=[\ 24\ \ 24\ \ 24\ \ 24\ \ 24\ ]$$

and

$$F=[F_i]=[\ 168\ \ 180\ \ 174\ \ 126\ \ 123\ ].$$

By enumerating the 2^5-1 possible allocations, it is found that a copy of the file should be allocated to nodes 1, 4 and 5 giving a cost of 705. The detailed application of the heuristic is shown in Figure 5, giving a solution of 717. In general, this method will give a solution very close to the optimal solution and the computation complexity is very low when compared with that of generating the optimal solution. The five examples on a 19-node problem solved by Casey[1] are compared with the solutions using the proposed heuristic and is shown in Table II. It is seen that the results do not deviate substantially from the optimum solutions. The results indicated here are somewhat preliminary. For the sake of simplicity, a more complicated algorithm is not presented. This algorithm, together with the analytical results and the theoretical studies will be presented in a future paper.

TASK SCHEDULING

Definition of the problem

This problem is related very strongly to the problem of query decomposition. After the query has been decomposed, the *Query Scheduling Problem* (*QSP*) is to sequence the processing of the sub-queries on the DDB for a given distribution of the files on the DCS defined by the FAP. Depending on the ways in which the sub-queries are processed, QSP can further be classified into *Sequential Query Scheduling Problem* (*SQSP*) and *Parallel Query Scheduling Problem* (*PQSP*). In SQSP, the sub-queries are processed

TABLE II.—Comparison between Casey's Solutions on a 19-node Problem and the Solutions of the Proposed Algorithm

Problem	Update/Query Percent	Casey's Optimum Cost	Cost using Proposed Algorithm	Time on CDC6400 (sec.)
1	10	117596	123073	7.7
2	20	188738	200971	7.7
3	30	242581	246107	7.7
4	40	291790	298690	7.7
5	100	431720	615342	7.7

