Reliability Enhancement of Real-Time Multiprocessor Systems through Dynamic Reconfiguration*

Kai Yu and Israel Koren
Department of Electrical and Computer Engineering
University of Massachusetts at Amherst
Amherst, MA 01003

Abstract

Enhancing the reliability of a system executing real-time jobs is, in many cases, one of the most important design goals. A dynamically reconfigurable system offers an approach for improvement of reliability. To achieve high reliability, the most suitable recovery action must be used when a fault occurs, which means that some kind of optimal recovery strategy should be followed. This is called a dynamic recovery strategy. To satisfy the service requirements of real-time jobs with hard deadlines, a more powerful system, intuitively, should always be preferred. On the other hand, higher processing capacity means more processing modules and electronics parts, which may result in more frequent faults and a higher risk that the system will fail to complete the real-time jobs prior to their deadline. In this paper, we investigate the reliability enhancement of a real-time distributed computing system with hard deadlines through the employment of dynamic recovery strategies. Since the classical reliability evaluation technique is not applicable to a dynamically reconfigurable system, we present a new approach to reliability evaluation. The results show that the optimal recovery policy can significantly improve the system's reliability, that both the job arrival rate and the job's deadline have significant effect on the optimal reliability and optimal policy, and that for a given workload and deadline, the maximum of the system reliability can be achieved at a certain (optimal) configuration.

1: Introduction

In a real-time distributed computing system, to meet the service requirement of jobs with hard deadlines, the system must keep a high momentary processing power and minimize disturbances from both transient and permanent faults. The design objective is to achieve a high probability that the system is functioning and satisfying the requirements of real-time jobs in both the fault-free state and the fault state. In the fault-free state, a system should offer an adequate processing power (or service rate) to real-time jobs. For a given job arrival rate, the higher the system service rate, the higher the reliability of the service. In the fault state, the probability that the system is able to satisfy the real-time job's requirements is reduced, which means that frequent faults will diminish the system reliability. Higher processing power requires more processing elements, which in turn produce more frequent faults. Therefore, there is a balance point at which the system can achieve the maximal reliability.

This work was supported in part by DARPA, under contract B855, and managed by the Space and Naval Warfare Systems Command under Contract N00039-94-C-0165.

Faults occur in computing systems, most of them either intermittent or transient faults, as a result of physical environment variants and electronic wear-out. Some recovery procedure is therefore commonly used to avoid having to prematurely give up system resources. Several recovery techniques have been suggested and are being used in various fault-tolerant systems [5-9, 11, 12, 14, 16]. These include instruction, memory and communication retries, and more complex steps such as program rollbacks, program reloads, and module replacements. However, any recovery procedure will result in some overhead and delay of service which will reduce the reliability. As system sizes increase, the occurrence rate of intermittent or transient faults increases as well. Consequently, the recovery overhead can dramatically reduce the system reliability. As a result, if some of the processors experience a high rate of fault occurrences, causing multiple invocations of the recovery procedure, we may consider disconnecting them from the system altogether.

Different recovery techniques handle different faults with varying efficiency. The recovery procedure must be carefully designed to reliably recover the system from faults with minimal overhead. Many different kinds of recovery strategies have been proposed. The dynamic recovery technique [3][15] is one of the most sophisticated. The underlying idea is that for a given system mission time, the choice of the recovery action should be concerned not only with the current system state and the fault symptom, but also with the remaining mission time. The alternative recovery actions constitute a set in which each action has only limited recovery capacity, and can cover only certain faults with a high probability of success and short overhead.

Berg and Koren [3] developed a model for a computing system consisting of one operational module and several standby modules. When a fault occurs in an operational module, the system can either retry the faulty module or replace it with a standby. The goal of the system recovery strategy is to maximize the application-oriented availability and minimize the cost of the faults and their corresponding recovery actions.

Xia, Chow, and Luo [15] extended the model to computing systems consisting of multiple operational and standby modules. When a fault occurs in an operational module, the system can either retry the faulty module, replace it with a standby module, or completely disconnect the faulty module, leaving a degraded system. The goal of the system recovery strategy is the same as in [3].

