Initial Report

of

The Distributed Database Working Group

of

The British Computer Society

edited by: D. A. Hickman

technical editors: G. Gibbons
J. Shepherd

graphic design: R. E. Deeley

Initial Report of The Distributed Database Working Group of The British Computer Society

Edited by: D. A. Hickman

Technical editors: G. Gibbons
J. Shepherd

Graphic design: R. E. Deeley

Word processing systems by Sigma Data Systems Ltd., Southampton

Published by The Distributed Database Working Group of The British Computer Society

Printed in England by Ashford Press

Autumn 1983

The contributing membership of the Distributed Database Working Group comprises:

Michael Agnew, LDR Systems;

Maristella Agosti, Universita di Padova, Italy; *

Fatima Ahamed, Thames Polytechnic;

Richard Barker, C.A.C.I. Inc.-International;

Mike Bibby, Standard Telephones and Cables;

Steve Bowran, CAP Scientific;

Christian Esculier, Independent Consultant, Paris, France;

Gerry Gibbons, ICL;

Paul Gretton-Watson, SCICON;

Dave Hickman, Independent Consultant, Southampton;

Mike Imber, SCICON;

Roger Johnson, Birkbeck College, London University;

Jim Pimpernell, Logica VTS;

Michael Shave, Liverpool University;

Jeremy Shepherd, SCICON;

Nahed Stokes, Queen Elizabeth College, London University;

and John Webb, ICL.

The views expressed in this report are those of the members of the DDBWG, and are not necessarily the official views of either the British Computer Society or the members' employers/parent organisations.

* Maristella Agosti is partially supported by the Italian National Research Council, Progetto Finalizatto Informatica, under contract no. 81.01548.97.

Acknowledgements

The Group wishes to express its gratitude to the following, who contributed earlier work upon which we have drawn for this report and/or provided commentary upon draft versions:

Judy Chapman, The Open University;

Misbah Deen, University of Aberdeen;

Martin Hammer, Independent Consultant;

Janet Jefferys, John Hoskyns & Co. Ltd.;

Hugh Robinson, The Open University;

and Ed Tozer, DMW Database (U.K.).

EDITOR'S NOTE

This, the first report of the DDBWG, sets out its early findings in the hope of providing some signposts for those who are contemplating the introduction of this type of technology. In this area, there are many questions still unanswered, and probably as many more still unasked.

It is also the Group's intention to make contact with others who are active in this relatively new area of data processing. To this end, interested persons are invited to contact the Group through its secretary, by writing to:

The Secretary,
Distributed Database Working Group,
c/o The British Computer Society,
13, Mansfield Street,
London,
W1M OBP.

CONTENTS

FOREWORD

This report provides guidance to those managers and designers who are faced with design decisions where the options include storage of master data at several nodes of a network. To this end, it is addressed specifically to data processing practitioners, rather than following the more customary pattern for reports of working parties and groups and looking for its readership towards academics and the members of standards bodies.

Strictly, the term 'distributed database' should be reserved for cases where the data is closely co-ordinated between nodes, probably using system software (i.e. a 'distributed database management system') to do so. In practice, at this early stage of development, it is better to regard the term as referring to a class of technical topics, including those of locating relevant data at specific nodes, and of propagating updates across the network. These topics arise whenever a networked solution is proposed to support major applications requirements. Increasingly, such networked solutions are being proposed.

When the Distributed Database Working Group (DDBWG) began its work early in 1980, it found that almost all distributed database work was confined to research laboratories, and consisted of limited research papers and experiments. Since then, the upsurge of interest in distributed systems of all kinds has caused an increase in distributed database activities, to a point where early software products are becoming available from suppliers. These are extensions to existing software products, and their more advanced features are exploratory in the absence of widespread user experience.

The growth of distributed database systems during the 1980s will be strongly influenced by changes in the costs of data communications, compared to those of other forms of communication. Currently, corporations are heavily dependent upon the use of public mail, telephone and telex systems to co-ordinate between departments, business locations and international operations. As far as data processing is concerned, the best understood type of solution is that of the central host installation, supporting central databases and a network of local and remote terminals.

Where reliable data links of sufficiently large capacity and low cost are available, and there are no objections on other grounds, such an installation can thus provide its services through the network to all parts of an enterprise. In current practice, link capacities are likely to be limited, communication costs high, and other constraints (such as use of existing local computers, local autonomy and resilience) important. Hence, the handling of distributed data is a very real issue in the network systems currently being considered or designed. Even for fully centralised systems, distributed database techniques are found to be valuable in providing for, and controlling, co-operation between distinct but related databases within a single installation. The Group intends that this report will enable its readers to avoid the pitfalls, and to achieve the benefits, of the distributed database approach.

John McLeod Webb

CHAPTER ONE

INTRODUCTION

The Background to Centralised Databases

In the mid-1960s, the first database management systems (DBMS) became available for implementation on large mainframe computers. It was anticipated that their introduction would mark the end of the chaos wrought by piecemeal application development and the beginning of a new era of controlled and centralised development. The database approach was seen at that stage mainly as a vehicle for integrating and rationalising the information needs of the divers users within large organisations. The new approach was not intended for the first time computer user; rather, it was for users familiar with the problems and pitfalls of traditional methods of building computer applications programs.

The high proportion of data duplication was the root of some of these problems. Data duplication meant that a change to the definition of a data item was likely to affect many programs and different records in a variety of files. This explains the vast amount of time and effort spent on system and program maintenance (which has been estimated at as high as 46 per cent of programmers' time). Data duplication also meant that an update would have to be applied to all copies of the data, thus increasing processing costs. As these updates were rarely synchronised, a high level of inconsistency existed at most installations, and much processing was required to keep it to an acceptable level. High development cost, long lead times, and a lack of both robustness and flexibility were also common system problems. Lack of flexibility was especially apparent when it came to dealing with exceptional situations, ad hoc requests and simple enhancements.

In the context of all these problems, 'databases' were seen as a panacea. Databases would provide multiple access paths to the same data, thereby satisfying many different requirements and eliminating the duplication of both data and processing common to most implementations. Databases would enforce standards, and, within an organisation, provide centralised control over the definition and use of data. Safeguards would be available to prevent unauthorised or accidental access to data, and additional facilities would allow easier recovery from most types of failure.

Moreover, the database management system, acting as a buffer between applications programs and data, would insulate programs from changes to the data, thereby reducing maintenance effort. Generally, databases would offer better and more flexible tools for applications development, allowing a faster response to ever-changing user requirements.

On the basis of this argument, many installations decided on the use of a DBMS: as their number increased, so did the number of DBMS products on the market. Meanwhile, the database approach, techniques and implementations were the focus of a substantial research effort, as demonstrated by the proliferation of publications, conferences and seminars on the subject.

Conclusions Upon Centralised Database Usage

The experience of early - and subsequent - users of DBMS has been reviewed and summarised in various reports, both in the United Kingdom and abroad. The findings in these reports are not always consistent, although there is substantial agreement among DBMS users on the most important points.

One clear fact is that not all DBMS users are committed to a database approach. Many have used a DBMS package as a sophisticated file handler in the development of one or more applications, whilst not attempting to integrate these applications. Among users committed to a database approach, the original drive towards integration has been tempered by a new-found realism, based on the fact that integration is not always either practicable or feasible within the time-scale and resources available. Political and organisational factors can also prevent full integration, as can others of a technical nature.

Political difficulties arose when the process of integrating computer systems across departmental boundaries appeared to threaten the independence of the individual departments concerned. It was not surprising, in such circumstances, that some departments resisted the whole approach or gave it less than wholehearted support.

Even when inter-departmental politics were not an issue, the development of database applications was often considerably delayed by the excessive number of people involved in the process. This increased the time needed for activities such as arranging meetings and vetting and securing agreement on work done. In other cases, the length of time and the human resources which would have to be committed to identifying a company-wide view of information requirements has, in fact, precluded the completion of such a study - especially since these requirements are rarely static over a long period of time.

Other factors which have worked against the comprehensive integration of computer systems are the extent of investment in existing applications software and the use of applications packages which are not DBMS-based, and therefore cannot easily be integrated into the overall approach. At a technical level, the design of large integrated databases is normally a complex task requiring specialist skills. Many options are available to the designer, and trade-offs need to be carefully assessed.

Few designs have been totally free from data duplication. In almost every case, a certain amount of data redundancy has been introduced for reasons of performance, integrity, and ease of conversion or programming. Implementation is further complicated by the loading and re-organisation problems associated with large, complex databases

- even when specialist software is available to perform these tasks.

In the light of past experience, it seems impractical to develop computer systems which are more integrated than normal everyday business functions warrant. For the same reason, it may be impractical and costly to make all these tasks interact with each other, regardless of the potential benefits to the business. The solution to business problems which result from poor departmental structure and poor communication between the departments lies in an implied reorganisation of the business. Attempting to solve such problems at only the computer level, by building integrated computer systems, is likely, in the long run, to prove both a costly and an ineffective exercise.

Where, however, business problems are contained within departmental boundaries and the interfaces between departments are well-established and fully functional, separate databases could be developed to support the various functions. Database design is similar to many other tasks in that it can be managed more easily when divided into smaller tasks, which can be carried out independently within overall guidelines.

Although databases have not in all cases been entirely successful as vehicles for integrated computer systems, they have met success in many other ways.

As the experience of many thousands of database users indicates, they have succeeded in the context of a single function or group of related functions. By providing file handling capabilities over and above those supplied by most operating systems, they have fulfilled a real need for the cross-relation of information which is stored in different files, and in many cases they have allowed complex applications to be built more easily.

Databases have also simplified development and maintenance by providing a high - if varied - level of data/program independence. Most existing DBMSs support a definition of the data and of independent user views - those required by particular applications programs or particular queries.

Another consequence of database technology is the development of new analysis methodologies, which provide a better framework for communication between computer staff and non-computer staff. With the advent of program generators and complete system builders, it is likely that these analysis techniques will play an increasing role in future applications development, whether or not it is DBMS-based.

Looking to the future, some very definite trends are becoming evident, the most obvious being a commercial tendency towards decentralisation.

Another major development is cheaper hardware. The purchase of a computer no longer requires high level decisions within the company, simply on financial grounds; neither does it necessarily involve a central data processing department. The availability of cheap hardware, which is compact in size and capable of functioning in a normal office environment, is making it possible for an increasing number of departments to purchase and install their own computers.

A third pronounced movement is towards the use of applications packages. The availability of a specific software package is often the main reason for the purchase of a particular mini-computer or micro-computer. Packages and, to a lesser extent, program generators are lessening the dependence of first time users on central data processing departments for the supply of applications software.

So where does this leave databases? How is it possible to reconcile the principles of database - which stands for integration and centralisation - with the current trend on the part of departments towards going it alone?

The Role of Distributed Databases

This is where the distributed database comes in. Recent developments in communications are providing the environment for automatic exchange of information held on separate computers. With present technology, however, this exchange is only possible in a limited form and between specific types of hardware and software. A genuine exchangeability of information across a broad range of hardware and software can only be accomplished if comprehensive standards for that purpose are defined in both the communications and database fields. Work on distributed databases is directed towards the latter.

For the users of a centralised database, a distributed database will provide the means of decentralising their computer operations in an orderly fashion, thus increasing computer availability, decreasing dependence on a central computer, and allowing the processing of the data to be done at source, where it can be monitored and controlled in the most effective way.

For those installations using a variety of equipment to run a spectrum of tasks, distributed database could provide the means to exchange, extract and consolidate information scattered across various computers without duplicating the information and processing, and with a minimum impact on existing systems.

Distributed databases may provide newcomers to the field of computers with the option of starting small, knowing that future requirements can easily be accommodated - on separate computers if necessary - without conversion or interface problems.

Basically, a distributed database system will be able to process a properly formulated request for information by a user at one location for information held at a different location. As the volume of information and the number of locations increase, it becomes impracticable for every individual to keep track of who has what information at any point in time in order to formulate such a request.

Consequently, a second function of distributed database will be to provide some form of 'transparency' - a mechanism through which a request for (possibly) multi-site information is analysed by the distributed database management system. As a result of this analysis, requests might be issued to several sites. Individual replies will then be collated and the result returned to the location from which the original request issued.

The third task of distributed database will be to cope with the updating of information held at more than one location, and with the dependent problems, such as the consistency and integrity of the data. Related information at several locations must be kept in step. In some cases, this may involve maintaining exact images. In other cases, the information will need to be brought up to date at periodic intervals only. Obviously, the shorter the period of inconsistency allowed the more complex the task of synchronising the updates - especially when problems of hardware, software, and communications are taken into account, and when concurrent access and processing are allowed.

CHAPTER TWO

CONTEXT

Technical Innovation

Distributed databases had begun to be used for practical applications by the beginning of the eighties. To understand why they should have come into being at this time (and how they have evolved), it is necessary to review the concepts of data distribution, database and distributed database, and their feasibility over the past two or three decades in the context of technological development. The key is in the data - how it is possible to store, process and transport it. These three criteria will clarify the major evolutionary steps along the path from the pre-computer age to the present time.

Before the computer, the medium of data storage was, essentially, paper. The paper varied in colour, quality, stiffness, formality, permanence. Nonetheless, from the scribbled note to the identity card, it was all paper. Such processing of data as was practicable was done by people. Existing documents (in the broadest sense) were annotated, or their content merged, wholly or in part, to produce a new document, possibly with the assistance of a calculating machine and/or typewriter. Data transportation was achieved by the physical transmission of documents, or the telephone, telegraph and telex services.

In this situation, technology could offer little to information services, which were almost entirely people-dependent. Data required within an organisation was collected and held in the particular department which had responsibility for its processing. Each function of an enterprise held its own - private - data, which it defined, collected, elaborated, processed and recorded to allow its own services to operate correctly. Data for other functions of the enterprise was usually provided as summary results.

With the production of computers on a commercial basis, the first clear advance became evident. Data was stored on sequential media - punched card, paper tape, magnetic tape. The processing of numerical data was left to the machine. Some tranportation of data was done by tape and printout.

This first step into commercial computing was, as one might expect, the start of the revolution. Nonetheless, it was characterised by poor man-machine communication, very poor machine-machine communication and the necessity of the computer being in the continual care of experts. Information systems were, however, transformed. The computer was a very expensive piece of equipment, both to obtain and to maintain. It needed a closely controlled environment, since it generated much more heat than would permit it to function were that heat not removed. It occupied a lot of space. It had many electromechanical components, which of its nature it used to destruction. Its advantages were speed, accuracy and the ability to re-accept for further processing data it had previously stored.

These advantages were quite sufficient for large enterprising concerns to find it worthwhile to establish computer centres. The data was still peculiar to a function, and usually still in the keeping of that function area. The processing of that data was, however, centralised. The computer centre became the hub of information processing.

The third clearly identifiable stage was the adaptation of data storage to direct access methods. Data could now be stored so as to be accessed with little regard for storage sequence. Processing had achieved sophistication and was no longer confined to mathematical processes. The direct transportation of data from machine to machine became possible.

This was the explosive stage of the evolution. Instead of forty-eight words of core and an overlay drum whose changes of speed could actually be heard, 128, 256 - even 512 kilobytes of main store was no longer unusual. Direct access backing-store attained capacities which were measurable in integral numbers of megabytes.

The natural result of this increase in power and capacity was that every function of every computerised organisation mechanised more and more of its workload. The approach was to consign the drudgery to the machine and keep only the interesting work for the people, in the belief that the department would be able to store and process all the data it would need.

Few people will recall when they first knew that there were data items that were common to several departments, several files, several systems within their organisation. This knowledge, however, brought about the next revolution.

Inevitably, several departments within the same organisation would hold the same data items. Generally, items with such common usage were known by different descriptions in different departments. For example, an employee's company reference number could well have been referred to as his works number, clock number, payroll number, personnel reference number, etc., in different departments, and each might have found this number an essential ingredient of one or more of their data files.

Further, this number may have been used as a reference point for recovery of the same data items on various files - in either the same or different departments. It was not illogical, since all the various departments' various files were being held and processed in the computer department, that some rationalisation of such duplicated data should be enforced. This of itself, however, went nowhere towards solving the problem of whose name to use for the data item. Logic again suggested that no friction would be generated if all the names were retained - and then it would become unnecessary for the departments to actually know that the files had been merged. Thus, the database concept - with its essential database management system.

Data storage, on line, could now be held in one vast indexed file, and the capacity of a single backing-store device did not necessarily limit its size. Since the only actual access to the file was by the DBMS, it could span banks of such devices - and the user needed only to know that he was being presented with the data he requested in the form in which he requested it. Data processing was now at a remove from the user, and he was unlikely to know everything which affected what he still thought of as 'his' data. This centralised control of the data led to data transportation by networking, packet-switching and other telecommunications techniques.

