A DISTRIBUTED COMPUTER SYSTEM

DAVID J. FARBER

DEPARTMENT OF INFORMATION AND COMPUTER SCIENCE

UNIVERSITY OF CALIFORNIA, IRVINE

IRVINE, CALIFORNIA 92664

TECHNICAL REPORT NO. 4, SEPTEMBER 1970

TABLE OF CONTENTS

ABSTRACT i

REASONS FOR THE DISTRIBUTED COMPUTER SYSTEM DEVELOPMENT ii

CURRENT STATE OF THE ART iii

INTRODUCTION iv

RESEARCH GOALS OF THE DISTRIBUTED COMPUTER SYSTEM PROJECT 1

GENERAL PHILOSOPHY AND ASSUMPTIONS 3

DESCRIPTION OF A PROTOTYPE DISTRIBUTED COMPUTER SYSTEM 4

A DETAILED SCENARIO OF SYSTEM OPERATION 6

OTHER ISSUES OF IMPORTANCE 10

FAILURE DETECTION AND RECOVERABILITY 12

DUPLEX NODES 18

COAXIAL CABLE RELIABILITY 19

CONCLUSION 20

ACKNOWLEDGMENTS 21

FIGURE 1 - PROTOTYPE DISTRIBUTED MACHINE 22

ABSTRACT

The goal of this project is to plan and develop a computing system that will provide interactive computing services with a high degree of reliability, a variety of language systems, incremental expansion capability, competitive costs, a respect for human factors considerations; and the project will require only a modest amount of system programming resources.

REASONS FOR THE DISTRIBUTED COMPUTER SYSTEM DEVELOPMENT

A need exists in many university and industrial computation centers for a highly-reliable multi-language computer system with low per-user costs. The current approaches of the multi-language general purpose time-sharing system do not achieve these goals. The desire to do all things for all users has produced unsatisfactory solutions in the form of high cost and low reliability.

There have been a number of single-language dedicated-machine systems which have been capable of achieving a low cost per user. These systems, however, force the user either to commit himself to the one language or to have access to a number of independent machines with the attendant difficulty of moving files, different operating procedures, etc. Also, (in a small dedicated environment) it is expensive to support the large amounts of different peripheral equipment needed to satisfy many users. An alternative solution is to have a system with a variety of dedicated single-language machines. The capabilities desired are an ability for users to select which language they desire to use at any particular moment, and that data files be accessible by all machines. This is the general approach that we will take in the development of the Distributed Computer System (DCS 1).[1]

---

[1] "Research Proposal for a Distributed Computer System" - Submitted to NSF, Feb. 1970 by University of California, Irvine

CURRENT STATE OF THE ART

The main efforts in the evolution of high reliability general purpose systems have followed two paths. The first of these has been the interconnection of large computers in an effort to provide alternative computational paths and a variety of services. Examples of such systems are the ARPA network, the Bell Laboratories in New Jersey installations, and the eastern network of IBM 360/67 users. In effect, each of the units on such a network acts as a store and forward node using trunks as the communications method.

The other path has evolved in a local context and has illustrative examples at both a hardware and systems level. Its main characteristic is to provide a high speed bus between units of a computational system. The DEC PDP 11, the Collins C System and an IBM 360 multiplexer (also an example of this trend) have utilized a high speed bus to allow access to a wide variety of peripheral devices. In this approach there has been no system-wide use of this basically hardware-oriented facility. The work of H. B. Baskin of UC Berkeley[2] and certain Bell Laboratories efforts has been to produce total systems based on a loose coupling of computational units. Both of these approaches utilize a central controlling facility to schedule tasks and control functions.

It is our feeling that there are many advantages to be gained from a distribution of both user-oriented tasks and system processes over many units within a system. We also feel that having a variety of specialized mini-processors available will lower the effective costs to a user. It is to explore the consequences of this approach that we propose the Distributed Computer System.

