Design of distributed data base systems

R. A. Davenport*

London School of Economics, Houghton Street, London WC2

Distributed data base systems combine the central control of integrated data with the resource sharing of distributed processing. This paper examines the various structuring possibilities available to the designer of a distributed data base system and details the options available at each stage of the design process.

(Received August 1979)

The geographically distributed computer user currently has two approaches available for implementing an integrated data processing system.

(a) The data files and processing can be centralised with access by remote systems (centralised system).

(b) The system can be implemented with a number of computers, typically small, at the various locations, each with its own auxiliary storage and terminals (distributed system).

Centralised systems have predominated in the computer field for a number of years. There are a number of reasons for this. Designers found that a few large computers could do the work of several small or medium ones at lower cost (Sharpe, 1969). A central computer centre can support a wide range of capabilities that would be prohibitively expensive to provide at each of several smaller centres. A continuing lack of qualified computer staff reinforced this significant cost benefit and the emergence of data base technology (Martin, 1975; Palmer, 1975) that enabled organisations to integrate data concerning their operations, accelerated this trend of centralisation.

In addition, if operations are centralised the attraction and development of competent technical staff is made easier as is the maintenance of complex hardware and software. Control of operations by management is simplified if those operations are centralised. But at the present time an opposing trend towards distributed systems is developing which has been evident for the past three or four years (Canning, 1974). Over half of respondents to a recent survey in Europe indicated that they are planning to implement or are implementing a distributed system (Datamation, 1976). This change in direction has come about for a number of reasons. Progress in the concentration of components within silicon chips has been spectacular and this in turn has led to rapid decreases in the costs of computing hardware (Butler, 1976). This progress in cost performance has been exploited by minicomputer manufacturers considerably earlier than it has been by mainframe manufacturers. This, together with the absorption of a large proportion of the power of a mainframe by its complex operating system has meant that the economy of scale arguments of processors, i.e. Grosh's Law, do not hold to the same extent as previously. However, certain hardware economies of scale do exist, particularly in auxiliary storage and certain operating costs, e.g. personal costs continue to exhibit substantial economies of scale.

In addition, the trend to remote input and processing will continue because users wish to access the computer directly, i.e. terminal based systems. A recent European study (Peters and Bunn, 1976) predicts that the terminal population of Europe which is now 450,000 will grow to two million by 1985. The majority of terminal based systems up to the present time have been centralised. Centralised systems, however, have a number of disadvantages. First, with a centralised system, data communication costs are significant and have not shown the same rate of cost decrease as hardware. While terminal costs for a specific application may change very little, there is expected to be a steadily reducing cost for producing the information and a steadily increasing cost for transmitting the data. In particular, Donaldson (1976) has surveyed overall cost trends for the United Kingdom and his results are shown in Fig. 1. Second, management has also become aware that a centralised system, which may have economic benefits in areas such as operations, introduces a number of undesirable side effects. These include the complexities of trying to service an increasing community of remote users becoming greater due to the number of user types increasing and the need to deal with more concurrent events. Third, and perhaps most important, management must be willing to endorse and enforce standardised, centralised data processing project development which is often contrary to the management philosophy of the organisation which is hierarchical and decentralised.

As well as overcoming the above problems, distributed systems have a number of positive points. First, distributed systems can be developed on an incremental basis with only as much computing power being installed as is required at that moment in time. Second, reliability of a distributed system is likely to be better. If a large central system goes down, everything attached to that system stops. If one node of a distributed system goes down processing can continue on other nodes even though it may be with reduced capacity. Distributed systems, then, are aimed to fit more closely to the structure of the organisation by moving the computing power to the user. A more extensive discussion of the above points can be found in Down and Taylor (1976), Canning (1976a; 1976b; 1976c), Champine (1977) and Emery (1977). Since both centralised and distributed systems have significant advantages, a systematic methodology (Davenport, 1978a), for choosing between the two approaches should be employed. A summary of the advantages of centralised systems is given in Table 1 and of distributed system in Table 2.

<table>
<thead>
<tr>
<th>Table 1</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Centralised system advantages</strong></td>
</tr>
<tr>
<td>Operations economy</td>
</tr>
<tr>
<td>Certain hardware economies</td>
</tr>
<tr>
<td>Unified control of operations</td>
</tr>
<tr>
<td>Powerful capabilities</td>
</tr>
<tr>
<td>Increased systems integration</td>
</tr>
<tr>
<td>Competability</td>
</tr>
<tr>
<td>Development of professional staff</td>
</tr>
<tr>
<td>Simpler maintenance</td>
</tr>
<tr>
<td>Easier security</td>
</tr>
</tbody>
</table>

*Since we originally received Mr Davenport's paper we have been saddened to learn that he died of a stroke on 27 May 1980.
to the application programs. At each node the system software consists minimally of an operating system, a data base management system (DBMS) (Martin, 1975; Palmer, 1975) and communications management. With the exception of the latter, the software components may also be similar or dissimilar. The aim then of the distributed data base is to gain the benefits of the data base approach by the central control of integrated data, whilst at the same time not imposing centralisation on data processing.