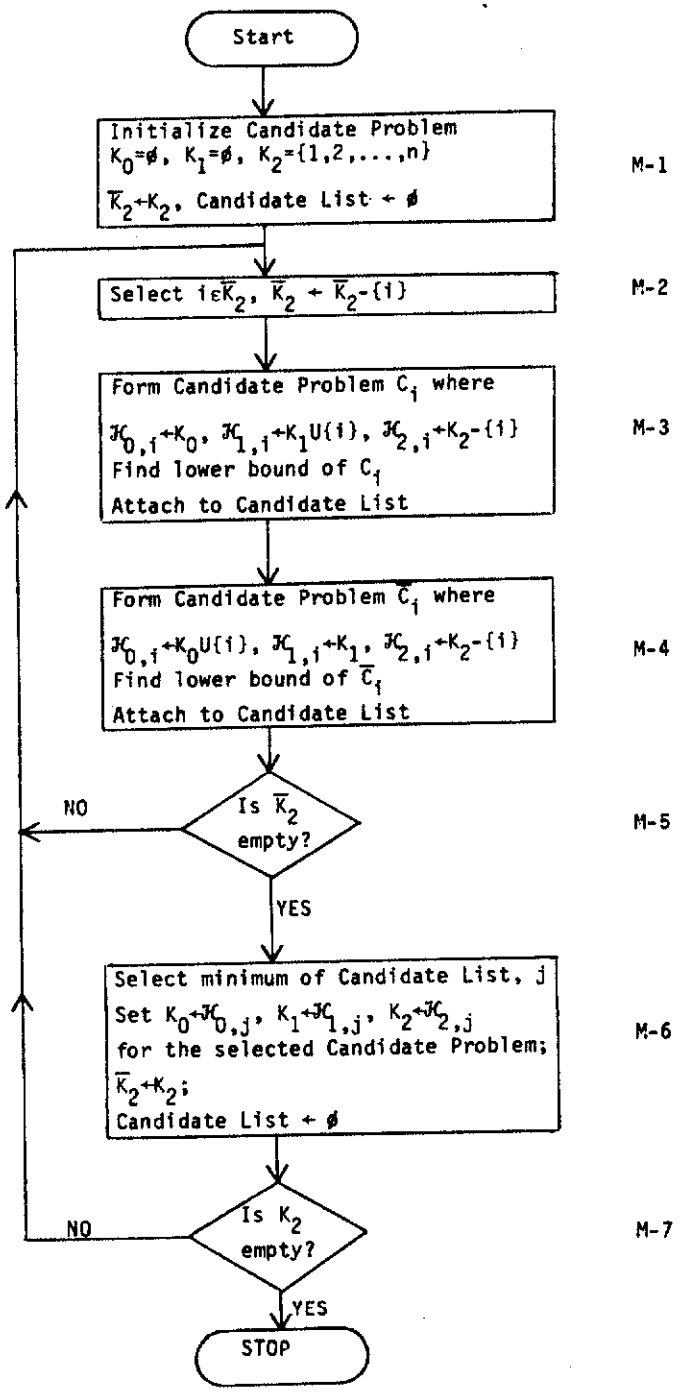

Figure 4—File assignment algorithm.

in sequential order. Using the results produced by the processing of the previous sub-query, the processing of the present sub-query will produce some results to be used by the next sub-query in sequence. If the files used by the sub-queries are separated geographically, intermediate results have to be transferred over communication lines. The objective is to minimize the amount of communications required. In PQSP, multiple queries are sent to different nodes and they are processed in parallel. The results after the processing are sent back to the requesting location. In this case, the response time may be smaller because all the communications are done in parallel (it is assumed that the major overhead is in communications and not in processing). For a compromise between the amount of communications and the response time, a combination of sequential and parallel query processing may be used. The QSP is a very similar problem which has been studied in other areas, e.g. the deterministic scheduling of multi-processors, the scheduling of requests in a computer system, etc. The results obtained there may therefore be extended to this study.

In order to solve the QSP, the notion of task must be defined. A *task* is defined to be a simple request which uses a resource for a finite amount of time. A request is said to be simple if no other resource is needed during the processing of this request. A complex request can always be broken down into a sequence of simple requests. A resource on a DDB can be physical, such as a communication channel, a processor, etc., or it can be logical, such as a file. The tasks are usually governed by a precedence graph so that a task cannot be processed until its predecessor has finished processing. The task scheduling problem is to sequence the processing of the tasks, subject to precedence constraints, so that some overall optimization criterion is satisfied. The criterion can be the maximum completion time of all the tasks if the objective is to maximize the throughput of the system; or it can be the sum of the completion times of all the tasks if the objective is to minimize the average response time.

To schedule the processing of queries, they are first decomposed into multiple tasks and the tasks are subsequently scheduled. The general task precedence graph for the processing of a query in the PQSP which requires the use of geographically distributed files is shown in Figure 6. On a DCS, the communication overhead, which includes time to set up the communication path and the queueing delay to transmit the messages, is usually much larger than the processing overhead for a query. Therefore, the time required to process a task at a node in Figure 6 is usually negligible as compared to the time to pass the results over the communication sub-system. The communication overhead on a DCS is dictated by the configuration of the interconnection mechanism. Many models have been designed to study the behavior of these delays, e.g. in Reference 23.

The QSP is usually solved with distributed control, that is, there does not exist a primary node which schedules all the processing of the queries on the DCS. Further, complete information for optimal scheduling are usually not available due to the high overhead in distributing them. The tasks are usually scheduled at each node sub-optimally without assembling all the necessary information before the scheduling.

Assumptions used to simplify the problem

Certain assumptions are often used so that the problem can be simplified.