The database, once established, brought about a change in the way the organisation was envisaged. It became the data-driven model of company activity. All data was required to become part of the database - and the organisation began to realize that the most valuable asset in its data processing department was not the big, expensive mainframe computer but the data. In general,an organisation cannot operate without its data. Data is precious. So, data must be available for the use of the whole organisation and will no longer be permitted to be a merely departmental asset. As a result, it becomes the company, rather than the department, which is the owner of the data.

The next innovation is in one sense the smallest, in another the biggest: the micro-chip. This is the technological innovation which has brought about the greatest change in thought and approach, because of its effect on the cost of computing. It is not only the cost of manufacture of computers which it has decimated, but also the cost of environmental control and the cost of maintenance. Waste heat is still an enormous problem for the computer industry, but the dimensions of the problem have been so reduced as to be almost disregarded for many purposes.

No longer is it necessary to build an artificially controlled environment to house computer power - so it is no longer necessary for a company to house all its computer power in one place. In these circumstances, the disadvantages of centralisation carry more weight. Cost-effectiveness no longer demands the mighty central mainframe. In consequence, the big machine starts to look like a vulnerable, inelegant dinosaur.

Thirty people will obtain a better response from thirty minis than from one mainframe. Not all of the users need access to all of the data all of the time. If the mainframe is down, nobody has a computer - if one mini is down, the other twenty-nine users may not even need to find out.

The advantages of the centralised system are not all nullified by this ability to put the power where it is needed. The data being the property of and available to the whole company on a single entry basis is a powerful argument against scattering the power. One particular advantage of the mini- and micro-computers is the existence of user-friendly languages and control systems which enable non-expert users to enter and run a high proportion of their own applications. But what happens when something goes wrong? It is clearly advantageous if the experts still have access.

Fortunately, it is possible to have the best of both worlds. Computers can be networked, allowing data to be passed between them and processing to be carried out at remote sites. They can also be networked in such a way as to have access to data stored in each other's on-line backing store. Clearly, to gain most advantage from this type of access, the machines need common methods of access to the data and a degree of commonality of data structure. In short, to replace the central database on a single, central machine, they need to share a distributed database.

In one sense, this brings the wheel full circle. Private data can be defined and processed locally. Local people can have direct responsibility for their data, their processing - and their computer. The great difference from the earlier situation is that the communication difficulty does not exist, because the local computer is not isolated. The local computer is now linked to those of all other departments. Common data is communally available. Data which is of purely local use can be stored locally and accessed only locally. Data which is used mainly locally but occasionally elsewhere can be stored locally and accessed both locally and elsewhere. Data which is in common widespread use can be replicated on as many nodes as is thought desirable - but it still only needs to be entered at one.

It is possible to follow the complete cycle of development: to start with a single mini-computer and follow all the evolutionary stages to the distributed database. It is equally possible to join the progression at any stage or to omit any stage in the process. It would not, until the micro-chip, have been financially reasonable to follow the whole process, but even the biggest fourth generation mainframe is constructed from similar chips to those used in the smallest personal computers. We have no reason to think that the distributed database is the culmination of this computer evolution, but a next step is not yet visible. What is sure is that many will find, in the next few years, that the best available solution to their information system problems is a distributed database system.

Needs

How should an organisation decide if it needs, or would gain significant benefits from, a distributed database system? Depending upon the standpoint within the organisation, the features which appear beneficial or otherwise can be quite different. It is probably best, therefore, to examine the pros and cons from several different viewpoints. Those views represented by the company itself, with its needs for financial and organisational control; the data processing department, which needs full information on applications to give the best advice; the departmental user, who needs to receive and to transmit data both within and outside the department; and the individual user, whose view may be comparatively insular, should illustrate most of the advantages and disadvantages.

Firstly, the company as a commercial organisation, whether retooling from an existing mainframe system or making a major advance into data processing, will be looking for efficient, cost-effective and, preferably, cheap computer systems. Reliability of access and use may also be of great importance, as indeed may data security. A network of mini- or micro-computers is likely to prove cheaper than a mainframe of equivalent power, and further nodes could be added with far less disruption of the existing service.

To the company, the main advantage of a central mainframe machine would be the co-ordination, standardisation and control of data which a centralised computer facility affords. Obviously, this is better exercised with a database system than without one. However, this advantage also accrues from a distributed database system, especially where the company dictates that all nodes shall implement the same local database management system. There is also the fact that only a general power failure will bring the whole of a distributed system down, whereas a local fault may render the whole of a centralised configuration inoperable.

Many companies operate from more than one site, and transfer of data and results to and from a central facility can cause inefficiency - an inefficiency which is minimised by distributing the processing power to the data source and passing on only those results which need to be referenced or acted upon elsewhere. To distribute the computing power but retain effective control, distributed database appears to be the best approach currently available.

What then of the data processing (D.P.) department? Being a service department, it is really looking to the needs of others. Making the different machines in the company network communicate with each other can be difficult. Standardising to a common external view of the data can overcome this difficulty, and this implies the use of a distributed database management system. Standardising the machine-machine interface in this way also minimises friction between departments which need to co-operate, thus making the D.P. department's job easier. A distributed database system also gives the D.P. department the opportunity to assist departments with an excess of data for their own storage capacity to utilise surplus storage capacity at other nodes.

Another advantage of the distributed database system, which cannot be claimed for any other approach, is in the possibility of real-time integrity of the system. Simply, if all nodes are connected by telecommunication links to all other nodes, and their communication devices have the facility to bypass their own node, and each local database is replicated at a distant node, then at least two nodes have to go down before the complete system experiences any deterioration other than in response time.

The departmental user is concerned with the availability of the data and processing power necessary to his systems. Local power appeals to him because it will be available when his department needs it. With a distributed database, the department has no problem of gathering and co-ordinating data required for processing elsewhere, nor of checking that data procured elsewhere has arrived in the department for processing. The diplomatic control of D.P. standards is facilitated by the viewpoint that "our computer must talk to their computers". Also, best use can be made of the departmental node's storage capacity, as rarely accessed data from other nodes need not be replicated at the departmental node.

The individual user may not take account of these broad considerations. He wants fast response from his own terminal. He wants control over his private data - that which is of use only to him and that for whose condition he is responsible. Data which is of use to only a single user will not normally be referenced by other users. Data for which a particular user is responsible becomes available on the distributed database network as the responsible user updates it. If he has an occasional requirement for other data from a distant location, he can access it when he needs to - within privacy limits, he does not have to set up a special prior arrangement with the user on whose machine the data is held. Also, heavy use of facilities by other departments at specific times will not cause him any deterioration in response. The nodal structure gives him greater reliability - other users' problems with hardware (or, indeed, software) will rarely affect him. He may, however, be unhappy at the necessary price - the constraints on data structures and access methods arising from the need for standardisation to permit inter-nodal working.

More general benefits are also likely to accrue. As with each of the previous stages of computer evolution, the distributed database concept makes it possible for larger scale applications to be implemented. One area in which this stage is very likely to promote activity is in the multi-site applications of the companies within a group. With a distributed database, it is possible for company data and group data to share a node, only the group data selectively being accessible to other nodes in the network.

So, which type of organisation needs, or would greatly benefit from, the distributed database approach? Those who want large machine power without large machine cost would do well to consider it. For multi-site operations which have a partial commonality of data, distributed database would appear to be particularly appropriate. Organisations wishing to use a site-by-site approach to computerisation might well find distributed database their most suitable philosophy, allowing, as it does, such an operation to be implemented by a centrally-based systems control team. Those organisations whose potential users would, by sheer weight of numbers, overwhelm the ports of any single machine might find a solution to their difficulties in a distributed database system. So, too, might organisations which wish to retain unit autonomy and still enjoy the benefits of inter-departmental co-operation. In summary, potential beneficiaries are almost everyone for whose needs a single smaller fourth generation machine or micro-computer is inadequate.

Suitability

The availability of a distributed database system offers an organisation the opportunity to do much more with its data than it previously found possible. Some of these possibilities enhance the value of the data: others, however, do not.

The organisation may achieve a database system in which the data is always available where and when it is required and is always correct: a system which is reliable, and secure from outside interference: a system which recovers quickly when it does occasionally fail - and perhaps even hides from the user the fact that a failure has occurred: a system which is easy to use and understand: in fact, an almost ideal database system from the user's point of view.

Alternatively, it may end up with a database system which is slow, and corrupts or loses data: a system which often fails and is easily compromised: a system which takes ages to recover and always fails dramatically: a system which is difficult to use and impossible to understand: in fact, from the user's viewpoint, a total disaster.

So, the first decision to be made is whether to have a distributed database system at all. For some organisations, a distributed database will bring many advantages, giving its users the potential to access much more data in many more ways. For other organisations, the disadvantages might outweigh these advantages. Data access patterns may minimise the advantage gained by having a distributed database, as opposed to a collection of local databases or a central database, and this is outweighed by the extra cost and complexity of the distributed database approach. It is important to choose a distributed database for sound commercial and technical reasons, and not simply because it is the "New Technology". Let us use two contrasting business scenarios to illustrate the advantages and disadvantages of distributed database systems - an air traffic control system and a branch banking system.

Each of the scenarios requires a highly reliable service, but their detailed reliability requirements are quite different. In a branch banking system, the principal reliability requirement is that no data is lost when the system breaks down, so that at some future time the system can be restored to the state immediately prior to the breakdown. The loss of the whole system would, of course, be fairly dramatic, but the majority of the processing and data accessing is carried out on the local nodes, and so the loss of a single node would be more of a nuisance than a disaster. Recovery could be carried out by simply re-processing the most recent transactions until the database was again up to date. In contrast to this, a breakdown in an air traffic control system could cause the loss of the whole system, producing an unacceptable danger to life. Here, only immediate data is useful, and it is not possible to regenerate the database by reprocessing. There must be a back-up copy of the up-to-date database immediately available. .

In a distributed branch banking system, the vast majority of the processing is of local data. This results in a large saving in telecommunication costs compared with a central system, as such costs could be very high for a national system with several thousand branches. However, against this has to be weighed the fact that a distributed database system implies the availability of local computer systems. This could prove very costly for a large network with many local branches. In an air traffic control system, all the data needs to be accessible to all parts of the system at all times, with perhaps a few minor exceptions. These requirements are most economically satisfied by a centralised system - a distributed system could be used, but the cost would be very high to ensure the necessary reliability and speed of access.

Branch banking transactions tend to be fairly self-contained, and operate on a small, well-defined subset of the data. The data structure itself is relatively simple, and the various logical parts of it are well-defined and geographically separable. This type of data is well suited to a distributed solution. In contrast, most air traffic control transactions require access to a large and complex subset of the data. The data itself is totally integral, with a high structural complexity, and is best modelled by a centralised database.

CHAPTER THREE

TYPES OF SOLUTION

General

The complexity of users' requirements in the field of distributed databases varies considerably, the different degrees of complexity being best illustrated by an example. Take as a model a fictional international airline (Ruritanian Airways), which keeps aircraft spares at airports in different parts of the world. Consider the situation when a Ruritanian Airways aircraft is stranded in London, and has an urgent need for a new engine of a certain type (Type A). The airline has implemented a distributed database, which carries details of its spare parts holdings at different airports around the world, and which can be accessed from each of these airport locations (as well as the Airways' provisioning offices). The engineers servicing the aircraft have found that they have no engine of Type A at hand in London, so they need to identify the nearest airport which has such an engine, and determine the best way to schedule transport for it.

They might begin their use of the distributed database by asking a question directed at specific nodes, probably of the order of: "Do the holdings in Paris or Berlin include an engine Type A?". This is an example of the simplest form of DDB query, and is referred to as 'Queries addressed to specific nodes'. In other words, the transaction only involves interrogating the database, not updating it, and nodes which are interrogated are specified by the user.

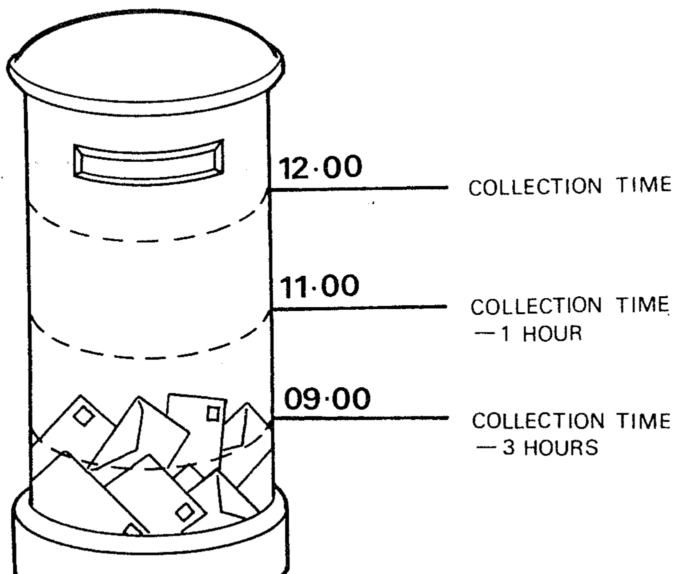
Now suppose that the engineers have found that there is no engine of Type A in Paris or Berlin, and they now wish to locate any Type A engine within Ruritanian Airways' holdings anywhere in the world. The question to be put to the DDB is then: "Does any holding include an engine of Type A?". This type of query is referred to as 'Queries not specifying nodes'. The transaction is still only interrogating the database, but the node is not now specified - i.e., the node is transparent to the user.

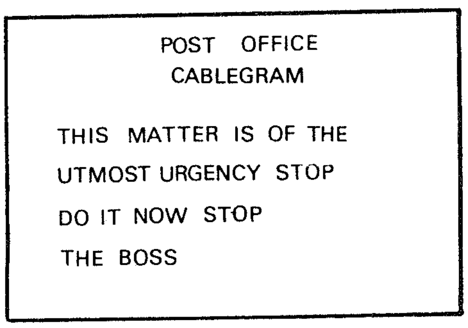
What if even this worldwide search through the DDB does not find a suitable engine? This situation has been anticipated, and a reciprocal arrangement has been made with another airline (Transylvanian Airlines) to use each other's spare parts when urgently required, and for Ruritanian Airways to interrogate Transylvanian Airlines' DDB in such circumstances.

The situation is now complicated by the fact that Transylvanian Airlines and Ruritanian Airways have been supplied by different computer manufacturers, and their DDBs have quite different architectures and protocols, implemented on different types of computer. Nevertheless, they have designed interfaces between the two DDBs which will enable them each to interrogate the other, provided that the necessary authority is obtained. The appropriate enquiry is performed, and an engine of Type A is located in Rome (it might even have been in London). This type of enquiry is referred to as 'Queries across nodes having different regimes'. Like the previous enquiries, it interrogates the database without updating it; but, unlike the earlier enquiries, it has to operate across regimes of different hardware and/or software.

The engine is next flown from Rome to London, and the engineers of Ruritanian Airways need to update the DDB of Transylvanian Airlines in order to account for the removal of the engine from the Rome holdings. This type of transaction is referred to as an 'Update across nodes having different regimes'.

If the engineers had found an engine of Type A within the DDB of Ruritanian Airways, then their own DDB would have been updated, instead of that of Transylvanian Airlines, and in this case we would have been dealing not with an 'Update across nodes having different regimes', but with either a 'Periodic update (homogeneous case)' or an 'On-line update (homogeneous case)', depending upon the urgency with which Ruritanian Airways treat such matters, but, in either event, it would be the case where only one DDB is involved.




