STANFORD RESEARCH INSTITUTE
Menlo Park, California 94025 · U.S.A.

A STUDY OF FAULT-TOLERANT COMPUTING:
FIRST SEMI-ANNUAL TECHNICAL PROGRESS REPORT

by
Peter G. Neumann
Jack Goldberg
Karl N. Levitt
John H. Wensley
Computer Science Group
Stanford Research Institute, Menlo Park, Ca

25 August, 1972

ARPA Order Number
1998, 27 December 1971

Program Code Number
2P10

Name of Contractor
Stanford Research Institute
Menlo Park, California 94025

Effective Date of Contract
12 January 1972

Contract Expiration Date
14 January 1973

Amount of Contract
$149,700.00

Contract Number
N00014-72-C-0254

Principal Investigator
Peter G. Neumann,
Phone 415-326-6200,
ext. 2375

Scientific Officer
Director, Information
Systems Program
Mathematical and Information
Sciences Division
Office of the Navy
800 North Quincy Street
Arlington, Virginia 22217

Short Title of Work
FAULT-TOLERANT COMPUTING

Sponsored by and prepared for the
Defense Advanced Research Projects Agency
Arpa Order Number 1998

The views and conclusions contained in this document are those of the authors and should not be interpreted as necessarily representing the official policies, either expressed or implied, of the Defense Advanced Research Projects Agency or of the U. S. Government.
(Form Approved Budget Bureau No. 22-R0293)

Approved:

David R. Brown
David R. Brown, Director
Information Science Laboratory

Peter G. Neumann
Peter G. Neumann,
Principal Investigator

SRI Project 1693

DOCUMENT CONTROL DATA - R & D

(Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)

1 ORIGINATING ACTIVITY (Corporate author)

Stanford Research Institute

2a. REPORT SECURITY CLASSIFICATION

Unclassified

2b. GROUP

3 REPORT TITLE

A STUDY OF FAULT-TOLERANT COMPUTING:
FIRST SEMI-ANNUAL TECHNICAL PROGRESS REPORT

4 DESCRIPTIVE NOTES (Type of report and inclusive dates)

6-month technical report covering 12 January - 1 August 1972

5 AUTHOR(S) (First name, middle initial, last name)

Peter G. Neumann, Jack Goldberg, Karl N. Levitt, John H. Wensley

6 REPORT DATE

25 August 1972

7a. TOTAL NO OF PAGES

62

7b. NO OF REFS

15 + 62 in appendices

8a. CONTRACT OR GRANT NO

N 000 14-72-C-0254 (ONR)

b. PROJECT NO.

c.

d.

9a. ORIGINATOR'S REPORT NUMBER(S)

9b. OTHER REPORT NO(S) (Any other numbers that may be assigned this report)

10 DISTRIBUTION STATEMENT

Approved for public release; distribution unlimited

11 SUPPLEMENTARY NOTES

None

12 SPONSORING MILITARY ACTIVITY

Defense Advanced Research Projects Agency

13 ABSTRACT

This report describes technical progress in the first half year of a study of fault-tolerant computing. Steps toward the development of economical fault tolerance and high availability are discussed. Appendices contain a survey of various existing systems and system designs, as well as a paper on a hierarchical framework for fault-tolerant computing systems.

14 KEY WORDS	LINK A		LINK B		LINK C	
	ROLE	WT	ROLE	WT	ROLE	WT
Fault-tolerant computing						
Computer reliability						
Computer availability						

A STUDY OF FAULT-TOLERANT COMPUTING: FIRST SEMI-ANNUAL TECHNICAL PROGRESS REPORT

by
Peter G. Neumann
Jack Goldberg
Karl N. Levitt
John H. Wensley
Computer Science Group
Stanford Research Institute, Menlo Park, Ca

25 August, 1972

TABLE OF CONTENTS

A STUDY OF FAULT-TOLERANT COMPUTING:
FIRST SEMI-ANNUAL TECHNICAL PROGRESS REPORT

by

Peter G. Neumann
Jack Goldberg
Karl N. Levitt
John H. Wensley
Computer Science Group
Stanford Research Institute, Menlo Park, Ca

25 August, 1972

1. TECHNICAL REPORT SUMMARY

This document provides the first technical progress report in the one-year study of fault-tolerant computing which SRI is carrying out for ARPA under Contract Number N00014-72-C-1254. The first half year of the contract is covered.

1.1. PURPOSE OF THE PROJECT

The general objectives of our study are to evaluate and to advance the state of the art in fault-tolerant computing systems. The scope of the study includes theoretical as well as practical considerations. The major task areas are

(1) to survey and evaluate existing systems (and system concepts) and relevant existing theory;

(2) to define and evaluate new directions for the development of computing systems with high availability and extensive fault-tolerance capability, with low cost.

We are seeking guidelines for the design and development of highly economical systems with long life, near-perfect fault tolerance and extremely high availability. Understanding of the tradeoffs among these goals is also being sought.

1.2. PROGRESS TO DATE

The study has progressed on many fronts. The following efforts are considered in some detail in Sections 3.1 to 3.11, respectively.

(1) Survey of systems and system designs for fault tolerance
(2) Bibliography of relevant literature
(3) Investigations into the causes of failure
(4) A hierarchical framework for fault tolerance
(5) Fault-tolerant memories
(6) Fault-tolerant processing
(7) Reliability modelling
(8) Operational implications
(9) Reliability tradeoffs among time, space and complexity
(10) Other efforts

References are included to six project documents /1-6/.

1.3. CONCLUSIONS TO DATE

We feel at this point that the goal of obtaining significant fault tolerance at costs substantially less than duplication or triplication of hardware can be met under a wide range of operating requirements. Particularly in large systems with somewhat flexible real-time constraints, the cost can be quite low.

Numerous useful techniques exist, some of which have been carefully studied in recent years. Various newer techniques which we are investigating also seem promising. Framed-burst coding (in which bit clusters, or "frames", are treated together -- see Section 2.5) seems particularly appropriate for various advanced technology memory organizations. This requires only a logarithmic increase in the size of memory for single-frame burst error correction (and a small increase in the overall cost due to encoding and decoding). Sparing of chips on a frame basis, or of blocks of memory, is also highly effective. In memory-dominated systems, selective replication of critical processing capability may be used without greatly affecting the logarithmic cost increase. This is greatly facilitated by the hierarchical framework mentioned in Section 2.4, which appears to be very promising in other respects as well. Interspersed on-line diagnostics are very important, as are reliable reconfiguration with sparing and other concepts such as reliable (e.g., distributed) power supplies. In general, distributed logic-in-memory designs also hold some promise.

No fundamental gaps in the state of the art have been uncovered that prevent the attainment of high degrees of fault tolerance. On the other hand, up until now fault tolerance has generally implied substantial cost. For example, hardware cost increases by factors of two, three and four due to fault tolerance are common. (The overall cost increases may actually be higher, if the storage required for fault-tolerance software is considered along with the increased execution time.) The relatively high cost has been a consequence of some limitations in the art that have hindered the attainment of economical fault tolerance. Notable among these is that it has not been possible to avoid considerable replication of hardware. As mentioned above, we feel that this can now be surmounted to a considerable degree, except in systems with critical real-time requirements on uniformly correct performance for all outputs. Nonuniformity of constraints and requirements can be used to great advantage in system design. Another limitation involves the ability to achieve practical systems with long unmaintained lifetimes. This too now seems surmountable.

In addition, there are still some gaps of understanding, e.g., concerning space-time tradeoffs, the relative efficacy of replication versus coding in arithmetic and logical operations, etc. Furthermore, we are keenly aware that the problem is not just one of good hardware design. The work of this project is aimed at providing guidelines for the design of good economical fault-tolerant systems. Thus although good hardware design is paramount, our work must also consider implications of the system design on simplifying the operational and human aspects, which play a critical role in keeping a system highly available. This includes considerations that affect the fault tolerance and the continued availability of the operating system, and those that lessen the critical dependence on skilled operators and maintenance personnel.

On the basis of the work thus far, we expect significant and favorable results over the second half year of this project. We anticipate that the final report will include a carefully balanced integrated approach (or family of approaches) toward achieving economical systems with high availability and fault-tolerance. It is possible that as a result of this project we will recommend further work toward the detailed design and evaluation of a specific system. This system would presumably be for a particular class of applications, such as the network interface systems for a multi-computer network (e.g., the ARPA network).

The implications of this research on the Department of Defense include the following. Systems that are extensively fault tolerant can now be implemented. Such fault tolerance can be employed to yield error-free computation with rather long maintenance-free life. The cost of fault-tolerance is greatest when all outputs have critical real-time constraints. Otherwise space-time tradeoffs permit considerable economy. Especially for large computing systems, it is possible to achieve the goals economically. High availability is a somewhat simpler matter, and can be achieved with a much smaller cost increase.

2. PROBLEMS IN FAULT-TOLERANT COMPUTING

A system that is designed for fault tolerance, high reliability and/or long life may embody various of the following functions:

(1) Detection of errors,
(2) Prevention of error propagation,
(3) Location of faults,
(4) Replacement of faulty units,
(5) Rolling back of the system and/or the applications programs.

With static fault tolerance (e.g., fault masking), the above functions are implemented at once, in that a coding or voting mechanism handles all five functions (the last two trivially). With most dynamic techniques, particularly those carrying out real-time control, the above functions are performed in rapid sequential order (e.g., within 10-100 msec of the occurrence of the fault). In systems with less critical real-time requirements, there is a longer time period possible for these functions.

The five functions above are suggestive of the following problem areas.

(1) Techniques for detecting errors due to faults involve the use of redundancy (in equipment and/or in time) to check the validity of a unit. Coding techniques efficiently handle the situation for memory and simple arithmetic units. A problem exists with regard to arbitarary control logic, for which duplication has been the primary vehicle for fault-detection. We have been investigating three possibilities with respect to control logic. The first approach involves realizing most of the control portion of the processor as a memory function. In essence this involves a hierarchy of microprogramming techniques wherein the lowest levels embody the highest speed. Memory coding techniques are used for error detection, except for the residual arbitrary logic which is handled by duplication. The second approach also relies upon microprogramming, but here a combination of equipment and time redundancy is utilized to check a computation. With each microprogram is associated a checking microprogram that is executed following the main microprogram. A third approach relies

upon data-dependent error detection. That is, we are looking at coding techniques for logic such that if a failure occurs it is detected with some probability p, dependent on the inputs. Thus if p is close to 1 on the average and if the failure is permanent, that failure will eventually be detected. We have also been investigating problems on the use of feedback in detection.

(2) The prevention of error propagation takes several forms. One involves the use of replication and voting, another the use of error-correcting coding. Another approach is to resort to delay, by refusing to give any output at all until a guaranteed correct output can be obtained.

(3) The location of a failure requires that some form of diagnostic procedure be carried out subsequent to the failure detection. With regard to memory failures we have been looking at frame-error-locating and correcting codes and at more conventional diagnostic procedures. Similarly we have briefly investigated byte-locating arithmetic codes.

(4) In approaching the problem of faulty unit replacement, the first step is to identify a suitable partitioning of the system. The partitioning can be accomplished at the level of an entire processor (albeit a small one), or at the level of memory blocks, or of frames within a word. For this case the replacement is quite easy, requiring only a modification in the address. However, reliable switching is very important. For the computer utility application, a large processor may be too expensive to represent a viable discardable unit, so that a greater number of smaller processors may be desirable.

(5) With regard to real-time systems, most proposed fault-tolerant systems incorporate single-instruction rollback. General-purpose utilities usually let the user worry about his own rollback. For real-time applications, automatic program restart and system restart are essential. Fortunately, the nature of the environment usually permits them to be implemented easily. For the utility environment it also seems that user-invisible rollback can be effectively achieved. For example, a compiler might select rollback points where the pertinent status information can be automatically

checkpointed. However, there is a severe problem in recovering from major catastrophes such as power failures and certain critical hardware malfunctions. System rollback is clearly more difficult in an unknown environment.

(6) Analysis of system design is another important problem area. Evaluation, e.g., via design verification, modelling, simulation, is particularly important in terms of reliability, fault tolerance and availability.

(7) The development of systems with critical constraints can be difficult. There is a need for structured system design to facilitate the development process. The hierarchical framework of Section 3.4 aids greatly in this respect.

3. SUMMARY OF PROGRESS

The following paragraphs describe progress on specific work being conducted under this project. Essentially all tasks are directly related to the goal of obtaining as much fault tolerance as possible for as little cost as possible, commensurate with the nature of the system requirements.

3.1. SURVEY OF SYSTEMS AND SYSTEM DESIGNS FOR FAULT TOLERANCE

A census has been made of fault-tolerant systems and system designs. A first version is given here as Appendix I, and includes a very superficial summary of each system. To be able to represent systems in a more-or-less canonical way, we have designed a questionnaire (included in Appendix II), which we have sent to architects of most of these systems. The replies received thus far are given in Appendix II. Most of them contain significant detail, and permit ready comparison of the various goals, motivations, principles, techniques and achievements. The design of the questionnaire itself has exposed many dimensions of meaningful comparison among systems and reflects many different design approaches based on widely varying goals and constraints. (An earlier version of the survey was distributed at the Second International Symposium on Fault-Tolerant Computing, held in Boston, June 19-21, 1972. It formed the basis for a well-received panel discussion entitled "Approaches to the Architecture of Fault-Tolerant Computing", chaired by Jack Goldberg and including John Wensley as a panelist.)

In order to classify the numerous fault-tolerance architectures, we have selected three categories of systems, corresponding to three roughly disjoint applications areas:

(1) General-purpose computing utilities,
(2) Ground-based special-purpose systems
(3) Aerospace systems

The processor power and total system cost are more or less decreasing from (1) to (3), as is the size of memory required. The degree of preplanning possible for the computations generally increases in this order. The reliability and availability requirements usually increase in criticality from (1) to (3). (RELIABILITY is the probability that the system will perform satisfactorily for at least a given period of time when used under stated conditions. AVAILABILITY is the probability that the system is operating satisfactorily at any point in time when used under stated conditions, where the total time considered includes operating time, active repair time, administrative time, and logistic time...RELIABILITY ENGINEERING, ed. W. H. von Alven, Prentice Hall, 1964, pp 14-15.)

Most of the prior research efforts have been devoted to category (3). We feel that the problems in this category are basically solved. Replication, multiprocessing, coding, and sparing are commonly found techniques. While the relative cost of fault-tolerance is high in many of these systems, this is not always necessary. We feel that many of these systems may be over-engineered.

The situation is less well developed for the other two categories, which still seem relatively primitive (cf. the Census of Appendix I).

Intuitively, the relative cost of fault tolerance could be substantially less than in category (3), because of the looser constraints, and because of the possibilities of advantageously using averaging effects in bigger systems. This report tends to justify this statement.

3.2. BIBLIOGRAPHY OF RELEVANT LITERATURE

In March 1968 R. A. Short published a rather comprehensive bibliography /7/ (containing 347 references) which resulted from an SRI study for NASA. He is now helping us augment that bibliography with about 500 additional references. A system of descriptors and cross-indices is expected to be used which will greatly enhance the usefulness of the bibliography. References to systems mentioned in this report are found in Appendices I and II.

3.3. INVESTIGATIONS INTO THE CAUSES OF FAILURES

Early investigations have led to various (sometimes obvious) conclusions regarding the significant sources of system failures.

* Magnetic core memories operated with low access times (one microsecond or less) are major sources of system failures. Primary memory still dominates most systems (especially large ones) with respect to cost, size, and sources of unreliability.

* Peripherals are still a problem, although good system design should be able to prevent peripheral failures from "crashing" the system. (Several well-known systems are quite sensitive in this area, due to poor system design. For example, a system should be able to survive errors in reading most files, a capability which is facilitated by hierarchical design.)

* In several technologies transient or intermittent faults are more significant (and more common) than permanent faults (e.g., "stuck-at"),

although the latter are more commonly considered in the literature. These arise in many ways, e.g., timing errors, data-dependent faults, and marginal design. Correlated faults are also problematic in practice, both operationally and in terms of analysis. For example in an LSI implementation, a single chip failure may result in multiple chip-output errors. Physical couplings are still a major source of difficulty, both in bonding and in pin connections.

* Problems in operations and in the inner-core of the operating system are a major source of trouble in large computer utilities. Even if the hardware were faultless, there are still enormous problems in keeping such a system operational with high availability. These problems are distributed among weak or inadequate software designs and questionable operating practices, as well as an occasional hardware design error. Two notable recent cases involve a ten-hour outage of an ESS No. 1 installation and a 29-minute outage of the NY Stock Exchange MDS-1. Both systems are designed for fault tolerance and high availability, but both experienced major outages attributable to human frailty (maintainers and system programmers, respectively) that aggravated troubles due to hardware difficulties. (Good system design can help circumvent these problems.)

3.4. A HIERARCHICAL FRAMEWORK FOR FAULT TOLERANCE

Peter Neumann has formulated a hierarchical framework for fault-tolerant computing systems, relevant to both hardware and software. It is described in a paper to be presented at COMPCON 72 in September /2/, included here as Appendix III. This framework facilitates the dynamic alteration of fault-tolerance techniques, as best suited to the current computing needs. This holds great promise for the attainment of economical fault tolerance, especially if real-time constraints are not uniformly critical. It is immediately applicable to large computing systems, and is useful to small systems as well. The hierarchical framework greatly facilitates the control over the exchange of redundant equipment for occasional slight increases in time, as well as enhancing the development and operation of the system.

3.5. FAULT-TOLERANT MEMORIES

Several investigations are under way involving fault tolerance in memories. Memory organizations suitable for advanced technologies are being investigated. Multiple-bit-per-chip ("frame-per-chip") organizations seem very powerful, and make framed-burst coding (also called phased-burst coding) highly advantageous. Peter Neumann has examined such an organization, and shown that such coding can be highly effective and economical /3/. Karl Levitt has incorporated these concepts into the BUCS (Bus Checker System) design /8/, which appears to offer good reliability at low cost. Although designed to take advantage of an aircraft environment, its balance of design seems to have significant implications for our goals. [Such coding techniques are also found in MDC (see Appendix I and II).] John Wensley has been examining the problem of reliably reconfiguring at the chip level in a large memory requiring many chips (somewhat akin to the problem of making page relocation mechanisms reliable). One rather promising solution involves distributed control. Also applicable is some earlier work in our Computer Science Group /9/ on reliable switching, useful for example in the switching of spare frames. In a related effort, Jack Goldberg has been studying distributed processor designs for fault tolerance /4/.

3.6. FAULT-TOLERANT PROCESSING

We have been investigating fault tolerance for logic operations and for arithmetic operations. In systems which are memory dominated, replication of processing units may be reasonably economical. In other situations coding may be desirable. Peter Neumann has shown /5/ that single faults can be detected in logic operations by performing all arithmetic logic operations in an arithmetic unit of suitable design, without additional (redundant) circuitry. Monteiro and Rao /10/ have recently investigated arithmetic-logic units with greater fault coverage.

3.7. RELIABILITY MODELLING

John Wensley has been investigating existing work on reliability modelling and its relevance to fault-tolerant computer design. Reliability modelling for fault-tolerant computing has been the subject of several studies /11-15/. Models have been proposed and analysed, but in general they do not answer certain important questions that arise in consideration of many fault-tolerant computers. We note some of these deficiencies here.

Existing models are more concerned with "survivability" than with reliability. In general the assumption is made that when a fault occurs it totally disables the unit in which it occurs, and that that unit must then be removed from future computational significance (possibly being replaced by a spare), or its significance removed by such techniques as voting in a TMR system. This does not allow accurate modelling of transient faults, in which a unit may survive, but some data may have been corrupted. In addition such models do not handle repetitive transients (intermittents) and the loss of effective computer power due to the CPU activity involved in error detection, correction, masking, etc.

A further deficiency of such models is the common assumption that any correction of a fault (by sparing, reconfiguration, masking, etc.) can be modelled by a single parameter, representing the probability of correct recovery from a fault. This does not allow the modeller to distinguish for example between faults that can be removed by voting and those whose erroneous effects cannot be so removed (e.g., a mistyped line).

The last deficiency to be noted is that the models do not take into account the fact that faults have different effects depending on the state of the system at the time of the fault. In some computer systems a dynamic trade-off is possible between reliability and computing power (e.g. SIFT, ARMMS). Faults that occur in the high reliability mode are less serious than those that occur in the high computing-power mode.

The existing models have further drawbacks. However the above discussion clearly shows a need for research aimed at developing modelling techniques that can handle some of the specialized problems arising in the analysis of fault-tolerant computers.

The consequence of the above is that the reliability estimates may be very pessimistic, which could result in very costly over-engineered designs. It is clear that improved methods of assessing reliability are required.

3.8. OPERATIONAL IMPLICATIONS

Various implications of system design on system reliability have been considered, in response both to problems arising in existing systems and to problems introduced by newer designs. As an example, John Wensley has examined the existing time-shared computer systems at SRI, (both TENEX systems), to ascertain the possibilty of utilizing an automatic checkpoint scheme is such environments. A conceptually simple scheme consists of simultaneously recording on magnetic tape all data that is normally placed on drums or disks. The saving of such data would provide a continuous history of the state of the users files so that unavailability of user files due the loss of a disk or drum could be circumvented. Initial studies show that a magnetic tape system of quite modest performance could handle the total data currently written onto drums and disks. (In one test the average over 3 1/2 hours was a bandwidth of 15 Kch/sec, which is well within the state of the art of magnetic tape systems. With an assumption of 50 per cent efficiency of tape utilization, a single 2400 ft. tape could store the total traffic of 24 minutes of computer activity.) Here good design in input-output hardware can be of great help, permitting simultaneous writes.

In another direction, Multics incremental backup experience shows half-hour delayed backup is very helpful to users in the event of system crash. However, recovery after damage to certain system files may be a very lengthy procedure. Combatting this problem requires very careful system design in a distributed system such as Multics. Another problem in Multics is introduced by operator errors in performing manual reconfiguration. Significant care in hardware design can aid considerably in such problems.

3.9. RELIABILITY TRADEOFFS AMONG TIME, SPACE, AND COMPLEXITY

David Huffman has been investigating the implication of information feedback on reliability and fault tolerance. Coding theory shows that knowledge of the outputs of a communication process can greatly improve the efficiency of reliable communication. Similar results are being sought for reliable computation. A fairly realistic model is being developed, so that the results may have some practical significance.

A related area of investigation involves the effects of time-space tradeoffs on reliability. Slight relaxation of critical real-time constraints seems to have potential for dramatically reducing the cost of hardware required for fault tolerance. We are beginning to understand this problem better. (It clearly has a bearing on 2.4.) Liason with Anatol Holt is anticipated to see if his formalisms can be of help.

