Evaluation of Fault Tolerance Latency from Real-Time Application's Perspectives

Hagbae Kim, *Member*, *IEEE*, and Kang G. Shin, *Fellow*, *IEEE*

Abstract—Information on *Fault Tolerance Latency* (FTL), which is defined as the total time required by all sequential steps taken to recover from an error, is important to the design and evaluation of fault-tolerant computers used in safety-critical real-time control systems with deadline information. In this paper, we evaluate FTL in terms of several random and deterministic variables accounting for fault behaviors and/or the capability and performance of error-handling mechanisms, while considering various fault tolerance mechanisms based on the trade-off between temporal and spatial redundancy, and use the evaluated FTL to check if an error-handling policy can meet the Control System Deadline (CSD) for a given real-time application.

Index Terms—Fault tolerance latency (FTL), temporal/spatial and static/dynamic redundancy, error-handling, Control System Deadline (CSD), dynamic failure.

1 Introduction

Control computers used for safety-critical applications like flight and process controls must be equipped with appropriate fault tolerance mechanisms which guarantee the safe operation of the system even in the presence of component failures. Fault tolerance is achieved via temporal and/or spatial redundancy and, hence, its design methodologies are characterized by the trade-off between these two types of redundancy. Most design criteria in non-real-time systems deal with optimization of spatial redundancy, whereas, in real-time systems, time is so valuable as to motivate trading space for time. A fault tolerance policy should be selected and implemented to recover from errors within a certain time limit, the *Control System Deadline* (CSD) [14], [15], defined as the maximum time the underlying controlled system can stay in its admissible state space without receiving correct services from its digital controller computer. *Fault Tolerance Latency* (FTL)—which is defined as the total time spent on such sequential error-handling stages as error detection, fault location, system reconfiguration, and recovery of the contaminated application program [13]—is, therefore, essential to the design and and evaluation of fault-tolerant computers.

There have been several attempts to evaluate quantities related to FTL. Some researchers treated the recovery process as one event, lumping all the sequential stages, such as fault detection/isolation, system reconfiguration, and resumption of the contaminated program, to 1) derive analytically a simple expression for the recovery-time distribution or 2) evaluate the general models of error-handling with simulations. In [5] and [12], a truncated normal distribution (with a displaced exponential function) capturing "general" short periods of normal recovery, as well as "special" long durations of rare abnormal recovery, according to experimental data and instantaneous probabilities, thus characterizing only the effectiveness of error-handling mechanisms, were used to model recovery procedures.

The authors of [4], [10] proposed experimental and statistical methods (i.e., sampling and parameter-estimation) for characterizing the times of fault detection, system reconfiguration, and computation recovery based on hardware fault injections in the Fault-Tolerant Multiple Processor (FTMP), rather than modeling the total recovery time. In [1], [13], the recovery times were defined as FTL and estimated for a pooled-spare and N-modular redundant systems by describing the effects of various fault tolerance features. However, the results were given in a specific application context, using spatial redundancy only, and assuming that the time required for each stage of fault/error recovery is approximated to be in a deterministic range.

In this paper, we analytically model FTL, covering the various recovery mechanisms (e.g., retry, rollback, restart) based on the trade-off between temporal and spatial redundancy. We first study the times required for all individual error-handling stages. Especially, we describe in detail the time required for system reconfiguration, which is generally the most time-consuming error-handling stage. Then, we tailor these results properly to represent various error-handling scenarios or policies. (A *policy/scenario* is composed of a set of sequential error-handling stages.)

Our analysis is based on the assumption that the latencies of error-handling stages are stochastic, depending upon the random characteristics of error detection and fault behaviors; the active duration of a fault affects significantly the success/failure of a temporal-redundancy method (i.e., instruction retry or program rollback). Although all sequential actions are intrinsically not independent, we assume that the time required for each stage is independent of that of the others because the random aspects of the

• *H. Kim is with the Department of Electrical and Computer Engineering, Yonsei University, Seoul, Korea. E-mail: hbkim@bubble.yonsei.ac.kr.*
• *K.G. Shin is with the Real-Time Computing Laboratory, Department of Electrical Engineering and Computer Science, The University of Michigan, Ann Abor, MI 48109-2122. E-mail: kgshin@eecs.umich.edu.*

Manuscript accepted 15 Dec. 1999.
For information on obtaining reprints of this article, please send e-mail to: tc@computer.org, and reference IEEECS Log Number 111106.

latencies of some (individual) stages result from independently behaving faults (occurrence and duration) and detection schemes, while the other stages have deterministic latencies. This assumption allows us to derive the probability density function (*pdf*) of FTL by convolving the *pdf*s of the random variables representing the individual latencies.

Our results focus on a sequence of error-handling stages and can also be used for other well-developed reliability or dependability models [2], [3].

In Section 2, we describe general fault tolerance features by classifying fault tolerance mechanisms and considering the trade-off between temporal and spatial redundancy because we focus on the latency of general fault tolerance mechanisms applying both types of redundancy. Section 3 examines the effects of individual error-handling stages, from the occurrence of an error to its recovery on FTL, and combines these results to evaluate the FTL of a general error-handling policy covering various stages making a trade-off between temporal and spatial redundancy. There, we focus on system reconfiguration, a predominant contributor to FTL, by considering the effects of such parameters as task size, CPU speed, and the bandwidth rate of bus/interconnection network. In Section 4, we argue for the importance of FTL information to the design and validation of fault-tolerant control computers. We present there a contrived example that selects an appropriate error-handling policy based on the FTL information. The paper concludes with Section 5.

2 Generic Fault Tolerance Features

A fault is defined as the malfunctioning/damaged part of a system occurring internally due to physical defects during manufacture or due to component aging, or as environmental interferences or disruptions occurring externally. An error is the manifestation of one or more faults. Computer system failures occur due to manifested faults (i.e., errors) or deviations from the program-specified behaviors. It is desirable to select an appropriate policy so as to preserve the program-specified functions even in the presence of faults/errors.

Fault tolerance is achieved via spatial and/or temporal redundancy: that is, systematic and balanced selection of protective redundancy among hardware (additional components), software (special programs), and time (repetition of operations). Thus, design methodologies for fault-tolerant computers are characterized by the trade-off between spatial and temporal redundancy. Using these two types of redundancy, a fault-tolerant computer must go through as many as 10 stages in response to the detection of an error, including fault location, fault confinement, fault masking, retry, rollback, diagnosis, recovery, restart, repair, and reintegration. Design of a fault-tolerant computer involves selection of an appropriate error-handling policy that combines some or all of these stages.

