A DISTRIBUTED COMPUTING APPROACH TO MISSION OPERATIONS SUPPORT

RONALD L. LARSEN
NASA's Goddard Space Flight Center
Greenbelt, Md.

Summary The ever-evolving nature of spacecraft Support operations requires computing systems which can adapt to dynamic support requirements. These systems must exhibit a high degree of reliability, perform a wide spectrum of services, and be able to grow incrementally in response to demand. A distributed computing network is proposed as a system architecture displaying the desired attributes. The proposed system employs functional specialization of resources, featuring high performance scientific processing, real-time response, and interactive operations. Fundamental to the approach is the notion of process-to-process communication, which is effected through a high-bandwidth communications network. Both resource-sharing and load-sharing may be realized in the proposed system.

Introduction The operational support of orbiting spacecraft requires coordination and cooperation among many groups of people and among the systems designed and operated by these people. At NASA's Goddard Space Flight Center, ground-based operations are conceptually and organizationally divided into five parts: communications, satellite operations, information processing, tracking and data network operations, and mission operations support computing. Each of these parts comprises autonomous computer systems, and data communication lines interconnect the five systems.

Within the context of this paper, mission operations support computing includes orbit determination, attitude (satellite orientation) processing, maneuver computation, resource scheduling, and a host of miscellaneous capabilities to respond to the unique requirements of a given mission. Of current interest is the form of the evolution of mission operations support computing. In particular, what type of computer systems will respond most effectively to a set of diverse and dynamic requirements? Overall system reliability is of prime concern, with incremental expendability the next priority interest. Processing requirements include high performance scientific computations, real-time telemetry and tracking data processing, and interactive terminal support.

A distributed computer network is presently being proposed as a viable means of fulfilling these requirements. Distributed computing is an approach to systems design which effects the sharing of network-wide resources. Computer systems are interconnected through digital communications facilities, and the resulting network appears to the user as one computing facility.

The concepts of distributed computing have been explored in several research, organizations recently, resulting in the development of prototype, experimental systems. Perhaps the best known effort is being conducted at the University of California at Irvine. They have constructed the Distributed Computing System (DCS), a set of minicomputers which communicate via a specially designed communications ring (2-6). The University of Maryland is constructing the Distributed Computer Network (DCN) using several DEC PDP-11's, a Univac 1108 dual-processor, and a Univac 1106 (10-11). The-February 1975 issue of Datamation contains several articles related to distributed computing. One of them (19) describes a rather esoteric design being developed at Carnegie-Mellon University involving up to sixteen DEC PDP-11 series minicomputers sharing up to sixteen memory boxes through a 16 X 16 switch.

This paper describes the philosophy behind the Distributed Environment for Mission Operations Support (DEMOS). Its prime technical design requirements include:

1. high performance floating point processing
2. real-time response
3. interactive operations
4. high data throughput
5. data base management
6. program development support (e.g. time-sharing)

An additional set of high level requirements are associated with the operations environment:

1. fail-soft reliability
2. modularity
3. incremental expendability
4. minimal cost
5. ease of maintenance
6. smooth transition process

Gross characteristics of the major hardware and software components are discussed, as well as their integration into a complete system.

Throughout the discussion, problem areas to which specific attention may be directed are noted. The paper closes with a brief account of several pragmatic factors influencing design choices.

Distributed Computing The broad spectrum of user requirements experienced in many installations has fostered the development of distributed computing networks. The most frequently stated objective of such a network is to promote the sharing of resources among a set of computer systems, which may be as close together as in the same room, or separated by thousands of miles. This approach has much to offer both those who supply services and those who use systems. Expensive resources such as plotters or proprietary software packages need not be replicated at each installation where demand exists. Instead, access can be gained to a given resource via the network. Information availability similarly can be enhanced through network facilities. A data base maintained at one facility can be accessed by other facilities with the currency of the information assured. Thus distributing resources among several processing sites and providing communication facilities between them is likely to make sense both economically and practically.