---

[2] "A Modular Computer Sharing System", Herbert Baskin et al., Communications of the ACM, Oct. 1969

INTRODUCTION

The fundamental concept of the Distributed Computer System (DCS 1) is to elaborate on the autonomous component organization prevalent in modern computers. We will extend this idea into a loosely coupled adaptive environment of computational units capable of wide ranges of performance and high reliability.

In a typical DCS 1, there are a set of common and perhaps duplex facilities which service the file and storage pools accessible to participating machines. These files and storage pools are maintained and controlled by 'special' purpose processors tailored for this function. A participating system will request items of data to which it has access rights by sending a request along with both the requestor's identity and the name of the servicing unit. The request will be initiated by placing a message on the Data Ring. The data ring is a high speed looped data channel. At some time after the request, a message from the servicing machine will return, either notifying the requesting machine as to the status of the item, or giving the data. Similarly, a small processor could be built to have a small local store and to 'page' in from bulk common storage new pages as needed. In addition, one could have specialized processing units on the ring, such as processing units that do compiling, text editing, etc. Other units would send and receive work from the cooperating unit. Special purpose processors could be constructed and interfaced for data acquisition, display output, tablet input, etc.

The control of this environment will be accomplished by a set of processes. The fact that these processes will be executed on a variety of the processors of the DCS 1 illustrates the distributed features of the system.

There will be a number of supervisory processes to run on a subset of processors of the DCS 1. Four important processes are:

1) recognize-new-terminal
2) log-in
3) log-out
4) process-scheduler

We will briefly discuss the problems caused by our multi-processor (with possible parallel processes) environment. We feel that these problems are not substantially different from those found in any multi-processor-multi-tasking environment. This is not to claim that the problems are either easy or already solved, it is rather to state that we believe we understand the state of solution to the problems and the areas which need further investigation.

RESEARCH GOALS OF THE DISTRIBUTED COMPUTER SYSTEM PROJECT

A number of issues occur in the design of the Distributed Computer System. Each could be treated as a major area of research. In the DCS 1 project we will only attack the broad problems in those areas believed to be critical to the project. These include:

1) Effect of distributed control on performance and system flexibility.
2) Data Ring Protocols
3) Fail-safe design of both the communications hardware and the software system.
4) Transferability of data between different types of processors[3].
5) Statistical behavior of the ring environment including measurement of system performance.

---

[3] "RADC Program Transferability Study," T. Cheatham, D. Farber, G. Mealy, E. Morenoff, K. Sattley - REPT RADC-TR-68-431, Nov 1968, CFSTIAD 678 589.

The inter-process communications problem is in many ways a virgin territory for research. Some preliminary exploration has been done by the Multics[4], the ARPA Network [5, 6, 7], and the NPL[8] groups which have some direct applicability to the DCS 1 project. The basic plan used in this paper is drawn from work at the RAND Corporation [9] and represents a usable technique. Of all the issues to be investigated by the DCS 1 project, this should have the highest payoff in basic understanding.

In the prototype DCS 1 we can combine different processing units and devices with a minimum amount of effort, while meeting the goal of producing a reliable, easily extensible, computer environment. It will include the ability to provide a variety of small time-sharing subsystems and, at the same time, and on the same DCS 1, an environment usable by a variety of data-gathering experiments.

---

[4] "Inter-Process Communications," J. Saltzer, Multics System Programmers Manual.

[5] "ARPA Network Working Papers," S. Crocker.

[6] "A system for Inter-Process Communication in a Resource Sharing Computer Network," D. C. Walden, ARPA Working Group, Aug. 1970.

[7] "Multiple Computer Networks and Intercomputer Communications," ACM Symposium on Operating System Principles, Gatlenburg, Oct. 1967.

[8] "Communications Networks to Serve Rapid-Response Computer," D. W. Davies, IFIPS Congress, Edinburgh, Aug. 1968.

[9] "QGAM-A Queued Graphic Access Method," D. Farber and W. Josephs, The RAND Corporation.