Also related is our desire to obtain some results on complexity-reliability tradeoffs. The results of Cowan and Winograd and of Elias are based on some highly unrealistic assumptions. We have some hopes of obtaining meaningful results with more realistic assumptions.

3.10. OTHER EFFORTS

We are also devoting some effort to other problems. One is the massive-transient recovery-problem, in which a highly correlated fault source (e.g., a power surge) has left all units of the system suspect (see the end of Appendix III). Another problem is the "unflexed unit" problem, i.e., of how to diagnose normally unused but critical portions of the system (hardware and/or software), as for example the disaster recovery process itself. The general role of diagnosis is also being considered. Bernard Elspas /6/ has examined an extension of the model for self-diagnosis of Preparata, Metze and Chien. This extension holds some promise for useful results.

4. REFERENCES

4.1. PROJECT DOCUMENTS

1. J. Goldberg, P. G. Neumann and J. H. Wensley, Survey of fault-tolerant computing systems (Version 2), August 8, 1972. This survey contains our questionnaire, 12 system representations, and many relevant references beyond those given here. Reproduced here as Appendix II.

2. P. G. Neumann, A hierarchical framework for fault-tolerant computing systems, to be presented, IEEE Computer Society Conference, San Francisco, Ca., Sept. 12-14, 1972. (Preliminary work on this paper received some support from NASA-Langley Research Center under Contract NAS-1-10920, prior to the start of the ARPA contract.) Reproduced here as Appendix III.

3. P. G. Neumann, Framed burst correction, Project memorandum, May 4, 1972.

4. J. Goldberg, Distributed computing for reliability, Project memorandum, Feb. 3, 1972.

5. P. G. Neumann, A note on reliable arithmetically derived logic operations, Project memorandum, July 3, 1972.

6. B. Elspas, An analysis and generalization of the connection assignment model for fault-diagnosable systems, Project memorandum, March 22, 1972.

4.2. OTHER REFERENCES (See also Appendices for further references)

7. R. A. Short, The attainment of reliable digital systems through the use of redundancy -- a survey, Computer Group News, pp. 2-17, March 1968.

8. J. H. Wensley, K. N. Levitt, M. W. Green, P. G. Neumann, J. Goldberg, Fault Tolerant Archtectures for an Airborne Digital Computer, Stanford Research Institute, Report of Task I, Contract NAS1-10920, 24 July 1972 (Final Report -- draft version).

9. K. N. Levitt, M. W. Green and Jack Goldberg, A study of the data commutation problems in a self-repairable multiprocessor, SJCC 1968, pp. 515-527.

10. P. Monteiro and T. R. N. Rao, A residue checker for arithmetic and logical operations, Digest of the 1972 International Symposium on Fault-Tolerant Computing, IEEE Computer Society, June 19-21, 1972.

11. W. G. Bouricius, et al., Reliability modelling for fault-tolerant computers, IEEE Trans. Comp. vol C-20, pp 1306-1311, Nov. 1971.

12. F. P. Mathur, Reliabilty analysis and architecture of a hybrid redundant digital system: generalized triple modular redundancy with self repair, AFIPS Conf. Proc., SJCC, vol 36, May 5-7, 1970

13. F. P. Mathur, Reliabilty modelling and architecture of ultra-reliable fault tolerant computers", PhD thesis, UCLA, Computer Sciences Dept., June 1971.

14. F. P. Mathur, On reliability modelling and analysis of ultra-reliable fault-tolerant digital systems, IEEE Trans. Comp. vol C-20, pp 1376-1381, Nov. 1971.

15. J. Kruus, Upper bound for the mean life of self-repairing systems, Coord. Sci. Lab., Univ. Illinois, Urbana, Rep. R-172, July 1963.

APPENDIX I
CENSUS OF FAULT-TOLERANT COMPUTING SYSTEMS
(SRI, first version, August, 1972)

Following is a list of systems and system designs providing significant fault-tolerance and/or availability. Those systems indicated by "($)" are considered in greater detail in our Survey of Fault Tolerant Computing Systems (see Appendix II), where references are included. Several systems are described in what is referred to here as the "Intermetrics Report" (J. S. Miller, D. J. Lickly, A. L. Kosmala and J. A. Saponaro, Multiprocessor Computer Study, Final Report, Contract NAS 9-9763, Intermetrics, Inc., Cambridge, Mass, March, 1970). Other systems are given terse references here, where available. Abbreviations: P = Processor, M = Memory, SEC = single error correction, (D)ED = (double) error detection.

A. GENERAL-PURPOSE COMPUTING UTILITIES, generally good availability, human users, modest reliability, maintenance permitted

1. Multics, MIT and Honeywell, Cambridge, Mass (F. J. Corbato); ARPA-funded development. See E. I. Organick, A Guide to Multics, MIT Press.
* General-purpose computing utility (time-sharing, batch), with high availability and very high file integrity. Four systems currently operating.
* 2 P (GE-Honeywell 645s), multiprocessed, manual on-line reconfiguration of Ps and Ms, extensive fault isolation via the ring mechanism for protection and via file system access control, half-hour lag incremental file backup, variable-depth system rollback, redundancy in the file directory structure. (Single ED, minimal error checking in most systems, not mentioned below.)

2($). PRIME (nee MCS), University of California at Berkeley (H. Baskin); ARPA
* Reliable, secure, modest computer utility, high availability. In development.
* 5 P (design practical for 3 P to 8 P), with highly restricted possible connectivity of M and disk, strict isolation with no memory sharing or multiprogramming, "spontaneous" reconfiguration via a reliable self-checking switch. About 10% overhead for fault-tolerance.

3. Carnegie-Mellon University; NSF.
* Research system development with applications to ARPA speech understanding project; in design
* 16 P x 16 M (PDP 11s), with reliable crosspoint switch

4. University of Newcastle-on-Tyne, England; Scientific Research Council.
* General computing; in design
* PDP 11s

5. Various commercial time sharing services gain availability (but not necessarily reliability) by having cross-switchable Ps, Ms and secondary memory.

B. GROUND-BASED SPECIAL PURPOSE SYSTEMS, controlling the environment (or controlled by it), generally higher reliability and availability, often tighter real-time constraints than in A above, human maintenance usually possible.

6($). ESS (Electronic Switching Systems), Bell Labs, Naperville, Illinos.
* Telephone switching system; long-term continuous availability, with occasional errors tolerable to customers (?). Over 200 Number 1 ESS in operation, many more Number 2 ESS, TSPS.
* 2 P (1 functional, 1 standby checking and diagnosis), automatic reconfiguration. Separate nonalterable program store with SEC. 50% of all programs are diagnostics. Millions of hours of experience have aided in improving hardware and software reliability. People problems still very difficult (operations, maintenance).

7. FAA (Federal Aviation Admin.), IBM. See IBM Sys J., vol 6, no 2, 1967.
* Air traffic control, long-term continuous availability. Untolerated nontransient errors can be disastrous. 20 systems at ATC centers covering the continental United States.
* Up to 4 P (IBM 9020), up to 12 M. Program-controlled error analysis and reconfiguration, gracefully deconfigurable. 5-second battery backup power supply. Relies heavily on good and highly-available field engineers.

8. MDS-2 (Market Data System), New York Stock Exchange
* Stock trading ticker control. Near-continuous availability, no transaction losses permitted. Operational August 1972. Precursor MDS-1 operational for 7 years.
* 3 P (360/50), 2 multiprocessing with shared M & LCS (but 1 P basically monitoring), 3rd P normally spare (while running background jobs), extensive program checking.

9($). COMEX, Pacific Coast Stock Exchange
* Stock trading control; near-continuous availability, no transaction losses permitted, small real-time lag permitted. Operational since 1969.
* 2 complete systems (each has 360/50 plus 2 PDP 8s), one in San Francisco, one in Los Angeles, capable of running separately or cross-switched (interconfigurable).

10. NASDAQ, National Association of Securites Dealers Automated Quotations See Datamation, March 1972, pp. 42-45.
* On-line interactive system to facilitate trading of OTC securities; high availability; operational since end of 1971.
* 2 P (1108s), multiprocessing under EXEC 8, capable of running simplex. Dual records in file structure, automatic recovery techniques.

11. CLC, Bell Labs, Whippany NJ; ABMDA (Safeguard)
* Safeguard missile defense; continuous availability when (and if) required. In development since mid-60s.
* Up to 10 P, multiprocessed with on-line sparing, separate program memory not processor-writeable; program retry; ED via four-bit checking on 64-bit words.

12. MULTIPAC, General Telephone and Electronics, Waltham, Mass; NASA-Ames. See IEEE Trans. Aerospace and Electronic Sys., Sept. 1971, pp. 974-981.
* Data handling for deep space probes. Availability not critical. Design only.
* Up to 5 P, 15 M (4 K each), gracefully degradable to 1 P, 1 M. Manual reconfiguration of software and hardware via ground-based diagnosis, reprogramming, reassembly and transmittal of a new system back into space. Ability to fix live bugs (hard and soft) thus also exists.

13. Standard Telecommunications Lab, Harlow, England. See Electrical Review, 6 Feb 1970, pp. 1-3.
* Real-time control
* 1 P, SEC/DED in M, in transfers, and in I-O; duplication of punch/reader and of M access switches; triplication of control and of function unit. 52% of hardware due to fault tolerance

14. Foxboro 88, Foxboro Corp. Process control using 2 P (PDP 8s)

C. AERO-SPACE SYSTEMS, usually with ultra-high reliability and availability requirements, usually critical real-time contraints, human maintenance usually not possible. At least the first four efforts have resulted in installed or prototype systems. The remaining efforts represent mostly designs in various stages of completion.

15. SIRU (Strapped-down intertial reference unit), MIT Draper Lab (A. L. Hopkins, Jr.). See Intermetrics Report (reference above).
* Apollo guidance. Used in Apollo program.
* 2 P (1 as standby), M duplicated.

16($). JPL-STAR, JPL, Pasadena Cal (A. Avizienis); NASA
* Unmanned outer-space travel computer, long-life availability without maintenance. Prototype in operation since 1969.
* 1 P (uniprocessing), heavy use of coding (residue checking for SED in memory and arithmetic, ED in op codes), duplicated logic operations, triplicated monitoring and control (TARP = test and repair processor), sparing by power switching. User-provided rollback points. 60% of hardware due to fault tolerance

17($). ACGN, CERBERUS, etc., MIT Draper Lab, Cambridge, Mass (A.L. Hopkins, Jr.); NASA/MSC.
* Apollo manned space on-board control, very high reliability during the mission without maintenance. Prototype exists.
* At least 1 processing unit (up to 6), multiprocessing among processing units, replication within each processing unit and within memories (without coding). Two concepts:
(a) duplexed processing units, triplexed scratchpad memories, triplexed memories and buses, with spares;
(b) triplexed processing-scratchpad units.
About 80% redundant

18($). MECRA, Electronique Marcel Dassault, St. Cloud, France; DRME
* General-purpose design for special-purpose applications, including aerospace. Prototype now working.
* Duplex arithmetic, Hamming code (7,4) as DED on coded decimal representations (with six unused combinations), sparing, microprogrammable reconfiguration. About 66% redundant

19($). MDC (Modular digital computer), IBM Yorktown Hts, NY,;
NASA-Huntsville.
* Modular system, wide range of high-reliability applications; design only
* m P, multiprocessing and replication as well. FO-FO-FS (fail-operational on first and second faults, fail-safe on the third) in 4 P fault-tolerant mode, detection mode also possible. Microdiagnostics, b-adjacent multiple errors handled in M, extensive self-checking.

20($). MSC (Modular spacecraft computer), Ultrasystems (Newport Beach Ca) and Raytheon (Waltham); SAMSO/SYT (Los Angeles Ca)
* Reconfigurable guidance and control, space shuttle use; long-life reliability.
* The Raytheon entry in this effort has 1 P, identical subP and subM reliably switchable with sparing. SEC in M plus 3 spare bits reliably switchable via "rippler", burst-error detection in mass M, triplicated control, duplicated configuration control.
* The Ultrasystems entry is similar to the JPL STAR.

21($). SIFT, Software implemented fault tolerance, SRI (John Wensley);
NASA-Langley
* Airborne control (commercial aviation); availability of correct results during flight; some tasks more critical than others, permitting slight degradation of less critical tasks.
* Multiprocessing with variable software replication, dependent on application program (software reconfigurable). Fault tolerance via software techniques avoids need for special hardware, permits use of existing designs. Connectivity is restricted: P can modify only its own M, can read all Ms, preventing fault propagation. Executive uses the same fault-tolerance procedures as the application programs. About 75% redundant

22($). ARMMS, Hughes, Fullerton CA (W. L. Martin); NASA-Marshall (MSFC)
* Spaceborne control; long-life reliability
* n P, dynamically reconfigurable, e.g., as independent-process multiprocessing or as replication with sparing. 20%-80% redundant (variable)

23($). Intermetrics multiprocessor, Cambridge Mass (J. S. Miller), outgrowth of EXAM; NASA-ERC (Houston)
* Manned orbiting space station
* m P (1 to 8, nominally 3), each P duplicated, coding in M (ED), buffered instruction retry, save within interrupted instruction.

24($). Autonetics (N. Am. Rockwell, Anaheim, L. J. Koczela); NASA-MSC
* Space shuttle; long-life reliability
* 4-level redundancy, FO-FO-FS (cf. MDC) requires 80% redundancy, less for lower fault tolerance.

25. BUCS (bus checker system), SRI (Karl Levitt), NASA-Langley. See SRI Final Report, NAS1-10920, 1972 (Reference 8 of this Report).
* Aircraft control, as in SIFT
* 5-10 (local) P & M units, each duplicated internally, frame coding in central M, bus checker coordinates restart mechanism, periodic diagnoses of M and of unflexed processor functions. About 33% redundancy.

26. TOPS, JPL (Gilley). See IEEE Trans. Astr-Aero, Sept. 1970.
* Thermo-electric outerplanet space travel
* Related to JPL-STAR.

27. MFC, Hamilton-Standard; NASA-ERC. See Intermetrics Report.
* Modular flight computer
* 3 P, 3 M, cross-configurable, TMR or 3 P multiprocessor

28. ALPHA, CDC. See Intermetrics Report.

29. AADC, Honeywell; NASA, AADC Naval Air Systems Command. See Intermetrics Report.

30. IRAD, Litton. See Intermetrics Report.

31. SDC-Burroughs; USAF-Wright-Patterson, Multiprocessor

32. S-3, Univac

33. COSMOS, RCA (cf. SUMC)

APPENDIX II
SURVEY OF FAULT-TOLERANT COMPUTING SYSTEMS (revised Aug 1972)

Jack Goldberg, Peter G. Neumann and John H. Wensley
Computer Science Group, SRI, Menlo Park, CA, 94025

This appendix presents replies to a questionaire sent to architects of various fault-tolerant computing systems. It is hoped that the questionaire will itself be useful as a descriptive form and that the replies will aid in understanding and comparing the systems included here. To this end the questionaire has been designed to permit a concise description of each system, its goals, its motivations, its principles, its structure, its techniques, and its achievements to date.

The first issue of this document was distributed informally to conference participants at the Second Symposium on Fault Tolerant Computing, Boston, June 19-21, 1972. It was intended to support the panel discussion "Approaches to the Architecture of Fault-Tolerant Computing", chaired by Jack Goldberg. The replies given here are included essentially in their entirety. Significant efforts not represented here include several existing systems (such as IBM's FAA system, Bell Lab's CLC and various query systems) as well as numerous design and development efforts (e.g. government systems, and systems under development at Carnegie-Mellon University and the University of Newcastle-on-Tyne).

The contents of this appendix are as follows.

The preparation of this document and the questionaire was supported by the Defense Advanced Research Projects Agency of the Department of Defense, and was monitored by ONR under Contract Number N00014-72-C-0254. The views are clearly those of the named contributors and do not necessarily represent any official policies of ARPA or the U. S. Government.

SURVEY OF FAULT-TOLERANT COMPUTING SYSTEMS--QUESTIONNAIRE

Jack Goldberg and Peter G. Neumann, SRI, Menlo Park CA

1. IDENTIFICATION of the system
1.1. NAME: What is the relevant name of the system (and/or project)?

1.2. RESPONSIBILITY: What is the responsible organization?

1.3. SUPPORT: What are the sources of support?

1.4. PARTICIPANTS: Who (and what organizations, if relevant) are the principal participants?

1.5. START: What was the date of conception?

1.6. COMPLETION: What was, or is expected to be, the completion date? (Specify prototype acceptance date, or design completion date if design only.)

1.7. BIBLIOGRAPHY: What are the most relevant references?

2. MOTIVATION for the system
2.1. PURPOSE: What is the main purpose of the system (e.g., general-purpose computing, real-time air-traffic control, store-and-forward)?

2.2. PHYSICAL ENVIRONMENT: Where does the system operate (e.g., ground-based, airborne, spaceborne)?

2.3. COMPUTING ENVIRONMENT: How does the system relate computationally to its environment (e.g., locally, remotely, via a network, interactively, via peripherals, with human users)?

2.4. COMPUTING OBJECTIVES: What are the specific computing objectives, regarding capability, capacity, performance (throughput or response), configuration scaleability, maximum real-time delays, etc. (as relevant)?

2.5. RELIABILITY OBJECTIVES: What are the specific system reliability objectives, with respect to desired availability during what period, minimum time to system failure, maximum permitted duration of outage, etc.?

2.6. DYNAMIC VARIABILITY: How may these objectives vary during operation? (E.g., how may performance degrade? May performance be explicitly exchanged for increased reliability?)

2.7. PENALTIES: What are the penalties arising from faulty operation? (Possible examples include loss of life, badly decreased performance, the necessity of manual intervention, loss of revenue, etc.)

2.8. CONSTRAINTS: What explicit physical constraints exist (e.g., with respect to size, weight, power, cost)?

2.9. TRADEOFFS: What critical tradeoffs exist among the objectives?

3. DESCRIPTION of the system
3.1. ARCHITECTURE
3.1.1. CONFIGURATIONS
3.1.1.1. INTERCONNECTIVITY: What is the basic configuration, and what restrictions exist on interconnectivity? (You may choose to include a block diagram, a PMS diagram a la Bell and Newell, or other useful representation.)

3.1.1.2. RANGE: What is the range over which configurations are sensible (minimum to maximum), e.g., how many processors, how many memory modules (of what size and word length, and with what restrictions if any), etc.?

3.1.1.3. CAPABILITY: What is the effective computing power of the smallest sensible configuration in 3.1.1.2? Please compare it roughly with a well-known system (e.g., 360/40, 65, 195), and cite a ball-park figure for the number of additions per second. Capability required for fault-tolerance should not be included.

3.1.2. EXECUTIVE and operating system
3.1.2.1. MODES of operation: How does the system operate? (E.g., is each processor multiprogrammable? Is independent-process multiprocessing possible? Is cooperative-process multiprogrammed multiprocessing possible?)

3.1.2.2. SOFTWARE organization: What is the structure of the system software? How is it distributed with respect to the hardware?

3.2. FAULT TOLERANCE
3.2.1. FAULTS TOLERATED: What faults are tolerated by the system, with what resulting effects on system behavior?

3.2.2. FAULTS NOT TOLERATED: What faults cannot be tolerated by the system, and what are the corresponding effects? Identify the weakest links.

NOTE: Faults may be characterized in many ways, including type (e.g., faulty hardware at various levels such as a chip, module, bus, power supply, arithmetic unit, processor, memory; faulty software such as in the executive, in a compiler, or in an applications program; faulty usage and bad inputs), nature (e.g., timing considerations, old age, various physical phenomena), duration and frequency (e.g., one-shot, recurrent, permanent), scope (e.g., isolated faults, correlated or independent multiple faults, with varying degrees of propagation), effect (random, predictable), etc.

3.2.3. TECHNIQUES: What basic techniques are employed to provide fault-tolerant capability, and when, where, and how are they used? Include hardware and software techniques.

NOTE: Applicable techniques include (possibly in combination) replication (e.g., triple-modular redundancy at various levels, redundant computations using independent algorithms), coding (e.g., error-detecting or -correcting codes on a bus, in memory, in arithmetic), repetition and rollback, reconfiguration (including removal without replacement and replacement with spares), diagnostics (e.g., stand-alone, on-line, interactive; preventive, emergency; remote, local), protection (of processes, data, programs, etc.), and outside intervention (human or otherwise). These techniques may be used statically (e.g., always invoked) or dynamically (e.g., configured as needed); at various module levels in hardware and software, in combination with certain events and with certain other techniques.

3.3. NOVELTY: What are the most unusual design features of this system?

3.4. INFLUENCES: What other efforts (systems, research) have had an influence on your system design?

3.5. HARD-CORE: If there is a concept of "hard-core" in your system, what is its significance? (Please define your concept.)

4. JUSTIFICATION for the system
4.1. RELIABILITY EVALUATION: How is reliability estimated and/or demonstrated (e.g., via analysis, simulation, stimulation of faults, theoretical arguments)?

4.2. COMPLETENESS OF EVALUATION: How complete is your design evaluation?

4.3. OVERHEAD: What percentage(s) of total system resources do you attribute to the achievement of fault-tolerance? (Consider cost, logic, execution time, memory, etc., as applicable.)

4.4. APPLICABILITY: What is the potential range of applicability beyond that stated in sections 2.1 - 2.4 above?

4.5. EXTENDABILITY: In what ways could the system design be advantageously extended, with what increase in cost, and to what effect?

4.6. CRITICALITIES: How critically do the design choices match the design goals? (E.g., could slight changes in goals result in great savings in design, implementation, and/or operation? Is multiprogramming or multiprocessing critical? Is the choice of hardware critical?)

4.7. IMPLICATIONS: What special requirements (if any) does the basic design impose (e.g., on the hardware designers, on the software developers, on users and maintainers)?

5. CONCLUSIONS
5.1. STATUS: What is the current status of the system?

5.2. EXPERIENCE: What conclusions can you reach based on your experience with the system to date (e.g., in design, implementation and operation)?

5.3. FUTURE: What is planned for future development or use of the system?

5.4. ADVANCES: What developments (theoretical or practical) would be desirable for significantly advancing the state of the art in fault-tolerant computing?

6. COMMENTS (Please include any comments on your system, on this questionnaire, etc. which you would like to add. Opinions, prejudices and philosophies are welcomed.

