Optimal Checkpointing of Real-Time Tasks

KANG G. SHIN, SENIOR MEMBER, IEEE, TEIN-HSIANG LIN, STUDENT MEMBER, IEEE, AND YANN-HANG LEE, MEMBER, IEEE

Abstract—**Analytical models for the design and evaluation of checkpointing of real-time tasks are developed. First, the execution of a real-time task is modeled under a common assumption of *perfect coverage* of on-line detection mechanisms (which is termed a *basic model*). Then, the model is generalized (to an *extended model*) to include more realistic cases, i.e., *imperfect coverages* of on-line detection mechanisms and acceptance tests. Finally, we determine an optimal placement of checkpoints to minimize the mean task execution time while the *probability of an unreliable result* (or *lack of confidence*) is kept below a specified level.**

In the basic model, it is shown that equidistant intercheckpoint intervals are optimal, whereas this is not necessarily true in the extended model. An algorithm for calculating the optimal number of checkpoints and intercheckpoint intervals is presented with some numerical examples for the extended model.

Index Terms—**Checkpointing, failure coverages, mean task execution time, on-line detection mechanisms and acceptance tests, optimal placement of checkpoints, probability of an unreliable result, rollback and restart failure recovery.**

I. Introduction

CHECKPOINTING a database system is defined as the operation of saving the current version of the database (called a *checkpoint*) on a separate secure device (such as backup tapes) and also saving the before-image and the after-image of all transactions made between two successive checkpoints (called the *audit trail*). When an error is detected, the system will stop normal operation and start a procedure for *rollback recovery,* which restores the system to the most recent checkpoint and then reprocesses the transactions recorded on the audit trail. Since checkpointing is an effective and economic method for improving reliability of database systems (compared to the hardware redundancy technique), it has been widely used and studied by many researchers.

Although most previous works have dealt with the performance evaluation of checkpointing in database systems (or other transaction-oriented systems), the same basic concept can be applied to real-time tasks. Note that a real-time task has stringent requirements for fast and correct execution. The studies in [1] and [2] have shown that checkpointing can greatly reduce the mean time of running a long program on an unreliable computing system. In their discussion, equal intercheckpoint intervals are used for inserting checkpoints in the program. However, the degree of confidence in execution results has not been addressed. The purpose of this paper is to consider checkpointing as a viable method to satisfy both of the above requirements (i.e., fast and correct execution) for real-time tasks under more realistic assumptions. Mathematical models will be developed first and the optimal solutions will then be derived. Our discussion begins with a brief review of the checkpointing techniques used in database systems.

Since for database applications the system is unavailable to users during error recovery, an obvious objective of checkpointing database systems is to maximize the portion of the time the system is available to users, i.e., *system availability.* Another useful objective is the *mean response time,* which is the average time a user has to wait until the system completes his transaction request. Availability and mean response time have been the primary criteria for evaluating the performance of checkpointing in database systems. The variables commonly used in such studies are 1) *checkpointing time* which is the time required to save a checkpoint, 2) *recovery time* which is the time needed to reload a checkpoint and reprocess the audit trail, and 3) *intercheckpoint interval* between two successive checkpoints. The only controllable variable is the intercheckpoint interval. The checkpointing time is system-dependent and usually assumed to be constant within a system. The recovery time depends on the length of the audit trail which is often assumed to be proportional to the number of transactions on the audit trail and hence depends upon the system load and the intercheckpoint intervals. Consequently, most of earlier works [3]–[8] have been to determine the optimum intercheckpoint intervals that either maximize the system availability or minimize the mean response time.[1] A common assumption in these works is that errors are detected immediately upon their occurrences.

Young [3] made a first-order approximation to the optimal intercheckpoint interval which minimizes the system overhead between errors under the assumption that 1) checkpointing time and intercheckpoint intervals are fixed, 2) errors do not occur during error recovery, and 3) error occurrence is a Poisson process. Chandy *et al.* [4] proposed three models: *A*, *B*, and *C*. All three models assume high system availability, fixed checkpointing time, and a Poisson error occurrence process. Model *A* further assumes that errors cannot occur during error recovery and the system load is constant. Model *B* allows errors to occur during error *recovery*. Model *C*

Manuscript received July 22, 1985; revised January 13, 1986 and March 6, 1987. This work was supported in part by NASA under Grants NAG-1-296 and NAG-1-492. An opinions, findings, and conclusions or recommendations expressed in this paper are those of the authors and do not necessarily reflect the views of NASA.