POST OFFICE
CABLEGRAM

THIS MATTER IS OF THE
UTMOST URGENCY STOP
DO IT NOW STOP
THE BOSS

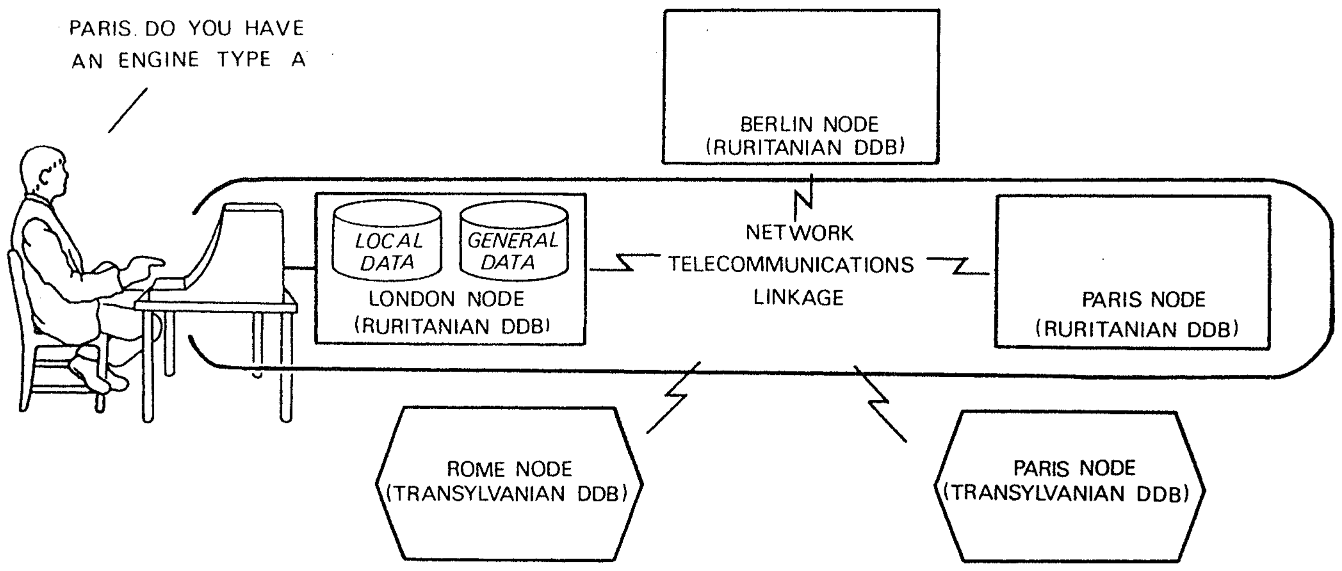
PERIODIC UPDATE
(No action until collection)

ON-LINE UPDATE
(Immediate action)

In the updating situation, it is necessary to differentiate between the 'On-line update' and the 'Periodic update' mentioned in the previous paragraph. Which is implemented in a particular case would be dependent upon the application. In the case of an airline, on-line updating techniques would almost certainly be used, since time is of the essence. In some other applications, however, time delays may not be considered so important. For example, a group of research establishments with a common interest might well use a DDB to enable researchers at any member establishment to locate all the papers and articles on a given topic, regardless of where they might be held. In such a case, the relatively high cost of on-line updating might not be justified and periodic updating techniques would then be preferred.

The six different situations described above each pose different technical problems for the designers of the DDB software. These design problems, and the issues which they raise, are described in the following sections, which treat the situations in order of increasing complexity. At the end of the chapter, a final section, on a 'Highly reliable service', is added. This is intended purely as an introduction to the special considerations and techniques which are involved in those cases where very high levels of reliability are required from a DDB. Typical examples would be military command and control applications, medical applications, or those involving the movements of fissile materials.

(a) Queries Addressed to Specific Nodes


PARIS. DO YOU HAVE AN ENGINE TYPE A
BERLIN NODE (RURITANIAN DDB)
LOCAL DATA
GENERAL DATA
LONDON NODE (RURITANIAN DDB)
NETWORK TELECOMMUNICATIONS LINKAGE
PARIS NODE (RURITANIAN DDB)
ROME NODE (TRANSYLVANIAN DDB)
PARIS NODE (TRANSYLVANIAN DDB)

In this, the simplest example of access to non-local data, the users at a node are usually able to satisfy their needs for data by accessing the data held at that local node. However, they may also require information which is held at a different node, but may only interrogate the distant node, since updating facilities are implemented only on a local basis. To allow a simple solution to these requirements, it is assumed that the access to remote data is not critical in terms of performance, privacy, or reliability. This assumption is reasonable where that remote data might be obtained by some other means, such as a telephone call to the user's counterpart at the other location. For such non-critical needs, a high degree of co-ordination of queries is unnecessary; each node can co-ordinate its own queries. The example is further simplified by the assumptions that the nodes support identical data management regimes; that there are few nodes, and fixed responsibilities for holding data; and that all applications software is already located at those nodes where it will be used. Hence, each user can easily remember what remote data exists, and there are no major obstacles to its being available on demand.

The obvious approach is for a user to identify another node explicitly as part of the query whenever he requires access to remotely held data. Where he directs parts of one compound enquiry to different nodes, he receives separate results: there is no attempt to re-combine these with each other, or with locally derived data.

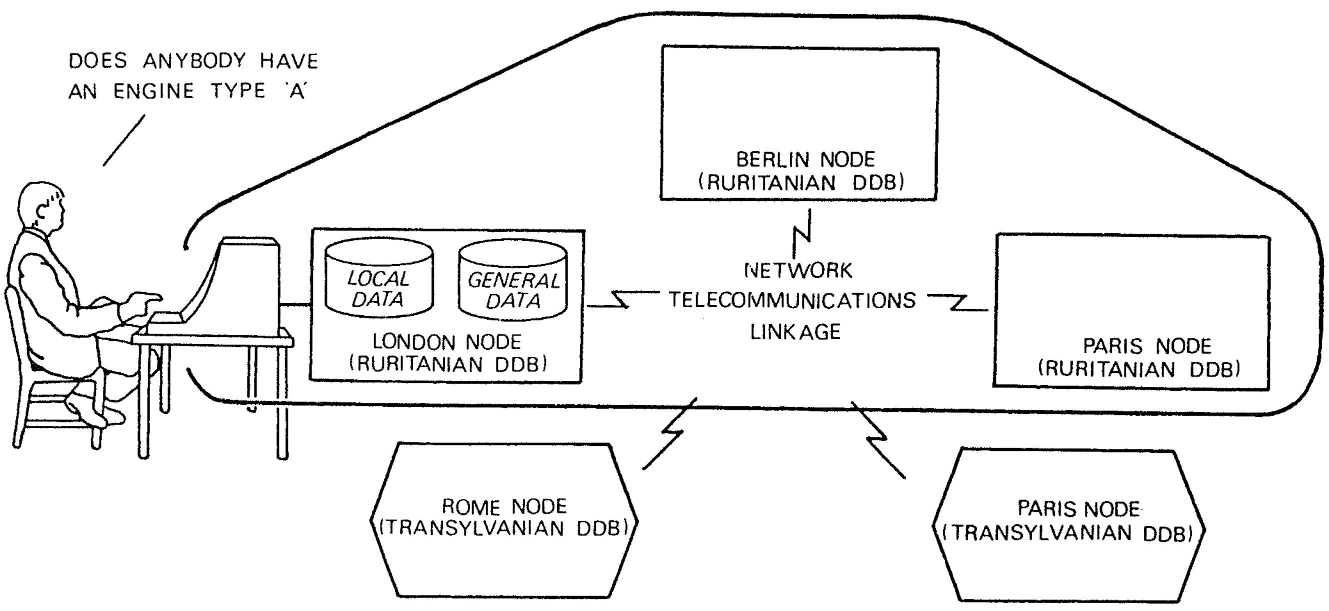
Despite the apparent simplicity of this solution, several important design decisions must be made. The user interfaces need to be aligned across the network, so that a user can express both local and remote queries in the same way. The underlying processes have to translate the logical node identifiers into host machine identities, and notify failure conditions to the applications software for the users' benefit. Queries may contain criteria for selecting relevant data, and such selection could take place either at the remote node concerned or on return of potentially relevant data to the user's node.

The same choice applies to the summarising of received data. Although it is clearly more efficient in terms of data transmission to select and summarise the data at the remote node where it is stored, that approach splits the query processing between nodes, and is likely to be rejected in favour of processing at the user's node in the circumstance where retrieved data volumes are low and communications traffic is not critical.

The same input and output types are likely to be used at several locations, which suggests a need for a data dictionary at each node, particularly where users' error rates are high. This facilitates control of input formats and validation rules and supply of standard output formats, and also caters for local variations in those formats to suit users' preferences. The dictionary contents concerned with data integrity and compatibility would have to be centrally defined, as could defaults for presentation of the data, but users would be able to adjust the appearance of screens and reports as required.

It is, of course, feasible to remove any of the simplifying assumptions, such as the presentation of separate results from each node. However, any increased capability has to be supported by applications or systems software, and the assumptions specified above demonstrate how many of the problems of distributed data handling can be avoided by choosing a limited set of system requirements.

(b) Queries Not Specifying Nodes

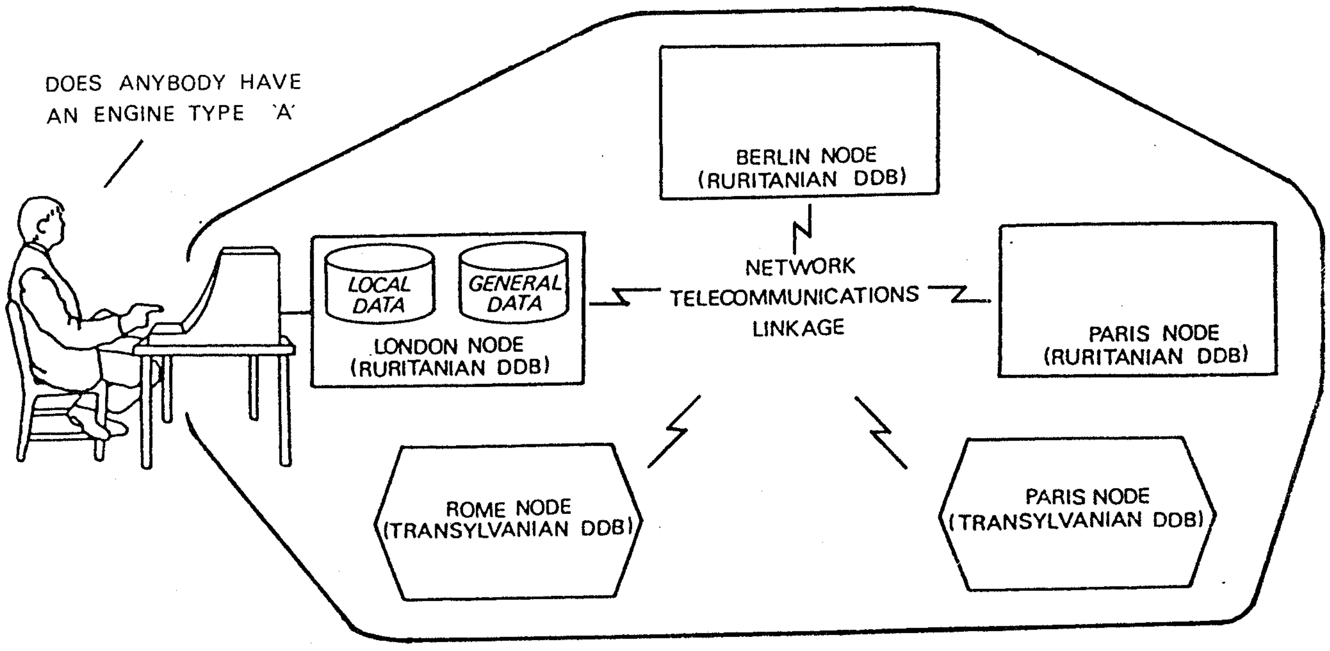

DOES ANYBODY HAVE AN ENGINE TYPE 'A'
BERLIN NODE (RURITANIAN DDB)
LOCAL DATA
GENERAL DATA
LONDON NODE (RURITANIAN DDB)
NETWORK TELECOMMUNICATIONS LINKAGE
PARIS NODE (RURITANIAN DDB)
ROME NODE (TRANSYLVANIAN DDB)
PARIS NODE (TRANSYLVANIAN DDB)

This example also addresses the basic requirement, to give each user read-only access to data which is held at one or more distant nodes. A major new feature is added, namely that users do not have to know and identify the logical node from which they wish to retrieve data on each occasion. Where this constraint is applied only to end-users, the applications software can be designed to use knowledge of the exact distribution of the data in order to contact relevant nodes for data on behalf of the end-user. Where the constraint is also applied to system designers and programmers, the systems software is forced to perform the functions involved in locating data and making its distribution across nodes transparent to both end-users and applications software. Clearly, someone ultimately has to know and control the data distribution, in order to perform database administration.

The simplifying assumptions are retained, particularly that of performance of remote accesses not being critical. If such performance were to be critical, then the necessary overhead of obtaining data through the network might be found onerous, because delays in answering queries which involved data held at locations remote from the user would become evident.

Since the user submitting a query is not aware of whether that query involves local and/or remote accesses, there must be some facility for analysing it into sub-queries against individual nodes. Even without critical performance constraints, that process has to choose a reasonable sequence of retrievals, so as to avoid unnecessarily large data transfers across the network. This analysis can take place at the user node, for simplicity of control; at a central node, to reduce the workload on other nodes; or everywhere, by broadcasting the entire enquiry. The last approach is best suited to the situation where each node can readily identify and respond to that part of the query which is appropriate to itself.

(c) Queries Across Nodes Having Different Regimes


DOES ANYBODY HAVE AN ENGINE TYPE 'A'
BERLIN NODE (RURITANIAN DDB)
LOCAL DATA
GENERAL DATA
LONDON NODE (RURITANIAN DDB)
NETWORK TELECOMMUNICATIONS LINKAGE
PARIS NODE (RURITANIAN DDB)
ROME NODE (TRANSYLVANIAN DDB)
PARIS NODE (TRANSYLVANIAN DDB)

In this case, users are to be given access to data held at remote nodes under different data management regimes. A common example of this is where outlying user nodes are on smaller machines supporting only conventional file structures, whereas one or more large machines on the network support a full DBMS. Alternatively, more than one major DBMS may be in use, with a requirement to share data resources across the nodes concerned. Large systems may have both multiple DBMS types and conventional files. It is impractical to expect end-users to interface differently with each node, according to its regime. Equally, it is highly desirable that a uniform interface to both local and remote nodes be available to programmers.

Although the distribution of data across identical nodes can be made almost transparent to users, distribution across different nodes is likely to be visible at least to programmers and quite possibly to end-users as well. This visibility can take the form of constraints on the range of data objects and structures which are handled by the interfaces between nodes. A good example of this in a business context is the supply of current selling prices from a central node, and the return of sales statistics by outlying nodes, all other data being local to particular nodes. In general, transparency of data distribution and data management regime is a matter of degree, dependent upon the effort invested in extending the interfaces between local processes. For example, a design limitation may be that any data which a DBMS has to make available to other nodes must be held in a form directly equivalent to serial and indexed files. However, such a constraint affects the service received by local DBMS users, who presumably make most use of their local data.

The fundamental decision to be made is the level at which the data management differences are to be reconciled. Where existing machines are being linked into a network, high priority is likely to be placed on retaining the existing user interfaces, which will reflect local data management regimes at programming interfaces and (indirectly) at end-user interfaces. In these circumstances, a sub-query which needs access to data on a different regime has to be transformed into an equivalent expression in terms of that other data model. Where simpler nodes are sending sub-queries to a main DBMS node, only the latter should handle the transformations, with the option of re-directing sub-queries to outlying nodes for data not held centrally. This approach avoids overloading smaller nodes by the need to transform queries to a central standard, particularly where the data is already available without such transformation.

The chief alternative is for all queries to be expressed in terms of a single standard data model. Where this model is independent of all regimes used for storage, a high degree of flexibility is gained, at the cost of having to transform every sub-query twice in each direction. Where one of the storage regimes is adopted as the standard, a choice remains of whether it is applied only to sub-queries between the regimes, or it will form the basis for expressing queries at the end-user level, at all nodes. In the latter case, transformations are needed for local sub-queries upon any nodes which do not adhere to the standard.

Transformations have to cater not only for a correspondence of data objects and action requests, but for correspondence of exception conditions also. Simple conditions, such as 'no data for that key value' may be directly comparable, whilst more complex ones are likely to be meaningless under a different regime. The obvious course is to group finer distinctions within broad categories, when failures or other conditions are to be reported across regimes. The concept of currency is particularly difficult to map between regimes, and reinforces the need to express sub-queries at as high a level as possible, above the peculiarities of the individual regimes concerned. Where it is available, use of a high-level query facility is valuable for avoiding user involvement in the differences of data models. Ideally, uniform dictionary and directory facilities should also be employed in support of query processing, but for many combinations of regimes such portable software is not available. In its absence, either multiple sets of the control data must be maintained in parallel across the network, or a central version must be shared with other nodes.

(d) Periodic Update (Homogeneous Case)

The requirement addressed here is that of making occasional updates which involve more than one site, as well as accommodating queries. As this is the first case dealt with in which updates, as well as queries, are involved, an extra constraint is imposed on the facilities which must be provided by the system, to permit temporary storage of updates at the local node.

The periodic update facility raises a number of fresh issues. First, the question of when a periodic update service becomes an on-line update service. A service which imposes a time delay of a few minutes between submission and execution of an update would be regarded by some users as on-line, and by others as periodic. For this reason, no firm line is set between a periodic update service and an on-line update service; it is assumed that the threshold lies somewhere in the region of five minutes. The question may, to a large extent, be linked to the forms of transmission allowed. Clearly, the absence of a continuous teleprocessing link will imply periodic (as opposed to on-line) update, but the converse will not always obtain.