SURVEY OF FAULT TOLERANT COMPUTER SYSTEMS

Algirdas Avizienia
UCLA Computer Science Dept., Loe Angalas, CA and
Spececreft Computer Section, JPL, Pasadena, CA, June 1972

1. IDENTIFICATION

1.1 NAME: JPL-STAR (Self-Testing-And-Repeiring) Computer

1.2 RESPONSIBILITY: Spacecraft Computer Section, Astrionics Diviaion of the Jet Propulaion Leboretory, Pesadena, Californie.

1.3 SUPPORT: NASA - Office of Advanced Reseerch and Technology (via JPL)

1.4 PARTICIPANTS: A. Avizienis, D. A. Rennels, J. A. Rohr, F. P. Mathur, G. C. Gilley

1.5 START: 1961

1.6 COMPLETION: Operational - Spring 1969 (laboratory model), modifications continue

1.7. BIBLIOGRAPHY:

*1. A. Avizienia, et al., The STAR (Self-Testing and Repairing) Computer: An investigation of the theory and practice of fault-tolerant computer design, IEEE Trans. Computer C-20, pp. 1312-1321 (Novamber 1971).

*2. A. Avizienie, "Design of fault-tolerant computers," FJCC, pp. 733-743, 1967.

*3. A. Avizienis, "An axperimental self-repairing computer," Information Processing, IFIP, Vol. 2, pp. 872-877, 1968.

*4. A. Avizienia, F. P. Mathur, D. Rennels, and J. A. Rohr, "Automstic maintenance of aerospace computers and spececraft information and control systems," Proc. AIAA Aerosp. Comput. Syat. Conf., Paper 69-966, pp. 1-11, September 8-10, 1969.

*5. A. Avizienia, "Concurrent diagnosis of erithmetic processors," Digeat of the lat Annual IEEE Comput. Conf., pp. 34-97, 1967.

*6. A. Avizienis, "Arithmetic error codes: Coet and effectiveneaa studies for application in digital system design," IEEE Trans. Comp., C-20,, pp. 1322-1331,Nov 1971.

*7. F. P. Mathur end A. Avizienis, "Rsliability analysis and architecture of s hybrid-redundant digital syatem: Generalized triple modular redundancy with self-repeir," SJCC, pp. 375-383, 1970.

*8. F. P. Mathur, "On reliability modsling and enalysis of ultrareliable fault-tolarant digital systems," IEEE Trans. Comp., C-20, pp. 1376-1382.

*9. G. C. Gilley, "Automatic maintenance of spacecraft systems for long-life, deep-space missions," Ph.D. dissertation, Dept. Comput. Sci., UCLA, September 1970.

*10. F. P. Mathur, "Rsliability estimation procedures and CARE: The computer aided reliability estimation progrem," Jet Propul. Lab. Quart. Tech. Rev., Vol 1, October 1971.

2. MOTIVATION

2.1 PURPOSE: Experimental laboratory GP machine; auitable for spacecraft control

2.2 PHYSICAL ENVIRONMENT: Laboretory environment

2.3 COMPUTING ENVIRONMENT: Local I/O facilities

2.4 COMPUTING OBJECTIVES: Capeble of automatically maintaining an unmanned spacecreft

2.5 RELIABILITY OBJECTIVES: 100,000 hour aurvival with 0.95 reliability; tolerance of transient faults; outage for recovery below 50 msec.

2.6 DYNAMIC VARIABILITY: Maximum computing powar required at end of mission

2.7 PENALTIES: None for lab model; loss of spacecraft for flight model

2.8 CONSTRAINTS: None for lab modal; for the flight model tha waight of tha subsystem was not to excaed 40 lb. and the power consumption was not to ba greeter than 40 W.

2.9 TRADEOFFS: None

3. DESCRIPTION

3.1 ARCHITECTURE

3.1.1 CONFIGURATIONS

3.1.1.1 INTERCONNECTIVITY: See Figure

3.1.1.2 RANGE: One processor of each claas (operating); 16 memory modules of 4096 words each (maximum operating memory)

3.1.1.3 CAPABILITY: 500 KHz maximum clock rate and byte-serial operation in laboratory model.

3.1.2 EXECUTIVE

3.1.2.1 MODES: Only one processor operates at a given time (Single-processor organization)

3.1.2.2 SOFTWARE: The programming subsystem consists of three modules: an assembler, a loader, and a functional simulator. An executive program facilitates coordinated use of these modules. The operating subsystem coneists of two modules: the resident executive module and the applications programs module. The programming subsystem has been implemented on the Univac 1108. The modules of the operating system of the STAR computer aoftware system consist of the resident executive module and the application module. The STAR resident executiva augments the self testing snd repairing features of the hardware in addition to its normal functions. The standard fastures include interrupt control, input/output proceesing and job scheduling. Novel features incorporated due to the fault-to'er--- architecture of the STAR computer include a "col [illegible] ability, reconfiguration processing, rollba [illegible], and diagnosis of faulty units. The cold start [illegible] ty resets the hardware and software after a disaster restart aa well as prior to an initial load. Reconfiguration processing is required for memory replacement, since software sssistance is required to load a newly activeted memory unit. All programs running on the STAR computer require rollback (recovery) points. The resident executive provides rollback status storage and controls events which are nonrepeatable, i.e., they may not occur more than once even if a rollback takes place. Finally, it implements diagnosis for faulty units to determina the cause and extent of failures for possible reuse. The present application modules include floating point arithmetic subroutines, and test and demonstration programs. The application programs that will be required for space missions are a part of the TOPS control computer subsystem project.

3.2 FAULT TOLERANCE

3.2.1 FAULTS TOLERATED: The principal goal of the design is to attain fault tolerance for a variety of faults: transient, permanent, random, and catastrophic.

3.2.2 FAULTS NOT TOLERATED: (a) Transients at a rate higher than allowed by the length of "rollback" segments of programs; (b) shorted bus wires (isolators are employed) or power switch "on" failures.

3.2.3 TECHNIQUES:

*1. All machine words (data and instructions) are encoded in error-detecting codes and feult detection occurs concurrantly with the execution of the programs.

*2. The computar is divided into a aet of replaceable functional units containing their own instruction decoders and sequence generatora. This decentralization ellows aimple fault-location procedures and simplifies system interfaces.

*3. Fault detection, recovery and raplacement are carried out by special-purpose hardware. In the case of memory damage, software augments the recovery hardware.

*4. Transient faults are idantified and their effects are corrected by the repetition of a segment of the current program; permanent faults are eliminated by the replacement of faulty functional units.

*5. The replacement is implemented by power switching: units are removad by turning power off end connected by turning power on. The information lines of all units are permanently connectad to the busea through isolating circuits; unpowered units produce only logic "zero' outputs.

*6. The error-detecting codes are supplamented by monitoring circuits which serve to verify the proper synchronization and internal operation of the functional units.

*7. The "herd core" teat and repair processor (TARP) is protacted by triplication end replacement of failsd members of the triplet.

3.3 NOVELTY: Power switching, status signals, encoding of ins.ructions, emphasis on transient-recovery with program survival.

3.4 INFLUENCES: Theoretical work by Reed and Brimley; Kruus and Seshu; Griesmer, Miller and Roth.

3.5 HARD-CORE: The "hard core" monitor of the STAR system is designated as TARP (test and repair processor) in the Figure. Th. TARP monitors the operation of the STAR computer by two methods: (1) testing every word sent over the two data buses for validity of its code; snd (2) checking the status messages from the functionsl units for predicted responses.

Three fully powered copies of the TARP sre opersted at all times together with n standby spsres (n = 2 in the present design). The outputa of the TARPs sre decided by a 2-out-of-(n+3) threshold vote. When one powered TARP disagrees with the other two, the recovery mode is entered and an attempt is made to set the internal stste of the disagreeing unit to match the other two units. If this TARP rollbsck attempt fails, the dissgreeing unit is returned to the standby condition snd one of the standby units receives power, goes through the TARP rollbsck, and joins the powered triplet. The computer is now restarted, a rollbsck performed, and stsndard operation continues. Becsuse of the three unit requirement, design effort has been concentrated on reducing the TARP to the least possible complexity. Experience with the present model haa led to several refinements of the design.

The replacement of faulty functional units is commanded by the TARP vote and is implemented by po switching. It offers several advantages over the swi ng of informstion lines which connect the o the bus. The number of switches are reduced to on nit, power is conserved, and strong isolstion is d for cstsstrophic failures. Magnetic pow witches have been developed which are part o. each uni power supply and are designed to open for most intern fsilures. The threshold function ia inherent in the control windings of the switch. The information lines of each unit are permanently connected to the buses through component-redundant isolation circuits. The signal on a bus is the logic OR of all inputs from the units, snd unpowered units produce only logic zero outputs. The power switch snd the buses utilize component redundancy for protection against fatal "shorting" fsilures.

4. JUSTIFICATION

4.1 RELIABILITY EVALUATION: The computing operstions for the analysis was done with the aid of the computer-aided relisbility estimation (CARE) program, which was developed ss s design tool during the reliability study. CARE is s softwsre psckage developed on the Univac 1108. CARE msy be intersctively accessed by a designer from a teletype console to calculate his reliability estimstes. The input is in the form of s system configurstion description followed by queries on the various reliability parsmeters of interest and their behavior with respect to mission time, fault coversge, fsilure rstes, dormancy factors, allocsted spsres, and partitioning. The CARE program is extensible, snd it msy be updated to incorporste new reliability models as they become availablc. Physicsl fault-injection experiments are currently in progress.

4.2 COMPLETENESS OF EVALUATION: Experiments sre expected to continue through 1972.

4.3 OVERHEAD: Depends on the number of spares. With one spare for each module--sbout 150 percent extra cost (i.e., 60% overhead).

4.4 APPLICABILITY: Various real-time applications that require very fast recovery.

4.5 EXTENDABILITY: Spare processors could be utilized in a multiprocessor mode. Additional buses and supervisory mechanisms would be required.

4.6 CRITICALITIES: The design goal was a better understanding of replacement systems. In order to retain contact with the practice of computer design, it was decided to design and construct an experimental general-purpose digital computer which would incorporate dynamic redundancy (i.e., fault detaction and replacement of failed subsystems) as integral parts of its structure. The design objectives have baen carried out and the system, called the STAR computer, began operation in 1969. The modular nature of the STAR computer has allowed systematic expansion and modifications that are still being continued.

An early objective of the design is to study the class of problems which sre encountered in transforming the theoretical model of s self-repairing system into s working computer. State-of-the art integrated circuit and memory technology wss employed in the design. This objective sppesrs to hsve been attained reasonably well.

4.7 IMPLICATIONS: Designers must give (s) advance attention to modulsrizstion and coded operands; (b) special software features sre needed (see 3.1.2.2); (c) users must observe "rollbsck" rules in programming.

5. CONCLUSIONS

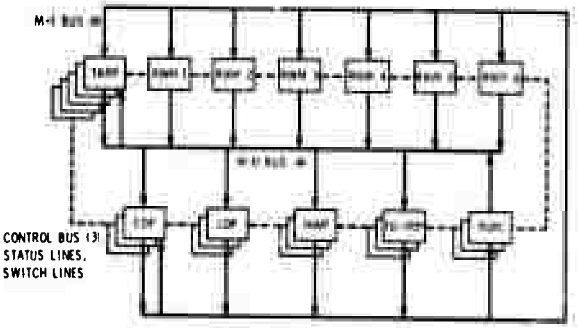
5.1 STATUS: Operating in lsborstory; being extensively tested snd modified to improve wesknesses thtt sre uncovered.

5.2 EXPERIENCE: Practicsl implementstion of replscement systems is feasible. Transient faults can be systematicslly eliminsted without programm loss. Transient tolersnce can be specified in terms of "durstion" snd "frequency" psrsmeters.

5.3 FUTURE: The research and development progrsm which led to the STAR computer is continuing in several directions. The design of several improved second generation STAR functional units is under wsy, including s new arithmetic processor, s control processor for medium-scsle integrsted-circuit implementation, snd the shared READ-WRITE memory unit for the storage of automatic maintenance information from the spacecraft telemetry system. Analysis of sutomatic maintenance algorithms and design of a command/data bus for their implementation sre under intensive study. Other current investigstions sre concerned with the following areas: (1) hsrdwsre-software interaction in a fault-tolerant system with recovery, especially the intersction of the TARP snd the opersting system; (2) studies of sdvsnced recovery techniques, i.e., post-cststastrophic restart, TARP replacement schemes, recovery from massive interference, partial utilization of failed units; (3) advanced component technology, especially methods to attain bus and power switch (i.e., hard core) immunity to faults; (4) heuristic studies of fsult tolerance by interpretstion of extensive experiments with the STAR bresdbosrd as the instrument; (5) design of a second-genersticn STAR-type computer with universsl processor snd storage modules, snd their implementstion by large-scale integrstion; (6) computstional utilization of the spsre units for supplemental tasks in s multiprocessing mode.

5.4 ADVANCES: (a) methods of coversge messurement; (b) technology advances in isolstor snd switch design; (c) studies in restsrt ("roll-bsck") implementstion by sutomstic methods.

6. COMMENTS: Design, construction, and testing of laboratory models is criticslly important to advsnce the stste of the art and to gsin acceptsnce among prsctitioners of design in industry.



STAR computer organization.

COP Control processor, contains the location counter and index registers.
LOP Logic processor, (two copies are powered).
MAP Main arithmetic processor.
ROM READ-ONLY memory, 16,384 permanently stored words.
RWM READ-WRITE memory unit (4096 words, two copies powered, 12 units directly addressable.).
IOP Input/Output processor, contains I/O buffer.
IRP Interrupt processor, handles interrupt request.
TARP Test and repair processor, (three copies powered).

SURVEY OF FAULT-TOLERANT COMPUTING SYSTEMS

W. C. Carter
IBM Thomas J. Watson Research Center
Yorktown Heights NY 10598

1. IDENTIFICATION

1.1. NAME: I am reporting mainly on a long-term reaearch effort in techniques for fault-tolerant computer architecture. The relevent prior publications heve used, for example, the terms "modular architecture", "self-repairing computers", "dynamic checking", "fault diegnosis", "stand-by sparing" or "dynamic recovery" in the titles and the authors heve been some subset of the participants named in 1.4. For present purposes I will talk about a paper Moduler Digital Computer system called MDC whose principal properties will be specified later. For reality, some requirements will be imposed which have nothing to do with fault tolerance per se. This system does not really exist, and will not exist, but is specified to provide a focus for our fault tolerant computing reaearch.

1.2 RESPONSIBILITY: IBM Research.

1.3 SUPPORT: Support haa come from IBM, U. S. Air Force and NASA.

1.4 PARTICIPANTS: W. G. Bouricius, W. C. Carter, E. P. Hsieh, D. C. Jeasep, Jr., G. P. Putzolu, J. P. Roth, P. R. Schneider, C. J. Tan, A. B. Wadia.

1.5 START: Formal initiation occurred in March, 1966.

1.6 COMPLETION Open ended. No end item is scheduled.

1.7 BIBLIOGRAPHY:

*Roth, J. P. "Diagnosis of automata failures: a calculus and a method", IBM Journal, vol. 10, 4, 1966.

*Bouricius, W. G., Hsieh, E. P., Putzolu, G. R., Roth, J.P., Schneider, P. R., Tan, C. J., "Algorithms for detection of faults in logic circuits", IEEE TC, Vol. C-20, Nov. 1971.

*Bouricius, W. G., Carter, W. C. and Schneider, P. R., "Reliability modeling techniques and tradeoff studies for self-repairing computers", ACM National Conference, San Francisco, California, August, 1969.

*Bouriciua, W. G., Carter, W. C., Roth, J. P. and Schneider, P. R., "Investigations in the design of an automatically repaired computer", Paper Number 6.4 Conference Digest of the First Annual IEEE Computer Conference, Chicago, Illinois, September 6-8, 1968.

*Carter, W. C. and Schneider, P. R., "Design of dynamically checked computers", IFIPS, Edinburg, Scotland, August, 1968.

*Carter, W. C., Jessep, D. C., Wadia, A. B., "Error-free decoding for failure-tolerant memories", 1970 IEEE Computer Conference, Washington, D. C., June, 1970, pp. 229-239.

*Carter, W. C., Jessep, D. C., Bouricius, W. G., Wedia, A. B., McCarthy, C. E., Milligan, F. G., "Design techniques for MARCS" (Modular Architecture for Reliable Computer Systems), NASA Contract NAS8-24883, RA12, IBM T. J. Watson Research Center, Report Number 70-208-002. March 26, 1970.

*Carter, W. C., Jessep, D. C., Wedia, A. B., Schneider, P. R., Bouricius, W. G., "Logic design for dynamic and interactive recovery", IEEE TC, Vol. C-20, Nov. 1971.

2. MOTIVATION

2.1 PURPOSE: Real time control, data acquisition and data management.

2.2 PHYSICAL ENVIRONMENT: Aerospace applications have predominated in specific design decisions. Modularity should ensure wide applicability.

2.3 COMPUTING ENVIRONMENT: The MDC is planned to be able to run the gamut from being insulated from human control, serving a variety of sensors and effectors, to being able to accept ground-based human directed control.

2.4 COMPUTING OBJECTIVES: Predicted configuration scaleability primarily under internal control including systems which are fault tolerant by masking redundancy, by stand-by redundancy, or by software checks; systems whose use of power is variable (but whose thruput is affected); and systems operating in parallel. The major objective is to provide means for meeting various requirements with a high degree of confidence.

2.5 RELIABILITY OBJECTIVES: The system is to be designed to meet varying specific mission reliability objectives with a high degree of certainty. Examples are survival for n years with a probability p; "Fail operational, fail operational, fail safe", or reliability variable with mission task.

2.6 DYNAMIC VARIABILITY: As stated above, dynamic variation of system parameters such as performance, reliability and power consumption with confidence in the design as a major objective.

2.7 PENALTIES: Variable with mission, ranging from loss of human life through expensive flight hardware to abortion of flight objectives.

2.8 CONSTRAINTS: Hardware must be designed to fit weight, power and size requirements, yet able to have thruput compatible with mission requirements and to support the software necessary for reasonable programming effort per mission.

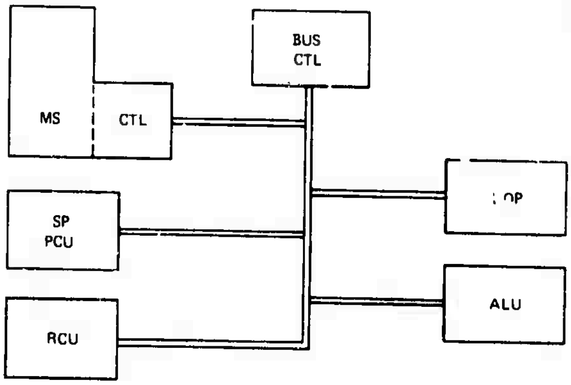
2.9 TRADEOFFS: Hardware efficiency and potential thruput are traded for 1) system reliability as defined per mission phase; 2) simplification of recovery process and other basic executive functions; 3) high malfunction coverage and design certification; 4) ease of program validation; 5) convenience of programming and ease of diagnosis for external equipment; 6) system flexibility.

3. DESCRIPTION

3.1 ARCHITECTURE

3.1.1. CONFIGURATIONS

3.1.1.1. INTERCONNECTIVITY: The basic uniprocessor configuration consists of partitioned computer subunits attached to several busses. The basic subunits are (see attached rough diagram): ALU, Scatch and Program Control Unit, Bus Control, I/O Processor and Recovery Control Unit. The bus orientation remains, but the units may be modified (microprogrammed) for varying missions. The system consists of replicas of the basic subunits, with configuration control governed by the RCU and Executive Program. A major problem is the interface design to meet the constraints of fault tolerance, long life, and varying modes of operation. The memory is encoded with a b-adjacent error correcting code and spare b wide subunits per basic module.



3.1.1.2. RANGE: The range of the system is not frozen in the architectural concept. After four processors the law of diminishing returns sets in sharply and further partitioning may well be a better bet for long life. The memory will consist of modules, each module consisting of b-wide units with b-adjacent coding and spare b-width units. The upper limit depends upon the hardware available, but hardware does not appear to be critical.

3.1.1.3. CAPABILITY: The order of 10E5 to 10E6 additions per second per basic system with a minimum of 256K-512K 32-bit words of memory. I/O will be handled by up to 4 16-bit parallel channels with 50,000 transfers per second simultaneously on one input and one output channel. The I/O processor will handle the details of I/O control under direction from the processor Executuve.

3.1.2 EXECUTIVE: The standard executive control (allocation, scheduling, dispatching, I/O) will be achieved by replicated software routines. These tasks have not been studied much.

3.1.2.1. MODES OF OPERATION: Each processor is multiprogrammable. System operation includes fault masking, multiprocessing with hardware fault detection and multiprocessing with software analysis. The mode of operation of most concern is that of recovery initiation, the interaction of the recovery and error analysis programs of the executive and the RCU. Recovery and audit programs always run background whether the system is in fault masking, fault detection or software analysis modes.

3.1.2.2. SOFTWARE ORGANIZATION: The system software will be distributed among the processors and analyzed by audit routines for early detection of errors.

3.2. FAULT TOLERANCE
3.2.1 FAULTS TOLERATED: In the error-masking mode, any number of faults which affect only one partitioned sub-unit can be tolerated. The system handles transient faults with instruction retry or permanent faults with hardware controlled reconfiguration. The cause is irrelevent as long as the interface detects disagreement. The disagreement circuits are self-checking so faults in them are detected. Initially the same malfunction in three units is necessary to defect the system. After reconfigurations two faulty units may escape detection. In the error detection mode, faults causing a single subunit to be in error are detected. At this point the same errors in two units will be undetected. Diagnosis and software recovery is necessary for continuation.

Faults detected by software checks are detected and recovery should follow in the unchecked multiprocessing mode. Faulty software may be detected by the RCU time-out tests and system evaluation procedures.

3.2.2. TECHNIQUES: In hardware fault tolerant mode the system should FO - FO - FS for each one of the partitions of the system if four copies of the basic computer are used. Diagnosis can continue the computation with one partition unchecked. Detailed fault analysis must be performed to validate such goals. In hardware fault detection mode the system should run at least two multiprocessor hardware checked systems. A fault would be detected, and diagnoses would allow continuation with one partition unchecked by hardware. Achieving such hardware/firmware/diagnosis goals depends upon the development of many tools of fault analysis. The memory encoding is b-adjacent multiple error correcting and/or multiple b-adjacent error detecting. The codes used are variants of Reed-Solomon codes with combinational self-checking translators which pass only correct code words. Standard single instruction retry is available.

