A DISTRIBUTED MEDICAL DATA BASE: NETWORK SOFTWARE DESIGN

by
Ernest Chang
Computer Science Department
University of Waterloo

Research Report CS-75-21

July 1975

A DISTRIBUTED MEDICAL DATA BASE: NETWORK SOFTWARE DESIGN

Ernest Chang
Computer Science Department
University of Waterloo

ABSTRACT

This paper describes the software design aspects of a distributed medical data base, to be implemented on a homogeneous minicomputer loop network of PDP-11/45s, using the UNIX operating system. Each user at a node, operating within a virtual USER MACHINE, is seen to interact with the FILE MACHINEs of either the local or distant node(s), through his local virtual NETWORK ACCESS MACHINE. This last resource has two logical components: a switch, which routes messages, and a Network Interface.

I. INTRODUCTION

1. Background

The rationale for the distributed data base design discussed in this paper is given in Chang (1974). In brief, the operating environment is that of a medical information system, in which the producers of information are distributed geographically, e.g., among physicians, laboratories, pharmacies, and emergency rooms. The types of data produced by the components of such a health care system vary widely. Even among physicians in a common category (such as general practitioners), the detail and format of data recorded varies from physician to physician. The data from such an environment is difficult to manage in a centralized computer system, and the normal solution of using a common simplified data format is a severe restraint on users. We maintain that each user should have the capability of defining data records suited to his needs.

Furthermore, we observe that such medical systems, partly because of their distributed nature, tend to exhibit strong locality of geographic reference, in that most of the data generated at a source is referenced locally - as in a typical physician's office. Therefore, where the requirement for user-definition of user-owned information is high, and data sharing is very important, data files should be owned locally by the creator of that data, but accessible to others through a communications network. Furthermore, such data files should possess the characteristic that all users be able to view them in ways most suitable to their needs. To this end, it is proposed that access to data fields be through the NAMEs of the fields, that the physical structure of the data records be defined

by the owners, but that access to the data be through logical mappings of the NAMEs of the fields onto the physical record. This is implemented through the following scheme: an elementary data item is known as a field. Fields are uniquely named, and are collected into data records. Data records which have the same configuration of data fields are stored in the same data base files. Data base files have two major segments - a data segment, which is the collection of data records, and a descriptor segment. The descriptor segment has two types of entries - a physical map, and a variable number of logical maps. A physical map describes the names of fields in a data record, and their storage attributes - type, range, and physical location in the record. Logical maps define ordered subsets of the fields in a data record, and represent the logical views of the data available to different users. In other words, physical maps describe storage structures, and logical maps describe user-perceived information structures (CODASYL, 1971). In this way, non-owners may see data in ways particular to them, and the owner may also restrict access to certain fields by not permitting them to be included in any logical access maps. It is recognized that a data base of this type will require system-wide agreement as to the meaning of names, which is argued as being preferable to system-wide knowledge of data formats.

The further argument is made that where centralized data storage is no longer necessary, and the complexity of data organization can be decreased, minicomputers may be used economically to implement such information systems. Since a single record format no longer has to store all relevant data from all possible sources, local record structures can be

much simpler. Although data now becomes distributed, concurrent access to data at different nodes should actually improve the access time relative to that of a large centralized system. Since most access to a data item is by its owner, we see that in the distributed system, most activity is local, whereas in the centralized system, all accesses must pay the overhead of searching through the entire data base.

System reliability is also enhanced by the use of a distributed system, as the failure of one node of the network does not necessarily mean the failure of the whole system. This is an important consideration in many application areas.

2. Implementation

This paper deals with the design and implementation of such a distributed data base. Briefly, a computer communications network connects a number of minicomputers (known also as nodes). The topology adopted for the network is a ring structure. Specifically, a Newhall loop has been chosen because the anticipated traffic is "bursty" - characterized by long idle periods with heavy loading when busy (Newhall, 1969). The network is homogeneous as each node will have the same minicomputer, running under the same operating system. This is the UNIX operating system, developed at Bell Labs (Ritchie, 1974), which is capable of time-shared multi-processing. The availability of UNIX only on PDP-11/45s makes a strong case for using these particular minis. The distributed data base is built on top of the operating system in the following way. At each node, there is implemented a virtual Data Base

Machine. A user of the Data Base Machine is said to be in data base mode: else, local mode is in effect. The Data Base Machine is itself built from other virtual machines, to be defined below. These are implemented as concurrent processes, through the facilities of UNIX. Each Data Base Machine has access to a single set of Data Base Files, known as the Data Base File Collection of the node. In this implementation, users in data base mode can only retrieve information from the network. Updates, insertions and deletions must take place locally, and are restricted to the owner of the Data Base File. This is a constraint that we impose for the sake of both simplicity and data security.

