DATA SHARING IN DISTRIBUTED COMPUTING SYSTEMS

Fabio Schreiber,
Università di Parma
Parma, Italy

In this chapter the different ways of sharing data through computer networks are reviewed, and attention is focused on distributed databases. The distribution features of a DDB are examined with respect to the application needs. Possible architectures for a DDB management system are considered and a quick mention is made of the problem of keeping data integrity.

1. INTRODUCTION

The availability of commercially operated computer networks has brought to an outstanding level the importance of software shareability and transferability among many different users, working on different computers. In fact, even if the efforts of achieving program portability from one machine to another are as old as programming itself, they have not achieved, to date, highly satisfactory results and everybody has undergone the frustrating experience of transferring a standard Fortran routine from one machine to a different one having to recode more or less extended parts of it. In a computer network, where software sharing by hundreds of users is standard and fundamental, the capability of a highly efficient and reliable transportability becomes vital; the possibility of on-line operation makes the request for efficiency much more dramatic than when sharing occurs by sending magnetic tapes around the world.

While other parts of this volume focus on the problem of program sharing, in this chapter we shall point out the major issues in sharing data among a community of users and we shall show, in particular, the architectures and the functions of Distributed Database Management Systems (DDBMS).

It is worth mentioning that, while the sharing of programs is mainly achieved by means of techniques which enhance their portability, the sharing of data is mainly achieved by techniques which access them where they are "naturally" stored. The rationale for that lies in the very large volume of many databases which makes it very expensive, if not impossible, to move them around the world.

Therefore, since the beginning of the distributed computing systems era, data have been considered one of the most interesting resources to be shared in the system, and a considerable number of studies have been made and prototype systems for distributed data management have been implemented in recent years. The reasons claimed for such an interest are many; among them the following can be mentioned:

— *Reliability*: a strongly centralized system is highly vulnerable to damage caused by hardware and/or software failures or by inadvertancy. Once damaged, operation is denied to everybody. On the other hand, a distributed system, even if suffering from failures, undergoes a graceful degradation, still allowing users to operate on it with a reduced functionality. Conversely recovery procedures, to reestablish data integrity in a distributed system after a failure, are more complex, as we shall see later on.

— *Load sharing*: computing centers having complementary utilization profiles can be connected into a distributed system in such a way as to better utilize existing resources by sharing the total load among them. Even the time difference between two computing centers placed far away from each other could be a reason for sharing computing power among similar loads but with peaks occurring at different "system absolute time".

— *Resource sharing*: one of the most popular reason claimed for distributed systems is the possibility of using very specialized resources ranging from entire computers (e.g. ILLIAC IV array computer), to special terminals (e.g. graphical plotters or displays, robot arms, etc) to specialized software (e.g. special compilers, application programs, etc), to common data (i.e. distributed databases), which are held in a common pool.

— *Efficiency*: handling very large masses of centralized data can result in a loss of efficiency of the system. For example, the unloading and reloading of a database to perform physical reorganization and maintenance can be a very long operation resulting in reduced availability. Also response times can be severely affected if the database grows above a critical size. In a distributed system these deficiencies can be overcome by a suitable distribution of data among the different machines allowing the optimization of several parameters.

— *Efficacy*: a distributed philosophy naturally allows the decomposition of a complex multifunction system into a set of simpler interconnected systems, each of them being specifically designed to perform at the best cost/performance level. Also a better service to the user is claimed for distributed systems, even if this factor cannot be easily defined and quantified.

— *Flexibility*: a distributed system, by its own nature, can grow and modify its structure without major restructuring efforts. Often, components of the system can be added or taken away only by modifying some system tables without disturbing the on-going operations. This fact allows a "physiological" fitting to the needs of the organization the information system works for.

— *Human engineering*: a very strong centralization of EDP services has proved harmful to the psychological attitudes of the employees and of the middle management; lots of misunderstandings arose between the end users and the system analysts placed far apart from each other, and in many cases they led to the rejection of the computing services by the end users themselves. Also, the sense of power associated with the possession of data contributed to the success of a decentralized philosophy for the EDP systems and data management.