K. G. Shin and T.-H. Lin are with the Real-Time Computing Laboratory, Department of Electrical Engineering and Computer Science, The University of Michigan, Ann Arbor, MI 48109.

Y.-H. Lee is with the IBM Thomas J. Watson Research Center, Yorktown Heights, NY 10598.

IEEE Log Number 8715434.

[1] The probability distribution of response time is not usually available in a manageable form.

assumes that the arrival rate of transaction requests varies widely with time in a cyclic fashion. Optimal intercheckpoint intervals are determined for models A and B to achieve a maximum availability, while for model C the objective is to minimize the number of transactions arriving during checkpointing and error recovery. Gelenbe [5] studied this problem with a queueing model and assumed that intercheckpoint intervals, checkpointing time, and intererror intervals are all independent and exponentially distributed. Errors are not allowed to occur during checkpointing and error recovery. He then showed that the optimal average intercheckpoint interval which maximizes the system availability is a function of the system load. With a similar model, Gelenbe and Derochette [6] derived an expression for the optimal average intercheckpoint interval which minimizes the mean response time. They also found that the optimal intercheckpoint interval which minimizes the mean response time is usually smaller than that maximizing the system availability. Using theory in Markov renewal process and semi-regenerative process, Baccelli [7] obtained an analytic expression of the mean response time for the same model as in [6].

Recently, Tantawi and Ruschitzka [8] proposed a general model where an arbitrary distribution of the intererror interval is considered. In their model, errors may occur during checkpointing and error recovery, and checkpointing intervals are allowed to depend on the audit trail reprocessing time and error distribution. A general expression for the system availability was derived. An equicost strategy for selecting the intercheckpoint interval was also proposed and shown to be superior to the equidistant strategy. (Other works typically use the equidistant strategy with little justification.)

Checkpointing in a real-time system is quite different from that in a database system. A real-time system usually includes many tasks which do not communicate during their execution, although these tasks can exchange information through shared system memory, e.g., FTMP [9], [10] and the SIFT computer [11]. Because a real-time task is usually executed periodically, access to the system memory by any task is allowed only at the beginning (for input) and the end (for output) of execution. This restriction can be justified by the fact that real-time applications are usually well-defined to be decomposed in such a fashion and do not have luxury to allow for a long delay in accessing shared resources such as system memory or bus. Thus, a real-time task needs no audit trail since all the data are local once the task is initiated. When a failure occurs, the task has to redo all the computation after the last checkpoint or start all over again from the beginning.

Fast and correct execution of tasks is of the utmost importance to real-time systems. Thus, a real-time computer system generally will have hardware redundancies (e.g., multiple processors, memories, and buses). As in most multiprocessor systems, e.g., Cm*, we do not allow multiprogramming on each processor of the system. When a task is assigned to a processing unit in the system, it will run on that unit until it finishes (as long as the unit does not fail). So, a failure affects only the task which is running on the failed unit. Hence, it is no longer appropriate to think of the checkpointing technique from a *system-oriented view,* but rather each task should be considered as a unit on which checkpointing is applied, i.e., a *task-oriented view.*

Taking the task-oriented view, we shall derive two mathematical models to describe the behavior of task execution and occurrence of and recovery from errors. These models will then be used to determine optimal intercheckpoint intervals and an optimal number of checkpoints for a task by minimizing the mean task completion time subject to the required confidence in execution results.

The paper is organized as follows. In the following section, we introduce the general concept, design issues, and related terminology in the checkpointing of real-time tasks. Section III formally states the problem and presents the assumptions to be used. Section IV derives the optimal checkpointing strategy for the basic model. Section V considers the extended model where a numerical algorithm is developed for calculating an approximate solution. Section VI summarizes our results.

II. Checkpointing Real-Time Tasks

Checkpointing a real-time task means occassionally saving the state of the task on other safe devices such as tapes, disks, or even other (redundant) memory modules. The state of a task includes values of data variables and contents of the internal registers.[2] The saved states of a task are called *checkpoints*[3] or *recovery points* (RP's) [12], [13]. To ensure the correctness of the saved checkpoint, an *acceptance test* must be applied to the checkpoint before saving it [14]. There are also *on-line detection mechanisms* to detect fault manifestations during task execution [13], [15], [16]. When a module fails and the failure is detected either by the acceptance test or the on-line detection mechanism, the most recently saved checkpoint for the task running on this module will be loaded to a good module, and the task then resumes execution from that checkpoint.