Microdiagnostics under executive program control with program variable input patterns will be used for fault analysis. The executive software will use the standard fault tolerent techniques - two way lists with pointer verification before proceeding, stored data and programs will be tagged with redundant identification, read only programs will allow simple updating etc. Rollback and restart will be used for multi-processing with hardware or software error detection. The RCU monitors constantly for catastrophic faults - those not detected by the hardware and software tests. The standard time-out tests and system performance evaluation routines are run and controlled by the RCU. Power is conserved under program control by forcing n cycles between memory accesses, imposed by a counter with program changeable contents.

3.3 NOVELTY: Reconfiguration under hardware control in fault masking mode. Choice of computer fault masking, multiprocessing with fault masking and various forms of detection, multiprocessing with hardware error detection by comparison, multiprocessing with software error detection. Storage reliability by b-adjacent multiple error detecting and correcting codes. Self checking memory translators, checking circuits, and error-analysis ciruits. Use of power under program control.

3.4 INFLUENCES: 1. JPL Star - the total effort; 2. SRI, Techniques for the Realization of Ultra-Reliable Spaceborne Computers; 3. MIT -Draper Lab. for spaceborne multiprocessors; 4. Rapid emergence of LSI for feasibility of much redundant hardware.

3.5 HARD-CORE: Assuming that hard core means hardware, redundant or not, whose failure will produce undetected errors, there is no such hardware in this system. Hopefully, the software can be validated so that equal claims can be made for it.

4. JUSTIFICATION FOR THE SYSTEM
4.1 RELIABILITY EVALUATION: Architectural reliability evaluation by interactive program using exponential failure assumption for the units. Determination of component failure rates by analysis based upon previus data, experience, and analysis. Logic fault analysis of circuits in design stage by interactive fault simulation programs. Diagnostic pattern evaluation by simulation programs. Memory failure predictions by careful probabilistic fault analysis to predict error patterns, programmed computation of the circuit failure constants, programmed evalutation of reliability. Programmed analysis of RCU functions. Theoretical analysis of design, with hardware and software, in complicated situations (guided by simulation).

4.2 COMPLETENESS OF EVALUATION: Major unsolved problem.

4.3 OVERHEAD: Variable. In the processors about a 3 1/2 :1 logic count penalty is paid (the cost is much less). In the memory about a 3:2 storage penalty is paid. In the software the cost is unknown, but considerable.

4.4 APPLICABILITY: The concepts can be used elsewhere, the system is oriented toward space and extremely high reliability applications.

4.5 EXTENDABILITY: This computer is too reliable to fit into most other systems. For extension some of the fault tolerant techniques in the computer must be eased for better total system balance.

4.6 CRITICALITIES: Multitasking, as with all Executive-controlled recovery systems, is critical, achieved here with multiprogramming. Multiprocessing is an imposed condition, but small system simplifications would result if this condition were relaxed. Design validation tools are critical.

4.7 IMPLICATIONS: Architects must perform automated error and recovery analysis while doing system specification. Human analysis is too fallible. Hardware designers must have and use tools to do fault analysis as they design. After the first pass they must do design validation and iterate. Software designers must participate in the initial decisions, must produce more techniques for producing self-checking programs, and must produce the tools for program validation. Applications programmers must validate their programs (top down programming techniques will help), and must follow system rules (not so far known).

5. CONCLUSIONS
5.1 STATUS: This system is the collection of a group of ideas from a research project.

5.2 EXPERIENCE: None to report to date.

5.3 FUTURE: The system will be pursued only in a modified form as a paper study only.

5.4 ADVANCES: The problems of validation - hardware and software - will provide many a bottleneck for fault tolerant computing. The basic problem of definition of fault tolerant computing will be with us - do we consider any algorithm, procedure?

SURVEY ON FAULT-TOLERANT COMPUTER SYSTEMS

Albert L. Hopkins, Jr., MIT Draper Laboratory
Cambridge, Mass. 02139, May 1972

I. IDENTIFICATION

1.1. NAME: I am reporting on a long-term development effort which has been supported by different projects at different times. The following titles have been used for published reports:

* "A Fault-Tolerant Information Processing System for Advanced Control, Guidance,and Navigation".

* "Space Transportation System Data Management System".

In addition, an experimental three-processor three-scratchpad breadboard has been given the acronym CERBERUS for the three-headed dog in classical mythology. The acronym engendered the title: Controlled Error Recovery Behavior Employing Redundant Use of Scratchpads. In what follows, I use "the system" to mean the general concept, rather than a specific hardware design. This gives me the advantage of being able to be light on my feet and adapt to any situation before the fact.

1.2. RESPONSIBILITY: This work is in the Digital Development Group of the Charles Stark Draper Laboratory, a division of M.I.T.

1.3. SUPPORT SOURCES: So far all support has come from the NASA Manned Spacecraft Center.

1.4. PARTICIPANTS: MIT and NASA/MSC.

1.5. START: Work in this area began in 1966.

1.6. COMPLETION: Open ended. No end item is scheduled.

1.7. BIBLIOGRAPHY:

* R. L. Alonso, A. L. Hopkins, Jr., and H. A. Thaler, "Design Criteria for a Spacecraft Computer", Spaceborne Multiprocessing Seminar, pp. 23-28, NASA ERC, Boston Museum of Science, Oct. 1966.

* R. L. Alonso, A. L. Hopkins, Jr., and H. A. Thaler, "A Multiprocessing Structure", Digest of the First Annual IEEE Computer Conf., pp. 56-59, Chicago, Sept. 1967.

* A. I. Green et al., "STS Data Management System Design", MIT C.S. Draper Laboratory, Cambridge, Mass., Report E-2529, June 1970.

* A. L. Hopkins, Jr., "A Fault-Tolerant Information Processing Concept for Space Vehicles", IEEE Trans. Computers, Vol. C-20, pp. 1394-1403, Nov. 1971.

2. MOTIVATION

2.1. PURPOSE: Real time control, data acquisition and data management.

2.2. PHYSICAL ENVIRONMENT: In principle it could be any, but aerospace applications have predominated in design decisions.

2.3. COMPUTING ENVIRONMENT: Systems considered here are envisioned as largely self-contained information processing systems serving a variety of sensors and effectors including human operators. Such systems would be distributed, hierarchical and redundant. Central fault-tolerant multiprocessors would communicate over serial data buses to local processor complexes embedded in subsystems of the total system. A principal application considered for this approach was the Space Shuttle, where the Orbiter would have one central multiprocessor with adequate redundancy and spare hardware to be operational after three malfunctions. Each subsystem or group of identical subsystems would be served by single or redundant local processors, as appropriate, to fulfill the redundancy requirement for that subsystem or group.

The Booster stage of the Space Shuttle would, in this concept contain a system similar to that of the Orbiter, capable of communicating with it by way of a serial bus connecting the two central multiprocessors. All communication between a central multiprocessor and its local processors would be via a serial data bus.

2.4. COMPUTING OBJECTIVES FOR THE CENTRAL MULTIPROCESSOR: Variable from the order of 10E5 (i.e., 10 to the 5) to the order of 10E6 operations per second, with memory capacities of from 2E14 to 2E17 words of main random access memory. Input-output bandwidth 10E5 useful bits/sec on a 10E6 pulse-per-second bus. Reaction time order of 10 milliseconds.

2.5. RELIABILITY OBJECTIVES: Various types of objectives. One example is airline applications where less than one catastrophic system malfunction in 10E7 flights is sought. Other objectives are stated in terms of the number of individual malfunctions which can be tolerated in a flight, such as "Fail operational, fail operational, failsafe" (FO-FO-FS). The system is generally meant to be used in very high reliability applications.

2.6. DYNAMIC VARIABILITY: Graceful degradation is available as a means of exchanging performance for reliability.

2.7. PENALTIES: In the Space Shuttle application, as in possible aircraft applications, human life is concerned, as well as expensive flight hardware.

2.8. CONSTRAINTS: In Space Shuttle and aircraft, approximately 2 cubic feet, 120 lb., 300 watts. (Estimate for a central multiprocessor).

2.9. TRADEOFFS: Hardware efficiency is traded for 1) system reliability, 2) high malfunction coverage, 3) ease of program verification, 4) system flexibility.

The number of faults tolerated is variable through a combination of replication and sparing. Processors and memories can be added (deleted) to increase (decrease) processing and memory resources.

3. DESCRIPTION OF THE SYSTEM

3.1. ARCHITECTURE

3.1.1. CONFIGURATIONS

3.1.1.1. INTERCONNECTIVITY: The system makes extensive use of replication, and consequently connections have a high cost. Serial and byte-serial buses are used between basic units. Multiplexers are employed to prevent single unit malfunctions from spreading to all copies of a redundant bus. The canonical interconnection scheme is shown in Figure 1.

3.1.1.2. RANGE: No range limits have been determined, but the following numbers may be typical for an aerospace application. There are two current competitive conceptualizations of the system. These numbers represent the newer and less well developed concept.

* 6= Number of simultaneous job steps in process
* 3= Degree of replication of each processor-scratchpad
* 3= Number of spare processor-scratchpads
*21= Total processor scratchpads = 6 x 3 + 3
* 4= Number of independent memory blocks of 16K
* 3= Degree of replication of each block
* 3= Number of spare blocks
*15= Total memory block modules = 4 x 3 + 3

The numbe. of processor-scratchpads and memory blocks can be increased up to the practical bandwidth limit of the processor-memory bus and the I/O bus.

3.1.1.3. CAPABILITY: The order of 10E5 to 10E6 additions per second and the order of 2E14 words of memory. Three processors would be the smallest "sensible" number.

3.1.2. EXECUTIVE

3.1.2.1. MODES OF OPERATION: All programs are segmented into job steps which are dispatched by a floating form of executive. Each job step occupies one processor full time while it runs. Multiprocessing is the normal operating mode. Multiprogramming of each processor is not envisioned.

3.1.2.2. SOFTWARE ORGANIZATION: I/O processing is quasi-dedicated to one processor (i.e. it can float but does so only when malfunction makes it necessary). Executive, monitor, and reconfiguration programs are run on an as-needed basis by each processor as it finishes a job step.

3.2. FAULT TOLERANCE

3.2.1. FAULTS TOLERATED: Individual units (e.g. processor, memory unit, multiplexer) can malfunction one at a time with no restriction on what the nature of the malfunction is. Errors are masked by the system until it reconfigures itself to a fault-tolerant state.

3.2.2. FAULTS NOT TOLERATED: Certain malfunction pairs which occur simultaneously or close together in time can produce loss of data and many require a program restart. Incorrect specifications or program malfunctions can defeat the system. Systematic hardware malfunctions in which the same malfunction occurs in two redundant units can defeat the system.

3.2.3. TECHNIQUES: Two different concepts.

First concept: all processors are duplexed for detection. All scratchpads are triplexed for masked dump capability. Single instruction restart. Graceful degradation of processor-scratchpad groups. Triplex memory units with dedicated spares. Triplex buses with spares. Multiplexers isolate buses from failed groups of units.

Second concept: processor-scratchpad units are organized into groups of three under software control. Each looks for disagreement. If disagreement occurs, continue running to end of job step, then enter reconfiguration program. Graceful degradation of individual processor-scratchpad units (rather than groups of three scratchpads and two processors as in first concept). Triplex memory units with non-dedicated spares. Triplex buses with spa... Multiplexers isolate buses from failed individual units (rather than groups as in first concept).

In both concepts, software configuration control is used, which is valid as long as a working processor group, memory group, and bus-multiplexer group are available. Multiplexers participate in configuration control.

3.3. NOVELTY: Single instruction restart. Absence of interrupts and program rollbacks. Distributed monitor and reconfiguration functions. Use of multiplexers to isolate bus and unit malfunctions. Fault-tolerant clock. Hierarchical system with fault tolerance extended into subsystems.

3.4. INFLUENCES: Rapid emergence of LSI memories and processors has encouraged use of replication and partitioning with simple, identical units. Apollo Guidance Computer experience prompted elimination of interrupts and rollback for the sake of program verification. Carter and Bouricius for reliability models. Avizienis for concepts of fault tolerance.

3.5. HARD CORE: Assuming that hard core means non-redundant hardware, there is no hard core in this system. Configuration control is a software function using the available hardware to configure the system.

4. JUSTIFICATION

4.1. RELIABILITY EVALUATION: So far mostly geared toward FO-FO-FS. Some Probabilistic analysis. No reliability projections as yet since hardware has not been selected and failure rates are therefore not known.

4.2. COMPLETENESS OF EVALUATION: Hareware not selected, hence failure rate not known.

4.3. OVERHEAD: About 80% of the system is devoted to the achievement of fault tolerance.

4.4. APPLICABILITY: This concept is applicable to most digital control environments, depending on the economics of the application regarding fault tolerance.

4.5. EXTENDABILITY: Extendability probably does not apply, since the system is still loosely specified.

4.6. CRITICALITIES: The system is most cost-effective compared to other systems when the number of faults to be tolerated is high and where ultra-high reliability is sought. For single-fault tolerance and less high reliability, the system configuration might be changed.

4.7. IMPLICATIONS: In an ultra-high reliability application, specifications and programs must be proven to be correct. In this system, applications programmers must also segment their programs into short job steps.

5. CONCLUSIONS

5.1.1 STATUS: This is a research project with a breadboard experimental unit almost completed.

5.2. EXPERIENCE: None to report to date.

5.3. FUTURE: Some parts of the system still need to be designed and prototyped. Exeperiments must be conducted on a full-scale prototype system.

5.4. ADVANCES: The following will be beneficial.

*Demonstrated field experience with various fault-tolerant concepts.

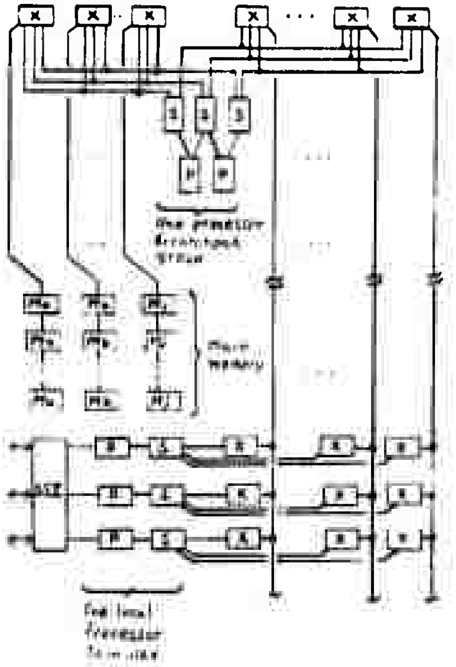
*Practical techniques for generating correct programs.

*Practical ways of verifying that a program is correct.

6. COMMENTS

The questionnaire was good in the sense of being thorough, but in my haste to respond to it I wonder if I have omitted significant material. An additional comment about this system is that it has been configured around integrated processors and memories which resemble those that are available today. The hardware efficiency number given in Section 4.3 is very misleading, because the cost of the hardware can be the least important cost of the system, if the hardware is conventional and not overly expensive. This system is expected to save in costs of system intgegration, program verification, and operational reliability experience. These savings may be far in excess of the hardware cost.

As an additional note, the replicated approach used here results in a coverage of 1.0 against single malfunctions. Coded approaches generally give lower coverage, difficult to quantify, and often impossible to verify in the field.



P...Processor
S...Scratchpad memory
M...Memory module
X...Multiplexor
SSI...Subsystem interface

SURVEY OF FAULT-TOLERANT COMPUTING SYSTEMS

W. L. Martin, Hughes Aircraft Company
Fullerton, California 92634, May, 1972

1. IDENTIFICATION

1.1. Name: Automatically Reconfigurable Modular Multiprocessor Systems (ARMMS)

1.2. RESPONSIBILITY: Astrionics Laboratory, Marshall Space Flight Center, NASA

1.3. SUPPORT: Same as 1.2

1.4. PARTICIPANTS: The participating organizations include NASA MSFC, Hughes (system design), M&S Computing, Inc. (executive software under subcontract to Hughes); Auburn University (executive control approaches under contract to NASA). Principal participants by name are as follows: NASA - Dr. J. B. White, Sherman Jobe; Hughes - W. L. Martin; M&S - T. T. Schansman; Auburn - Dr. David Irwin.

1.5. START: The date of conception was circa 1968 in a concept document written by Dr. White. MSFC has been developing technology under their Space Ultrareliable Modular Computer (SUMC) program since shortly thereafter. The system design effort being performed by Hughes was solicited in May, 1971 with a contract in October, 1971.

1.6. COMPLETION: The Hughes system definition contract will be completed in April, 1973. Construction of a breadboard or prototype may follow with completion date uncertain.

1.7. BIBLIOGRAPHY: Various planning documents have been written at NASA. Dr. White may be contacted for these. The Hughes effort is divided into three phases, with the Phase 1 report released on April 15, 1972. It is titled "Design of a Modular Digital Computer System", DRL 4, Phase 1 Report, Hughes Aircraft Company FR 72-11-450. Two other papers have been submitted for publication. Their fate is uncertain as of yet, but interested parties may obtain copies from W. L. Martin at Hughes. These are the following:

*J. L. Bricker, A Unified Method for Analyzing Mission Profile Reliability for Standby and Multiple Modular Redundant Computing Systems which allows for Degraded Performance (submitted to the IEEE Transactions on Reliability Theory).

*J. L. Bricker and W. L. Martin, Reliability of Modular Computer Systems with Varying Configuration and Load Requirements (submitted to 1972 IEEE Computer Society Conference).

2. MOTIVATION

2.1. PURPOSE: ARMMS is to be applicable through modularity to diverse types of space missions ranging from launch vehicles, to space stations to deep space probes.

2.2. PHYSICAL ENVIRONMENT: Spaceborne

2.3. COMPUTING ENVIRONMENT: See 2.1, 2.2.

2.4. COMPUTING OBJECTIVES: The motivating computing objective is to be able to configure systems which are fault tolerant through TMR or other redundant modes or to use the modules in parallel for high computing capacity and to be able to reconfigure from one type to the other dynamically. Maximum capacity in a non-redundant mode is to be "several million" additions per second.

2.5. RELIABILITY OBJECTIVES: One specific reliability objective is that the probability of survival of at least a simplex computer after 5 years should be at least 0.99 (with no on-board maintenance). The overall intent, however, is that the system should be able to be configured to meet specific mission reliability objectives whether they be stated in terms of maximum recovery time, number of failures tolerated, etc.

2.6. DYNAMIC VARIABILITY: As noted in 2.4, dynamic variability of configuration is one of the primary motivations.

2.7. PENALTIES: See 2.1.

2.8. PHYSICAL CONSTRAINTS: There are no explicit physical constraints except those implied by the nature of the intended spaceborne application. However, an implicit physical constraint is the difficulty of contriving an approach to a large (by aerospace standards) computing capability fault-tolerant design within the confines of weight and power budgets which may prevail for interplanetary missions.

2.9. TRADEOFFS: At the current stage of the design, there are many critical tradeoffs yet to be made.

* For a computer which will be built after 1975, what device complexity and failure rates should be assumed? Almost all aspects of the design are critically affected by this question. Some of the more crucial ones are the maximum complexity of any module; the degree to which processors must be sub-partitioned; the resulting cost in switching hardware; the maximum number of replicates of any one module type which must be accommodated; and the complexity of the configuration control software.

* The basic ARMMS concept developed by NASA incorporates a dedicated executive module rather than a floating executive. Resulting tradeoffs include specific definition of functions to be performed, specification of status monitoring and reconfiguration parameters, and a design approach which yields sufficiently high reliability for the executive module.

* The system architecture is not yet defined in any complete sense. Questions yet to be resolved include specific definition of allowed modes of operation; definition of the means of interconnecting the modules; placement and use of voters; use of error-correcting codes for memory data; maximum number of replicates per module class; specific techniques for memory data protection; and fault tolerance features within each module class. At present, we are making tradeoffs based on two major configuration alternatives. Although few tradeoff conclusions have been reached, the predominating evaluation criteria are almost certain to be the following:

* Implementation feasibility - Any design feature which does not seem to us to be feasible in any major sense (e.g., pin count, excessive power, design cost) will be rejected. We are not particularly interested in developing new theories or techniques of fault-tolerant computing but are very interested in developing a much-needed testbed based on the research performed over the last 5-10 years.

* Suitability to the multi-mode configuration requirements - ARMMS is intended to be usable in configurations ranging from a simplex computer to TMR with standby spares. Any feature which imposes excessive overhead cost for the benefit of one configuration at the expense of others is suspect. For example, added hardware per module for internal fault tolerance multiplies the hardware penalty paid in TMR mode.

3. SYSTEM DESCRIPTION: As seen from the tradeoff discussion above, no firm system description is possible now. Therefore, the responses in this section are necessarily brief and incomplete.

3.1. ARCHITECTURE
3.1.1. CONFIGURATION
3.1.1.1. INTERCONNECTIVITY: All processors, I/O, and executive controller may access all of main memory (a study of the desirability of identifying an additional level of memory, cache or task oriented was made, with a negative conclusion reached). The most probable scheme is a system of replicated busses with access control governed by the executive module. The nature of spaceborne I/O activity is biasing us toward a direct processor I/O data path which can be used for transmitting short bursts of data. The executive controller will monitor the other system modules via a time-shared bus. This bus ordinarily polls the modules in sequence but may be interrupted by the processors on task completion or other time-critical event. No direct interaction of modules of a given class (e.g., processor-to-processor) is planned.

3.1.1.2. RANGE: The general approach to achieving the large capacity mentioned previously is to maximize the individual processor performance so that throughput is not dependent on a large number of parallel instruction streams. (Three is a desirable upper limit.) The maximum main memory capacity is to be large enough (e.g., 256K-512K words) to support the high-throughput goals. The word length is to be 32 bits as dictated by the choice for the NASA SUMC processor. Cumulative I/O data rate capability is to be 10 million bits per second. In all cases, maximum number of modules per class (and the memory module capacity) will be determined primarily by reliability considerations. A least upper bound s 4 for each class.

3.1.1.3. CAPABILITY: (See 2.4.)

3.2. FAULT TOLERANCE: (The system is still too much conceptual to allow a decent response. All faults are to be tolerated. None are to be not tolerated. All techniques will be considered. Ask again in a year and let's see how it turned out.)

3.3. NOVELTY: On the one hand, there's nothing that one can point out as being fundamentally novel (this is true of most machines, I think). On the other hand, there are no machines that I know of that have successfully implemented a variable redundancy approach such as is being sought. The choice of a dedicated executive module is the only deviation at the block diagram level from other multiprocessors (but this module is a rather close parallel of the TARP in STAR).

3.4. INFLUENCES: JPL STAR; NASA ERC Modular Computer; NASA MSFC SUMC; IBM, "Architectural Study for a Self-Repairing Computer; SRI, Techniques for the Realization of Ultra-Reliable Spaceborne Computers.