— *Economy*: some authors claim also economical reasons in favour of distributed information systems. However, too little experience has been gained up to now on the economical dynamics of distributed systems to formulate conclusions. We can only point out that while the hardware costs are decreasing, the costs of software tend to rise (or to remain constant, at best) penalizing the considerably higher amount of complexity involved in the system software to control distributed systems. On the other hand, the costs of telecommunication facilities, the third component of distributed systems, are heavily affected by political factors, which mainly dictate the leasing tariffs.

2. DATA SHARING IN THE DISTRIBUTED ENVIRONMENT

When speaking of distributed computing systems we mean a collection of computing devices connected through telecommunication facilites. However the connection can be made in many different ways, giving rise to very different families of systems. In particular we shall mention three main classes:

a) Tightly coupled systems
This class includes multiprocessor devices which communicate through one or more shared buses and/or one or more shared memories.

b) Loosely coupled/high bandwith systems
This class includes those systems constituted by many processors connected through serial high-bandwith telecommunication lines such as coaxial cables or optical fibers. They are usually called *Local Computer Networks.*

c) Loosely coupled/low bandwith systems
This class includes those systems constituted by many processors connected through serial low-bandwith telecommunication lines such as the telephone or the data networks. They are usually called *Geographical Computer Networks.*

The system architecture heavily influences the structure of the data management system. Systems of type a) can be usefully used in building database machines, i.e. special purpose computers entirely devoted to the management of large quantities of data. Systems of type b) tend to be used in connection with a distribution-by-function philosophy. In this case the machine devoted to data management is called a back-end computer, even if this name is not peculiar to such architectures. The back-end computer can be a general purpose machine on which the database management system runs as the only function, or it can be a database machine.

Finally, in systems of b) and c), data can be stored at different nodes in the form of local databases, which can be accessed by any other node through the computer network.

In the following we are going to examine this solution in more detail. Fig. 1 shows a computer communication network with several host (nodes) connected to it.

In such a structure many different ways of sharing data are possible, and have been described in the technical literature. In this section, we are going to mention very briefly those which are the most typical. Then, in the rest of the chapter, we shall focus our attention on the most evolved: the Distributed Database (DDB).

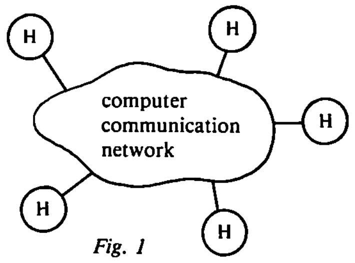

Fig. 1

Independent Databases Connected to a Computer Network

The development of commercial computer networks like ARPANET, DATAPAC, EURONET, INFONET, TELENET, TRANSPAC, and many others, allows network users to access remotely databases, existing on host computers connected to the network itself. These databases, however, are completely independent both as to their contents and as to their management systems (Fig. 2).

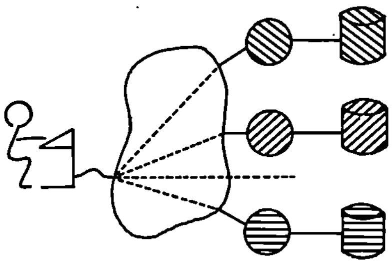

Fig. 2

What differentiates this system from a classical centralized teleprocessing (TP) system is the fact that the computer network allows a homogeneous connection with many different machines and, hopefully, lower operational costs.

Independent Databases with a Standard Command Language

This kind of system is a direct evolution of those described above. In this case it is still the task of the user to connect himself to the appropriate host computer and to ask it for access to the database he wants to interact with. However, the user is now freed from the need of knowing the details of the database itself and of its management system since he is provided with a standard command language which interfaces all the — possibly heterogeneous — local data management systems (Fig. 3).

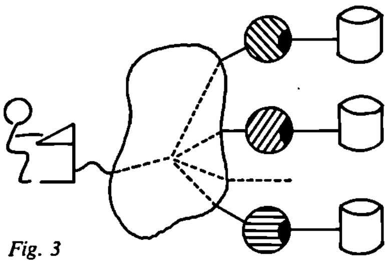

Fig. 3

Such a kind of system is represented by the DIANE service, which is offered by EURONET, interfacing as many as 300 Databases stored at 36 different Hosts (April 1982).