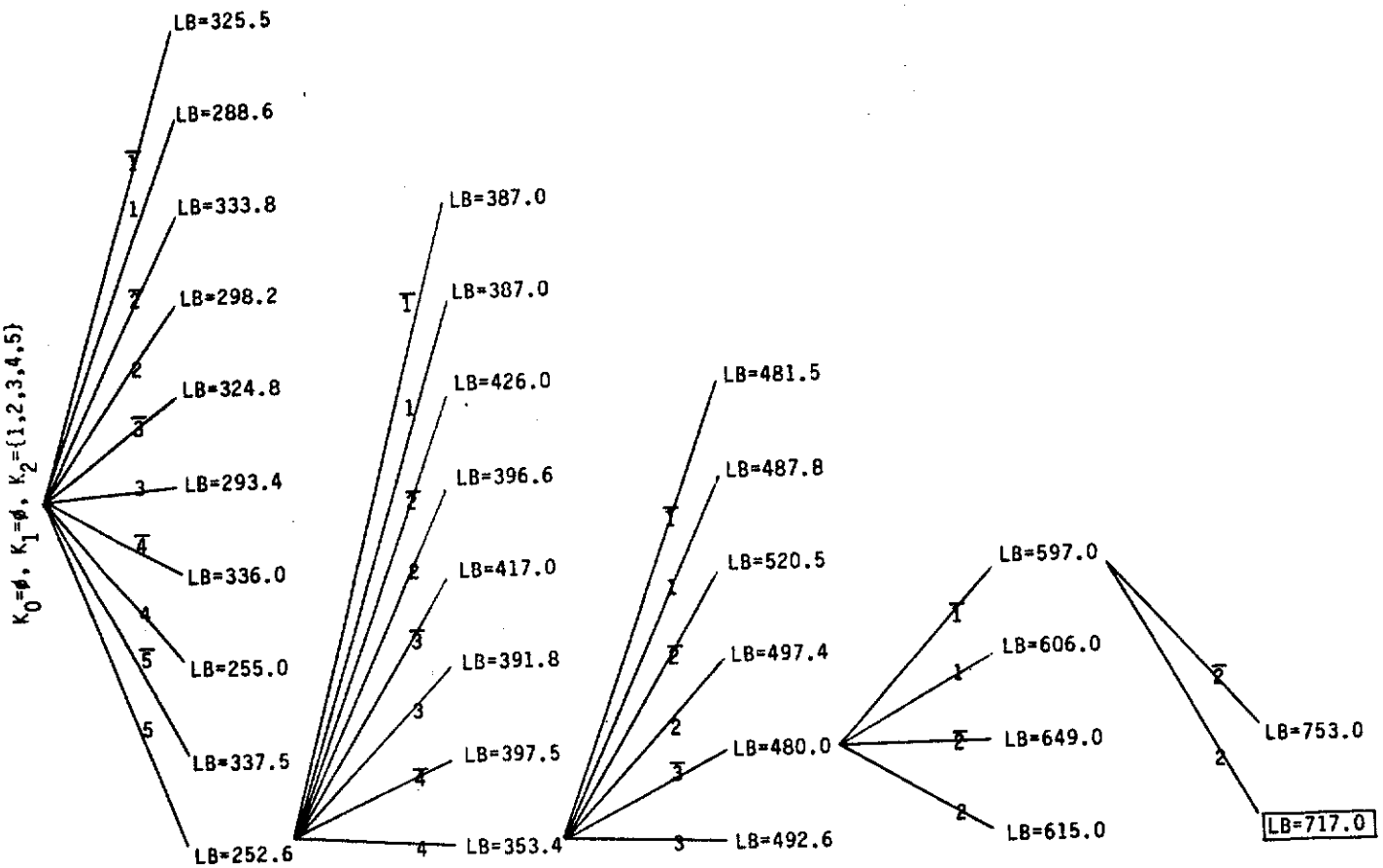

Figure 5—Application of file assignment on Casey's 5 node example.

Communication overhead

The processing overhead is usually much smaller than the communication overhead and they are usually ignored. This assumption will eliminate many tasks in the precedence graph.

Static versus dynamic algorithms

Static algorithms schedule a set of tasks available at the time of scheduling and a set of tasks that are known to arrive at fixed future times. The schedule does not change during the duration of the processing of these tasks. On the other hand, dynamic algorithms are more flexible and they reschedule all the available tasks whenever a new task comes in. The advantage of dynamic algorithms is that they allow task initiations to be dynamic and do not restrict the schedule to the order determined initially, but they have the disadvantage of larger overhead. With the use of dynamic algorithms, the assumption that there are precedence constraints among the tasks can also be relaxed. Whenever a task enters the system, all the tasks in the system are rescheduled dynamically. The choice between the use of static and dynamic algorithms is system dependent. If the arrivals of requests are indeterminate, then dynamic algorithms are usually better. On the other hand, if the arrivals of requests can be determined precisely, then static algorithms should be used. The choice between static and dynamic may also be dictated by the overhead in each type of algorithm, and a judicious choice must be made by the designer.