GENERAL PHILOSOPHY AND ASSUMPTIONS

The philosophy of the system will be based on the tried and true principle of "keep loose." This, translated into the equivalent terminology of the computer field, says that things will change as you enter a major hardware-software project so it pays not to engineer too closely. In the case of the DCS 1, this will reflect itself in our unwillingness to design a system for the properties of a specific data ring, and in the lack of rigidity of the formats and assumptions about terminals and inter-processor messages.

A major area of application of the "keep loose" policy will be in the area of the actual data ring. There are a number of possibilities for the actual architecture of the ring ranging from the variable message length self-synchronizing system used by Farmer-Newhall[10] through the schemes currently being used at Bell Telephone Laboratories by Dr. H. McDonald's group. These schemes include both a polled fixed length message system and a larger fixed length non-polled system. The scheme in this paper fits best into the philosophy of the DCS 1 system since it requires no central directing agency to function. During the initial planning phase of the DCS 1 project the pros and cons of each scheme will be more fully explored.

Briefly then, the following description represents one of a number of possible ways of constructing the DCS 1. We have explored many of the alternative paths and will investigate several more before entering the development stage.

---

[10] "An Experimental Distributed Switching System to Handle Bursting Computer Traffic," W. D. Farmer and E. E. Newhall, Bell Telephone Laboratories; ACM Communications Seminar, Pine Mountain.

DESCRIPTION OF A PROTOTYPE DISTRIBUTED COMPUTER SYSTEM

We first set the scene for a discussion of each of the constituents of the DCS 1. Figure 1 illustrates a small DCS 1 capable of operating a small BASIC system. We propose this system as a prototype on which to test our ideas.

Ring Structure

Each of the processors and each of the terminals is attached to the ring by a node package. The node packages and the ring form the common communications system which tie together the processors to form a Distributed Computer System. The use of this communications system to achieve reliable, low-cost operation constitutes the main problem of the system design. The detailed definition of a node will be described later. At this time it is sufficient to describe certain characteristics of the node and the ring. The data ring is evolved from the T1 digital carrier system of the Bell Telephone System, which was developed in 1962 and has been in widespread commercial use since 1965. We believe that under the conditions that exist at UCI, the ring could be run at a rate of from 50-100 megabits/sec (similar PCM systems have been run at 200-300 megabits/sec).

Node Functions

The node has a number of functions to perform. When it is given a message to transmit, it will buffer the message from the processor, then wait until it finds an idle block of the correct length. It will then place the source identification in the block, set the block busy bit, and transmit the messages onto the ring. Upon transmission of a message and receipt of an acknowledgement, the node then notifies its attached processor that

it is available for use.

When the node is in receive mode, it looks at messages streaming past for its identification in the destination slot. It will accept its messages and check them for error. If they are error-free it will generate an acknowledgement response to the transmitter. If the message is in error, it will notify the processor attached to the node of this fact.

It may be necessary for the node to refuse to transmit for more than K units of time. After that, it will operate in receive mode, accepting only control messages in order to avoid hogging the ring.

Message Structure

Messages which are placed on the ring are of fixed lengths and will currently come in two sizes. The short message is designed for transmitting either a one character ASCII message or a control function, while the long message is for block transfers. The messages consist of a header, the message text, error checking information and an acknowledge bit. The header consists of a length code, a source identification, a destination identification, and a bit to determine whether or not the message block is in use. In addition, there is a need for a bit designating whether this is a control or an information message.

Control Messages

The function of the control messages is to affect the performance of the node. Control messages are not sent to the attached processor but are interpreted by the node itself. They will be used for status checking, maintenance, setting of destination in certain cases, and for traffic regulation. A full study of the set of control messages needed is not yet complete. However, certain ones will be mentioned in the remainder of this paper.

A DETAILED SCENARIO OF SYSTEM OPERATION

We now look in detail at the actual message flow as certain changes of state take place in the system. The messages and control actions induced by a state change will help illustrate both certain problems which need to be solved and how we expect the system to operate.