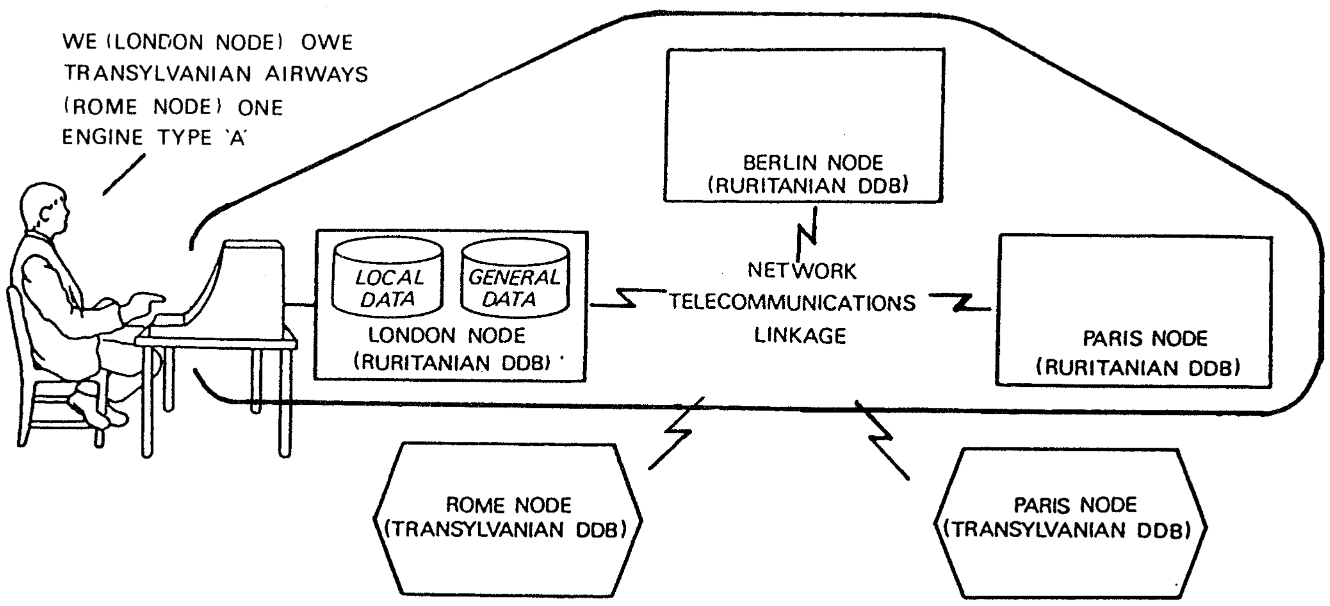
In most cases, the existence of a continuous teleprocessing link will allow the system designer the option of providing an on-line update service if he so wishes. In a minority of cases, where data rates are very low (as may, for example, be the case in aerospace or telemetry applications), even a continuous link will only enable periodic update.

The question of whether it is necessary to broadcast the updates throughout the network will depend on the number of sites involved in individual transactions, and on the message load imposed on the network by the use of the technique. The risks of network congestion and long response-times have to be balanced against the more complex system design work involved in the direction of updates to specific nodes. For practical purposes, in most cases where the update service is periodic, updates will be directed to specific nodes, rather than being broadcast.

The question of how database integrity is preserved concerns the speed with which any loss of integrity can be detected and corrected. Techniques such as two-phase commit protocols, which minimise the risk of loss of integrity, may need to be backed up by periodic consistency checks, designed to determine whether database integrity has been preserved.

A typical solution to the user's requirement will be one in which he can execute multi-site query transactions at will, but if he wishes to update data at a remote node, he needs to store the updates in his local node until they can be processed in batch mode. An update processing routine then operates either at regular intervals or on a queue-driven basis.

(e) On-line Update (Homogeneous Case)


WE (LONDON NODE) OWE TRANSYLVANIAN AIRWAYS (ROME NODE) ONE ENGINE TYPE 'A'
BERLIN NODE (RURITANIAN DDB)
LOCAL DATA
GENERAL DATA
LONDON NODE (RURITANIAN DDB)
NETWORK TELECOMMUNICATIONS LINKAGE
PARIS NODE (RURITANIAN DDB)
ROME NODE (TRANSYLVANIAN DDB)
PARIS NODE (TRANSYLVANIAN DDB)

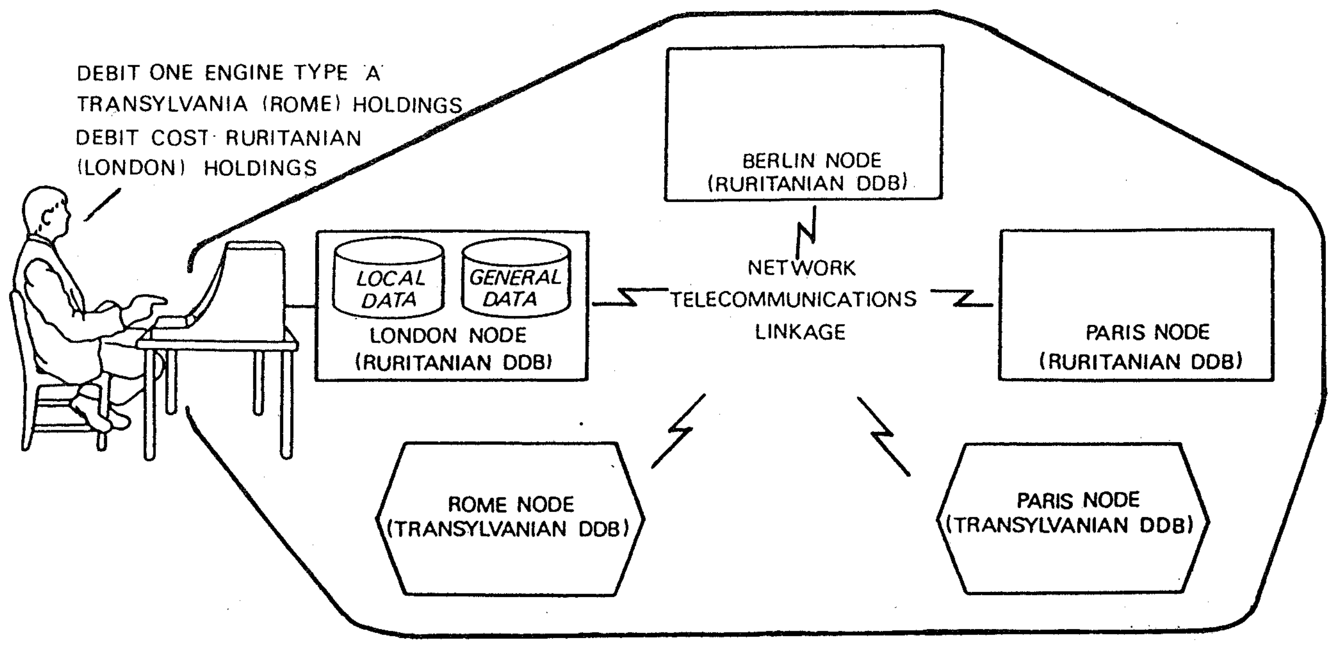
For applications where some or all of the data must be consistent at all times, a system such as that described in section (d) of this chapter is inadequate. The period between updates has to be reduced to the time between processing of transactions causing changes in the data. A transaction can require updates to be carried out at a number of nodes. From the viewpoint of the applications programmer, it is simpler if he does not know the number of copies of data and their locations. However, such provision of replication and location transparency implies a degree of complexity usually found only in systems like those described in section (g) of this chapter. Consequently, this section assumes that neither replication nor location transparency is provided.

Since the system provides no transparency, the applications programmer must control the updating process: this raises several questions pertaining to updates across the system, and further important issues in the synchronisation of updates and the handling of nodes which are currently off-line.

The alternative approaches are to transmit updates to either all nodes or only specific nodes, the choice being dependent largely on the control structure. If local nodes have details of data distribution, then the update of specific nodes can be carried out. If data to be updated is held at most nodes, it may be simpler to broadcast the update to all nodes and allow those which hold the relevant data to make the appropriate changes. If details of location are held centrally, then the update can be transmitted to the central node for forwarding to relevant nodes. If each update must be carried out simultaneously across all sites, it is necessary to lock the relevant records at each node. This is complicated, and adds substantially to the complexity of the control system.

A further problem which must be considered is the node which is currently off line. There is a variety of reasons for a node being off line: it may be due to a communication failure, leaving the node operational but isolated; the node may be closed down due to a local fault in either hardware or software; night-time closing may be local routine. Most solutions to this problem utilise a log file held at one or more nodes to resynchronise the node when it is restarted.

(f) Update Across Nodes Having Different Regimes


DEBIT ONE ENGINE TYPE 'A' TRANSYLVANIA (ROME) HOLDINGS
DEBIT COST RURITANIAN (LONDON) HOLDINGS
BERLIN NODE (RURITANIAN DDB)
LOCAL DATA
GENERAL DATA
LONDON NODE (RURITANIAN DDB)
NETWORK TELECOMMUNICATIONS LINKAGE
PARIS NODE (RURITANIAN DDB)
ROME NODE (TRANSYLVANIAN DDB)
PARIS NODE (TRANSYLVANIAN DDB)

This case extends that of section (c) from query handling to update processing, across different regimes of data management. As for queries, there has to be a known correlation between the data objects, between the actions for data manipulation, and between the resulting conditions, for such updating to be feasible. The more similarity there is between the regimes, the more detailed the cross-regime interface can be. For instance, updates across databases conforming to the CODASYL data model are in principle far easier to specify than those across, for example, a CODASYL and a relational data model.

The most robust form of solution is one which translates all incoming update requests, regardless of origin, into a standard form which is independent of the various data models on the network. In this form, each request is analysed (by application code or a distributed DBMS) into sub-requests upon local and remote nodes. These sub-requests are transformed by the nodes concerned, and processed in terms of the data management routines at those nodes. The results are transformed back to the standard format, returned to the originating node, and there combined with local results into a single reply. This is again transformed for presentation at the local user interface. Since the transformations are applied to all update requests, not just to sub-requests upon other regimes, the overheads could be crippling, or require dedicated processors to support a 'black box' transformation service at each interface.

Although high-level query packages provide some ready-made protection from differences in data management, generalised update packages are rare, particularly for use across differently implemented data management regimes. Nevertheless, it may be possible to adopt the syntax and extend the semantics of a standard query facility, to express update requests in a uniform manner across a network. Since the variety of data needed for control of data integrity, availability, and privacy is much greater than for queries, the number of data dictionaries and directories which can provide active support of update processing across multiple machines is very small. Those supporting such capabilities are being developed from central dictionary/directory functions, primarily for use on homogeneous networks.

Use of a single shared dictionary/directory imposes such overheads upon the already heavy data traffic of updating across nodes that availability of those functions at each node is almost a necessity. Hence, a different dictionary and directory capability is likely to be required for each regime, together with the means of keeping their contents in step across the network. In principle, the latter function would be performed by the distributed update capability, but this would only be possible where the dictionary/directory software conformed to the network standard for expressing update requests.

Performance is clearly a major issue for multi-regime updating. Consider the process of analysing a user's update request into sub-requests. A particular kind of decomposition may be optimal for one data management regime, but unacceptably inefficient for another. This shows that, where performance is a significant factor, agreement of standard syntax and semantics for sub-requests does not provide more than token transparency of differences between nodes. The analysis process at each node has to take account of the different needs of the nodes to which sub-requests are to be sent before generating those sub-requests. Since a single update request cannot sensibly be decomposed into two different sets of sub-queries, some inefficiencies are, in practice, inevitable. The important thing is to choose those inefficiencies which will least affect the overall service provided to the users.

(g) Highly Reliable Service

A highly reliable service in a distributed database implies that a high reliability must be provided by the separate components of the DDB. It also implies that availability levels must be high, and that the risk of loss of integrity must be very small. Specifically, the hardware at each node must have high reliability; the risk of messages being lost by the network must be very low; concurrency must be maintained; and deadlock situations must be detected and resolved. In addition, it is important that the DDB provides a reliable service in situations in which a number of nodes have failed. Where this problem is of prime importance, special techniques may be needed.

DDB technology does not as such provide techniques for improving the reliability of the hardware at individual nodes. It is possible, by the application of reliability theory, to calculate the availability of the network for a given purpose, where availability is expressed as:

$$\frac{\text{MTBF}}{\text{MTBF} + \text{MTTR}}$$

where MTBF is 'mean time between failures' and MTTR is 'mean time to recovery', provided that information on network connectivity, node reliability and link reliability is available. However, although considerations of reliability at individual nodes are important, they are not generally regarded as part of DDB technology.

The risk of loss of a message by the network is a function of the reliability of individual nodes and links, and of the network routing techniques used. Network routing techniques fall into two groups, probabilistic and deterministic. Probabilistic techniques are claimed to be more efficient at finding the best route through the network, but they have the disadvantage that a very small, but finite, risk of a message being lost by the network message routing software does exist. Deterministic routing techniques, although they may have efficiency drawbacks, are more suitable for DDB applications requiring high reliability since they do not involve this risk. However, they do not entirely eliminate the risk of a message being lost at some point due to the possibility of node or link failure.

The risk of message loss is most threatening during the processing of an update, as such a loss could result in a loss of database integrity. Recovery from this situation can be very difficult, particularly in an environment in which updates are being actioned in rapid succession. Many designers of DDBs have chosen an approach which is based on a 'commit protocol' to minimise risks which arise during the updating process.

A commit protocol is a high-level protocol which is concerned purely with the preservation of database integrity. It is independent of lower level protocols which are concerned with the delivery of messages. The purpose of a commit protocol is twofold. Firstly, it ensures that all sites involved in the processing of a multi-site transaction agree on whether or not to process the transaction. Secondly, it reduces to an absolute minimum the time period (or 'window') during which the failure of a link or node can result in loss of database integrity.

The most commonly used commit protocol is the 'two-phase commit protocol'. In the first phase of its operation, each site involved in the transaction is queried as to whether it can commit its part of the transaction. Each then enters a 'ready' state, prepared to either commit or abort the transaction on the decision of a previously selected 'transaction co-ordinator'. In the second phase, the transaction co-ordinator transmits its decision, and all the sites then either commit or abort. The technique carries high overheads, especially in large networks, so to optimise performance specific variations have been implemented. The most important of these are the 'linear' and 'centralised' two-phase commit protocols.

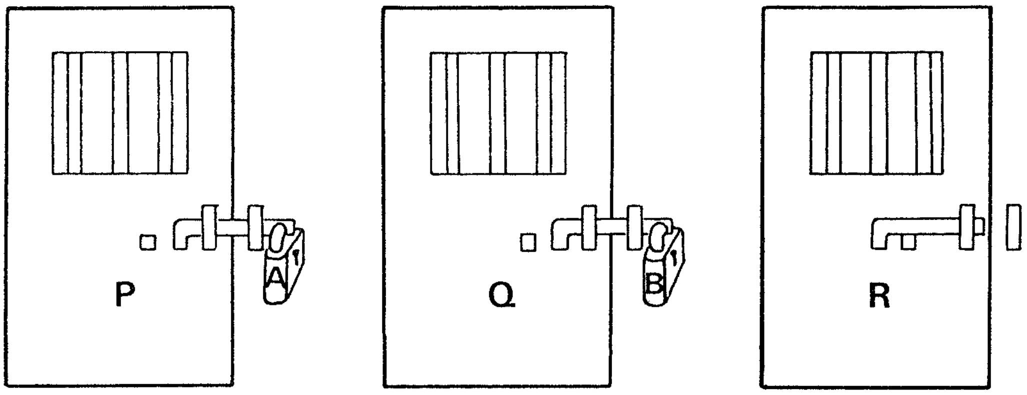
In the linear two-phase commit protocol, sites involved in the transaction are ordered into a sequence, and the commit proceeds with the messages along the sequence. Each site enters the ready state as it passes the transaction to the next, the last site in the sequence becoming the transaction co-ordinator, and so initiating the commit or abort decision, which is then relayed through the sequence in reverse order.

The centralised two-phase commit protocol uses the site which originates the transaction as the co-ordinator, and this site transmits both the alerting and the commit/abort messages. Comparison of the two approaches indicates that the linear protocol requires fewer messages, but the centralised technique obtains better response times for large numbers of sites.

For certain types of network topology, the two-phase commit protocol can be replaced by the more specialised 'safe-talk protocol', which eliminates the final acknowledgement message of the commit phase, showing a saving of twenty-five per cent of messages over the two-phase commit protocol. Other protocols are being researched, and it is as yet uncertain which will prove the most efficient for particular requirements.