Distributed computing systems appear to the user as one (perhaps geographically distributed) computing facility. Resources available throughout the system are made available to a user without his needing to know where they reside. In fact, the processor residency of his own processes is transparent to him, and may be dynamic, illustrating a fundamental objective of this approach: to take advantage of those benefits which accrue when processes are allowed to migrate among the processors of a distributed system. These benefits include load balancing and increased reliability. A processor malfunction can potentially result in resurrecting its processes on other processors, and reconfiguring the system around the failing site. A distributed system also exhibits substantial potential for growth. New resources (processors, peripherals, etc.) can be introduced into the system incrementally to respond to changing demand. From the user’s point of view, he sees the same system as he is accustomed to, but now with increased power. Both growth and reliability in this case result from the dynamic nature of a distributed system. Resources can be added and deleted at will, with available services increasing or decreasing in response.

The design goals of such a system imply autonomous processing sites with distributed control. Each processor must be capable of functioning on its own, viewing the remaining system much as a user would: as a facility which can serve it. Of course, this relation is symmetric; each processor must be willing to serve the other processors also. Centralized control is precluded by the reliability requirement. Failure of the centralized control would introduce a single point of failure which could disable the entire system.

Emphasis in a distributed system is on process-to-process communication. This implies that a process need not (and probably will not) know the residency of a process with which it is communicating. The software and hardware communication mechanisms are constructed in such a way that messages are routed from the originating process to the destination process, whether those two processes exist in the same machine, or are on processors separated by thousands of miles. This type of communication facility is one of the key distinguishing factors of a distributed computing system. Not only user processes, but also system processes, and even portions of the operating system itself may be distributed among various processors.

Scheduling and resource allocation become more difficult in a distributed environment than in a single processor environment. Each site of a distributed system is responsible for the execution of those tasks presently in its queues. Two options are available. A site may schedule a task's execution on its own processor, or it may decide that the task's requirements can be better handled at another site. In the second event, the task is sent via the communication facilities to the alternate site. The destination site then must make the same determination. Should it keep the task, or send it on elsewhere? Obvious tradeoffs are involved here. Transmission costs must be factored into any realistic algorithm a site uses to determine the optimum site to process a task. It is entirely possible that a task sent to another site may be sent back to the originating site, or that a loop may develop, where each site in the loop is saying, "No, you do it!" Where transmission costs are substantial and delays significant, the problem becomes even more difficult, because each site's knowledge of the other site's operating conditions (e.g. queue lengths, resource availability) is likely to be out-of-date, and expensive to update.

DEMOS Hardware Mention was made earlier of the need for high performance processors for scientific applications. But we may observe that real-time requirements, interactive terminal support, and data management requirements do not need such a sophisticated unit. An immediate case can be made for at least two classes of processors. We will call the high performance processors "Class A" processors, and the data-oriented units "Class B." A Class A processor should probably have a word size of at least 32 bits and have double precision floating point hardware. A Class B processor is an information processor which should be able to support many external interfaces and specialize in input/output operations. These units are going to need to communicate with each other efficiently and simply, where efficiently means with minimal interference with their primary functions, and simply is meant to imply consistency and uniformity. We therefore introduce a third processor, Class C, whose specialty is communications. At least one Class C processor is associated with each Class A and Class B processor. Its sole function is to manage its associated processor's communications with the other units.

Figure 1 illustrates the organization of these processors into the DEMOS building block, which we call a cluster. The Class C processors are interconnected among themselves to form a high bandwidth communications network. The precise topology of this network is currently undetermined, and unimportant to this discussion, hence it is represented by the shaded area.

A DEMOS cluster is functionally tuned to a specific mode of operation, for example, orbital analysis. Its composition is completely variable. Any number of Class A processors may be included, as well as any number of Class B processors. The typical cluster is expected to include a total of six to ten Class A and B processors. This range of numbers strikes a reasonable balance between cluster complexity and composite processing potential. Figure 1, with its six processors, then represents a modest cluster.

The basic cluster exhibits many of the properties required of a mission operations support system. Its reliability is assured by the complete absence of critical points. The loss of any one processor does not disable the system. Its composition may be specified to provide any degree of processor redundancy desired. This same property enables maintenance to be performed by temporarily removing a processor from the cluster. In general, system availability and reliability is assured for hardware through multiple redundancy. A cluster can be configured to provide whatever degree of reliability is preferred (and afforded). As should be obvious, a cluster can be expanded or contracted at will, by adding or removing processors respectively. This provides the potential to grow incrementally in response to changing requirements, and to upgrade the system as improved processors become available.