Distributed Database

For many years, database has been a synonym for a highly centralized data organization. This is not in contradiction with a physical partitioning of data among many different sites. In fact, what distinguishes a distributed database from the systems described in the preceeding sections, is that the data, even if stored at different sites, all belong to the same universe and are logically related with each other. Therefore the user is given only a global knowledge either of the whole database (network or global conceptual schema) or of part the database relevant to his application (global external schema), and an integrated access method which makes the physical data location transparent to him. Moreover, the integrity and consistency of the whole database must be preserved at all sites, in spite of possibly partially executed transactions aborted while operating on several sites simultaneously (Fig. 4).

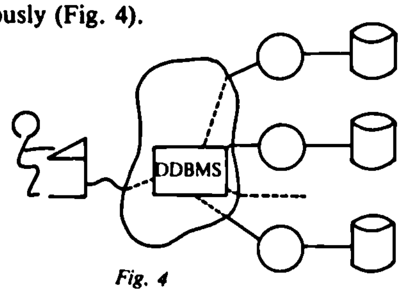

Fig. 4

3. DISTRIBUTION FEATURES IN A DDB

Necessary for the existence of a DDB are the distribution of the processing power (i.e., the existence of a computer network) and a physical distribution of data. However a third parameter, the *control* over the DDB, further partitions the DDB systems into two major classes:

— *Hierarchical DDB's*, when a master node exists in the system which controls the flow of the distributed operations in the whole DDB - exceptions being made for those operations belonging to a transaction which can be completely processed at a single site (local transactions). The control is kept over the update operations (e.g. for integrity, consistency, etc.) the locking conditions (e.g. for deadlock prevention and resolution), the recovery operations, etc. Even if such a control philosophy has many limitations, because the master node can easily become a bottleneck, it can be usefully used, taking advantage of the greater implementation simplicity, in those information systems which are hierarchical by their very nature or which are built on a hierarchical (star or tree-shaped) computer network.

— *Distributed control DDB's*, when the control over the DDB is distributed among the different nodes the system and is performed through an exchange of messages and parameters among them. The obvious advantages of the distribution are now balanced by a considerably higher complexity of the DDBMS software and, in general, by a lower efficiency.

As to data, they can be distributed in many different ways. In the simplest cases the distribution is made at the *file level*, that is the logical file is also the unit of physical distribution. However, in many applications it is convenient to consider a partitioning below the file level. In this case we can consider the logical file composed of many records subdivided into fields, and we can consider its partitioning into physical fragments in one, or in a combination, of the following two modes:

— *Horizontal partitioning*: all the fragments have the same logical structure (i.e., the same sequence of fields); records belong to a fragment according to some distribution criteria. This corresponds to making a selection on a field value which is characteristic of a particular node.

— *Vertical partitioning*: the structure of the distributed fragments is derived from the structure of the original file by subsetting (or projection) operations, i.e. by storing only some fields at each node. Each logical record is then fragmented into its vertical partitions and then stored in the DDB.

Until now we have not considered the possibility that data be stored in multiple redundant copies at different sites. Historically, this was one of the first forms of DDB to have been considered and the reasons in favour of data replication lie in an increased reliability, shorter response times to enquiries, and cost optimization. However the existence of multiple copies of a data item poses some problems as to the consistency of the different copies when updating operations occur, i.e. how much and how long the contents of the different copies can be different. Some applications ask for a very *tight consistency*, that is the contents of every copy must exactly the same at every moment in such a way as a query could not distinguish from which copy the answer came; other applications can require only a *weak consistency*, in the sense that only one copy of the data item is updated by a transaction, while all others are updated in a deferred mode by the DDBMS. More precisely, we can consider the DDB under the following aspects:

a) *Permanence*: data items can be *permanently distributed*, i.e. once loaded they have a life of their own, or *temporarily distributed*, i.e. a master data item exists and replicated copies are distributed when and where they are needed; possibly they must be refreshed from the master copy to keep consistency.

b) *Consistency*: the copies of a replicated data item are *fully dependent* on each other when tight consistency is required; they are *partially dependent* when weak consistency is sufficient; they are independent where updates on one copy do not affect the other copies of the data item. It must be noticed that independence does not necessarily imply the lack of interrelations among the different copies; since it is often used together with temporary distribution, consistency is kept only at the level of the master copy and hence by refreshment of the other copies.

