A DISTRIBUTED FAULT TOLERANT ARCHITECTURE

FOR NUCLEAR REACTOR CONTROL AND SAFETY FUNCTIONS

Myron Hecht, Jeffrey Agron, and Sara Hochhauser
SoHaR Incorporated
Los Angeles, California

Abstract

A new fault tolerant architecture that provides tolerance to a broad scope of hardware, software, and communications faults is being developed. This architecture relies on widely commercially available operating systems, local area networks, and software standards. Thus, development time is significantly shortened, and modularity allows for continuous and inexpensive system enhancement throughout the expected 20-year life. The fault containment and parallel processing capabilities of computers network are being exploited to provide a high performance, high availability network capable of tolerating a broad scope of hardware, software, and operating system faults. The system can tolerate all but one known (and avoidable) single fault, two known and avoidable dual faults, and will detect all higher order fault sequences and provide diagnostics to allow for rapid manual recovery.

1 Introduction

Control systems on many operating commercial nuclear reactors are obsolete, and increasing maintenance problems dictate conversion to computer-based (digital) systems. At present, several such digital systems have been qualified and installed. However, they rely on complex software and a centralized system architecture. A consequence of the centralized architecture is that a failure in a software function can cause a failure of the entire system even if the hardware is replicated.

This paper describes a distributed fault-tolerant real-time control system called the Extended Distributed Recovery Block, or EDRB intended to replace analog systems as well as to be used in new designs. The application imposes some unique requirements because the design must be approved in a regulatory environment where standards are still evolving. Nevertheless, the potential benefits of the EDRB including modularity, higher reliability, greater functionality, better maintainability, and lower cost are significant and constitute the motivation for this work. Section 2 describes the EDRB architecture; section 3 discusses its fault tolerance characteristics; and section 4 describes the development methodology.

2 Architecture

The Extended Distributed Recovery Block (EDRB) is a variation of the Recovery Block, a fault tolerant software structure first proposed by Prof. Brian Randell in 1975 [RAND75], restated to include real time features [HECH86], and restructured for distributed systems [KIM84].

Figure 1 is a top level diagram of a real time control system based on the EDRB. There are three types of computers, i.e., nodes, on the network:

operational: response for active control of the reactor and related systems

supervisor: responsible for control of the network and the operational computers

auxiliary: responsible for non-control functions such as maintaining the system data log

Operational computers on the network are paired into active and shadow nodes. The active node executes a function, checks its result with an on-line assertion, and passes control to the shadow in the event of a failure. Both active and shadow nodes are required to issue periodic health reports to each other and to the supervisor. When such a report is not issued by the active node, the shadow requests permission from the supervisor to reset the failed node. If the supervisor concurs, recovery is attempted by means of a hardware restart of the failed computer. This mechanism will be effective for operating system faults (e.g., deadlock or non-availability of a critical resource) and transient hardware faults (e.g., a power surge or a memory read/write error). It is possible for an inconsistent state to occur, e.g., the active node spuriously decides to become a shadow node or a shadow node makes an incorrect decision to assume control.

In the event of such a state, the supervisor node will detect the problem from periodic status reports. It then sends an arbitration message to the operational nodes in order to restore consistency. However, a well behaved failure of the supervisor itself, i.e., one in which the supervisor ceases to output valid data, will not have any direct effect because of its passive role during normal operations. All analyses thus far have shown that in the event of such a failure, recovery and data restoration could be achieved within a matter of minutes, and that loss of a recovery capability during this time does not significantly affect overall system availability.

A simplified view of the software structure in a node pair and its interaction with the supervisor is shown in figure 2. During each iteration cycle or frame, the active node runs the primary version of the software while the shadow node runs the alternate version. If the primary routine fails to yield an acceptable result (as defined by an acceptance test) in the allowed time, control is passed to the shadow node. Output for the current cycle is lost, but the alternate routine results are output at the next cycle.