A Class A processor will not be expected to maintain a large array of secondary storage. Certainly it may require swapping devices and storage for data its processes require immediate access to, but extensive file storage is more the job of a Class B processor. Similarly, terminal interfaces and communication into and out of a cluster are the responsibility of a Class B processor. A Class A processor can be correctly characterized as self-centered. It maintains that data its processes are most interested in, but in general is not concerned with other processor’s data requirements. A Class B processor, on the other hand, can be quite gregarious. Its concern is the welfare of the cluster. Data which the cluster as a whole has shown an interest in recently is maintained by the Class B processors of that cluster. Therefore, the secondary storage requirements of Class B processors could potentially be substantial.

The communications facilities must provide high bandwidth (at least a megabit per second) digital communications. The components of each cluster are expected to be geographically close to each other (e.g. on neighboring equipment racks), so that no modems should be required, and communication lines will probably be relatively short pieces of coaxial cable.

The Class C processors will need a modest amount of buffer storage and perhaps an associative store (content-addressable memory). The processor itself is expected to be a microprocessor.

DEMOS Software The basic function of any operating system is to manage the resources of the system. In the context of DEMOS this means sharing system-wide resources to effect load balancing. Queues of work building up at one processor should automatically be redistributed to lesser loaded processors. Emphasis in DEMOS is on performance, where performance includes throughput, response, and robustness. This stands in contrast to OS/360, where the emphasis was on functional scope. IBM's efforts to provide all things to all people resulted in reliability problems and an interminable series of new releases. DEMOS's goals are less lofty. We wish to design a system which can be counted on to reliably support mission operations.

DEMOS is to be a process-oriented system. This is perhaps one of the most important concepts in distributed computing. A process need have no knowledge of its own location or the location of another process with which it needs to communicate. It need only know the process's name, or some other equivalent identification. Further, hardware resources are assigned processes to manage them. A process which wishes to read a record from a tape, for example, does this indirectly by communicating with the process which manages the tape. This approach merges the resource allocation problem into the process management problem, facilitating a unified approach to each. In this environment, the interprocess communication mechanisms become a predominant factor and can mean the difference between success and failure.

Each DEMOS processor must maintain its autonomy. It can rely on no other processor for its own fundamental activities. Therefore each processor will have its own kernel system to support its basic machine interfaces. For example, processor scheduling is an independent activity on each processor. That process of highest priority will be allocated the processor whenever it becomes available. Note that the mechanism for setting priority need not (and probably will not) be a kernel function. In fact priorities may be determined by a process on another machine, or even a person external to the system. It is not the policy of setting task priority which is important here, but the management of processor time in accordance with a priority scheme. Memory management is a similar function. Each processor must bear the responsibility of maintaining in its main memory that set of information to which it needs quickest access. Many schemes have been suggested for memory management, but some form of segmented virtual memory with appropriate hardware assist (i.e. address mapping) seems particularly well suited to DEMOS. This point will be discussed more later; the important point is the locality of the function. Interrupt handling also must be done by each processor. Usually all three of these items: processor scheduling, memory management, and interrupt handling, are highly interdependent. Each processor must contain a kernel system which performs each of these bare, essential functions. Note that the kernel's primary function is to enable the processor to decide what to do next at a microscopic level.

Certainly the scope of services provided by DEMOS must be far more extensive than those discussed so far. Each processor must also support the interprocess communication by which we achieve virtualization of process residency. When a process sends a message to another process, that message is routed from the originating process to that processor's resident communications system. A list of resident processes is checked and if the destination process is resident on the same processor, message transmission can be facilitated easily by software. If, on the other hand, the destination process name is not found in the list, then the message must be sent into the communications network for routing to the appropriate processor.

Resource allocation is a non-trivial problem which is currently stimulating substantial research. The question is, of course, how does one processor, confronted with a process's resource requirements (which may be dynamic) evaluate which processor (at that moment) can best provide the desired service.

Protection is fundamental to any system. User processes must be protected from each other, and the operating system itself should be impenetrable. The most successful protection mechanisms suggested to date make it physically impossible for one process to gain access to unauthorized areas. This is accomplished by mapping a process's logical addresses into the memory's physical addresses. The appropriate mapping tables are inaccessible to the user, and generate address faults if the user attempts an unauthorized memory access. This form of protection can be implemented on a segmented or paged virtual memory. A complete discussion of protection is beyond the scope of this paper, and of course, protection itself extends beyond merely memory protection. One must also consider file access, and access to "privileged" instructions, for example. The interested reader is referred to [17] for an excellent discussion of protection.