The expected real-time workload of the system must be a factor when determining the configuration of the system. For a given workload, the higher the processing power, the faster the response, and the higher the probability with which the system can complete a job before its deadline. On the other hand, more processing elements in operation may lead to a greater number of transient faults, which can reduce the reliability. The workload must, therefore, be one of the most important factors in the design of a dynamic recovery procedure.

In this paper, we investigate the reliability of real-time multiprocessor systems with hard deadlines. To simplify the analysis, we still assume that the system consists of multiple identical operational and standby modules, and that the alternative recovery actions are retry, replace, and disconnect. The multiprocessor system is allowed to operate in a degraded state. The objective here is to maximize the reliability of the service of real-time jobs with hard deadlines during the mission time. When a fault occurs in a module, a recovery procedure is selected based on relevant information such as the current system state, the remaining mission time, the workload, and the fault symptoms (e.g., whether the faulty module has previously experienced faults), so as to minimize the deterioration of the reliability caused by the degraded processing power or by the fault and its recovery action. The results show that the optimal recovery strategy can significantly enhance the real-time system reliability. The optimal policy will vary when the workload is changed. We also find that for given processor's transient fault rate and workload, the maximal reliability of a multiprocessor system can be achieved using a certain (optimal) number of processors; i.e., the reliability cannot be increased by simply using more active processors. In order to represent the fault arrival process precisely, we allow the arrival of permanent faults and transient faults to be

represented by two different Poisson processes. We also assign different fault arrival rates for those modules that have experienced faults in the past and those that have not.

The rest of this paper is organized as follows. In Section 2, the basic assumptions on the system are stated, and the optimization problem and the system reliability functions are described. The optimality equations are presented in Section 3. Some numerical examples are included in Section 4. Final conclusions are presented in Section 5.

2: Assumptions and model description

We denote by $\mathbf{s}$ a fault-free system state, $\mathbf{s} \in \mathcal{S}$ where $\mathcal{S} = \{(a_0, a_1, s_0, s_1)\}$; a_0 is the number of "new" nodes in operation, i.e., nodes which remained fault-free since the beginning of the mission; a_1 is the number of "old" nodes in operation, i.e., nodes which have previously experienced faults; s_0 is the number of new spare nodes; s_1 is the number of old spare nodes. A failing operational module can be disconnected or replaced by a spare one. Faults in operational modules occur according to two different Poisson processes since permanent faults are mainly due to component wear-out mechanisms, while most intermittent faults are the result of environmental causes. Permanent faults arrive at the rate λ_p, and transient (intermittent) faults arrive at the rate λ_0 on new nodes and λ_1 on old nodes ($\lambda_0 < \lambda_1$). The real-time jobs are processed in a distributive manner and the system service rate is $\mu(n) = C_\mu \cdot n^\beta$, with $C_\mu > 0$, $0 < \beta < 1$, and $n = a_0 + a_1$. The job arrival process is a Poisson process with rate γ, and there is a queue for buffering the incoming jobs if required. The real-time jobs with a hard-deadline have an identical deadline Δ, which means that the entire system fails whenever it fails to complete the service for any one of the jobs before its deadline. We further assume that at any time the system is in operation, the following condition holds:

$$\mu(n) \geq \gamma >> \sum_{i=1}^{n} (\lambda_T^i + \lambda_p^i) = \sum_{i=1}^{n} (\lambda_T^i) + n \cdot \lambda_p$$

where $\lambda_T^i \in \{\lambda_0, \lambda_1\}$. This assumption implies that the job arrival rate is much greater than the frequency of fault occurrences, so that the system will be in a steady-state during most of the interval between fault occurrences.

For a real-time computing system, we hope that it can service all arriving jobs within their deadline. The system reliability is, therefore, defined as the probability with which the system can service all the arriving jobs within their deadline throughout the system mission time.