In Data Base mode, a user can only send requests and receive data from the distributed data base. Each request may be sent to one or more nodes, to be processed against one or more files. Because of this multiplicity of responses that may be generated from one request, we do not model an interactive form of transaction between user and data base. Rather, the user should be able to enqueue requests. This model presumes that at least many if not most of the transaction cycles will be long. (We leave to a later implementation the less general case of high-priority short transaction-cycle messages, on which the user must wait for completion before proceeding to the next activity). Acknowledgement from different nodes (or negative acknowledgement) should be made to the user for a given request. As responses arrive, the user should again be informed. Since it is not desirable to interrupt the user arbitrarily (indeed, the user may be signed off), clearly another device must be created. This is the request-log, of which more will be said below.

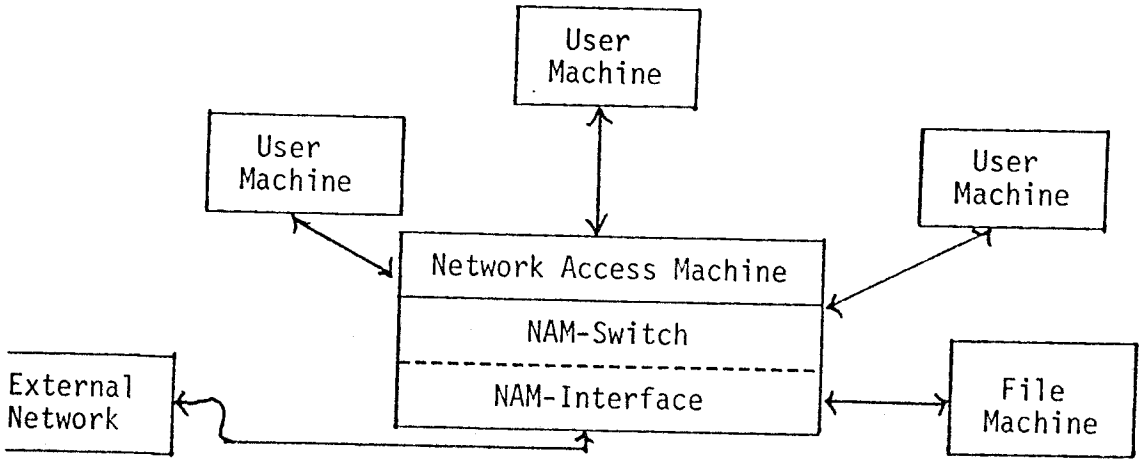
II. GENERAL ORGANIZATION

A distributed data base is the data and their access mechanisms, implemented in a computer communications network as described above. Each node of the network has a Data Base Machine, which gives users access to the distributed data base. A Data Base Machine is a virtual machine implemented at a node through UNIX. Each data base machine has three main parts: the first is a variable set of USER MACHINES. The next is the FILE MACHINE, which accesses the Data Base File Collection at a node, and the last major component is the NETWORK ACCESS MACHINE, which connects the Data Base Machine to the communications subnetwork, and vice versa. Conceptually, each user has a User Machine, which serves as the interface to the distributed data base. Requests are generated by a user on a User Machine and responses for the request are returned to the user from the network. The facility which accesses the File Machines of the network is the Network Access Machine. The network space known to the Network Access Machine includes the local node itself. Thus, a request may be sent to a local File Machine without treating it as a special case.

Another requirement of the Network Access Machine is that access to a different physical communications subnetwork should be accomplished easily. This can be done by separating the communications functions of the Network Access Machine from the control functions. Thus, there are two logical sub-machines of the Network Access Machine. One is called the NAM-Switch, which controls the flow of information from process to process, and the other is called the NAM-Interface, which incorporates all of the features of particular communications subnetworks.

The File Machine is the Data Base Machine resource which examines a request, processes it against one or more files of the Data Base File Collection, and sends the retrieved information back to the inquirer. The data thus returned is also a Data Base File, with a descriptor segment and a data segment. In this case, only one logical map will be present - the one which was used to access the original Data Base File. The physical map will correspond in organization to the logical map. The requestor thus has the capability of adding this file to the Data Base File Collection of the node.

Given the general functions of the components of the Data Base Machine, their interconnections are then simple:



III. SOFTWARE DESIGN

In the discussion below the following abbreviations and conventions will be used:

UM stands for a virtual USER MACHINE

NAM stands for a virtual Network Access Machine

FM stands for a virtual File Machine

We begin, by considering not each of the virtual machines, but the mechanism for passing information from node to node and process to process.