Spatial redundancy is classified into two categories: static and dynamic. Static redundancy, also known as masking redundancy, can mask erroneous results without causing any delay as long as a majority of participant modules (processors or other H/W components) are nonfaulty. However, the associated spatial cost is high, e.g., three (four) modules are required to mask a non-Byzantine (Byzantine) fault in a TMR (QMR) system. The time overhead of managing redundant modules—for example, voting and synchronization—is also considerable for static redundancy. Dynamic redundancy is implemented with two sequential actions: fault/error detection and recovery of the contaminated computation. In distributed systems, upon detection of an error, it is necessary to locate the faulty module before replacing it with a nonfaulty one. Although this approach may be more flexible and less expensive than static redundancy, its cost may still be high due to the possibility of hastily eliminating modules with transient faults[1] and it may also increase the recovery time because of its dependence on time-consuming error-handling stages such as fault diagnosis, system reconfiguration, and resumption of execution.

To overcome the above disadvantages, temporal redundancy can be used by simply repeating or acknowledging machine operations at various levels: micro-operation/single instruction (retry), program segment (rollback), or the entire program (restart). In fact, one of these recovery schemes is also needed to resume program execution in case of dynamic redundancy. This temporal-redundancy method requires high coverage of fault/error detection so as to invoke the recovery actions quickly. (The same coverage is also required in case of dynamic redundancy.) The main advantages of using temporal redundancy are not only its low "spatial" cost but also its low recovery time for transient faults. However, the time spent for this method would have been wasted in case of permanent or long-lasting transient faults, which may increase the probability of dynamic failure.

The relations between the temporal and spatial redundancy required (and the associated redundancy-management overhead) are shown in Fig. 1 for several fault tolerance mechanisms. In case of time-critical applications, an appropriate fault tolerance mechanism can be found from the top left of Fig. 1, i.e., paying a small amount of temporal redundancy at the cost of spatial redundancy like N-modular redundancy. When the timing constraint imposed by the underlying control application is not tight, we can save the cost of spatial redundancy by increasing temporal redundancy (i.e., a larger amount of time for retry, rollback, or restart recovery), which enhances the system's ability in recovering from more transient faults before the faulty modules are replaced. Increasing temporal redundancy, however, increases the possibility of missing task deadlines or dynamic failure.

3 Evaluation of Fault Tolerance Latency

The recovery process that begins from the occurrence of an error consists of several stages, some of which depend on each other, and FTL is defined as the time spent for the entire recovery process. Thus, all the stages necessary to

1. Note that more than 90 percent of faults are known to be non-permanent; as few as 2 percent of field failures are caused by permanent faults [11].

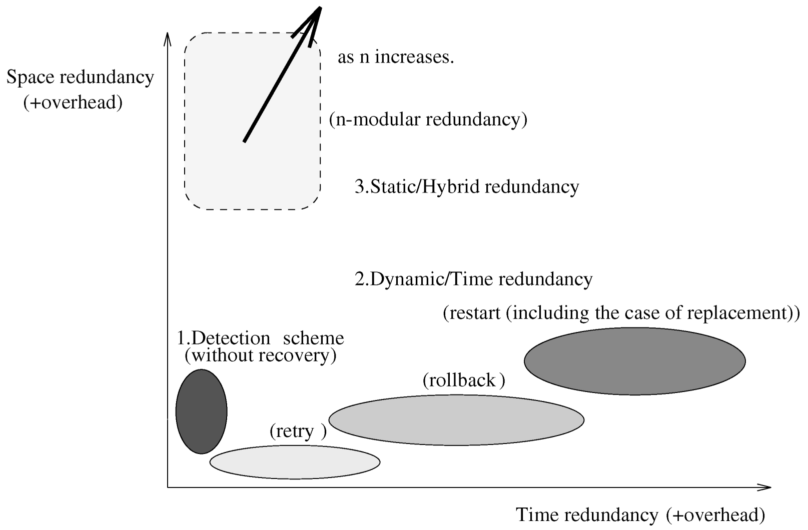
Space redundancy
(+overhead)
as n increases.
(n-modular redundancy)
3.Static/Hybrid redundancy
2.Dynamic/Time redundancy
(restart (including the case of replacement))
1.Detection scheme
(without recovery)
(rollback)
(retry)
Time redundancy (+overhead)

Fig. 1. Trade-off between temporal and spatial redundancy for various fault tolerance mechanisms.

handle faults/errors upon occurrence of an error should be studied and their effects on FTL analyzed.

In a specific application context, the recovery times were estimated in [1], [13] by decomposing the fault-recovery process into stages and analyzing the effects of various fault tolerance features on FTL. A similar approach is used here for the evaluation of FTL, but for more general fault tolerance strategies (than those in [1], [13]) based on temporal redundancy.

Fig. 2 depicts various scenarios from the occurrence of an error to its recovery, covering static and dynamic redundancy, temporal-redundancy methods, and combinations thereof. Each *path* represents one error-handling scenario, which may occur as a result of selecting an error-handling policy corresponding to the path and the success/fail result of the selected method (depending upon the fault behaviors when a temporal-redundancy method is applied). For example, an unsuccessful retry (after the retry period expires) implies another error detection, which may trigger rollback or restart (with or without switching out the faulty module). As shown in Fig. 2, error-handing processes are classified according to the adopted fault/error detection and recovery mechanisms.

3.1 Individual Error-Handling Stages

First, we divide the error-handing process into several stages, as shown in Fig. 2, and evaluate the time spent on each individual stage. Upon detection of an error, if a temporal-redundancy method, such as retry or rollback, is applied as a primary means of recovery, we may need a secondary recovery method composed of such stages as fault diagnosis, system reconfiguration, and resumption of execution, depending on the recovery results of the temporal-redundancy method used. Whether a temporal-redundancy method is successful or not depends on the adopted policies and the underlying fault behaviors. Thus, we must represent the effects of certain stages on FTL probabilistically due to random fault/error behaviors. Now, we describe the features of individual error-handling stages using spatial and temporal redundancy and examine their effects of the time required for each stage on FTL.