Deterministic versus probabilistic processing time

The processing time for a task can be assumed to be deterministic or probabilistic. In the deterministic case, it is possible to determine which order can best satisfy the optimization criterion and therefore all the tasks can be scheduled in a specific order. However, in a probabilistic case, it is difficult to do so when the processing times of all the tasks are governed by a common distribution. Certain assumptions, e.g. the distribution of job size, have to be made before an analytical evaluation is possible.[5] The theory of

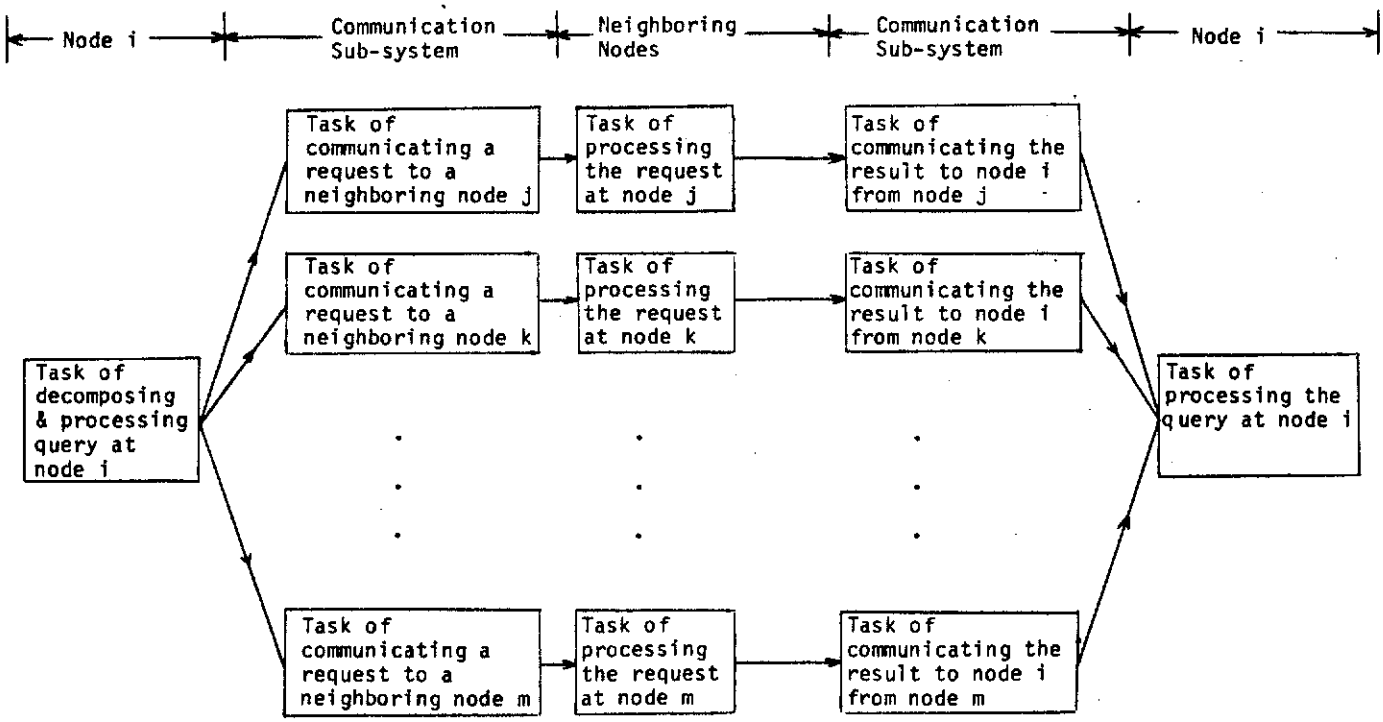

Figure 6—Task precedence graph for the processing of a query in the PQSP which requires the use of geographically distributed files.

scheduling developed now are mostly applicable to the deterministic case.[15] They can be used to approximate the probabilistic case when the average or the worst case processing times are used. Lastly, the difficulty of the scheduling problem can be assessed easily in most cases under the deterministic assumption. NP-completeness of the problem can usually be shown or a polynomial algorithm can be found. The QSP under the independent query assumption, can be shown to be NP-complete. Under this situation, the designer has to look for good heuristics which can be executed within the real time constraints. However, the evaluation of these heuristics is usually difficult. Evaluation methods and techniques are usually of three kinds, analytical techniques, simulations and approximation algorithms. Analytical techniques generally have to make some simplifying assumptions about the system parameters in order for the solution to be tractable and the results obtained are usually not accurate. On the other hand, simulations are almost always expensive to run, and it is difficult to exhaust all the possible cases of the system. A third type of evaluation algorithms are approximation algorithms.[40] There are two classes of these approximations, one guaranteeing a near-optimal solution always, and the other producing an optimal or near-optimal solution "almost everywhere." These types of algorithms are still in the research stage and an unifying approach in designing algorithms of this type is still lacking. The future trend is in the direction of investigating good approximation algorithms for scheduling queries.