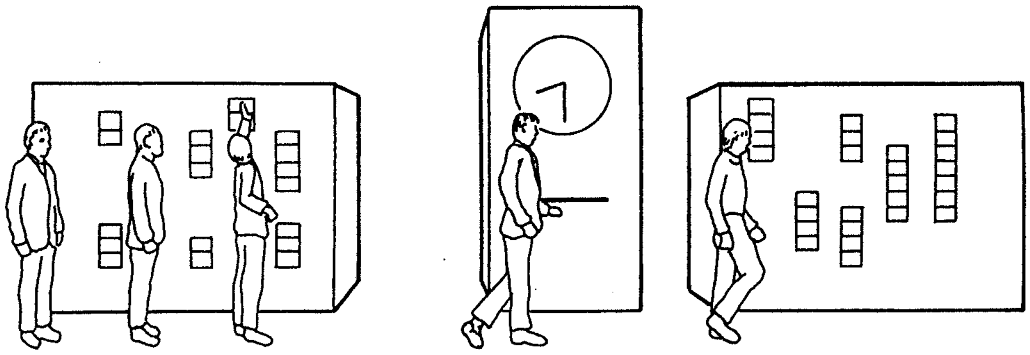
In DDBs, the techniques used to ensure the maintenance of database integrity during concurrent access by a number of transactions are collectively referred to as concurrency control. Three main approaches to concurrency control - locking, time-stamping, and majority consensus methods - have been developed. Certain optimisation techniques have also been developed, mainly with a view to reducing the high overheads associated with the locking process.

The use of locks on data items was originally developed for use with centralised DBMSs, and has subsequently been extended for use with DDBs. The most commonly used method is 'two-phase locking', which requires that each transaction obtains locks on required data before processing it, and that a transaction does not obtain any new locks after it has released a lock. Various two-phase locking techniques have been developed. One - 'primary site' - relies on a central lock controller to manage the locks; others involve distributing the locks with the data. Where the data is replicated, it is possible to designate one copy as the primary copy, and to lock this copy only.



LOCKING (Data item P is locked to transaction A, Q to B - item R is not locked)

Time-stamping involves the association of a unique time-stamp with both each transaction and each data item. Database concurrency is maintained by only permitting the updating of a data item by a transaction bearing a later time-stamp. Time-stamps can also be used to synchronise reading and writing operations. They can involve high storage costs, but, when data items have not been updated for a period, their timestamps may be dropped, without risk to concurrency.

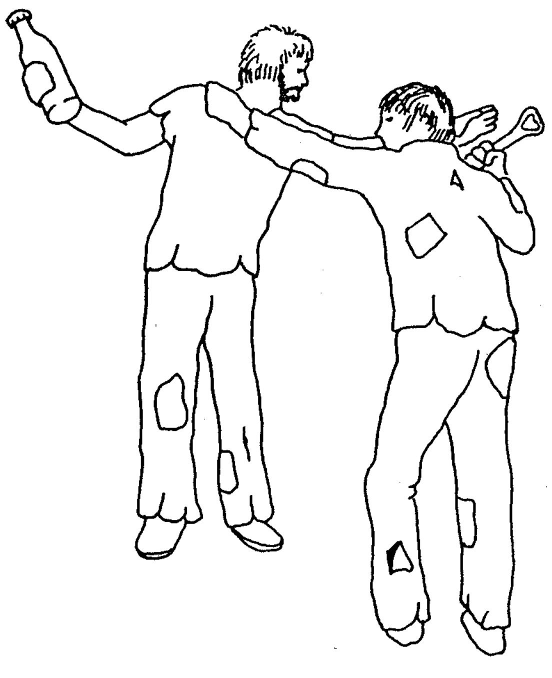


TIME-STAMPING

The majority consensus (or voting) algorithm relies on a count of 'votes' from individual nodes to decide whether a data item should be updated. Its main advantage is its resilience against the failure of individual sites. However, its complexity is a drawback, and it has not yet been used in the more important DDBs.

MAJORITY CONSENSUS (VOTING)

A deadlock situation arises when two or more transactions cannot be completed because each is waiting for some other transaction within the same cycle to release some resource. A common technique for detecting deadlock cycles in centralised DBMSs is to use a 'transaction-wait-for' (TWF) graph. A deadlock cycle is normally broken by forcing one of the contributing transactions to abort, then resubmitting it when the other transactions in the cycle have been executed.



In a DDB, deadlocks can be detected by each node maintaining a copy of the TWF graph. Since this involves large overheads in exchanging data between sites to keep the graph up to date, other techniques have been developed. One is the use of a global TWF graph, which is held at a designated site: other techniques rely on concurrency control methods to resolve deadlocks: in still further cases, total reliance is placed on the DBMS software at individual sites to detect and resolve global deadlocks.

DEADLOCK

CHAPTER FOUR

METHODOLOGY AND COST

General

Having looked at the concept of distributed database, considered contexts in which this approach to data management might be adopted, and suggested a number of specific types of model of such systems, we can now examine more deeply some of the essential features of distributed database design.

The design of a DDB system can be seen to have the same objectives and many of the same problems as that of a centralised DBMS, but the data for a DDB system is held at a number of separate nodes. So the main characteristics of the database approach (such as data independence, data sharing, minimum redundancy in stored data, data integrity, data consistency, etc.,) do apply, with the difference that each node can own and be responsible for its own data, and with the added problems of distribution, synchronisation over a communication network, and possible lack of homogeneity among nodes.

In this chapter, we will discuss techniques which are essential in the construction and use of a distributed database, under the following headings:

(a) Database definition,

(b) Data independence,

(c) Inter-node communications,

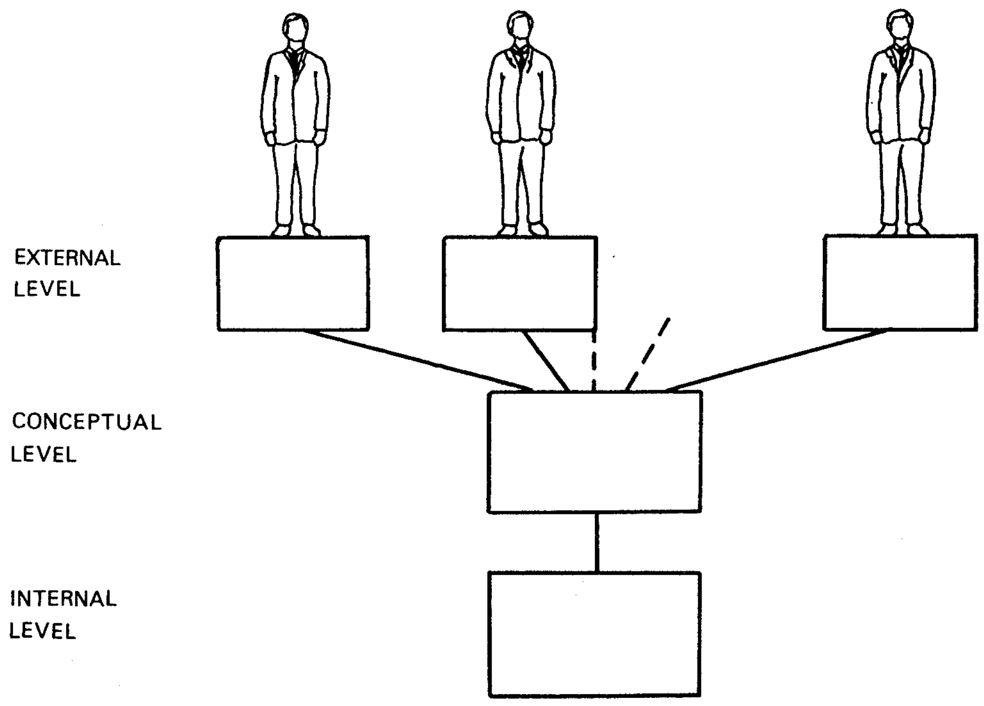
(d) Access control and integrity,

(e) Failure conditions, resilience and recovery,

and (f) Performance, re-organisation and restructuring.

(a) Database Definition

Readers may well be familiar with the ANSI/SPARC model of a database, which utilises three levels of database definition - a conceptual (or community) level, which defines the logical structure of the overall database, independent of any particular application or user view; an external (or individual) level, which defines a subset of the database as seen from one viewpoint (there will be a number of such viewpoints); and an internal (or storage) level, which defines the form in which the data is actually held. These three levels can be identified in both distributed and centralised databases, but, in the general case, a DDB must also take account at each level of a further dimension - the autonomy of its constituent nodes.



THE LEVELS OF DATABASE DEFINITION

Global is the term used for functions with a centralised view of the data (e.g., a global query is a query asking for pertinent data from other nodes), local being that for functions which refer to just one node. This terminology helps to identify and describe various aspects of the architecture of distributed database definition, as in the following definitive terms.

A Global Conceptual Schema defines the structure of the overall integrated DDB, independent of any one application or node. This schema may be held at one designated node, or copied at each constituent node of the network. In the latter case, it is necessary to ensure that the copies remain consistent, as well as maintaining data consistency.

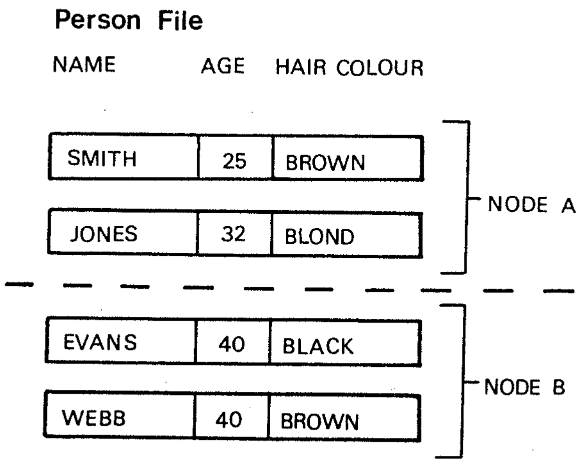
A Global External Schema is a subset of the integrated DDB as seen from one viewpoint (e.g., a sales manager's viewpoint). The same viewpoint may be held at different nodes. For instance, if each node corresponds to a subsidiary company of a corporation, then the sales manager of each company may be able to access the corporate DDB to obtain information about the corporate sales situation. The 'view' seen by each sales manager would be the same.

A Local Conceptual Schema is an application-independent definition of the database available at a site. Local databases will often have been created before the integrated database is set up, and will not necessarily be subsets of the overall system. For example, a local database might contain data about local public services, which is of neither interest nor relevance on the global scale.

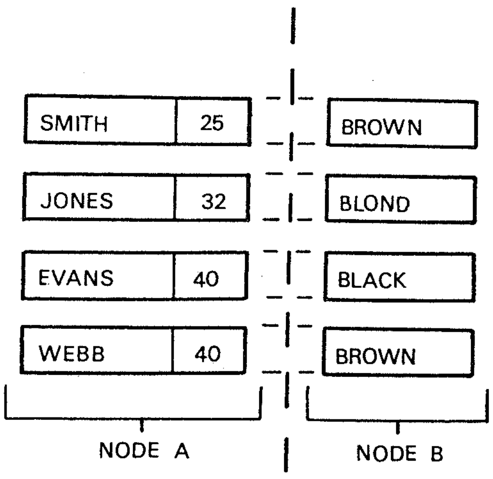
A Local External Schema is a subset of a local database as seen from one viewpoint. This, for example, could be a sales manager's view of his own company (as distinct from the parent corporation).

Similar comments can be made about internal schemas. 'View' is sometimes used as a synonym for 'schema', but it is preferable to use 'view' for a set of occurrences (e.g., the actual data seen by a sales manager on different occasions) and 'schema' for the definition of a view.

It is clear that a DDB system needs more levels of description than a single database. Unfortunately, there is no DBMS or DDBMS known to this group which enables either a user or the database administrator to make the clear distinction between levels of interest and locality which is implied by the examples above. However, the examples do illustrate the necessity of introducing a method of describing connections between data from the different local databases, if there is to be global use of the totality of the data. A number of different architectures for this have been proposed.



Horizontal Partitioning



Vertical Partitioning

One matter to be considered is the criterion for the distribution of data - is it to be held at the node of origin, or at the node of greatest use (however this is defined)? Given that data is to be partitioned, is horizontal partitioning (such that a node will hold complete records, but not a complete set of records) or vertical partioning (so that a node will hold a complete set of certain fields - attributes - of the records, but no complete records) to be preferred?

Some decisions involve the operational performance of the database as much as its definition. Problems which arise in this area are the required or desired extent of replication, whether constituent nodes will all use the same DBMS for their local databases (an homogeneous DDB) or whether the DDBMS must also convert global queries from one type of DBMS to another (a heterogeneous DDB).

A further necessary definition - that of a Mapping - is: the process whereby data or data manipulation commands are translated from one structure or view to another. Mappings are particularly pertinent to DDBs, their essential basis being an adequate knowledge of the source and object structures involved.

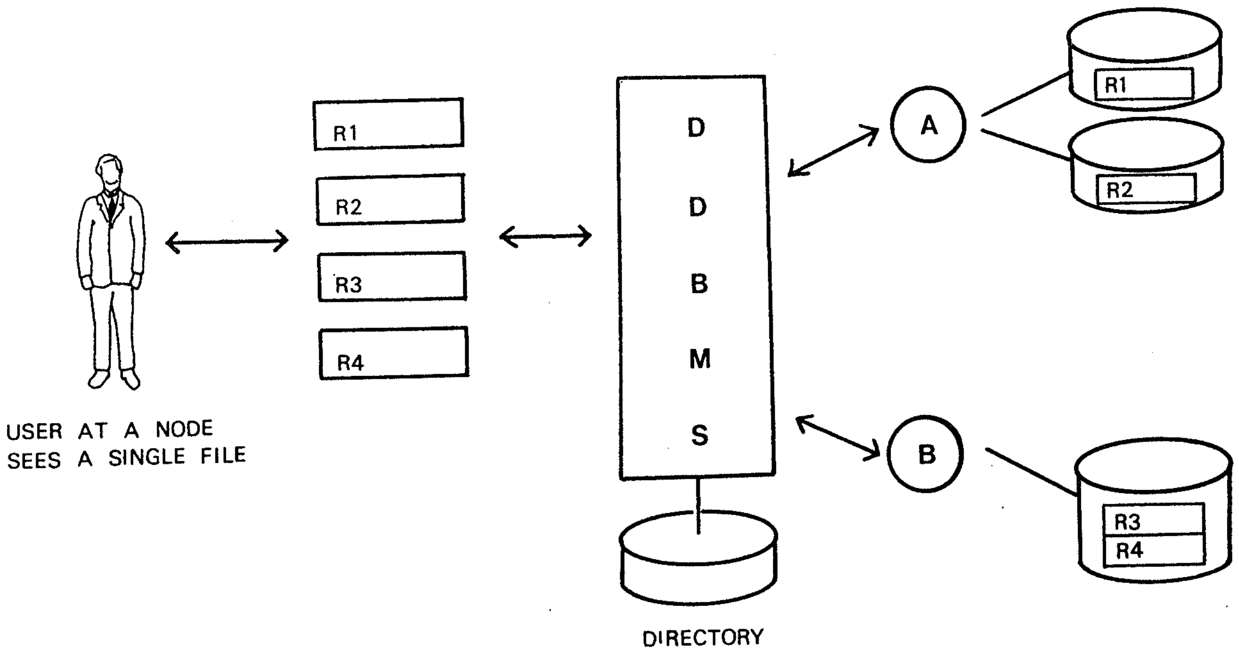
A data dictionary can be used to store the various schemas, as with a centralised database, but its importance is even greater in a distributed context, as it becomes a fundamental tool of documentation for the administration of the DDB system. Just as for schemas and data, a distinction can be made between a local data dictionary, which relates only to its local database, and the global data dictionary, which relates to the total DDB. The global data dictionary can draw much of its data from the various local data dictionaries. Homonyms and synonyms, for example, are likely to be especially common in a DDB, and must be resolved at the global level.

(b) Data Independence

A principal objective of any database management system, whether centralised or distributed, is to make it possible to alter the database without affecting the applications programs. The degree to which this goal is achieved varies greatly, as performance or fail-safe factors may constrain the system. Here, three aspects of data independence are discussed, the distributed database features being emphasised within each.

A primary requirement is that data access by programs should not be dependent upon database organisation. A centralised database system strives to shield the applications programs from the organisation of the database such as use of randomisation methods, data type conversion, new indexes, et cetera. The necessity is the same in a DDB, and, as the above topics are already well documented, they are not further discussed here.

A common feature of both centralised and distributed database systems (though less complex in the centralised case) is location transparency, that is that the application has no knowledge of the physical location of data. In the distributed case, the application should not be required to know even the nodal location of data. This transparency is achieved through use of a directory which holds the locations of all the sections of the database. The three main types of directory basis can be described as centralised, local, and replicated master.