A hardware *fault* is defined as an incorrect state caused by the physical change in a component, whereas an *error* is defined to be the erroneous information/data resulting from the manifestation of a fault. As we classified in [15], there are two important classes of detection mechanisms: one is termed the *signal-level detection mechanisms,* and the other is termed the *function-level detection mechanisms.* At the signal level, the manifestation of a fault is captured by built-in on-line detection mechanisms before the fault generates an error in a program. Undetected faults may generate errors which may then be captured by the function-level detection mechanisms. The acceptance test is one of the function-level detection mechanisms.

Consider the assumption that errors are detected immediately upon their occurrences. Extensive efforts have been made to design various "failure[4]" detection mechanisms, yet no detection mechanism can proclaim to cover all possible

[2] The task state may even include the program code if it does not have a backup on some other memory device or if the program itself may change during execution.

[3] In this paper we shall use the terms "checkpoint" and "state" interchangeably.

[4] We shall use the term "failure" to represent either fault or error, depending on the context.

failures. Even if all failures are covered, there may be some latency between occurrence and detection of a failure under any existing detection mechanism. If checkpointing is performed between a failure occurrence and its detection, the checkpoint saved could be incorrect and, thus, the subsequent rollback recovery following the detection of this failure may become unsuccessful. In such a case, the task has to be restarted from the beginning (i.e., *restart recovery*) if only one checkpoint is saved. This reinforces the fact that an acceptance test is needed to assure the correctness of the checkpoint to be saved. If the acceptance test detects an error or abnormality, then an appropriate error recovery should be initiated. The error coverage of an acceptance test is again less than 100 percent, i.e., imperfect coverage. The imperfect detection coverages of the on-line detection mechanism and the acceptance test imply the existence of a nonzero probability of a task ending with latent errors. The task may or may not produce correct results under latent errors; however, we no longer have any confidence in the results. These results are hence called "unreliable." We quantify this fact by the *probability of an unreliable result, E,* as a measure of *lack of confidence.*

In our models, the *computation time* of a task is an estimate of the time for computing the task under a fault-free situation, and the *execution time* of a task is the time needed to complete the task under the occurrences of faults. The computation time of a task can be determined *a priori* by averaging over repeated tests and validations of the task, and hence is a constant. The computation time has no direct connection with the hard deadline associated with the task. Tasks should be triggered by some mechanism (e.g., real-time clock) in such a way that their hard deadlines will be placed well after worst-case computation times under a fault-free condition. Note, however, that the execution time may vary due to the random occurrences of failure and will thus be treated as a random variable.

There are other aspects of checkpointing (e.g., sensor data and some integrated quantities such as time, velocity, or position) that are not addressed here.[5] These will usually impose an upper limit on the intercheckpoint interval. Further, the placement of checkpoints is dictated by functional and programming considerations as well as the optimization aspects to be discussed in this paper. For the latter, we will consider two parameters of checkpointing: i.e., 1) the duration of checkpointing and 2) the correctness of checkpoints.

III. Problem Statement

Consider a task with a fault-free computation time T and the total execution time w. Define W as the mean execution time and E as the probability of producing an unreliable result at the end of the task. Suppose further that n (yet unknown) checkpoints will be inserted at T_i, $1 \le i \le n$, during task execution. The ith checkpoint is established when the task execution has successfully progressed up to T_i. Define the ith intercheckpoint interval, I_i, $0 \le i \le n$, as the computation time between the ith and $(i+1)$th checkpoints, i.e.,

$$I_i = T_{i+1} - T_i - t_c, \qquad 0 \le i \le n \tag{3.1}$$

where t_c is the checkpointing time, and for consistency, we let $T_0 = 0$ and $T_{n+1} = T$. Using the above definitions, our problem can be stated formally as

Problem P: Minimize W of a task with respect to n and I_i, $0 \le i \le n$, subject to $E \le E_{\text{spec}}$, where E_{spec} is the desired level of confidence in execution results.

Note that P is a realistic problem which arises frequently in the system design process. The constraint on E, E_{spec}, can be viewed as a requirement that must be satisfied in order to achieve the desired level of confidence in the execution result. Also, note that real-time constraints such as hard deadlines, correctness, memory sizes, etc., are included implicitly in P, since 1) the optimal criterion is related to the task execution time, and 2) E_{spec} is derived from both correctness and execution time.

