Issues in Distributed Processing - An Overview of Two Workshops

Richard H. Eckhouse, Jr.
Digital Equipment Corporation

John A. Stankovic
Andries van Dam
Brown University

Two workshops on distributed processing were held at Brown University in 1976 and 1977, sponsored by the Army Research Office, the National Science Foundation, and the Office of Naval Research. The workshops had three goals:

(1) to classify various ongoing research efforts, to identify salient characteristics of distributed processing, and to propose standard terminology;

(2) to establish what constitutes the state of the art, to discover common areas of research, to exchange specific solutions that might be generalized, and to determine which techniques worked (or did not) and why;

(3) to identify problem areas and to indicate fruitful directions for future research.

As might be expected, we were only partially successful. This summary is intended to convey the spirit of the meetings and to provide an overview of the important technical interchanges. The actual detailed account of the sessions is recorded in the full transcripts, cited in the editors' overview to this issue. Participants' names and affiliations are listed in the table.

Themes and viewpoints

As Philip Enslow points out in his article in this issue, the field of distributed processing currently suffers from the lack of a precise definition of the term. A frequent comment throughout the workshop was, "But that's not distributed processing!" We felt the meaning of the term "distributed processing" should identify a specific set of research problems and issues, and much argument therefore centered on identifying such salient issues and trying to establish whether they were new or unique to distributed processing. There are many dimensions (aspects such as processors, data, and control) of a system which may be distributed—as Enslow again discusses—and each dimension really has a spectrum of values; therefore, where each person draws the line between distributed and nondistributed systems becomes largely a matter of personal interpretation.

Even though the researchers could not agree among themselves, they generally agreed that industry's notions of what constitutes distributed processing are not rigorous enough. Decentralized computing with minicomputers or intelligent terminals connected to a mainframe, in a star or hierarchical configuration, is certainly a valuable technique. The participants did not consider this to be distributed processing in the general sense, because either control remains centralized, or frequent interaction between cooperating processes on both machines is lacking. Similarly, the existence of a subnetwork over which processors communicate does not in and of itself mean the system performs distributed processing.

Participants generally agreed that distributed processing is made possible by the price-performance revolution in microelectronics. It has its historical roots in satellite minicomputers that relieve the load on mainframes, tightly coupled multi- and parallel processors, and loosely coupled networks. Many of the advantages desired from distributed processing systems, such as increased performance and availability, are therefore familiar from studies and implementations of multiprocessor systems. The amount of technology transferred from these fields, however, seems inadequate. Many participants agreed that the single characteristic distinguishing distributed processing systems from classical architectures was decentralized system-wide executive control, a concept about which very little is known. Douglas Jensen's article in this issue discusses such control in more detail. Other participants claimed distributed processing could occur at any level—architectural, operating system, or application.

The ability of a distributed processing system to provide extensibility seemed to be an important theme. Basic building blocks might be processor-memory pairs built from conventional microprocessors or specially designed modules or "cells." The number of modules needed would be connected to a communication

subnet of arbitrary topology, such as a bus, ring, or other form, with hardware-supported facilities for interprocessor and interprocess communication.

Participants also agreed that the field is still so young that little formalism and few fundamental principles have emerged. Most of the papers and discussions dwelt on specific techniques, mechanisms, and experimental results. As more experience is gained and the field matures, more attention will presumably be paid to methodology and the systematic study of design tradeoffs.

Major issues

Interprocess communication. Interprocess communication was a dividing issue at both workshops. The argument centered on the comparison of two forms of message communication; on one side, between processes that have disjoint address spaces, and, on the other, processes that share memory.