**Required software**

There are a number of functions required to be handled in a distributed data base system. Ideally, there would be only one integrated piece of software. However, the most likely method of development, because of the amount of effort required, is that additional software will be written to interface to the standard components supplied by the manufacturer or software vendor in order to handle the distributed data base aspect. The standard components would be the same for either a single computing facility or a distributed system (Fig. 2).

The first component is the standard operating system of each computing facility. The second required component is network communication software (Canning, 1976b) that allows information exchanges between programs that are remote from one another. The architecture of such software involves three levels of control for information changes. These are (a) control of the physical transmission of data, (b) control of logical message links between processes (running programs) and (c) control of specific kinds of dialogue (e.g. file transfer).

The third required component is a data base management system. Ideally, the data base management system should be available on minicomputers as well as mainframes so as to take advantage of the cost performance benefits that are present in such types of computing facilities and to allow the power of the computing facilities to be matched to the local needs while at the same time providing data base facilities. The fourth component, the additional component, is a control structure or network component. It is this component that handles the distributed data base aspect. This component is responsible for knowledge of where data is held in the system, for achieving transparency, for maintaining integrity, for providing translations and for preserving data base consistency. In the majority of existing distributed systems the network components tasks are handled by application programs. A critical factor for software complexity is whether the control of the network component is centralised or distributed. With centralised control a central site is in overall control. With distributed control there exists dynamic master/slave switching between nodes so that communication can occur between

---

**Table 2**

<table>
<thead>
<tr>
<th>Distributed system advantages</th>
</tr>
</thead>
<tbody>
<tr>
<td>Communication failsoft capability</td>
</tr>
<tr>
<td>Central site failsoft capability</td>
</tr>
<tr>
<td>Lower communications data rate and cost</td>
</tr>
<tr>
<td>Configuration specialisation</td>
</tr>
<tr>
<td>Exploitation of low cost devices</td>
</tr>
<tr>
<td>High system performance</td>
</tr>
<tr>
<td>Modular implementation</td>
</tr>
<tr>
<td>Modular upgrade</td>
</tr>
<tr>
<td>Transfer of computing resources to user environment</td>
</tr>
<tr>
<td>Simplified costing</td>
</tr>
</tbody>
</table>

**Distributed data base systems**

Distributed systems in the terms that they will be discussed in this paper are identified by three criteria (Emery, 1977; Cox, 1976).

- The first criterion is that the system possesses two or more geographically displaced computing facilities (nodes). A computing facility consists of a processing unit with main store, associated auxiliary storage and communication capabilities, the node need not have the same computing facilities, and typically serves a separate organisational subunit. Each node must be application logic oriented so that application programs are loaded and run at each node.

- The second criterion is that the two or more displaced computing facilities are linked, which is accomplished normally through telecommunications.

- The third criterion is that there are relatively weak interactions among the distributed computing facilities. This characteristic excludes tightly coupled processors working in parallel on a cooperative computing task.

A particular form of a distributed system is a distributed data base system. In the broadest sense a distributed data base system could be considered to exist when each processor of the system has permanent files attached. However, this does not imply any relation between the data bases, which are seen to be merely elements of the different processors.

Therefore, a distributed data base exists when a logically integrated data base is physically distributed over several distinct linked computing facilities—logical integration means that each node has potential access to the entire data base. Ideally, the physical distribution of the data base is transparent.
them on an adhoc basis without a central site being employed. In CODASYL data base terms (Martin, 1975; Palmer, 1975) there will have to exist a data description of the entire logical data base, the superschema, which is responsible to the network component. Each computing facility will have a data description of its own local data base, the schema, and data descriptions of the data required for each of its programs, sub-schemas, may be repeated in a number of computing facilities or they may be unique to one. A sub-schema may involve data held on a number of computing facilities. Where a data record exists on more than one computing facility, some indication of occurrences will also have to be provided. This is illustrated in Fig. 3.

When an application program makes a request for data, the network component determines if the data is held locally. If it is not, then a message is sent via the communication software to another node in the system. If the system is centrally controlled the request will go via the central node otherwise it will be sent direct. When the structure and location of data changes in the system, the network component must make appropriate changes to the superschemas, schemas and sub-schemas in all relevant computing facilities.

System design objectives

In any undertaking, the objectives that are expected to be achieved by that undertaking should be clearly defined. Computer systems design is no exception. There are a number of objectives that a design may be expected to achieve—unfortunately, only some are quantifiable. Hence the ability of the design to achieve one or several of these unquantifiable objectives rests on the ability of the designer to make subjective judgement and cannot be achieved by an automatic process. The relative importance of each design objective will have a considerable effect on the design strategy.

The objectives that are likely to be most important to the majority of users are:

1. **Performance**—the ability to process transactions at a given rate (throughput) within given time limits (response). The requirement may be applied to a single transaction type or to the average for all transaction types. The response time requirement for a single transaction type may be such that one is willing to reduce the overall throughput in order to guarantee that response time requirement. A distributed system has particular advantages as regards this objective because of the close proximity of processing power to the user's location.

2. **Economy**—the ability to produce adequate performance at an acceptable cost. This can be expressed as the amount of resources consumed by the system in terms of CPU utilisation, secondary storage space, communication, links, terminals, operations personnel, floor space, etc.

