M.I.T. LABORATORY FOR COMPUTER SCIENCE
Computer Systems Research Division

December 2, 1976
Request for Comments No. 132

TRIP REPORT: WORKSHOP ON DISTRIBUTED FAULT TOLERANT SYSTEMS

by Liba Svobodova

Recently I attended a workshop on Distributed Fault-Tolerant Computer Systems. This memo summarizes and analyzes those issues and problems discussed at the workshop that might be of interest to our group. The workshop was sponsored by the IEEE Technical Committee on Fault-Tolerant Computing and the Technical Committee on Mini-Micro Computers. The purpose of the workshop was to explore the relation between distributed computing and fault tolerance. The interests of the people assembled at the workshop ranged from the design of fault-tolerant switching circuits to the problems of reliability of data base systems. The workshop program covered a variety of issues, but the whole notion of distributed computing and, consequently, what reliability problems are solved and what new problems are introduced by distribution, remained somewhat vague.

The distributed systems discussed at the workshop fall basically into two classes:

- multiprocessor systems where each processor can handle any of the tasks performed by the system,
- functionally distributed systems where processors are dedicated to specific functions.

Various tradeoffs were investigated along these lines. While pooling of processors in a multiprocessor system seems to be more desirable from the standpoint of hardware reliability, functional distribution is seen as a possible way to reduce software complexity and, consequently, improve software reliability. Multiprocessor configurations have been used as a means for achieving fault tolerance for many years; multiprocessor systems provide redundancy required

for fail-safe operation. Functionally distributed system seem to be attractive for special purpose applications where programming could be reduced essentially to a control of the functional units (it sounds somewhat like Dennis' Packet Flow Architecture). It was agreed that the problem of distribution ought to be further investigated to determine:

1) What are the primary reasons for distribution beside fault-tolerance, and,
2) Is the distribution selected under these pressures also optimal from the point of fault-tolerance?

One reason for going into distributed processing is the advantage of physical distribution. Physical distribution of computational facilities provides an additional level of fault tolerance, namely, robustness in view of a physical destruction of a part of the system. The need for such robustness has been discussed in our group in relation to data bases that, because of the importance of their contents, may be a potential target of attack or sabotage. Perhaps a more obvious situation where such robustness is important is a highly automized control process in a possibly destructive environment. Military applications, of course, belong to this category.

At first glance, it seems to be desirable that such a physically distributed system would operate as a "multiprocessor" system, that is, the rest of the system would absorb processing of the tasks normally handled by the destroyed processing module. The system would permanently reconfigure itself and continue operating, though with lower performance. It such a system were functionally distributed, physical destruction of one processing module would lead not to performance degradation, but functional degradation. But even such systems have their justification, especially where a computing facility is only an element of a larger and more complex system, where the correct operation of the system depends on more than just the functions performed by the computing element. In such systems, distribution of computing facilities should be based on the structure of the containing system. For example, in new airplanes where most of the control functions will be computerized, it is planned to have specialized computational modules that will be located close to the corresponding sensors and control hardware. If the sensors are destroyed, it does not matter if the attached processor is destroyed too, since there is no need to continue related computations.

Other reasons for going into distributed processing quoted by workshop participants included:

- easy expansion (and reduction) of the system,
- size advantages (for example, n small modules can be more easily fit into the available space in an airplane than a single large computer with an equivalent capacity).

Functional distribution was said to result in less complex software, facilitate independent debugging of individual modules, make realization of hardware changes easier, and give better performance than multiprocessor-like arrangement.

Fault tolerance in distributed systems was discussed mostly in terms of redundancy, malfunction containment, reconfiguration, and also physical distribution as summarized above. A different set of considerations was presented by Harris Liebergot from Sperry Univac. He defined a "distributed" system as a system with distributed intelligence; such a system could be possibly implemented as a single physical piece of hardware, but would be built of logically distinct microprogrammed units. Replacement of hardwired logic by microprogram lowers the cost for error detection, since error detection reduces to parity checking on the control store but, as a tradeoff, microprogrammed logic is slower. If a system were built of microprogrammed units operating in parallel, cost of error detection could be reduced without substantial loss of performance.

Functional distribution could be viewed as an extension of modular decomposition used in software design, where a module implements a single abstraction. Roy Levin and Fred Pollack from Carnegie-Mellon proposed a unified approach to software and hardware design, based on three principles:

- mutual suspicion, that is, both software and hardware modules use non-trusting interfaces,
- information hiding, that is, only the abstract specification of a module is known to other modules, information about the actual implementation is not accessible,
- fault detection on module level.

Roy Levin discussed mostly the problems of hardware design, with the conclusion that mutual suspicion had been completely ignored in the PDP/11 design. Fred Pollack talked about use of the information hiding principle for design of fault tolerant software. His claim that some fault tolerant techniques can be better implemented in software stimulated a lengthy discussion on the advantages and limitations of hardware and software implementation of fault tolerance.