CONCLUSION

In this paper, we have studied in detail the issues of resource management of data on a distributed data base. These issues are divided into three related levels, namely, the query level, the file level and the task level.

On the query level, the major issue is the decomposition of user queries so that parallelism in processing can be maximized and the amount of communications on the system can be minimized. It is shown that the approach using decomposition is deficient when the query is non-decomposable. In this case, the files needed to process the query must be moved to a common location before the query can be processed. An algorithm is proposed in this paper which preanalyzes the type of accesses on the system and introduces redundant information onto the files so that file transfers can be reduced.

On the file level, the issues are the compression of data for efficient storage and communication and the placement and migration of files for efficient accesses. In data compression, the existing techniques has been classified into four areas—run length encoding, differencing, statistical encoding and value set schemes. A multi-level compression scheme is proposed so that data is compressed through a set of cascaded stages. In the area of file placement and migration, a file assignment algorithm has been proposed. In general, this algorithm gives a solution very close to the optimal solution and the computation complexity is very low when compared with that of generating the optimal solution.

On the task level, the problem is to sequence the processing of the sub-queries for a given distribution of the files on the distributed system so that overlap in processing can be maximized. It is shown that the problem of query scheduling on a distributed data base is NP-complete. The future directions of research are therefore in the search of effective approximation algorithms.

The issues we have discussed in this paper encompasses the spectrum from the processing of the query to the scheduling of the requests. However, many other issues may arise in the design of the data base. These include other issues in operational control, such as directory management, concurrency control, security and privacy, etc. and they may affect the strategies used in data management. The study of these issues, however, are beyond the scope of this paper.

APPENDIX—DERIVATION OF THE LOWER BOUND OF A CANDIDATE PROBLEM

We derive in the appendix the lower bound of a candidate problem given the state of it. We can rewrite the objective function (Equation 1) on condition on K_0, K_1 and K_2.

$$Z = \sum_{i \epsilon K_1} F_i + \sum_{i \epsilon K_0} Q_i * \min_{j, Y_j = 1} S_{i,j} + \sum_{i \epsilon K_2} Q_i * \min_{j, Y_j = 1} S_{i,j} + \sum_{i \epsilon K_2} F_i Y_i$$

We have

$$\min Z = \sum_{i \epsilon K_1} F_i + \sum_{i \epsilon K_0 \cup K_2} Q_i * \min_{j, Y_j = 1} S_{i,j} + \sum_{i \epsilon K_2} F_i Y_i \quad \text{(A-1)}$$

subject to $Y_i = 0, 1$

where Q_i is defined in Equation 2, and F_i is defined in Equation 3.

Equation A-1 is a non-linear integer program, we can rewrite it in the form of a linear program. Let

$U_{i,j}$ = fraction of accesses made from node i to node j;
P_i = set of indexes of those nodes that can access node i;
n_i = cardinality of P_i.

$$\min Z = \sum_{i=1}^{n} \sum_{j=1}^{n} Q_i S_{i,j} U_{i,j} + \sum_{i=1}^{n} F_i Y_i \quad \text{(A-2)}$$

$$\text{s.t.} \sum_{j=1}^{n} U_{i,j} = 1 \qquad i = 1, \ldots, n \quad \text{(A-3)}$$

$$0 \le \sum_{i \epsilon P_j} U_{i,j} \le n_j Y_j \qquad j = 1, \ldots, n \quad \text{(A-4)}$$

$$Y_i = 0, 1 \quad \text{(A-5)}$$

Equation A-3 is true because the total amount of fractions must be summed to 1. Equation A-4 is derived by summing over all $i \epsilon P_j$, the inequality $0 \le X_{i,j} \le Y_i$ which says that only nodes with a copy of the file can supply users' demands. By relaxing constraint A-5, it becomes a linear program and the value of Z obtained will provide a lower bound to the original integer program. The solution to the linear program (Equation A-2 to Equation A-4) has been solved in Reference 7. The solution is:

$$Y_j = \frac{1}{n_j} \sum_{i \epsilon P_j} U_{i,j} \quad \text{(A-6)}$$

$$Z_{i,j} = \begin{cases} 1 & \text{if } S_{i,j} + \frac{g_j}{n_j} = \min\limits_{k \epsilon K_1 \cup K_2} \left(S_{i,k} + \frac{g_k}{n_k} \right) \\ 0 & \text{otherwise} \end{cases} \quad \text{(A-7)}$$

$$g_k = \begin{cases} F_k & k \epsilon K_2 \\ 0 & k \epsilon K_1 \end{cases}$$

The complexity of the solution is $O(n^2)$.

REFERENCES

1. Casey, R. G., "Allocation of Copies of a File in an Information Network," *AFIPS SJCC*, 1972, pp. 617-625.
2. Chen, T. C., and I. T. Ho, "Storage Efficient Representation of Decimal Data," *CACM*, Vol. 18, No. 1, Jan. 1975, pp. 49-52.
3. Chu, W. W., "Multiple File Allocation in a Multiple Computer System," *IEEE Trans. on Computers*, Vol. C-18, No. 10, Oct. 1969, pp. 885-889.
4. Codd, E. F., "A Relational Model of Data for Large Shared Data Bases," *CACM*, Vol. 13, No. 6, June 1970.
5. Coffman, E. G., Jr., and P. J. Denning, *Operating System Theory*, Prentice Hall Inc., N.J., 1973.
6. Date, C. J., *An Introduction to Data Base Systems* , 2nd Edition, Addison-Wesley, 1977.
7. Efroymson, M. A., and T. C. Ray, "A Branch and Bound Algorithm for Plant Location," *Operations Research*, May-June 1966, pp. 361-368.
8. Epstein, et. al., "Distributed Query Processing in a Relational Data Base System," Report No. UCB/ERL M78/18, Electronics Research Laboratory, University of California, Berkeley, 1978.
9. Eswaran, K. P., "Placement of Records in a File and File Allocation in a Computer Network," *Information Processing 74*, IFIPS, North Holland Publishing Co., 1974.
10. Fajman, R., and J. Borgelt, "WYLBUR: An Interactive Text Editing and Remote Job Entry System," *CACM*, Vol. 15, No. 5, May 1973, pp. 314-333.
11. Foster, D. V., et. al., "File Assignment in Star Network," *Proc. of the 1977 Sigmetrics/CMG VIII Conf. on Comp. Perf.: Modelling, Measurement and Management*, Washington, D.C., Nov. 1977, pp. 247-254.
12. Fry, J. P., and E. H. Sibley, "Evolution of Data Base Management Systems," *Computer Surveys*, Vol. 8, No. 1, March 1976, pp. 7-42.
13. Ghosh, S. P., "Distributing A Data Base with Logical Associations on a Computer Network for Parallel Searching," *IEEETSE*, Vol. SE-2, No. 2, June 1976, pp. 106-113.
14. Goldstein, R. C., and A. J. Strand, "The MacAIMS Data Management System," *Proc. ACM-SIGFIDET Workshop*, Nov. 1970, pp. 201-229.
15. Graham, R. L., et. al., "Optimization and Approximation in Deterministic Sequencing and Scheduling: A Survey," *Proc. of Discrete Optimization 1977*, Vancouver, Aug. 1977.
16. Grapa, E., and G. G. Belford, "Some Theorems to Aid in Solving the File Allocation Problem," *CACM*, Vol. 20, No. 11, Nov. 1977, pp. 878-882.
17. Hofri, M., and C. J. Jenny, "On the Allocation of Processes in Distributed Computing Systems," *IBM Research Report*, RZ905, 1978.
18. Hu, T. C., and J. Tucker, "Optimal Computer Search Trees and Variable Length Alphabetic Codes," *SIAM J. of App. Math.*, Vol. 21, 1971, pp. 514-532.
19. Huang, T., "An Upper Bound on the Entropy of Run Length Coding," *IEEE Trans. on Info. Theory*, Vol. IT-20, No. 9, Sept. 1974, pp. 675-676.
20. Huffman, D. A., "A Method for the Construction of Minimum Redundancy Codes," *Proc. IRE*, Vol. 40, Sept. 1952, pp. 1098-1111.