3. **Implementability**—the ability to produce the system possess-

4. **Availability**—the ability of the system to produce the required performance over an acceptable proportion of its operating life. This is a function of the system's ability to recover from software or hardware failures, the likelihood of such failures (reliability) and the correct processing of transactions (accuracy). Such matters as the speed of detection of failures, time to recover from failures, mean time between failures, likelihood of errors in the input being processed and therefore requiring processing at a later stage, are included in this requirement. The failsoft characteristics of distributed systems give them an advantage in meeting this objective.

5. **Flexibility**—the ability of the system to deal with a change in the flow of transactions, a change in an application or to incorporate a new application at minimum cost or make use of a new technology. This requirement then is essentially a measure of how resilient the system is to change. Such a change may be due to an increase in the volume of transactions over the design level, variation in the ratios between the volumes of different transaction type, be it in the dialogue, the data structure or the processing performed. Distributed systems have an advantage in meeting the flexibility objectives because of the localised effect that any change has, i.e. only one site at a time affected.

The literature tends to stress economy and performance as the overriding objectives. In many individual situations this will not be the case.

**Data analysis**

The increasing use of data base technology is leading to new attitudes towards data processing. Traditionally the emphasis during system analysis has been on the 'processing'; with the data base approach the emphasis changes towards the 'data' on which the application systems are based. Previously, data structures have been designed for particular applications, 'application dedicated files'; the data base is centred around the 'shared data resource', with data structured independently of individual application requirements. Data analysis is concerned with identifying the data resources of an organisation. The objectives of data analysis is to determine the fundamental nature of an organisation's data resources and to organise and document all relevant facts concerning this data and thus provide a sound basis for data base design. Data analysis, which is more fully described elsewhere (Davenport, 1978c) is regarded as consisting of two dependent projects: entity analysis and functional analysis.

Entity analysis provides a means of understanding and documenting a complex environment in terms of its entities and their attributes and relationships. The emphasis is on the things which make up the environment, rather than on the details of how these things are used.

Functional analysis is concerned with an understanding and documentation of the basic business activities with which the organisation is concerned.

**Entity analysis**

1. The first step in entity analysis is to define the areas of data to be analysed. This can be done by identifying each of the
main application areas that the system is to handle, either now or in the foreseeable future. Examples of main application areas would be order entry, inventory control, production control, etc.

2. Next, for each application area, determine the principal entity types. An entity is a thing (person, place, event, object) that is of interest to the organisation. There is a distinction between entity and entity type. 'Customer' is an entity type while 'Smith' is an entity. Each entity type must be capable of precise definition; each entity must be uniquely identifiable. For example, in one organisation a customer may be defined as a person or organisation to whom they invoice; each customer may be uniquely identified by his name and address.

3. Having identified the entities, the next step is to identify the relationships that exist between the entities. For example, when one says that a person called J. Smith 'lives in' the house called 10 Pine Avenue, 'lives in' is a relationship between person and house. Relationships can involve varying numbers of entities. This complexity can be expressed diagrammatically in a form similar to the entity-relationship model (Chen, 1976), which in turn is an extension to the data structure diagram of Bachman (1969). The resulting diagram, an entity model, is for the application area. Two graphical symbols are employed; a box to represent an entity type and a line to represent a relationship type. The degree of the relationship can be one to one (husband and wife), one to many (father and children) and many to many (employees and projects). Arrows are inserted in the direction of 'many'.

4. The properties or descriptive values associated with entities must also be considered during entity analysis. These are termed attribute types. Among the attributes of customer entities are likely to be name, address, account number, date of last payment, size of customer, credit rating and many others. One attribute type must be chosen to identify the entity uniquely, i.e. the key or the identifier.

5. The normalisation process of E. F. Codd (1971) can be used to examine the attributes associated with an entity to ensure that there are no hidden dependencies between attributes which ought to be represented directly in the entity model. Normalisation requires three actions to be performed on the attributes of an entity. First, repeating groups are removed. Next, attributes are removed which are dependent on only some of the identifying attributes. Finally, any attributes are removed which are not directly dependent upon the identifying attributes.

6. Steps 2 to 5 are repeated for all application areas and the entity models produced are integrated to produce an overall entity model for the area of data of interest.

7. The functional entity models, representing each of the business activities analysed must be checked against the overall entity model, representing the organisation as a whole. Inevitably, certain aspects of the environment are detected during functional analysis which have been missed during entity analysis.

Functional analysis
Closely connected with entity analysis is functional analysis, which is concerned with an understanding and documentation of the basic business activities with which the organisation is concerned. Functional analysis can be divided into the following phases.

1. In each application area a number of functions will be carried out. It is necessary to identify each of these functions and to describe them. This is the first stage. Normally functions will be triggered by an event, where an event can be defined to be a stimulus to the organisation and functions can be defined to be the tasks that must be carried out as a direct result of an event, e.g. 'an order is placed' is an event while 'record the order', 'schedule production' and 'produce the invoice' are functions.

2. From the description of the function it is possible to produce the access path, the entities, attributes and relationships involved in each function, and the function entity model, which is a subset of the complete entity model that is of interest to the function. Each access path is analysed to determine the point of entry to the entity model. Then the path through the entity model taken by the function is documented. The use of relationships to retrieve information is recorded. The criteria used to select a particular entity or group of entities are documented. If a function creates, modifies or deletes an entity or relationship, then this is documented. In addition, the frequency of use of the function should be estimated and also the number of occurrences accessed for each entity on the access path. For each entity an attribute usage matrix should be produced which records the functions accessing the entity versus the attributes employed and also how the attribute is employed (created, destroyed, retrieved, modified).