3.5. HARD-CORE: The executive module is hard-core. The effect is to be minimized by simplifying the module as much as possible end by internal redundancy (which may ultimately result in replication).

4. JUSTIFICATION
4.1. RELIABILITY EVALUATION: To date, reliability has been evaluated solely by analysis (as described in the two papers mentioned in 1.7). Later in the effort, we expect to extend the analysis to include coverage and switch unreliability. We also expect to simulate the logical performance of the intermodule switches and to simulate the injection of faults.

4.2. COMPLETENESS OF EVALUATION: I'm not sure that I understand the question. But whatever you mean by design evaluation, I'm sure that I wish we had more time and money to do it better.

4.3. OVERHEAD: Since the configuration is dynamic, the percentages of resources attributed to the achievement of fault-tolerance also vary with time. An upper limit is probably 80%; a lower limit is probably 20% (in cost, logic, execution time, etc.).

4.4. APPLICABILITY: Applicability to other than space applications is questionable.

4.5. EXTENDABILITY: I think that it is more likely that the system design can usefully contract than that it can be usefully extended.

4.6. CRITICALITIES: The major difficulty of the design is the breadth of the goals. The critical problem is therefore to find a set of design choices which complies reasonably well with all the goals (e.g., we want high speed and capability but require low weight and power). However, I don't think that slight changes would critically affect the design. (Also, as a side observation, while one is in the midst of a system design, all choices seem critical, don't they?)

4.7. IMPLICATIONS: (Let me plead that this question seems too vague. I don't know where to start with a brief response.)

5. CONCLUSIONS
5.1.1 STATUS: The status is sufficiently described by the above comments, I think. In summation, we are about one-third of the way through a system definition phase.

5.2. EXPERIENCE: It appears that component technology is contributing more to the feasibility of highly reliable machines than architecture concepts are. As recently as 2 or 3 years ago, gate failure rate of 10E-7 per hour seemed optimistic. At present, gate failure rates of 10E-10 per hour are credible for the space environment. On the other hand, the assumption that dormant failure rates are a small fraction of active failure rates appears questionable. For a long-life machine in an unmanned environment, these two factors are of major significence to the system designer.

5.3. FUTURE: There are two conflicting possible futures of ARMMS. The pessimistic view is that it will go the way of 10 or 15 similar paper design efforts and will die with only a final report to commemorate its non-existence. The optimistic view is that it will appear sufficiently promising in concept that NASA will continue its development and eventually attach it to a mission. Planning is of course being directed toward the optimistic alternative.

5.4. ADVANCES: I cannot add anything to the lists of theoretical problem areas and needed areas of investigation which SRI described in its reports under contract NAS 12-33. In particular, I agree that there have been too few case studies which can be evaluated.

A major practical advance which is needed is the identification and exploitation of specific applications in which fault-tolerant machines can be justified economically. It is significant, I think, that the Bell ESS-I and System-360 FLT's instruction retry, etc., represent the most extensive application of fault-tolerance and diagnostic techniques. Both are in areas where the payoff for high reliability is great. Although aerospace applications have supported much of the research in fault-tolerant machines, I am skeptical that there is a sufficient mass of money there to lead to very widespread results in fielded systems. The situation is analogous to that which has existed for associative processing for 10 years, in that the glamour, concepts, and techniques are often apparent but cost considerations ultimately lead to more conventional choices.

Also, I wonder if "fault-tolerant computing" is too narrow a view and that many of the basic ideas would be applicable to a discipline of "Fault-tolerant systems". Perhaps there are other equally fertile, but less plowed, fields to be conquered.

6. COMMENTS: (See 5.4)

SLRVEY OF FAULT-TOLERANT COMPUTING SYSTEMS

John H. Wanslsy, Stanford Reeearch Instituta
Menlc Park, Ca. 94025, May 1972

1. IDENTIFICATION

I.1. NAME: SIFT (Softwsre-Implamentad Feult Tolarance), project: dasign study of a fault tolerant digital computer

1.2 RESPONSIBILITY: SRI

1.3 SUPPORT: NASA Langlay

1.4. PARTICIPANTS: J. Goldbsrg, K. Levitt, R. Ratner, J. Wensley, H. Zeidler, M. Green

1.5. START: August 1971

1.6. COMPLETION: Expsrimentel vereion 1973, final design 1974

1.7. BIBLIOGRAPHY: Technicsl Prograss Narratives 1-7; "SIFT - Softwsre Implemented Feult Tolarance," submittsd to FJCC 1972

2. MOTIVATION

2.I. PURPOSE: Control processing in an advancad tschnology traneport (aircraft) including navigation, stability augmentetion, angina control, instrument blind landings, etc.

2.2. PHYSICAL ENVIRONMENT: Airborna -- ths systam concapt howaver is epplicable to any environment.

2.3. COMPUTING ENVIRONMENT: Real-time

2.4. COMPUTING OBJECTIVES: Configuration scaleability, graceful degradation, traneportebility of concept to any procsssor or memory design.

2.5. RELIABILITY OBJECTIVES: Minimum probability of srronaous results, and of loss of computing capacity during aircraft flight.

2.6. DYNAMIC VARIABILITY: Variable dagrees of feult tolerance for tasks of differing criticality. Ability to trads off batwsen computing power and fault tolerance.

2.7. PENALTIES: Worst case - human lives; intermediata - sircreft damage; least case - nead to ebort flight objectives.

2.8. CONSTRAINTS: hardwera must be daeignsd with waight, eize and power raquirements consiatant with aircraft raquiraments. The basic concept of the systsm ie only effacted by ths constraint that maintenece cannot be cerrisd out during flight.

2.9. TRADEOFFS: Computing capacity vs. reliability

3. DESCRIPTION: A systsm erchitecture in which fault toleranca is schiavad with no special fault-tolarant hardware.

3.1. ARCHITECTURE: A multi-computer (see Fig 1)

3.1.1. CONFIGURATIONS: No constraints sre prssent on procassor or memory dssign. Fault toleranca is achieved by ths restrictad connaction of processors and memorias, and by software control.

3.1.1.1. INTERCONNECTIVITY: Processing modulas comprising e procaseor and memory are connectad via multiple busses. Tha interconnection is designed so that processors may only rasd (and not write) into the memory of other modulas. The busses ara used as alternativa routes rathsr than as multipls simultaneous tranemiasion paths.

3.1.1.2. RANGE: Tha scale of the system is not frozen in the architectural concept. It is envisaged that a minimum coufiguration would contain thres procassing modulee and three busses. The deaign does not (at present) placa any limit on the maximum configuration. Greatar feult tolerance is achieved with a larga numbar of low-capability units rather than with a small number of high capability units.

3.1.1.3. CAPABILITY: Ths dssign concapt is valid ovar the entire range of processor, memory and bus capability.

3.1.2. EXECUTIVE: Executiva control (allocation, schaduling, dispetching, reconfigurstion, etc.) is achievsd by raplicstad softwsra sxacutive routines.

3.1.2.1. MODES: Tha primary operating moda is on rapetitive rsal-time cslculstions involving many loosely connsctad tasks. Both multiprocassing and multiprogramming ara included.

3.1.2.2. SOFTWARE: Tesks ara multiprogrammed in each processing module. Each task for which fault toleranca is demanded ie prasant in more than one module. A loose synchronization of task processing is schiavsd by tha system exacutive (which itsalf is rsplicated and loosaly synchronized). Software fault detsction is csrrisd out batwaen each itsration of a task bafore arroneous rasults era used by the next itsration or other tasks.

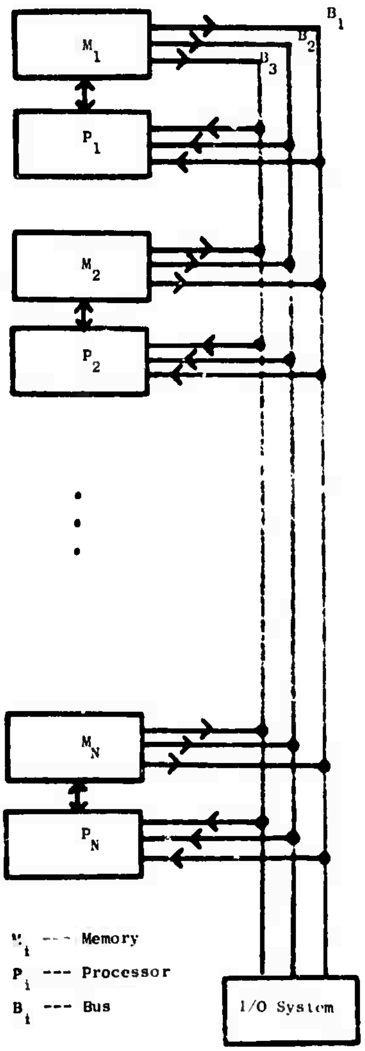


Figure 1 System Configuration

3.2 FAULT TOLERANCE

3.2.1. FAULTS TOLERATED: The system is tolerant to faults in any unit (processor, bus or memory). The faults may be the erroneous result of an action (calculation, transmission or storage) or the failure of a unit to carry out any action.

The system handles transient, and permanent faults, treating long-term intermittent faults as permanent. The reconfiguration procedures can bring back into service a unit that was at one time subject to faults but has since recovered or been repaired.

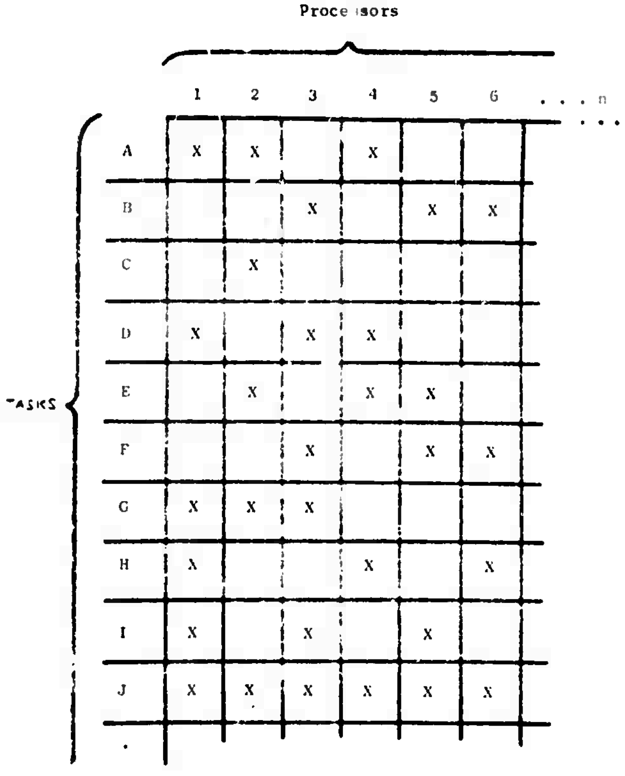
The cause of the fault (electrical, mechanical, etc.) is not of importance, the only consideration is whether the results of actions in replicated units agree or disagree.

Independent multiple faults can be tolerated to any degree depending on the extent of replication of the function. Correlated faults both in hardware and software are not tolerated to the same extent as uncorrelated faults. The loose synchronization of tasks assists in tolerating faults which are correlated in time rather than function. One-shot faults do not cause removal or reconfiguration of units from the system. The propagation of a fault from any unit to another can only occur if both units are faulty.

3.2.2. FAULTS NOT TOLERATED: Multiple correlated faults that are not detected by a voting procedure, or by repeating the task, e.g., simultaneous identical failure of two memory units when threefold replication is used. Massive faults that reduce the system to a size too small to handle the computing load.

3.2.3. TECHNIQUES: Fault detection is carried out by replication and voting. Other fault detection methods (hardware or software) are compatible with and can be incorporated into the system concept. Fault correction (or tolerance) is achieved by voting after replication in most cases but can be supplemented by other techniques such as repetition or roll-back. The allocation of resources to tasks can be changed either when faulty units are removed or when the mission demands different fault tolerance and/or computational power.

3.3. NOVELTY: Lack of need for special hardware units to facilitate fault tolerance. Ability to trade off fault tolerance with computing power. Applicability of the system concept to different memory or processor designs.



Processors

TASKS

	1	2	3	4	5	6	. . . n
A	X	X		X			
B			X		X	X	
C		X					
D	X		X	X			
E		X		X	X		
F			X		X	X	
G	X	X	X				
H	X			X		X	
I	X		X		X		
J	X	X	X	X	X	X	
.							

Figure 2 An Example of Task/Processor Allocation

3.4. INFLUENCES: The design is influenced by the need to avoid special hardware for fault tolerance, freezing fault tolerance techniques at design time, designs geared to particular size and speed computers.

3.5. HARD CORE: I don't mean anything by "hard core" in the system described. I can imagine other system concepts in which the term has meaning (but little utility).

4. JUSTIFICATION

4.1. RELIABILITY EVALUATION: By analysis, assuming uncorrelated faults of equal probability in each part of the system (chip, connector, cable, etc.).

4.2. COMPLETENESS OF EVALUATION: Incomplete.

4.3. OVERHEAD: Variable, typically a 3-1 cost penalty is paid for fault tolerance.

4.4. APPLICABILITY: General; the design is applicable to any environment.

4.5. EXTENDABILITY: Unlimited.

4.6. CRITICALITY: Multiprocessing is critical. Multiprogramming is highly desirable (see Fig 2).

4.7. IMPLICATIONS: There are no implications on the hardware designers of processors and memories. The busses are constrained in the way units communicate. The applications' software must be implemented so that input data for a program is fetched by calling a general system routine which carries out fault detection and correction.

5. CONCLUSIONS

5.1. STATUS; A conceptual design of hardware, software and fault tolerance procedures exists.

5.2. EXPERIENCE: Software design studies show that the time and memory requirements of the fault detection and correction routines are reasonable.

5.3. FUTURE: The projection is for an experimental version of the system to be built.

5.4. ADVANCES: I/O units with fault tolerance capability.

Barry R. Borgerson, Computer Systems Research Project
University of California, Berkeley, May 1972.

1. IDENTIFICATION
1.1. NAME: PRIME

1.2. RESPONSIBILITY: Computer Systems Research Project (CSRP), U. C. Berkeley

1.3. SUPPORT: ARPA - Contract No. DAHC70 15 C 0724

1.4. PARTICIPANTS: Herbert B. Baskin, Principal Investigator; Roger Roberts, Principal Programmer; Barry R. Borgerson, Head, Hardware R & D.

1.5. START: 7/1/70

1.6. COMPLETION: " prototype to be running about 7/73

1.7. BIBLIOGRAPHY

*Baskin, Herbert B., Barry R. Borgerson and Roger Roberts, "PRIME - An Architecture for Terminal Oriented Systems," Proceedings of the 1972 SJCC, AFIPS Press pp. 431-437.

*Borgerson, Barry R., "A Fail-Softly System for Time Sharing Use," Digest of the 1972 International Fault Tolerant Computing Symposium.

*Quatse, Jesse T., Pierre Gaulene and Donald Dodge, "The External Access Network of a Modular Computer System," Proceedings of the 1972 SJCC, AFIPS Press, pp. 783-790.

*Fabry, R. S., "Dynamic Verification of Operating System Decisions," CSRP Document No. P-14.0, Univ. California, Berkeley, Issued 2/23/72.

*Borgerson, Barry R., "Spontaneous Reconfiguration in a Fail-Softly Computer Utility," CSRP Document No. P-15.0, Univ. California, Berkeley, Issued 2/29/72.

*Borgerson, Barry R., "Dynamic Confirmation of System Integrity," CSRP Document No. P-19.0, Univ. California, Berkeley, Issued 4/24/72.

2. MOTIVATION
2.1. PURPOSE: General-purpose, interactive, multi-access computing.

2.2. PHYSICAL ENVIRONMENT: Ground based

2.3. COMPUTING ENVIRONMENT: Remote access over telephone lines and eventually over the Arpanet.

2.4. COMPUTING OBJECTIVES: This is not the primary motivating area in our system design. We anticipate that the original configuration of PRIME will support about 100 users with a worst case response time of less than two seconds for trivial jobs.

2.5. RELIABILITY OBJECTIVES: Because we will be able to repair units as they become faulty, we are aiming for continuous availability. The system performance should never degrade below 75% of its peak capacity.

2.6. DYNAMIC VARIABLITY: Performance cannot be dynamically traded for reliability. However, provisions may someday be added which will allow dynamically trading performance for intraprocess integrity (See Section 6).

2.7. PENALTIES: The effects of intraprocess data contamination (See Section 3.3.2) due to system failures will strongly depend on the nature and purpose of the process. There seems to be no way to generalize about this. If the system itself were to crash, this would no doubt lead to a loss of revenue if PRIME were transferred to a commercial environment.

2.8. CONSTRAINTS: There are no specific constraints of size, weight, and power. The self-imposed constraint on cost is to try to build a fault-tolerant system that is as close in cost as possible to any current system with comparable power and capabilities.

2.9. TRADEOFFS: (Too complicated to deal with briefly; see Sections 4.4, 4.6 and 6.)

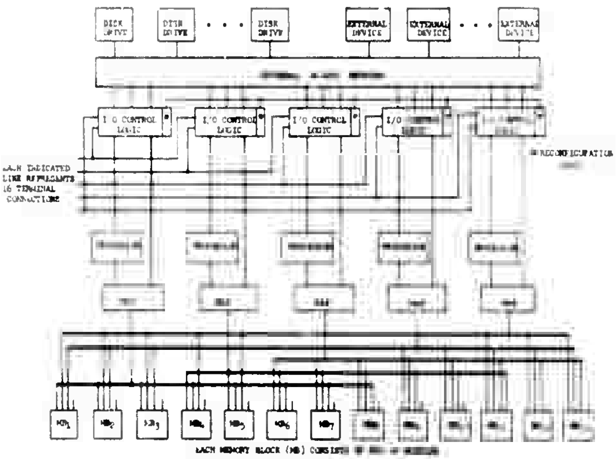
3. DESCRIPTION
3.1. ARCHITECTURE
3.1.1. CONFIGURATION
3.1.1.1. INTERCONNECTIVITY: Figure 1 is a block diagram of PRIME. The External Access Network (EAN) allows any processor to connect to any disk drive, external device, or other processor. Each processor has three such independent paths into the EAN. The EAN connectivity remains universal over the different system sizes. Universal switching between all processors and all memory blocks is not provided. Instead, each processor always connects to exactly 64K of memory regardless of the size of the system.

3.1.1.2. RANGE: The PRIME architecture will usefully accommodate from 3 to about 30 processors. Each processor could connect to from 16K to 128K of primary memory. Depending on the type of disk drives used, from 1 to 5 drives per processor would be reasonable. The current system has been designed to operate with from three to eight processors without requiring any additional hardware or software design. Useful memory sizes range from 64K to about 256K. Disk drives range from about six to 24. Each processor to be used in the initial implementation of PRIME will be a Meta 4 (Digital Scientific Corp.). The Meta 4 is a general-purpose, 16-bit, 32-register, 90ns-cycle time microprocessor. The memory is 33 bits wide, about 600 ns cycle, and made from 1024-bit MOS chips. The disk drives are double (track) density 2314-type drives that have been modified to transfer information on two heads at a time. The initial configuration will have five processors, 104K of memory, and 15 disk drives.

3.1.1.3. CAPABILITY: The capability is not accurately known at this time.

3.1.2. EXECUTIVE
3.1.2.1. MODES: At any given time, one processor is designated the Control Processor (CP) while the rest function as Problem Processors (PPs). User processes are run on the PPs. Multiprogramming is not used, but processes are overlap-swapped. In order to achieve a very high interprocess integrity, it was decided never to let two processes share memory; hence, cooperative-process multiprocessing is not possible with PRIME.

Figure 1 A Block Diagram of the PRIME System



3.1.2.2. SOFTWARE: The system software is divided into three sections. There is the Central Control Monitor (CCM) which runs on the Target Machine of the CP; the Extension of the Control Monitor (ECM) which resides directly in the microcode of each processor; and the local Monitor (LM) which runs on the Target Machine in the PPs. The CCM is responsible for scheduling processes, allocating resource, and consummating interprocess message transfers. The ECM includes the disk, terminal, and communication controllers, logic for double-checking critical CCM decisions, bootstrap logic, and some intelligence to deal with reconfiguration. The LM contains the file and working-set management systems. The CCM does not get involved with a process after it has started the process up. The procedure followed by the CCM is to allocate the necessary resources, initiate the roll in, and let the LM and ECM take over from there. The CCM will not get involved again until the process either times out or blocks itself. The LM deals only with user processes; it is completely isolated from the rest of the system. Because of this, users will be free to provide their own LM if they do not like the standard one provided.

3.2. FAULT TOLERANCE

3.2.1. FAULTS TOLERATED: PRIME will tolerate all internal faults. That is, the system is expected to continue operating even in the presence of any arbitrary software or hardware faults. The system will reconfigure to run without any piece of hardware that becomes faulty, and mechanisms exist for limiting the effects of any software fault. PRIME has been designed to provide continuous service to (almost) all terminals. In most cases, a faulty unit will be repaired and returned to service before another failure occurs. However, the system will still continue to operate with a substantial part of the resources removed from active use. The system should almost never degrade to below 75 percent of its maximum capacity. In addition to continuity of some minimum service, interprocess integrity violations are prevented at all times; this includes the relatively unstable periods between the onset of a fault and the detection and isolation of the faulty unit.

3.2.2. PAULTS NOT TOLERATED: Only environmental faults are not tolerated by PRIME. The most common of these faults would be in the A.C. power and air conditioning. Since it is easy to see how to back these resources up, no effort has been made to incorporate fault tolerance with respect to these units within PRIME. While PRIME as a system will continue to run in spite of internal failures, individual processes may occasionally get clobbered. That is, no special provisions have been made in PRIME to guarantee interprocess integrity. Hence transient failures will frequently cause contamination of information for some process. Also hard failures will often clobber one process before being detected. The most serious disruption will probably occur when a disk drive fails. When this happens, all of the processors that were using that drive will be suspended until an operator can recover their data, either by moving the disk pack to another drive, or recovering from tapes in the unlikely event of a head crash. But even in this worst-case catastrophy, only a small part of the users (about 7 percent in the initial system) will be affected.