Error Detection Process—Since FTL begins with the occurrence of an error, we are mainly interested in the *error latency*, defined as the time interval from the error generation to the error detection [16], which depends on the active duration of a fault and the underlying detection mechanism. Error-detection mechanisms are generally classified into 1) signal-level detection mechanisms, 2) function-level detection mechanisms, and 3) periodic diagnostics.[2] Let t_{el} and f_{el} be the error latency and its probability density function (pdf), respectively. Several well-known pdfs, such as Weibull, Gamma, and log-normal distributions, were considered in [4] to model $f_{el}(t)$. Let F_{el} denote the Probability Distribution Function (PDF) of t_{el}, then $F_{el}(t) = \int_0^t f_{el}(x)dx$. Let Δt_i be the mean execution time of one instruction, then $F_{el}(\Delta t_i) \approx 1$ for a high-coverage signal-level detection mechanism. The function-level error detection latency depends on the detection mechanism used and the executing task and is commonly larger than the signal-level detection latency.

Fault Masking—This method filters out the effects of faulty modules as long as the number of faulty modules is not larger than $\frac{N-1}{2}$ for N-modular redundancy in a form of static (or hybrid) redundancy. Although the time required for this type of recovery is almost zero, the method induces the time overhead of redundancy management, such as synchronization and voting/interactive consistency techniques, even in the absence of faults, which increases with the degree of redundancy [9].

Fault Diagnosis—In distributed systems, it may be necessary to locate the faulty module and/or to determine certain fault characteristics upon detection of an error by a function-level detection mechanism. Let t_d and p_d be the time spent for fault diagnosis and the probability of locating the faulty module (i.e., diagnostic coverage). The coverage p_d increases with t_d and greatly affects the results of the subsequent recovery, hence, FTL. Note that the time, t_d, taken for diagnosis is likely to be deterministic, because diagnostics are usually programmed a priori.

2. Not covered in this paper.

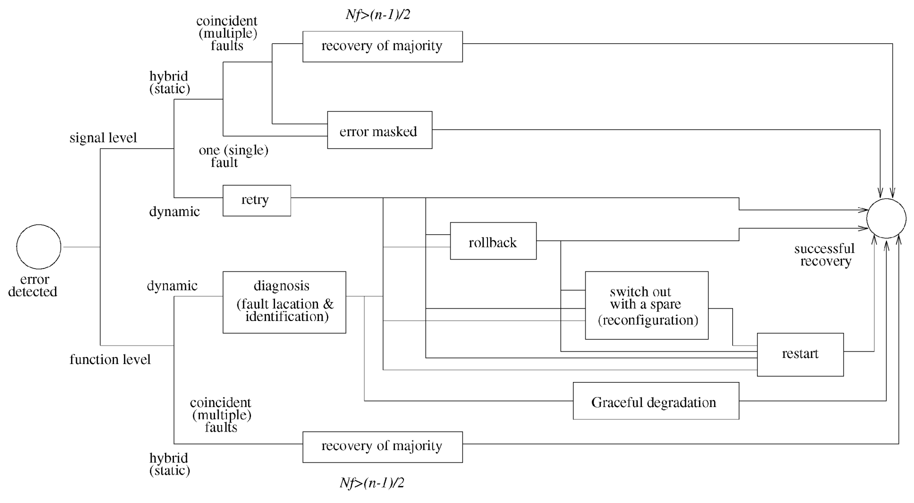

Fig. 2. All possible error-handling scenarios.

System Reconfiguration—Reconfiguration, which is usually the most time-consuming error-handling stage, is the only means to remove a permanent fault. When a fault is located and identified as permanent, the faulty module must be replaced with a spare module or switched off, thus allowing for graceful degradation. This process is necessary for both dynamic and hybrid redundancy. Specific hardware, like the Configuration Control Unit (CCU) in FTMP [6], may be dedicated to system reconfiguration. This process (of using cold spares) generally consists of

1. switching power and bus connections,
2. running built-in-test (BIT) on the selected spare module,
3. loading programs and data,
4. initializing the software.

When warm spares are used, Steps 1 and 2 are not needed. The time taken for this process is also likely to be deterministic, which depends upon program size, system throughput, processor speed, and bus bandwidth. Let t_r be the time spent for system reconfiguration. We assume that t_r lies within a deterministic interval, $t_{r1} \leq t_r \leq t_{r2}$, where t_{r1} and t_{r2} are determined by the type of reconfiguration and several other factors described above. In fact, these parameters were determined analytically in [8], as well as experimentally in [1], [13].

Graceful Degradation—In on-line repair, the faulty component may be replaced immediately by a backup spare in a procedure equivalent to system reconfiguration or operation may continue without the faulty component's contributions, as is the case with fault masking or graceful degradation. Graceful-degradation techniques always use redundant hardware as part of the system's normal resources, preferring operation with degraded performance to no operation at all. Since normal operation does not stop upon detection of an error, the time required for this type of "recovery" is zero as in the case of fault masking.

Retry—This is the simplest recovery method using temporal redundancy, which repeats the execution of a micro-operation or instruction. To be effective, this method requires immediate error detection, i.e., almost perfect coverage of a signal-level detection mechanism yielding t_{el} smaller than Δt_i. The *retry period*, defined as a continuous-time interval or the number of reexecutions, is the maximum allowable time for retry. Let t_{rp}, t_a, and F_a be the retry period, the active duration of a fault, and the PDF of t_a, respectively. The result of a retry depends on t_{rp} and t_a. When a retry is successful, the time it has taken is equal to the fault duration, t_a, which is certainly smaller than t_{rp}. However, it is equal to t_{rp} when the retry becomes unsuccessful, and an alternative recovery method will be followed, thus increasing FTL.

Rollback—is needed when $t_{el} > \Delta t_i$. When an error is detected by a signal- or function- level detection mechanism, it rolls back past the contaminated part of a program (following a system reconfiguration in case of dynamic redundancy). It may be invoked as another step of recovery after an unsuccessful retry. Let Δt_c and N_c be the intercheckpoint interval and the maximum number of checkpoints necessary for rollback recovery, respectively. The time taken for the rollback process depends on t_{el}, Δt_c, the number of checkpoints maintained, and the way checkpoints are selected for rollback. For simplicity, we assume that Δt_cs are equidistant. (It is not difficult to extend our results to the case of nonequidistant checkpoints, although the notation will become more complex.) When the rollback is successful, the time taken to restore the contaminated segment of a program is larger than t_{el} but smaller than $t_{el} + \Delta t_c$; that is, equal to $\lceil \frac{t_{el}}{\Delta t_c} \rceil \Delta t_c$, where $\lceil x \rceil$ is the smallest integer that is larger than x. If the fault is active during the entire period of rollback or the contaminated part is larger than the reexecuted part of the program, the rollback recovery will fail and the corresponding "wasted" time (increase of FTL) is equal to $N_c \Delta t_c$.