Other information that is required for design should be recorded for each function. This will include response time requirement, frequency, availability and currency, which is a measure of how 'up-to-date' the data required for the function has to be. For the entity types additional information recorded should include occurrences, growth rates, accessing rights and ownership.

Data analysis is very much a reiterative process and it is unlikely that a completely satisfactory entity model will be produced on the first iteration. The process described above should be employed regardless of whether a centralised or distributed approach is to be employed. In order to aid the design process, some additional information should be recorded. This information is concerned with identifying the locations of functions and the data requirements of those locations. Matrices are produced of location versus function, location versus entity type and location versus attribute type.

Distributed data base configuration
If a systems designer has decided to implement a distributed data base system, there are a number of decisions that have to be made concerning the configuration of the system. These decisions concern the following factors:

1. Control structure
2. Data distribution
3. Accessing method
4. Directory
5. Currency requirement.

Control structure
Two distinct control structures (Canning, 1974), can be identified for distributed systems:

1. Central control
2. Distributed control

The term 'control' means the handling of message traffic between nodes, the synchronising of update transactions and the initiation of recovery actions affecting more than one node, i.e. network component tasks. With central control overall control of the system is vested in one node. Typically such a system will be implemented physically in a hierarchical manner.
(Fig. 4) with the computing facilities sharing tasks in a structured manner, with each component to some degree controlled by the higher level members of the hierarchy. The central control node, which is responsible for overall control, is located at the top of the hierarchy.

With distributed control (Fig. 5) all computing facilities cooperate at an equal level logically, to perform a set of tasks. Linking between nodes is on a dynamic basis (dynamic master/slave switching) and control of the data flow is performed by the node that originally established the link.

**Data distribution**

The distributed data base itself can be a replicated and/or a partitioned data base. In a replicated data base (Fig. 6) all or part of the conceptual data base is replicated at two or more nodes. Fig. 6 shows a distributed data base where all occurrences of each data type are held in the storage device at location A, while only some of the occurrences are held at location B and C. A number of possibilities exist concerning the data structure. For example, each location may have the same data structure as shown in Fig. 7. The data structure is in the form of a Bachman diagram (Bachman, 1969). The boxes represent record types and the lines represent links between record types. An alternative is for the complete data structure to be present at only one location and the other locations contain only subsets of the complete structure as shown in Fig. 8.

In a partitioned data base (Fig. 9), the conceptual data base is separated into sections and the sections spread across multiple facilities. A number of possibilities exist concerning the data structure. For example, the data structure may be repeated in each location as shown in Fig. 10, so the partition is only by data occurrence or value. As an alternative, the data structure itself may be partitioned between locations so that a particular data type is found in only one location as shown in Fig. 11.

It should be stressed that partitioning and replicating of the
data base are not necessarily mutually exclusive. Both types of
data distribution may be present in a particular implementation.
For example, Fig. 6 which shows replication of the data
between A and B and between B and C and between A and C
also exhibits some degree of partitioning of the data base
between B and C.
The distribution of data is one problem of data base design
peculiar to the distributed environment. There is a considerable
literature (for example, Levin and Morgan, 1975; Mahmoud
and Riodan, 1976) concerning this problem, frequently
referred to as file allocation. These research efforts have
applied classical mathematical programming techniques to
variants of the following problem.

Given: a description of user demand for service stated as the
volume of retrievals and volume of updates from each node of
the network to each file (piece of the data base).

Given: a description of the resources available to supply this
demand stated as the network topology, link capacities and
costs and the node capacities and costs.

Determine: an assignment of files to nodes which does not
violate any capacity constraints and which minimises total
costs.

The variation on this problem which have been explored
include:
(a) consideration of time varying and uncertain demand,
(b) consideration of the dual problem of constraining costs and
determining capacities; and
(c) the use of heuristics to reduce the computational complexity
of finding an acceptable solution.

However as Rothnie and Goodman (1977) point out, the
problem stated above is only a small piece of the file allocation
problem in a distributed data base as discussed in this paper.
There are three specific shortcomings of the existing body of
research in this field which limit the usefulness of these results:
1. The user demand model is stated as a set of requirements to
access a given file from a given node. This model does not
adequately reflect user demand for data base access involving
more than one file.
2. A second serious problem with existing file allocation results
is the complete neglect of synchronisation costs in updating
redundantly stored data.
3. The assumption that complete files should be the unit of
assignment of data to nodes seems inflexible. There are many
situations in which permitting a partitioning of files will
reduce data accessing and storage costs.

Accessing method
A transaction is defined to be a logical unit of work typically
equivalent to a user activity which is manifest as the execution
of an application program. The execution of the application
program is typically initiated by an input message from a
terminal operator. After the input message is processed and the
data base is accessed, and perhaps amended, results are produced
which are transmitted back to the terminal operator (output
message). The input message and the associated output mes-
sage are referred to as a message pair. There may be several
message pairs for a single transaction occurrence.