Let us look at the DCS 1 in an idling state with no terminals having power on and no non-supervisory processes running in any machines. At this stage of the DCS 1 operation, a number of supervisory processes must be run to insure system integrity. For instance, there is a process which requests a status response from each node attached to the ring. The result of asking for status will be to obtain a one byte message having, among other information, the power-on state of the processor attached to the node. There will also be processes which will test ring integrity by sending messages in a ping-pong fashion to each other. All of these processes will be periodically required to interrogate a central file work-to-be-done list via a scheduler process, in order to ascertain if there is a higher priority process than that currently being executed. It is also possible for a scheduler process to interrupt a process by sending a control message to the processor's ring node. Upon receipt of the control message, the node will activate a hardware interrupt line in the processor if such is available. This task might also be accomplished by sending a message to the process occupying the processor we desire to use.

Let us assume that a "terminal" has just turned power on. The status checking process will, during its periodic scan, note a change of status on that node. It will then send a message to a file machine which has access to the central process-waiting queues and system status messages.

The file machine will update the terminal power-on records and will enqueue a log-in process request with the identification of the node. It is necessary for one or more scheduler processes to be periodically activated within the system in order to service the process-waiting queues. Any idle processor may be loaded with a scheduler and, in addition, certain of the large machines and file machines may have resident DCS 1 schedulers. In the event that there are no idle machines and no resident schedulers, it will be necessary periodically to cause one of the processors to become idle by a DCS 1 interrupt message from one or more of the file machines, or from an internal clock interrupt from within the processor.

When a scheduler is active, its function, as in any system, is to scan the processes-waiting queues, choosing the highest priority process to be run. It then scans central records to ascertain which processors can be assigned to this process and then determines which of those is doing a process of lowest priority. Note that due to the wide variety of both processors and the things being done by processors, it is not realistic to expect that every process can be run on every machine. This may depend, for instance, on which language the process has been written in, or whether that processor has enough resources to run the process even if a version of the process is coded for it. Also, in certain cases, for instance in real time experiment monitoring, it is not feasible to interrupt the processor.

When a valid processor is found, a "swap" protocol is activated which will save the existing context of the process currently running on the chosen processor and place the log-in process into that processor. This swap protocol results in a swapping of the non-write-only portions of the process either to a local file or to a central file via the ring. It is

expected that most of the swapping will be done via the ring since this will allow re-establishment of the swapped process in another equivalent processor rather than in just the one it was swapped out of.

The scheduler then sends a message to the log-in process to tell it the identification of the node to be serviced. The log-in process then sends a control message to the terminal's node, changing the destination of all originating traffic to the log-in supervisor. The status of the node before the transmission of this message is one that precludes the transmission of outgoing information. A task of the log-in process is to identify the type of equipment attached to the node and, in addition, to ascertain the validity of that equipment on the ring. Therefore, there must be a formal and standardized "hand-shake" and verification procedure which will identify the device to the log-in process. On a teletype it is likely that this would be done via an answer-back drum and thus perhaps a first start on a standard protocol is for the log-in to send a "Who Are You" (WRU) to the device on the node.

The procedures from this point on still need careful examination since they bear on the human engineering of the system. In broad terms the originating user, which may be a teletype, graphic scope, or processor, needs to be able to perform classes of actions and receive the following classes of services:

1) Upon receipt of the information as to what type of equipment is attached to the node, it will be necessary, in general, to call on an intermediary to communicate to the device. This intermediary may be a module within the log-in process or it may be a process in another processor which will act as an "agent" for that device, hiding the peculiarities and

conventions of the device from the rest of the system. In this case, most likely found with graphics terminals, the log-in process asks for the availability of the agent process for this terminal. When the agent process is scheduled it sends a message to the log-in process giving the node number for the terminal and also places in the system records the fact that all future transactions for this node, in both directions, will be handled by the agent. The agent will, when ready, transmit to the terminal interface its node number. Thus, all messages in the system to and from the terminal will pass through an agent process.

