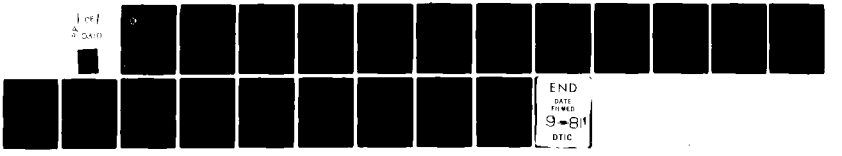
AD-A103 101 KANSAS STATE UNIV MANHATTAN DEPT OF COMPUTER SCIENCE F/G 9/2
RESEARCH IN FUNCTIONALLY DISTRIBUTED COMPUTER SYSTEMS DEVELOPME--ETC(U)
FEB 76 F J MARYANSKI, V E WALLENTINE DAAG29-76-G-0108
UNCLASSIFIED CS-76-03 NL



END
DATE FILMED
9-81
DTIC

AIRMICS

Army Institute for Research in Management Information and Computer Science

313 Calculator Bldg.
GA Institute of Technology
Atlanta, GA 30332

Technical Report

RESEARCH IN FUNCTIONALLY DISTRIBUTED COMPUTER SYSTEMS DEVELOPMENT

Kansas State University

Virgil Wallentine

Principal Investigator

VOLUME XVII

IMPLEMENTATION OF A DISTRIBUTED DATA BASE SYSTEM

U.S. ARMY COMPUTER SYSTEMS COMMAND FT BELVOIR, VA 22060

1 Interim rept.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE	READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER: (6) Research in Electrical Distributed Computer Systems Development. Volume XVII.	2. GOVT ACCESSION NO.: AD-A103101
4. TITLE (and Subtitle): IMPLEMENTATION OF A DISTRIBUTED DATA BASE SYSTEM. AD-A103160	3. RECIPIENT'S CATALOG NUMBER
	5. TYPE OF REPORT & PERIOD COVERED: Interim
	6. PERFORMING ORG. REPORT NUMBER: CS-76-03 ✓
7. AUTHOR(s): (10) Fred Maryanski, Virgil E. Wallentine (5.)	8. CONTRACT OR GRANT NUMBER(s): (15) DAAG 29-76-G-0108 ✓
9. PERFORMING ORGANIZATION NAME AND ADDRESS: Kansas State University, Department of Computer Science, Manhattan, KS 66506	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS: (11)
11. CONTROLLING OFFICE NAME AND ADDRESS: US Army Research Office, P O Box 12211, Research Triangle Park, NC 27700	12. REPORT DATE: Feb 1976 ✓
	13. NUMBER OF PAGES: 18 pages ✓
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office): US Army Computer Systems Command, Attn: CSCS-AT, Ft. Belvoir, VA 22060 (12) 21	15. SECURITY CLASS. (of this report): Unclassified
	15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

The findings in this report are not to be construed as an official Department of the Army position, unless so designated by other authorized documents.

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

DBMS
DDBMS
Distributed processing
Network Systems

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

-over-

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

591123

-ABSTRACT-

In this paper we present an overview of data base management systems (DBMS), the motivation for distributed data base systems (DDBMS), a set of possible network topologies served by the distribution, the mechanisms necessary to integrate (and communicate between) the DDBMS system elements when distributed across a nonhomogeneous network of minicomputers, and some implementation details on a prototype system. The current prototype distributes the DBMS and application program function across an IBM 370/158 and a (minicomputer) NOVA 2/10. In the near future, a third machine, the Interdata 85 minicomputer, will be added to the network. The DBMS used is a network system as specified by CODASYL. The emphasis in this paper will be on the problems posed by the heterogeneous machines and the intertask (processor) communication system which is utilized in the distribution of data, programs, and control.

Implementation of a Distributed Data Base System

•Technical Report
CS76-03

V.E. Wallentine and F.J. Maryanski

February 1976

Dept. of Computer Science
Kansas State University

This work has been partially supported by the U.S. Army Research Office Grant No. DAAG29-76-G-0108.