Transactions in a distributed data base system, the processing
of which requires access to data which is remote from the point
where the transaction has been initiated, can be handled by one
of three methods.
1. Transaction switching (Fig. 12) where the input messages
for a transaction generated by a terminal attached to one
computing facility (local) wishes to initiate the execution of
an application program on a remote computing facility.
The messages are transmitted to the remote computing
facility and cause the execution of the application program
and perhaps amendment of the data base section resident
there. Results are transmitted back to the local computing
facility which passes those results to the terminal that
originated the transaction. Transaction switching can be
summarised as moving the transaction to the location of the
data.

2. Split processing (Fig. 13) where the processing of the
transaction is split into a number of components. Each
component consists of the processing of an application
program and the accessing and, perhaps, amendment of a
data base section wholly within the confines of a single
computing facility. When one component finishes, it passes
intermediate results to and activates the next component in
a remote computing facility. When all components have
completed, the final results are transmitted back to the
computing facility where execution of the first component
took place and those results are passed back to the terminal
(local) that originated the transaction. Split processing
involves the execution of application programs on the same
computing facility as the data base section that is being
accessed and can be summarised as having the processing
local to the data.

3. Remote access (Fig. 14) where the processing of the

(a) logical structure definition (names of records, names of relationships, their domains, etc.)
(b) physical structure definition (data field formats, inverted fields, etc.)
(c) file statistics (size, etc.)
(d) accounting data (who has accessed the file, who owns the file, etc.).

For a distributed data base an additional category of directory information must be added: the location of each piece of the data base in the network.

The system software (the data base management system and the network component) must have access to this information to analyze user requests, choose and execute an access method and account for the resources to be used. Therefore a decision has to be made concerning the structure of the directory and where the directory is located.

Chu (1976) has investigated the performance of four directory structures. In each of the four structures it is assumed that each node has its own local directory which contains information about the data stored at that node. The structures are:

1. **Centralised**
   A master directory is located at one of the nodes. When a user requires data that is not stored at his local directory the master directory is consulted. The centralised directory must be updated when there is a change in the storage location or contents (addition, deletion) of a data base section. There are communication costs incurred for each transaction that requires remote data.

2. **Extended and centralised**
   This is a modification of the centralised directory. Once the user finds the location or description of data, that information can be appended to his local directory. Should the user require that information again, the directory information can be obtained from the local directory, thereby reducing the communication cost as well as time for querying the master directory. However, when the information of that data at the master directory is updated, updating of the information of that data in the local directories is also required.

3. **Local**
   In the local directory case, there is no master directory in the system. When information about requested data is not stored in the user’s local directory, the user queries all the other local directories in the system until the requested data has been located. Such a system requires high communication cost.

4. **Distributed**
   In the distributed directory case, each node in the system has a master directory. The advantage of the system is its fast response time. The disadvantages of this system are the cost of storing master file directories at each node and the communication cost for updating all these directories.

The general conclusions of Chu’s investigations, assuming that transmission cost was much higher than storage cost, were that at low directory modification rates distributed file directory yields the lowest operating costs. As the modification rate increases extended centralised directory provides the cheapest solution, while at high modification rates the centralised solution is the cheapest. Of the order of a third of all transactions were assumed to require remote data.

**Currency**
While speed of response is important, another prime question is how up-to-date the reference data needs to be. For the user,
few systems really need reference data to be up-to-date to the last transaction. The tighter the specification, the more current need be the updating process and usually the more costly the design. For this reason designers often choose to update files overnight in batch as it simplifies many technical aspects and can meet system objectives. Currency will be a factor of particular relevance to a distributed system when more than one copy of data exists, i.e. the conceptual data base is replicated. When there is more than one copy of data the currency requirement can be one of the following:

1. Delayed, which implies that amendments are made to only one copy of the data and the other copies are amended at some later time. This will tend to reduce data communications traffic at the cost of data consistency.

2. Immediate, which implies that all copies are kept in step. This will require accesses to be blocked to all copies, where the content of the data concerned is to be changed. All copies are then amended and only after all amendments are completed are any of the copies available for access.

**Integrity of distributed data base systems**

Some important design decisions that must be made by the designer concern the means by which the integrity of a distributed data base system is preserved. Preserving the integrity of the data in a distributed data base system poses complexities of an order of magnitude greater than preserving integrity in a centralised data base system due to the multitude of interactions possible when programs are capable of updating concurrently several local data base sections attached to computing facilities different from those in which the updating programs are running. These programs must be prevented from interfering with one another. Such prevention, or concurrent access control, usually operates by means of locking data base elements (record, page, item) when there is a possibility of interference. Locking may delay other programs that require access to the same elements but prevent these programs from damaging one another through conflicting use of the data base. The most complex case of delay caused by locking is deadlock, in which a program has one element locked and wishes to lock a second while another procedure has the second element locked and needs the element locked by the first program. Each program may wait for the other to release its lock and this mutual interlock can continue indefinitely unless resolved by the system software. Deadlock is a serious problem in a centralised data base but a potentially much more serious one in a distributed data base. For example, deadlock can occur between two programs each in different processing facilities and each accessing data from two other data bases. The system software ideally should recognise that a deadlock condition has occurred which implies that it must continually monitor locks applied on data in all local data bases. An examination of the deadlock problem has been performed by Chu and Ohlmacher (1974). Recovery is necessary in any system. The effect of data base updates that were only partially complete at the time of failure must be reversed and the failed programs restarted to execute again if appropriate. This condition becomes even more complex in a distributed data base system. If a program of a computing facility with updates outstanding at multiple location fails, co-operating actions must be initiated and synchronised. The problem becomes greater with a distributed data base system which permits linkages between records in several different local data bases. A fuller analysis of the problem of integrity in a distributed data base system is given in Davenport (1978b).