Process synchronization and interlocking is another mechanism a distributed computing operating system must provide. These mechanisms can be implemented through the interprocess communication facilities, but are greatly facilitated by machine instructions such as the S/360 test and set (TS) which interrogates a bit, can change its status, and return an indication of its status all in one memory cycle. This type of instruction can be used to preclude two processes from inadvertently gaining access to a critical region of code concurrently. It is generally advisable to express any process interdependencies explicitly via appropriate synchronization primitives, thereby removing time dependencies. This becomes especially critical in a distributed environment where a divergent class of processors may be represented.

The overall integrity of a DEMOS cluster is the responsibility of each component processor. This is to say that each processor should be very suspicious of the other processors. Perhaps each one maintains an active process which tests remaining portions of the cluster. If two processors (say) both conclude that a third is malfunctioning, they may initiate a recovery procedure which reconfigures the cluster, removing the suspect processor and attempting to restart on another processor those processes on the failing machine. The malfunction could then be logged and the system operator notified of the contingency.

Support of a wide range of input/output devices has been a longstanding operating system function, and so it is in DEMOS. I/O support is generally provided through “access methods” which on one end present to the user a reasonably intuitive set of commands (e.g. OPEN, READ, WRITE, CLOSE) and on the other end cater to the idiosyncracies of a particular device. These routines are generally invoked through subroutine CALL or program-generated interrupt to the operating system. A DEMOS process could expect to invoke an access method through the interprocess communication facilities, since in general the process itself and the I/O units it may be using need not be on the same processor. Through the communication facilities, the physical location of I/O devices is made transparent to the user.

Data management services in DEMOS are constructed to serve a hierarchy of storage devices. At upper levels of the hierarchy one finds high speed, low capacity media (e.g. main memory); as one moves down the hierarchy, capacity increases as speed decreases. A typical hierarchy might put drum storage under main memory, then moving-head disks, then tape. What is important is that a user need not be aware of the physical device on which his data resides. The level of activity experienced by a unit of data influences at what level in the hierarchy that data is stored. Highly active data is kept near the top, where it may be accessed quickly. As data ages and is used less, it migrates to lower levels, until it reaches the lowest level of offline storage. From the user’s point of view, all data is accessible as if it were online. The access time he experiences, however, will increase markedly relative to the depth in the hierarchy at which his data resides.

DEMOS Communications The prime element of the interprocess communication mechanism is the network of communication lines and Class C processor which interconnects the Class A and B processors. To be effective, the Class C processor should accept as much of the communications burden as possible. Traffic between a host (A or B) and a communications processor (CP, Class C) should be limited to only communication which is of actual concern to the host. Therefore, messages addressed to processes not resident on the host should never reach the host. In addition, communication time from origin to destination should be minimized, and the overall success of the communication network should not depend on every CP functioning normally.

To achieve the first objective, i.e. shielding a host from superfluous messages, a CP must be aware of its host's activities. This could be facilitated, for example, through a list maintained by the CP of those entities resident on its host which are accessible to the network. Such a list could contain the names of processes currently resident on the host, and might include file names. Messages directed to these processes and requests for these files could be routed by the CP to the host. When a new process is started, the host would send its name to the CP for entry into the store, and when a process terminates or migrates to another processor, its name would be removed from the CP's store.

The topology of the network and the "intelligence" of the CP's must be matched in such a way that either a message will successfully reach its destination, or the originator will be notified of the failure. An acceptable topology may be very simple. A communication line could run from each processor to its immediate neighbors, for example, forming a "ring." A multipath topology ultimately promises more reliable direct communication with higher net bandwidth, but at higher communication cost and increased complexity of the CP., In the limit, the topology of the CP network could form a fully connected graph. In addition to being expensive, this approach makes growth very awkward, for each CP must be modified to accept a path to the new CP. Therefore one must design the CP in such a way that it provides an acceptable communications bandwidth, while maintaining needed flexibility to add or delete CP's from the network. Store-and-forward forms of communication should not be relied upon unless adequate fault detection mechanisms are present to detect a failing CP.