Within an operational node, a software module called the *node manager* is the focus of fault detection and recovery. The node manager determines the state of both its own computer and its companion. It determines whether to run the primary or alternate routine, and whether its status is active or shadow. It both generates a heartbeat message and monitors the heartbeat of its companion operational computer. When the companion computer heartbeat message is not received for two cycles, the node manager initiates a hardware restart after receiving concurrence from the supervisor.

The supervisor node provides system configuration, node role (i.e., active or shadow), and application specific parameters needed by the operational computers for initialization and recovery after restart. The supervisor must also consent to a restart request before any operational node performs a restart on its companion. Because issuing a restart command results in the disabling of an operational node (i.e., either the active or shadow computer) for a significant period of time, spurious restarts can cause system failures. For example, if the shadow node fails to detect a heartbeat from the active computer, the problem may exist in its own communication interface or in the operating system rather than a problem in the primary. Issue of a restart command under these circumstances could result in the system being left without any functional computer -- a situation far worse than if there were only a single computer in charge of control. This is the primary reason for requiring consent of a third computer, i.e., the supervisor before a restart is allowed. This strategy ensures that no single failure results in loss of system control. However, it does so at the cost of additional computing resources and a longer response time than would be necessary if no concurrence were needed before issuing a restart.

Transient failures in communication links are handled by allowing for the absence of two heartbeats before a restart sequence is initiated. This strategy also allows for a greater tolerance in cycle synchronization. However, in order for this measure to not present a system hazard, the smallest allowable control delay must be at least four times as large as the length of a cycle. Table 1 summarizes the fault detection and recovery logic used by both the node manager and supervisor in response to various system states.

The EDRB node manager considers the previous failure history before deciding which version of the software to run and on which node. For example, when there have been failures in the previous cycle using the primary routine and there is a failure in the current cycle, the alternate routine will be used and there will not be a switchover. The alternate routine will be used for three subsequent iterations and then the primary routine will once again be tried. Some special cases must also be considered in the recovery logic. For example, if there is a primary failure on one node and an alternate failure on the shadow, then control is retained at the active node, and the supervisor is notified. If this simultaneous failure happens multiple times, then the supervisor sounds an alarm. As was the case with the hardware/system software recovery strategy, this algorithm results in additional processing and complexity before a recovery action is attempted. However, the benefit gained by this additional computing is system stability and resultant increased reliability.

The EDRB network structure is shown in Figure 3. There are two types of networks: the *node pair* node network and the *system* network. This partitioning ensures reliable rapid response time for time critical functions between operational nodes for large networks. Each is dual redundant and currently implemented using an ARCNET [SMS87] token bus network. The node pair network is used for local communication between the operational nodes. Messages are passed between the nodes as short datagrams, i.e., connectionless messages. The transmission rate and the presence of only two stations means that communication delays contributed by the network will be small compared to the internal node computer buses. The global system networks are also dual redundant ARCNET networks which connect all operational nodes with the supervisor. Dual redundancy in the global network is necessary because of the large number of stations, any of which could issue a spurious reconfiguration request thereby halting communications for a period of several seconds. The probability of simultaneous on two networks is minimal.
Consideration was given to use of virtual circuits rather than connectionless datagrams on at least one of the system networks. Such communications is well supported on the operating system used in this design [QUAN88]. However, the heartbeat transmissions are repetitiive and time-critical. Thus, delays due to the retries which are a feature of virtual circuits can create significant problems. In some cases, the excessive delays brought on by retries together with the operating system responses to delayed communications resulted in loss of the network for periods of minutes.

Hardware resets are performed using a specially designed circuit board installed in each computer. The purpose of this board is to attempt a recovery when a hardware or operating system fault is detected. Design goals in the board were to prevent a spurious reset and to reliably provide a reset signal to the computer bus upon command. These design goals resulted in the use of two independent signal lines, one of which must be held high constantly while the other must be pulsed according to a specific sequence. If both operations are not performed correctly, the board will not output a reset signal. In order to ensure that a reset signal will be issued when a correct reset command has been issued, the logic is implemented on two separate channels.