3.2.3. TECHNIQUES; The basic system-wide technique used to achieve fault tolerance is to allow the system to degrade gracefully by reconfiguring to run without any faulty units. At the heart of the scheme is a distributed architecture with a multiplicity of all functional units except the EAN, which is designed to fail softly on its own. Fault detection is accomplished by a variety of methods which include parity memory and buses, surveillance tests on each processor after each job step, a double check on all critical system-wide decisions made by the CP, and fault injection in such areas as error detectors and the seldom used reconfiguration logic. After a fault is detected, an initial reconfiguration causes a processor not involved in the detection to become the new CP. This virtual "hard-core" then initiates diagnostics to locate the faulty unit, isolate it, and reconfigure the system to run as efficiently as possible without it. A small amount of dedicated hardware associated with each processor guarantees that the initial reconfiguration will be accomplished properly. It is possible to logically isolate each major unit at its system boundaries so that the system can run fine-mesh diagnostics or exercise the hardware to aid in locating the faulty component. In the case of a failure of the isolation logic, any unit can be dynamically powered down to provide guaranteed isolation from the rest of the system.

3.3. NOVELTY: The distributed nature of the system, including the distributed intelligence in the form of the ECMs, provides a very powerful structure whereby fault tolerance is achieved without the use of any "reliable" hardware. Very high-performance low-cost disk drives have been incorporated in such a way as to allow these devices to be used as second level storage, third level storage, and the swapping medium. By distributing these three functions over many identical physical units, very high availability is achieved at what is actually a lower cost and with higher overall performance than would be possible with three distinct types of units. PRIME automatically responds to faults by reconfiguring to run without the faulty unit. Since there is a multiplicity of all functional units except the EAN, it is quite easy to run without any particular unit. Rather than make the EAN "reliable," a more economical approach was taken whereby carefully controlled failure modes were designed into it. This results in a failure within the EAN manifesting itself as a failure of a small number of ports, which is equivalent to losing whatever is attached to those ports, and the system was already designed to handle that eventuality. The reconfiguration structure is also very interesting. Whenever a failure is detected, an initial reconfiguration takes place which establishes a new processor as the CP. The new CP, which is one not involved in the detection of the fault, is then used as the temporary "hard-core" to initiate diagnostics, locate the fault if indeed it exists, and remove the faulty unit from the system. The distributed intelligence of PRIME has been used to provide double checking on all critical system functions, which in turn guarantees that there will be no interprocess interference. Probably the most unusual general feature of PRIME with respect to fault tolerance is that it is self-diagnosing and self-repairing without incorporating any "hard-core."

3.4. INFLUENCES: Many previous efforts have, of course, influenced us, but no single system stands out as having special influence.

3.5. HARD-CORE: No, there is no "hard-core" in PRIME. Instead, the concept of a "floating hard-core" exists whereby a working processor is pressed into service as the Control Processor whenever a malfunction is detected. This is consistent with the overall system philosophy of not having any "reliable" hardware anywhere in the system.

4. JUSTIFICATION

4.1. RELIABILITY EVALUATION: Reliability will be demonstrated by stimulation of faults.

4.3. OVERHEAD: The cost of the additional hardware that has been incorporated in PRIME specifically for fault tolerance is less than 10 percent of the total hardware cost of the system. Less than 10% of each processor's useful time is devoted to fault-tolerant functions, since the surveillance programs are run during what would otherwise be idle time while processes are being swapped.

4.4. APPLICABILITY: PRIME has been very carefully designed to perform economically in a particular environment. If it was to be used in another environment, a detailed analysis would have to be performed to determine what changes would have to be made to allow it to perform adequately in the new environment. In particular, most other potential environments would require that steps be taken to guarentee intreprocess integrity.

4.6. CRITICALITIES: The choice of disk drives is quite critical since a low cost/bit is necessary as well as a high bandwidth due to the different functions these drives perform. Since 3330-type drives were not available when this design started, 2314-type drives were selected and modified to transfer at 5MHz. Also, the EAN had to be carefully designed with well-specified failure modes. However, the primary memory and the processors are simply "off the shelf" items. As for goals, the decision to not provide intraprocess integrity checks has been carefully exploited in the design of PRIME and has provided a very substantial cost savings.

4.7. IMPLICATIONS: Heavy reliance is placed on periodic checking of hardware rather than concurrent checking. Thus, the ability to inject faults into the appropriate areas has been a difficult requirement placed on all of the hardware designers. The most notable software requirement imposed by the basic design is the clear division of the operating system into three parts, one of which can be furnished by a user. The only significant requirement placed on a user is that he must be aware that no intraprocess integrity checks are made (just like in all current time-sharing systems)

5. CONCLUSIONS
5.1. STATUS: The design of PRIME is about 95 percent completed, and implementation has begun on both the hardware and software. The first version capable of reconfiguring in the present of a failure should be running by August, 1973.

5.2. EXPERIENCE: The main conclusion that the respondent can make regarding the design of PRIME is that by somewhat limiting the goal of the PRIME system, it was possible to create a system that should exhibit excellent fault-tolerant characteristics at a much lower incremental cost than that of any other fault-tolerant system known to him.

5.3. FUTURE: The near future will be devoted to building PRIME. After that, evaluation and tuning will take place with connection to the Arpanet very likely.

5.4. ADVANCES: It seems that the most significant development that would aid the PRIME system would be the availability of a general-purpose, self-checking processor. Since 100 percent self-checkability is extremely difficult to design into a processor, the best course of action here seems to be to wait for LSI processors of sufficient power to be built. These processors should be so inexpensive, compared to the rest of the hardware cost, that running two of them simultaneously and comparing outputs should be a very attractive procedure economically. In fact, the current processing element in PRIME could be broken into several subprocessors: one for communications, one for the disk controller, one for the terminal controller, two for the Target Machine, etc. Probably only the Target Machine processor would have to be duplexed because the others can have independent checks on the validity of their results. With this procedure, intraprocess integrity would be possible at an insignificant incremental cost.

6. COMMENTS: I eagerly await the results from this SRI study. I have experienced a great deal of difficulty locating any other efforts at designing and building what I consider to be truly gracefully degrading self-repairing systems. Most of the effort in fault-tolerant computing to data seems to be centered around military systems, or even moreso, around space exploration systems. This typically dictates that a fixed amount of computing power be made available at all times; hence, the lack of action around fail-softly systems. Of course, by providing fault tolerance through graceful degradation, very substantial cost savings can be realized over the "redundant" methods. In addition to allowing the system's performance to degrade in the presence of faults, we have chosen not to guarantee intraprocess integrity. The combination of these two concessions has allowed us to design a very economical fault-tolerant time-sharing system. There is little doubt that the anticipated degradations will be quite acceptable for a wide range of applications. The lack of intraprocess-integrity guarantees, however, will be a limiting factor in expanding this architecture into other areas. Of course, hardware provisions could be added to guarantee introprocess integrity, and the resultant system would still be more economical than most other fault-tolerant systems. A more promising approach, and one which we will undoubtedly explore in the reasonably near future, is to leave the hardware as is and run critical programs twice on two different processors. This will allow the system cost to remain very low, and will also allow intraprocess integrity guarantees. Thus, only those processes that that need the guarantee will have to pay for this added feature. A final aspect of the PRIME architecture that should be investigated is whether it can more economically provide a guaranteed computing power in some environments than can be provided by a "redundant" system. It can be overbuilt by an amount sufficient to guarantee that its degraded condition is powerful enough to handle the necessary computing, with background power available most of the time.

Jacques J. Delamare, Electronique Marcel Dassault (E.M.D.), 55, quai Carnot, 92 - Saint-Cloud France, June 1972

1. IDENTIFICATION
1.1. NAME: MECRA (Maquette Experimentale de Calculateur a Reconfiguration Automatique).

1.2. RESPONSIBILITY: E.M.O. (Electronique Marcel Oassault).

1.3. SUPPORT: Support has three sources: O.G.R.S.T. (Delegation Generale a la Recherche Scientifique) with preliminary studies; O.R.M.E. (Direction des Recherches et des Moyens d'Essais) with realization of MECRA project; E.M.O. (Electronique Marcel Dassault) in each case.

1.4. PARTICIPANTS: Jacques J. Delamare, Gerard Germain, Jean-Claude R. Charpentier, all of E.M.O., and four researchers from "Centre de Calcul Numerique de Toulouse".

1.5. START: May 1970

1.6. COMPLETION: July 1972, this consists of a demonstration of fault tolerance and reconfiguration capabilities. Evaluation of reliability performance is expected to be in Autumn 1972.

1.7. BIBLIOGRAPHY: "The MECRA: a Self Reconfigurable Computer for Highly Reliable Process", IEEE vol C-20 no. 11, pp. 1382-1388, Nov. 1971. A report also due end of 1972.

2. MOTIVATION
2.1. PURPOSE: The system was conceived for research in fault-tolerant computer architecture, feasibility, and reliability evaluation. The idea for further development is a real-time medium-sized computer for aircraft.

2.2. PHYSICAL INVIRONMENT: System operates in EMD laboratories.

2.3. COMPUTING ENVIRONMENT: A single peripheral allows communication with MECRA.

2.4. COMPUTING OBJECTIVES: Main objectives of the project were not computing objectives. However addition and multiplication are performed with 11 decimal digits plus sign operands. Complete addition needs less than 300 microsec. Such delays relate to the cycle time of microprogram memory (1 microsec), to response time of discrete circuits, to unused time intervals in each microinstruction cycle, (allowing hardware modifications), and lastly by the microsoftware package (allowing reconfiguration).

2.5. RELIABILITY OBJECTIVES: Practical experience and a concrete basis for evaluation such as:
reliability gain with different kinds of redundancy,
hardcore contribution in failure probabilities,
hardcore contribution with different architectures,
reliability gain with reconfiguration,
cost increase in control with reconfigurability,
lost time due to reconfiguration (during and after),
hardcore response time with respect to computing time.
These reliability objectives were only of interest for high probabilities of success (probabilities higher than .9).

2.6. DYNAMIC VARIABILITY: Computing speed but not accuracy may degrade with reconfiguration (20% maximum). Performance cannot be exchanged for increased reliability such as : two processors each one having its own job, switched to parallel processing on the same job and checking one another.

2.7. PENALTIES: Penalties from faulty operation can be of several kinds: /Loss of time due to recovery processes, lessened performance after self-reconfiguration, loss of service./ Manual interventions have not been investigated, but will be necessarily improved as a consequence of self-testing and self-healing capabilities of MECRA.

/SRI note: The text enclosed in slashes is an SRI paraphrase of the original survey response./

2.8. CONSTRAINTS: Circuitry size might not exceed four times the size of the equivalent irredundant computer.

4.7. IMPLICATIONS: Heavy reliance is placed on periodic checking of hardware rather than concurrent checking. Thus, the ability to inject faults into the appropriate areas has been a difficult requirement placed on all of the hardware designers. The most notable software requirement imposed by the basic design is the clear division of the operating system into three parts, one of which can be furnished by a user. The only significant requirement placed on a user is that he must be aware that no intraprocess integrity checks are made (just like in all current time-sharing systems).

5. CONCLUSIONS

5.1. STATUS: The design of PRIME is about 95 percent completed, and implementation has begun on both the hardware and software. The first version capable of reconfiguring in the present of a failure should be running by August, 1973.

5.2. EXPERIENCE: The main conclusion that the respondent can make regarding the design of PRIME is that by somewhat limiting the goal of the PRIME system, it was possible to create a system that should exhibit excellent fault-tolerant characteristics at a much lower incremental cost than that of any other fault-tolerant system known to him.

5.3. FUTURE: The near future will be devoted to building PRIME. After that, evaluation and tuning will take place with connection to the Arpanet very likely.

5.4. ADVANCES: It seems that the most significant development that would aid the PRIME system would be the availability of a general-purpose, self-checking processor. Since 100 percent self-checkability is extremely difficult to design into a processor, the best course of action here seems to be to wait for LSI processors of sufficient power to be built. These processors should be so inexpensive, compared to the rest of the hardware cost, that running two of them simultaneously and comparing outputs should be a very attractive procedure economically. In fact, the current processing element in PRIME could be broken into several subprocessors: one for communications, one for the disk controller, one for the terminal controller, two for the Target Machine, etc. Probably only the Target Machine processor would have to be duplexed because the others can have independent checks on the validity of their results. With this procedure, intraprocess integrity would be possible at an insignificant incremental cost.

6. COMMENTS: I eagerly await the results from this SRI study. I have experienced a great deal of difficulty locating any other efforts at designing and building what I consider to be truly gracefully degrading self-repairing systems. Most of the effort in fault-tolerant computing to date seems to be centered around military systems, or even moreso, around space exploration systems. This typically dictates that a fixed amount of computing power be made available at all times; hence, the lack of action around fail-softly systems. Of course, by providing fault tolerance through graceful degradation, very substantial cost savings can be realized over the "redundant" methods. In addition to allowing the system's performance to degrade in the presence of faults, we have chosen not to guarantee intraprocess integrity. The combination of these two concessions has allowed us to design a very economical fault-tolerant time-sharing system. There is little doubt that the anticipated degradations will be quite acceptable for a wide range of applications. The lack of intraprocess-integrity guarantees, however, will be a limiting factor in expanding this architecture into other areas. Of course, hardware provisions could be added to guarantee introprocess integrity, and the resultant system would still be more economical than most other fault-tolerant systems. A more promising approach, and one which we will undoubtedly explore in the reasonably near future, is to leave the hardware as is and run critical programs twice on two different processors. This will allow the system cost to remain very low, and will also allow intraprocess integrity guarantees. Thus, only those processes that that need this guarantee will have to pay for this added feature. A final aspect of the PRIME architecture that should be investigated is whether it can more economically provide a guaranteed computing power in some environments than can be provided by a "redundant" system. It can be overbuilt by an amount sufficient to guarantee that its degraded condition is powerful enough to handle the necessary computing, with background power available most of the time.

Jacques J. Delamare, Electronique Marcel Dassault (E.M.D.), 55, quai Carnot, 92 - Saint-Cloud France, June 1972

1. IDENTIFICATION

1.1. NAME: MECRA (Maquette Experimentale de Calculateur a Reconfiguration Automatique).

1.2. RESPONSIBILITY: E.M.D. (Electronique Marcel Dassault).

1.3. SUPPORT: Support has three sources: D.G.R.S.T. (Delegation Generale a la Recherche Scientifique) with preliminary studies; D.R.M.E. (Direction des Recherches et des Moyens d'Essais) with realization of MECRA project; E.M.D. (Electronique Marcel Dassault) in each case.

1.4. PARTICIPANTS: Jacques J. Delamare, Gerard Germain, Jean-Claude R. Charpentier, all of E.M.D., and four researchers from "Centre de Calcul Numerique de Toulouse".

1.5. START: May 1970

1.6. COMPLETION: July 1972, this consists of a demonstration of fault tolerance and reconfiguration capabilities. Evaluation of reliability performance is expected to be in Autumn 1972.

1.7. BIBLIOGRAPHY: "The MECRA: a Self Reconfigurable Computer for Highly Reliable Process", IEEE vol C-20 no. 11, pp. 1382-1388, Nov. 1971. A report also due end of 1972.

2. MOTIVATION

2.1. PURPOSE: The system was conceived for research in fault-tolerant computer architecture, feasibility, and reliability evaluation. The idea for further development is a real-time medium-sized computer for aircraft.

2.2. PHYSICAL INVIRONMENT: System operates in EMD laboratories.

2.3. COMPUTING ENVIRONMENT: A single peripheral allows communication with MECRA.

2.4. COMPUTING OBJECTIVES: Main objectives of the project were not computing objectives. However addition and multiplication are performed with 11 decimal digits plus sign operands. Complete addition needs less than 300 microsec. Such delays relate to the cycle time of microprogram memory (1 microsec), to response time of discrete circuits, to unused time intervals in each microinstruction cycle, (allowing hardware modifications), and lastly by the microsoftware package (allowing reconfiguration).

2.5. RELIABILITY OBJECTIVES: Practical experience and a concrete basis for evaluation such as:
reliability gain with different kinds of redundancy,
hardcore contribution in failure probabilities,
hardcore contribution with different architectures,
reliability gain with reconfiguration,
cost increase in control with reconfigurability,
lost time due to reconfiguration (during and after),
hardcore response time with respect to computing time.
These reliability objectives were only of interest for high probabilities of success (probabilities higher than .9).

2.6. DYNAMIC VARIABILITY: Computing speed but not accuracy may degrade with reconfiguration (20% maximum). Performance cannot be exchanged for increased reliability such as : two processors each one having its own job, switched to parallel processing on the same job and checking one another.

2.7. PENALTIES: Penalties from faulty operation can be of several kinds: /Loss of time due to recovery processes, lessened performance after self-reconfiguration, loss of service./ Manual interventions have not been investigated, but will be necessarily improved as a consequence of self-testing and self-healing capabilities of MECRA.

/SRI note: The text enclosed in slashes is an SRI paraphrase of the original survey response./

2.8. CONSTRAINTS: Circuitry size might not exceed four times the size of the equivalent irredundant computer.

3. DESCRIPTION

3.1 ARCHITECTURE

3.1.1. CONFIGURATIONS

3.1.1.1. INTERCONNECTIVITY: See IEEE paper. The basic configuration is a microprogrammed monoprncesaor with a bus architecture. A restriction csn be seen here since addresses sre binsry coded, whereas data are Decimal Hamming coded. This has no importance for the purpose of the project, but would not have been used on a prototype.

3.1.1.2. RANGE: Control Unit Configuration:

Maximim	Minimum
4 counters	3 counters
3 spare counters	0 apare counters
8 registers	6 registers
4 spare regiaters	0 spare registers
3 multiplication processors	1 multiplication processor
2 addition processors	1 addition processor
4 'and ' logic processors	3 or 2
4 'or' logic processors	3 or 2
4 'exclusive or' processors	3 or 2
4 'inverter' blocks	3 or 2

Note: Any logic function can fail completely and csn be reconfigured with three other functions. In several casea a failed logic function csn be reconfigured with only two other function.

Memory configuration: Three memory blocks - 4 K 16-bit words. Each memory block has its own address decoder circuits. At each memory cycle a 48-bit word is read or written; this word contains two identical words of 24 bits each, so thst any one of the three blocks can be declared void and the computer still runs if the other two operate properly. Efficiency of address error detection reaches 50% on each memory block. After sny read restore cycle, each eight-bit byte (6 bytes) is checked and is switched or not on busses. Then error detection efficiency is 50% with instructions or microinstruction (if there is only one erroneous bit) and 100% with data (if there is one or two erroneous bit).

3.1.2 EXECUTIVE

3.1.2.1. MOOES: MECRA is a monoprocessor.

3.1.2.2. SOFTWARE: There are three working modes on the computer: user mode, test-diagnosis mode, decision and reorganization mode.

a) In the USER mode the computer executes the user program.

b) The TEST-DIAGNOSIS mode is set in motion in two different ways to which two different programs correspond. The first is set in motion by interrupts when a failure has been detected by hardware checkers. The goal of this program is to localize precisely where the failure occured. The second program is set in motion periodically and its purpoae is to test the computer with the data configurations which reveal failures best. This program allows detection of the errors which cannot be detected by the hardwsre checkers (i.e. an erroneous data with correct encoding). These two programs update a atatus table which contains the status of computer components (failed or not, number of transient failures). They also decide to stop the computer when certain catastrophic failures occur or to set in motion the decision and reorganization mode.

c) In the OECISION ANO REORGANIZATION mode, a program analyzes the atatus word (in the status table) of the component in which one of the two test-diagnosis programs has detected a permanent failure and it decides either to reconfigure or to stop the computer.

3.2. FAULT TOLERANCE

3.2.1. FAULTS TOLERATEO: Any single fault is tolerated in memories, arithmetic and logic units (since they are mounted in a duplex acheme) or in logic units (quadded redundancy). Any error detected on the busses, switches the MECRA to interrupt programs, while all writing in memories, registers or counters is inhibited. Multiple errors csn also be tolerated in number of cases. Multiple errors can lesd to repair or to loss of service as said sbove (2.7.).

3.2.2. FAULTS NOT TOLERATED: Faults not tolerated include errors in the main control circuit, which leads to s design with an increased degree of microprogramming snd minimised control circuita. Also not tolerated are errors undetected at the memory output. Power supply failures have not been investigated in MECRA.

3.2.3. TECHNIQUES: One of the gosls of MECRA is an investigation of as many fault-tolerance techniques as possible, such as triple modular redundancy, qusdded redundancy, duplex redundancy at very low level (clock) and higher level (memories and srithmetic circuits), random redundancy (counters, registers), error detecting codes (Hamming d = 3) and parity bit, repetition, rollback, reconfiguration with removal without replacement, reconfiguration with replacement, diagnosia - stand-alone, preventive and emergency, local protections of process snd dsta. These techniques are used statically. It does not seem possible to describe these techniques in detail in thia paper, since it would require a description of the whole computer. Other techniquea were slso investigated but not used on MECRA, such as atopping the computer during noisy periods, snd control of correct microprogram linking.

3.3. NOVELTY: When the project started, two ideas unusual in the literature were employed in MECRA: address decoder redundancy in memories so ss to separate address errors and data errors, aingle-error-free hard-core.

3.4. INFLUENCES: A synthesis of efforts which came almost exclusively from the U.S.A. - universities, laboratories, and research institutes.

3.5. HARD-CORE: This is defined as a circuit, interconnecting several redundant functions, whstever its own redundancy level (it is a relative concept).

4. JUSTIFICATIONS

4.1. RELIABILITY EVALUATION: Reliability is not demonstrsted, it is computed, in two steps using a model. The first step concerns analysis and drawing a network model, the second step concerns random failure assignment into the model. After a great number of trials, the program furnishes results (e.g. curves, marginal probabilities...).

4.2. COMPLETENESS OF EVALUATION: Program evaluation is now being tested.

4.3. OVERHEAD: Approximately 60% to 70% of total system resources are devoted to fault tolerance (same percentage for logic, cost, and time).

4.6. CRITICALITIES: Use of decimal coded chsracters seems not well-fitted to fault-tolerant computers. Thia change could result in great savings in deaign. Other points are not critical.

4.7. IMPLICATIONS: The basic design assumes low-level integrated circuits, wtih a very small number of different circuits.

5. CONCLUSIONS

5.1. STATUS: The system is now operating and will be delivered in July 72, evalustion will follow during October snd November.

5.2. EXPERIENCE: Everything is possible, except, perhaps a sufficiently low cost, and reliable packaging snd wiring of components. Note that LSI would put problems to fault-tolerant computers because they need more pins to check redundant functions before connecting all together. This would probsbly lead to simultaneous use of LSI, MSI and small scale integrated circuits. Component manufacturers have not yet taken into account fault-tolerance constraints, but they will probably do so soon.

5.3. FUTURE: First prototype is projected 1976 - 1977, Current computer ia projected 1980, Use: Missiles, aircraft, real-time monoprocessors.