**Data distribution—currency combinations**

Four combinations of data distribution and currency can be identified for a distributed data base system. The selection of a particular combination may affect the choices available for control, directory and access method. The combinations are as follows.

**Delayed update—partitioned data base**

Delayed update to a partitioned data base necessitates a read-only mode of operation of the complete conceptual data base when operating in transaction processing node. Transactions, generated by terminals, are processed by application programs which require a read-only mode of access to the data base. The access method for these read-only transactions can be transaction switching, split processing or remote access. Transactions which when processed will cause amendments to be made to the data base are simply validated and added to a transaction file. This necessitates the access method for transactions, which will cause amendments to the data base, to be transaction switching. The transaction file in each node is input to an application program which is run in a batch processing mode at some convenient time (e.g. overnight) when there is no transaction processing activity. It is the processing of this application program which actually causes amendments to be made to the data base.

**Delayed update—replicated data base**

Delayed update to a replicated data base requires that one node is designated as containing the master copy of the section of the data base that has been replicated. Any amendments are made only to the master copy. Other nodes operate in a read-only mode, i.e. transactions, the processing of which results in changes to the data base, are processed at the node that contains the master copy. The access method used for accessing sections of the data base held on remote nodes can be either transaction switching or split processing. Normally such accessing will only be necessary for transactions, the processing of which will cause amendments to the data base. If split processing is the access method, only one component of the application program that is processing the transaction (the one resident in the master node) will cause amendments to be made to the data base. The read-only copies of the data base section involved are updated in a batch mode at some convenient time (e.g. overnight). One method (Murray, 1977) of handling the reconciliation of the copies of a replicated data base is for each physical record to have an indicator field to denote an insert/amend/delete action. The indicator is set to one of five values:

A—this record has been inserted
B—this record has been amended
C—this record is to be deleted
D—this record is to be deleted and the node containing the second copy has been informed
Z—this record has not been changed.

The processing of transactions which cause the master copy to be amended will also cause the setting of the indicator field to be altered. A utility program is run at the master site at some suitable time (e.g. overnight) to transfer all changed records to the remote slave node where another utility program is run which updates the read-only copy using the indicators. An unload utility program is run on the master node to change C indicators on the master copy to D and a utility program running on the remote slave node changes the appropriate records on the read-only copy to D. Further utility programs are run on both sites to delete D records.

**Immediate update—partitioned data base**

If immediate update of a partitioned data base is required, any one of the three access methods may be employed.
1. **Transaction switching**
If the access method employed is transaction switching then the actual processing of the transaction, which involves the execution of an application program which accesses and amends the data base, is confined to a single node.

2. **Split processing**
If the access method is split processing this implies that to process a transaction, accesses and amendments may need to be made to data base sections attached to more than one node. In order to maintain content consistency of the data base the processing of the transaction may require sole accessing rights to data in more than one node. Accessing of data may be blocked for a significant period of time because of the data communications necessary between the nodes involved in the processing of the transaction. If content consistency between data items held in different nodes is not necessary, then the amount of time that the data is blocked will be reduced appreciably since the data communication time element will be removed from the length of time that data is blocked.

3. **Remote access**
If this access method is chosen, it is certain that the processing of a transaction which requires sole access to data on a remote node will cause that data to be blocked to further access for an appreciable period of time. This is because of the messages that must be transmitted between the nodes to request the data to read and lock the data, to modify the data and to release the data. As Rothnie and Goodman (1977) pointed out the delay associated with locking in a distributed system can be some 2 to 3 orders of magnitude greater than the delay typically encountered when setting locks in a centralised system. In the case of split processing the length of time that an item of data is blocked may only be for the duration of processing in one node if there is no requirement to maintain consistency between items of data in different nodes.

**Immediate update—replicated data**
Immediate update in a replicated data base is likely to be a viable solution only if the majority of transaction processing, which requires data held on remote nodes, only need to read the data. A very small proportion of processing of transactions will require amendments to be made to remote data. If immediate update is required of all copies of data in a data base which contains some degree of replication, two access methods are possible: split processing and remote access.

1. **Split processing**
If split processing is employed, this implies that identical application programs are run when amendments are to be made against each copy. In order to keep the copies consistent the application programs are initiated at the same time. The node where the transaction has been received will have to determine the data that is to be amended and request that sole access is given to that data for all copies. Once sole access is granted the application programs that cause amendments to the data base are initiated at the nodes containing copies of the data to be amended.

2. **Remote access**
With split processing there is no saving in time since in order to keep the data base consistent, locking messages must still be sent to all nodes containing copies of the data to be amended. Therefore remote access which does not require the synchronising of application programs but only the synchronising of data amendments is a more straightforward approach. A possible sequence of actions is the following:
   1. Inhibit accesses at all copies
   2. Lock copies as soon as current accesses finish
   3. Amend all copies
   4. Unlock all copies.