Benefits of the EDRB architecture include:

- *Fault Containment:* Each computer is isolated by the well defined interfaces of the LAN. Interprocess communications occurs via message passing, and there is no common memory or time source. Hence, many sources of common mode failures on the control system are eliminated. For example, a function running on one computer can not affect a function on any other node by overwriting its memory or instruction space, causing the operating system to fail by entering an infinite loop, or related problems. Similarly, the effects of hardware failures on one processor will not propagate to adjacent processors through an electrical path because they do not share a common computer bus. The probability of a simultaneous failure induced by data, an interrupt, or other external cause is reduced.

- *Partitioning:* Rather than a single complex program to control the entire reactor, the distributed architecture allows the control function to be partitioned into many smaller, independent modules which are easier to design, code, and verify. One derived benefit of partitioning is extensibility due to well defined interfaces incorporated into the architecture. A second benefit is improved performance due to the increased computing power of multiple networked computers.

- *Maintainability:* The isolation and partitioning provided by the networks facilitate on-line maintenance of the system. A permanent hardware failure or non-recoverable software failure requires the installation of a replacement. However, this maintenance operation is relatively simple because of the network interfaces.

The absence of a common time source and shared memory which the fault containment benefits but complicates system coordination, management, and recovery. Three specific implementation challenges were:

- *Concurrency control of multiple copies of state data:* Both operational nodes and the supervisor must have accurate updated copies of system state data, i.e., iteration count, active node identification, active software version (primary/ alternate), and sensor/actuator staates. The general strategy is for the operational nodes to broadcast these data in the periodic heartbeat. The operational nodes exchange heartbeats over the dedicated redundant ARCNET with no appreciable delay. The operational nodes also transmit messages to the supervisor are passed over the shared system links that have a worst case transmission delay of 1 second in each direction. Under normal circumstances, this delay has no operational impact. If, however, the two operational nodes enter an inconsistent state, there will be a delay in the response time for the supervisor to arbitrate. The delay also affects the beginning of a hardware restart upon detection of no heartbeat. This is the worst case fault recovery time which is approximatley, in the current system, is approximately 3 seconds.

- *Synchronization:* Processors in a node pair can have a maximum time skew of one iteration cycle, i.e., 0.5 seconds. This time skew affects the recovery logic as well as the design of application software. Because of this time skew, all real time control algorithms must be able to tolerate an outage of two iteration cycles, i.e., 1 second, after an operational node hardware or software failure.

- Recovery Logic: The Operational node manager and supervisor routines needs to account for the time skew and must be based on the information received in the heartbeat messages. Each node manager must determine whether it has received a valid and current heartbeat message, whether the companion node is functioning, what its own status is, what version of the software it should run, and whether any responses are

outstanding (e.g., a confirmation for a request to switch or to restart the companion node).

3 Fault Tolerance Characteristics

The EDRB tolerates a broad range of hardware, system software, and application faults. Table 2 lists typical fault classes and the response of the EDRB architecture. The safety analyses performed in conjunction with the EDRB design (see next section) showed that there were no hardware single points of failure. Use of a conservative design and commercial components minimizes the possibility of a design flaw triggering a common mode failure. The redundant LANs and the separation of global and local communications virtually eliminates the chance of any single network failure (e.g., jabbering node or blocking tasks) disabling all network communications. Diverse global networks also minimize the possibility of a single fault disabling all communications with the supervisor. Moreover, loss of the supervisor is not in itself a system critical failure.

Although the operating systems are the same on all nodes, the chances of a simultaneous failure is small because the nodes are not running in lockstep and are not tightly synchronized. Furthermore, in nearly all cases, the operational nodes will be running different versions of the applications software (e.g., active runs the primary version and the shadow runs the alternate). As will be noted below, the operating system is a mature commercially available product with a large installed base. Although anomalies have been observed during development, most were in display management and had little to do with the basic task dispatching. Thus, there is a basis for assuming both independence (due to loose coupling) and a reasonably long system software MTBF.