ABSTRACT

In this paper we present an overview of data base management systems (DBMS), the motivation for distributed data base systems (DDBMS), a set of possible network topologies served by the distribution, the mechanisms necessary to integrate (and communicate between) the DDBMS system elements when distributed across a non-homogeneous network of mini-computers, and some implementation details on a prototype system. The current prototype distributes the DBMS and application program function across an IBM 370/158 and a (mini-computer) NOVA 2/10. In the near future a third machine, the Interdata 85 mini-computer, will be added to the network. The DBMS used is a network system as specified by CODASYL. The emphasis in this paper will be on the problems posed by the heterogeneous machines and the inter-task (processor) communication system which is utilized in the distribution of data, programs, and control.

I. Motivation for Distributed Data Base Systems

A. Back-End Data Base Management Systems

It has been a long standing practice in computing to attempt to maximize the amount of processing performed by the Central Processing Unit (CPU). Generally, this has been accomplished by reducing the amount of time the CPU must wait for information from the slower input/output devices. Several major developments in computer science have resulted from the desire to maximize CPU utilization. Among these developments are interrupt driven I/O, slave I/O processors, direct memory access, multiprogramming, multiprocessing, and distributed networks.

The advent of data base management systems (DBMS) has resulted in more efficient organization of data on secondary storage devices and the development of powerful high-level languages for the manipulation of data. A statement in such a data manipulation language (DML) could result in several I/O operations being performed. In a standard multiprogramming environment, execution of one such DML statement could force the system into multiple task switches. This would result in considerable overhead. To reduce this overhead, Canaday, et al [1] have developed the concept of a back-end computer for data base management. In a back-end system, a minicomputer is used as a dedicated processor for the DML statements. The minicomputer has exclusive access to the secondary storage devices upon which the data resides. Whenever the host computer encounters a DML statement, a message is transmitted to the minicomputer indicating the DML statement and its arguments. The back-end machine then performs the DML statement and all its necessary I/O operations. In effect, the back-end computer is treated as a sophisticated I/O device for the host machine. The incorporation of the back-end minicomputer into the system reduces CPU

swapping on the host computer and provides for an overall increase in availability of system resources.

The two principal advantages of a back-end DBMS are the reduction in primary memory requirements within the host computer and an increase in overall system performance. Primary memory is reduced in the host by moving data base functions and buffer areas to the back-end machine. In the system being developed, the overall reduction in primary memory requirements in the host machine is 31K + N*20K where N is the number of application programs executing in the host. Details regarding the distribution of software between the host and back-end machines are given in Section II of this paper.

B. Distributed Systems

A back-end DBMS as originally conceived [1] consisted of a large general purpose computer as the host machine and a dedicated mini computer as the back-end processor. Figure 1 illustrates a typical back-end configuration. The host computer executes the application programs which generate requests for information from the data base. These access requests are sent via the host message system to the back-end system.

Once a request has reached the back-end system, it activates the DBMS and is queued at individual DBMS tasks. Each task is scheduled by the back-end message system to access the data base a number of times until the appropriate DML operation has been performed. Upon completion of the data base operation, a message is returned to the application program via the intervening message systems, I/O drivers, and the inter-computer communications channel. Such messages consist of requested data and status conditions on store and retrieval requests.

The DBMS configuration discussed thus far is composed of two computers. Several extensions to this basic configuration are possible. Let us first

consider a network DBMS with several application and several back-end machines (Figure 2). In this arrangement, an application program may access any data base, provided there is a connection to the managing back-end machine for that data base.

In all previous discussions of back-end DBMS, it has been assumed that the back-end machine will be dedicated to data base operations. In some cases, this restriction may inhibit full utilization of system resources. Provided that the back-end machine has a multiprogramming executive system, there should be no difficulty in allowing it to perform tasks other than DML operations. In a generalized situation, a processor could be performing DML operations on a data base while at the same time executing an application program which requests data base information of another processor in the system. Such a processor is considered to be bi-functional. In a generalized DBMS network, the only restriction as to the function of a processor is its physical connection to secondary storage. Figure 3 displays a DBMS network with host, back-end, and bi-functional nodes.