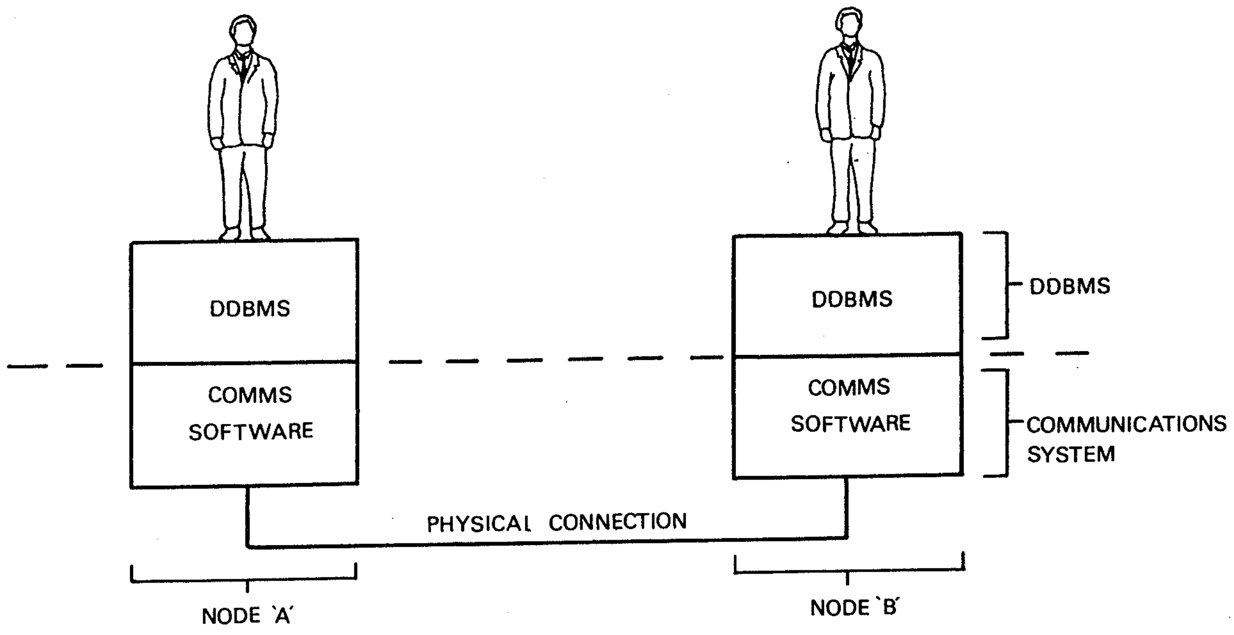
LOCATION TRANSPARENCY

A centralised directory implies that all nodes, other than that at which the directory is resident, must enquire of it the location of data to be accessed. A local directory basis means that if data to be accessed is not at the local node, a search for that data must be made at other nodes. This can be very costly in communication terms, unless the communications network provides efficient broadcast facilities. The replicated master directory approach involves replicating the master directory (or the centralised directory) across all nodes, with the effect of fast response times and little communications traffic for directory enquiries. However, it is costly in storage terms and requires careful version control.

Above, a simple allocation of record types to nodes is assumed. However, as explained above, more complex rules for distributing data are possible, resulting in partitioning or replication. The third type of independence required is therefore independence from the particular scheme of partitioning or replication in use.

(c) Inter-Node Communications

The nodes in a distributed database system require to communicate with each other. The communications network which is used to support this requirement should not be seen as part of the distributed database system. Rather, the communications network should be seen as a separate system, providing a service both to the distributed database management system and to any other software requiring inter-computer communications.



SEPARATION OF COMMUNICATIONS FROM DDBMS

Since the communications network is considered to be a separate system, the techniques required to construct such a network are not described here. However, some of the general issues are introduced.

The main requirements that a distributed database system will have for a communications network are that it can transmit messages between any nodes, and that the transmissions are reliable and error free. A further important aspect is whether the communications are required in a homogeneous or a heterogeneous machine environment.

Communications networks available have progressed from old-fashioned inflexible topologies (e.g., star or hierarchical networks) to ones where connectivity is permitted between all nodes - essential for a DDBMS. There are two main reasons for this advancement: firstly the ability of national telecommunications administrations (e.g., British Telecom) to provide extensive public data network (PDN) facilities, and secondly the advent of layered network architectures.

The PDNs are based on digital transmission techniques and replace the analogue circuitry of the original telephone systems. These digital networks, utilising packet or circuit switching techniques, offer the advantage of higher transmission rates with reduced error occurrences. Computer manufacturers have shown their support for the PDNs through the introduction of products capable of being connected to the networks.

The major manufacturers have begun to implement layered network architectures with very full functionality, e.g., IBM with Systems Network Architecture and UNIVAC with Distributed Communications Architecture. The lower layers cover the transport of data over communications links. The higher layers are concerned with supporting the applications programs specifically by managing and providing formatting of the messages to be exchanged. This high degree of functionality is, however, only readily available in homogeneous networks.

Perhaps more significant is the advancement being made towards international standards for communications networks applicable to a heterogeneous environment, e.g., the International Standards Organisation with its basic reference model for Open Systems Interconnection (OSI). This describes a framework for a seven-layer architecture based on existing and rapidly developing standard protocols. The CCITT recommendations X21, HDLC and X25 fit into the lower three levels of the OSI model: however, much work is yet required at the higher levels of this model.

(d) Access Control and Integrity

Problems of access and integrity which occur in a DDB concern the privacy, integrity and consistency of the database, and concurrent access to it. There are also indirect problems, such as deadlock, which result from using certain sychronisation methods in attempts to prevent some of the problems mentioned above.

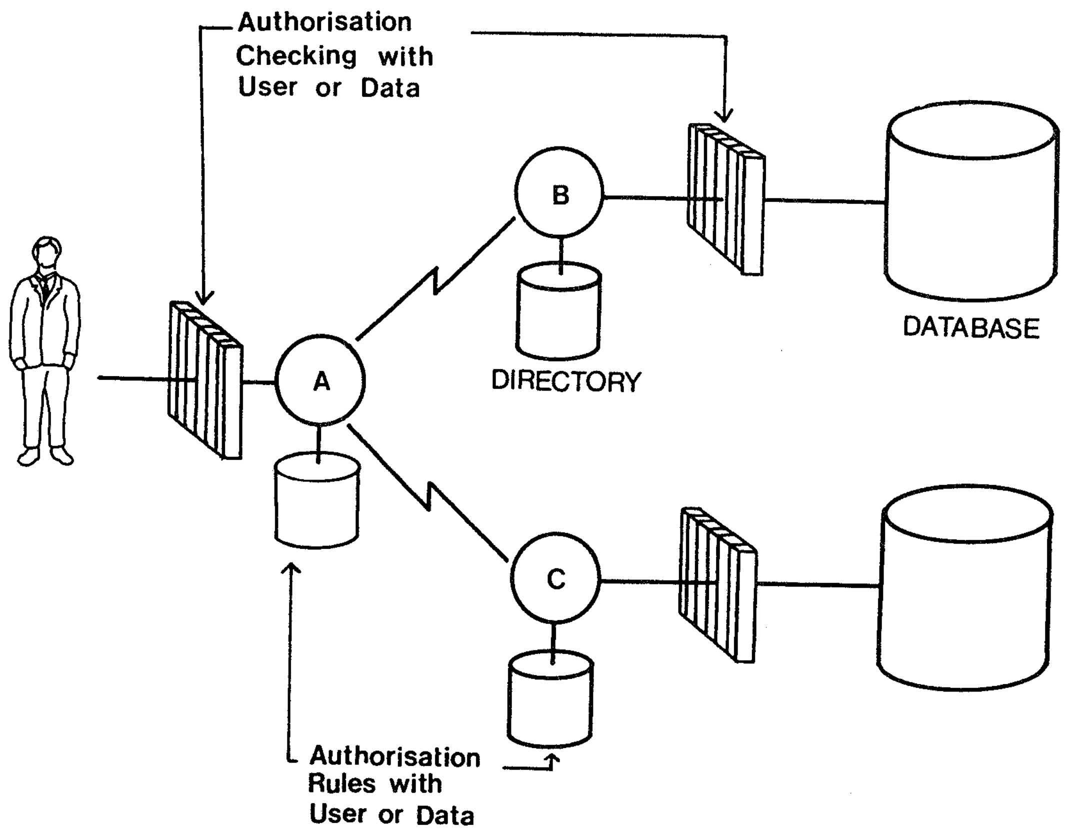
Security of Access

In organisations where information is kept, there is always a risk that data will get into the wrong hands. A security control mechanism is necessary to ensure that confidentiality is maintained. Privacy is the right to restrict access to data. Authorisation is the right to view restricted object types (applications programs, view tables, data). To obtain unauthorised information is an access violation, of which four types have been identified, viz. theft of media, interception of communication lines, violation of computer security, and decoding the radiation produced by computers and communications equipment into information being processed. The privacy problem exists in a centralised DBMS, but it is more serious in the distributed situation, as data is stored in more than one location and is accessed from many different places.

The mechanisms used in a system to prevent access violation and to maintain privacy within it are referred to as security control, three important aspects of which are the where, the how, and the cost.

Security control can be centralised, with authorisation checking performed at a single node, or distributed, with checking performed at each node. In the distributed case, the process for authorisation checking of requests could be at the user end, with requests checked at initiation, or at the data end, with authorisation checked at each access to the database. Authorisation records can be held at either the user end or the data end. Security may be checked as a separate process, or may be integrated with the data access procedures. A key criterion for choosing between strategies is whether the system aims to improve execution efficiency or to increase site autonomy. Checking authorisation at the user end should save on communication time, and so increase execution efficiency, whilst checks at the data end increase nodal autonomy within the network and imply one mechanism per node.

Methods of security control are dependent on the type of violation prevention required. There are many forms of safeguard against access violation, which can be implemented at different levels in the system. The first is simply to prevent intruders logging in to the system by the use of passwords. Inside the system, there can be software safeguards at conceptual and external schema levels, such as logical security constraints and access control tables. Logical security constraints allow the user to view restricted statistical data (e.g., mean wage, number of employees) but not single elements (e.g., a particular employee's wage). Access control tables prevent unauthorised users from viewing particular restricted data. At the internal (storage) level, privacy transforms may be used to encode data into an unintelligible form. This method can also be used in the prevention of effective interception of communication lines.



Authorisation Checking with User or Data
B
DATABASE
A
DIRECTORY
C
Authorisation Rules with User or Data

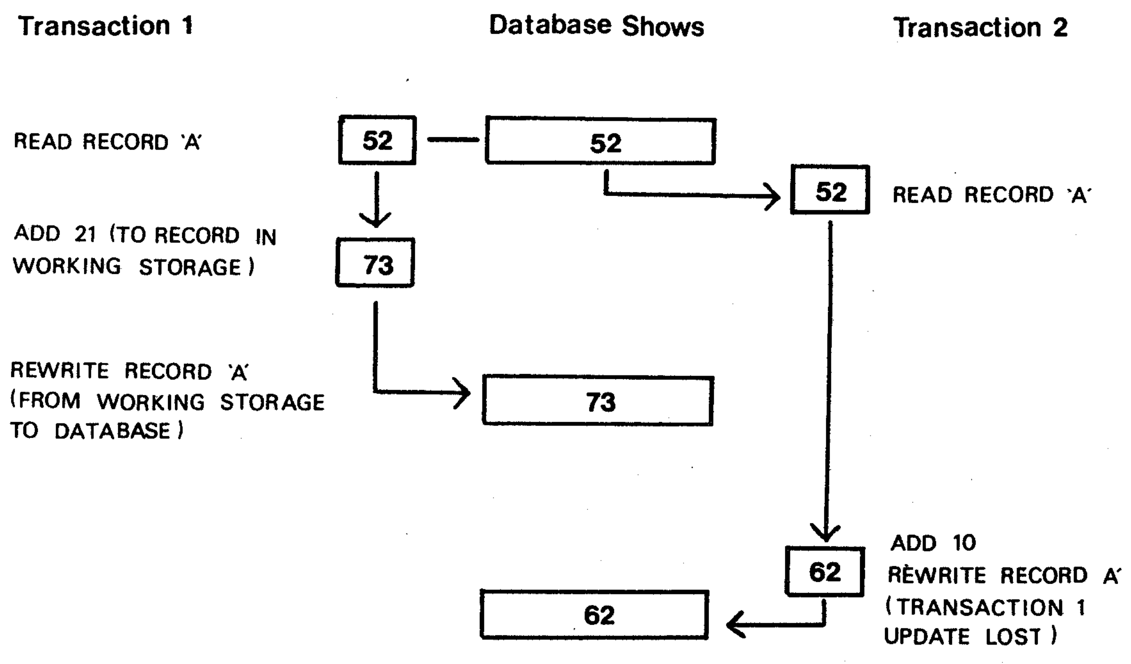
SOME OPTIONS FOR STORING AND APPLYING AUTHORISATION RULES

When weighing the cost of security control mechanisms, the user must consider the effect an information leak would have on the organisation. This should put the value of protecting restricted information into perspective. Having then analysed the threat, the organisation can implement relevant security mechanisms. Communication costs should be low if authorisation checks are made at transaction initiation, as in the user end type of system. However, this requires that authorisation records are held at the user's node. In fact, authorisation records will normally be held at many such nodes, in which case storage costs are incurred, due to replication of these records. Alternatively, authorisation records may only be kept local to the data. With this method, high communication costs would be incurred if security checks are made as a separate process. A data end security check affords the possibility of saving in storage of authorisation records, but the user pays in execution time and communication costs between nodes. The main cost incurred by installing logical security constraints is an increase in execution time, as every query must be checked for validity.

Consistency, Integrity and Concurrency

The consistency constraints of a system are defined by rules and descriptions in the conceptual schema. There are three factors which threaten the consistency of a database: user update errors (which can introduce inconsistent data); concurrency conflicts (which can induce improper sequences of operations to be executed); and site crashes (which can prevent the completion of update transactions). When consistency is maintained at all nodes, it is said to be strong, but when the possibility exists that one of the nodes in the system is temporarily inconsistent, the consistency of the system is weak.

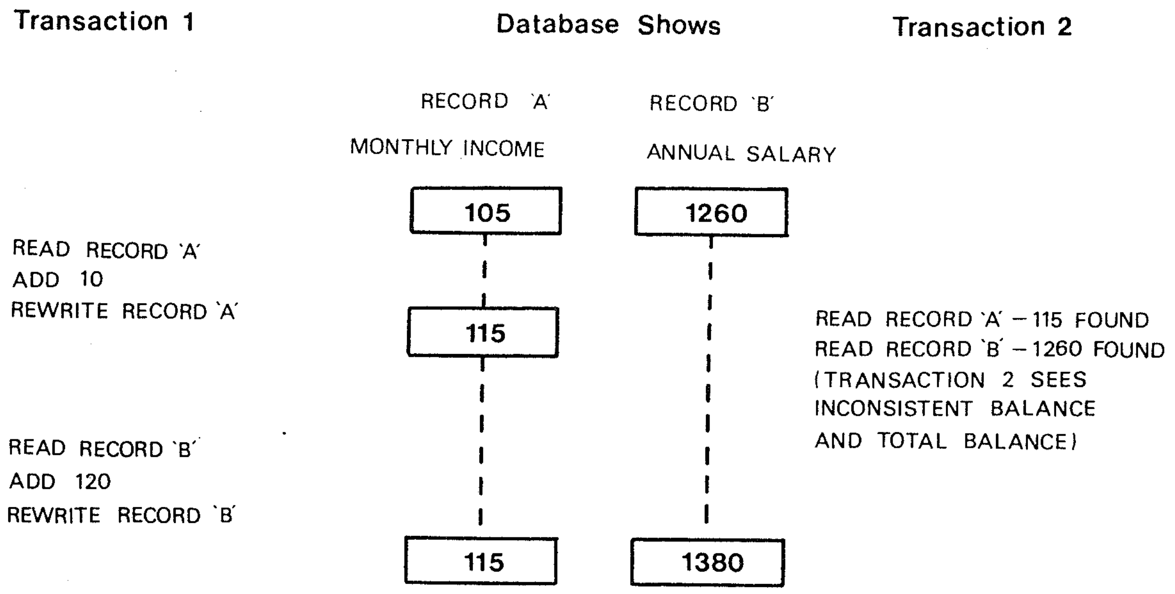
A system is said to be concurrent if it is able to execute more than one transaction in parallel. The objectives of concurrency are to increase throughput, improve response time and achieve better system utilisation. The price paid for allowing concurrency into a system is the high probability that inconsistencies will enter the databases unless preventative measures are taken. Concurrency control mechanisms are used to co-ordinate concurrent accesses to the database in a single DBMS, multi-access system. In a distributed system, the control mechanisms at each node must be extended to take account of the updates being processed at all the other nodes.



Transaction 1
Database Shows
Transaction 2
READ RECORD 'A'
52
52
52
READ RECORD 'A'
ADD 21 (TO RECORD IN WORKING STORAGE)
73
REWRITE RECORD 'A' (FROM WORKING STORAGE TO DATABASE)
73
ADD 10
REWRITE RECORD A' (TRANSACTION 1 UPDATE LOST)
62
62

LOST UPDATE

Three types of concurrency error can occur in a system. Lost update describes the situation in which two transactions read and alter the same field in a database record in such a manner that the overlap of transaction execution causes the record to be effectively altered only once. A dirty read results when a transaction reads the output from an incomplete transaction, 'seeing' a database state which could not be observed in a system which has no concurrent transactions. Inconsistent reads are caused by related data records being read by one transaction whilst another updates them, the first transaction not seeing consistent record values.