Most of the research in fault tolerance assumes faults in the form of physical hardware failures or software and hardware design errors. The third possible source of faults is the system input. In general, a system is required to operate correctly even if the input contains errors, be the errors accidental or deliberate. Correct operation, however, can have several different interpretations. In some cases, the system is required merely to detect an error in the input and report it. In other cases, the system may be required to produce a correct result in spite of an error in the input. One example of the latter approach was a presentation by Victor Lesser from Carnegie-Mellon. He uses redundancy in the input data and applies AI techniques to handle errors and uncertainties in perceptual problems. In control applications, the computer module interfaced to a variety of analog devides might be the most reliable element in the complete system. Thus the overall reliability of the system could improve if the computer was able to tolerate errors in its input. Finally, the system's environment must be regarded as a source of faults. The environmental "errors" can cause physical failures of hardware components, induce errors in the input, or, as already discussed, completely destroy a part of the system.

To what extent a system must tolerate different types of faults depends of course on the system's function. To organize and relate the variety of issues and systems discussed at the workshop, Jack Goldberg from SRI drew a space of fault tolerance issues in terms of fault type, criticality, degree of tolerance, maintenance, and redundancy. The criticality and the required degree of tolerance are determined by the system's function. The criticality is measured in terms of what is at stake in case of a failure: human life, organizational life, or economics. The degree of tolerance could be assessed on three levels: correctness, fail safety, and availability. Jack Goldberg used this classification scheme to outline the fault tolerance issues in a distributed data base system. The following is his description:

Dominant fault types:	Breakdown in hosts and communication
	Poor data entry
	(both may be accidental or deliberate)
Criticality:	Organizational life
	Economics

Degree of tolerance: Correctness of small fraction of data
Access to semantically related data (fail safety)
Continuity of information flow (availability)

Maintenance: Human back-up for lost information (operational)
Automatic house keeping (off line)

Redundancy: Low average order of data replication
Intrinsic computational redundancy

This may be a fair description of a typical data base system, but there are important cases that involve different issues. The criticallity concerning human lives is generally associated with real time control systems, but human lives might be also threatened by releasing sensitive information to wrong hands. As for the maintenance, it may not be possible to rely on human back-up for lost information due to the large volume of information processed or because there is no human element involved!

Reliability of a system is its ability to provide a specific level of functionallity in a particular environment. John Meyer from the University of Michigan argued that reliability analyses should be based on the behavioral view of the system rather than the usual structural view. The system can "fail" in many ways internally, in its structure, yet externally still exhibit satisfactory, or at least tolerable, behavior. Such "partial success" is difficult to express in the structural model.

Consider a simple example. From a structural view, reliability of a distributed system can be expressed as a probability that a given subset of modules (processors, memories, communication links) is working correctly. From the behavioral view, reliability may have to be expressed in terms of a certain function or functions being performed within time T. Thus, functionally reliability means not only availability of computing resources but also speed of execution. Also, from the behavioral view, reliability requirements often change as a function of time: the system has to operate correctly only when actually in use, which may be only a fraction of its total life time.

The behavioral view of reliability analysis is applicable to the problem of consistency in data base systems with multiple copies of data files. We all agree that in practical applications perfect consistency of data base is not necessary. The reliability of such a system can be defined in terms of the level of inconsistency in the data base. The system performs "correctly" as

long as the level of inconsistency does not exceed a specified limit. Theoretically, the methods used in reliability analysis are applicable here, but the real problem is how to model the system, its environment, and the inconsistency relation.

Finally, in my short informal presentation, I described the types of distributed systems considered in our research, with emphasis on the type of "fault tolerance" required:

1) The system consists of self-contained members that have agreed to cooperate in creating a coherent system at some specified functional level. It must be possible to physically disconnect a member from the rest of the system; the rest of the system must continue operating correctly, but possibly with degraded functionality.
2) Each individual member must be able to continue operating correctly in case of a failure elsewhere in the system which may result in a complete isolation of the member from the rest of the system; however, such failure may possibly cause a degradation in the system functionallity.
3) While availability of information is important, it is not always critical; however, an unauthorized release or damage of stored information may be critical.

As the last issue, I would like to discuss the relation between reliability and performance. In the literature on performance evaluation of computer systems, reliability (mostly in the sense of availability) is often considered to be a measure of performance. However, it seems to be more appropriate to regard performance (in the sense of execution speed and throughput capacity) as a reliability issue. That conforms with the previously discussed notion of behavior-based analyses of system reliability. From the behavioral view, reliability means correct execution of permitted operations, refusal to perform operations that are not permitted, and performance.