When a fault occurs in an operational module, there are three alternative recovery procedures: retry, replace, and disconnect. In the retry procedure, the system first retries the task on the faulty module from the last checkpoint for a certain number of times. If these retries fail, the module is either replaced or disconnected. In a replace action, the faulty module is replaced by a spare one, and the uncompleted task is resumed from its last checkpoint. In a disconnect action, the faulty module is disconnected, and the uncompleted task is rescheduled on another module. The replaced or disconnected module can be tested off-line and, if still functional, it can serve as a spare. Any one of these procedures can bring about a system crash, leading to a catastrophic system service termination. Also, the faults and these recovery procedures will cause some service delay which will reduce the probability that the real-time job will be serviced within the deadline. We denote by p_{rtr}, p_{rpl}, and p_{dis} the probability of failure of the three recovery actions, retry, replace, and disconnect, respectively. Retry is the most reliable and efficient recovery procedure for transient faults, but it always keeps in place modules which have already experienced several faults in operation, usually leading to more frequent faults later on. If the fault is permanent, the faulty module must be replaced by a spare. If new spares are not available, additional modules which have previously experienced faults will become active and the system will experience more frequent transient faults,

greatly reducing its reliability. At this point, disconnection becomes a reasonable choice, although it forces the system to run at a lower processing power and reduces the probability with which the fault-free system can service the arriving jobs within their required deadline.

Therefore, a recovery strategy should be derived based on the available information to determine which recovery procedure should be executed so as to maximize the system reliability or minimize the system cost.

The optimization problem is

Given: The mission time $[0, T]$ and the three alternative fault-recovery actions:

$$\{retry, replace, disconnect\}$$

Select a recovery policy which will **maximize** the system reliability.

3: Fault-free system reliability and optimality equations

3.1: The fault-free system reliability function

Given system state $\mathbf{s} = (a_0, a_1, s_0, s_1)$ and job arrival rate γ, following the steady-state queueing analysis in [1], we can determine that the response time for an arriving job is

$$Pr\{response\ time \leq t\} = 1 - e^{-t/W}$$

where W is the average response time $W = 1/(\mu - \gamma)$.

Given a deadline Δ, the probability for the successful completion of a job is, therefore, $1-e^{-\Delta/W}$. When the system is in the fault-free state, it can offer a constant rate of service μ. Denote by $r(t)$ the probability with which all the arriving jobs can be serviced by the fault-free system over the time interval $[0, t]$. We then obtain the following equation:

$$r(t) = \sum_{i=0}^{\infty} e^{-\gamma t} \frac{(\gamma t)^i}{i!} \cdot (1 - e^{-\Delta/W})^i = e^{-(\gamma e^{-\Delta/W})t} \tag{1}$$

3.2: Optimality equations

Let $R(\mathbf{s}, t)$ denote the optimal reliability for the system in state $\mathbf{s} = (a_0, a_1, s_0, s_1)$ with remaining mission time t. The system total fault arrival rate is $\lambda(s) = a_0\lambda_0 + a_1\lambda_1 + (a_0 + a_1)\lambda_p$. For the mission interval $[0, t]$, the system is fault-free with probability $e^{-\lambda(s)t}$. The first fault arrives in the interval $[t - u, t - u + du]$ with probability $\lambda(s) \cdot e^{-\lambda(s)(t-u)}du$, where t and u are the system's total mission time and remaining mission time, respectively. If the system stays fault-free, it can offer satisfactory service with probability $r(s, t)$, which equals the probability $r(t)$ of the system in state $\mathbf{s}$. Otherwise, upon the arrival of the first fault, the optimal recovery action, the one which maximizes the reliability for the remaining mission time, is performed. The corresponding reliability is reduced by a factor of $(1 - p_{opt\_action})$, where $opt\_action \in \{retry, replace, disconnect\}$. The system is configured to another functional state and continues its operation until the next fault occurs. Hence, we have the following optimality equation for the system reliability function:

$$R(s, t) = r(s, t) \cdot e^{-\lambda(s)t} + \int_0^t \lambda(s) \cdot e^{-\lambda(s)(t-u)} \cdot r(s, t-u)$$

$$\times \{\sum_{i=0}^{1} \frac{a_i(\lambda_i + \lambda_p)}{a_0(\lambda_0 + \lambda_p) + a_1(\lambda_1 + \lambda_p)} \max[h^i_{rtr}(s, u),\ h^i_{rpl}(s, u),\ h^i_{dis}(s, u)]\} du \tag{2}$$

where $\mathbf{s} = (a_0, a_1, s_0, s_1)$ and $h^i_{rtr}(s,u)$, $h^i_{rpl}(s,u)$, $h^i_{dis}(s,u)$ are the reliability functions for the remaining mission time u when the corresponding recovery action, retry, replace, or disconnect, is taken.