Restart—If a large portion of the program execution is contaminated due to a large t_{el}, its execution is repeated from the beginning. The time (computation loss) taken for the restart process depends on 1) the time to detect an error and 2) the types of restart (i.e., hot, warm, and cold restarts) following a system reconfiguration. We use t_o and F_o to denote the time of error occurrence measured since the beginning of program execution and the PDF of t_o in a program, respectively.

3.2 Stringing Individual Stages Together

As mentioned earlier, all error-handling scenarios can be described, as shown in in Fig. 2, by the "paths" composed of several sequential stages from error detection to the corresponding recovery. Clearly, an error-handling policy depends on several mutually exclusive events, where one event represents a scenario and its occurrence depends on fault behaviors and the policy-related parameters. The probability of the occurrence of each event can thus be calculated by using the PDF of fault active duration (F_a) and the policy parameters such as Δt_c, N_c, or t_{rp}. The FTL of a certain error-handling policy is therefore obtained by using the probabilities of all possible events/scenarios and the times spent for these events/scenarios. Note that the time spent for each scenario is obtained by adding the times spent for all error-handling stages on the path representing the scenario. Likewise, we can obtain the PDF of FTL for a certain error-handling policy:

$$F_l(t) = \sum_{i=1}^{n} F_l(t|S_i)P(S_i), \tag{3.1}$$

where S_i indicates the ith scenario of an error-handling policy and $P(S_i)$ and n are the probability of the occurrence of S_i and the number of all possible scenarios in the selected error-handling policy, respectively. Equation (3.1) describes $F_l(t)$ as a weighted sum of conditional PDF. Each S_i's conditional PDF is computed by convolving the PDF of the time spent for every stage on the corresponding path or error-handling policy. The time spent for every possible error-handling stage is described as deterministic values or random variables with certain PDFs. We will investigate each error-handling stage individually.

Now, we characterize the error-handling process into four policies according to the types of error-detection mechanisms and recovery methods combined with temporal and spatial redundancy:

1. restart after reconfiguration,
2. rollback,
3. retry, and
4. retry then rollback.

These cover various dynamic- and/or temporal-redundancy methods. Specifically, we can describe the error-handling policies as follows. (Note that the number of all possible scenarios in each error-handling policy is defined by n.)

Policy 1 ($n = 2$): $S_1 =$ successful restart after diagnosis and reconfiguration, $S_2 =$ unsuccessful restart due to incorrect diagnosis then repeat.

Policy 2 ($n = 2$): $S_1 =$ successful rollback after diagnosis, $S_2 =$ unsuccessful rollback then restart after diagnosis and reconfiguration.

Policy 3 ($n = 2$): $S_1 =$ successful retry, $S_2 =$ unsuccessful retry then restart after reconfiguration.

Policy 4 ($n = 3$): $S_1 =$ successful retry, $S_2 =$ unsuccessful retry and successful rollback, $S_3 =$ unsuccessful retry and unsuccessful rollback then restart after reconfiguration.

While signal-level detection mechanisms can capture the faulty module immediately upon occurrence of an error and can thus invoke retry in Policies 3 and 4, function-level detection mechanisms—that cause a nonzero error latency and thus require the diagnosis process to locate the faulty module—may be used for Policies 1 and 2. The probabilities of scenario occurrences and the (conditional) PDFs of the above scenarios are derived by using the variables defined earlier for individual error-handling stages.

For simplicity, we do not consider the occurrence of an error/failure due to a second fault during the recovery from the first fault. If we need to consider the effects of such an error/failure, we cannot derive a closed-form PDF of FTL, but can instead derive the moments of FTL by using recursive equations, which can then be used to derive the PDF of FTL numerically.

Policy 1 is a simple form of dynamic redundancy, i.e., to restart the task from the beginning after identifying and replacing the faulty module with a nonfaulty spare. The first scenario is a successful restart with correct diagnosis. Thus, the probability of its occurrence is equal to that of successful diagnosis (p_d) and the time (t_l) spent on this scenario becomes

$$t_l = t_{el} + t_d + t_r + t'_o, \tag{3.2}$$

where t_d and t_r are deterministic variables and t_{el} is a random variable with the PDF, F_{el}. t'_o is also a random variable indicating the time of error occurrence given the fact that an error had occurred during the execution of a task ($0 \le t'_o \le T$), i.e., having the conditional distribution $F_o(t|an\ error)$ as its PDF. Let $t = t_l - t_d - t_r$, then

$$P(S_1) = p_d, \tag{3.3}$$

$$F_l(t|S_1) = F_{el}(t) * F_o(t|an\ error). \tag{3.4}$$

Similarly, $P(S_2)$ and $F_l(t|S_2)$ are derived for the second scenario. Since an unsuccessful restart (of the second scenario) wastes more time than the first scenario by the amount of the incorrect diagnosis time plus the (error) latency for a second error detection due to this incorrect diagnosis, t_l is changed to

$$t_l = t_{el} + t_d + t_{el} + t_d + t_r + t'_o = 2t_{el} + 2t_d + t_r + t'_o. \tag{3.5}$$

Let $t = t_l - 2t_d - t_r$, then

$$P(S_2) = 1 - p_d, \tag{3.6}$$

$$F_l(t|S_2) = F_{el}\left(\frac{t}{2}\right) * F_o(t|an\ error). \tag{3.7}$$

Policies 2 and 3 use rollback with diagnosis and retry upon error detection, respectively. Reconfiguration is also called for if the temporal-redundancy approach became unsuccessful. Since the first scenario of Policy 2 is a successful rollback, the probability of its occurrence depends on the probability of successful diagnosis (p_d), the parameters of rollback (Δt_c and N_c), the error latency (t_{el}), and the fault active duration (t_a). For a successful rollback, 1) a faulty module must be identified with correct diagnosis, 2) t_{el} must be smaller than $N_c\Delta t_c$, which is the maximum allowable time for rollback, and 3) the fault must disappear within $N_c \Delta t_c$. Let p_t be the percentage of transient faults, then

$$P(S_1) = p_t p_d F_a(N_c \Delta t_c) F_{el}(N_c \Delta t_c). \tag{3.8}$$

The time spent for this case is simply obtained as

$$t_l = t_{el} + t_d + \left\lfloor \frac{t_{el}}{\Delta t_c} \right\rfloor \Delta t_c. \tag{3.9}$$

