Contents

8 The Distributed Recovery Block Scheme **189**
8.1 INTRODUCTION 189
8.2 NON-NEGLIGIBLE FAULT SOURCES AND DESIRABLE RECOVERY CAPABILITIES 190
8.3 BASIC PRINCIPLES OF THE DRB SCHEME 192
8.4 IMPLEMENTATION TECHNIQUES 198
8.5 EXPERIMENTAL VALIDATIONS OF REAL-TIME RECOVERY 202
8.6 ISSUES REMAINING FOR FUTURE RESEARCH 205
8.7 CONCLUSIONS 207

8

The Distributed Recovery Block Scheme

K.H. (KANE) KIM
University of California, Irvine

ABSTRACT

The *distributed recovery block* (DRB) scheme is an approach for realizing both hardware fault tolerance (HFT) and software fault tolerance (SFT) in real-time distributed and/or parallel computer systems. Since the initial formulation in 1983 of the basic structuring and operating principles by the author, the scheme has been steadily expanded with supporting testbed-based demonstrations and the expansion is still continuing. In this chapter, the basic principles of the DRB scheme, the implementation techniques established, and some of the demonstrations conducted, are briefly reviewed. The abilities for real-time detection and recovery for intentionally injected hardware faults and software faults were demonstrated but the experimental works have not yet gone far enough to test the abilities for tolerating unforeseen software faults. The limitations of the DRB scheme as well as major issues remaining for future research are also discussed.

8.1 INTRODUCTION

The *distributed recovery block* (DRB) scheme is an approach for realizing both hardware fault tolerance (HFT) and software fault tolerance (SFT) in real-time distributed and/or parallel computer systems. Since the initial formulation in 1983 of the basic structuring and operating principles [Kim84a], the scheme has been steadily expanded with supporting testbed-based demonstrations and the expansion is still continuing.

It uses a *pair of self-checking processing nodes* (PSP) structure together with both the software implemented internal audit function and the watchdog timer (WDT) to facilitate real-time HFT. For facilitating real-time SFT, the software implemented internal audit function and multiple versions of a real-time task software which are structured via the *recovery block*

Software Fault Tolerance, Edited by Lyu

scheme [Hor74, Ran75, Lee78, Ran94] and executed concurrently on multiple nodes within a PSP structure, are employed.

The DRB scheme has been established with emphasis on the following three aspects.

(1) The target applications are of the real-time application type. Therefore, techniques which incur excessive overhead or yield excessive fault latency or recovery time have been consciously avoided.
(2) The target systems in which the DRB scheme will be used, are primarily *distributed computer systems* (DCS's) and *parallel computer systems*.
(3) The faults to be dealt with are both hardware faults and software faults. Therefore, the DRB scheme has been developed in such a form that proven HFT techniques can be easily integrated into the DRB scheme.

The decision to focus on real-time applications was made early on due to our belief that the significant advances achieved by computer industry in late 1970's and 80's pushed the issue of HFT in non-real-time or soft-real-time systems off the list of critical research topics. The emphasis on developing techniques applicable to DCS's and parallel computer systems was also based on our decision to be in line with technology evolution trends. In early 1980's, it was already safe to say that all future real-time application systems of moderate to large sizes and complexities would contain DCS's and some of these DCS's would in turn contain parallel computer systems as subsystems.

The emphasis on handling not just hardware faults but also software faults was based on our recognition that while the computer manufacturers succeeded in getting the issue of HFT under control, no similar progresses were made in the area of SFT. In spite of the hopes spread widely in the software engineering research community in 1970's, it became clear by early 1980's that production of largely error-free real-time software of non-trivial sizes was as distant a goal as ever, certainly not achievable before the entry into the 21st century.

In this chapter, the basic principles of the DRB scheme, the implementation techniques, and some of the demonstrations conducted, are briefly reviewed. The abilities for real-time detection and recovery for intentionally injected hardware faults and software faults were demonstrated but the experimental works have not yet gone far enough to test the abilities for tolerating unforeseen software faults. The limitations of the DRB scheme as well as major issues remaining for future research are also discussed.

8.2 NON-NEGLIGIBLE FAULT SOURCES AND DESIRABLE RECOVERY CAPABILITIES

8.2.1 Hardware Faults

Both permanent hardware faults, e.g., the crash of a processing node (consisting of one or more processors, local memory, communication, and other peripheral components), and temporary malfunction of hardware components are non-negligible potential causes for appearance of incorrect computational results. These faults are among the types of faults which DRB scheme was designed to handle, although the essential core of the scheme alone does not have the best coverage of such faults. However, the DRB scheme provides a flexible structure into which various proven HFT techniques [Avi71, Car85, Toy87], e.g., *triple modular redundancy* (TMR), *pair of comparing pairs* (PCP) used in Stratus computer series [Wil85], *error correcting code* (ECC), etc., can be easily incorporated.

8.2.2 Software Faults

During 1980's and even in late 1970's major computer system vendors have succeeded in producing fault-tolerant and yet economic modules of substantial complexity, thereby enabling production of "cost-competitive" computer systems with significant HFT capabilities and opening a sizable market for fault-tolerant computing. Therefore, the remaining challenge is in the problem of tolerating design faults, primarily in the software domain. Besides the problems related to incomplete or inconsistent specifications of computation requirements, there are largely two ways in which software faults arise. One major source for software faults is the selection of inadequate or insufficient algorithms which do not cover all realistically possible application situations. The other major source for software faults is the error in converting a selected algorithm into a program for a specific execution environment. Avoiding software faults is particularly difficult in real-time DCS's. Formal verification of designed algorithms or implemented programs has not yet advanced to the point where the absence of any error in sizable real-time software can be verified by a machine even with a moderate amount of assistance from human verifiers. Rapid progress in this formal verification technology is not likely to occur before we are well into the 21st century.

In a DCS, an algorithm may involve concurrency and distributed segments. A frequently occurring class of design faults are the *integration faults*. When a large number of modules are assembled into a DCS, the system developer often overlooks the possibilities of the system entering inconsistent states. This is due to the functional complexity of the integrated system. Individual components meet their specifications but the specifications are often incomplete, if not inaccurate, with respect to their integration with others. Therefore, it is nearly impossible to avoid such integration design faults in developing large-scale DCS's.

8.2.3 Generic Forward Recovery and Action-Level Fault Tolerance

In real-time applications, any computation results sent to the environment or the passage of real time can not be recalled. Therefore, the room for using *backward recovery* in a beneficial way is rather severely limited. This is not to say that old state information is not needed in recovery. A design issue here is how and what part of the recorded old state information can be utilized in achieving *forward recovery*.

A highly desirable forward recovery technique is one of a generic type. In the absence of generic forward recovery schemes, the design of fault-tolerant real-time computer systems will remain as a highly artistic error-prone discipline. Many *exception handling* approaches that have appeared in literature are examples of highly application-specific forward recovery schemes. Such approaches assume localization of the damage (due to the fault) to a very small area within the system, e.g., one or two program variables.

More generic forward recovery schemes which are capable of dealing with a broader range of fault conditions and more widely spread damages include the DRB scheme, the voting N-version programming (NVP) scheme [Avi75, Avi85], the PSP scheme, the TMR scheme, etc.