The design does contain two single points of failure in each operational node software: the node manager and the acceptance test. The node manager is identical for all nodes in the system and can be subjected to an intensive verification even though it is not a simple section of code (approximately 1000 lines of C). The acceptance test is application dependent and must therefore be unique for each pair of operational nodes. It is therefore essential that the acceptance test for each application be kept simple in order to allow for thorough verification within cost and schedule constraints. The need for a simple acceptance test may be the primary criterion for partitioning a control system application into EDRB operational pairs.

Even if a node pair fails, however, the EDRB fault containment characteristics minimize the probability that the failure will propagate to other nodes. For example, the prototype application of the EDRB is the Sodium Process Facility at the Experimental Breeder Reactor II (EBR II) at Idaho Falls, ID. The sodium process facility is responsible for converting waste coolant from a sodium cooled reactor into a sodium hydroxide solution. The Barrel Drain System which melts sodium metal packaged in 55-gallon drums and empties it into a holding tank is independent of the sodium reaction system, which takes molten sodium from a separate set of holding tanks and converts it to a solution of sodium hydroxide and water. Thus, partitioning along subsystem boundaries allows the EDRB network to continue operation after a simultaneous node pair failure because that the corresponding system can be shut down without affecting the remainder of the facility.

4 Development Methodology

The development methodology is as important as the design because of the intended application. The ultimate success of the EDRB depends as much on regulatory acceptance of the development methodology as on the functionality and performance of the final product. Computer-based ("digital") control systems are a recent development in U.S. operating reactors.

Hence, there is currently little guidance provided by standards organizations or the Nuclear Regulatory Commission. Instead, appropriate standards and practices from other applications (primarily aerospace) were selected and appropriately tailored. Key elements of the development approach are:

- adherence to well-established standards and interfaces for hardware, system software, networks, and high level languages

- use of system safety analyses to identify potential single points of failure and other design weaknesses

- development on hardware which is nearly the same as the target system

- use of an intensive verification methodology for software components critical to sustaining system operation

- creation of tools on the development system to allow for distributed system testing including application-level fault injection and simulation of system software and hardware faults

4.1 Hardware, System Software, and LAN Components

The current practice of control system qualification for nuclear power plants required extensive safety analyses, special testing, and a quality assurance "audit trail" of paper work. This approach is impractical for computer-based systems and does not even consider the additional failure modes induced by software. In order to provide an equivalent high level of assurance in a digital control system, proven subsystems with a large base of operating experience should be used. This operating experience can substitute for an exhaustive analysis providing that failure modes and reliability characteristics are well understood.

IBM PC/AT compatible computers were chosen for the operational and supervisor nodes because of these considerations. Such computers can be purchased in industrial grade enclosures, and interfaces are available for any conceivable process control or data acquisition function. These systems have adequate capacity to meet a response time of less than one second. The QNX [QUAN88] operating system was chosen because it is a mature, stable, and well supported real time operating system. Its C language compiler supports all system functions including timing, interrupt handling, interprocess communications, interprocessor communications, and other input/output handling. This support reduces the probability of an obscure interface defect which might not be detected prior to operation.

The ARCNET [SMS87] local area network was chosen for its technological maturity, straightforward cabling scheme, deterministic response time (due to token passing), and error handling features implemented in hardware. Among the latter are a 16-bit CRC, ACK/NAK signals, and token loss detection together with a recovery protocol. Currently, the ARCNET is implemented on twisted pair, coaxial, and fiber optic cables at transmission speeds of up to 100 MBit/sec. The development system uses coaxial cable with a 2.5 MBit/sec transmission rate.

Experience during development has confirmed the expectations in this conservative hardware and system software configuration. With the exception of one bad batch of ARCNET LAN interface circuit boards, reliability of the hardware and system software was never an issue.

4.2 Safety Analyses

The objectives of the EDRB safety analyses are:

(1) identify potential failure modes and design errors,

(2) assess their impact on system functional performance, throughput, and safety,

(3) uncover design defects and omissions,

(4) verify the effectiveness of system fault tolerance, and

(5) facilitate subsequent qualification activities by performing the analyses early in system development.