Let $t = t_l - t_d$, then

$$F_l(t|S_1) = F_{el}(t) * F_m\left(\frac{t}{\Delta t_c}\right), \tag{3.10}$$

where F_m is a *PDF* for $m = \lfloor \frac{t_{el}}{\Delta t_c} \rfloor$. The probability of the second scenario being exclusive of the first one is equal to $1 - P(S_1)$

$$P(S_2) = 1 - p_t p_d F_a(N_c \Delta t_c) F_{el}(N_c \Delta t_c). \tag{3.11}$$

The time spent for this scenario is increased to

$$\begin{aligned} t_l &= t_{el} + t_d + N_c \Delta t_c + t_d + t_r + t'_o \\ &= t_{el} + 2t_d + N_c \Delta t_c + t_r + t'_o. \end{aligned} \tag{3.12}$$

Let $t = t_l - 2t_d - N_c \Delta t_c - t_r$, then

$$F_l(t|S_2) = F_{el}(t) * F_o(t|an\ error). \tag{3.13}$$

For Policy 3, which does not require fault diagnosis due to the assumed immediate and correct detection of errors with signal-level detection mechanisms, the probabilities of scenario occurrences and PDFs are derived similarly to Policy 2. For a successful retry, the error must be detected before contaminating the result of executing the instruction that will be retried (Δt_i) and the fault must become inactive within t_{rp} if the time spent is $t_{el} + t_a$. Thus,

$$P(S_1) = p_t F_a(t_{rp}) F_{el}(\Delta t_i), \tag{3.14}$$

$$F_l(t|S_1) = F_{el}(t) * F_a(t). \tag{3.15}$$

When a retry is unsuccessful, the time spent for this becomes

$$t_l = t_{el} + t_{rp} + t_r + t'_o. \tag{3.16}$$

Let $t = t_l - t_{rp} - t_r$, then

$$P(S_2) = 1 - p_t F_a(t_{rp}) F_{el}(\Delta t_i), \tag{3.17}$$

$$F_l(t|S_2) = F_{el}(t) * F_o(t|an\ error). \tag{3.18}$$

Policy 4 has three scenarios whose probabilities and PDFs are obtained by combining those of Policies 2 and 3. The first scenario is a successful retry, for which $P(S_1)$ and $F_l(t|S_1)$ are equal to those of the first scenario in Policy 3 (i.e., (3.15)). The second scenario is a successful rollback following an unsuccessful retry. Thus, $P(S_2)$ and $F_l(t|S_2)$ can be obtained by modifying (3.8) and (3.10) to include the effects of an unsuccessful retry. Let $t = t_l - t_{rp}$, then

$$P(S_2) = p_t[F_a(N_c \Delta t_c) F_{el}(N_c \Delta t_c) - F_a(t_{rp}) F_{el}(\Delta t_i)], \tag{3.19}$$

$$F_l(t|S_2) = F_{el}(t) * F_m\left(\frac{t}{\Delta t_c}\right). \tag{3.20}$$

The third scenario is to restart with reconfiguration following an unsuccessful retry, then rollback when the time spent is $t_l = t_{el} + t_{rp} + N_c \Delta t_c + t_r + t'_o$. Thus, if $t = t_l - t_{rp} - N_c \Delta t_c - t_r$ then

$$P(S_3) = 1 - p_t F_a(N_c \Delta t_c) F_{el}(N_c \Delta t_c) \tag{3.21}$$

$$F_l(t|S_3) = F_{el}(t) * F_o(t|an\ error). \tag{3.22}$$

With these derived probabilities and conditional PDFs, we can compute the *PDF* of FTL for each policy from (3.1).

4 Application of Fault Tolerance Latency

A real-time control system is composed of a controlled process/plant, a controller computer, and an environment, all of which work synergistically. A control system does not generally fail instantaneously upon occurrence of a controller-computer failure. Instead, for a certain duration, the system stays in a safe/stable region or in the admissible state space, even without the updated control input from the controller computer. However, a serious degradation of system performance or catastrophe, called a *dynamic failure* (or system failure), occurs if the duration of missing the update (or incorrect update) of the control input due to malfunctioning of the controller computer exceeds the control system deadline (CSD) [14], [15]. The CSD represents system inertia/resilience against a dynamic failure, which can be derived experimentally or analytically using the state dynamic equations of the controlled process, the information on fault behaviors associated with environmental characteristics (such as electro-magnetic interferences [7]), and the control algorithms programmed in the control computers [14].

When an error/failure occurs in a controller computer, the error must be recovered within a certain period to meet the timing constraints and to avoid a dynamic failure. Roark et al. [13] called this period the *Application Required Latency* (ARL) and presented several empirical examples of ARL for flight control, missile guidance, air data system, automatic tracking, and recognition applications. It is important to note that one can derive the ARL analytically using the CSD information [14] because the sum of ARL and the maximum time to execute the remaining control task to generate a correct control input is equal to the CSD. Using this information about the CSD of the controlled process and the FTL of the controller computer, one can select (or design) an appropriate error-handling policy by making a trade-off between temporal and spatial redundancy while meeting the strict timing constraint:

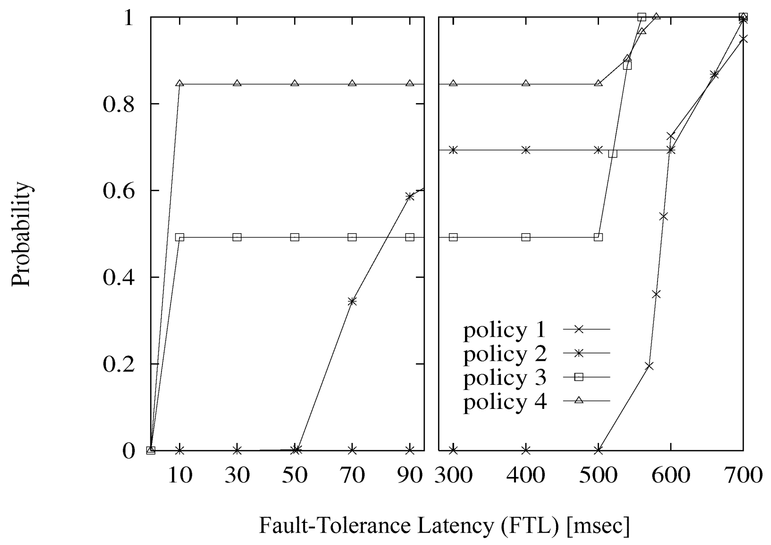

Fig. 3. PDF of the FTL with cold spares.