In order to maximize system resource utilization in a network DBMS, the workload of the nodes must be balanced. This can be accomplished if tasks can be distributed among the processors in the network in an equitable manner. A distributed DBMS (Figure 4) can realize this goal. In a distributed DBMS, each node is bi-functional. An application program may be executed by any processor. The selection of the application processor is done automatically and is transparent to the user. As in previous topologies, data base functions are limited by physical connections to the machine which controls the I/O device on which the data base resides.

II. Data Base Management Software

A. Central DBMS

The data base software to be used in the distributed data base network is obtained by altering an existing data base management package. A brief

description of the main features of the DBMS is given below. Details may be found in References [2,3]. The DBMS contains three basic languages--a data definition language (DDL) for describing the data base to be accessed by a program, a data manipulation language (DML) which prescribes the manner in which the data are transferred between the application program and the data base, and a Device Media Control Language (DMCL) which maps the data onto physical storage.

The DDL is divided into two components, schema DDL and sub-schema DDL. A schema DDL is employed to describe the entire data base in terms of the characteristic relationship between the data items. The sub-schema describes the data base as viewed by the application process. The sub-schema provides for limiting the access of a program to a particular portion of a data base, or allowing a particular program to modify a data base for its own purposes without affecting the use of the data base by other programs.

The DML is used to augment the host high-level language of the data base management system. The DML provides the capability of performing complex data set manipulations in a single high-level language statement. The host language may be any general purpose high-level language. In keeping with CODASYL recommendations, the DML features have been added to COBOL by means of a pre-processor. The preprocessor translates DML statements into COBOL for compilation and execution.

To implement the DBMS both preprocessing and execution software modules are necessary. The preprocessing modules include the DDL schema processor, the DDL subschema processor, DMCL processor, and the DML preprocessor. The major execution modules are the subschema interface (IDMS), the data base manager (DBMS), the data base I/O routine (DBIO), and the multitasking control program (CAMP).

Figure 5 illustrates the actions taken when a DML statement is executed.

A DML statement takes the form of a call to the interface routine. The interface routine then transmits a message instructing the DBMS routine to perform the appropriate data base function. If the DML statement requires information not presently in the system data buffers the DBIO routine is called to perform the actual I/O operation. The data obtained and pertinent status information is transmitted back to the application program through the DBMS and IDMS routines. Multi-tasking of the DBMS routine occurs under the control of the CAMP monitor. All actions affecting the data base are recorded on a journal tape for recovery purposes.

B. Distributed DBMS

In order to implement the DBMS system in a distributed network, the software must be allocated across the application (host) and back-end machines. In keeping with the philosophy behind the back-end concept, modules and tables used in managing and accessing the data base are transferred to the back-end. Figures 5 and 6 depict the distribution of software between the application and back-end computers. This division is intended to minimize the number of requests the application CPU makes of the data base manager and to keep the amount of information actually transferred between machines at a minimum. Inter-machine communication is accomplished by employing the inter-computer communications channel to transmit information between the interface routine and the data base manager. This transmission takes place under the aegis of a general message system which is described in the next section of this paper.

To optimize overall system performance, the DBMS, DBIO, the recovery portions of CAMP, and the recordkeeping operations must be transported to the back-end. Since the data base operations will be performed by the back-end machine, it is necessary to have the subschema available for validation of the DML requests. Positioning the subschema in the back-end machine will

also minimize the amount of traffic on the inter-computer communications channel.

A substantial portion of the data base software is removed from the application machine. Since the subschema has been moved to the back-end, much of the validation performed by the interface routine in the present version can be performed on the back-end machine by the DBMS. The interface routine in the back-end version need only transmit information between the application program and the message system. The functions of CAMP on the host machine are to check for abnormal termination in the application program and to insure that the area associated with each DBMS task is open. The task control functions of CAMP are handled by the message system and the operating system in a network environment.

III. Inter-Computer Communication System (ICCS)

A. Introductory concepts

The ICCS prototype was developed to provide the requisite communication lines between application tasks and DBMS tasks. It consists of two sub-systems, as shown in Figure 7, which coordinate the exchange of both control information and data. The Multicomputer Communications System (MCCS) executes on the IBM 370 as a CMS machine under VM/370. The Inter-Task Communications System (ITCS) executes on the Data General Nova 2/10 under the RDOS operating system.