Often the choice among the different features is dictated by the peculiarities of the applications the system is built for. In many cases queries are real distributed process, while updates are made only on local data. Also, the mode of operation — batch, real time, etc. — can influence the distribution feature of the DDB system.

4. ARCHITECTURES FOR DDBMS

Gross Architecture

In designing a DDBMS the first step is to define its gross architecture. In the field of the centralized DBMS a considerable popularity has been gained by the ANSI/SPARC model, which should allow a maximum of physical and logical data independence. It defines three layers, called schemata:

- The *Conceptual Schema* which represents the data belonging to the whole organization, as seen by the enterprise administrator;
- The *External Schemata* each of which represents the "view" of the organization as seen by a particular application program;
- The *Internal Schema* which represents the way data are physycally stored in the system.

When several DBMS have to cooperate in a distributed environment to constitute a DDBMS, a fourth level has to be added: the Network Schema (NS) or Global Conceptual Schema, mentioned before, which defines the data global to the whole distributed database. We shall not go into a detailed discussion of the possible different architectures arising from the level at which the NS is placed in the model, but we shall only show two of them which are the most general:

a) *The Integrated DDB* (Fig. 5). In this architecture the ownership of data stored in the local DDB belongs to the distributed computing system. All the external schemata are defined starting from the NS which represents the logical union of the logical conceptual schemata existing on the different nodes.