In the Honeywell Experimental Distributed Processor (HXDP), which Jensen's article describes, there is no sharing of main memory; each processor has its own private memory. All interprocess communication occurs via explicit messages along a common bus. All processes are considered equally remote from one another, so all messages are transmitted on the global bus, even if they are between a source and destination in the same processor. This type of interprocess communication is sometimes said to be inherently inefficient when compared to memory sharing, the inefficiency being the main disadvantage of a message mechanism. However, in HXDP this inefficiency is ameliorated by implementing the message mechanism in hardware. Software implementations of similar message mechanisms[1] may include means to detect when the destination of the message is in the same host as the source and, if so, to bypass the transmission of the message onto the subnet, thereby increasing interprocess communication efficiency.

The chief advantage of the message mechanism, which Enslow regards as a main requirement for distributed processing, is that it permits autonomous processes to cooperate independently of their location in the distributed processing system.

The Computer Modules (Cm*) Project at Carnegie-Mellon University[2] has a flexible interprocessor memory access scheme. The Cm* architecture consists of clusters of microprocessor-memory pairs sharing memory segments that have uniform, protected references to objects (both data and program via a "capability" mechanism. This capability mechanism forms the lowest level of abstraction upon which both memory sharing and message interprocess communication are built. In those instances in which memory sharing is utilized for interprocess communication, access to the shared memory is still controlled by the capability mechanism. To use the message mechanism one merely sends information to a particular kind of segment, a message segment. The receiving process shares this message segment and is able to read the message. Once again the system insures proper access via the capability mechanism. Transmission of messages is very fast since the capability to read a message is transmitted rather than the physical data itself (which has variable length and is typically larger than the capability).

There was much debate about the relationship between logical and physical interconnections ("coupling"). Some participants felt that on the communication level there was a distinct difference between a message-based system and a shared-memory system. The difference has a profound effect on the level of cooperation between processes and hence the range of suitable applications. In a message-based system, explicit cooperation may be required from software at the destination; such cooperation may be desirable, if it helps keep the system running in the event of a hardware component failure by localizing errors. In contrast, in a shared-memory system a process has access to data belonging to another process without its explicit cooperation. In the event of failure, the consequences could be widespread and catastrophic. A consensus evolved that the important distinction between interconnection mechanisms is not the degree of coupling (e.g., tight or loose), or its bandwidth, but rather the protocols for communication, and efficiency versus protection tradeoffs.

Message broadcasting is another important point. The issue is not so much whether a broadcast capability is needed, because a mechanism can be included in almost any configuration. The real question is how much effort is needed or warranted to produce an efficient broadcast mechanism. "Are we going to use broadcasting as a basis for building our systems?" We should examine our goals to determine whether an efficient broadcasting facility is required at the lowest system levels. If it is only

Table 1. Participants who presented papers. (Previous affiliation shown in parentheses.)