In addition, the degree of recovery achieved by an adopted forward recovery scheme is of great importance. The most desirable type of computer systems for use in safety-critical applications are those that have the capability of *action-level fault tolerance*, i.e., accomplishing critical actions (output actions of critical real-time tasks as specified) successfully in spite of component failures. Therefore, techniques for realizing action-level fault tolerance aim for

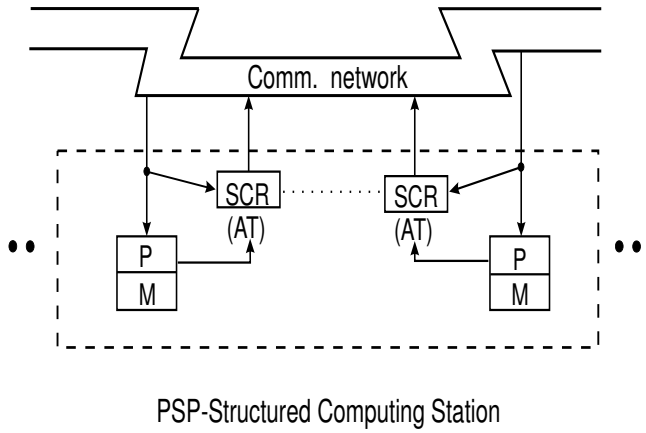

Figure 8.1 A PSP station

much higher degree of dependability than those aimed for merely aborting some tasks and cleansing system states upon component failures. Both the DRB scheme and the voting NVP scheme mentioned above are action-level fault tolerance schemes.

8.3 BASIC PRINCIPLES OF THE DRB SCHEME

The development of the DRB scheme started with a view of a real-time DCS as an interconnection of *computing stations*, where a computing station refers to a processing node (hardware and software) dedicated to the execution of one or a few application processes. Some computing stations may perform system resource management functions only. The DRB scheme is a technology for constructing highly fault-tolerant computing stations. As mentioned in Section 8.1, the DRB scheme is a composition of two component technologies: the *pair of self-checking processing nodes* (PSP) scheme and the *recovery block* (RB) scheme.

8.3.1 The PSP Scheme

The essence of the PSP scheme is to use two copies of a *self-checking computing component* in the form of a *primary-shadow* pair [Kim93a] as shown in Figure 8.1.

A computing component is said to be *self-checking* if it possesses the capability of judging the reasonableness of its computation results. The internal audit logic represented as AT in Figure 8.1 for checking the computation results in a processing node can be implemented with or without special hardware support. A well known case of the PSP scheme implemented with a special internal audit hardware mechanism is the *pair of comparing pairs* (PCP) scheme used in the Stratus system [Wil85]. A generic software approach for the internal audit logic is to provide a run-time assertion or an acceptance test routine [Hor74, Ran75, Yau75].

The PSP scheme imposes a restrictive constraint on the structure of a computing component. Each computing component iterates computation cycles and each cycle is *two-phase structured*. That is, each cycle consists of an input acquisition phase and an output phase. During an input acquisition phase, the component may take one or more data input actions

along with data transformation actions but no output action. Analogously, during an output phase, input actions are prohibited and one or more data output actions are allowed.

Basically, the PSP structure is adopted to facilitate parallel replicated execution of real-time tasks without incurring excessive overhead related to synchronization of the two partner nodes in the same PSP-structured computing station. Figure 8.2 illustrates a PSP-station in a local area network (LAN) based DCS. Here the self-checking function is implemented in the form of an acceptance test routine. For the sake of simplicity in discussion, Figure 8.2 depicts the special case where the following assumption holds:

(A1) The arrival rate of data items is such that each time a data item arrives, no other data items are being processed.

Node A is the initial *primary* node and node B is the initial *shadow* node within this computing station. Each of the partner nodes in a PSP station is assumed to contain its own local database which keeps persistent information used during more than one task cycle. Both nodes obtain input data from the multicast channel (built on a system-wide multi-access communication network). The next step for the primary node A is to inform the shadow node B of the ID of the data item that the former received (or selected in general) for processing in the current task cycle. This step is not essential in this special case subject to (A1). Node A and B process the data item and perform their self-checking concurrently by using the same acceptance test routine. Since node A passes the test, it delivers the results to both the successor computing station(s) and node B, and then starts the next task cycle. By receiving the output from node A, node B detects the success of node A and, if node B has also succeeded in its acceptance test, it too starts the next task cycle.

Suppose that the PSP-structured computing station in Figure 8.3 is the successor station. Nodes C and D process the data received from node A and perform their self-checking concurrently but this time the primary node C fails in passing the acceptance test or crashes during the processing of the data item whereas the shadow node D passes. Node D will learn the failure of node C by noticing the absence of output from node C. Node D then becomes a new primary and delivers its task execution results to both its successor computing station(s) and node C. Meanwhile, node C, if alive, attempts to become a new useful shadow node by making a retry of the processing of the saved data item. If node C passes the self-checking test this time, it can then continue as a useful shadow node and proceeds to the next task cycle.

In many applications, assumption (A1) does not hold. It is thus necessary to provide input data queues in each node within a PSP station. Each node may contain multiple input data queues corresponding to multiple data sources. Therefore, it is important for the partner nodes in a PSP station to ensure that they process the same data item in each task execution cycle. This is the main reason why the step of reporting the ID of the selected data item is taken by the primary node in Figure 8.2 and also in Figure 8.3. The gap between the time the primary node acquires a new input data item and the time the shadow node acquires a copy should be within a reasonable bound, certainly not to exceed a task execution cycle of the primary node by much. In addition, in typical DCS's with multicast channels, receiving data and placing them into input data queues within each node is handled by an independent unit (often called a LAN processor) operating concurrently with the main processor(s) in the node executing the application task(s).