INCONSISTENT READ

Database integrity is the guarantee that all data values are correct in respect to the transactions processed To maintain integrity, users and applications should not be allowed to introduce inconsistencies or errors into the system as a whole. Factors which allow errors and inconsistencies into the system also hinder the maintenance of integrity in a distributed database management system.

Co-ordination Techniques

Consistency, integrity and concurrency are all inter-related. Concurrency improves execution time and reduces redundancies in the system, but also increases the problems of integrity and consistency. A mechanism to control concurrency also assists in the maintenance of consistency and integrity. Two-phase locking, majority consensus and time-stamping are techniques on one or more of which concurrency control mechanisms are usually based. The two-phase methods are used in mechanisms to prevent inconsistencies in a system. However, they can encourage states of deadlock in the system. Mechanisms which use time-stamping are detection-type algorithms. Majority consensus techniques include global locking and majority vote methods.

In global locking techniques, the site initiating the update transaction alerts all other nodes to prepare to update the data affected. Each node replies whether it is clear, or already processing a concurrent update. If all nodes holding a copy of the data are ready, then all copies are locked and the transaction is executed at each node. If any of the nodes is busy, the transaction is rejected or delayed. Although this method maintains consistency, it allows few updates to be executed concurrently. The advantage of this technique is that all copies of data are updated together. It is cost-effective if the ratio of reads to updates is high.

The primary and secondary update mechanism assumes full replication, and is a centralised method. One node is allotted the role of primary updater. Each update is received by this node, which updates its database and then informs the other nodes (known as secondary nodes) of the update. At the secondary nodes, updates can be logged and run periodically. The main disadvantage of this method is the delay in updating at secondary nodes. It can be modified into a dynamic system for back-up purposes. If the primary node fails, a secondary node assumes the role of primary: then, when the original primary returns, the new primary informs it of its new secondary status and missing updates.

In the majority vote strategy, a transaction is initiated by informing each node holding a replica of the required data. If a majority of these nodes accepts the update, then it is executed and a version number is assigned to the data. Version numbers are monotonically increasing to aid detection of updates lost at any node. Where a node fails and restarts, the database version number held by the majority of current nodes rules and the failed node is reset. A node outside the majority may request to update, but, before it is authorised to initiate the transaction, the node must join the majority by catching up on updates.

The majority read update technique again uses the method of version numbers to check for lost updates. Updates are executed sequentially and are accepted only if the version number on an update is one greater than that of the previous update. Node failure and loss of updates are highlighted by version number. This and the majority vote algorithm aim to maintain consistency and integrity in a distributed database management system, but they do not allow for parallel processing, so the concurrency capability is lost with these two methods.

Deadlocking

Techniques which use locking protocols to maintain integrity and consistency in a distributed database management system can encourage system deadlocks. A deadlock occurs when two or more processors are prevented from further processing because each is holding a resource needed by the other(s). The ideal system would have a deadlock prevention mechanism. As backup to preventative mechanisms which have not been proved to fail safe, many deadlock detection algorithms are available. Should a system detect a deadlock, it must be reset. Three phases are then required - prevention, detection, and resolution.

To prevent deadlock, the pre-ordering of resources or the use of time-stamping of updates can be used. These methods will reduce concurrency on the system, and so increase execution time. Wait-for graphs are used to detect cycles of processes waiting for each other. This method requires large amounts of storage and high communications reliability for the graphs to be credible. To resolve deadlocks, roll back procedures are used to restore the system. A log is required to record all committed updates, this too requiring a significant amount of storage.

There have been many simulations of distributed database management systems built so that the effects and frequencies of deadlocks could be studied. One of these projects concluded from both the simulation and observations measured on an operational database that "deadlocks are rare creatures", which raises the question of the necessity of their prevention. It may be cheaper to check the system for deadlocks periodically, or when the system appears to be in a wait state.

(e) Failure Conditions, Resilience and Recovery

Failure Conditions

In a state of the art DBMS, set in a modern computer operating environment, much of the development effort has been put into both resilience to failure and recovery from failure. Where data is shared between users on a single computer, it is obviously vital that both the number of failures and the effects of such failures as they occur are minimised. Where data is shared between potentially far larger numbers of users on a multi-computer system incorporating varied hardware, DBMSs and other components, the problems are proportionately greater. However, many of the techniques currently in use may be extrapolated to the distributed database situation. The closest parallel is found in virtual machine operation environments which cope with concurrent use of a variety of file handlers and DBMSs, giving shared data access to many users.

The types of failure which a distributed database system must handle range from human error by terminal users, through those of the communications and terminal sub-systems, database management, operating systems and other software sub-systems to failures of memory, processors, and other hardware devices. Many failures are to be expected on a regular basis, e.g., mistyping, occasional loss of data over long communications lines, errors in applications programs - even main store memory and discs are now becoming significant when considering the likelihood of failure. This has happened during a period which has seen their inherent reliability improve but the number of such components within a system dramatically increase. As an example, a 96Kb machine with 64Mb of disc backing store of ten years ago might now be replaced by a 16Mb machine with 18000Mb of backing store. Foreseeable failures should result in only a local loss of service, and that service should quickly be restored. Other, less expected, failures may well result in a more widespread loss of service. Nonetheless, the inherent flexibility of a multi-node system should enable the effect to be kept to one node.

In a distributed database system, the types of failure which are most difficult to handle are those which occur during the life of a transaction which includes update processing on more than one node. A subsequent recovery process has to ensure that data at each node is returned to the state it was in immediately prior to the commencement of the transaction. This process, known as a 'multiple sub-system roll-back', relies upon the integrity of the data being maintained by some form of lock, and on each node having sufficient 'intelligence' to take part in such a co-ordinated roll-back operation. The intelligence takes the form of copies of the data, reflecting its state before the transaction; integrity locks; detailed knowledge of system activity at the time of failure; and knowledge of which other nodes or sub-systems are involved in this particular transaction.

Resilience

Resilience to failure normally comes from some form of built-in redundancy. Instead of a single component carrying out a function, two (or more) components are used - duplexing or multiplexing. Should failure occur within any one component, the function may be continued by the other. In simple systems, the failed component is replaced or repaired when the system is next quiescent. More advanced systems have the capability of recovering the failed component and restoring fully duplexed operations without closing the system. Examples of this kind of duplexed technique are duplexed disc/tape files, duplexed devices, multiple copies of software, and alternative communication routes.

However, there is no satisfactory substitute for producing systems whose probability of failure is small. This goal involves every stage from design to quality control on finished hardware and software systems. Even so, for some years to come we must continue to expect powerful DBMSs to produce errors in their less widely used facility areas. This is understandable, as a complete DBMS system may contain over a million instructions. In both hardware and software systems, it is often advisable to remain within the well-proven nucleus and not "boldly go where no man has gone before".

Probably the most important resilience feature in a distributed database is the provision of alternative routing methods to nodes, closely followed by the replication of data.

Some systems currently in their experimental stage are working on the premise that components such as processors will have occasional intermittent failures. A set of identical processors could each carry out the same elementary function in parallel, the most frequently occurring result being taken. This approach enables manufacturers to use cheaper, individually less reliable, components to give a system with inherently very reliable characteristics. Similar approaches may well be useful in distributed database, where several nodes may be able to process any one transaction.

Recovery

All recovery systems rely on taking preventative measures in anticipation of failure. The simplest method is the taking of backup copies of data, so that in the event of failure, the data is restored and all work subsequent to the copy is re-processed. Another technique is the recording of before-images of data as part of the update process. When a failure occurs, the database is rolled back, i.e. the recorded before-images are re-applied, in reverse sequence, until the database has once more become consistent. Subsequent transactions can then be re-actioned. This process would usually only involve those transactions which were being actioned at the moment of failure. The converse of this method is the recording of data after-images as part of the update process. After a failure, an earlier copy of the database is re-instated and then rolled forward, by the application of transaction after-images since the copy, until consistency is threatened. From this point, subsequent transactions are reprocessed. This approach, again, would usually be expected to involve only those transactions which were in process at the time of the failure.

A more complex variation on this method is the use of delayed updates. Changes are stored until the completion of the causative transaction, then copied to an after-image file and to the database. Recovery is again effected by the roll forward technique. A combined before-image/after-image technique is of particular importance in distributed situations where a single transaction may update several separately located database components. On failure, transactions in process would be rolled back, but the failure may have occurred when the after-images generated by a transaction have been completed but have not all been applied. A roll forward action is then necessary to restore a current, consistent situation. This technique would be based on a two-phase commit protocol.

In a distributed situation, each node should have a DBMS which is capable of both the roll back and roll forward functions, irrespective of the particular mechanisms implemented. A two-phase commit strategy or similar technique (usually under the control of the node initiating the transaction) is then used to co-ordinate the sub-systems. Each node must notify the controlling node when after-images have been secured to its log file. When all have done so, the transaction is deemed complete (known as the 'commitment point'); each node is then asked to commit the update to its database. When this task is complete, locks are released, and the next transaction can be processed.

Failures prior to the commitment point are recovered by a roll back operation, and those after this by a roll forward operation. This well-proven technique has one major drawback. Each transaction can involve a very large number of short messages to co-ordinate the system in case of failure, and these can result in unacceptable performance.

(f) Performance, Re-organisation and Restructuring

Performance

With existing technology, performance in a distributed database could be unacceptable, particularly in a system having a high proportion of transactions which update data on more than one node. Components which contribute to performance include all sub-systems invoked to complete the job in hand, as well as any ancillary processes triggered by the job, such as recovery or the gathering of statistics. Many factors which are usually important become insignificant, where response-time is concerned, when compared with network transfer time and inter-site locking overheads.

To determine the significance of a component to the job in question, it is necessary to know the time it is expected to take to complete its function, the likely queuing time, and the number of times it is used within the job. Potential bottlenecks can then be identified and removed, until response time is acceptable. Commonly identified bottlenecks include network transfer time; resource locking (i.e. database locks across the network); recovery actions to cope with locking and other problems; sub-system co-ordination activities (e.g., the handshakes required within the two-phase commit strategy); updating replicated data on other sites (which probably exists as backup or to improve performance when data is subject only to enquiries); and dictionary/directory accesses (where the dictionary used to control remote data access is not held locally).

Normal considerations, such as central processor utilisation and disc transfers, tend to take an important but secondary place. Until inter-site communication performance improves by at least an order of magnitude, such considerations as encryption, protocol conversion (both communication and data aspects) and other inter-node management functions should aim to ensure that the task is performed adequately, rather than putting excess effort into tuning for performance.

Techniques to optimise performance include those used by state of the art conventional DBMSs and communications techniques, such as replication of data which is used frequently for retrieval purposes only; clustering of data on frequently used access paths; indexes; data pre-loaded into main store (or virtual store) and parallel processing. The techniques of most value will be those which minimise utilisation of inter-site communication devices (such as parallel lines and message compression) and those which reduce the effect of resource locking (such as careful data design and application access sequences). Some of the techniques will reflect the type of data manipulation language in use. Those which relate to groups of records, or to tables, will offer far superior performance to those which reference record occurrences only one at a time.

Since performance could well be the factor which makes or breaks a distributed database, designers must be able to predict the performance of such a system with reasonable accuracy, so that users can be told what to expect. This is no easy task. Most suppliers have performance figures which vary between poor and barely adequate, unless backed up by intensive use of sizing specialists. The ideal situation would be one where the results of data and function analysis may be used to generate an optimum distribution design and prediction of the resultant performance within the business constraints. Given performance prediction, the next requirement is for the variety of alternative storage and access techniques expected from comprehensive DBMSs, to permit tuning of the system to the required response time, throughput and resource utilisation.

Most current sub-systems have built-in monitoring facilities to give statistics such as number of transfers, average resource utilisation, and delays awaiting resources. It should be possible to relate the details from all sub-systems, to give a total picture and identify bottlenecks in a manner which relates to the original performance predictions. The system can then be re-sized and re-tuned to achieve an improved compromise.

There are systems in operation which re-tune certain aspects of the system in an heuristic manner. One such system, for example, dynamically re-locates data from one node to another depending upon recent and predicted usage. Suppliers of distributed database systems may well be advised to design their initial offerings in such a manner as at least to collect the data necessary for administrative action or the later addition of heuristic tuning.

Re-Organisation

State of the art database technology has the concept of a conceptual schema and an internal schema. A conceptual schema is the one to which an application program would relate (via an external schema), and typically defines records (or tables), sets, data items and symbolic keys. An internal schema, however, relates to methods of data storage (hashed, clustered, sequential, etc.,) and to access methods (through pointers and indexes). Changes to either storage or access methods are the subject of re-organisation activity, which may be triggered by poor performance or database full conditions.

Re-organisations within a distributed database fall into two categories, those within the scope of a single node, and those which affect more than one node. In a single node, the problems are basically no different from those of re-organising a centralised database. Such a re-organisation would normally involve logically unloading the data, sorting it into a sequence dictated by the loading characteristics of the new internal schema, and then reloading it. One would normally plan such an activity carefully, to reduce the impact on users and ensure the integrity of the data. A medium-sized database might well take a long weekend to re-organise, so performance predictions are again important, particularly as a database may well have large numbers of users who could be disrupted. It is often possible to copy data to be re-organised and use the copy in 'retrieval only' mode to supply a limited service whilst re-organisation takes place.

Re-organising data over more than one node may be done for a variety of reasons. It may be decided to replicate frequently used data onto another site, which need present no particular problems apart from that of bulk data transfer. It may be necessary to disseminate data from one node across several, to remove a bottleneck or cater for the node having reached its maximum capacity. These and other inter-node problems need new tools.

Restructuring

Often, re-organisation and restructuring are confused, as some suppliers find it convenient to provide utilities which combine them. Restructuring involves changes to data which are needed to reflect changes to the conceptual schema, and often require changes to applications programs. Examples include such relatively simple situations as adding new record types, tables, sets, or data items, and more complex ones, where, for example, a record type may be split into two or more new record types to support some new application area. Some changes can be performed in situ, but the data may have to be unloaded and manipulated prior to reloading within a new conceptual (and internal) definition. In a distributed database, the problems are very similar to those of re-organisation.

Dynamic Re-organisation and Restructuring

The scenario is discussed, in academic and international circles, of a database, distributed or centralised, with a conceptual and/or internal definition which changes relatively frequently. At each change, instead of stopping and changing all the data and programs to reflect the new definition, it is envisaged that the data would be dynamically re-organised (or restructured) when next accessed. Programs could similarly use old external views to access data which is now held in a manner reflecting new definitions. The papers currently available suggest that the subject is not adequately understood at this time to consider within the already complex situation of a distributed database.

CHAPTER FIVE

SPECIAL REQUIREMENTS AND FUTURE DIRECTIONS

General

The preceding chapters have discussed distributed database systems as currently implemented or under development. This chapter addresses factors which may result in special solutions being quickly developed or which may affect longer term trends. Such factors will primarily be developments in technology, or user requirements which cannot be satisfied by current solutions.

Developments in technology and changes in user requirements are necessarily linked. As computer hardware becomes cheaper and more powerful, and as digital networks become more readily available, user expectations will increase. As the general awareness of users regarding the capability of computers increases (through personal computers, computer education, and so on), new and more diverse applications will be considered.

This chapter moves from special user requirements to developments in computer technology. The user requirements section does not attempt to identify requirements of relevance only to a single application, but rather those which are likely to have a wider influence on future developments.

Special User Requirements

Broader, unconventional applications of computer technology are presently being explored. The growth of DBMSs has been, and that of distributed database systems is being, based on a range of typical, proven requirements which tend to correspond to conventional data processing situations. However, some of the areas into which there is a pressure for expansion have clear implications for distributed database technology.

Much is heard of the electronic office. A distributed database could well provide an inter-site link for an organisation's word processor based document system, with immediate access and retrieval facilities available to authorised sites whenever a new document is collated or an earlier one updated by word processing techniques.

The use of facsimile and graphical input would also gain value from the use of distributed systems, such probable usage having implications for the need of development of high bandwidth communications systems and larger capacity storage devices to cope with the bit-streams generated. The same techniques would facilitate the storage, distribution and retrieval of speech and pictures.

Where a distributed system supports a diversity of users, there will be pressure to permit on-line additions and alterations to its schema. Particular integrity problems arise from a need to define and immediately utilise a new record type. Again, simultaneous re-organisation may be impractical across a distributed system. Such a situation could arise from differences of time zone, for example, within the organisation. The solution here may be for the distributed database management system to recognise more than one simultaneous version of the schema, some data being stored according to each version.