$$h^i_{rtr}(s,u) = (1-p_{rtr})[\frac{\lambda_p}{\lambda_i+\lambda_p}\max\{(1-p_{rpl})R(s',u),\,(1-p_{dis})R(s'',u)\} + \frac{\lambda_i}{\lambda_i+\lambda_p}R(s''',u)]$$

where

$$s' = \begin{cases} (a_0, a_1, s_0-1, s_1), & \text{if } i=0 \text{ and } s_0>0 \\ (a_0-1, a_1+1, s_0, s_1-1), & \text{if } i=0 \text{ and } s_0=0,\ s_1>0 \\ (a_0+1, a_1-1, s_0-1, s_1), & \text{if } i=1 \text{ and } s_0>0 \\ (a_0, a_1, s_0, s_1-1), & \text{if } i=1 \text{ and } s_0=0,\ s_1>0 \end{cases}$$

$$s'' = \begin{cases} (a_0, a_1-1, s_0, s_1), & \text{if } i{=}1 \\ (a_0-1, a_1, s_0, s_1), & \text{if } i=0 \end{cases}$$

$$s''' = \begin{cases} s, & \text{if } i{=}1 \\ (a_0-1, a_1+1, s_0, s_1), & \text{if } i=0 \end{cases}$$

Similarly,

$$h^i_{rpl}(s,u) = (1-p_{rpl})[\frac{\lambda_p}{\lambda_i+\lambda_p}R(s',u) + \frac{\lambda_i}{\lambda_i+\lambda_p}R(s'',u)]$$

where

$$s' = \begin{cases} (a_0, a_1, s_0-1, s_1), & \text{if } i=0 \text{ and } s_0>0 \\ (a_0-1, a_1+1, s_0, s_1-1), & \text{if } i=0 \text{ and } s_0=0,\ s_1>0 \\ (a_0+1, a_1-1, s_0-1, s_1), & \text{if } i=1 \text{ and } s_0>0 \\ (a_0, a_1, s_0, s_1-1), & \text{if } i=1 \text{ and } s_0=0,\ s_1>0 \end{cases}$$

$$s'' = \begin{cases} (a_0, a_1, s_0-1, s_1+1), & \text{if } i=0 \text{ and } s_0>0 \\ (a_0-1, a_1+1, s_0, s_1), & \text{if } i=0 \text{ and } s_0=0,\ s_1>0 \\ (a_0+1, a_1-1, s_0-1, s_1+1), & \text{if } i=1 \text{ and } s_0>0 \\ (a_0, a_1, s_0, s_1), & \text{if } i=1 \text{ and } s_0=0,\ s_1>0 \end{cases}$$

and

$$h^i_{dis}(s,u) = (1-p_{dis})[\frac{\lambda_p}{\lambda_i+\lambda_p}R(s',u) + \frac{\lambda_i}{\lambda_i+\lambda_p}R(s'',u)]$$

where

$$s' = \begin{cases} (a_0, a_1-1, s_0, s_1), & \text{if } i{=}1 \\ (a_0-1, a_1, s_0, s_1), & \text{if } i=0 \end{cases}$$

$$s'' = \begin{cases} (a_0, a_1-1, s_0, s_1+1), & \text{if } i{=}1 \\ (a_0-1, a_1, s_0, s_1+1), & \text{if } i=0 \end{cases}$$

The dynamic programming technique is employed to solve the problem. It computes the reliability associated with the current system by using the information of previously solved, lower level, smaller size systems. In other words, we have to compute the reliability functions recursively in some order from the minimal state (0, 1, 0, 0) to the final state $\mathbf{s} = (a_0,\ a_1,\ s_0,\ s_1)$. The detailed algorithm for solving the problem is not described here for the sake of brevity.