SHAHID BOKHARI	UNIVERSITY OF MASSACHUSETTS
OLIN BRAY	SPERRY UNIVAC
CARL CHRISTENSEN	BELL LABORATORIES
ALAN CLEARWATERS	NAVAL UNDERWATER SYSTEMS CENTER
JOHN COOPER	U.S. DEPARTMENT OF THE NAVY
N. X. DANG	INSTITUT NATIONAL POLYTECHNIQUE DE GRENOBLE
D. JULIAN DAVIES	UNIVERSITY OF WESTERN ONTARIO
JOHN DAY	UNIVERSITY OF ILLINOIS
RICHARD ECKHOUSE	DIGITAL EQUIPMENT CORP.
PHILIP ENSLOW	GEORGIA INSTITUTE OF TECHNOLOGY
DAVID FARBER	UNIVERSITY OF DELAWARE (UNIV. OF CALIF., IRVINE)
PETER FEILER	CARNEGIE-MELLON UNIV.
JAMES FOLEY	GEORGE WASHINGTON UNIV. (UNIV. OF NORTH CAROLINA
SAMUEL FULLER	CARNEGIE-MELLON UNIV.
ROBERT GORDON	PRIME COMPUTER, INC. (NAVAL UNDERWATER SYSTEMS CENTER)
JAMES HAMILTON	DIGITAL EQUIPMENT CORP.
LEONARD HAYNES	OFFICE OF NAVAL RESEARCH (NAVAL SURFACE WEAPONS CENTER)
ELMAR HOLLER	KERNFORSCHUNGSZENTRUM KARLSRUHE
E. DOUGLAS JENSEN	HONEYWELL, INC.
GERARD LELANN	IRISA, UNIVERSITE DE RENNES
MING T. LIU	OHIO STATE UNIVERSITY
MAMORU MAEKAWA	TOSHIBA R&D CENTER
ERIC MANNING	UNIVERSITY OF WATERLOO
ROBERT MILLSTEIN	MASSACHUSETTS COMPUTER ASSOCIATES
JOHN MORRISSEY	IBM CORPORATION
DAVID NELSON	PRIME COMPUTER, INC.
RICHARD PEEBLES	DIGITAL EQUIPMENT CORP. (WATERLOO UNIVERSITY)
HAROLD STONE	UNIVERSITY OF MASSACHUSETTS
HOWARD STURGIS	XEROX CORPORATION
CARL SUNSHINE	RAND CORPORATION
RICHARD SWAN	CARNEGIE-MELLON UNIVERSITY
ANDREW TANENBAUM	VRIJE UNIVERSITEIT, AMSTERDAM
ROBERT THOMAS	BOLT BERANEK & NEWMAN, INC.
ANDRIES VAN DAM	BROWN UNIVERSITY

to be used occasionally, it can always be realized at the higher levels, as in Arpanet, by sending distinct messages to all necessary processes.

Distributed operating systems. Today there are at least two general approaches to designing and developing distributed processing systems. One way is to interconnect and integrate already existing computational resources (usually heterogeneous) to increase utilization and resource sharing. Another way is to design and build the system from scratch using off-the-shelf hardware and software components.

The Arpanet is an example of the first approach; its building blocks are traditional heterogeneous computer systems. Distributed processing on the Arpanet requires that processes running under native operating systems on their respective hosts cooperate through the communication subnet and various layers of communication protocol. RSEXEC and NSW, network operating systems built on top of existing host operating systems, present a uniform access mechanism to the various Arpanet host computer system resources. They are described in more detail in the article by Forsdick, Schantz, and Thomas in this issue.

Two prominent examples of the second approach are the previously mentioned HXDP and Cm*, whose operating system was designed to support both parallel and distributed computing.[3] The Cm* operating system provides (a) capability operations, (b) message operations, (c) environment creation, and (d) module creation and loading. This multilevel modular operating system is written as a normal user program. Its hierarchical structure permits larger systems to be built, in which the interconnections and dependencies of the different levels grow linearly rather than exponentially with the size of the system.

The major distinguishing characteristics of a distributed operating system —system-wide control of all resources without global, centralized state information—was widely discussed in the two workshops, but no participant could offer examples of such novel systems or algorithms for providing such control. For example, Arpanet network operating systems (NSW and RSEXEC) are only partially decentralized and do not control all of the host resources; Cm*'s first operating system contains centralized state information; HXDP's operating system is being designed as a fully decentralized executive but is not yet implemented. However, the Arpanet communication subnet is cited as a simple model for fully decentralized control; it uses local routing tables in each IMP and an adaptive routing strategy, in which the routing tables are continuously updated to reflect the probabilistically optimal routing through the net. The DCS bidding scheme in which resource allocation is handled by having each processor's kernel respond to resource allocation requests based on the knowledge of its own availability may also provide insight. Finally, the beginnings of a formalism for the difficult notion of many operating systems cooperating to provide resource management with each having only a probabilistic knowledge of the global state is being developed.[4]

Distributed data bases. The successful implementation of many distributed systems requires solutions to problems of data management in a distributed environment. These problems include data allocation, concurrency control and update processing, failure recovery, and query processing.