DEMOS System Architecture Having considered some basic aspects of DEMOS, let us now consider how a mission operations support environment may look. The basic component in DEMOS has been identified as the cluster. A cluster can be functionally tuned to perform a specific class of activities very well. It is therefore proposed that the computational activities of mission operations support be partitioned into classes, the elements of each class being functionally related and highly interdependent. Each of these classes would then be assigned to a cluster. Such an organization would be designed to focus data flow within the cluster. Interaction within each cluster would be deliberately high, while intercluster activity would be lower. Figure 2 illustrates such an environment.

Orbit operations, and attitude and maneuvers have been assigned separate clusters not because of a difference in function, but more from level of activity considerations. Each of these requires sufficient computational power to justify individual clusters. The Network Resource Scheduling Cluster, however, is functionally very different. Raw computer power is not as important as a highly interactive facility to monitor tracking network activity and respond to real-time tracking site requirements. The fourth proposed cluster specializes in data management and interfacing DEMOS to the outside world (e.g. the interface to the communications network linking DEMOS with Goddard's other operations groups). Most human interfaces (e.g. terminal devices) will be supported by the Support Cluster.

Perhaps the most important function of the Support Cluster is data management. All data going into and out of DEMOS goes through the Support Cluster. This localization of control over data is essential to ensure proper accounting for and the overall integrity of data. This cluster manages the primary data bases and the data archive for DEMOS. Any data worth saving, be it programs or data, is stored in this cluster. The data requirements of the other clusters are satisfied by sending them copies of the data they desire, and data they generate which is to be saved is returned to the Support Cluster. Clusters other than the Support Cluster need not overly burden themselves with elaborate provisions to manage data storage. Data may be discarded at will by them to free up needed storage space, since another copy of that data is guaranteed to reside in the Support Cluster.

Communications between clusters is supported in precisely the same way as communications within a cluster. Class B processors will typically host two Class C communications processors: one for intracluster communications, and one for intercluster communications. Clearly a hierarchical structure for communications is emerging. One can visualize this as a tree (Figure 3).

Processors within the same cluster comprise the leaves of the tree, and their communication can be effected very rapidly through the first level of the hierarchy. Intercluster communication forms the next level of the tree, allowing any DEMOS processor access to any other DEMOS processor, albeit at greater cost if the processors reside in different clusters. The Goddard computer communications network is tvte third level (going up) in the hierarchy, providing communication between DEMOS and other Goddard facilities. The fourth envisioned level would link all Goddard facilities with other networks (e.g. ARPANET). This level could potentially provide research scientists on an international scale access to experimental data in a much timelier fashion than is now provided (through the postal service, for example).

DEMOS Pragmatics In proposing any new system, certain realities must be recognized and addressed. First, present day operations are performed on several large scale S/360 systems. These systems provide not only operations support, but also general purpose computing for the Goddard community as a whole. How can they be incorporated into DEMOS? One solution may be to modify the S/360 operating system slightly to enable the 360’s to simulate a true DEMOS site. The feasibility of this approach has been demonstrated by the University of Maryland in integrating their Univac machines into the Distributed Computer Network. By adopting this approach in DEMOS, both sophisticated general purpose facilities of the S/360 machines and their high performance processing power become available to DEMOS users. At the same time, non-DEMOS users can continue to use the 360’s as they always have, oblivious of the machines’ role in a distributed computing environment.

The next point to be addressed is the type of processor ideally suited to being a DEMOS Class A or B processor. The economies of the minicomputer industry certainly make these machines an inviting choice. The recent announcement of 32-bit architecture in top-of-the-line models makes the prospects even more appealing. Careful attention must be given to the specific architecture of these machines, however, such as the addressing structure employed. Availability of support software (e.g. compilers, utilities), or rather the lack of same, is also a point of concern. A minicomputer user today must be prepared to work in a rather austere environment, largely lacking the software facilities taken for granted on the large-scale systems. One who is willing to commit to a substantial development effort, however, will potentially enjoy a system which can evolve with the user, growing to meet new requirements incrementally without the trauma of periodic total system replacements.

Conclusions In this paper an alternative to conventional large scale third generation computer system architecture has been discussed. It has been shown that distribution of resources and control results in increased reliability and exhibits potential for incremental growth. Through functional specialization a distributed system may be tuned to very specific operational requirements. Such an approach allows one an alternative to selecting a computer system on the basis of the most complex function he needs to perform, at the cost of mediocre performance overall. As technology advances and requirements change, a distributed facility can evolve gradually.