1. General Message Structure

The purpose of the Data Base Machine is to pass messages from a user to a file machine and back to the user. We make the following distinctions: Work Files are created by processes in order to hold data that is to be passed to another process and thereby again passed on or consumed. However, process to process communication as to which work files to pass and to whom will be done through Headers. The term message will encompass both. Headers will contain enough information to enable a naming algorithm within any process to construct from the header, the name of the work file it is supposed to access.

There are three types of work files: queries, DATA (from Data Base Files that satisfy the query), and responses (ACK, NAK, etc.). Thus, there are three types of headers. They share the same format:

Type	Q, D or R
From (Q)/To (D,R)	Node #
	User ID
	Process #
To (Q)/From (D,R)	Node #
	File Name

The header type can be either Q (for query), D (for data), or R (for response). The next field, labelled From (Q)/To (D,R) is for the FROM NODE # for queries. For data and response files, this field represents the TO NODE #. The User ID and Process # identify the original source of the request for which this header has been generated. The field To(Q)/From(D,R) is analogous to the field described above - i.e. for query files, it contains the TO Node #, and for the others, it contains the FROM NODE #. The File Name is the local name of the Data Base File relevant to the query. We name work files as follows:

User ID/Process #/(To/From) Node/File Name/Type(Q,D or R)

This construction will yield unique work file names, as should be readily evident to the reader.

2. User Machine (UM)

The user may:

a) construct a query, through interaction with the user machine. The query might be of the following type:

1. A request for the names of the nodes thought to exist in the network in the directory of a node.
2. A request for the file names of the data base collection of a node.

3. A request for the logical maps pertaining to a Data Base File.
4. A search vector, consisting of a Boolean combination of field-names and values.

b. send a query, again by interacting with the UM. The query may be sent to all nodes (Broadcast), or to particular nodes, given the assistance of the UM. Further, the query might be applied only to a specific file or file(s), if so desired.

c. examine his request-log. Each SEND of a query (a request) will generate an entry in the log. The user should be able to ask for a display of such entries. Furthermore, each such entry points to a list of the nodes to which the request has been sent, and the status of the request at the node in question (e.g., ACK, NAK, etc.). Each such node should in its turn point to a list of files at that node which have returned data satisfying the query. This list, in order to be complete must include a 'NO MORE FILES' entry for the node. The request-log seems to be the only feasible way of keeping track of many requests in an environment where transaction-cycles may be long, and each request may generate multiple responses.

d. examine the data that has been returned for a particular file from a particular node in response to a particular query. Each such response will generate a distinct file when received, and the user can examine these files at will. Each such data file will also carry its own physical-logical mappings, so that it can be displayed interpretively. The user must be able to access these files through a set of utilities, which should include the following kinds of functions:

1. Sort.
2. Print from Record N and Print from Record M to Record N (as in a line-oriented text editor).
3. Print records satisfying simple conditionals on field-names and values.

e. delete entries from the list described in (c) and files returned as described in (d). In addition, he should be able to include such data files, if he wishes, as part of the Data Base file collection.

The system may:

a) interact with the user to edit the query being constructed.

b) interact with the user to SEND the query thus constructed. This function may require the UM at the system level to carry information about nodes in the network.

c) The UM must receive messages from the Network Access Machine, and enter the relevant pointers in the user request-log. Even if the user has signed off from his/her User Machine (or perhaps signed on to UNIX as strictly a local user), the UM must have access to the user's request-log. In addition, the UM must set a flag so that the user can be informed at a convenient time that messages have been received (for example, when a carriage-return is struck).

3. Network Access Machine

We think of the Network Access Machine as providing all of the logic and protocol needed to decide which messages get sent to what process at what node. Thus, a query (in general) is to be sent to a File Machine. If it is to be sent to the local FM, the NAM passes the Header to the local File Machine. If it is to be sent to one or more remote nodes, the NAM is able to do so, through knowing the protocol required for the network.

The NAM has been stated to consist of two lower machines, the NAM-Switch, and the NAM-Interface.

The NAM-Switch

In first considering the NAM-Switch, we think of it as performing two functions: SEND to a process in the network (including the local node), and RECEIVE from the interface.

Switch-send

a. UM may ask a Q-file to be sent to a node. If the node is local, the header is passed on to the local FM.

b. FM asks that an R-file (respond ACK or NAM) be sent to the process that generated the Q-file being processed by the FM. If that was local, the header is passed to the local UM. If not, it is passed to the NAM-Interface.

c. FM asks that a D-file is sent back to the user. The same decisions as in (b) are made.