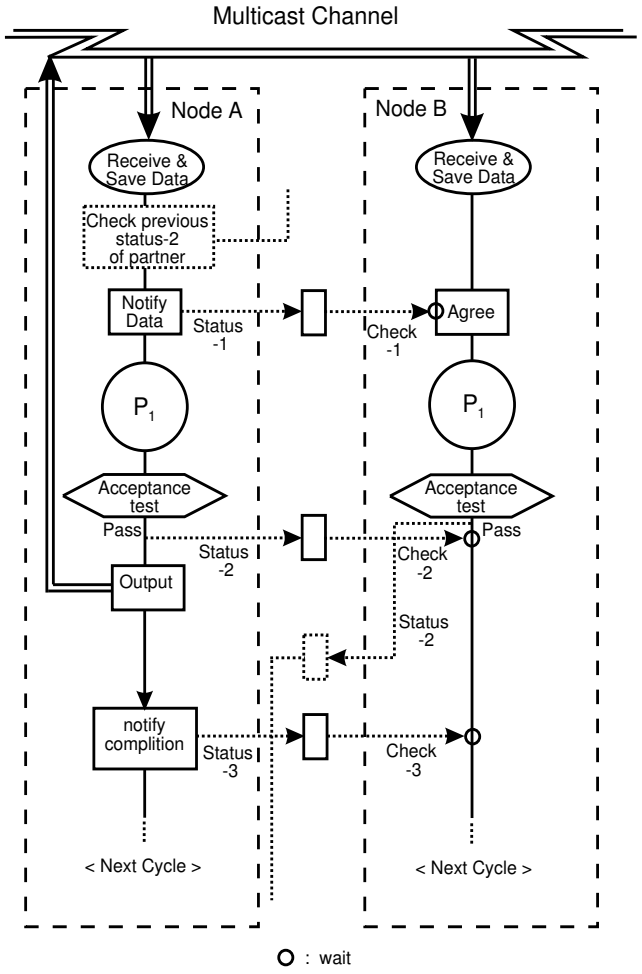

Figure 8.2 A fault-free task execution cycle of a PSP station

8.3.2 The Algorithm Redundancy Component of the DRB Scheme

In order to support handling of not only hardware faults but also software faults, the above primary-shadow PSP scheme can be extended by incorporating the approach of using multiple versions of the application task procedure. Such versions are called *try blocks*. The extended scheme is the *distributed recovery block* (DRB) scheme and it uses the recovery block language construct to support the incorporation of try blocks and acceptance test [Hor74, Ran75, Ran94].

As seen in Chapter 1, Section 1.3, the syntax of recovery block is as follows: *ensure* T *by* B_1 *else by* $B_2 \cdots$ *else by* B_n *else error*. Here, T denotes the *acceptance test* (AT), B_1 the *primary try block*, and B_k, $2 \leq k \leq n$, the *alternate try blocks*. All the try blocks are designed to produce the same or similar computational results. The acceptance test is a logical expression representing the criterion for determining the acceptability of the execution results of the try blocks. A try (i.e., execution of a try block) is thus always followed by an acceptance

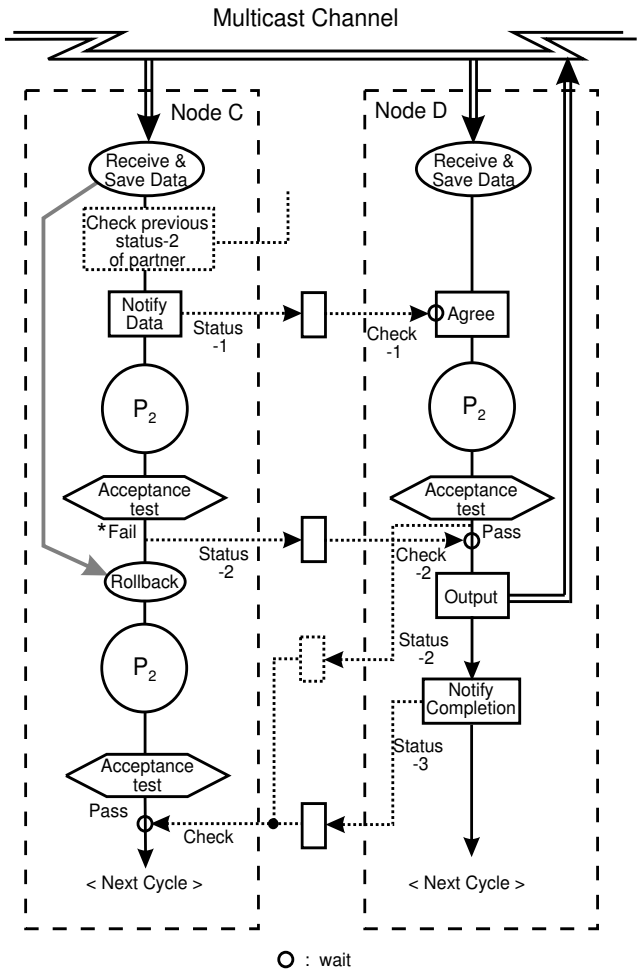

Figure 8.3 A task execution cycle of a PSP station involving a failure

test. If an error is detected during a try or as a result of an acceptance test execution, then a rollback-and-retry with another try block follows.

In the DRB scheme, a recovery block is replicated into multiple nodes forming a *DRB computing station* for parallel redundant processing. In most cases a recovery block containing just two try blocks is designed and then assigned to a pair of nodes as depicted in Figure 8.4. A try not completed within the maximum execution time allowed for each try block due to hardware faults or excessive looping is treated as a failure. Therefore, the acceptance test can be viewed as a combination of both *logic* and *time* acceptance tests.

As shown in Figure 8.4, the roles of the two try blocks are assigned differently in the two nodes. The governing rule is that *the primary node tries to execute the primary try block whenever possible whereas the shadow node tries to execute the alternate try block*. Therefore, primary node X uses try block A as the first try block initially, whereas shadow node Y uses try block B as the initial first try block. Until a fault is detected, both nodes receive the same

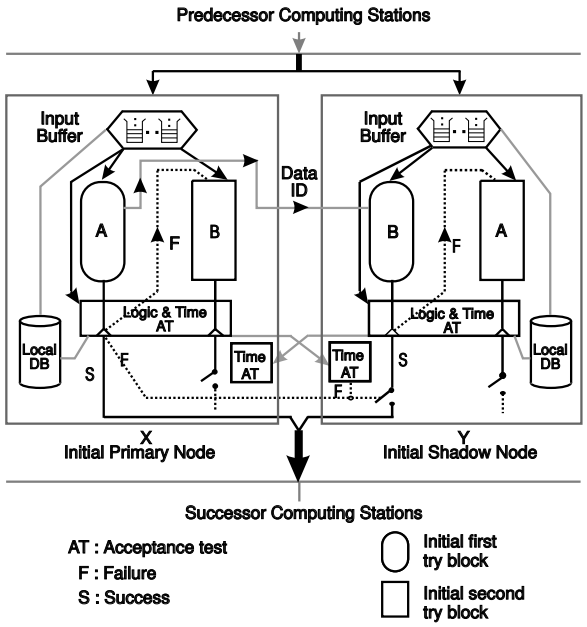

Figure 8.4 A DRB computing station

input data, process the data by use of two different try blocks, and check the results by use of the acceptance test concurrently.

If the primary node fails and the shadow node passes its own acceptance test, the shadow immediately delivers its processing results to the successor computing stations. The two nodes then exchange their roles, i.e., the shadow assumes the primary's role. In case the shadow node fails, the primary node is not disturbed. Whichever node fails, the failed node attempts to become an operational shadow node without disturbing the (new) primary node; it attempts to roll back and retry with its second try block to bring its application computation state including local database up-to-date.

8.3.3 The Types of Faults Covered

In order to analyze the types of faults covered by the DRB scheme, it is useful to view a DRB station as consisting of three types of components : processing nodes (including a recovery block running), communication network, and processing-node-to-network links. In general, each component may exhibit two types of fault symptoms observable outside the component :

(1) *omission failures* (some expected outputs are never produced) of which special cases include continuous omission failures exhibited by crashed nodes and
(2) *faulty value output*.

The probability of a faulty value output being caused by the communication network or a node-network link can be reduced to a negligible level by incorporating relatively inexpensive mechanisms such as an error-correcting code scheme. A self-checking component structure is adopted to reduce the probability of a faulty value being output from the component. Internal

audit mechanisms make a component to exhibit omission failures instead of outputting faulty values. An acceptance test is used in the DRB scheme as such an internal audit mechanism. Also, the damaging effects of omission failures of the communication network or a node-network link can be mitigated by the supplementary capabilities embedded in the sender and receiver computing components for detection of such communication failures and subsequent retry.