As this approach is yet young, many open problems still exist. The type of operating system to be designed, and the algorithms to be employed are largely unanswered questions. Communications technology is just entering an era where the sophisticated communications of a distributed environment are realized and economically feasible.

Despite the substantial development necessary to realize DEMOS, the benefits which can be expected to accrue make that development worthwhile. The feasibility of distributed computing as a concept has been demonstrated in numerous research projects. Technology is now to the point where one can seriously propose application systems based on these concepts. Without a doubt in the near future many systems such as DEMOS will in fact be proposed, designed, and implemented.

REFERENCES

1. Denning, P. J., “Virtual Memory,” Computing Surveys, vol. 2 no. 3, pp. 153-189; September 1970.

2. Farber, D. J., “A Ring Network,” Datamation, vol. 21 no. 2, pp. 44-46; February 1975.

3. Farber D. J., and Heinrich, F. R., “The Structure of a Distributed Computer System - The Distributed File System,” Proc. International Conference on Computer Communications, pp. 364-370; October 1972.

4. Farber D. J., and Larson, K. C., “The Structure of the Distributed Computing System - The Communications System,” Proc. Symp. Computer Communications Networks and Teletraffic, Polytechnic Press, New York, N. Y., pp. 539-545; 1972.

5. Farber, D. J., and Larson, K. C., “The System Architecture of the Distributed Computer System - The Communications System,” Proc. Symp. Computer - Communications Networks and Teletraffic, Polytechnic Press, New York, N. Y., pp 21-27; 1972.

6. Farber, D. J., et.al., “The Distributed Computing System,” Proc. Seventh Annual IEEE Computer Society International Conference, pp. 31-34; February 1973.

7. Hansen, P. B., “Operating System Principles,” Prentice-Hall, New Jersey; 1973.

8. Kahn, R. E., “Resource-sharing Computer Communications Networks,” Proc. IEEE, vol. 60 no. 11, pp. 1397-1407; November 1972.

9. Larsen, R. L., “A Conceptual View of the Distributed Environment for Mission Operations Support (DEMOS),” NASA Goddard Space Flight Center, X-document (in press).

10. Lay, W. M., et.al., “Design of a Distributed Computer Network for Resource Sharing,” Proc. AIAA Computer Network Systems Conference; April 1973.

11. Lay, W. M., et.al., “Operating Systems Architecture for a Distributed Computer Network, “Proceedings of the 1974 Symposium, Computer Networks: Trends and Applications, pp. 39-43; May 1974.

12. Mills, D. L., “Communication Software, “Proc. IEEE, vol. 60 no. 11, pp. 1333-1341; November 1972.

13. Pyke,.T. N., and Blanc, R. P., “Computer Networking Technology - A State of the Art Review,” Computer, vol. 6 no. 8, pp. 12-19; August 1973.

14. “Support of Air Force Automatic Data Processing Requirements through the 1980's,” (SADPR-85) Volume III, Technology, Appendix VI, including Annex A, NTIS No. AD-783-768; June 1974.

15. Tsichritzis, D. C., and Bernstein, P. A., “Operating Systems,” Academic Press, New York, N. Y.; 1974.

16. Walden, D. C., “A System for Interprocess Communication in a Resource Sharing Computer Network,” Comm. ACM, vol. 15 no. 4, pp. 221-230; April 1972.