In process control applications, although relatively small quantities of stored data may be involved, changes to data occurring on one computer may have to be distributed rapidly to other computers in the system. Further, the fact that a change has occurred to the data may have to be signalled to an application-level program in the recipient computer.

Government legislation or other pressures from society may affect requirements upon distributed database systems. Requirements for protection against unauthorised access could especially affect the design of communications facilities. For instance, legal requirements for individuals to be able to find out what data is stored about them, and to record objections or amendments, may well be enacted within a country. Also, privacy laws may result in restrictions on data transfer across national boundaries, and distributed database systems affected may be required to guarantee compliance.

Military requirements are likely to have some impact on the direction of developments. Much basic research on distributed databases in the U.S.A. is funded from military sources. Whilst much of this is general in nature, military requirements must inevitably be taken into account. In the forefront of these are surviveability and interoperability. Surviveability is the requirement to continue to provide a service following the destruction or loss of part of the total system. Research in this area will yield benefits for availability and resilience in commercial systems. Interoperability is the requirement to exchange or share data with other systems. Research in this area is likely to improve understanding of the problems of heterogeneous, federated systems.

A requirement to make best use of hardware resources always exists. The requirement is greatest for the resource which is relatively most expensive. With the cost of memory and processing units falling rapidly, communications are likely to be the most expensive resource in many situations. Optimising the use of communications will lead to the need for query optimisation, for optimising the number of replications and the locations of data in a network, and for efficient communications protocols.

Developments in Computer Technology

The cost and availability of computer hardware is the fundamental factor influencing the development of distributed databases. The increasing cheapness of computers and storage devices has led to the widespread introduction of computers into organisations, often in a haphazard and poorly co-ordinated manner. Even small departments can now afford to buy their own computers. The result is that corporate data is spread across a number of separate computers, often of different types. Since the data is often logically related, and of interest to more than one department, this inevitably leads to pressures for exchange or sharing of data.

In the short term, cheap solutions which involve copying entire files from computer to computer and reformatting on an ad hoc basis are adopted. As the problems of inefficiency and inconsistency become apparent, the pressures for a better solution will grow and heterogeneous distributed database software will be required.

This situation has interesting parallels with the development of database management systems. Initially, individual applications were allowed to hold their own files independently, but within a single computer. However, the problems outlined above also occurred in this case, with the resultant requirement for databases.

Whilst the cost of hardware in general continues to fall, some items of equipment will always be relatively more expensive than others. Although departments may be able to afford their own computers, some items of equipment will always be too expensive for single installations to justify. Such equipment might well include mass storage devices, content addressable file stores, large database machines, and so on. Given the low cost of 16 and 32 bit micro-computers which will soon be powerful enough to perform functions originally associated with mainframes, even items such as tape drives could become relatively too expensive for some users. In this situation, there is a clear case for sharing any type of data storage device between computers, and these should be under the control of the distributed database management system. Future distributed database management systems will therefore be required to cater for a range of device types and capabilities within a network.

Communications is a critical factor in the development of distributed databases. Protocols for ensuring data integrity generate a high message load for a communications system. Where fast response times are required, a communications system will be required to deliver messages very quickly. High speed local area networks are becoming available, which should be able to meet many requirements, but the capacity of distributed database systems to saturate networks should not be under-estimated. For databases distributed across geographically dispersed sites, data communications is much more likely to prove a limiting factor. The difference in communications capacity between local area networks and long haul networks is likely to impose a requirement on the distributed database management system for two levels of integrity protocol. The long range protocol might provide lower integrity for a lower cost in message traffic, or might complete updates remotely following transaction completion at the initiating site.

The two levels of protocol could also create a requirement for two levels of distributed database. A local distributed database would span the local area network, and also form a single node within a long range distributed database. From the viewpoint of the long range distributed database, the local distribution would normally be irrelevant, and should not be visible.

A significant factor in the ease of implementation of heterogeneous databases is likely to be the International Standards Organisation's Basic Reference Model for Open Systems Interconnection. Whilst standardisation of communications will be beneficial, it is not clear that the current ISO model is entirely consistent with distributed database requirements at the upper (application, presentation and session) levels.

CHAPTER SIX

CONCLUSIONS

This initial report has outlined some of the background to the subject, indicating where there is a need for a distributed database and which kinds of data and processing requirements are best suited to the distributed database approach. It presented various types of solution, in ascending order of complexity. It then reviewed the methods used to achieve the solutions, and considered some of the technical issues involved.

The distributed database approach is a result of the convergence of the earlier disciplines of database and data communications. At the time of writing this report, there are relatively few examples of either customised solutions or standard products which can vouch for its future potential. Hence it is reasonable to ask whether DDB is mature enough to be considered practical now. Is it important for the future of data processing, and when (if ever) will it come into widespread usage? How will it develop in such directions as ease of implementation and efficient operation?

What forms will that development take? Much progress has already been made in the area of standard components, such as those for data access between machines irrespective of geographic separation. This progress will yield a range of distributed database management systems (DDBMS) and supporting software. The earlier examples are naturally limited in their handling of recovery from failures, and in their ability to operate across different machine regimes, but some progress in these areas also is expected. Standards for DDB are unlikely to appear in a usable form for several years yet. Instead, compatibility will initially be achieved through a combination of similar data management regimes, plus data communications standards. Since the latter are usually employed at a low level (e.g., ISO layers 1 to 4), and data management applies at a middle level (records, tuples), the upper (application) levels will still have to be supplied by the user organisation in most cases. Some suppliers will adapt their applications packages to enable multiple instances of those packages to co-operate between computer sites. Even then, some constraints will still apply in terms of performance and ease of control. One important phenomenon which will impact DDB performance, and shape its development, is the availability of high capacity data links (i.e. 1 Mbit/second upwards).

There are significant pitfalls along the path of DDB implementation, and this report has explained several of these, together with approaches for avoiding them. Perhaps the most noteworthy difficulties are those affecting user authority and procedures, software construction, performance and exception handling. The report has therefore emphasised the care and planning effort required to ensure success whatever the form of solution.

There are also opportunities - to co-ordinate business operations more closely between sites, to reduce the costs of several kinds of communication, and to decrease dependence on particular ADP resources. Distributed databases are emerging from the experimental stage into the practical stage. There is a growing awareness on the part of government and commercial organisations that fragmented data processing solutions are inadequate, and that a coherent approach is required in future. The Working Group is confident that in the next few years the state of the art will advance rapidly, the efficacy of distributed databases will be demonstrated, standards will be proposed, and a variety of software tools for distributed data management will become widely available. As these changes occur, distributed database methodology will become an integral aspect of most organisations' data processing activities. Naturally, each organisation will proceed at its own pace. Some organisations are already constructing or operating their own DDB systems. The Group hopes that this initial report will be of value to those who are considering whether to do likewise.

APPENDIX A

GLOSSARY OF TERMS

ACCURACY A database is accurate if its user data is correct, complete and timely in relation to the real world.

BLACK BOX A unit whose inputs and outputs are well-defined but whose internal mechanisms are not known.

CODASYL Conference On Data Systems Languages - a voluntary organisation with the objective of designing and developing techniques and languages to assist in data systems analysis, design and implementation.

COMMIT PROTOCOL A means of ensuring that a given update is applied fully, or not at all.

COMMITMENT UNIT A quantity of processing by an application program to take the database from one state of integrity to another.

CONCEPTUAL SCHEMA Defines the overall logical structure of a database independently of any particular application view, or of the storage mechanisms used.

CONCURRENCY CONTROL Mechanism which ensures that all changes made to a database during a transaction's processing are not visible to other concurrent transactions until the transaction's changes are committed.

CONNECTIVITY The extent to which individual data elements in a database are bound together.

CONSISTENCY A database is consistent if its user data and system data are in accordance with the rules and descriptions (i.e. consistency constraints) contained in its global schema and distribution schema.

CONTROL Control is centralised if the control information is held at one node only. Control is decentralised when the information is held at each node.

DBMS see database management system.

DDB see distributed database.

DDBMS see distributed database management system.

DML see data manipulation language.

DATABASE ADMINISTRATION The responsibility for maintaining the database within an organisation.

DATABASE MANAGEMENT SYSTEM The software which manages a database.

DATA DICTIONARY A dictionary describing the data stored in a database, and containing all relevant information about that data.

DATA INDEPENDENCE The independence of different views of the data in a database, allowing one view to be changed without disturbing the other views.

DATA INTEGRITY The protection of the data in a database against loss or damage resulting from failure during operation of the system.

DATA MANIPULATION LANGUAGE The interface language used by the application programmer to communicate with the database from the host language.

DEADLOCK A problem which can arise whenever two or more contending processes wish to exercise exclusive control over common resources.

DETERMINISTIC ROUTING The determination of a route according to a set of rules.

DISTRIBUTED DATABASE A single database whose parts are distributed across multiple computers connected by a communications system.

DISTRIBUTED DATABASE MANAGEMENT SYSTEM A generalised utility for servicing the requests of application programs for global external objects - and hence ultimately for storage schema objects at some local site(s).

DISTRIBUTION SCHEMA A definition of the location of global schema objects over the sites of a distributed database.

EXTERNAL INTERFACE A capability to transfer data and/or commands across a boundary of the system (e.g., to/from a machine or other network).

EXTERNAL SCHEMA Sum definition of all of the various types of external record in the external view of the database (i.e. the content of the database as it is seen by a particular user).

GLOBAL Affecting the whole system.

GLOBAL ACCESS CONTROL SCHEMA A description of the global privacy constraints.

GLOBAL EXTERNAL SCHEMA An application-orientated view of the global schema for use by one or more application programs.

GLOBAL RELATIONSHIP A logical association between data types whose instances may reside at multiple nodes.

GLOBAL SCHEMA A description of the logical structure of the totality of data comprising a distributed database.

HANDSHAKE A pair of messages, where the second message is a direct response or acknowledgement from the recipient to the sender of the first.

HETEROGENEOUS A heterogeneous system accommodates different DBMSs at its nodes, irrespective of whether the schema content and hardware are the same or different.

HOMOGENEOUS A homogeneous system uses the same DBMS at all its nodes, irrespective of whether the schema content and hardware are the same or different.

ISO see International Standards Organisation.

INCONSISTENCY A database is inconsistent if two or more pieces of data in that database simultaneously hold values that cannot be mutually reconciled. A special case of inconsistency occurs when two copies of the same data item have different values.

INTEGRITY Integrity of a database implies that all data values are correct in respect to the transactions processed. A database may not be up-to-date, but when all transactions have been processed, the database will accurately reflect the real world situation.

INTERNAL SCHEMA A definition of the way in which data is stored in a database. It describes the internal organisation of the data, rather than the view of any particular user.

INTERNATIONAL STANDARDS ORGANISATION An organisation which co-ordinates the formulation and ratification of standards in a wide range of areas, including computing.

MAPPING A mathematical term for a set of rules by which the representation of a group of objects can be changed to another form. Thus, centigrade temperatures may be mapped onto the fahrenheit scale - or a relational data model may be mapped onto an entity/set data model - in both cases with neither loss nor gain of information.

NETWORK Two or more computers connected together by communication lines.

NODE A place in a network to and from which messages can be sent. At each node there is usually a simple hardware device to handle the messages. To this device can be connected one or more computers (or users) that originate and receive messages.

ON-LINE An activity carried out whilst its originator waits for its completion.

PACKAGE A group of computer programs written to carry out a particular function, e.g. payroll, and intended to form a 'standard problem solution' - obviating the need for each user to derive his own programs.

PACKET-SWITCHING A technique in which a message is broken up into a series of pieces - 'packets' - which are then transmitted to the recipient independently over a network and re-assembled at the receiving site. The process of message fragmentation and re-assembly is internal to the packet-switching system and is invisible to both sender and recipient.

PARTITIONED DATA Partitioning is the division of data in a file into a series of pieces. There are two ways of doing this: first, by data value - for example A-D, E-K, L-R, S-Z in separate files: second, by data type - for example names and addresses on one file and salary information on another. Combinations of the two techniques are possible.

PORT The physical socket on a computer to which a communications line can be connected.

PROBABILISTIC ROUTING Dynamic determination of the 'best' next step of the route, based on accumulated statistics.

PROGRAM GENERATOR A program which, from information provided in prescribed form by a user, can create, wholly or partially, another program.

PROTOCOL A precisely defined set of rules governing the way in which information can be transferred between two parts of a computer system, e.g., two computers in a network or two collaborating programs.

REAL WORLD The relevant parts of the environment within which we exist. The actual existence of those facts, or pieces of data, which are to be recorded and modelled within the database - for example, a warehouse and its contents exist in the 'real world', while a database may record 'location' and 'stock balance'.

REPLICATED DATA Data held at more than one place in a database, either centralised or distributed.

RESILIENCE The ability of a system to recover from a fault without loss of any information.

SCHEMA A set of rules which defines the structure of a database, or of some particular aspect of it.

TWF see transaction wait for graph.

TRANSACTION A discrete instruction issued by a user, e.g. add new customer, 'tell me how many we have in stock?'.

TRANSACTION WAIT FOR GRAPH A technique used to ensure that two transactions being processed simultaneously by a computer do not prevent each other from continuing, thereby stopping any further progress on either transaction. See also DEADLOCK.

TWO-PHASE COMMIT PROTOCOL A technique for ensuring that all copies of the same piece of data in a distributed database are simultaneously updated. The first phase ensures that all DBMSs have received the update; the second phase is the issuing of the authorisation to carry out the update, when all involved have acknowledged.

TWO-PHASE LOCKING The way of ensuring that concurrent access conflicts do not occur under a two-phase commit protocol. All DBMSs lock all relevant records before any DBMS commences updating; no records are released until all DBMSs have successfully completed updating.

USER FRIENDLY A characteristic of systems which have been designed to suit the convenience of the user of these systems - rather than making the user amend his way of working to suit the convenience of the system developer. Assessment of user friendliness remains highly subjective.

BIBLIOGRAPHY

M. Adiba, C. Delobel: The problems of the co-operation between different DBMS. IN: Architecture and models in data base management systems. G.M.Nijssen (Ed.); North-Holland Pub. Co., 1977, pp. 165-186.

M. Adiba, J. C. Chupin, R. Demolombe, G. Gardarin, J. le Bihan: Issues in distributed data base management systems: a technical overview. IN: Proc. 4th Int. Conf. on very large data bases, West Berlin, Germany, Sept. 13-15, 1978, pp. 89-110. (Also IN: Issues in data base management. H.Weber & A.I.Wasserman (Eds.); North-Holland Pub. Co., 1979, pp. 127-153).

BCS: The British Computer Society Data Dictionary Systems Working Party Report. IN: SIGMOD Record, v.9, no.4, pp. 2-24.

A. F. Cardenas, M. H. Pirahesh: Database communication in a heterogeneous data base management system network. IN: Information Systems, v.5, pp. 55-79.

G. A. Champine, R. D. Coop, R. C. Heinselman: Distributed computer systems: impact on management, design and analysis. North-Holland Pub. Co., 1980.

C. J. Date: An introduction to database systems (3rd Ed.). Addison-Wesley, 1981.

R. A. Davenport: Design of distributed database systems. IN: The Computer Journal, v.24, no.1, pp. 31-41.

C. Delobel & W. Litwin (Eds.): Distributed databases. North-Holland Pub. Co., 1980.

I. W. Draffan & F. Poole (Eds.): Distributed databases. Cambridge University Press, 1980.

J. N. Gray: A discussion of distributed systems. IBM Research Report RJ2699, 1979.

ISO/OSI: Data processing - open systems interconnection - basic reference model (ISO/TC97/SC16). IN: Computer Networks, v.5, no.2, pp. 81-118.

B. G. Lindsay, P. G. Selinger, C. Galtieri, J. N. Gray, R. A. Lorie, T. G. Price, F. Putzolu, B. W. Wade: Notes on distributed databases. IBM Research Report RJ2571, 1979.

H. Lorin: Aspects of distributed computer systems. Wiley, 1980.

E. I. Lowenthal: A survey - the application of data base management computers in distributed systems. IN: Proc. of 3rd Int. Conf. on very large data bases. IEEE, 1977, pp. 85-92.

J. Martin: Design and strategy for distributed data processing. Prentice-Hall, 1981.

F. J. Maryanski: A survey of developments in distributed data base management systems. IN: Computer, Feb. 1978, pp. 28-38.

M. Miller: A survey of distributed data base management. IN: Information and Management, v.1, 1978, pp. 243-264.

National Computing Centre: Trends in distributed computing systems. NCC Publications, 1978.

PACTEL (PA Computers and Telecommunications): Distributed database technology. NCC Publications, 1979.

T. A. Rullo (Ed.): Advances in distributed processing management, v.1. Heyden, 1980.

B. Yormark: The ANSI/X3/SPARC/SGDBMS architecture. IN: The ANSI/SPARC DBMS Model. D.A.Jardine (Ed.). North-Holland Pub. Co., 1977.