ITCS and MCCS perform identical functions, i.e. control the exchange of messages between machines and tasks. But they were constructed under quite diverse constraints. MCCS is a single task implementation of the message control system and was targeted to execute under a single task operating system such as CMS. ITCS is a multi-tasking version and is constructed to run under a multi-tasking system with an efficient inter-task communication

system (such as those provided in real-time operating systems on mini-computers).

The operation of the inter-task message system proceeds as displayed in Figure 7. The application program issues DML statements. These DML requests are converted to a series of messages to be exchanged between the interface and the DBMS tasks. Each message is sent to a unique DBMS task in the back-end (one for each defined level of multi-threading of the data base) via MCCS and ITCS. Each is routed to the appropriate task (indicated by a unique name) via the chosen hardware link (telecommunications, channel-to-channel adapter, etc.). Upon receipt of a message at ITCS, it is queued until requested by the DBMS task. Upon completion of the DML function, messages are transmitted back to the application program interface. These messages compose the response to the task in the host that issued the DML. These messages are again transmitted via the MCCS/ITCS communication link.

Messages (both data and control) are directed to a particular task via a SEND procedure; and messages are requested from a task via a RECEIVE procedure. Their usage in our prototype system in a back-end DBMS environment is given in Figure 8. The SEND procedure identifies the name of the task addressed by a TO_ID parameter. The RECIEVE procedure optionally identifies a particular task to receive a message in FROM_ID, or receives a message from any task on a first-come-first-serve manner. RECEIVE can also specify whether the task issuing the receive is to wait for that message or proceed unconditionally. In the remainder of this section we describe the function and structure of ICCS as it currently exists. And we further discuss its dependence on currently available software and hardware communications components.

B. ICCS Software Structure

The functional characteristics of the ICCS are as follows:

1. it provides synchronization between tasks as well as processors;
2. it provides a message exchange system between tasks through which both data and control information can pass;
3. it handles buffer management in both the host and back-end processor;
4. it isolates the application program interface and the DBMS tasks from any knowledge of the physical location of the others, i.e. whether the host and back-end are connected locally or remotely;
5. it provides a well-defined interface (the CALL statement) to both the interface and the DBMS task; and
6. it is modular and hierarchical in nature so as to permit the straightforward modification necessary to adapt ICCS to most operating and teleprocessing system environments.

The hierarchical structure of ICCS will be discussed in terms of its integration into the operating, I/O control, and teleprocessing systems on conventional computers. Figure 9 displays the different levels of software necessary for application programs to do I/O. For local connections, the I/O driver isolates the operating system from interrupt handling. In order to handle remote connections, the line protocol handles error detection and retransmission, code transparency, code conversion and line control. In all cases the application program activates the operating system by a Supervisor service call. But at this level the application programmer still must do his own synchronization, buffer management, and routing (addressing to machine as well as task). The message system ICCS was created to relieve the application programmer of such complexity.

The current structure of ICCS is presented in Figure 10. The interface gets control from the application program via the CALL. The interface in turn invokes ICCS with a sequence of fixed block messages which compose the

data and/or request to be made of (sent to) a DBMS task. The CALL specifies only a task name as the entrance to the DBMS, and has no concern as to its location. MCCS either sends the sequence of buffers to ITCS (if it has space available in its buffer space), or queues the messages in its own buffer space until space becomes available. If the application program needs to wait for a result, it will issue another CALL to MCCS to receive data from a task. A receive can specify the wait or proceed option. This is noted by MCCS. When ITCS sends a response (also using the CALL interface between ITCS and the DBMS task), MCCS communicates the message to the appropriate application program - again by its unique process (program, task) name. (See Figure 11 for message flow possibilities.)