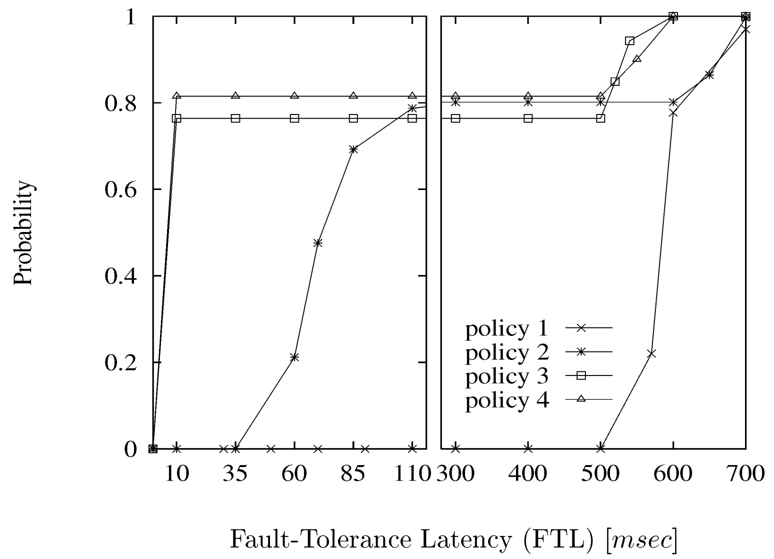

Fig. 4. PDF of the FTL with policy parameters different from those of Fig. 3: $p_d = 0.97$, $N_c = 5$, and $t_{rp} = 2$.

$$FTL \leq \ deadline \\ - maximum\ time\ to\ execute\ the\ remaining\ control\ task. \quad (4.1)$$

One can also estimate the system's ability of meeting the timing constraint in the presence of controller-computer failures, which is characterized by the probability of no dynamic failure using the evaluated deadline and FTL. The timing constraints should be satisfied for all errors/failures occurring sequentially during the life of one task.

Due to their possible dependency on each other and also on the underlying application, it is sometimes difficult to obtain accurate values/models of the parameters and latencies of individual stages that are used by the proposed method to evaluate FTL. Fortunately, however, some of these parameters are determined during the design process (as opposed to measuring them at run-time) and others had previously been evaluated by several researchers. For example, one can obtain $\{T, N_c, \Delta t_i, \Delta t_c, t_{rp},\}$ by using design parameters, $\{t_{el}, t_e, t_a, p_d, p_t\}$ by using field data, and t_r by using the results in [1], [8].

We now present an example to demonstrate the usefulness of the evaluated FTL. Consider a pooled-spares system,[3] which consists of multiple modules connected via a backplane. Let the basic time unit be one millisecond and let the task execution time in the absence of errors and the mean execution time of one instruction be given as $T = 50 (= 0.05 sec)$ and $\Delta t_i = 0.002 (= 2 \mu sec)$, respectively. The error latency is assumed to follow an exponential distribution with mean 12 and 0.002 for function- and signal- level detection mechanisms, respectively. The fault occurrence and duration are also governed by exponential distributions, where the mean value of active duration is 0.5, and

3. The system consists of power supplies, input/output modules, and a set of identical data processing modules, a subset of which are assigned to processing tasks. The remaining modules can be used as spares in case of an error/failure.

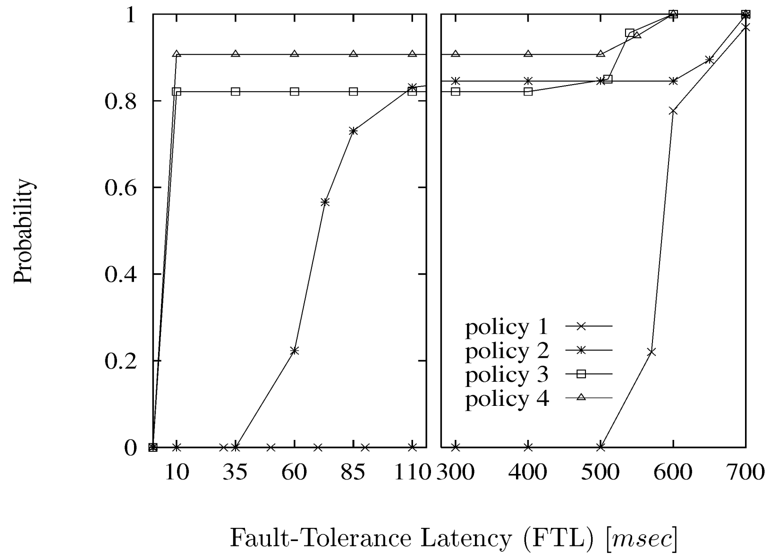

Fig. 5. PDF of the FTL under fault environments different from those of Fig. 3: $p_t = 0.95$ and $E(t_a)$ (the mean active duration) $= 0.25$.

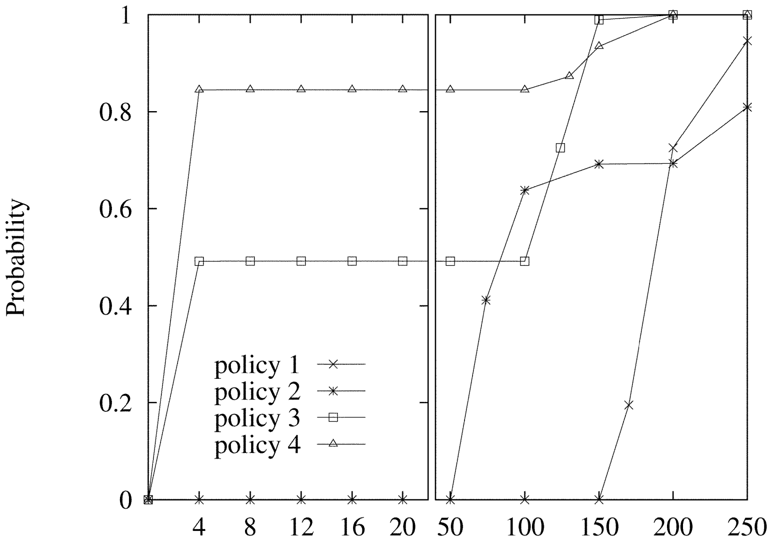

Fig. 6. PDF of the FTL with a reconfiguration strategy different from that of Fig. 3: warm spares.

the percentage of transient faults (p_t) is about 0.9. Then, given that an error occurred during T, the occurrence time, t_e, is uniformly distributed over T. The diagnosis time, t_d, is 50, which is assumed to yield coverage $p_d = 0.95$. When cold spares are used, it is assumed to take 500 units of time $(= 0.5sec)$ for system reconfiguration. We also assume that this value can be reduced to 100 by using warm spares. When applying rollback recovery, we set $\Delta t_c = 5$ and $N_c = 4$, whereas the retry period, t_{rp}, is set to 1. Under these conditions, the PDF of FTL are evaluated for the four representative policies by using the method developed in Section 3. These functions are plotted in Fig. 3.