The primary guidance for this work is the Department of Defense Military Standard 882A which identifies two types of Safety Analyses: Failure Modes and Effects Analysis (FMEA) and Fault Tree Analysis (FTA). The Failure Modes and Effects Analysis, which is further defined in Military Standard 1629A identifies strengths and weaknesses in the design by considering the effects of a failure of each part of the system. The most significant output of the FMEA is the critical items list which identifies hardware and software whose failures could cause a Class I (catastrophic) or Class II (severe) system level failure. The list is used to determine which components should be subjected to more rigorous testing.

FTA is a top-down approach in which specific system-level faults or hazards are postulated and the lower level failures that might cause these faults are identified. The output of the analysis are minimal cut sets, i.e., those sets of events whose simultaneous occurrence could cause a system-level severe fault. Those cut sets with only one or two events represent a single point of failure or two simultaneous events which potentially may cause the severe fault. They must be further analyzed to determine whether a redesign is necessary; justification will be provided if a finding for no further action is made.

4.3 Development System

The EDRB development system is virtually identical to the target system. The only differences are enclosures, power supplies, and other aspects of the hardware which are transparent to the software. Figure 4 shows the development system configuration which consists of a process simulation computer and an EDRB real time control. The simulation computer runs software to simulate subsystems of the Experimental Breeder Reactor II (EBR II) Sodium Process Facility [ANL88] and also includes a graphic simulation of operator control panels. There are two pairs of operational nodes, each of which controls a complete subsystem, i.e., responding to commands from the control panel, opening valves, operating heaters, monitoring flow, shutting

down heaters, preventing illegal sequences of operations, and sounding alarms.

The ability to develop the software on a such a close replica of the target system in conjunction with a simulation of the process to be controlled has been extremely valuable. Examples of anomalies which have thus far been observed include:

- faults related to timing and synchronization
- interactions causing "freezing" of operational nodes due to deadlocks related to serial port failures
- a set of conditions that could cause one of the global networks to be blocked indefinitely thereby prevent its use
- a set of conditions that caused indicators on the display panels to "ping-pong" or blink on and off regularly due to a design problem in the node manager

Had the development occurred in the usual manner, i.e., a separate host computer for development and compilation and a target computer for execution, these interactions would not have been observed until the integration phase, when correction is always more difficult and costly.

4.4 Verification

All software units were subjected to two inspection-based verification techniques (e.g., peer reviews and code walk-throughs) as well as functional testing using the facilities described in the next section. For critical software, the Extended Condition Table (ECT) methodology [TAI87], an automated implementation of the of a method described by Goodenough and Gerhart [GOOD75]. The basic principle of this method is to combine structural the following three software testing techniques in a manner that the strengths of each method overlap [HOWD82]:

1. *Structural*: Creation of test cases that exercise all paths through the program (a loop would count as one path).
2. *Special Values*: Exercise paths with special values of variables related to safety and reliability concerns identified in the system safety report. Special values are those just inside and outside the allowable ranges, at the boundaries of discontinuities, and the entry condition and exit conditions of loops.
3. *Functional*: Use additional values for variables and control actions related to safety and reliability concerns that are related to entry conditions for branches that are not expected to be part of a path, and ensure that the results are consistent.

4.5 Integration Testing and Validation

Integration testing is performed on the target system and consists of two major activities:

1. Ensuring that pairs of software modules interact through the entire ranges of all data passed between them.
2. Ensuring that the hardware, system software, fault detection and recovery

software, and aplication software function in accordance with their specification in the presence of injected hardware and software faults.

Interface testing was performed conventionally and does merit further discussion in this paper. However, the conduct of fault injection testing is a less defined area and hence, our approach is discussed in this section.

Test driver software was written for injection of software faults; data recording and analysis software on the operational nodes captures the results. Figure 5 shows the test configuration for the supervisor and two of the operational nodes. The network exercise the node manager, the single critical software module which is present in all operational nodes.