21. Jenny, C. J., "Process Partitioning in Distributed Systems," *IBM Research Report*, RZ873, 1977.
22. Karp, R. M., "Reducibility among Combinatorial Problems," *Complexity of Computer Computations*, R. E. Miller and J. M. Thatcher (eds.), Plenum Press, New York, 1972, pp. 85-104.
23. Kleinrock, L., *Queuing Systems, Vol. II: Computer Applications*, John Wiley & Sons, 1976.
24. Knuth, D. E., *The Art of Computer Programming, Vol. 3: Sorting and Searching*, Addison-Wesley, Reading, Mass., 1973.
25. Levin, K. D., and H. L. Morgan, "Optimizing Distributed Data Bases—A Framework for Research," *Proc. NCC*, 1975, pp. 473-478.
26. Loomis, M. E. S., "Data Base Design: Object Distribution and Resource Constrained Task Scheduling," Ph.D. Dissertation, Comp. Sci. Dept., UCLA, 1975.
27. Loomis, M. E. S., and G. J. Popek, "A Model for Data Base Distribution," *Symp. on Trends and Applications*, 1976, Computer Networks, IEEE, 1976, pp. 162-169.
28. Mahmoud, S., and J. S. Riordon, "Optimal Allocation of Resources in Distributed Information Networks," *ACM Trans. on Data Bases*, Vol. 1, No. 1, March, 1976, pp. 66-78.
29. Morgan, H. L., and K. D. Levin, "Optimal Program and Data Locations in Computer Networks," *CACM*, Vol. 20, No. 5, May 1977, pp. 315-322.
30. Ramamoorthy, C. V., and T. Krishnarao, "The Design Issues in Distributed Computer Systems," *Infotech State of the Art Report on Distributed Systems*, 1976, pp. 375-400.
31. Ramamoorthy, C. V., G. S. Ho and B. W. Wah, "Distributed Computer Systems—A Design Methodology and its Application to the Design of Distributed Data Base Systems," to appear *Infotech State of the Art Report on Distributed Systems*, 1979.
32. Schieber, W. D., and G. W. Thomas, "An Algorithm for Compaction of Data," *J. Library Automation*, Vol. 1, No. 4, Dec. 1971, pp. 198-206.
33. Sickle, L. V., and K. M. Chandy, "Computational Complexity of Network Design Algorithms," *Information Processing 77*, IFIPS, North Holland Publishing Co., 1977.
34. Stone, H. S., "Multi-processor Scheduling with the Aid of Network Flows," *IEEE Trans. on Software Engineering*, Vol. SE-3, No. 1, Jan. 1977, pp. 85-93.
35. Stone, H. S., "Critical Load Factors in Distributed Computer Networks," Report ECE-CS-77-2, University of Massachusetts, Amherst, Mass., 1977.
36. Stone, H. S., "Program Assignment in Three-Processor Systems and Tricut Partitioning on Graphs," Report No. ECE-CS-77-7, University of Massachusetts, Amherst, Mass., 1977.
37. Stone, H. S., and S. H. Bokhari, "Control of Distributed Processes," *Computer*, July 1978, pp. 97-106.
38. Titman, P., "An Experimental Data Base System Using Binary Relations," *Proc. IFIP Working Conf. on Data Base Management*, Corsica, 1974.
39. Todd, S. J. P., "The Peterlee Relational Test Vehicle—A System Overview," *IBM Systems Journal*, Vol. 15, No. 4, Dec. 1976, pp. 285-308.
40. Weide, B., "A Survey of Analysis Techniques for Discrete Algorithms," *Computing Surveys*, Vol. 9, No. 4, Dec. 1977.
41. Wong, E., and K. Youssefi, "Decomposition—A Strategy for Query Processing," *ACM Trans. on Data Bases*, Vol. 1, No. 3, Sept. 1976, pp. 223-241.
42. Wong, E., "Restructuring Dispersed Data from SDD-1: A System for Distributed Data Bases," *Comp. Corp. of America Tech. Rep.* CCA-77-03, 1977.

ACKNOWLEDGMENT

We would like to thank Mr. C. R. Vick and Mr. J. E. Scalf for many helpful discussions related to this work.