17. Watson, R. W., “Timesharing System Design Concepts,” McGraw Hill, New York, N. Y.; 1970.

18. Withington, F. G., “Beyond 1984: A Technology Forecast,” Datamation, vol. 21 no. 1, pp. 54-73; January 1975.

19. Wulf, W., and Levin, R., “A Local Network,” Datamation, vol. 21 no. 2, pp. 47-50; February 1975.

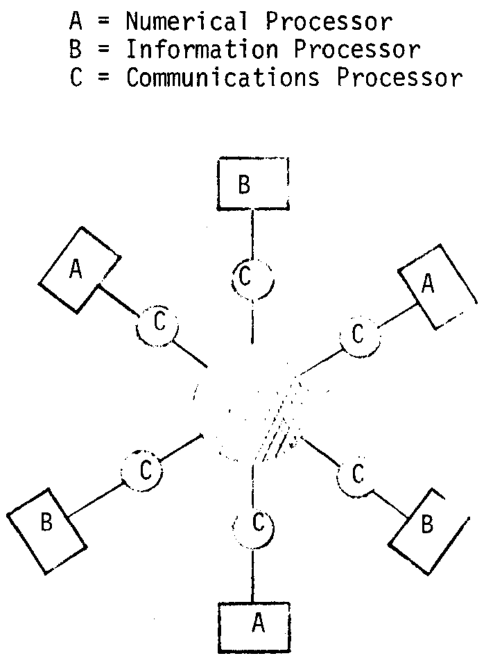

Figure 1. DEMOS Cluster

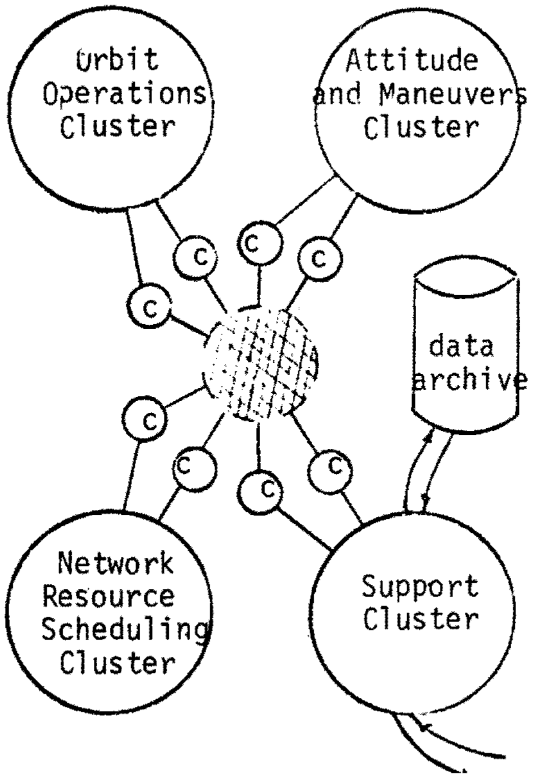
Orbit Operations Cluster
Attitude and Maneuvers Cluster
data archive
Network Resource Scheduling Cluster
Support Cluster
C

Figure 2. DEMOS Computational Environment

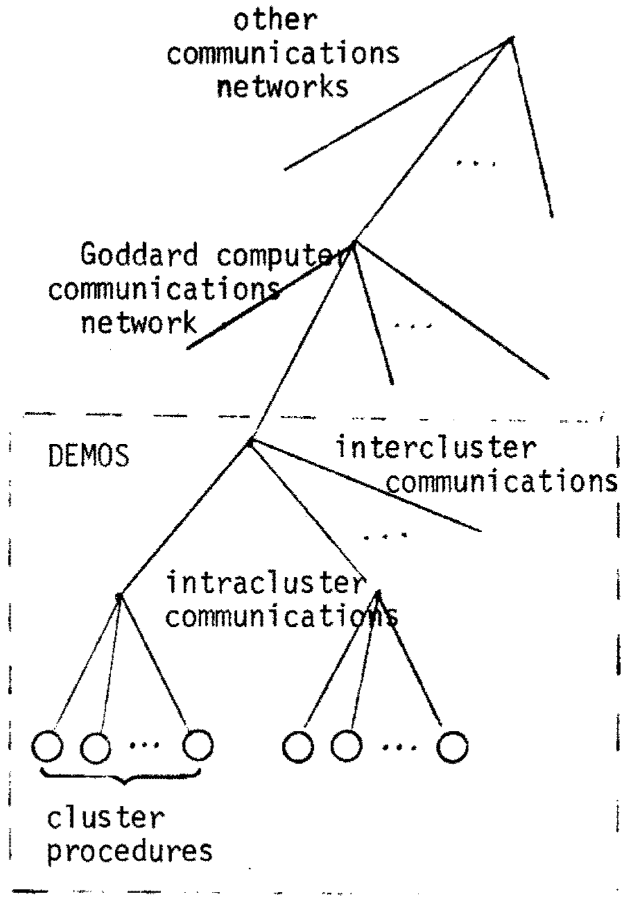
other communications networks
Goddard computer communications network
DEMOS
intercluster communications
intracluster communications
cluster procedures

Figure 3. Communications Hierarchy