Since ICCS is comprehensive in its message exchange capabilities, an application to distributed data bases is possible. That is, since communication is at the task level, the DBMS task may be running on any machine in the network. In either case, ICCS will perform the correct transmission of (route) buffers. See Figures 11 and 12 for the flow of control in ICCS and the movement of data buffers through the distributed system. Note (See Figure 11) that the application program can activate DBMS tasks in the host (the machine on which it is executing) using path 1; and it can invoke a distributed DBMS task via path 2. While the prototype system message flow for two data bases is indicated in Figure 11, the message flow necessary in a system of multiple (more than 2) data bases on multiple machines is achieved by adding new message systems. (MCCS is needed if the new operating system is a single task system and ITCS is needed if it is a multi-tasking system.). Note that an arrangement of four copies of a message system (each being MCCS or ITCS) establishes ICCS between any application program and any number (from 1 to 4) of DBMS tasks in the network topology of Figure 4. Since ICCS supports a "store and forward" function on buffers, there need only be a possible route between machines; and it need not be direct.

When an application program sends a message to a DBMS task in the same machine, ICCS queues messages destined for a host DBMS task (a CMS machine in the prototype) in the "out" list of MCCS first. Then it merely links it to the "in" list of MCCS or ITCS. (See Figure 12 for details.) In case a distributed DBMS task is to be invoked, the messages are sent (by MCCS or ITCS via the hardware connection) to be queued (when buffer space is available) in ITCS or MCCS. The reverse operation is achieved in the same manner. This generalized message exchange mechanism achieves the synchronized operation of the application program and the DBMS tasks. In order to avoid deadlock on message buffers, which can arise when incoming messages are allocated all buffers, the buffer allocation algorithm of the T.H.E. system [4] is implemented. This scheme reserves a minimal amount of buffers for outgoing messages and maintains a safe "region" of buffer allocation to input, output, and computing processes.

C. Heterogeneous architecture implications

Both hardware and software structures of various architectures impact the distribution of the DBMS. In the hardware category, application programs must be portable to an appropriate machine. In the current prototype, any language (which has compiler support in a particular machine) which supports the CALL statement to an external procedure provides such portability. Further architectural dependence arises in the development of the DBMS for each machine. These considerations are addressed in another paper [5]. The last consideration for hardware dependency involves the characteristics of the inter-computer connections. This only impacts the particular line protocol. (This must be compatible between machines. In our prototype, IBM binary synchonous is used.) And since the ICCS system utilizes a commercially available operating system in each machine, it is isolated from the particular hardware connection.

A second order impact on the distribution of the software is due to the operating system services in a particular machine. Vendor-supplied software systems were viewed as too large to construct. Therefore, a minimal set of characteristics are presented below which must be available in the operating system of each machine. These items are most appropriate for mini-computer real-time executives, since it is basically a mini-computer network we wish to accommodate.

1. The system must be multi-tasked with inter-task communications which minimally exchange a one word (16-bits) parameter.
2. It must support dynamic priority change via a system task. ICCS uses this feature to schedule itself at the highest priority over application and DBMS task functions since it manages buffer space.
3. It must support a synchronizing primitive to protect the buffer space manager from unwarranted intrusion in its update process. This mechanism can be semaphores, locks, or the ability to disable interrupts for a short period of time. In the portable version of ITCS (written in FORTRAN with real-time extensions) the interrupts are disabled and later enabled via a protected (special) system call. This call is unavailable to the application program due to the access validation within the system module. This module (very small) must be added to each machine's system modules. This feature is necessary since the SEND and RECEIVE functions are independent tasks which utilize a common buffer manager.
4. The system need only support basic direct and basic sequential access methods.
5. The system command processor must be structured as a task with which messages can be exchanged. This permits an operator to distribute tasks around the system from any machine (as long as it is not currently in running state).

IV. Summary and Future Enhancements

This paper develops the theme that a data base management system and an inter-task (machine) communications system are the central elements of a data base accessing system which is distributed in a network of mini-computers. The data base management languages (DML, DDL, and DMCL) isolate the user from the file and command system; and the message exchange facility makes the network transparent to the user, the application program and the interface between the application program and DBMS task. Furthermore, the Data Manipulation Language (DML) establishes a "clean" and well specified interface to utilize in distributing applications and DBMS tasks across the network.