4: Numerical examples

In this section we present several numerical examples illustrating the application of our proposed dynamic recovery procedure.

The system is operated with the initial state $\mathbf{s} = (10, 0, 1, 0)$ and the following parameters: fault rates $(\lambda_p, \lambda_0, \lambda_1) = (0.0001, 0.001, 0.01)$; service rate $\mu(n) = n^{0.7}$ (jobs per second); system crash probabilities $(p_{rtr}, p_{rpl}, p_{dis}) = (0.01, 0.02, 0.015)$; job arrival rate $\gamma = 2.0$ (jobs per second); job deadline $\Delta = 10$(seconds); and system mission time: $T = 100$(hours).

The optimal reliability functions are shown in Figure 1.

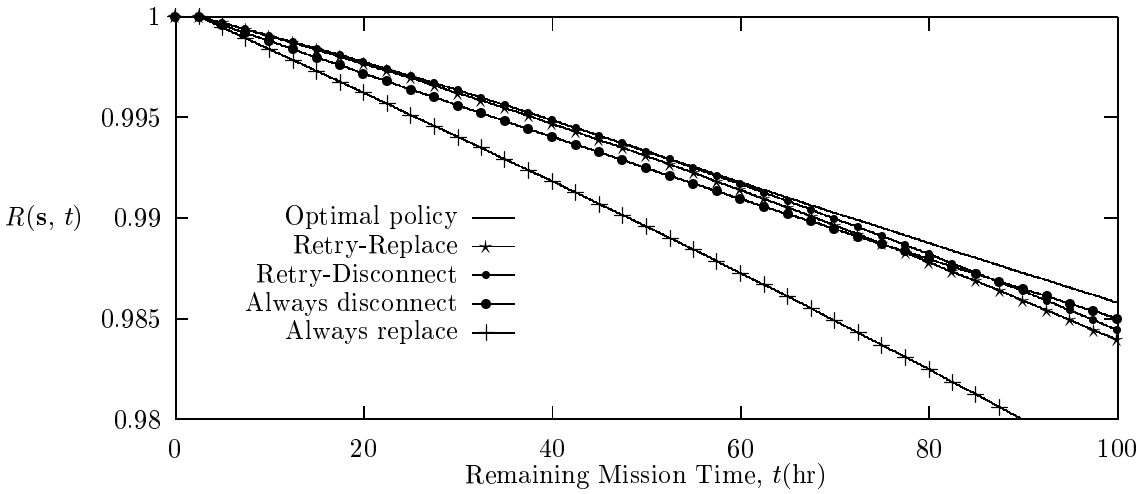

Figure 1: The reliability for the optimal recovery policy vs. simple policies ($\mathbf{s} = (10, 0, 1, 0)$, $\gamma = 2.0$, $\Delta = 10$).

The corresponding optimal recovery policy is

$$\begin{cases} \text{retry-disconnect}, & \text{if } 0 \le t < 45; \\ \text{disconnect}, & \text{if } 45 \le t \end{cases}$$

From Figure 1 we can see that the optimal policy can enhance the reliability and reduce the failure probability. We can also see that the "Always disconnect" achieves the best reliability among the simple policies for a mission time of 100 hours. This is because the workload is relatively light, the system has more modules than are needed to complete the jobs, and disconnecting the redundant modules can therefore reduce the fault rate.

Obviously the system reliability and the optimal switching policy should be significantly affected by the size of the deadline. This is shown in Figure 2.

For a given deadline, the system processing power may become insufficient when the job arrival rate increases. To investigate the dependence on the job arrival rate, we set the deadline to 4 and vary the job arrival rate. The resulting reliability and optimal policy are plotted in Figures 3 and 4.

