Proc. 10th International Conference of the Israel Society for Quality Assurance, Jerusalem, November, 1994

Integrated Hardware and Software Fault Tolerance for Real Time Applications

Myron Hecht
SoHaR Incorporated
8421 Wilshire Blvd. Suite 201
Beverly Hills, California 90211, U.S.A

Abstract

A distributed fault tolerant system for real time process control based on an enhancement of the distributed recovery block is described. Coverage is provided for failures in hardware, system software, networks, and application software. Fault tolerance provisions are introduced at the system level and in application software using an architecture based on the distributed recovery block (DRB). This implementation allows use of standard off-the-shelf hardware and software components providing life cycle cost and extensibility benefits. Maintainability is enhanced through an automated restart capability and a logging function.

1. INTRODUCTION

Computer-based control is becoming more sophisticated and being applied to an increasing variety of applications. Many of these applications require high reliability. Examples are as diverse as pollution control, medical systems, building control, and radioactive waste management. This paper describes a new low cost technology, called the ***Enhanced Distributed Recovery Block,*** or ***EDRB*** to meet this need. The EDRB, is a fault-tolerant real-time distributed system architecture for critical process control applications. The underlying fault tolerance mechanisms are based on extensions to the Distributed Recovery Block [Kim84] which is in turn based on the classical recovery block [Randell75] with real time extensions [Hecht86]. Because of its ability to detect and recover from both hardware and software failures, the EDRB architecture has the potential to resolve common mode software failures concerns in safety grade systems. In addition, the architecture can be applied to control systems where a primary software version has been optimized for performance and economy and an alternate software version serves as a "lifeboat" to prevent a reactor trip in the event that the primary system fails to properly execute its control functions.

The EDRB is an outgrowth of research in distributed system fault tolerance at the University of California. In 1987, SoHaR was awarded a U.S. Department of Energy contract to develop a prototype system which was successfully demonstrated at the Experimental Breeder Reactor II (EBR II) in the spring of 1990. The EDRB is the commercial follow-on of this development. Enhancements on this architecture have been made to enhance performance, provide disk replication ("mirroring") The underlying design is also being implemented in a high speed version on a pair of VME chassis [Marzwell94] and on UNIX-based workstations for use in command and control

applications under a contract with the U.S. Air Force Rome Laboratory.

The following section provides a high level description of the overall architecture. Sections 3 and 4 describe the fault tolerant design and performance.

2. TOP LEVEL ARCHITECTURE

Figure 1 shows a top level diagram of an EDRB system for either safety or control applications. The primary building blocks of the architecture are pairs of redundant processors which are connected by redundant networks called *operational nodes*. Operational nodes are PC/AT compatibles based on the ISA, EISA, PCI, or Std-32 busses. These processors are responsible for control and also store non-recoverable state information (e.g., system modes, or status of control sequences). A pair of redundant operational nodes is simply called a *node pair*. Each operational node also has a specially designed reset circuit board which generates a hardware reset based on the receipt of an appropriate coding sequence from the companion operational node. This mechanism is effective for automated restoration of resources disabled by operating system failures (e.g., deadlock or non-availability of a critical resource) or transient hardware faults (e.g., a power surge or a memory read/write error). Reset signals are generated under software control of the companion operational node working with the consent of the supervisor node.

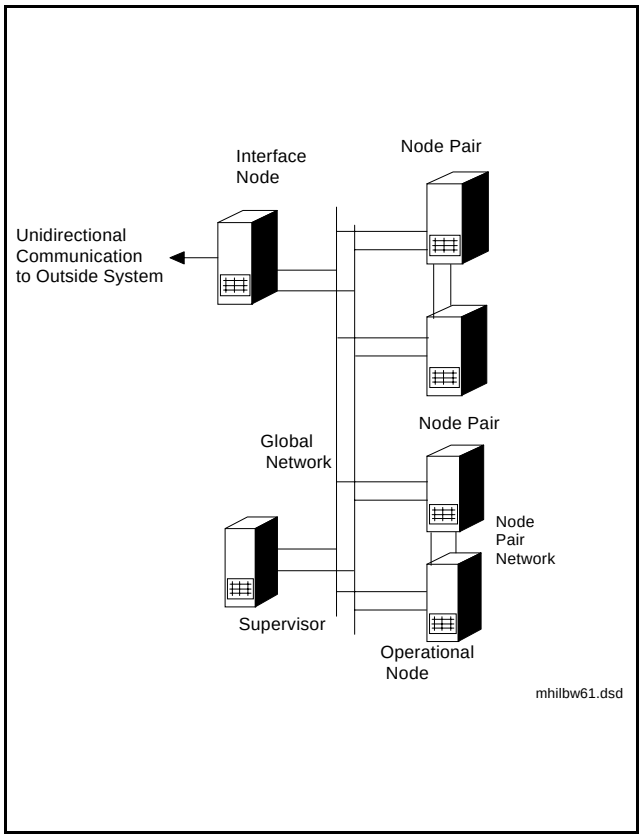

Figure 1 Top Level EDRB Architecture

Two levels of networks are used: a *node pair network* and a *global network*. The node pair network connects the two operational nodes in a node pair and is used only for communication between these nodes. This configuration minimizes communication delays and contains failures that might otherwise propagate to other node pairs. The node pair network is used for local communication between the operational nodes. The global network connects all operational nodes with the supervisor and display nodes. This traffic is less time critical than on the node-pair network. In order to limit the effects of global network noise, chattering nodes, or other active failure modes, operational nodes will disregard messages for a *cool off time* period once a message traffic rate threshold is exceeded.