Test cases identified using the ECT (or an alternative approach) are written as test files which are read into the application driver resident on the supervisor node. The test case files are written as a sequence of commands which include simulating

- failure of either version of an application routine on either the active or shadow node in any of up to 255 distinct failure modes (the application software itself must be programmed to react to these messages),

- waiting for a certain time until the failure is induced,

- loss of input data,

- loss of communications with a companion node

Test cases can also be input from a console for shorter runs or to interactively investigate the behavior of the network. The behavior of the node managers on each of the operational nodes is recorded in real time on a RAM disk at each node. Each test can last up to 7 hours, or 50,000 frames. Because of the large amount of data that must be analyzed for long tests, additional analysis software was developed to locate changes or search for a given event

The objectives of the test configuration are to

- test behavior of the node pair including failure response time,

- measure coverage (percentage of faults detected and recovered from),

- determine if there are any conditions which could cause prolonged "ping-ponging" (passing of control back and forth between nodes),

- measure number of frames without output during switching

The test driver software and result files are easy to use. Thus, they are exploited during development for testing out design alternatives and for debugging, as well as during the subsequent verification and validation testing. Although additional capabilities would certainly be desirable for debugging (e.g., multinode message tracing, being able to single step multiple nodes, or a full multiple node instruction trace), the existing test system is a significant addition to the development effort.

5 Conclusions

Builders of fault tolerant systems are ultimately attempting to deliver confidence to their users, and hence, the architecture and development methodology are as important as any of the functional specifications or throughput measurements. When fault tolerant systems are applied to nuclear reactors, *qualification,* one of the most, if not the most, challenging design and development issues must be faced. As discussed in this paper, achieving qualification requires

- proper selection of standards and the ability to defend the choice before a licensing body in the absence of clear guidelins
- safety analyses to characterize the extent of fault tolerance and to identify potential single points of failure and subsequent thorough testing and verification to ensure that faults are avoided to the maximum extent possible at these weak points.
- a test platform which facilitates efficiency and productivity in order to maximize the benefit of the schedule and resources available for testing
- a conservative choice in commercially available components together with sufficient documentation to demonstrate the inherent reliability and robustness of products which are incorporated into the design
- a thorough understanding of potential failure modes which in turn require a great deal of operational knowledge, experience, and understanding. Wi

The system described in this paper is currently still in development at SoHaR. It has an average measured response time of under 1 second to a broad range of hardware, system software, and application software faults which have been injected during testing. It can tolerate all but one known (and avoidable) single fault, two known and avoidable dual faults, and will detect all higher order fault sequences and provide diagnostics to allow for rapid manual recovery. The network will be tested at the Argonne National Laboratory Experimental Breeder Reactor II Sodium Process Facility (Idaho Falls, Idaho) in May, 1990. However, the major technical achievement in this effort, if licensing and qualification are obtained, is an example of the types of activites necessary to create a dependable, life critical real time distributed control system.

6 Acknowledgements

This work was performed under Small Business Innovative Research (SBIR) Contract AC03-87-ER80532 from the U.S. Department of Energy and Contract NAS1-18811 from the NASA Langley Research Center for the ECT Verification research. The authors wish to acknowledge the support and interest of Mr. Harry Alter, the technical monitor. We also wish to thank Dr. J. Sackett, L. Christensen, G. Chisholm, C. Groves, and R. Collins of the Argonne National Laboratory EBR II facility for their assistance in the Sodium Process Facility Direct Digital Control System. Finally, we express our gratitude for the additional assistance and technical guidance we have received from Mr. J.D. White, T. Wilson, and R. Kisner from the Oak Ridge National Laboratory.

References

ANL88 Argonne National Laboratory, EBR II Engineering Branch, "Functional Requirements Document for the Sodium Process Facility Direct Digital Control System", Idaho Falls, 1988

GOOD75 J.B. Goodenough and S.L. Gerhart, "Toward a Theory of Test Data Selection", *IEEE Trnasactions on Software Engineering*. Vol. SE-1, No. 2, June, 1975, pp. 156-173

HECH86 H. Hecht and M. Hecht, "Fault Tolerant Software", in *Fault Tolerant Computing*, D.K. Pradhan, ed., Prentice Hall, Englewood Cliffs, NJ, 1986

HOWD82 W.E. Howden, "Validation of Scientific Programs", *ACM Computing Surveys*. Vol. 14, No. 2, June, 1982, pp. 193-227