At present the network has two nodes with the anticipation of adding two more mini-computers locally and one additional mini and one large main frame computer at remote sites. Since the current system does not permit dynamic movement of tasks in the network (an operator must move them), a network control language (NCL) is being developed to define resource allocation in the network. The NCL operations are then converted to operating system dependent (in a particular machine) resource control functions. Further, refinement of the inter-computer communications system to support the communication between a distributed operating system's modules is proceeding.

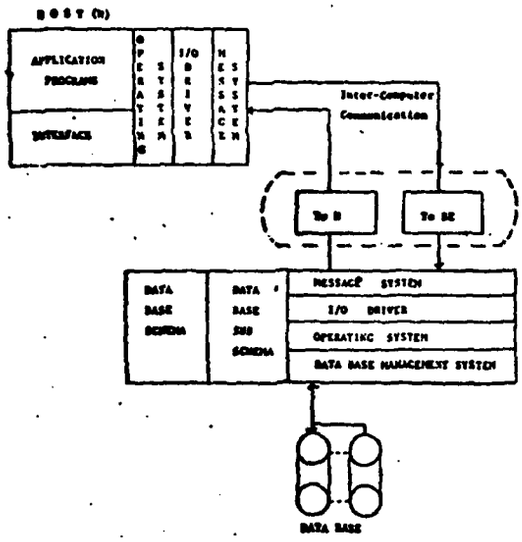


Figure 1. Back-end DBMS

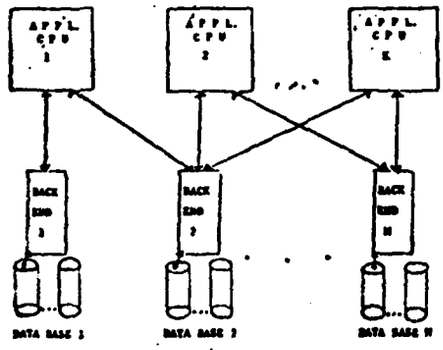


Figure 2. Network DBMS

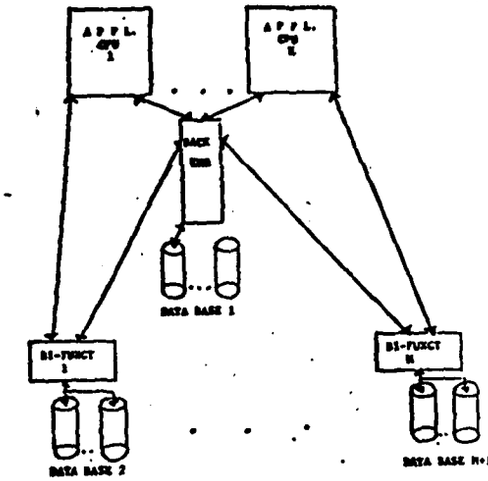