Therefore, the DRB scheme is capable of real-time recovery from the omission failures of processing nodes and preventing faulty value output actions of processing nodes to the extent determined by the detection coverage of the acceptance test mechanism. The tolerated omission failures of a node in a DRB station include those caused by

(1) a fault in the internal hardware of a DRB station,
(2) a design defect in the operating system running on internal processing nodes of a DRB station, or
(3) a design defect in some application software modules used within a DRB station.

8.3.4 Strengths and Limitations

The DRB scheme has the following major useful characteristics:

a) Forward recovery can be accomplished in the same manner regardless of whether a node fails due to hardware faults or software faults;
b) The recovery time is minimal since maximum concurrency is exploited between the primary and the shadow nodes;
c) The increase in the processing turnaround time is minimal because the primary node does not wait for any status message from the shadow node;
d) The cost-effectiveness and the flexibility are high because

(d1) a DRB computing station can operate with just two try blocks and two processing nodes and
(d2) the two try blocks are not required to produce identical results and the second try block need not be as sophisticated as the first try block.

On the other hand, the DRB scheme imposes some restrictions on the use of the recovery block scheme. A recovery block to be used in the DRB scheme should be two-phase structured. That is, the computation segment encapsulated within a recovery block should consist of one input acquisition phase and one output phase. During the input phase, the recovery block must not involve any output step (i.e., sending computation results to the outside) while it may involve multiple input steps. Similarly, during the output phase, the recovery block may involve multiple output steps but not a single input step. This restriction is essential to prevent interdependency among different DRB stations for recovery from being formed.

More seriously, we can not even predict how effective the use of acceptance test routines and alternate algorithm implementations will be in achieving SFT. We simply do not understand enough about the nature of software faults in real-time DCS's. They are simply the only kinds of efforts of a general and systematic nature which a system designer can make with the hope of realizing SFT. In other words, use of software redundancy is the only systematic conceivable approach toward the goal of SFT and the acceptance test routines and alternate algorithm implementations are the two most fundamental types of software redundancy. Also, the recovery block structure is the most flexible and yet easily understandable

and traceable structure for incorporation of software redundancy among all known structures. Therefore the underlying philosophy of the DRB scheme behind its adoption of the recovery block scheme as a component is simply to provide the most flexible structure which makes it easy for system designers to insert software redundancy into. Only future experimental studies will provide some indications of how effective system designers' redundant design efforts can be in achieving real-time SFT; this is not to imply that past experimental efforts did not produce some encouraging indicators but rather to point out that they have not produced cases sufficiently convincing to the general public.

8.3.5 Major Design Parameters

Three basic design parameters that must be chosen carefully to obtain a cost-effective DRB station, and that may be impacted by the types of communication architectures used, are the following.

(1) Mechanisms for ensuring input data consistency

Suppose each of the two fault-free partner nodes in a DRB station picks a new data item for the same task execution cycle. If these two data items have the same ID, then the two nodes are said to be preserving input data consistency. Complications can arise if the links between some nodes and the communication network are not reliable; certain data messages may arrive at one partner node but not at the other partner node. In general, it is necessary to take actions that explicitly ensure input data consistency.

(2) Mechanisms for sharing acceptance test results

The shadow node in a DRB station needs to learn the acceptance test result of its primary partner node with an acceptable delay. On the other hand, it is not essential in principle for the primary node to know the acceptance test result of the shadow node because as long as the primary node does not fail, it alone can satisfactorily meet the application requirements.

(3) Mechanisms for reliable communication of result data messages

Successful delivery of the result data message by the primary node to the successor computing station(s) must be confirmed by both partner nodes in the producer DRB station. In case of a failure, the primary node must learn it and then either make a retry for delivery or give up and become a new shadow node. (This means that the inability of a primary node to successfully deliver its computation result to at least one node in each successor computing station is treated as a failure of the primary node in performing a processing cycle. Upon such a failure the node attempts to become a new shadow node with the assumption that its partner node will detect the failure and become a new primary node.) The shadow node must also learn it so that it can decide whether or not to deliver its own result data message. This means that delivery of a result data message by the primary node must be followed by a reply with an acknowledgment message(s) by the successor computing station(s).

8.4 IMPLEMENTATION TECHNIQUES

8.4.1 Recursive Shadowing

The basic scheme described in Section 8.3 can easily be extended for use in configuring a DRB station consisting of more than two processing nodes and more than two try blocks. One of the most natural approaches is the *recursive shadowing* approach which is to treat the third node as a shadow node for the team of the first two nodes as depicted in Figure 8.5 [Kim91].

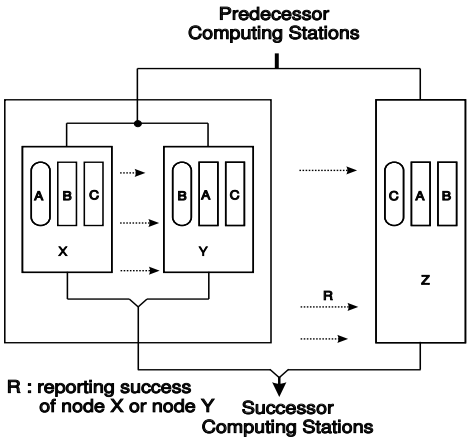

Figure 8.5 Recursive shadowing in the DRB scheme (adapted from [Kim91])

Node Z in the figure will normally use try block C as its primary try block and deliver its results only when both X and Y fail to produce acceptable results in time. Nodes X and Y behave like a single functional node with respect to interfacing with their shadow node Z. They must share responsibilities for providing their status information to node Z at various points as well as responsibilities for understanding the "useful/useless shadow" status of node Z. If node X or Y crashes, then it can be replaced by node Z and thus the computing station can start functioning as an ordinary two-node DRB station. Similarly, crash of node Z will result in the computing station functioning as an ordinary two-node DRB station. If both X and Y fail at their acceptance tests but are alive, then node Z becomes the new primary node and one of the two failed nodes (X and Y) should become the new secondary node (a shadow for node Z) and the other should become the third node (a shadow for the team of Z and the secondary node).

In an n-node DRB station, the n-th node functions as a shadow for the team of the first $n-1$ nodes. In the case of configuring a DRB station with two try blocks and three processing nodes, then the station structure will be essentially the same as that in Figure 8.5 except for the fact that node Z uses try block A as its first try block. A natural consequence of this recursive shadowing organization is the modest increase in the implementation complexity as the number of nodes used in a DRB station increases.

8.4.2 Virtual DRB (VDRB) Stations

When there are not enough nodes in a DCS to form dedicated DRB stations encapsulating all critical tasks in a given real-time application, an option worth exploring is to use the same node-pair to form multiple *virtual DRB* (VDRB) stations. Each of the VDRB stations hosted on the same node-pair is functionally equivalent to a DRB station when it is in execution using a time slice of the node-pair. Figure 8.6 illustrates a case of structuring VDRB stations.