KIM84 K.H. Kim, "Distributed Execution of Recovery Blocks: An Approach to Uniform Treatment of Hardware and Software Faults", *Proc. 4th International Conference on Distributed Computing Systems*, pp. 526-532, May, 1984, available from the IEEE

NPEC82 ANSI/IEEE-ANS-7-4.3.2-1982 "Application Criteria for Programmable Digital Computer Systems in Safety Systems of Nuclear Power Generating Stations", American National Standards Institute, New York

MIL-STD-882A "Safety Program Requirements", Available from Naval Publications Center, Philadelphia, PA

MIL-STD-1629A "Failure Modes Effects, and Criticality Analysis", Available from Naval Publications Center, Philadelphia, PA

QUAN88 Quantum Software Systems, Ltd., *QNX Reference Guide, Version 2.1*, Available from Quantum Software, Kanata, Ottawa, Canada, 1988

RAND75 B. Randell, "System Structure for Software Fault Tolerance", *IEEE Transactions on Software Engineering*, vol. SE-1, pp. 220-232, June, 1975

SMS87 Standard Microsystems Corporation, *ARCNET Local Area Network Controller Designer's Guide*, available from Standard Microsystems Corporation, Hauppage, NY, January, 1987

TAI87 Ann Tai, M. Hecht, and H. Hecht, "Extended Condition Table Method for Verification of Critical Software", Proc. COMPSAC'87, IEEE Catalog No. 87CH2422, Tokyo, July, 1987

Table 1. Examples of EDRB Recovery Logic

Condition	Node Manager Response	Supervisor Response
Normal Operation	Supply heartbeat to companion computer and status report to supervisor	Monitor Messages and record in log (no further action required)
Primary Routine Failure	If active, then request switch in status. If no other failures have occurred, then companion will confirm switch. Run alternate routine in next frame, then attempt to run primary. Inform companion when successful.	Monitors for consistent state (i.e., only one node active, only one node shadow)
Alternate Routine Failure	If shadow, then inform companion. If active companion is running primary, then continue to run alternate. If active, then request switch in status. If repeated failure on both primary and alternate, then perform auto-reset.	Monitors for consistent state (i.e., only one node active, only one node shadow)
Initialization	Determines initial role (active or shadow) based on node number and whether companion is already functioning.	None
Missing heartbeat	Send restart request to supervisor after two heartbeats are detected as missing	Responds with request OK or request denied. Logs request and informs operator if anomaly
Recovery	After receipt of OK from supervisor, node manager causes a restart in the companion computer through a hardware interrupt	Log action
	Detect situation and refer to supervisor	Specific actions depend on application and system being controlled. Supervisor currently only sounds alarm.

Table 2. EDRB Fault Tolerance Characteristics

Fault Class	System Response

Failure of the primary routine to complete execution in the required time or failure to produce a correct result (as determined by the acceptance test)	Operating system times out application, acceptance test detects failure, node manager requests switch to companion node. Companion node assumes control in next cycle and outputs result of alternate routine
Application is not executed	Node manager senses absence of acceptance test result, requests switch to companion node
Faults in the computing node hardware or system software	Node manager in companion node senses absence of heartbeat, assumes active role, and requests permission to reset from supervisor
External physical disturbances (e.g., power outage, fire, earthquake, etc.)	Separate power supplies, physical separation of companion nodes, separate routing of LAN cables
Spurious reset request	Supervisor denies reset requests. Repeated spurious requests cause a threshold to be exceeded which will cause an alarm for an operator response
Failures in communication links	CRC error detection on each of two redundant LANs. Absence of valid message on one LAN causes node manager to seek message from alternate link.
Absence of sensor input data	Detected by application and reported to node manager. Absence for two or more frame causes request to switch. Request will be denied by companion if data are absent on links to both operational nodes.
Multiple closely spaced failures in primary or alternate routine	Node manager algorithm prevents switching unless node has previously run the application successfully. Thus, repeated failures will cause control to remain in one node and to be retried until successful.