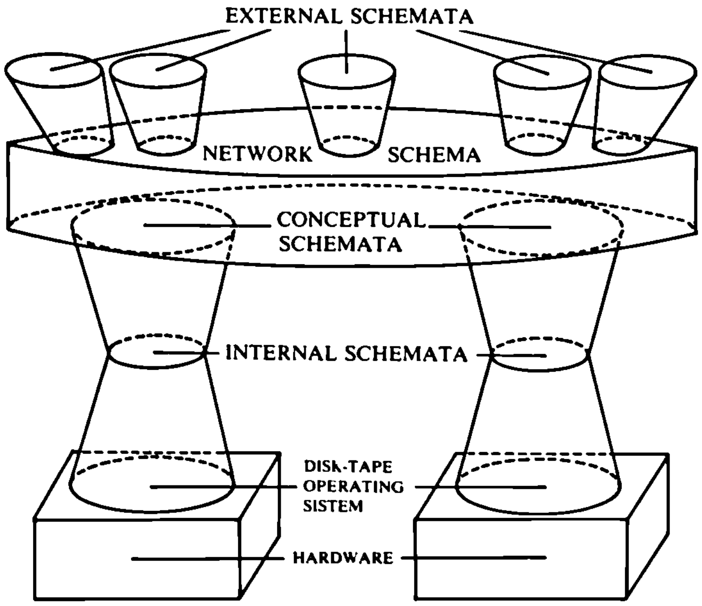

Fig. 5 - Logical Architecture of an Integrated DDB

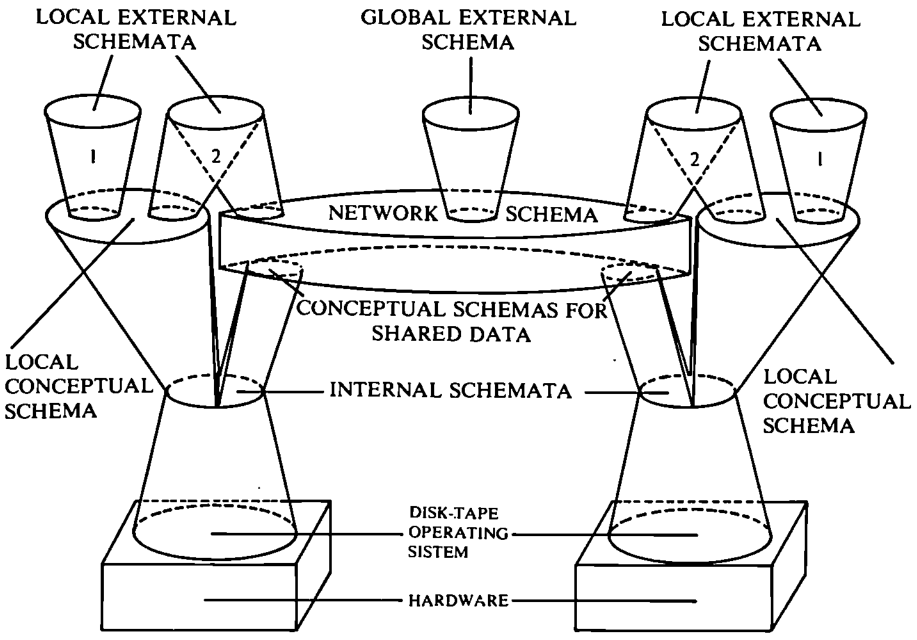

Fig. 6 - Logical Architecture for an Aggregated DDB

b) *The aggregated DDB* (Fig. 6). In this architecture the ownership of data belongs to the local system, which decides which of them are to be shared in the distributed system. In this case on top of the internal schema, which refers to physical data in a undifferentiated mode, two conceptual schemata are built, one for pure local information, the other for information which is to be shared in the network.

The integrated approach is easy to implement when a system has to be built from scratch, and even better if it is a *homogeneous* one - i.e., every local host has identical hardware, operating system, and DDBMS as all the others. The aggregated approach is well suited when the DDBMS has to be built interfacing several existing local systems, which very often are *heterogeneous* - i.e. they differ as to the hardware and/or as to the system software. However the aggregated approach can be used whenever privacy constraints are very strong and the amount of shared data is much lower than that of data which are only of local interest.

Software Architecture

Going into the inner architecture of a DDBMS, we recognize that the functions the software must perform are, generally speaking, the following:

— to create and maintain the DDB;
— to process the application transactions;
— to cope with hardware and software failures.

In the following we shall briefly discuss the mechanism of transaction processing, using the model of fig. 7. On the left hand side the logical levels are shown, in the center the evolution of the transaction and the relevant software functions and on the right hand side the additional information required to process it. We shall use a block numbering from the top to the bottom to identify each block.

Block 1

The first function performs the translation of the transaction from the external schema language, in fact in the most general case all the data models used in the different schemata can be different. The translation makes use of the mapping between the two schemata and some semantical integrity constraints can be checked at this level.

Block 2

Formal correctness and authorization constraints must be checked before going on. These operations are done with the aid of the data dictionary.

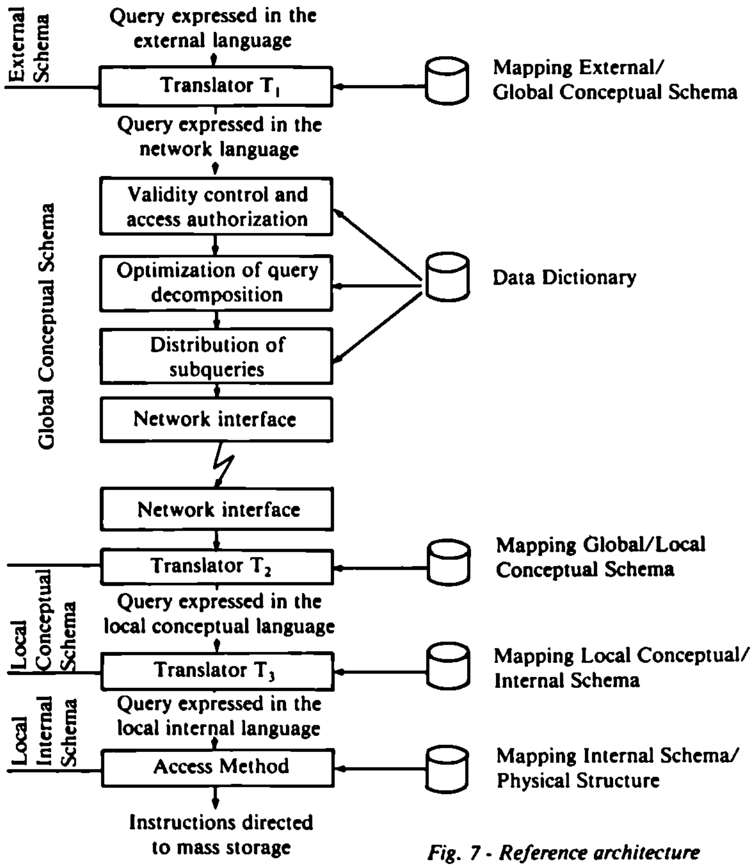
External Schema
Query expressed in the external language
Translator T_1
Mapping External/ Global Conceptual Schema
Query expressed in the network language
Global Conceptual Schema
Validity control and access authorization
Optimization of query decomposition
Distribution of subqueries
Network interface
Data Dictionary
Network interface
Translator T_2
Mapping Global/Local Conceptual Schema
Local Conceptual Schema
Query expressed in the local conceptual language
Translator T_3
Mapping Local Conceptual/ Internal Schema
Local Internal Schema
Query expressed in the local internal language
Access Method
Mapping Internal Schema/ Physical Structure
Instructions directed to mass storage