An interesting requirement imposed on each node-pair supporting multiple VDRB stations is that the task schedulers on both partner nodes must schedule the executions of *virtual nodes* (i.e., constituent nodes of VDRB stations) such that the executions of *partner virtual nodes* (belonging to the same VDRB station) are maximally overlapped in time.

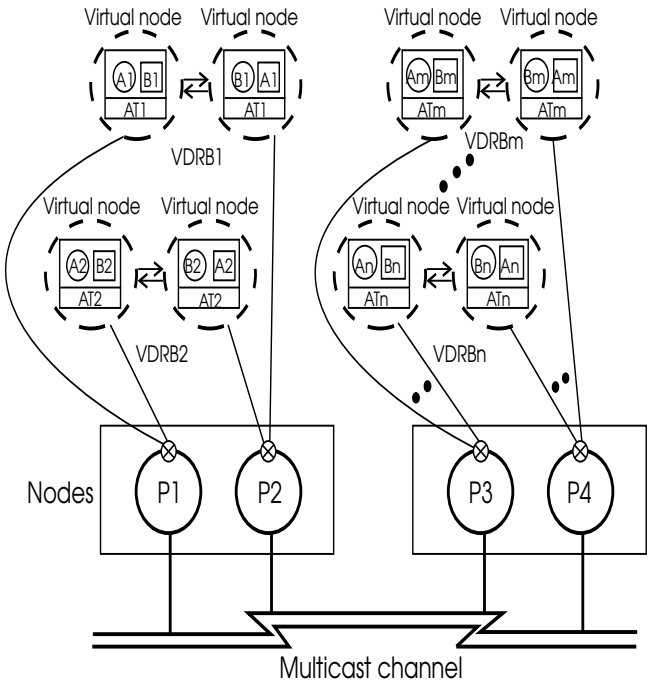

Figure 8.6 Virtual DRB stations

8.4.3 Supervisor Station

In a DCS consisting of multiple DRB stations, it is essential to incorporate a supervisor station [Hec91, Kim93a] or its decentralized equivalent. The supervisor station is in general responsible for the following:

(1) Detection of node crashes,
(2) Detection of misjudgments by the nodes in DRB stations about the status of their partner nodes (due to the faults occurring in communication links), and
(3) Network reconfiguration including task redistribution.

The first demonstration of the basic DRB scheme combined with the supervisor station was made by Hecht *et al* [Hec89, Hec91]. Often it is useful to structure the supervisor station itself in the form of a DRB station. Also, some of these functions, e.g., detection of node crashes, can be decentralized [Kop89, Mor86]. Research is currently active in this area.

The supervisor station can also detect the occurrence of a situation where the shadow node temporarily lags more than one task execution cycle behind the primary node. To do this will require the shadow node to periodically announce its progress.

8.4.4 LAN Based Systems vs. Highly Parallel Multicomputer Network (HPM) Based Systems

Since the DRB scheme is a technique for realizing a "hardened" real-time computing station and since both real-time computer systems based on highly parallel multi-computer networks

(HPM's) and those based on LAN's can also be structured in the natural form of interconnections of real-time computing stations, the application fields of the DRB scheme cover both HPM based applications and LAN based applications. On the other hand, the differences in interconnection structures and mechanisms between the HPM's and the LAN's can have impacts on the approaches for implementation of DRB computing stations.

In LAN based systems, the inter-node communication costs are greater and the costs of providing redundant communication paths are greater. Therefore, the overhead of ensuring input data consistency at the beginning of each task cycle as well as the overhead for status exchange between the partner nodes is much greater in LAN based systems than in HPM based systems. On the other hand, the difference in communication time costs between an one-to-one message communication and a broadcast or multicast of a message in LAN-based system is relatively small whereas it is significant in HPM based systems. Also, in some HPM's, nodes may be connected via shared memory modules. In such HPM based systems, data queues hosted on shared memory modules serve as communication media between DRB stations as well as between partner nodes belonging to the same DRB stations. Therefore, noticeable differences exist between efficient implementations of the primary-shadow cooperating protocols (as well as in the protocols for interaction between the supervisor station and a DRB station) in LAN based systems and those in HPM based systems [Hec91, Kim91, Kim93a, Kim94a].

8.4.5 Acceptance Tests (AT's)

Since the emergence of the recovery block scheme, the quality of the acceptance test has been a subject of continuous debates. Experiences gathered so far on the design costs and the fault detection effectiveness of the acceptance tests are still inadequate. A number of useful principles in deriving cost-effective acceptance tests have been identified [Hec86, Ran94]. In the author's laboratory, we have taken the view that in real-time applications design of effective acceptance tests based on physical laws or apparent boundary conditions existing in application environments is much easier than producing effective acceptance tests in many non-real-time data processing applications. For example, in an aircraft control system, the variables representing acceleration and rate of change of acceleration are not expected to indicate that the pitch attitude has changed faster than at a certain rate, e.g., from level to pointing straight down in 1/50 of a second [Hec80].

As mentioned before, timing tests are essential parts of the acceptance tests in real-time systems. In addition, some hardware-implemented fault detectors are nowadays cost-effective candidates for incorporation in any sizable real-time computer systems. Such hardware detectors can significantly reduce the burden imposed on the software-implemented acceptance tests. For example, an extension of the DRB scheme under which each of the nodes (primary and shadow) in a DRB station is implemented in the form of a comparing processor-pair has important attractive characteristics . Such an augmented DRB station should exhibit much shorter *detection latency* for most hardware faults (since comparison in a processor-pair occurs every instruction cycle or internal bus cycle) than a DRB station with no such augmentation does. In such an augmented DRB station, only some rare types of hardware faults and software faults will escape the guards set by the comparing processor-pair mechanism and will have to be detected by the software-implemented acceptance test with concomitant larger detection latencies.

8.4.6 Incorporation of Complimentary HFT Techniques

One way to classify fault tolerance techniques is as follows.

(Class A): *Detection by Hardware AND Recovery management by Hardware*.
Examples include the TMR scheme and the PCP scheme.
(Class B): *Detection by Hardware AND Recovery management by Software*.
Examples include a combination of the comparing processor-pair scheme and the forward recovery exception handler scheme, a combination of hardware detectors and the recovery block scheme without the software-implemented acceptance test, and a combination of hardware detectors and the DRB scheme without the software-implemented acceptance test.
(Class C): *Detection by Software (with optional assistance of Hardware) AND Recovery management by Software*.
Examples include the recovery block scheme, the PSP scheme, the DRB scheme, and a combination of the software-implemented internal audit function (or the run-time assertion) approach and the exception handler scheme.

The class-C scheme with hardware detectors include class-B schemes as special cases. The DRB scheme can be used as such a scheme. The DRB scheme can also be combined with a class-A scheme.

8.4.7 Implementation Techniques for the Algorithm Redundancy Part

The approaches for designing alternate try blocks to be used in DRB stations are no different from those used in the recovery block scheme [Hor74, Ran75, Ran94] except that there are two-phase structuring restrictions discussed in Section 8.3.4. In designing alternate try blocks, exploitation of data structure diversity is a useful principle to follow. Also, the principles of developing diverse versions used in the NVP scheme are fully useful in designing alternate try blocks [Avi88].