**Effect of design decisions**
A number of possibilities exist concerning the structure of a distributed data base system as the previous sections have shown. In order to select the most suitable structure it is important that as many structures as possible are examined with respect to the meeting of the system objectives. Performance and economy are the objectives that are the easiest to quantify while the remaining objectives will most likely have to be assessed subjectively. As far as performance is concerned it should be possible to construct a queuing theoretic model for each structure as demonstrated by Chang (1978). The variations in performance produced by different data location accessing methods have been demonstrated most forcefully by a simple example of Rothnie and Goodman (1977). Relational data-bases (Codd, 1971) are assumed to be employed in the system. The relations have the structure, size and location shown in Table 3. These relations provide information about supply relationships among certain suppliers and the projects they supply. Projects contain project identifiers (J-No) and the location (J-Location) of each project. Suppliers indicates the identifiers of suppliers (S-No) and their locations (S-Location). Parts tells the name (P-Name) and length (P-Length) of each part. Finally Supply ties these objects together by indicating which suppliers (S-No) supply which parts (P-No) to which projects (J-No).

A communications facility with a transmission delay of 1 sec and a bandwidth of 10,000 bps is assumed. The relatively long transmission delay has a substantial impact on the choice of a strategy for processing: it causes the transfer of data between processes on different machines to be much more efficient if it is accomplished as a continuous stream rather than as a series of separate interactions. Now, consider a query which selects the identifiers of projects in Boston which use 10 inch bolts. A DSL-ALPHA expression for this query is:

Range J Projects
Range Y Supply Some
Range P Parts Some
Get W. J-No where J.J-Location = "Boston" and
J.J-No = Y.J and P-No = P.P and
P.P-Name = "Bolt" and P.P-Length = 10

In order to calculate the costs of various strategies for processing this query an estimate is needed of the sizes of three intermediate results:
1. The number of projects located in Boston Count (J where J-J-Location = "Boston")
   Estimate = 1,000

2. The number of supply tuples for projects located in Boston Count (Y where Y-J-No = J-J-No and J-J-Location = "Boston")
   Estimate = 100,000

3. The number of parts which are 10 inch bolts Count (P where P.P-name = "Bolt" and P.P.Length = 10)
   Estimate = 10

With this preliminary information, Rothnie and Goodman consider (briefly) six alternative strategies for handling this query, and then calculate the communications delay incurred by each strategy. Delay in seconds is computed by the formula:
   \[ \text{Delay} = \frac{\text{Number of Interactions} \times 1 \text{ sec}}{\text{Volume}/10,000 \text{ bps}} \]

Strategies 1 and 2 involve an initial step of rendering the query local by moving all relations involved to a single site. When this step is completed the query is processed by some nondistributed algorithm. Strategy 1 involves moving the Parts relation to site A. This entails one interaction and 10^7 bits of total volume for a delay of 10^3 seconds or 16.6 minutes. Strategy 2 involves moving Supply and Projects to site C at a cost of two messages and about 10^8 bits of data transfer for a delay of more than 2.7 hours.

Strategies 3 and 4 mimic a strategy which is sometimes used in single site relational query processing called tuple substitution. In this scheme a multivariable query is processed as a sequence of 1-variable queries by repeatedly substituting tuples for certain variables into the query. Strategy 3 will substitute tuples from Supply and Project to generate a sequence of queries to process against the Parts relation. For example, if the Supply tuple (7036, 152475, 1567) and the Project tuple (1567, Boston) were substituted into the original query the resulting 1-variable query to process at site C would be:
   (P where P.P-No = 152475 and P.P-name = "Bolt" and P.P-Length = 10)

In essence the process has found a project in Boston which uses part 152475 and is asking if this part is a bolt. Using the estimated sizes provided above there will be 100,000 questions of this form to ask. Each of these queries to site C will require a separate interaction in each direction. This will generate a delay of 55.5 hours. Strategy 4 involves substitution of Parts tuples into the original query to produce two variable queries involving the relations Supply and Projects. Such queries can be processed entirely at site A. 10 such queries will be generated for a delay of about 20 seconds.

Strategies 5 and 6 involve the initial computation of subsets of the relations at sites A and C and then moving the subset at one of these sites to the other one to complete the query processing. Strategy 5 computes at site A all of the P-No and J-No pairs for projects located in Boston. This relation is then moved to site C to determine which of the P-No's correspond to 10 inch bolts. At site A this involves processing the query:
   Get W (J-J-No, Y-P-No) where
   (J-J-No = Y.J-No and J-J-Location = "Boston")

### Table 4

<table>
<thead>
<tr>
<th>Strategy</th>
<th>Description</th>
<th>Messages</th>
<th>Volume Delay (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Move Parts to A</td>
<td>1</td>
<td>10^7</td>
</tr>
<tr>
<td>2</td>
<td>Move Supply and Projects to C</td>
<td>1</td>
<td>10^4</td>
</tr>
<tr>
<td>3</td>
<td>Substitute tuples from Supply and Projects</td>
<td>2 \times 10^4</td>
<td>10^7, 2 \times 10^5</td>
</tr>
<tr>
<td>4</td>
<td>Substitute tuples from Parts</td>
<td>20</td>
<td>10^3</td>
</tr>
<tr>
<td>5</td>
<td>Move restricted join of Supply and Projects to C</td>
<td>1</td>
<td>10^3</td>
</tr>
<tr>
<td>6</td>
<td>Move restriction of Parts to A</td>
<td>1</td>
<td>10^3</td>
</tr>
</tbody>
</table>