2) Supply billing information, etc., from the terminal upon request.

3) The log-in process will communicate to the file processors the identification of the new "user." It is the function of the file process to define, in conjunction with the user, the rights, entitlements, domains, etc., of the user.

4) The log-in process will ascertain the services that the new user desires and will cause the scheduling of that service, e.g., BASIC, and then cause the attachment of the terminal to that process.

5) At some point the terminal may notify the process that has been servicing him that he desires to return to the log-in process for new services. The techniques required to do this at this time are rather similar to those that were used the first time with the exception of the need to do items 1 and 2.

OTHER ISSUES OF IMPORTANCE

Communications Philosophy

In the evolution of the design and in the creation of a system which is extendable and adaptable, a central issue is the philosophy of communications that will be used to tie together users with processes, and processes with processes. We will remove from the system structure the distinction between different devices for example, between a teletype and a processor. While it is true that they are used in different ways and have different capabilities, this is really a matter of local concern to the device, not to the system. For instance, a process residing in some processor should be able to "log-in" as a user to a BASIC machine and interact with it with little, if any, more difficulty than would a user at a teletype. The communications protocol must support this goal. An approach which looks promising for attaining this goal is proposed in a RAND Corporation report "QGAM, Queued Graphic Access Method," (1969)[9].

The basic philosophy behind the QGAM approach, and one we will follow in the design of our inter-process communications protocol, is that a process (a program or any other common definition will do) communicates with other processes only with permission granted by the receiving process. The establishment of this agreement is achieved by the process (A) which wishes to establish communications stating the desire to communicate with the other process (B) via a communications coordinator-process. The coordinator first looks in its records to establish whether or not B has granted prior permission a class of processes of which A is a member. If automatic permission has been granted, then a "connection" is established

---

[9] "QGAM-A Queued Graphic Access Method," D. Farber and W. Josephs, The RAND Corporation.

and A is told the name by which B will be known to it. At the same time, B is told that A has been given permission to talk with B. If automatic permission is not found, the coordinator notifies B of the request for connection and then B can either deny the request (this will result in an asynchronous interrupt of A), or can grant the request. B's normal condition is to be in a receptive state for messages from any source to which it has granted sending permission. In QGAM, however, it was permissible for B to allow receipt of messages at a given instant from only a subgroup of the allowed senders.

The following classes of message handling were allowed in QGAM:

Send to A only

Send to all destinations that A has a connection to

Receive from B only

Receive from all sources that A has a connection from

Connect to A

Disconnect A

This general approach seems to be a good method of handling the interprocess communications needs of the distributed machine.

FAILURE DETECTION AND RECOVERABILITY

Node Failures

In the communications system which connects the components of the DCS 1 there are a variety of conditions which can be classified as failures. At this point we will restrict our discussion to failures of the interface either to recognize messages meant for it or to fail to repeat messages meant for someone further down the line.

As has been stated before, one of the aims of the DCS 1 project is to insure a high degree of system reliability and individual availability for users. This goal, in conjunction with the desire to use a ring oriented interconnection system, requires that reliability features be designed into components of the system. We are dealing in the DCS 1 with a basically serial multi-component system. The serial nature of such a system normally implies that a failure in one component causes the system to cease to function. We must develop a schemum for which this will not be true.

We propose to design into each of the nodes, the following capability. There will be a standby primary circuit, and the choice of which primary is active is made either by a timing circuit within the primary, or via a control message which can be sent by a control diagnostic process operating in the DCS 1. In addition to the alternative primary, it may be necessary to provide a third state which is an active-repeater-only state. In this state, the system is completely isolated from the node secondary, and thus from the attached processor. (This is discussed in the section on traffic control.)