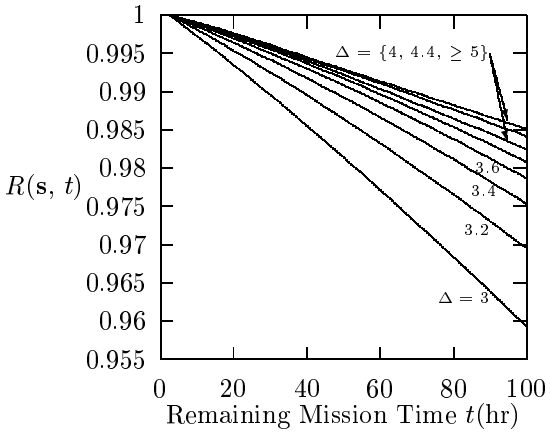

Figure 2: System optimal reliability for different values of the deadline (s = (10, 0, 1, 0), $\gamma = 2$).

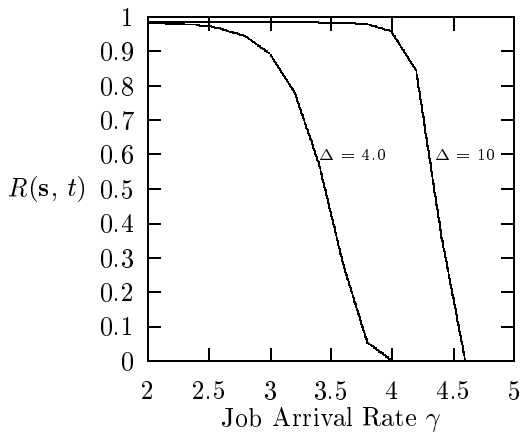

Figure 3: System reliability vs. job arrival rate.

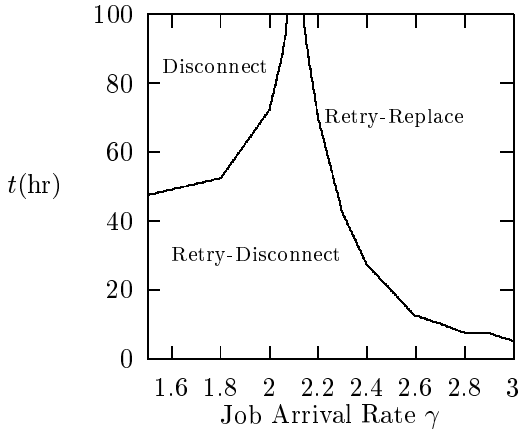

Figure 4: Optimal switching policy vs. job arrival rate and the remaining mission time ($\Delta = 4.0$).

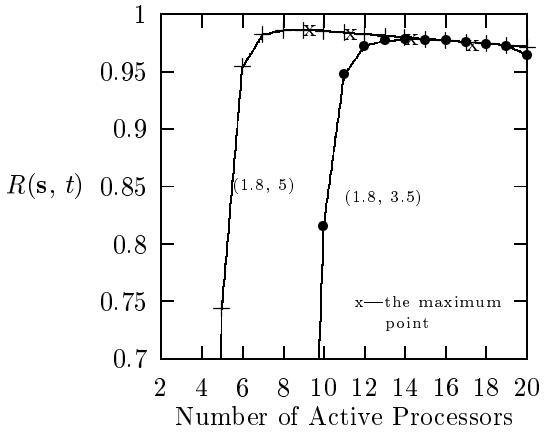

Figure 5: The system's reliability vs. number of active processors (s = (n, 0, 1, 0)).

The rapid decrease of the reliability in Figure 3 shows that the insufficiency of the system processing power, instead of the high frequency of fault occurrences, becomes the major contributor to reliability deterioration. Figure 4 shows that when the processing power is sufficient, the disconnect action is optimal in the case of a long remaining mission time, and that when the processing power is insufficient, the retry-replace action is optimal.

Finally, we calculate the reliability of a system with a single new spare and a varying number of active processors for two different values of the pair (γ, Δ). The results are shown in Figure 5. From this figure we can see that for different deadlines and job arrival rates, the maximal system reliability is achieved at different system configurations.