The EDRB architecture allows for operational nodes to be interfaced directly to the plant systems which they control or to plant-wide I&C networks or "data highways". The benefits of direct

interfacing are (1) enhanced fault containment because only a limited portion of the control system and a limited portion of the plant are interconnected and (2) faster response times. The benefits of connection to data highways are simpler hardware interfacing and to use hierarchical control.

3. FAULT TOLERANT DESIGN

The major innovation of the EDRB is its ability to simultaneously manage both hardware and software fault tolerance. Hardware fault tolerance and system software fault tolerance is handled using a scheme called *active redundancy*. One node pair member is always *active* (i.e., sends control signals to the plant); the other is *shadow* (i.e., standby). Members of the node pair exchange periodic status messages called *heartbeats*. If an operational node senses the absence of its companion's heartbeat, it requests confirmation of the failure from a second kind of node called the *supervisor*. If the supervisor confirms the absence of a heartbeat, the operational node will undertake recovery actions as described below.

Figure 2 Simplified EDRB software structure

Understanding the software fault tolerance requires a model of the node pair software architecture shown in figure 2. Under normal circumstances, the active node executes a *primary* version of a control task in parallel with an *alternate* version executed on the shadow node. Both nodes check the correctness of the results of the control task outputs with an *acceptance test*. Whether the control signal is actually sent to the plant is determined by a set of routines which are collectively referred to as the *executive layer* whose operation will be described in detail below. The most usual action of the executive layer routines is to output the control message of the primary routine if the acceptance test is passed. Actions of the executive layer under other circumstances will be described below.

Operational Node Fault Tolerance

The operational node fault detection and recovery software is implemented in the *executive layer*, a set of tasks running under the QNX (tm) real time operating system [Quantum92]. The two most important tasks in this layer are the *node manager*, which is associated primarily with the node on which it resides, and the *monitor* which is concerned primarily with generating a heartbeat and determining the state of the companion node.

The node manager task determines which *version* of the application software to run (i.e., the primary or alternate routine), and the *role* of the local node (i.e., active or shadow). Under normal circumstances, the application completes execution within its allotted time and passes its acceptance test criteria. The node manager will then allow output to the plant if the node is in the active role

at the time of application completion. However, if the node manager task determines that a role switch is necessary because of an application failure or timeout, it immediately sends a *switch request* to the companion. When a node manager receives switch request, it immediately sends a *switch confirm* to the companion. Switch requests and confirms are sent to the supervisor node only at the beginning of each heartbeat frame in order to limit message processing workload during recovery actions. The node manager considers the previous failure history before deciding which version of the software to run and on which node. A more complete description of the logic is described elsewhere [Hecht91]. The monitor task is responsible for (1) generating periodic heartbeats indicating the state of the node on which it is running and (2) detecting and recovering from failures on the companion node manifested by a sustained loss or delay of heartbeats (i.e. power failure, operating system crash, etc.). Any time the monitor determines that the most recently received companion frame number is 3 or more frames behind its own, it begins transmitting companion restart requests to the supervisor. These requests continue, one per frame, until one of two events occur: (1) the companion node's heartbeat falls back into the range of acceptability, or (2) the monitor receives a companion restart approval message from the supervisor. In the latter case, the monitor would then signal the node manager task to take over the active role and run the primary version of the application, irrespective of its previous state. The monitor task also determines a node's role or version during system startup (or restart). The logic used by the monitor in determining the node's initial state is as follows: if the companion node has already communicated that it is running in the active role, begin by running shadow/alternate, otherwise run active/primary. This information is then passed to the node manager, which takes over control of the node's role and version for the duration.

Supervisor Node Fault Tolerance

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 for a period of tens of seconds, 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.

Because the supervisor operates in strictly a passive role, it is not a single point of failure. It can not cause an operational node to change its state through a command or a restart request. Thus, a supervisor failure impacts the ability of the system to recover from unannounced failures involving

the monitor task, but does not affect system operation in the absence of such failures. The supervisor log is recorded to non-volatile storage, and the actions of the supervisor require three or less frames of operational history. Thus, the supervisor can fail and be manually restored. The EDRB tolerance of an extended supervisor outage allows greater flexibility in the design of the supervisor, assurance of an unambiguous single point of higher level system control, and a more sophisticated user interface. Results of quantitative availability analyses have shown that an outage on the order of minutes of the supervisor has little effect on overall system availability .