Switch-receive

The NAM-Interface receives messages from the remote nodes of the network, addressed to the local node. Clearly, these are of three types

(Q, R and D). The Switch-Receive sends the Q-file (by passing the header) to the FM, and similarly, R-files and D-files are passed to the UMs by passing their respective headers.

The NAM-Interface

Send-function:

When passed the header, the interface must generate the file name in preparation for transmitting it to a node of the network. In doing this, the Interface must establish a link to the node, and when it is prepared to send it must then do the file transfer according to the protocol expected (for example, in buffers of defined lengths, generates the check sum, etc.). The Interface must also control the hardware device(s) which allow communication.

Receive-function:

When the interface is ready to receive from a distant node, it must construct a header from the first part of the message received, and a file (named appropriately) for the data. The Interface then builds the rest of the file as the remainder of the message arrives, either directly or through a buffer(s). When the message is completed, the interface must terminate the transmission link, and pass the header onto the NAM-Switch-Receive, which processes the header as described.

4. File Machine

The File Machine is responsible for processing queries passed to it from the NAM. It responds with an acknowledgement if it recognizes the file (or files) asked for as being within the Data Base File Collection

local to its node. If not, it responds with an appropriate message (e.g., NAK). Such messages are passed, through their respective headers, to the NAM, which then routes them.

Queries which the FM processes are applied against the appropriate Data Base files. Each such transaction will generate a DATA-file which will be sent back to the proper User Machine by the NAM. After all files for a given query have been processed, the FM must send a final R-message indicating that this node has completed all processing.

5. File Deletion

In general, files which are actually transmitted to the network external to the local node are released from the system by the Interface, if sent successfully. Note that queries which are routed to the local FM never actually reach the Interface. For files received from the network, Q-files go to the FM, which will remove them after all processing is complete. R-files go to the UM, which may remove them after the User's Request-log has been updated. D-files which go to UMs are the responsibility of the users to delete. They may in fact become included into the Data Base Collection, so that some degree of iterative processing is possible.

6. Inter-Process Communication

The User Machines are virtual machines, which each user in Data Base Mode possesses. The Network Access Machine and File Machine are to be considered as system resources, one to a node. The problem is to pass information in an ordered way between these processes. The model we choose

is to consider that the FM and NAM each possess a QUEUE, of which it is the sole CONSUMER. These processes are driven by entries in the queues, which will be file-Headers. When the queue is empty, the process is idle. Headers are placed into queues by PRODUCER processes. For example, a user generates a Q-file, which the UM places in the queue of NAM-Switch. If the query is to be sent to a node in the external network, then the NAM-Switch places the header in the queue of the NAM-Interface, for final action.

The placing of a header into a queue, and the removal of a queue-entry are processes which should occur within critical regions. It appears that UNIX lends itself to this requirement without too much difficulty.

IV. EXTENSIONS

At the present time, we have only defined one type of transaction - from user to File Machine, and back. Each such transaction currently has only one priority. It will be quite feasible to extend the range of transactions to cover at least two or three priority classes. The highest would be user to user messages, each of which would be quite short. The next would be queries against single files, and the transactions of lowest priority would be those which required the searching of many files at any particular node. Because of the distributed nature of the data base, it makes little difference whether one or more nodes are being searched.

The only type of transaction being supported at present is data retrieval, given a particular query. In a more general case, the file which is sent may either be considered as a data segment, to be processed against another file (which is a program file), or vice versa. This generalization will be quite feasible within the network as designed. Finally, the rewriting of the Interface will allow a Data Base Machine to communicate to networks of different physical characteristics. This flexibility allows extensions in several directions: a single node being part of several networks, or many distributed data bases implemented in different homogeneous communications subnetworks.

ACKNOWLEDGEMENTS

This project was supported in part by the Province of Ontario Demonstration Model Grant DM187, within the environment of the Computer Communications Networks Group (CCNG), University of Waterloo.

BIBLIOGRAPHY

1. Chang, E.J.H., Linders, J., A Distributed Medical Data Base, Meth. Inf. Med., 4:13, Oct. 1974.

2. CODASYL Data Base Task Group Report, April 1971.

3. Manning, E.G., Peebles, R.W., A Homogeneous Network for Data Sharing - Communications. CCNG Technical Report E-12, University of Waterloo, March 1974.

4. Newhall, E.E., Farmer, W.D., An Experimental Distributed Switching System to Handle Bursty Computer Traffic., Proc. ACM Conference, Pine Mountain, Georgia, October 1969.

5. Ritchie, D.M., Thompson, K., The UNIX Time-sharing System. Communications ACM 17 (7): 365-375, July 1974.