Given data access patterns, a set of communication and storage resources, and performance criteria such as availability requirements, how can one allocate data objects to locations in a way that minimizes some cost function, such as access time? The problem in general was not discussed much at the workshops, but is receiving adequate attention in the distributed data base community.[5] A special case in which the design can take advantage of geographic locality of reference of data accessed by users was discussed extensively, and is further described in the article by Peebles and Manning in this issue.

The coordination of concurrent updates initiated by multiple users must be done in a way that preserves the consistency of the data base. The lack of a centralized locking mechanism, together with communication delays inherent in a distributed environment, makes the required synchronization more difficult than in a centralized environment.

Most of the presentations in the distributed data base area described approaches to aspects of this problem; Peebles and Manning discuss it further. The presentations included a timestamp algorithm, to ensure that each data base manager program acts on the update requests in the same sequence; a majority consensus algorithm, under which updating of all copies takes place if a majority of data base manager programs agree to it; and a design in which multiple slave sites are coordinated by a single master site for update processing. Other presentations described a distributed locking mechanism and problems of deadlock detection and avoidance.

When failures occur in system components or in intercomponent communications, the system must operate in a resilient manner. Such failures must not destroy the integrity of the data base. Ideally, the system should continue functioning, although perhaps in a degraded mode.

Also presented was a solution to a problem of failure recovery in a distributed environment[6] which involves organizing update activities into transactions that, though not themselves atomic—incapable of subdivision—can be treated as such in a limited sense. Those transactions that update the data base either make no change in the stored data or carry out the entire write, even if a crash occurs in the middle of the write. Peebles and Manning also discuss this approach.

Finally, locating stored data is not a simple task. The system program must determine where the data required by a query is stored; for a query involving data stored at several locations, it must also determine an efficient procedure for gathering and merging the data required to satisfy the query. Work is only beginning in this area.

Load balancing. The problems of automatic load balancing—that is, optimally assigning tasks to processors—are analogous to the optimal data allocation problem mentioned previously. While little useful work seems to have been done in the general case, some researchers have investigated graph theoretic and queuing algorithms, which find the optimal assignment of modules to maximize performance or minimize costs in the rather special but useful case of two-processor systems. The algorithms are further restricted to sequential execution of program modules. This restriction is quite acceptable in a time-shared host with one or more dedicated satellites, systems for which these algorithms were developed. In general, the application programmer writes

programs for a "virtual uniprocessor," without worrying about where a module will reside. However, he may bind a module to a particular processor if, for example, the module requires some resource unique to that processor. The uniprocessor abstraction provides transparency of distribution similar to that achieved by a network operating system.

Research has been done on the problem of dynamically reassigning ("migrating") a module from one processor to the other during program execution as a function of load on the host. It has been shown that entire clusters of related procedures migrate from host to satellite, as the load on the host increases. These migrations take place only at a few critical "breakpoint" values of host load. The cost of this migration can be computed by an extension of the graph theoretic assignment algorithm. Experiments have also verified simple algorithmic predictions of the degree of performance improvement permitted by migrations at these breakpoint values.[7]

An optimal assignment algorithm for the three-processor case has been developed, but it does not seem to generalize to n processors. Despite these useful techniques for simple systems, solution of the general case of the assignment problem and development of algorithms for program decomposition and partitioning remain as major tasks for designers and users of distributed processing systems. Furthermore, it is not even clear what criteria should be used for n-processor load balancing.

Interconnection structures. The advantages and disadvantages of different physical interconnection structures were topics of lively debate at the workshops. An Arpanet-like topology is considered reliable if at least two paths are available between each source and destination. Message routing and flow control algorithms, which prevent saturation of the subnet, play an important role in the efficient operation of this type of interconnection structure. Rings are less reliable, although they can provide a second path, which is utilized only if a break occurs in the first path. Routing and flow control algorithms in rings are considerably less complex than in more general interconnection structures.