W will contain about 100,000 tuples of (say) 100 bits each. This will be transmitted to C in one interaction with a delay of about 10^3 sec or 16.6 minutes. Strategy 6 involves the computation at site C of the P-No's for parts which are 10 inch bolts and transmitting P-No's to site A. Estimates indicate that there will be 10 such values and hence, that they will be transmitted to A in less than a second.

The delay incurred by each of these strategies is summarised in Table 4.

The main findings of the example were:

1. There is a very great variation in communication cost among a set of plausible distributed query processing schemes.

2. The absolute magnitudes of the communication delays are very great for the poor strategies and seem likely to dominate the total query processing delay in these cases.

3. The two dimensions of communication delay, end-to-end delay and bandwidth, are both important in choosing a distributed query processing strategy. For the parameters used in this example it is better to employ a strategy which transmits a batch of data at once (as in strategies 5 and 6) than small amounts of data in each of many separate messages (as in strategies 3 and 4).

4. While the communication delays did not reflect this, the best strategy (Strategy 6) provides opportunities for parallel processing which can reduce the total elapsed time for handling this query. Specifically, at site A, it is possible to compute the (J-No, P-No) pairs for Boston projects while at site C computing the set of P-No's for 10 inch bolts. Then when the P-No's are moved to site A the solution to the query can be obtained by taking the join of these intermediate results. For some queries this effect will be significant and can result in elapsed processing times for distributed queries which are actually less than for single site queries.

### Conclusion

This paper has attempted to outline the alternative structures available to the designer of a distributed data base system and has examined briefly the viability of these structures. In addition some of the problems which are unique to distributed systems and which must be solved by the designer are examined.

### References


© Heyden & Son Ltd, 1981


---

**Book reviews**

**Computer Organization: Hardware/Software**, by G. W. Gosline, 1980; 309 pages. (Prentice-Hall, £12.95)

The text has been designed as an introduction to digital computer organisation and computer structures without concerning itself with the associated topics of designing and building computers and is aimed at readers whose primary interest is software rather than hardware. The author considers the text to have two overall objectives: to allow students to acquire the organisational concepts of various extant computers, and to acquire an understanding of the effects that these organisations can have on the overall computational system design, effectiveness, efficiency and economy.

The first chapter introduces some basic concepts including data types, addressing structures, registers and, interestingly, the PMS notation first developed by Bell and Newell as an attempt to formalise a descriptive notation of computer systems. In the author's opinion, an opinion with which I concur, PMS notation is accurate and terse enough to be of importance and interest. Unfortunately this notation has evinced scant interest since its postulation in 1971. This chapter is then followed by an important chapter on instruction variations and instruction repertoires, a sound knowledge of which is a prerequisite for a thorough understanding of computer organisation.

After a chapter in which control units are discussed and some basic microprogramming concepts introduced, the characteristics of random-access, serial-access, and direct-access memories are considered. This is followed by a chapter on interrupts and a consideration of various data path topologies.

Chapter 6 looks at the ways that may be used to increase CPU processing power by providing parallelism at the instruction level and below and includes pipelining, instruction look-ahead and memory interleaving. In this context machines including the CDC STAR and 7600, CRAY1 and ILLIAC IV are discussed. The final chapter discusses computer networks and looks at various network topologies classified under multihop-line, loop, and hierarchical.

Since complete books and indeed series of volumes have been written on the material contained in the individual chapters of this book it has not been possible to cover exhaustively the material thus presented. Indeed the author has no pretensions of doing so. What he has done is to produce a well balanced introductory text for students who, ideally, will already have programming experience in both procedure level and assembly level languages and some knowledge of data structures and operating systems. It must also be reported that the author continues a welcome trend in books of this ilk; the inclusion at the end of each chapter of many stimulating and worthwhile problems, theoretical investigations and practical projects.

R. Lovett (Addlestone)


This book is the second edition of a book first published in 1971. It is aimed at an American audience of students 'enrolled in a conventional freshman-sophomore linear algebra course', but is well suited to the first year of a UK university course. In writing the book a special effort has been made to interweave the concrete with the abstract, and to present topics in a manner that will gently impose the necessary abstractions upon the student. Introductory chapters on matrices and vectors, and systems of linear equations are followed by a chapter (labelled 'optional') on linear programming. There follow three short chapters on the inverse matrix, determinants, and eigenvalues and eigenvectors of a matrix. Substantial chapters on vector spaces and linear transformations complete the book. Each chapter contains numerous and diverse examples, and also sections on 'applications', which reflect the fact that the book is also intended for non-specialist students of mathematics. The tailpiece of each chapter reviews the 'new vocabulary' of that chapter and is rather irksome. That apart the book is well organised and produced and deserves the success which it has enjoyed over the past decade.

N. Riley (Norwich)

© Heyden & Son Ltd, 1981