Other variables in our models are defined below. Let t_c be the checkpointing time which is assumed to be fixed and consists of two parts: the time for an acceptance test and the time for saving the current state. An acceptance test is automatically performed at the end of a task, but state saving is not needed following this last acceptance test. The last acceptance test is, however, assumed to take the same time t_c as the regular checkpointing, because the time for the acceptance test makes up the major portion of t_c if a high detection coverage is desired. However, the last checkpoint is not counted in the number of checkpoints. Failures are allowed to occur during normal execution as well as during checkpointing. Whenever a failure is detected during checkpointing, that checkpoint is considered to be invalid and, therefore, not saved. Due to the storage overhead, only the most recent checkpoint is assumed to be saved, i.e., at any time the state saving device can store only one checkpoint. Define

$$\tau_i \equiv I_i + t_c, \qquad 0 \le i \le n. \tag{3.2}$$

Henceforth, intercheckpoint intervals are used to mean either τ_i or I_i depending on the context. Fig. 1 shows a timing diagram of task execution.

On-line detection mechanisms can detect a failure upon its occurrence with the probability $d \in (0, 1]$ or cannot detect the failure at all (i.e., the failure is not covered by the mechanism) with the probability $1 - d$. Those failures undetectable by the on-line detection mechanism can only possibly be detected by the next level of detection, i.e., acceptance tests. If the system contains some latent errors, they will be detected by the acceptance test with a positive probability $c \in (0, 1]$. That is, the coverage of an acceptance test, c, is the conditional probability of detecting the incorrectness of the task state given that there are latent errors in the system.

Suppose a failure occurs and is detected within τ_i of its occurrence, and the system can recover from the failure by either rollback or restart recovery. For rollback recovery, we first restore the system to the most recently saved checkpoint and then resume the task from that point. Let r be a constant representing rollback setup time. The rollback setup time is

[5] This point was brought to the author's attention by an anonymous referee.

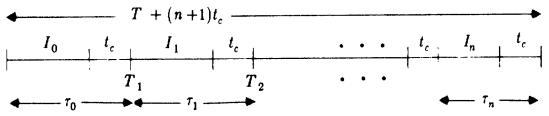


Fig. 1. Timing diagram for checkpointing in a real-time task.

the time measured from failure occurrence to the restoration of the most recent checkpoint. In case of unsuccessful rollback(s), the system will attempt a restart recovery. Let s be the restart setup time which is the mean time from failure occurrence to the restart of the task [including the time spent on unsuccessful rollback(s)]. The assumption of constant t_c and r is made mainly for two reasons: reasonableness and mathematical tractability. In most cases, these quantities do not vary much because the routines for establishing a checkpoint and setting up rollback differ only in the number of variables being saved/checked and this number is rather stabilized in most practical cases.

Rollback recovery may fail due to 1) failure of the communication link to the checkpoint saving device, and 2) failure internal to the checkpoint saving device, thereby making the saved checkpoints inaccessible or incorrect. In these cases, the task has to be recovered by restart.[6] Let p and q, where $p + q = 1$, be the probabilities of recovering a task by rollback and restart, under the condition that no latent failures (to be explained later) exist when the last checkpointing is done. If this condition is not satisfied, we will take a more pessimistic view and assume that restart recovery is inevitable.

Throughout the paper, we assume that arrival of failures is a Poisson process with rate λ. We also assume the system contains no latent failures when it starts a task.[7]

The major difference between the basic and the extended model is in the assumptions of coverages of the acceptance test and the on-line detection mechanism. In the basic model it is assumed that $d = 1$, i.e., detection of a failure coincides with its occurrence. However, in the extended model, no assumption on d and c is made. Note when $d = 1$, the value of c becomes irrelevant, that is, acceptance tests are not needed at all (since all failures can be detected solely by the on-line detection mechanism). The perfect coverage of the on-line detection mechanism implies that 1) there are no latent failures in the system, 2) the task execution results are always correct (i.e., $E = 0$), and 3) each checkpoint is always correct. Hence, in the basic model, the problem P is reduced to finding the solution that minimizes W without any constraint on E. However, it is practically impossible to design a signal-level detection mechanism with perfect coverage. In some cases, we cannot even accurately determine failure coverage. Thus, we have to consider imperfect failure coverages for the design and analysis of a real system.