Figure 3. Network DBMS with Bi-Functional Machines

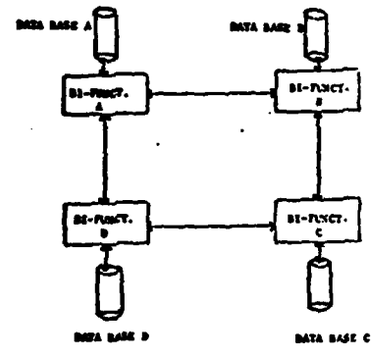


Figure 4. Distributed DBMS

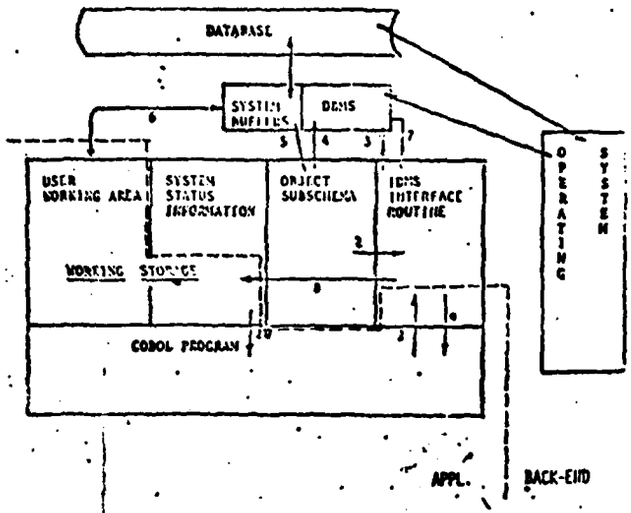


Figure 5. DBMS Software Distribution

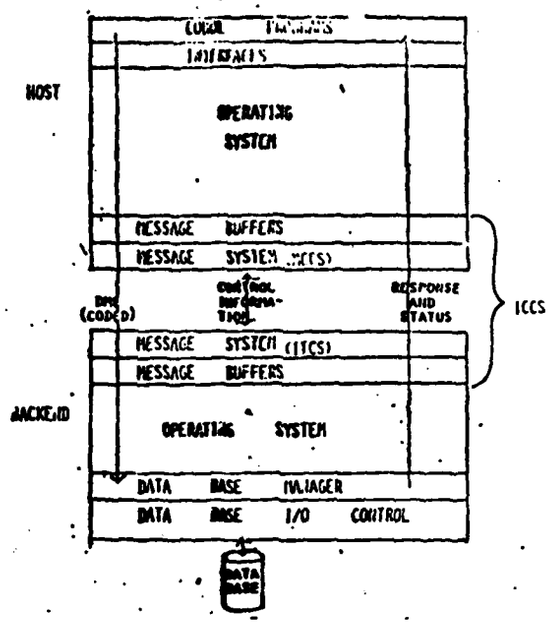


Figure 7. Application and DBMS Task Relationship in a Data Base Environment

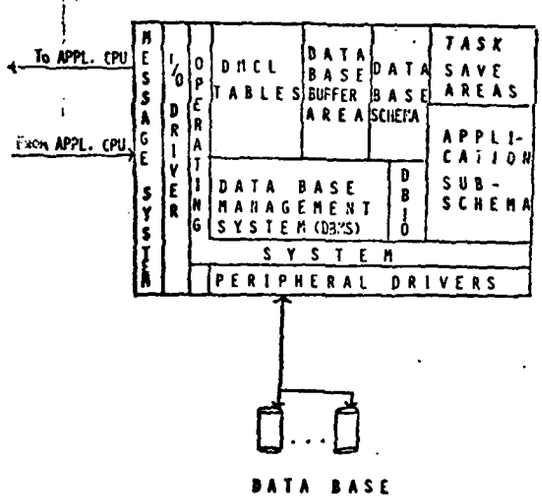


Figure 6. Distribution of Software in Back-End Computer

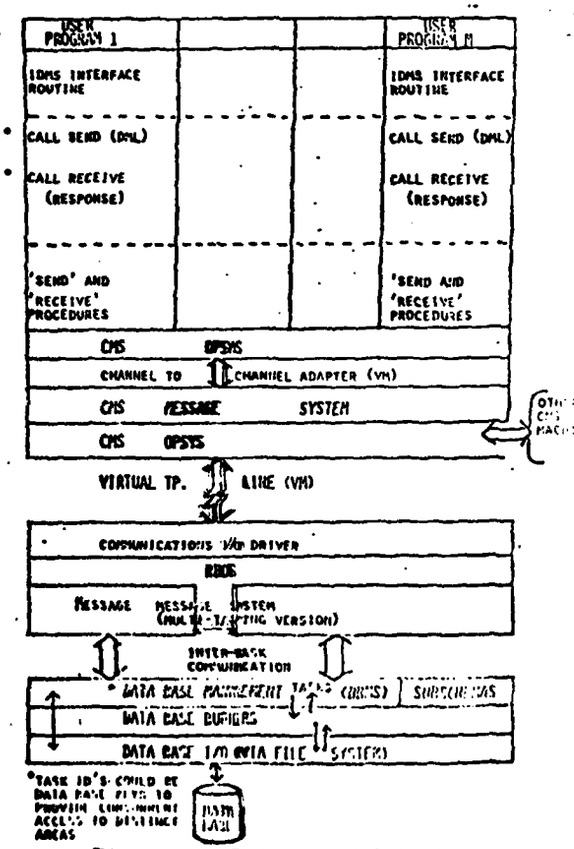


Figure 8. Integrated IDMS System

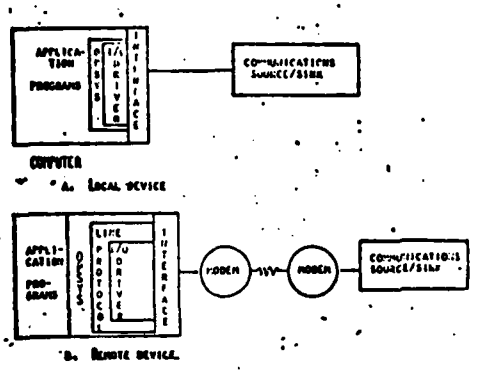


Figure 9. I/O Software Structure

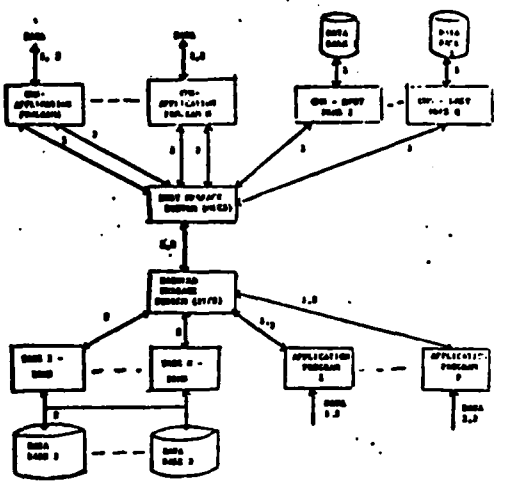


Figure 11. Flow of Messages in a Distributed Data Base Configuration

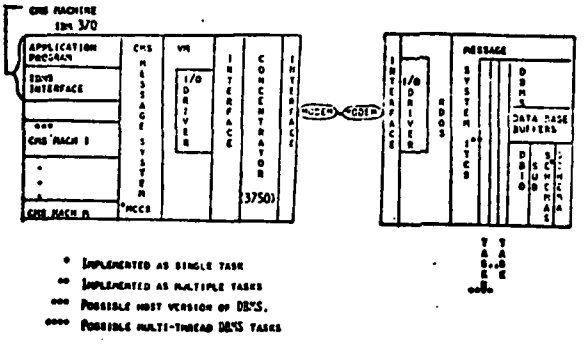


Figure 10. Structure of Message System in Prototype Configuration

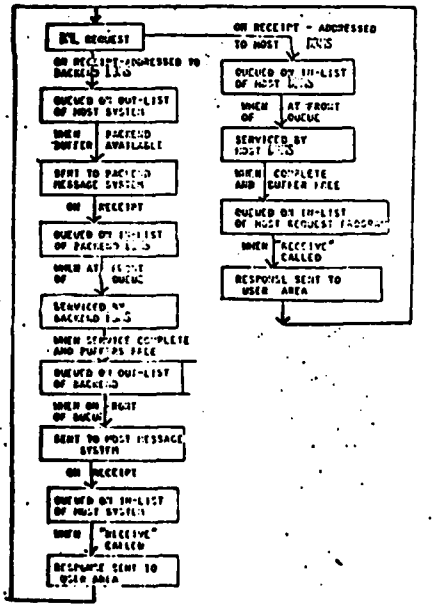


Figure 12. State transition diagram for Application Program - DBMS Task Message Exchange

V. References

1. Canaday, R. H., et al., "A Back-End Computer for Data Base Management," CACM 17, 10 (Oct., 1974) pp. 575-582.

2. IDMS DDL Reference Guide, Cullinane Corp.

3. IDMS DML Programmer's Reference Guide, Cullinane Corp.

4. Bron, C. Allocation of Virtual Store in the T.H.E. Multiprogramming System. In C.A.R. Hoare and R.H. Perrott (ed.) Operating System Techniques, pp. 168-193. Academic Press, 1972.

5. Wallentine, V.E., et.al. On the Implementation of a Backend Data Base Management System. Technical Report. Dept. of Computer Science, Kansas State University, 1975.

VI. Acknowledgements

The authors wish to acknowledge the programming effort of Sheldon Fox, Lee Allen, and Rich McBride. Their dedication to the job is appreciated.