It was also argued that despite superficial topological differences, rings and buses are similar in many respects. They both may have decentralized control; both may be two-address systems; both are essentially the same in terms of message synchronization, sequencing, clocking, and routing. One difference noted was that message removal from the subnet is done actively in a ring by a program as part of the protocol whereas a cable terminator does it passively in a bus.

Interconnection structures for tightly coupled systems are important but were not discussed to any great extent. Little agreement was reached in discussing these issues except that more work on comparisons and evaluations of different interconnection structures was required.

More fundamental research is necessary to determine how the goals of distributed processing should affect the architecture and interconnection structure of processors. Too often investigators are forced to utilize existing, off-the-shelf components in their research. Since such equipment is not designed for distributed processing, it limits progress toward the goals of distributed processing. Quite possibly, to attain a cost-effective distributed processing system, we must explore hardware/software tradeoffs that are quite unconventional by today's standards—as Jensen's article in this issue points out.

Open research areas

Numerous problems still face us today in distributed processing. The following informal outline of these problems and short statements on the state of the art are in no particular order, and are far from complete. Furthermore, it is too early to tell whether many issues are inherent problems, or whether they will be solved or finessed by rapidly evolving technology and understanding

- Both better definitions and a better taxonomy are needed to categorize designs and experiments, and to provide a common context in which they may be discussed, compared, and evaluated.

- No complete systems analysis or synthesis methodology exists, either for distributed processing or for simpler, centralized systems—although there does exist a small collection of tactics in the areas of physical and logical interconnections, resource allocation, and fault tolerance, detection, and isolation. We also know some of the advantages and disadvantages of different topologies, restricted almost exclusively to hierarchies, stars, buses, and rings (loops).

- What attributes of a problem make it suitable or unsuitable for distributed processing? How does one weigh the advantages and disadvantages of distributed versus centralized systems?

- How is reliability incorporated into a distributed processing system in terms of cost and performance?

- New algorithms are needed, to solve generalized problems in reliability, which is intimately related to redundancy in resources. In turn, however, redundancy causes problems with consistency, synchronization, maintenance, security, and other factors. Existing algorithms do not adequately solve more than specialized cases. In addition, we need a language in which to talk about reliability and a way to measure it.

- How does the system detect, categorize, isolate, and recover from a failure? How do we treat multiple failures? How much redundancy is needed and which resources require it to guarantee that recoverability is possible? Providing hardware recoverability is easier than providing it in software, but both are needed in distributed processing. Recoverability is a difficult research problem, especially for decentralized software doing error recovery. Hardware may be needed to assist in recoverability, as it is in error confinement.

- Protection and security may be easier in remote locations because those locations are isolated. On the other hand, "thin, noisy communication pipes" may make security difficult between such locations. Hardware assistance will probably be required.

- In modeling and simulation for distributed processing, existing techniques of queuing theory and graph theory are thus far inadequate, both in terms of their representational ability and their computational tractability. Today, inputs to simulations are poor. To get meaningful results one actually may have to build the distributed processing system, measure it, and then put these measurements into a simulation. Such a requirement defeats a major purpose of the simulation.

However, Petri nets and similar formalisms offer some promise in the modeling of asynchronous, concurrent execution of cooperating processes.

- User-level control of distributed processing requires a language containing a message protocol with process synchronization primitives, as opposed to an Algol-like "block structured" language with calls. The primitives required to program in a distributed processing environment have not, by and large, been identified.

- Who partitions and assigns programs and data in a distributed processing environment? What criteria are used? What is the resolution of the partitioning—that is, how finely are the modules divided? What is the persistence of the assignment, and when does it occur?

- Research is required in distributed operating systems to determine efficient methods of implementing decentralized system-wide control. In such a system, no one process has more than probabilistic knowledge of the global state. Almost all present operating system principles assume that the entire global system state is centrally located and available, except for some specialized I/O information held by peripheral controllers. In distributed processing systems, the state information is decentralized, and its totality may be incomplete and inconsistent with the status of the entire system at any particular moment.