The operation of the timer controlled alternative circuit is as follows: Whenever a message header is detected, a timer in the node is reset and started. The duration of the timer is such that except under abnormal conditions a new message will arrive in the node before the timer times out. When there is a communications ring failure and there is a timer timeout, the alternative primary circuit is made the prime primary and vice versa. The net effect of this is to cause an alternate set of circuitry to become effective in certain nodes of the ring. A notification that switching has occurred will be made available to all the associated processors which can then generate messages to the supervisory process (via the file system) and then cause the appropriate system diagnostic processes to be loaded in some processing unit. The diagnostic process must determine which of the units was the one at fault. One obvious technique for doing this is to have the diagnostic process reset one primary pair at a time for every node in the system. When the non-functioning node is switched to its normal state, the system will cease to function. This is detected either by having the diagnosing processor send a message to itself and waiting for its receipt, or by the diagnosing processor receiving a timeout signal from its node. Note that when the node is in the alternate state, the timer will still be functioning. The component switching caused by the timer timing out will not take place in the alternate state, but the time-out signal to the attached processor will be produced. When the diagnosing processor detects a bad circuit, it can leave that node in its alternate state and continue the scan. Thus, more than one bad node package can be detected and at the end of the diagnosing scan, all nodes not faulty will be in their normal primary state. A similar strategy

is used if a traffic controlling task detects a node which is hogging the ring and does not respond to an order to decrease its volume of transmission. In that case, the diagnostic task or some other delegated task will send, either over an order wire or as a normal control message, the instruction for that node to switch to the active-repeater-only state, thus having the effect of isolating the processor from the system.

Security Issue

In this section, we will restrict our consideration of security to that of restricting a processor from carrying on a supervisory function unless it is capable of adequately protecting that function from deliberate outside manipulation. For example, if a supervisory process was allowed to reside in a processor available to arbitrary manipulation, it would be possible for a user to wait until the processor was loaded with a supervisory function, then stop the machine, modify the process and then restart the processor with a faulty process that might cause a system malfunction.

A general purpose timesharing system such as a SIGMA 7 or PDP-10 is most likely secure, as would be special purpose processors either with locked consoles or residing in the computer centers.

Since it is also necessary to prevent an unsecured processor from generating a control message to another processor, say invalidly shutting off that processor, we propose some hardware features which operate as follows: In each node there is a control-permitted bit which can be set by a control message from another node. Only if the control-permitted bit of a transmitting node is set can that node transmit a control message. Thus, a way exists to effectively prevent unauthorized

transmission of control messages. At system initialization time, certain nodes will be placed in the control-permitted state. When an invalid attempt is made to transmit a control message, the error interrupt will be raised in the node to the processor. This will also set a bit in the node available to diagnostic processes.

Traffic Control

In the version of the DCS 1 described in this paper, traffic congestion of available information slots on the ring can occur. As a prelude to discussing this phenomenon we should first discuss briefly the message acknowledgment protocols that can be used. Our approach to this problem is that of being prepared to make some sacrifices of bandwidth and throughput in exchange for a basic system simplicity and a small degree of logic complexity. We can make this sacrifice since our data transmission is short-haul rather than long-haul, and our use of local repeatered coaxial cable provides an unusually large bandwidth at a low cost. It is in this way that our system differs from such networks as the ARPA network, which is a long-haul network with basic communications costs so large that a large part of the design of the net was devoted to optimizing the throughput, and stored-program computers are required at each of the nodes. A simple acknowledgment system would be either to allow a transmitter to send a separate acknowledgment message or to set a bit in the received message and let that message circulate around the ring back to the sender. The sender would then notice that a message has arrived which it sent, and that an acknowledgment bit is set. At this time the original sender would release the

message slot by resetting the block busy bit. Note that it is possible for the message to return to the sender without an acknowledgment. This will occur if the receiving node is not in a position to accept the message due to the state of the attached processor. In this event, the message will circulate around the ring until either the node accepts it or it is removed by a "garbage collector" process. Traffic congestion from this source is even more apparent if we allow a transmitting node to send messages to a given destination before getting an acknowledgment of the receipt of prior messages. In this case, there is also the problem of correct sequencing of the received messages within the receiving processor. The complexity of multiple messages is in some measure compensated by the ability of the system to handle and absorb bursty types of traffic.