8.5 EXPERIMENTAL VALIDATIONS OF REAL-TIME RECOVERY

Since the initial formulation of the DRB concept in 1983, several demonstrations of the performance of the scheme in practical application contexts were conducted. For example, several experiments involved application of the DRB scheme to adjacent computing stations in real-time parallel processing multi-computer testbeds. Other experimental applications of the DRB scheme to LAN based systems were also reported and several more advanced application developments are under way in several research organizations. In this section, a few representative examples are briefly reviewed.

8.5.1 Demonstrations of the DRB Scheme with Parallel Processing Multi-computer Testbeds

The experiment carried out by the author and his colleagues initially at the University of South Florida (USF) and later at the University of California, Irvine (UCI) in mid-1980's [Kim89], was aimed for validating primarily the real-time recovery capability and the formulated implementation approach of the DRB scheme and secondarily the ability of the DRB scheme

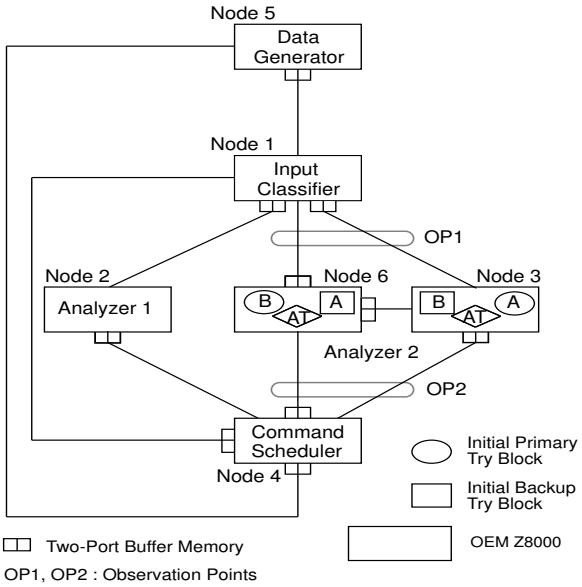

Figure 8.7 The parallel computing network configuration used for experimentation of the DRB scheme (adapted from [Kim89])

to detect and recover from the unforeseen design faults. While the experiment confirmed the real-time recovery capability and the implementation approach but came short of meeting the secondary, much more difficult objective.

The network configuration used is depicted in Figure 8.7. This network facility has been named the Macro-Dataflow Network (MDN). Each node in the MDN is a Z8000-based single-board microcomputer called the OEM-Z8000. The connection medium used between nodes is a two-port buffer memory developed in house and consisting of two independent memory banks of 16K bytes each. Nodes can exchange data through a two-port buffer memory nearly at the rate of local (on-board) memory access. A software nucleus implemented on each node supports concurrent programs consisting of asynchronous processes communicating through monitors, in particular, the programs written in the Extended Concurrent Pascal language [Bri77, Kim84b].

The distributed application program executed on the MDN during the experiment was written in Extended Concurrent Pascal. The distributed functions of this program are indicated in Figure 8.7. The DRB scheme was incorporated into nodes 3 and 6 (performing the Analyzer-2 function). Node 5 (Data generator) simulates a real-time device which generates stimulus data sent to the rest of the network and accepts the response (command). The remaining five data processing nodes contain an *input (i.e., stimulus data) classification process*, various *analysis processes*, constituting the intelligence of the solution algorithm, and a *control command scheduler process* that delivers the network's response to the real-time device. The stimulus data from Node 5 are first handled by the input classification process which distributes inputs to the rest of the network. The command scheduler process honors requests from various analysis processes to schedule commands for the real-time device.

The "travel times" of data sets passing through a computing station were measured to determine the execution overhead caused by the introduction of the DRB scheme into the network.

As a part of facilitating this measurement, "observation points" were established in the network. When a data set arrives at the designated observation point in the network, the node stamps the real-time and saves a copy of the time-stamped data in its local memory. When enough measured data is obtained, the time-stamped data is transferred to another computer system for data analysis. The observation points are usually established at the points where the nodes are ready to send messages to the successor nodes and also at the points where the nodes have received messages.

In this experiment, two observation points were set up in the network. Figure 8.7 shows these points established in the network. Observation point 1 (OP1) is set up where the primary and shadow nodes have taken the data set from the queue buffers connected to the predecessor node. Observation point 2 (OP2) is set up where both nodes are ready to put the result data sets into the queue buffers connected to the successor node.

During experimentation, faults were injected to examine their impacts on system performance. The types of faults studied include: 1) total node failure (simulated by node reset); 2) transient hardware faults simulated by random changes in the contents of certain memory locations; and 3) software faults such as infinite looping, arithmetic overflow, etc. The recovery block incorporated into nodes 3 and 6 in Figure 8.7 was written in Extended Concurrent Pascal and executed on an OEM-Z8000 microcomputer with a clock rate of 4M-Hz.

The DRB execution overhead consists of interprocess communication among nodes and the execution of the acceptance test. The overhead was evaluated by comparing the delay between OP1 and OP2 in the case of using the DRB against the delay in the case without the DRB. The gap between the two delays is the execution time increase due to the incorporation of the DRB. The average execution time increase measured was approximately 30 ms (milliseconds). Moreover, measurements were taken in instances where arithmetic overflows occurred in the primary node and the fast recovery capabilities of the DRB scheme were exercised. We noted that the fault occurrences and subsequent recovery actions did not cause any visible degradation of the system performance. Again, in the absence of fault, the execution time increase is caused mainly by the execution of the acceptance test and the communication of the acceptance test success to the backup node.

Considering the inefficient implementation language (Extended Concurrent Pascal), and the slow processor (4 M-Hz Z8001) used, the amount of execution time increase mentioned above is at least 20 times higher than that expected in the systems built with current off-the-shelf hardware and software tools. For example, use of a processor running at 20 M-Hz will result in speedup by a factor of 5. Use of a more efficient language (C or an assembly language in the extreme case) will result in additional speedup by a factor of about 4.

Figure 8.8 shows the case where the primary node is reset, resulting in the crash of the node. Later an arithmetic overflow occurs in the remaining node. The recovery from the first fault (the crash of the primary node) took about 60 ms. This recovery time is largely a function of the timeout period used in the DRB. When the second fault (the arithmetic overflow) occurred in the remaining node after the crash of the first node, the node had no choice but to roll back and retry with try block B. Therefore, the recovery time was very high, i.e., about 290 ms, as shown in the figure. Again, the recovery time can be easily reduced by a factor of 20 by implementing real application systems with current off-the-shelf tools.

The data discussed above was one indicator that the DRB scheme could be used in many real-time applications with tolerable amounts of time overhead.

Other experiments conducted with different parallel processing machines were reported in [Kim89, Kim91].