- Resource management takes on an added dimension of complexity in distributed operating systems as well. And, as discussed earlier, how do you perform automatic load sharing and balancing? using what criteria?

- More precise analysis of the various means of interprocess communication is required (shared memory, messages, procedure calls, etc.).

- Further investigation of broadcast versus point-to-point (circuit or message switching) communication is needed, although it is currently receiving some attention. An important question is how to broadcast to an unknown number of destinations and still receive definitive acknowledgements. Other issues in communication research include traffic types, topology, routing, flow control, reliability, error control, cost, modeling and analysis, and protocols.

- Interprocess communication in a heterogeneous network environment must be simplified, but differences in internal data representations present a problem that, as yet, has not been satisfactorily solved. If communication involves translation, there still exist problems in precision, and data representation and data type incompatibility. Among the more difficult problems are formats for passing programs, passing pointers, and passing the semantics (meaning) of data.

- Protocols for interprocess, interprocessor, and interuser communication, and file transfer present numerous problems. Their design and implementation must render them resilient to failure and abuse by aberrant or malicious software. Techniques are required for formal specification and verification that the protocol is correctly implemented. Verification must include proof that deadlocks are not possible or can be solved.

- Many other problem areas not peculiar to distributed processing gain added complexity in a distributed processing system. Among these are synchronization of shared variables, cost/benefit analysis, and the need for debugging aids and tools. ■

1. David J. Farber and Kenneth C. Larson, "The System Architecture of the Distributed Computer System—the Communication System," *Proc., Symposium on Computer Communication Networks and Teletraffic,* Microwave Research Institute, Polytechnic Institute of Brooklyn, April 1972, p. 21.
2. R. J. Swan, S. H. Fuller, and D. P. Siewiorek, "Cm*—a Modular, Multi-Microprocessor, *AFIPS Conf. Proc., Vol. 46,* 1977 NCC, p. 637.
3. Peter Feiler, "Implementation Issues of a Distributed Operating System, Notes on the Cm* Operating System," Distributed Processing Workshop, Brown University, August 1977.*
4. Gerard LeLann, "Distributed Systems—Toward a Formal Approach," IRISA, Universite de Rennes, France, August 1977.
5. Donald M. Austin, editor, *Proc., Second Berkeley Workshop, Distributed Data Management and Computer Networks,* Computer Science and Applied Mathematics Department, Lawrence Berkeley Laboratory, University of California, Berkeley; also National Technical Information Service. U.S. Department of Commerce; May 1977.
6. Howard E. Sturgis and Butler Lampson, "Crash Recovery in a Distributed Data Storage System," Distributed Processing Workshop, Brown University, August 1976.*
7. Janet Michel and Andries van Dam, "Evaluation of Performance Improvement in a Host/Satellite Distributed Processing System," Distributed Processing Workshop, Brown University, August 1977.*

*These papers appear in summary form in the proceedings of the respective workshops. For details, see the full papers, copies of which are available from the authors.

Acknowledgments

The authors benefitted from the hard work done by the session recorders on the full transcripts on which these summaries are based, and from conversations with Douglas Jensen and Robert Thomas about the highlights of the workshops.

Richard H. Eckhouse, Jr., is manager, computer systems architecture research, with Digital Equipment Corporation's Research and Development Group. Concurrently, he is an adjunct associate professor in the Computer and Information Science Department at the University of Massachusetts. His research activities include operating systems enhanced through microprogramming, virtual computer systems, multiminicomputer systems, and metalanguages applied to operating system semantics. Besides his many contributions to the field, he has recently published a book entitled *Minicomputer Systems: Organization and Programming.* He has consulted with several industrial and academic institutions. An author and consultant, Eckhouse is an active member of the ACM, besides serving as a reviewer and session organizer/chairman for several national conferences. He holds a BEE, MSEE, and PhD in computer science.

John Stankovic and Andries van Dam are guest editors of this issue. Their photos and biographies appear on p. 11.