5.4. ADVANCES: Oifferent fault tolerant computers can be roughly compared in terms of reliability versus mission time; but this will fall back to evaluations of components and wiring MTBF. Such data, estimsted by constructors, do not seem to give a sufficient common bssis for evaluations. Theoretical and conventional data on component MTBF seem to be needed for accurate comparisons among different fault-tolerant computers.

SURVEY OF FAULT-TOLERANT COMPUTING SYSTEMS

L. J. Koczela, North American Rockwell Corp.
3370 Mireloma Avenue, Anaheim, California 92803, May 1972

1. IDENTIFICATION

1.1. NAME: A Three Failure Tolerant Computer Syatem

1.2. RESPONSIBILITY: Electronics Group, North American Rockwell Corp.

1.3. SUPPORT: Manned Spececraft Center, NASA

1.4. PARTICIPANTS: L. J. Koczela, J. Jurison, D. Brosius - North American Rockwell; P. Sollock - NASA.

1.5. START: 1/1/70

I.6. COMPLETION: 1/1/71 (design concept)

1.7. BIBLIOGRAPHY: A Three Failure Tolerant Computer System, IEEE Trens. on Computera, November 1971

2. MOTIVATION

2.1. PURPOSE: Real-Time Central Guidance and Control Computer

2.2. PHYSICAL ENVIRONMENT: Spaceborne

2.3. COMPUTING ENVIRONMENT: The computer system interacts with avionics subsystems via a multiplexed data bus.

2.4. COMPUTING OBJECTIVES: 30,000 words of memory; 500,000 operetions/second apeed

2.5. RELIABILITY OBJECTIVES: Must tolerate firat two failures with no dsgradation in performance and third failure with no degradation in aefety.

2.6. DYNAMIC VARIABILITY: Third failure could heve less computational capacity.

2.7. PENALTIES: Would require manual intervention with possible loss of life.

2.8. CONSTRAINTS: No physical constraints but a relative weighting of importance between physical perameters.

2.9. TRADEOFFS: Size, weight and power least important.

3. DESCRIPTION

3.1. ARCHITECTURE

3.1.1. CONFIGURATIONS

3.1.1.1. INTERCONNECTIVITY: Four redundant computers interconnected by four voter switches at their I/O channels.

3.1.1.2. RANGE: 2 - 6 CPUs, no restrictions on word length.

3.1.1.3. CAPABILITY: 500,000 operations/second

3.1.2. EXECUTIVE

3.1.2.1. MODES: The executive may operats the redundant computera in many modes of operation: non-redundent independent computers, multi-programmed, multi-computer, and various combinations of redundancy such as comparison, voting, etc.

3.1.2.2. SOFTWARE: Software control is equally diatributed among the redundant computers - no central control exiata.

3.2. FAULT TOLERANCE

3.2.1. FAULTS TOLERATED: Any 3 faults. A fault can range from a single circuit element to a complete module such aa e CPU failing. A failure haa no effect on syatem behavior. The system actually tolerate more than three faulta of many different types but it will tolerate at least any three faulta.

3.2.2. FAULTS NOT TOLERATED: Software feults that ere not caught in debugging.

3.2.3. TECHNIQUES: The technique used is replicetion of hardwara with quedrupla redundancy. Computations are performed redundently and reconfiguretion ia accomplished without removal or replacement after failure detection by voting.

3.3. NOVELTY: Through the redundant use of adaptive votera operating on the input/output of redundant computera, any three failure can be tolerated.

3.4. INFLUENCES: None

3.5. HARD-CORE: No hard core exists.

4. JUSTIFICATION

4.1. RELIABILITY EVALUATION: Extensive fault simulations have been succeaafully performed.

4.2. COMPLETENESS OF EVALUATION: It is impossible to verify a deaign goal of 100 percent confidence.

4.3. OVERHEAD: For triple failure tolerance, about 80%, Iesa for lower failure tolerance.

4.4. APPLICABILITY: To many critical real-time control syatems, industrial. space and defense applications.

4.5. EXTENDABILITY: The design can be extended to tolerate different numbers of failures, eg. any two failures, any four failures, etc.

4.6. CRITICALITIES: Requirement for 100% confidence in tolerating any 3 failures is very critical, lowering to 99 percent or so would reduce complexity snd cost.

4.7. IMPLICATIONS: Hardware designers must insure independence of failures at computer I/O interfaces.

5. CONCLUSIONS

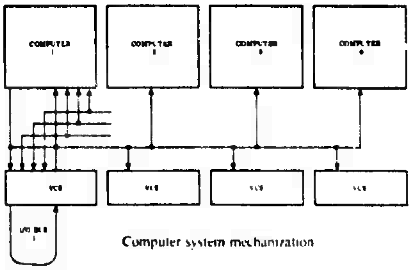
5.1. STATUS: System design concept completed, voter-switch detailed design completed, prototype hardware of voter-switch currently under development.

5.2. EXPERIENCE: A very rigid failure tolerance requirement can be met assuring that a minimum number of failures will be tolerated.

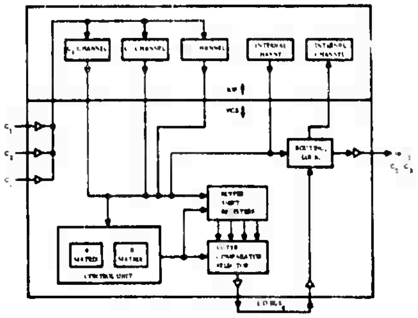
5.3. FUTURE; Possible use on space shuttle program

5.4. ADVANCES: A significant area that can enhance the state of the art in designing fault-tolerant computers ia analysis of failure modes of components and computer subsyatems in depth. Another very important area is error-free software.

6. COMMENTS: Much of the work on fault-tolerant computers is dedicated to single failures at the gate and circuit level. Unfortunately, in many cases this is not applicable to real world failurea when considering computers mechanized from state of the art LSI integrated circuits.



Computer system mechanization



VCS mechanization.

SURVEY OF FAULT-TOLERANT COMPUTINC SYSTEMS

J. S. Miller, Intermetrics Inc.
701 Concord Ave. Cambridge, Maes. 02138, May, 1972

1. IDENTIFICATION: The eyetem is referrad to only as the Intermetrics Multiproceeeor (occasionally abbreviated IMP). It is eponeored by the NASA Manned Spacecraft Centar, Houston, Texas. Personnel participating in the deeign, in addition to myself, are W. H. Vendever, A. L. Kosmala, S. F. Stanten, S. J. Schwartz, and A. Avakian. The project began in Juna, 1969, and has continued to dete, except for a thirteen-month interval betwean the original contract and the current one, which enda in about two months. Ona report has been publiehed: "Final Report--Multiproceseor Computer System Study", by James S. Miller, Daniel J. Lickly, Alex L. Kosmala, and Joseph A. Saponaro, in March, 1970. A second report ie presently in preparation. The work has been design-only; no hardware is involved.

2. MOTIVATION: The system ia oriented towards the general-purpoae computational requirements of a manned orbiting space station of ebout the 1980 time period. Its expected uses include reel-time statione control and data-acquieition functions, plus interactive and batch data processing operstions. The performance objectives are soft, but a real-time reeponse of 5 ms or better, and an equivalent of two million additiona per eecond for a three-procassor configuration seem adequate.

Because no hardware is being deeigned, no epecific reliability figure has been imposed. The use of the system for control of the apace atation iteelf pleces heavy emphasis upon continued operation, at reduced performance in the presence of faults. Although we expect that temporary outeges of the computer system will be tolerable, our efforte heve baen directed at avoidance of single-point failure modes.

3. DESCRIPTION: Although an earlier design favored the use of a single internal bus connecting all modulee, with caches et each proceesor incorporated to diminieh traffic, we have now settled upon a more conventional croeebar-like network. Each processor and main memory module possesees a bus, as does the I/O controller. Secondary storage is accessed via the latter unit. Configurations of up to eight processors, and ae few es one, are planned, with nominally three. A 32-bit word, with additional bits added for error-detection, has been chosen.

A full complement of multiprocessing is provided by the system software. Processea may be dependent or independent of each other. Each processor is multiprogrammed by a "floating" operating system executed by any processor, as neceeeary. Interprocess communication is eupported, and processes may field their own iterrupts if they choose.

The instruction set of the processor is designed to support the execution of high-order block-structured languages, such as Algol, PL/I, and HAL, the last being an Intermetrics-designed language aleo eponsored by NASA/MSC. The instruction set eomewhat reeemblee that of the Burroughs B6700, although eubetantial differences have been introduced. It is planned that only high-order source languages will be supported. The system is designed to tolerate "all" faults in the hardware, provided that second, independent, faults do not occur before recovery action is complete. Our experience with the error-prone discipline of software restarts on the Apollo program hes steered the deaign. Compreheneive error-detection facilitiee ere provided in the herdware, so that detection ie immediate and highly probable. Furthermore, eufficient redundant information ie maintained in independent locations so that the operating system end hardware capabilitiea are adequate to continue execution of all processes (subject to reduced performance limitations) without explicit perticipation by application software.

Processors are dually-redundant to provide complete error-checking capability. Instruction execution is devised so that inputs are always preserved until error-free completion is signalled; thus re-try is always possible. Processor state information is maintained in local memory which is externally accessible, so that execution of an instruction interrupted by a processor fault may be resumed by another processor if re-try proves unsuccessful.

Main memory is dynamically time-multiplexed, using verieble-size segments, similar to the B6700. The read-write functions of the memories ara implemented in a way that enables duplicate storage of information when this is specified via the high-order-language. Procedure code and other read-only data will be resident in secondary storage, and thus need not be duplicated in main store. Variable data may or may not be stored doubly; if not, the owning process will be marked for termination if such data is destroyed through memory fault. The duplicate storage of specified data, although supported by hardware, is relatively transparent to the memory management software; no epecially-designated memory modules are used.

4. JUSTIFICATION: The design is relatively complete, but has not been evaluated by any formal procedure. Somewhat deliberately, it wss bssed on working architectures to reduce the number of possibilities for unanticipated difficulty.

As mentioned above, processors are duplexed, and some information contsinad in main memory is stored redundantly. An additional "overhead" for fault-tolerance is triple-redundancy in certain access-elements of main memory, and the I/O controller.

In our judgement, the system we heve designed is suitable for implementation in any application where transparency to faults and continued operation are worth the cost of the redundancy. It is emphasized that comprehensive error-detection, apart from recovery, is responsible for much of the additional cost.

No claim is made that the system ie tolerant of software flaws. However, the insietence on use of high-order language ia expected to reduce the probability of such errors, both by making their commission less likely, and by implementing error-detection features in the langusges and compilers. Run-time checking will be provided via hardware features (where possible) and by compiler-inserted code to at least signal the occurrence of software misbehavior, before the effecta can propagate.

5. CONCLUSIONS: Intermetrics believes that its fault-tolerant HOL-machine would be cost-competitive with a "conventional" system manufactured in the same quantity, since the memory saved by usage of HOL instructions is expected to be significant. Further development will be pursued.

SURVEY OF FAULT TOLERANT COMPUTER SYSTEMS

Donald C. Wallace
Stanford Research Institute, Menlo Park Ca, June 72

1. IDENTIFICATION
1.1 NAME:COMEX- Online order handling system

1.2 RESPONSIBILTY: P.C.Service Corp. (subsidiary Pacific Coast Stock Exchange)

1.3 SUPPORT: Member firms of PCSE

1.4 PARTICIPANTS: Member firms of PCSE

1.5 START: Contract let 17 November 1967

1.6 COMPLETION: System accepted - 4 December 1969

1.7 BIBLIOGRAPHY: The most accurate desription of the COMEX is the final documentation delivered with the system.
Documents:
Specification for data processing and communication equipment for Pacific Coast Stock Exchange PC Service Corp., 1967
Proposal for Real-Time Order Handling System BBN #p68-DE-01,4 August 1967
Contract for Real-Time Order Handling System for Pacific Coast Stock Exchange BBN/PCSE,17 November 1967

2. MOTIVATION
2.1 PURPOSE: Real time odd-lot order execution

2.2 PHYSICAL ENVIRONMENT: Ground based

2.3 COMPUTING ENVIRONMENT: The system serves two trading floors, one in Los Angeles, the other in San Francisco.

2.4 COMPUTING OBJECTIVES: COMEX is designed to handle virtually all low-speed teletype speeds, levels and codes. It appears as a node on each of the connected broker firms communication networks and must conform to the line protocols and hardware constraints of that network. The design objectives were for 64 "nodes" in LA. and 64 in SF., and for a maximum message-switching traffic of 25,000 orders/transactions per day.

2.5 RELIABILITY OBJECTIVES: The system was designed to provide 99%+ uptime and with a no "message lost" criteria.

2.6 DYNAMIC VARIABILITY: The system is designed so that order entry is performed in real time, but the order execution process may lag an arbitrary period of time. In operation this lag never exceeds 20 minutes (approx.??).

2.7 PENALTIES: COMEX has various degrees of degradation, the ultimate being total manual operation and execution of the orders by the specialists on the trading floors. Esoteric software/hardware malfunctions could cause extremly large manual intervention problems as the system is really buying and selling stock on the behalf of members of the exchange.

2.8 CONSTRAINTS: The PCSE is really two exchanges with two different trading floors, one in Los Angeles and one in San Francisco. For reliability reasons the system is fully redundant. A PCSE constraint on the system was that the system be equally split between the two sites.

3. DESCRIPTION
3.1 ARCHITECTURE
3.1.1 CONFIGURATION
3.1.1.1 INTERCONNECTIVITY: See diagram which shows the twin IBM 360 computers and the 680 systems each of which includes a DEC PDP8 computer.

3.1.1.2 RANGE: The system is really two systems running in parallel. It is sensible to run them as single units or a fully redundant system. Two configurations are possible:-
Non-partitioned trading floors:
LA-remote680, SF-local680 and SF-360
SF-remote680, LA-local680 and LA-360
Partitioned trading floors:
SF-local680 and SF-360
LA-local680 and LA-360

3.1.1.3 CAPABILITY: COMEX consists of two (2) 360/50 computers plus the front-end communications systems.

3.1.2 EXECUTIVE and operating system: COMEX runs under IBM/360 DOS with its fixed number of multiprogram partitions option.

3.1.2.1 MODES of operation: The order execution process runs in a high priority partition of DOS while normal operation of PC Service Corp. computer operations are being run in other "foreground" and the background partitions. The communication process (in the 680's) is dedicated and allows no other functions.

3.1.2.2 SOFTWARE organization: Basically the 680's do character assembly (bits), line protocol interpretation (answer back, echo, etc...), message segment assembly, I/O buffering, transmission to local and remote 360's. The 360's do message switching, code translation, message decoding (syntax analysis), order queuing, decoding of NYSE and AMEX tickers (identify trades), execute queued orders, send confirmations to broker and specialist.

3.2 FAULT TOLERANCE
3.2.1 FAULTS TOLERATED: Essentially the system will tolerate any or all failures in a single system (i.e., backup or primary).

3.2.2 FAULTS NOT TOLERATED: Any simultaneous failures in both the primary and backup system causes loss of integrity of the data files. This is considered a catastrophic event and some manual correction and intervention for order execution and notification will be needed. (To my knowledge this has only occured once in the almost three years of operation.)

3.2.3 TECHNIQUES:
HARDWARE: The COMEX system is completely redundant (two of everything), and both systems run in parallel The major design criteria was that nothing should happen in one system half that could adversly effect the other. This led to the system interconnections (PCU) being unidirectional and step-locked in a "here's a word, take a word fashion. All TTY connections to the system are dual dropped and there is a hardware interlock to prevent both 680 machines from outputing to a line at the same time.
SOFTWARE: The software is designed to be very modular, and no control flow exists between functional routines. Control flow is between the COMEX scheduler/ executive and each functional module. Data is passed from function to function by means of stacks and lists, and standard system global routines are used to accomplish this. Both systems are actually performing the entire order execution task in parallel and there is really no communication between them. The only difference is that the "backup" system is not outputing transaction confirmations and order receipt notifications. The backup system maintains a queue of the last "n" messages to each line in the system. When switch-over occurs, these messages are output to the specialists/brokers with a "may be duplicate" tag.

3.3 NOVELTY: The interconnection of the DEC 680's and the S/360's is accomplished without requiring modifications or additions to the IBM operating system or providing "special" I/O modules. The 680's (two of them) have a S/360 channel equivalent (FCU) that talks to the IBM 2841 disk controler with the two channel feature (8100). This is the equivalent of having two 360 systems talking to one disk system. This is a standard IBM configuration possibility (though not supported by IBM software). If the user is willing to accept implementing his own read/write lock mechanisms there is nothing in the IBM system to preclude this mode of operation. Given all of the above it is now possible to write a communications system strictly at the user level using standard IBM I/O software. Data just "appears" on the disk and is read into the 360 and is in turn written on the disk and just "disappears". The data from the 680's is written as a sequentially ever growing file, capturing an entire day's transactions. This allows "rerunning" a day's transactions in real time to find obscure bugs.

3.4 INFLUENCES: After spending several years working on modified or bastard 360 Systems and realizing the effort level to maintain these systems given the frequency of new IBM releases, it seemed insane to design a system that relied on any thing except the most rudimentary features of the IBM monitor. The approach described has proven very successful in over three years of operation. To my knowledge no problems have been encountered due to the monitor/ Comex system interface.

4. JUSTIFICATION

4.1 RELIABILITY EVALUATION: The oystem has met and exceeded the deaign criteria over the laat 2 yeers of operation.

4.3 OVERHEAD: Since the aystem is totally redundant, at least half the :oet of the communications front end is due to reliability requirementa. The reliabilitv requirements of the ayetem probably did not contribute aignificantly to the software deaign, and probably helped in the checkout end operational phases.

4.4 APPLICABILITY: The syetem hes general epplicebility for communications and message switching aystems where the bese computer facility must be IBM (for what ever reasone). It offers significant cost savings when compered to an equivalent all-IBM equipment configuration. Its novel interfacing technique allows the usere to concentrate on the appllcetion program and offers long-term savings in effort by not having e modified IBM operating system. The aystem has apecific epplicability to other amall or moderete sized atock exchangea both U.S. and foreign.

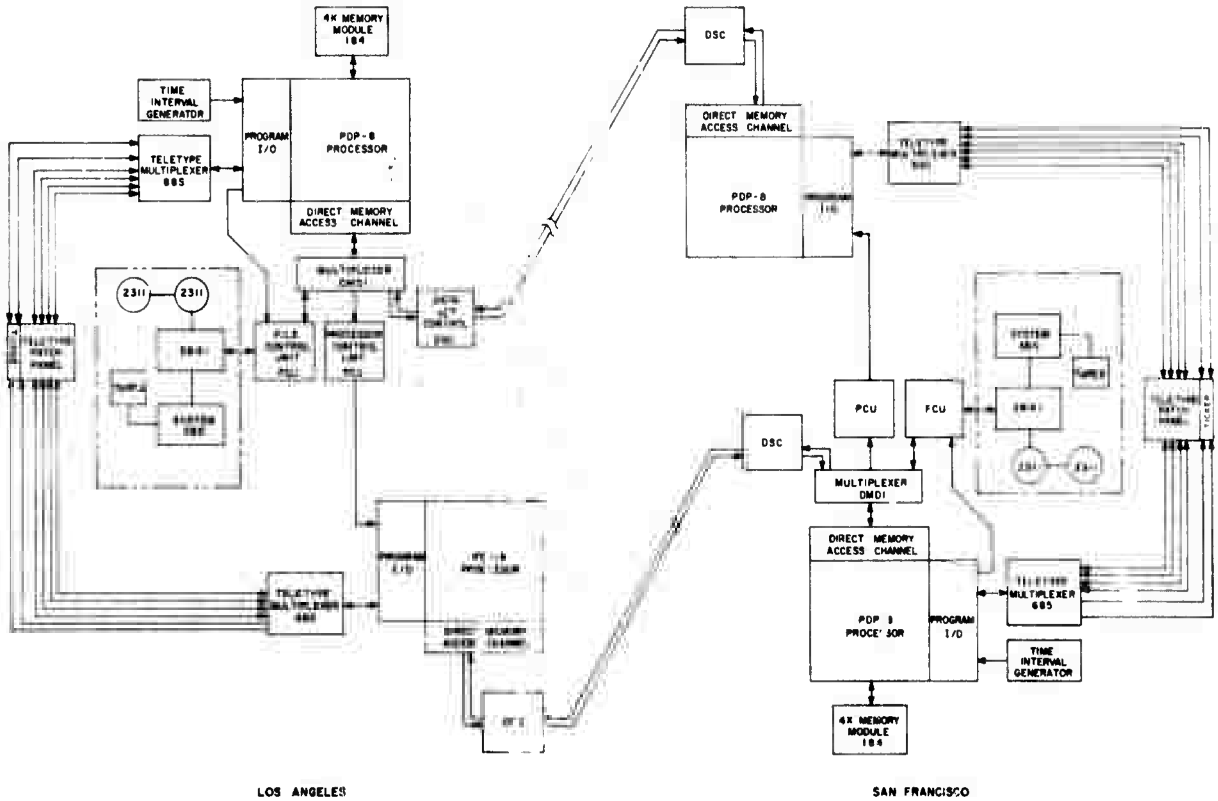
4.5 EXTENDABLILITY: There appear to be no obvious extentions to the systen as far as capacity is concerned. Two- or three-fold increases in throughput are possible whereas factors of ten are out of the question. Since the next obvious exchange automation task is either NYSE or AMEX and the volume of message traffic for those exchange is ataggering. COMEX most certainly has no real logical extension for these situations. Specific experience and techniques in dealing with automation of a stock exchange process may have general applicability.

4.6 CRITICALITIES: A specific goal in hardware design not to exceed the "atate of the art" was imposed by PCSE to gain assurance of reliability. This constraint caused the selection of hardware that most assuredly is obaolete by today's standards (e.g., bit serial TTY interface), greatly reatricting overall I/O capacity (like maybe a factor of 10).

5. CONCLUSIONS

5.1 STATUS. The system is currently handling 15% of the maximum message switching capacity of 25,000 order transactions per day. It is undergoing significant modification to handle round-lot traffic, which potentially will increase load to 50% of capacity within the next 18 months. Studies are underway to evaluate high-speed I/O capability.