Other message acknowledgment protocols are possible. A traffic study and simulation is required to ascertain the penalties and payoffs of different approaches. We would expect to use the non-multiple message approach in the early stages of development because of its simplicity.

As was mentioned, garbage collection must be carried out if for no other reason than to detect that certain receivers are not operational. That is, if a receiver goes out of service, then messages destined for it will circulate on the ring indefinitely. In the case of a system which allows only one message to be transmitted at a time, garbage collection can be achieved, at a cost in the node, by having the transmitter limit the time between when the message originates and when it decides that the message will not be accepted by the destination.

In that case, the transmitter will mark the message block idle and interrupt the attached processor to report a failure to transmit. If the attached processor is incapable of responding to this, the node will set a distinction validity bit. This error indicator will be sensed periodically by a system validity process.

DUPLEX NODES

In the DCS 1 environment, the ideal arrangement would be to have certain nodes operating in full duplex, that is, both receive and transmit. This would insure that the node while waiting to transmit would be receptive to messages destined for it. In such a situation it would receive the message on the next pass, but this increases apparent ring traffic. In general, full duplex may be a rather expensive way to equip all nodes (this requires a cost study), but an alternative does exist for devices whose traffic characteristics demand it. Such processors would have two nodes on the ring: one used, by convention, for transmitting and the other one for receiving.

COAXIAL CABLE RELIABILITY

The nodes of the DCS 1 are connected by a coaxial cable. While this basic cable is extremely reliable, it is appropriate to guard the system against deliberate interference. At the same time, we shall also protect the system against the destruction of an entire node. The technique used can be called "leapfrogging." We will assume a DCS 1 with an even number of nodes (if the number is odd, insert a dummy node - that is a primary only). The leapfrogging is achieved by using a second cable composed of segments which connect node N to node N+2. The effect of this is to provide an alternate path bypassing an intermediate node (N+1) as well as the segments of the basic coaxial cable which connect nodes N to N+1 and N+1 to N+2. In the event of failure(s) of the basic cable, this alternate path(s) will be switched in by means of relays. The detection of which segment(s) has failed will be done in a similar manner to that used to correct a node fault. (See: Node Failures)

CONCLUSION

We have given a broad look into the goals and structure of the Distributed Computer System. A recapitulation of the goals and an indication of how we will attain them is in order at this point:

We are planning to attain:

1) a high degree of reliability - the failure recovery protocols on the ring, the multiple number of processors capable of running needed processes, the schema for effecting system recovery all contribute to this.

2) a variety of programming languages - the ease with which a new processor or process can be integrated into the DCS 1 environment will allow a variety of languages to be available to a DCS 1 user.

3) a modest system programming resource requirement and an incremental expansion capability - the structure of DCS 1 forces disciplines in the construction of processes and in the definition of clean interprocess interfaces which will minimize programming difficulties. In addition, the flexibility of agent processes will allow new devices to be added with a small programming cost for each. The nature of the communication interface will allow new devices to be interfaced to DCS 1 with a low equipment cost. In addition, there exists a remarkable degree of flexibility in the association of controllers and devices due to the high speed ring.

4) competitive costs - the current cost of a port into a time-sharing system ranges from $4,000 to $10,000 per port. We expect to be competitive with this and offer better reliability and flexibility.

We plan to have a small demonstration system composed of two processors and two terminals with a restricted set of capabilities operational within 12 months of the start of the project. The second phase, extending the number of devices and services, will be operational at 18 months.

ACKNOWLEDGMENTS

I should like to acknowledge the contributions of many University of California, Irvine students and faculty to the development of the ideas presented. Particular credit is due to Professor J. Feldman, Dr. M. Hopwood and Mr. C. Will for their many contributions.