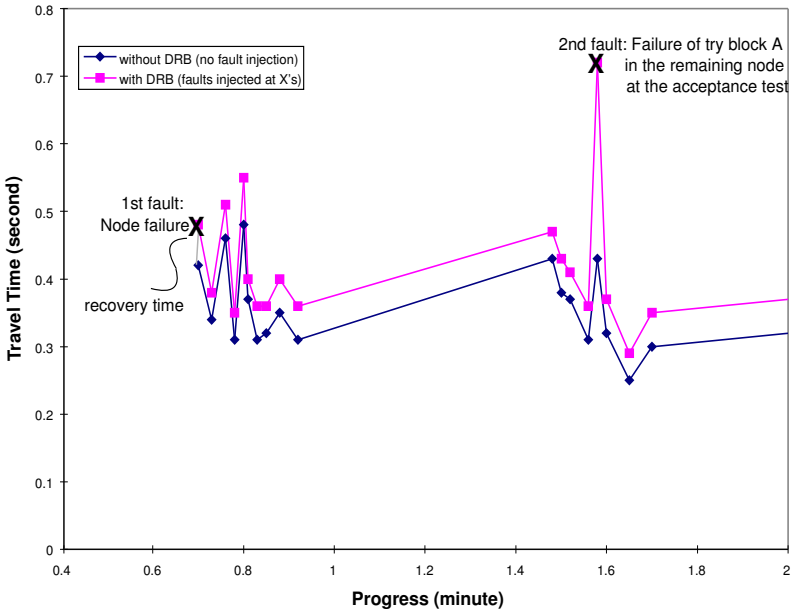

Figure 8.8 Data travel time measured (adapted from [Kim89])

8.5.2 Demonstrations of the DRB Scheme with LAN Based DCS's

Another major validation of the DRB scheme was conducted by a small company located in Los Angeles (SoHaR, Inc). They extended the DRB scheme for use in real-time local area PC networks for nuclear reactor control applications and produced a product prototype [Hec89, Hec91]. This was also the first demonstration of the basic DRB scheme combined with the supervisor station scheme. Figure 8.9 depicts a high level view of the product prototype developed.

This validation work confirmed the logical soundness of the implementation model of the DRB scheme augmented with the supervisor station scheme as well as the real-time recovery capability. However, it also came short of validating the the ability of the DRB scheme to detect and recover from the unforeseen design faults.

Quite a few other experimental applications of the DRB scheme to LAN based real-time DCS's have been performed [Arm91, Fra91, Kim94a] and some more are currently underway.

8.6 ISSUES REMAINING FOR FUTURE RESEARCH

8.6.1 Validation of Software Fault Detection and Recovery

While the potential of the DRB scheme, the recovery block scheme, and the NVP scheme for detecting and recovering from software faults has been widely recognized in the research com-

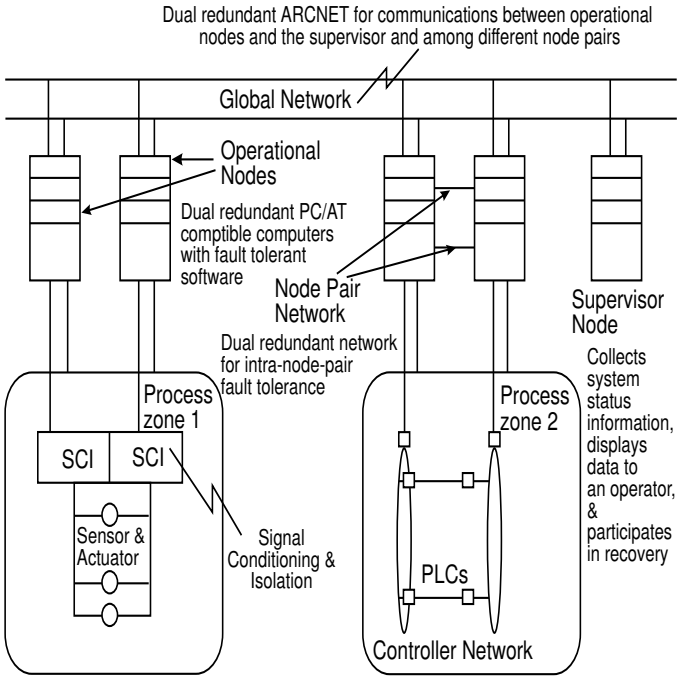

Figure 8.9 The DRB-based fault-tolerant LAN architecture developed by SoHaR, Inc.

munity, the mission of demonstrating this capability in convincing application contexts still remains to be accomplished. Successful accomplishment of this mission will require long-term persistent research effort since use of artificially injected faults will be an invalid approach. The application system used also needs to be of considerable complexity since otherwise the convincing cases of software fault occurrences are not likely to be encountered [Hua93]. In parallel with such experimental efforts, the work on analytical modeling and evaluation of the potential has been under way (e.g., [Dug94]) and produced some useful insights into the nature of software fault tolerance .

8.6.2 Integration with Real-time Network Configuration Management (NCM)

As mentioned in Section 8.4.3, some functions, if not all, of the supervisor station can be decentralized. The functions of the supervisor stations are all related to real-time network configuration management (NCM). This area is not mature yet. However, rapid progresses are expected in the next several years. Cost-effective integration of the DRB scheme and the emerging real-time NCM techniques is an important issue for future research in design of fault-tolerant real-time DCS's.

8.6.3 Adaptation of the DRB Scheme to Object Based Systems

Object-oriented structuring is now a firmly established principle not only in generic software engineering but also in development of real-time computer systems [Bih89, Kop90, Kim94b].

Efficient adaptation of the DRB scheme to object-oriented DCS structures is thus a meaningful subject for future research.

8.6.4 Handling of Faults Crossing DRB Station Boundaries

Since it is not practical to assume that the acceptance tests will have perfect fault detection coverage in all applications, it will be meaningful in some applications to consider what supplementary mechanisms can be provided to handle such faults crossing DRB station boundaries. Some promising approaches have been proposed [Bes81, And83, Anc90, Kim93b] but demonstrations of the effectiveness of the approaches have lagged behind and will require considerable amount of efforts in the future. Also, firm establishment of such schemes is required to address the issue of tolerating integration faults mentioned in Section 8.2.2 to a greater extent than that to which the DRB scheme addresses.

8.7 CONCLUSIONS

The DRB scheme is a cost-effective approach of basic nature for realizing both hardware fault tolerance (HFT) and Software fault tolerance (SFT) in real-time distributed and/or parallel computer systems. A reasonably rich set of implementation techniques have been established and thus the DRB scheme is a practical technology available for use in a broad range of real-time safety critical applications. However, further research, especially on the issues discussed in Section 8.6, is needed to realize the full potential of the DRB scheme.

ACKNOWLEDGEMENTS

The work reported here was supported in part by US Navy, NSWC Dahlgren Division under Contract No. N60921-92-C-0204, in part by the University of California MICRO Program under Grant No. 93-080, and in part by a grant from Hitachi Co., Ltd.

REFERENCES

[Anc90] M. Ancona, G. Dodero, V. Gianuzzi, A. Clematis, and E.B. Fernandez. A system architecture for fault tolerance in concurrent software. *IEEE Computer*, 23–32, October 1990.

[And83] T. Anderson and J.C. Knight. A framework for software fault tolerance in real-time system. *IEEE Transactions on Software Engineering,* 355–364, May 1983.

[Arm91] L.T. Armstrong and T.F. Lawrence. Adaptive fault tolerance. In *Proc. 1991 NSWC Systems Design Synthesis Technology Workshop*, Silver Spring, September 1991.