From the above evaluation of FTL, one can conclude that Policies 1 and 2 are acceptable only when $t_{hd} > 14T(= 700 = 0.7sec)$. While the FTL of Policy 1 is distributed around $12T(= 600)$ with a small variance, Policy 2 has a wide range bounded by $14T$, indicating that Policy 2 is less likely than Policy 1 to violate the timing constraint, (4.1), under the above chosen conditions. Policies 3 and 4 that use retry have better distributions as compared to Policies 1 and 2, which satisfy the constraint $t_{hd} > 12T$. However, to be effective, retry usually requires dedicated hardware and immediate error detection. (Note that the mean error latency of Policies 3 and 4 in Fig. 3 is 0.002.)

Fig. 4 plots FTL while varying the policy parameters. We adopt a more accurate diagnosis process with $p_d = 0.97$ and change rollback and retry policies to $N_c = 5$ and $t_{rp} = 2$, respectively. The error-detection mechanisms are also improved to decrease the error latencies to 10 and 0.001 for both function- and signal-level mechanisms. In this case, the mean values of FTL become smaller, but the upper bounds of FTL are not changed. Thus, we can draw the same conclusion as Fig. 3 in selecting an appropriate error-handling policy.

We also consider different fault parameters: p_t and the mean value of active duration are changed to 0.95 and 0.25, respectively. Since the temporal-redundancy approaches get better under such conditions, the FTLs of Policies 2, 3, and 4 cluster small values, as depicted in Fig. 5. However,

TABLE 1
Different Conditions between the Cases in Figs. 3, 4, 5, and 6

Item \ Figure	3	4	5	6
RS	CS	CS	CS	**WS**
p_d	0.95	**0.97**	0.95	0.95
N_c	4	**5**	4	4
t_{rp}	1	**2**	1	1
p_t	0.9	0.9	**0.95**	0.9
$E(t_a)$	0.5	0.5	**0.25**	0.5

RS: Reconfiguration Strategy, CS: Cold Spares, WS: Warm Spares

one still cannot neglect some possible FTLs larger than $10T$, albeit with small probabilities.

When the CSD, t_{hd}, is tight, like $t_{hd} \leq 10T(= 500)$, no policy can satisfy (4.1). It is shown in Figs. 4 and 5 that FTL does not change significantly even if the policy and/or fault parameters are changed. Considering the fact that reconfiguration is the most time-consuming among all the error-handling stages, we use warm spares to reduce t_r, which significantly skews the PDF of FTL to the right, as shown in Fig. 6, where all parameters but t_r are the same as those in Fig. 3. In that case, Policies 3 and 4 are suitable for systems with $t_{hd} > 4T(= 200)$.

If the timing constraint is tighter, e.g., $t_{hd} \leq 4T$, we can conclude that static redundancy (or hot spares) must be used at the expense of spatial redundancy since no policy using dynamic or temporal redundancy can satisfy the stringent constraint.

5 Conclusion

In this paper, we evaluated FTL for general error-handling policies that combine temporal and spatial redundancy. We investigated all the individual error-handling stages from error detection to complete recovery from the error and used some deterministic and random variables to model the times spent for these stages. This is in sharp contrast to the previous work that evaluated error-recovery times with simple models or deterministic data collected from experiments. As shown in the example—although the parameters used in the example are chosen somewhat arbitrarily, their choice would not change our conclusion—the evaluated FTL is a key to the selection of an appropriate error-handling policy, especially for real-time control computers.

The main contribution of this paper is twofold: 1) demonstrating the importance of FTL in designing and evaluating fault-tolerant computers from real-time application's perspectives and 2) analytically deriving the relations between the FTL (and the latencies of some individual stages) and the parameters of both fault behaviors and fault tolerance designs.

There are also related problems worth further investigation, including:

- It is important to derive, if possible, a closed-form pdf expression for the time spent for each individual stage. In case an exact closed-form pdf is not obtainable, an approximate expression may be used to determine an appropriate error-handling policy.
- The FTL strongly depends upon fault coverage, which was assumed to be a constant determined by the diagnosis stage. However, fault coverage is, in reality, not simple to determine due mainly to the effects of many coupled testing/detection methods. The task type and the failure occurrence rate also affect fault coverage. When periodic diagnoses are used, there is a trade-off between accuracy (fault coverage) and time (frequency and diagnosis time). It is important to study all the factors determining fault coverage and analyze its effects on FTL.

Appendix

List of Symbols

t_{el} : error latency
$f_{el}(t)$: probability density function of t_{el}
$F_{el}(t)$: probability distribution function of t_{el}
δt_i : mean execution time of one instruction
T : nominal task execution time
t_d : time spent for fault diagnosis
p_d : diagnostic coverage
t_r : time spent for system reconfiguration
t_{rp} : retry period
t_a : active duration of a (transient) fault
$F_a(t)$: probability distribution function of t_a
$E(t_a)$: mean active duration of a fault
δt_c : intercheckpoint interval
N_c : maximum checkpoints rolled back
t_o : time of error occurrence
$F_o(t)$: probability distribution function of t_e
p_t : percentage of transient faults
t_l : time spent on each scenario
$F_l(t)$: probability distribution function of t_l
S_i : the ith scenario of an error-handling policy
$P(S_i)$: probability of the occurrence of S_i
t_{hd} : control system deadline
t'_o : time of error occurrence given that an error occurred during a time interval T
n : number of all possible scenarios in a selected error-handling policy

Acknowledgments

The authors would like to thank Chuck Roark and his colleagues of the TI Microelectronic Center, Plano Texas, for the quantitative data of the reconfiguration time. The work reported was supported in part by a Texas Instruments Grant and the U.S. Office of Naval Research under Grant N00014-94-1-0229. Any opinions, findings, and conclusions or recommendations expressed in this paper are those of the authors and do not necessarily reflect the view of the funding agencies.

References

[1] P. Barton, "Fault Latency White Paper," technical report, Texas Instruments, Microelectronics Dept., Plano, Tex., Jan. 1993.
[2] R.W. Butler and A.L. White, "SURE Reliability Analysis," NASA technical paper, Mar. 1990.

[3] J. Dugan, K. Trivedi, M. Smotherman, and R. Geist, "The Hybrid Automated Reliability Prediction," *AIAA J. Guidance, Control, and Dynamics,* pp. 319-331, May 1986.