PD-04162
5-19

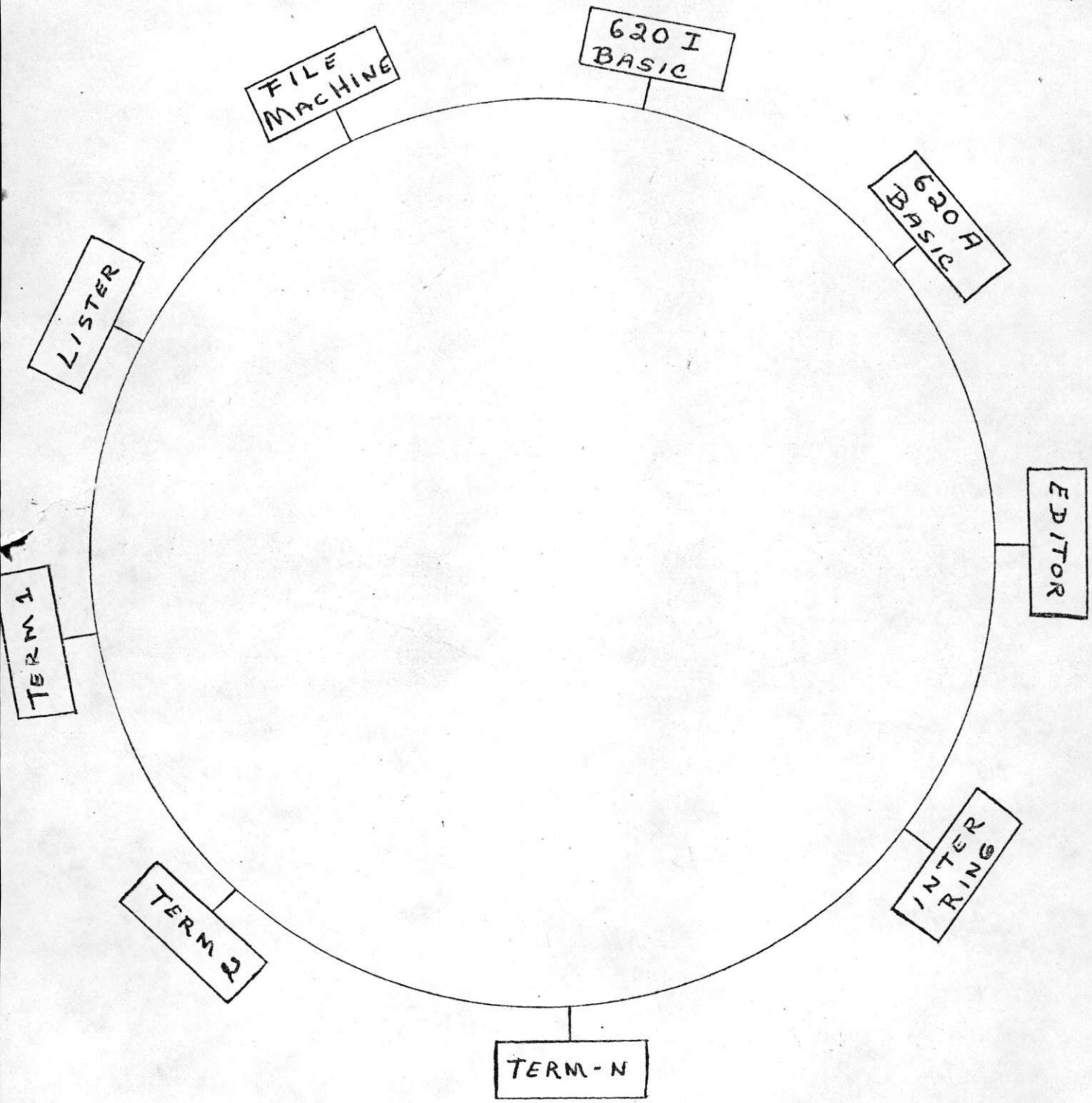

FIGURE 1 PROTOTYPE DISTRIBUTED MACHINE