Fig. 7 - Reference architecture

Block 3

A complex transaction involving data stored at many different nodes must be decomposed into many simpler transactions. This operation is very critical and it can be done in many different ways. A lot of work has been done and is still in progress to obtain the maximum of efficiency by reducing the amount of data to be transferred through the network.

Block 4

The eiementary transactions are sent to the execution node. This phase requires resource locking and synchronization mechanisms. Distributed concurrency control, distributed deadlock, and recovery are another area which stimulates much research.

Block 5

The network interface must offer a high level communication protocol for the colloquy among the local DBMS.

Block 6, 7, 8

These blocks correspond to the operations made by the local DBMS's where, by means of the mapping functions and of the file system, transactions expressed in a local conceptual language are translated into commands for the mass storage.

All these functions and the relevant software modules can be found, even if in different forms, in all the prototype DDBMS which have been developed and are under development (SIRIUS/DELTA, COSYS, SDD-1, POREL).

Processing distributed transactions, however, poses hard problems as to keeping data integrity when something goes wrong somewhere. In fact we saw that in Block 4 locking mechanisms can be set up; therefore either local or general deadlock conditions can arise somewhere and for this or for other reasons, an elementary transaction, belonging to a single global transaction, must be aborted. If the transaction was updating the database, all the actions triggered by the transaction must be aborted, and their effects cancelled wherever they took place, and the database must be restored to its original state to assure data integrity.

Therefore special protocols have to be designed to coordinate the actions of the elementary transactions on every involved site, in such a way as either all of them perform or all of them are aborted and their side effects are cancelled. Such a way of operation violates somehow the assumption of autonomy of local sites, however it is the only known way to obtain a distributed recovery procedure. These protocols have a two-phase structure; in a first phase the sites *prepare* themselves to perform the elementary transactions and send each other "ready to perform" messages. If all the sites are actually ready, in the second phase the transactions are performed. Otherwise, if at some site the elementary transaction was aborted, no "ready" message can be issued from it and in the second phase all the elementary transactions are aborted, and the local DB are recovered on the basis of "in doubt" back-up log files which had been built before updating the DB.

5. CONCLUSIONS

The purpose of this chapter was to give a simple introduction to the problem of data sharing in distributed systems, therefore many topics have been treated in a very concise way, while many others have been purposely left out. In the bibliography some papers are listed, out of a fast growing literature, which can help the reader to go deeper into the subject.

BIBLIOGRAPHY

A - *General Topics*

Adiba M., Chupin J.C., Demolombe R., Gardarin G., Le Bihan J. (1978) Issues in Distributed Data Management Systems: a Technical Overview. In Issues in Database Management, H. Weber and A.I. Wasserman (Ed.), North-Holland, pp. 127-153.

Aschim F. (1974) Data Base Networks: an Overview. Management Informatics, vol. 3, n. 1, pp. 13-27.

Baldissera C., Ceri S., Schreiber F.A. (1979) Basi di Dati Distribuite. Rivista di Informatica, vol. IX, n. 3.

Emery J.C. (1977) Managerial and Economic Issues in Distributed Computing, in Information Processing 77, Gilchrist (Ed.), North-Holland, pp. 945-955.

Rossetti A., Schreiber F.A. (1978) Architetture alternative del sistema di elaborazione per un sistema informativo aziendale. Automazione e Strumentazione, vol. XXVI, n. 1, pp. 7-17.