The EDRB can tolerate both well behaved (i.e., "fail silent") and less benign supervisor faults. For example, if the supervisor outputs invalid reset confirmations, there will be no effect unless a second failure occurs, i.e., a faulty shadow node believes it has issued a restart request. A "babbling" supervisor will be disregarded because operational nodes limit the amount of messages that they will process in a frame.

4. PERFORMANCE

Two key parameters in assessing the quantitative performance of fault tolerant systems are:

- *Recovery time:* Real time process control requires that hardware and software be ready to take over *rapidly* (in tenths of a second) in order to avoid process disruption, or even worse, system instability. Hence, recovery time is a key parameter in determining the suitability of a fault tolerant architecture for a real time application.

- *Coverage*: Coverage is the probability that a failure will be recovered from given that one has occurred. Although the details of such a model vary according to the implementation of each system, coverage in particularly dramatically affects system reliability in a dual redundant configuration [Arnold73].

For the EDRB, these parameters were empirically determined [Hecht93]. Our results indicate

- a recovery time of less than 20 milliseconds for failures covered by specific detection mechanisms such as an acceptance test or a switch request] and of less than 100 milliseconds (average of 65 milliseconds) for failures detected by means of a timeout. These results were achieved with a relatively modest hardware configuration (Intel 486 33 MHz processors).

- a coverage value of unity (1.0) for a single failure, and 0.98 (3 sigma lower limit value of 0.97) and a recovery time of 60 millseconds (using 20 msec frame sizes) *with an average of 2 simultaneous failures*.

The significance of these results is that the EDRB allows ***continuous operation*** in the presence of failures. Other types of fault tolerance, often referred to as *high availability*, are *not* suitable for real time control because tens of seconds or longer are necessary to restore service. For example, high

availability for office based applications or interactive transaction processing (e.g., ticket reservation systems or stock trading systems) only guarantees that the integrity of data stored on disk during a hardware or system software failure.

5. CONCLUSIONS

The EDRB is an important step forward in fault tolerant technology. It provides a number of advantages over other approaches due to:

- **Software and Hardware Fault Tolerance:** The most common connotation of fault tolerance is with respect to hardware failures. However, hardware fault tolerance alone is not enough. Software causes hangs, incorrect output, and other failures. Software defects have caused highly visible failures in electronic telephone switching systems, aircraft autopilots, satellites, medical equipment, and air defense. In many systems, software is already the leading cause of failure due to complex new functionality such as graphical user interfaces. The architecture described here provides integrated software *and* hardware fault tolerance for a comprehensive solution to your computer dependability concerns.

- **Adherence to standards**: A safety critical system is a significant investment of dollars, time, and technical commitment. In order to maximize the benefit of capital and research dollars, this system must be built for the future. It must be based on non-proprietary architectures. The EDRB runs on any IBM PC/AT compatible platform, uses ARCNET, ANSI C, and is compatible with the IEEE POSIX 1003.1, .2, and .4 standards. As a result, the applications developed under this architect ure will be extensible, and compatible with a wide array of existing hardware and software components.

- **Cost Effectiveness**: Constrained capital equipment budgets and a competitive environment require cost effective solutions. Because the EDRB is implemented on the IBM PC/AT architecture, highly dependable computing can be offered at significantly lower cost than any other alternative.

REFERENCES

Arnold73 T.F. Arnold, "The Concept of Coverage and Its Effect on the Reliability Model of a Repairable System", *IEEE Trans. on Computers*, Vol. C-22, pp. 251-254, March, 1973

Hecht86 H. Hecht and M. Hecht, "Fault Tolerant Software", in *Fault Tolerant Computing*, D.K. Pradhan, ed., Prentice Hall, Englewood Cliffs, NJ, 1986

Hecht91 M. Hecht, J. Agron, H. Hecht, and K.H. Kim, "A distributed fault tolerant architecture for nuclear reactor and other critical process control applications", *Proc. IEEE 21st Fault Tolerant Computing Symposium*, Montreal Canada, June, 1991

Hecht93 M. Hecht, J. Agron, and H. Hecht "A Distributed Software Fault Tolerant Architecture for Instrumentation and Control Applications", *Proc. American Nuclear Society meeting on Nuclear Plant Instrumentation, Control, and Man-Machine Interfaced Technologies, Oak Ridge, TN*, April, 1993

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

Marzwell94 N. Marzwell, K.S. Tso, M. Hecht" An Integrated Fault Tolerant Robotic Controller System for High Reliability and Safety" *Proc. Technology 2004*, Washington, DC, November, 1994

Quantum92 Quantum Software Systems, Ltd., *QNX Reference Guide, Version 4.1*, Available from Quantum Software, Kanata, Ottawa, Canada, 1992

Randell75 B. Randell, "System Structure for Software Fault Tolerance", *IEEE Transactions on Software Engineering*, vol. SE-1, pp. 220-232, June, 1975