[Avi71] A. Avižienis, G. Gilley, G.C. Mathur, D. Rennels, J.A. Rohr, and D.K. Rubin. The STAR (self testing and repairing) computer: an investigation of the theory and practice of fault-tolerant computer design. *IEEE Transactions on Computers*, C-20(11):1312–1321, November 1971.

[Avi75] A. Avižienis. Fault tolerance and fault intolerance: complementary approaches to reliable computing. In *Proc. 1975 International Conference on Reliable Software,* pages 458–464, Los Angeles, April 1975.

[Avi85] A. Avižienis. The N-version approach to fault-tolerant software. *IEEE Transactions on Software Engineering*, SE-11(12):1491–1501, December 1985.

[Avi88] A. Avižienis, M.R. Lyu, and W. Schuetz. In search of effective diversity: a six-language

study of fault-tolerant flight control software. In *Proc. 18th International Symposium on Fault-Tolerant Computing*, pages 15–22, Tokyo. IEEE Computer Society Press.

[Bes81] E. Best and B. Randell. A formal model of atomicity in asynchronous systems. *Acta Informatica*, Springer-Verlag, 16:93–124, 1981.

[Bih89] T. Bihari, P. Gopinath, and K. Schwan. Object-oriented design of real-time software. In *Proc. 10th Real-Time Systems Symposium*, pages 194–201, 1989. IEEE Computer Society Press.

[Bri77] P. Brinch Hansen. *The Architecture of Concurrent Programs*. Prentice Hall, 1977.

[Car85] W.C. Carter. Hardware fault tolerance. Chapter 2 in T. Anderson, editor, volume 1 of *Resilient Computing Systems*, pages 11–63, 1985. Wiley-Interscience.

[Dug94] J.B. Dugan and M.R. Lyu. System-level reliability and sensitivity analyses for three fault-tolerant system architectures. In *Proc. 4th International Working Conference on Dependable Computing for Critical Applications*, IFIP 10.4 Working Group, pages 295–307, January 1994.

[Fra91] J.S. Fraga, V. Rodrigues, and E.S. Silva. A language approach to implementation of the distributed recovery block schemes. In *Proc. 13th. CBC Conference on Computer Sciences,* Gramado, Brazil, August 1991.

[Hec80] H. Hecht. Issues in fault-tolerant software for real-time control applications. In *Proc. 4th International Computer Software and Applications Conference (COMPSAC)*, pages 603–607, October 1980. IEEE Computer Society Press,

[Hec86] H. Hecht and M. Hecht. Fault-tolerant software. Chapter 10 in D.K. Pradhan, editor, volume 2 of *Fault-Tolerant Computing: Theory and Techniques*. Prentice Hall, 1986.

[Hec89] M. Hecht, J. Agron, and S. Hochhauser. A distributed fault tolerant architecture for nuclear reactor control and safety functions. In *Proc. 1989 Real-Time Systems Symposium*, pages 214–221, December 1989. IEEE Computer Society Press.

[Hec91] M. Hecht *et al*. A distributed fault tolerant architecture for nuclear reactor and other critical process control applications. In *Proc. 21st International Symposium on Fault-Tolerant Computing*, Montreal, June 1991, pages 462–469. IEEE Computer Society Press.

[Hor74] J.J. Horning, H.C. Lauer, P.M. Melliar-Smith, and B. Randell. A program structure for error detection and recovery. *Lecture Notes in Computer Science*, 16:171–187, Springer-Verlag, New York, NY, 1974.

[Hua93] Y. Huang and C.M.R. Kintala. Software implemented fault tolerance: technologies and experience. In *Proc. 21st International Symposium on Fault-Tolerant Computing*, pages 2–9, June 1993. IEEE Computer Society Press.

[Kim84a] K.H. Kim. Distributed execution of recovery blocks: an approach to uniform treatment of hardware and software faults. In *Proc. 4th International Conference on Distributed Computing Systems*, pages 526–532, May 1984. IEEE Computer Society Press.

[Kim84b] K.H. Kim. Evolution of a virtual machine supporting fault-tolerant distributed processes at a research laboratory. In *Proc. 1st International Conference on Data Engineering*, pages 620–628, Los Angeles, April 1984. IEEE Computer Society Press.

[Kim89] K.H. Kim and H.O. Welch. Distributed execution of recovery blocks: an approach to uniform treatment of hardware and software faults in real-time applications. *IEEE Transactions Computers*, pages 626–636, May 1989.

[Kim91] K.H. Kim and B.J. Min. Approaches to implementation of multiple DRB stations in tightly coupled computer networks and an experimental validation. In *Proc. 15th International Computer Software and Applications Conference (COMPSAC 91)*, pages 550–557, Tokyo, September 1991. IEEE Computer Society Press.

[Kim93a] K.H. Kim. Structuring DRB computing stations in highly decentralized Systems. In *Proc. International Symposium on Autonomous Decentralized Systems*, pages 305–314, Kawasaki, March 1993. IEEE Computer Society Press.

[Kim93b] K.H. Kim. Design of loosely coupled processes capable of time-bounded cooperative recovery: the PTC/SL scheme. *Computer Communications*, 16(5):305–316, May 1993.

[Kim94a] K.H. Kim, L.F. Bacellar, K. Masui, K. Mori and R. Yoshizawa. Modular implementation model for real-time fault-tolerant LAN systems based on the DRB scheme with a configuration supervisor. In *Computer System Science & Engineering*, 9(2):75–82, April 1994.

[Kim94b] K.H. Kim and H. Kopetz. A real-time object model RTO.k and an experimental investigation of its potentials. In *Proc. 18th International Computer Software and Applications Conference*

(COMPSAC 94), Taipei, IEEE Computer Society Press, November 1994.

[Kop89] H. Kopetz, G. Grunsteidl, and J. Reisinger. Fault-tolerant membership service in a synchronous distributed real-time system. In *Proc. International Working Conference on Dependable Computing for Critical Applications*, IFIP 10.4 Working Group, pages 167–174, Santa Barbara, August 1989.

[Kop90] H. Kopetz and K.H. Kim. Temporal uncertainties in interaction among real-time objects. In *Proc. 9th Symposium on Reliable Distributed Systems,* pages 165–174, Huntsville, October 1990. IEEE Computer Society Press.

[Lee78] P.A. Lee. A reconsideration of the recovery block scheme. *Computer Journal*, 21(4):306–310, November 1978.

[Mor86] K. Mori. Autonomous decentralized software structure and its application. In *Proc. Fall Joint Computer Conference,* pages 1056–1063, Dallas, November 1986.

[Ran75] B. Randell. System structure for software fault tolerance. *IEEE Transactions on Software Engineering*, pages 220–232, June 1975.

[Ran94] B. Randell and J. Xu. The evolution of the recovery block concept. *Chapter 1 in this book*.

[Toy87] W.N. Toy. Fault-tolerant computing. A Chapter in *Advances in Computers.* volume 26, pages 201–279. Academic Press, 1987.

[Wil85] D. Wilson. The STRATUS computer system. Chapter 12 in T. Anderson, editor, volume 1 of *Resilient Computing Systems* pages 45–67. Wiley-Interscience, 1985.

[Yau75] S.S. Yau and R.C. Cheung. Design of self-checking software. In *Proc. International Conference on Reliable Software*, pages 450–457, 1975.