Consideration of both imperfect coverages and the probability of an unreliable result is more realistic and natural, and is thus a significant departure from previous works in the aforementioned references. Another assumption which is different from existing models in the literature is that the checkpoint saving device is subject to failures so that the saved checkpoints may be destroyed or inaccessible. Therefore, there is a nonzero probability of restart recovery even in the basic model.

[6] As pointed out earlier, previous works [3]–[8] assume that the checkpoint saving device and its link would never fail, and hence rollback recovery is always successful, i.e., no restart recovery is needed.

[7] Relaxation of this assumption for this work is not difficult.

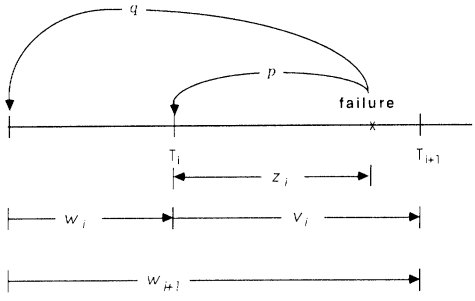


Fig. 2. Graphical explanation for W_i, V_i, and W_{i+1}.

IV. Basic Model

Let w_i, $1 \le i \le n + 1$, be the execution time from the beginning of the task to the first completion of the ith checkpoint, i.e., the end of τ_i, and let $W_i = E(w_i)$. Then, $w = w_{n+1}$ and $W = W_{n+1}$ are the total execution time and the mean execution time of the task, respectively. We shall derive a recursive expression for W in terms of W_i. As shown in Fig. 2, for $0 \le i \le n$ let v_i represent the task execution time in the interval $[T_i, T_{i+1}]$, and z_i be the computation time completed within τ_i, given that a failure occurs during τ_i. If Y represents the interval between two successive failures, then the density function of Y is $f_Y(y) = \lambda e^{-\lambda y}$, $y \ge 0$. The probability of failure occurring during τ_i, $F_i(\tau_i)$, becomes[8]

$$F_i(\tau_i) = \text{Prob}\,[Y \le \tau_i] = 1 - e^{-\lambda \tau_i} \qquad \text{for } 0 \le i \le n. \quad (4.1)$$

By definition, z_i represents the computation done within τ_i under the condition that some failures do occur during τ_i. The density function of z_i can be expressed as

$$f_{z_i}(t) = \frac{f_Y(t)}{F_i(\tau_i)} = \frac{\lambda e^{-\lambda t}}{1 - e^{-\lambda \tau_i}}, \qquad 0 \le t \le \tau_i. \quad (4.2)$$

Hence, the mean of z_i (denoted by $\bar{z}_i$) is calculated as

$$\bar{z}_i = \int_0^{\tau_i} t f_{z_i}(t)\, dt = \frac{1}{\lambda} - \frac{\tau_i e^{-\lambda \tau_i}}{1 - e^{-\lambda \tau_i}}. \quad (4.3)$$

In the interval τ_0, detection of a failure always leads to a restart, and after a task is restarted the process is renewed probabilistically for the variable w_1. Thus,

$$w_1 = \begin{cases} \tau_0 & \text{with probability } 1 - F_0(\tau_0) \\ z_0 + s + w_1 & \text{with probability } F_0(\tau_0). \end{cases} \quad (4.4)$$

The process is also renewed for w_i, $1 \le i \le n$, after rollback and/or restart recovery. Hence,

$$w_{i+1} = w_i + v_i \qquad \text{for } 1 \le i \le n, \quad (4.5)$$

[8] Although the subscript for F is not necessary in the case of Poisson failure process, it is adopted for clarity and extension to a general failure process.

where

$$v_i = \begin{cases} \tau_i & \text{with probability } 1-F_i(\tau_i) \\ z_i + r + v_i & \text{with probability } F_i(\tau_i)p \\ z_i + s + w_{i+1} & \text{with probability } F_i(\tau_i)q. \end{cases} \quad (4.6)$$

From (4.4), (4.5), and (4.6), the following recursive expressions are derived for $1 \le i \le n$.

$$W_1 = \tau_0 + \frac{F_0(\tau_0)}{1-F_0(\tau_0)}(\bar{z}_0 + s) \quad (4.7)$$

$$W_{i+1} = \frac{1-pF_i(\tau_i)}{1-F_i(\tau_i)} W_i + \tau_i + \frac{F_i(\tau_i)}{1-F_i(\tau_i)}(\bar{z}_i + pr + qs). \quad (4.8)$$