Spaccapietra S. (1978) Problematique de conception d'un Système de gestion de bases de données reparties. Thèse à l'Université Paris VI.

Schreiber F.A., Di Filippo C. and Zagolin M. (1980) Un processore per l'interfacciamento di sistemi eterogenei di Information Retrieval su una rete di calcolatori, Atti AICA 80, vol. 2, Bologna, Ott. 1980, pp. 1245-1263.

B - *Application Features*

Giovacchini L., Schreiber F.A. (1976) Some Considerations on Distributed Management Information Systems, Proc. Int. Symposium on Technology for Selective Dissemination of Information, IEEE Press, pp. 102-109.

Paolini P., Pelegatti G., Schreiber F.A. (1972) An Application Oriented Approach to Distributed Databases. Proc. Journées AFCET sur les bases de données répartities, pp. 139-151.

Rossetti A., Schreiber F.A. (1978) Architetture alternative del sistema di elaborazione per un sistema informativo aziendale, Automazione e Strumentazione, vol. XXVI, n. 1, pp. 7-17.

C - *Allocation Problems*

Casey R.G. (1972) Allocation of Copies of a File in an Information Network, Proc. Spring Joint Computer Conference, AFIPS Press.

Chu W.W. (1973) Optimal File Allocation in a Computer Network. In Computer Communication Networks, N. Abramson, F. Kuo (Eds.) Prentice Hall.

Levin D.K., Morgan H.L. (1977) Optimal Program and Data Location in Computer Networks, Communications of the ACM, vol. 20, n. 5.

D - *Architectures*

Adiba M., Chupin J.C., Demolombe R., Gardarin G., Le Bihan J. (1978) Issues in Distributed Data Base Management Systems: a Technical Overview. In Issues in Database Management, H. Weber and A.I. Wasserman (Ed.), North-Holland, pp. 127-153.

Baldissera C., Ceri S., Schreiber F.A. (1979) Basi di Dati Distribuite. Rivista di Informatica, vol. IX, n. 3.

Biller H., Neuhold E.J. (1977) POREL: A Distributed Database on an Inhomogeneous Computer Network. Proc. 3rd International Conference on Very Large Data Bases.

Rothnie J.B., Goodman N. (1977) An Overview of the Preliminary Design of SDD-1, a System for Distributed Data Bases. Proc. 2nd Berkeley Workshop on Distributed Data Management and Computer Networks.

Spaccapietra S. (1978) Problematique de Conception d'un Système de Gestion de Bases de Données Réparties. Thèse à l'Université Paris VI.

Toth K.C., Mahmoud S.A., Riordon J.S., Sharif O. (1978) The ADD System: An Architecture for Distributed Databases. Proc. 4th International Conference on Very Large Data Bases.

Tsichritzis D., Klug A. (Eds.) (1977) The ANSI/X3/SPARC DBMS Framework. Report of the Study Group in Database Management Systems, AFIPS Press.

E - *Query Processing*

Bracchi G., Baldissera C., Ceri S. (1979) Query Processing Strategies for Distributed Database Processing. EEC-CREST Advanced Course on Distributed Data Bases, Sheffield City Polytechnic.

Hever A.R., Yao S.B. (1978) Query Processing on a Distributed Database. Proc. 3rd Berkeley Workshop on Distributed Data Management and Computer Networks.

Pelagatti G., Schreiber F.A. (1979) A Model of an Access Strategy in a Distributed Database. In Database Architecture, G. Bracchi; G.M. Nijssen (Eds.); North-Holland.

Wong E. (1977) Retrieving Dispersed Data from SDD-1: a System for Distributed Databases. Proc. 2nd Berkeley Workshop on Distributed Data Management and Computer Networks.

F - *Data Integrity and Reliability*

Colliat G., Backman C. (1979) Committment in a Distributed DB. In Database Architecture, G. Bracchi, G.M. Nijssen (Eds.), North-Holland.

Lindsay G.B., Selinger P.G. (1979) Notes on Distributed Databases. EEC-CREST Advanced Course on Distributed Data Bases (Integrity), Sheffield City Polytechnic.

Martella G., Ranchetti B., Schreiber F.A. (1981) Availability Evaluation in Distributed Database Systems. Performance Evaluation. Vol. 1, n. 3, pp. 201-211.