5: Conclusions

Improving reliability is one of the most important goals in real-time system design. We apply a dynamic recovery technique to a real-time distributed processing system. The results show that the optimal recovery policy can significantly enhance the system reliability and the optimal system reliability is greatly affected by the job arrival rate and the deadline. Reliability evaluation of the dynamic reconfigurable system is complicated and the classical evaluation techniques are no longer applicable. The method employed in this paper provides a new approach for reliability evaluation of a reconfigurable system. In addition, the results show that for a given workload and deadline, the reliability of a real-time system cannot be increased by merely adding more active processors because of the increase in the total fault rate for the system. The maximum can be achieved only at a certain size—the optimal configuration.

References

[1] A. O. Allen, *Probability, Statistics, and Queueing Theory*, Academic Press, pp. 262-267, 1990.

[2] M. D. Beaudry, "Performance-Related Reliability Measures for Computing Systems," *IEEE Trans. Comput.*, Vol. C-27, pp. 540-547, June 1978.

[3] M. Berg, and I. Koren, "On Switching Policies for Modular Redundancy Fault-Tolerant Computing Systems," *IEEE Trans. Comput.*, Vol. C-36, pp. 1052-1062, Sept. 1987.

[4] P. L'Ecuyer, and J. Malenfant, "Computing Optimal Checkpointing Strategies for Rollback and Recovery Systems," IEEE Trans. Comput., Vol. C-37, pp. 491-496, Apr. 1988.

[5] N. Elnozahy, and W. Zwaenepoel, "Manetho: Transparent Rollback-Recovery with Low Overhead, Limited Rollback, and Fast Output Commit," *IEEE Trans. Comput.*, Vol. C-41, pp. 526-531, May 1992.

[6] R. B. Hagmann, "A Crash Recovery Scheme for a Memory-Resident Database," *IEEE Trans. Comput.*, Vol. C-35, pp. 839-834, Sept. 1986.

[7] D. B. Johnson, W. Zwaenepoel, "Recovery in Distributed Systems Using Optimistic Message Logging and Checkpointing," *Journal of Algorithms*, Vol. 11, pp. 462-491, 1990.

[8] I. Koren, Z. Koren, and S. Y. H. Su, "Analysis of a Class of Recovery Procedures," *IEEE Trans. Comput.*, Vol. C35, pp. 703-712, Aug. 1986.

[9] Y.-H. Lee, and K. G. Shin, "Design and Evaluation of a Fault-Tolerant Multiprocessor Using Hardware Recovery Blocks," *IEEE Trans. Comput.*, Vol. C-33, pp. 113-124, Feb. 1984.

[10] Y.-H. Lee, and K. G. Shin, "Optimal Design and Use of Retry in Fault-Tolerant Computer Systems," *Journal of the Association for Computing Machinery*, Vol. 35, Jan. 1988.

[11] Y.-H. Lee, P. S. Yu, and B. R. Iyer, "Progressive Transaction Recovery in Distributed DB/DC Systems," *IEEE Trans. Comput.*, Vol. C-36, pp. 976-987, Aug. 1987.

[12] A. M. Saleh, and J. H. Patel, "Transient-Fault Analysis for Retry Techniques," *IEEE Trans. Comput.*, Vol. C-37, pp. 323-330, Aug. 1988.

[13] K. G. Shin, T.-H. Lin, and Y.-H. Lee, "Optimal Checkpointing of Real-Time Tasks," *IEEE Trans. Comput.*, Vol. C-36, pp. 1328-1341, Nov. 1987.

[14] K.-L. Wu, and W. K. Fuchs, "Recoverable Distributed Shared Virtual Memory," *IEEE Trans. Comput.*, Vol. C-39, pp. 460-469, Apr. 1990.

[15] G. Xia, Y.-C Chow, and K. Luo, "A Dynamic Recovery Strategy for Fault-Tolerant Computing in Multi-Processor Systems," *IEEE Workshop on Fault-Tolerant Parallel and Distributed Systems*, pp. 150-157, July, 1992.

[16] R. M. Yanney, and J. P. Hayes, "Distributed Recovery in Fault-Tolerant Multiprocessor Networks," *IEEE Trans. Comput.*, Vol. C-35, pp. 871-879, Oct. 1986.