5.2 EXPERIENCE: Overall system operation has been highly satisfactory to the PCSE.



COMEX SYSTEM — PACIFIC COAST STOCK EXCHANGE

Werner Ulrich, Bell Labs
Naperville, Illinois 60540 May 1972

Essentially, this questionnaire represents the entire body of published material on maintenance aspects of ESS. I have therefore taken the liberty of sending a bibliography in place of a completed questionnaire. You will notice that most of the articles are quite brief with the exception of items 1 and 2 which are complete descriptions of the No. 1 and No. 2 ESS maintenance plan, and item 3 which is a longer article on a specialized detail of our trouble location manual or dictionary approach.

The bibliography, in addition to articles on No. 1 and No. 2 ESS, contains items on our data switching system (never commercially offered, item 5), the traffic service position system (item 9), and a military application of No. 1 ESS (item 14).

BIBLIOGRAPHY

1. Downing, R. W., et al., "No 1 ESS Maintenance Plan," Bell System Technical Journal, Vol. 43, pp. 1961-2020, September, 1964.

2. Beuscher, H. J., et al., "Administration and Maintenance Plan of No. 2 ESS," Bell System Technical Journal, Vol. 48, pp. 2765-2815, October, 1969.

3. Chang, H. Y. and Thomas W., "Methods of Interpreting Diagnostic Data for Locating Faults in Digital Machines," Bell System Technical Journal, Vol. 46, pp. 289-318, February, 1967.

4. Tsiang, S. H., Haugk, G. and Seckler, H. N., "Maintenance of a Large Electronic Switching System," IEEE Transactions on Communications Technology, pp. 1-9, February, 1969.

5. Aitcheson, E. J. and Cook, R. F., "No. 1 ESS ADF Maintenance Plan," Bell System Technical Journal, Vol. 49, No. 10, pp. 2831-2856, December, 1970.

6. Nowak, J. S. and Tuomenoksa, L. S., "Memory Mutilation in Stored Program Controlled Telephone System," 1970 IEEE International Conference of Communications, pp. 43-32-43-45.

7. Chang, H. Y. and Scanlon, J. M., "Design Principles for Processor Maintainability in Real-Time Systems," Proceedings of Fall Joint Computer Conferences, pp. 319-328, 1969.

8. Nowak, J. S., "Emergency Action for No. 1 ESS," Bell Laboratories Record, Vol. 49, No. 6, pp. 176-179, June/July, 1971.

9. Connet, J. R., Pasternak, E. J. and Wagner, B. D., "Software Defenses in Real-Time Control Systems," Second Annual International Symposium on Fault Tolerant Computing, June 19-21, 1972, Boston, Massachusetts.

10. Almquist, R. T., et al, "Software Protection in No. 1 ESS," 1972 IEEE Conference on Communications, June, 1972.

11. Ketchledge, R. W., "Service Experience with No. 1 ESS Equipment," International Conference on Electronic Switching, 1966 Proceedings, Paris, Edition Chiron, pp. 712-716.

12. Vaughan, N. E., "Experience with the No. 1 ESS," International Conference on Electronic Switching, 1966 Proceedings, Paris, Edition Chiron, pp. 704-711.

13. Haugk, G., "Early No. 1 ESS Field Experiences, Part 1, 2-Wire System for Commercial Implications," IEEE Transactions on Communications Technology, Vol. 15, pp. 744-750, December, 1967.

14. Seckler, N. N., "Early No. 1 ESS Field Experience, Part 2, 4-Wire System for Government and Military Implications," IEEE Transactions on Communications Technology, Vol. 15, pp. 751-754, December, 1967.

15. Johannesen, J. D., "No. 1 ESS Service Experience - Software," IEEE Conference on Switching Techniques for Telecommunication Networks, Conference Publication No. 52, pp. 459-462, April, 1969.

16. Staehler, R. E., "No. 1 ESS Service Experience - Hardware," IEEE Conference on Switching Techniques for Telecommunication Networks, Conference Publication No. 52, pp. 463-466, April, 1969.

Capt. L. A. Fry, Space and Missile Systems Organization (SAMSO), Los Angeles AFS, California, June 1972.

1. IDENTIFICATION
1.1 NAME: Modular Spacecraft Computer

1.2 RESPONSIBILITY: SAMSO/SYT, Los Angeles APS, Ca.

1.3 SUPPORT: Not available

1.4 PARTICIPANTS: Raytheon Company, Sudbury, MA., Ultrasystems, Inc., Newport Beach, Calif.

1.5 START: Project started mid-1971

1.6 COMPLETION: Architecture study completed January 1972. Other efforts continuing.

2 MOTIVATION
2.1 PURPOSE: Support of all satellite data processing requirements

2.2 PHYSICAL ENVIRONMENT: In satellite

2.3 COMPUTING ENVIRONMENT: Hardwired to environment

2.4 COMPUTING OBJECTIVES: Approximately 200K operations per second. Memory expendable to basic 32 bit word format with 64K words of memory.

2.5 RELIABILITY OBJECTIVES: High reliability for 5 year life

2.6 DYNAMIC VARIABILITY: Essentially no variability

2.7 PENALTIES: Loss of major satellite functions

2.8 CONSTRAINTS: 25 pounds and 30 watts.

3. DESCRIPTION
3.1 ARCHITECTURE
3.1.1 CONFIGURATIONS: Not available

3.1.2 EXECUTIVE
3.1.2.1 MODES: Interruptible but not a true multiprocessor

3.1.2.2 SOFTWARE: Not yet developed

3.2 FAULT TOLERANCE
3.2.1 FAULTS TOLERATED: Transient and permanent--all logic types. Also can tolerate catastrophic faults.

3.2.2 FAULTS NOT TOLERATED: Faults resulting from major physical damage.

3.2.3 TECHNIQUES: Replication; coding; repetition and rollback; and reconfiguration. Techniques used statically and dynamically.

3.3 NOVELTY: Extensive dynamic redundancy

3.4 INFLUENCES: Not available

3.5 HARD-CORE: Configuration Control Unit is triply-modular-redundant, controlling all retries and most reconfigurations.

4. JUSTIFICATION: Not available

5. CONCLUSIONS
5.1 STATUS: Performing interpretive simulation

5.2 EXPERIENCE: Architecture very suitable for intended application

5.3 FUTURE: Not available
5.4 ADVANCES: Not available

6. COMMENTS: Fault-tolerant computers can make a major contribution to long duration space missions.

APPENDIX III
A HIERARCHICAL FRAMEWORK FOR FAULT-TOLERANT COMPUT'NG SYSTEMS

Peter G. Neumann
SRI, Menlo Park, California 94025

IEEE Computer Society Conference
San Francisco, September 12-14, 1972

This work was supported in part by the Defense Advanced Research Projects Agency of the Department of Defense (monitored by ONR) under Contract N00014-72-C-0254 and in part by NASA Langley Research Center under Contract NAS-1-10920. No official views are implied.

ABSTRACT

A hierarchical design framework for fault-tolerant computing systems is considered here. The intrinsic flexibility and dynamic reconfigurability which result greatly enhance the effectiveness of system operation and system development.

INTRODUCTION

Recent efforts in developing large computing systems have led to the conclusion that carefully conceived internal system structure is beneficial to the whole development process. It is the basic contention of this paper that such structure is especially helpful in the development and operation of fault-tolerant computing systems, in which there is continuous correct performance despite internal malfunctions.

A framework is considered here which permits a wide range of techniques for fault tolerance to be applied at each of various levels in a hierarchy, when and where most effective. This hierarchical framework permits fault tolerance to be achieved at low cost, especially in systems with some real-time leeway. Various implications are examined. The framework is applicable primarily to designs for new systems. It is also suitable for the software of some existing systems. Finally, a problem is considered which is greatly simplified by employing the hierarchical framework. This is the massive-transient recovery problem, in which arbitrarily many unknown faults may have occurred.

FAULT-TOLERANCE TECHNIQUES

A variety of techniques exists for increasing system fault-tolerance /1-2/. Two basic types of techniques are usually found in execution, static and dynamic. STATIC techniques involve preplanned actions with no changes in the operating environment or in the flow of control ("fault-masking" via coding, replication with voting, etc). DYNAMIC techniques involve detection and diagnosis of faults, followed by a non-trivial corrective action. Examples include repetition (e.g., rolling back the entire system to an earlier valid state) and reconfiguration with or without replacement by spares (e.g., removing or working around faulty units, and either substituting spares or accepting a degradation in capacity). In certain cases, human intervention is useful. Pre-execution techniques are also useful, e.g. proofs of correctness of programs and design. Note that static techniques may be found in dynamic usage, e.g., replication used only for particular processes at certain times (see below). Similarly, conceptually dynamic techniques may appear in static usage (e.g., instruction retry in which all eventual results are buffered making rollback trivial).

LEVELS OF STRUCTURE AND DYNAMIC RECONFIGURATION

Many design approaches assume that essentially all single faults are equally critical. In reality, certain faults may be far more critical than others. Thus a system architecture is desired in which fault tolerance techniques may vary in time and space, depending on current usage and on the criticality of the errors which might otherwise result. As used here, "dynamic reconfiguration" implies alteration during execution of the fault-tolerance techniques, or of the rest of the hardware and software, or both. The framework presented here facilitates control of such reconfiguration. It reduces system overhead (hard and soft) due to fault tolerance, and increases the overall system effectiveness.

Numerous levels at which these techniques for fault tolerance may be applied are readily identifiable. These levels range from components to modules to processors; from bits of memory to words to blocks to memory modules to hierarchies of diverse types of memories (e.g., as in a virtual memory, in which all memories in the hierarchy appear to the external interfaces as a single level); from hardware to microprogram through various levels of operating system software to command software to user programs; from system elements to systems to networks of systems.

A very significant system structure is given by Dijkstra /3/. Details internal to implementation at a given level are made invisible to all higher levels by the interface language at the given level. Capability at that level is dependent on the capability of the next lower level, and is precisely that provided by the interface language. The use of these distinct "levels of invisibility", or "levels of abstraction", is highly beneficial to system development. Two familiar examples are the invisibilility of a cache memory to a program and the invisibility of multiprogramming to a user.

THE HIERARCHICAL FRAMEWORK

The desired hierarchical framework for fault- tolerant computing systems is as follows.

(1) Various levels of structure are established explicitly in the design as levels of invisibility. Control and communication facilities must be provided. (Useful mechanisms are known for this purpose, e.g., for coordinating among processes -- both to avoid conflicts and to permit sharing of programs, data and control -- and for communicating

among or within levels. Except for deadlock avoidance, these mechanisms are fairly clear-cut.)

(2) Associated with these levels are possible configurations of fault-tolerance techniques and possible modes of dynamic reconfiguration.

(3) Analysis, simulation, and operating experience should be used to study the relative effectiveness of these techniques under varying demands and of reliable algorithms for deciding how and when to switch among configurations. The suitability of the choice of levels should also be evaluated.

An illustration of this framework is provided by Table I. The first column of the table identifies some typical levels of invisibility. (Lower levels are toward the top of the table.) The second column gives examples of concepts invisible at each level. The third column showa techniques which can enhance fault tolerance at each level, and whose details should be invisible at higher levels. Those techniques in the table which lend themselves to dynamic reconfigurability are indicated by an asterisk. The dynamic control over reconfiguration of such techniques may be done internally, or via the interface language for the appropriate level. Techniques at one level may be applied relatively independently of those at other levels, if desired.

As an example, consider a system normally configured as five independent multiprocessed processors. At the VIRTUAL SYSTEM level, each user (or application environment) deals with a command language interface to the system. At the VIRTUAL PROCESS level, each virtual process may in turn employ one or more processes, either to exploit intrinsic parallelism (e.g., simultaneously processing, moving a file, and printing) or to provide redundant computations. At the VIRTUAL PROCESSOR level, these processes may be executed on the same or on different processors. The configuration might on occasion include two processors in a comparison mode with two identical processes (or with two different algorithms), or three processors in a voting mode, or even in rare cases five in a

INVISIBILITY LEVEL (examples)	INVISIBLE CONCEPTS (examples)	APPLICABLE FAULT-TOLERANCE TECHNIQUES (examples)
Components, chips	Technology details, fabrication methods	Intrinsically reliable technologies, good engineering, quality control, coding and fault-masking, replication
Modules	Board layouts, pin connections, timing	Conservative design, reliable connectors, environmental control; *Diagnosis, component replication, replacement
Functional units: Processors, central, etc.	Processor algorithms Address calculation Bus control Interrupts	*Automatic instruction retry, arithmetic coding Bounds checking, memory protection *Alternate routes, coding, degradable priority mechanisms *Race-free fail-operational interrupt design
Memories	Cache mechanisms Internal representation Internal configurations Device characteristics	*Automatic reloading *Coding on memory contents *Reconfiguration around bad memory (via paging, de-interlace) Use of read-only memories to avoid overwrite and aid recovery
Input-output	Media properties, device dependence	*Coding on contents of media and transmission *Verification, checking, reread and compare after write
Virtual hardware	Configurations	*Configuration sensing and self-reconfiguration, powering on-off (e.g., spares), distributing and replacing power supplies
Virtual processor	Multiprocessing--processor multiplexing for distinct processes Array computing Processor dispatching	Coding, handshaking on interprocessor communication, avoidance of interprocessor interference; *Replication of physical processors as a single (virtual) processor, voting as needed *Reconfiguration and replacement within the array *Configuration insensitivity via checked table-driving
Virtual process	Multiprogramming--process multiplexing on a virtual processor Process scheduling Virtual interrupts, process isolation	*Replication of virtual processors for a single process *Independent computational checks (via possibly distinct processes) as single virtual process; *Automatic rollback *Explicit measures of permitted degradation per process Safeguards on interprocess communication (vs. lost interrupts, blocked polling), avoidance of interprocess interference, intraprocess protection (rings, capabilities, master modes)
Virtual memory	Multiplexing of memory hierarchy: locations, relocations, backup and retrieval, directories	*Replication of critical data in various places in hierarchy, including reliable cheap backup store; *Automatic rollback Redundant pointers in directory structure and file maps to permit fast recovery; Access control on files (e.g., write protection) and the use of pure procedure to inhibit loss of critical data or programs and to aid in automatic rollback
Virtual input-output	I-O multiplexing, virtual devices Exception handling Asynchrony, buffering	Handshaking to avoid loss of information; *Status information *Device switchability, media replication *Coding (e.g., redundant headers); *Flexible error handling Race-condition and deadlock avoidance
Virtual system	User multiplexing, sharing of data System correctness	Isolation of users from the system and each other; *Controlled sharing (if any); Self-identifying descriptors *Validation, evaluation of effectiveness and correctness *On-line maintenance; Good compilers, diagnostics, debuggers
Virtual network	System multiplexing	*Coding on intersystem communication, alternate paths *Detailed status of network control and network requests

TABLE 1. Examples of techniques for fault-tolerance applicable to various levels of invisibility. Asterisks denote techniques particularly amenable to dynamic reconfigurability.

voting mode. In these modes there may be 4, 3, and 1 distinct virtual process(es), respectively, instead of 5 as in the fully multiprocessing mode. (Several of these modes are also useful if some processors are not operational, in which case replacement is also desirable.) The internal mechanics of such mechanisms should be mostly invisible to each virtual process.

At the VIRTUAL MEMORY level, device addresses are invisible. When being actively used, a virtual memory page may in fact be found in various states of recency and/or in various modes of replication on various devices in the memory hierarchy, even in the absence of fault-tolerance techniques. For example, in a paged environment, various instances of a given page may exist simultaneously in a cache-type memory, in primary memory and in secondary memory. If it is procedure that is "pure" (unchanged by execution), then the contents of all instances are identical (barring errors); if it is data, the instances may differ. In the present framework this natural temporary proliferation can be used constructively to provide checkpoints, thus greatly facilitating automatic rollback. This is especially useful with various instances of critical data.

At the MEMORY level, coding techniques offer very inexpensive fault-tolerance. Only 8 redundant bits are needed to provide single-error correction for 64-bit words in memory and arithmetic, a 13% increase in memory cost. (The cost of error correcting circuitry is small by comparison.) Coding techniques also lend themselves to dynamic reconfiguration. One such approach involves different uses of a particular encoding. Consider for example a code with Hamming (or arithmetic) distance 4 for single-error correction and double-error detection. When the multiple error rate is high, the code may better be used for triple-error detection (accompanied by increasingly loud cries for help). (Another example is using a byte-error correcting code as a multiple-error detecting code.) A second approach involves varying the encoding itself, e.g., changing the redundancy.

Similarly at the MODULE level, multiple arithmetic or functional units tied to a control unit may be used in replication for fault tolerance, in synchronsm as in the ILLIAC IV for handling parallelism in computation, or independently. The first of these applications substantially increases reliability, while the others may substantially increase the computational throughput.

Explicit levels of structure are now evident in a few recent operating systems. For example, the Multics protection hierarchy (see /4/) provides successive levels of resilience to errors in its levels of protectability. A spectrum of criticality exists with respect to faults. Only malfunctions (hard or soft) involving the lowest software level affect the viability of the system. Others have diminishingly serious affects on the correctness of operation as the level increases, e.g. aborting a user's process or one command. As with hardware, software techniques for fault tolerance may differ from level to level.

IMPLICATIONS OF THE HIERARCHICAL FRAMEWORK

There are numerous advantages of this hierarchical framework. These include considerations of reliability, computational capacity, and cost.

(1) A wide variety of techniques can be applied, each where it is most effective, responsive to the needs for fault tolerance and computing capacity, and subject to the cost factors. Each configuration can be dynamically altered, based on the current usage of the system. (This may affect more than one level at once.) The net cost of system fault tolerance can therefore be reduced, especially if rarely used techniques can be performed reliably in software. Considerable savings also result if occasional modest real-time delays are permitted (e.g., for diagnosis, recovery and reconfiguration), further reducing the need for dedicated hardware. Nonuniform costs also permit the reduction of the incremental cost of fault tolerance. If memory costs (including secondary storage) dominate total hardware costs, then the relatively small cost of redundancy in memory (e.g., logarithmic for single-error correction in memory and arithmetic) may dominate the incremental cost, even with replicated processors. If memory costs do not dominate, then

memory is relatively cheap and logic-in-memory architectures (see below) may be of interest. The framework also facilitates checkpoint mechanisms which permit varying degrees of rollback as needed, involving different levels of the hierarchy. On-site maintenance is also aided, as are on-line interactive diagnostics.

(2) In general, computing capacity not currently dedicated to fault tolerance is available for useful computing, assuming reasonable system balance. It is desirable to have pools of modules, of functional units, of processors, and of systems to configure among. The multiplicity of each pool should be large enough so that the mesh of graceful degradation is reasonably smooth and that the loss of any unit is not serious. This increases the overall system effectiveness, in terms of both computing capacity and fault tolerance.

(3) The intrinsic structure of the hierarchy enhances each stage of system development, including the stages of designing, implementing, documenting, debugging, certifying, analyzing, maintaining, and modifying a system. At each such stage the notion of levels of invisibility permits issues of fault tolerance relevant to lower levels to be abstracted and analyzed, aiding in isolating any side-effects. Thus the framework serves as a useful model as well.

(4) Recent technological advantages (e.g., LSI) significantly improve the cost-effectiveness of many of the techniques. These advances should also stimulate new architectural directions, such as multiprocessors with considerable multiplicity, and distributed-logic or logic-in-memory designs. The latter case involves large arrays of small memory elements, each containing processing capability. These arrays may be organized into subarrays of subarrays, possibly with structures geometrically oriented toward the problem to be solved.

There are of course many questions left unanswered.

(1) Questions of overhead and reliability resulting from the control of such systems must be examined carefully. It appears that the overhead

can usually be kept small, except when fault-tolerance limits are approached. It is obviously desirable that the mechanisms for controlling reconfiguration must themselves be fault tolerant, thrash resistant, and reconfigurable. Interference problems and intercommunication must also be handled reliably.

(2) This framework seems particularly effective for large general-purpose systems. How effective it can be under various circumstances, e.g., for small systems, for those with tight real-time constraints, requires further study.

(3) How can the various tradeoffs among fault tolerance, computing capacity, cost, overhead, etc., be characterized? Under what circumstances is it desirable to reconfigure? What kind of limiting behavior occurs as computing capacity or fault-tolerance capacity is reached? What are the penalties associated with having too many or too few levels? What happens to the notion of the "weakest link"? Can it be distributed among less weak links? How does it shift during reconfiguration?

THE MASSIVE TRANSIENT RECOVERY PROBLEM

As an example of a specific problem which can be greatly simplified by the adoption of the hierarchical framework, consider the "massive-transient" recovery problem:

> A correlated fault source (e.g.,a power surge or a bolt of lightning) has left all units of the system suspect, perhaps introducing both transient and permanent faults. The problem is for the system to diagnose and configure itself back into a working configuration and to validate itself for correctness, all under its own control.

This problem is essentially a generalized fault- tolerance problem, where performance may cease temporarily during and just after the massive transient. It is also closely related to normal system initialization. The hierarchical framework and the dynamic reconfigurability both aid greatly in solving this problem. One solution involves reestablishing minimally correct hardware by bootstrapping upwards from the lowest levels of the hierarchy, until a satisfactory rudimentary system is obtained. It is also desirable to validate downwards from the higher levels. This solution is aided by the use of a hard-wired non-volatile read-only memory which provides a basis of correct programs for recovery. Further help is offered if this memory is directly executable and the programs are pure, and if these programs operate only out of local memory at first. By working up the hierarchy, valid portions of the system begin to emerge. (Another solution might involve trying experiments on various configurations of the whole system.) Note that this problem may be intrinsically insoluble for a given system. It may also be insoluble for the particular massive transient, e.g., because not enough operational equipment remains to self-diagnose and configure a valid system, or even just to operate such a system. (More equipment might be required for diagnosis than for operation.)

CONCLUSIONS

The hierarchical framework presented here appears to have great potential in the design of fault-tolerant systems. It can increase the effectiveness of new systems as well as the ease and flexibility of their development and operation. It should increase in utility as technological advances permit much larger systems to be developed. Further study is intended.

ACKNOWLEDGMENT

The author is indebted to Jack Goldberg, Karl Levitt and John Wensley for many helpful comments.

REFERENCES

1. E.g., see IEEE Trans. on Computers, C-20, November 1971, and the Digest of the IEEE 1972 International Symposium on Fault-Tolerant Computing, Newton, Mass., June 19-21, 1972.

2. W. C. Carter, et al., Design Techniques for Modular Architecture for Reliable Computer Systems, IBM Report 70-208-0002 under Contract NAS8-24883, Yorktown Hts. NY, March 26, 1970.

3. E. W. Dijkstra, The structure of the "THE" multi-programming system, CACM 11, pp. 341-346, May 1968.

4. M. D. Schroeder and J. H. Saltzer, A hardware architecture for implementing protection rings, CACM 15, pp. 157-170, March 1972.