[4] G.B. Finelli, "Characterization of Fault Recovery through Fault Onjection on FTMP," *IEEE Trans. Reliability,* vol. 36, no. 2, pp. 164-170, June 1987.

[5] R.M. Geist, M. Smotherman, and R. Talley, "Modeling Recovery Time Distributions in Ultrareliable Fault-Tolerant Systems," *Digest of Papers, Fault-Tolerant Computing Symp.-20,* pp. 499-504, June 1990.

[6] A.L. Hopkins Jr., T.B. Smith III, and J.H. Lala, "FTMP–A Highly Reliable Fault-Tolerant Multiprocessor for Aircraft," *Proc. IEEE,* vol. 66, no. 10, pp. 1,221-1,239, Oct. 1978.

[7] H. Kim and K.G. Shin, "Modeling Externally-Induced Faults in Controller Computers," *Proc. 13th IEEE/AIAA Digital Avionics Systems Conf.,* pp. 402-407, Phoenix, Ariz., Oct. 1994.

[8] H. Kim and K.G. Shin, "On Reconfiguration Latency in Fault-Tolerant Systems," *Proc. IEEE 1995 Aerospace Applications Conf.,* pp. 287-301, Snowmass at Aspen, Colo., Feb. 1995.

[9] C.M. Krishna, K.G. Shin, and R.W. Butler, "Synchronization and Fault-Masking in Redundant Real-Time Systems," *Digest of Papers, Fault-Tolerant Computing Symp.-14,* pp. 152-157, June 1984.

[10] J.H. Lala, "Fault Detection, Isolation and Configuration in FTMP: Methods and Experimental Results," *Proc. Fifth IEEE/AIAA Digital Avionics Systems Conf.,* pp. 21.3.1-21.3.9, 1983.

[11] S.R. McConnel, D.P. Siewiorek, and M.M. Tsao, "The Measurement and Analysis of Transient Errors in Digital Computer Systems," *Digest of Papers, Fault-Tolerant Computing Symp.-9,* pp. 67-70, June 1979.

[12] J. McGough, M. Smotherman, and K.S. Trivedi, "The Conservativeness of Reliability Estimates Based on Instantaneous Coverage," *IEEE Trans. Computers,* vol. 34, no. 7, pp. 602-608, July 1985.

[13] C. Roark, D. Paul, D. Struble, D. Kohalmi, and J. Newport, "Pooled Spares and Dynamic Reconfiguration," *Proc. NAECON '93,* pp. 173-179, May 1993.

[14] K.G. Shin and H. Kim, "Derivation and Application of Hard Deadlines for Real-Time Control Systems," *IEEE Trans. Systems, Man, and Cybernetics,* vol. 22, no. 6, pp. 1,403-1,413, Nov./Dec. 1992.

[15] K.G. Shin, C.M. Krishna, and Y.-H. Lee, "A Unified Method for Evaluating Real-Time Computer Controller and Its Application," *IEEE Trans. Automated Controls,* vol. 30, no. 4, pp. 357-366, Apr. 1985.

[16] K.G. Shin and Y.-H. Lee, "Error Detection Process—Model, Design, and Its Impact on Computer Performance," *IEEE Trans. Computers,* vol. 33, no. 6, pp. 529-539, June 1984.

Hagbae Kim (S'90-M'94) received the BS degree in electronics engineering from Seoul National University, Seoul, Korea, in 1988, and the MS and PhD degrees in electrical engineering and computer science from the University of Michigan, Ann Arbor, in 1990 and 1994, respectively. Currently, he is an assistant professor of electrical and computer engineering at Yonsei University, Seoul, Korea, where he has been since 1996. He was a National Research Council (NRC) resident research associate at NASA Langley Research Center, Hampton, Virginia, from 1994 to 1996, where he participated in the Highly Intensity Radiation Field (HIRF) project assessing the effects of ElectroMagnetic Interference (EMI) on digital controller computers. His research interests include real-time systems, fault-tolerant computing, reliability modeling, and automation technologies. He is a member of the IEEE.

Kang G. Shin received the BS degree in electronics engineering from Seoul National University, Seoul, Korea, in 1970, and both the MS and PhD degrees in electrical engineering from Cornell University, Ithaca, New York, in 1976 and 1978, respectively. From 1978 to 1982, he was on the faculty of Rensselaer Polytechnic Institute, Troy, New York. He has held visiting positions at the U.S. Airforce Flight Dynamics Laboratory, AT&T Bell Laboratories, Computer Science Division within the Department of Electrical Engineering and Computer Science at the University of California, Berkeley, and International Computer Science Institute, Berkeley, California, IBM T.J. Watson Research Center, and Software Engineering Institute at Carnegie Mellon University. He is currently a professor and director of the Real-Time Computing Laboratory, Department of Electrical Engineering and Computer Science, The University of Michigan, Ann Arbor, Michigan. He also chaired the Computer Science and Engineering Division, EECS Department, The University of Michigan for three years beginning in January 1991.

He has supervised the completion of 40 PhD theses and authored/coauthored about 500 technical papers and numerous book chapters in the areas of distributed real-time computing and control, computer networking, fault-tolerant computing, and intelligent manufacturing. He coauthored (with C.M. Krishna) a textbook *Real-Time Systems* (McGraw Hill, 1997). In 1987, he received the Outstanding IEEE Transactions on Automatic Control Paper Award, and the Research Excellence Award in 1989 and the Outstanding Achievement Award in 1999 from The University of Michigan. In 1985, he founded the Real-Time Computing Laboratory, where he and his colleagues are investigating various issues related to real-time and fault-tolerant computing.

His current research focuses on Quality of Service (QoS) sensitive computing and networking with emphasis on timeliness and dependability. He has also been applying the basic research results to telecommunication and multimedia systems, intelligent transportation systems, embedded systems, and manufacturing applications.

He is an IEEE fellow and a member of the Korean Academy of Engineering, is the general chair of the 2000 IEEE Real-Time Technology and Applications Symposium, was the program chairman of the 1986 IEEE Real-Time Systems Symposium (RTSS), the general chairman of the 1987 RTSS, the guest editor of the 1987 August special issue of *IEEE Transactions on Computers* on real-time systems, a program cochair for the 1992 International Conference on Parallel Processing, and served on numerous technical program committees. He also chaired the IEEE Technical Committee on Real-Time Systems during 1991-1993, was a distinguished visitor of the Computer Society of the IEEE, an editor of *IEEE Transactions on Parallel and Distributed Computing*, and an area editor of the *International Journal of Time-Critical Computing Systems*.