Substituting $F_i(\tau_i)$ and $\bar{z}_i$ into (4.7) and (4.8) produces

$$W_1 = \left(\frac{1}{\lambda} + s\right)(e^{\lambda\tau_0} - 1) \quad (4.9)$$

$$W_{i+1} = (qe^{\lambda\tau_i} + p) W_i + (e^{\lambda\tau_i} - 1)\left(\frac{1}{\lambda} + ps + qr\right). \quad (4.10)$$

Applying (4.10) recursively n times, we can get $W = W_{n+1}$:

$$W = hy_0 \prod_{j=1}^{n} u_j + ky_1 \prod_{j=2}^{n} u_j + \cdots + ky_{n-1}u_n + ky_n \quad (4.11)$$

where

$$h = \frac{1}{\lambda} + s, \quad k = \frac{1}{\lambda} + pr + qs$$

$$u_i = qe^{\lambda\tau_i} + p, \quad y_i = e^{\lambda\tau_i} - 1. \quad (4.12)$$

The problem now is to minimize W with respect to n and τ_i, $0 \le i \le n$, subject to

$$\tau_0 + \tau_1 + \cdots + \tau_n = T + (n+1)t_c \equiv T'. \quad (4.13)$$

We shall approach this problem in two steps: first assume n is given and minimize W with respect to τ_i for $0 \le i \le n$, then use the above expressions to minimize W with respect to n. The following theorem provides a solution to the first step.

Theorem 1: For a given n, the minimum W is attained when

$$\tau_0 - \frac{1}{\lambda}\log_e\left(\frac{1+\lambda r}{1+\lambda s}\right) = \tau_1 = \tau_2 = \cdots = \tau_n. \quad (4.14)$$

Proof: Let θ be a Lagrangian multiplier. The optimal solution will then satisfy the differential equations

$$\nabla[W + \theta(T' - \tau_0 - \tau_1 - \cdots - \tau_n)] = 0$$

which is equivalent to

$$\frac{\partial W}{\partial \tau_0} = \frac{\partial W}{\partial \tau_1} = \cdots = \frac{\partial W}{\partial \tau_n} = \theta.$$

Then,

$$\frac{\partial W}{\partial \tau_0} = hy_0' \prod_{j=1}^{n} u_j$$

$$\frac{\partial W}{\partial \tau_1} = hy_0u_1' \prod_{j=2}^{n} u_j + ky_1' \prod_{j=2}^{n} u_j$$

$$\frac{\partial W}{\partial \tau_0} - \frac{\partial W}{\partial \tau_1} = \left(\prod_{j=2}^{n} u_j\right)(hy_0'u_1 - hy_0u_1' - ky_1') = 0.$$

Since $\Pi_{j=2}^{n} u_j = \Pi_{j=2}^{n}(qe^{\lambda\tau j} + p) \ne 0$, it must be true that

$$(hy_0'u_1 - hy_0u_1' - ky_1') = 0. \quad (4.15)$$

Substitute (4.12) for u_i and y_i and then use $y_i' = \lambda e^{\lambda\tau_i}$ and $u_i' = \lambda q e^{\lambda\tau_i}$ to get

$$hpe^{\lambda\tau_0} - (k - hq)e^{\lambda\tau_1} = 0. \quad (4.16)$$

Again, substitute (4.12) for h and k in (4.16) to get

$$\tau_0 = \tau_1 + \frac{1}{\lambda}\log_e\left(\frac{k-hq}{hp}\right) = \tau_1 + \frac{1}{\lambda}\log_e\left(\frac{1+\lambda r}{1+\lambda s}\right). \quad (4.17)$$

This proves the first part of the equality in the theorem. For $1 \le i \le n-1$,

$$\frac{\partial W}{\partial \tau_i} = u_i' hy_0 \prod_{\substack{j=1 \\ j\ne i}}^{n} u_j + u_i' ky_1 \prod_{\substack{j=2 \\ j\ne i}}^{n} u_j + \cdots + u_i' ky_{i-1} \prod_{j=i+1}^{n} u_j + ky_i' \sum_{j=i+1}^{n} u_j$$

$$\frac{\partial W}{\partial \tau_{i+1}} = u_{i+1}' hy_0 \prod_{\substack{j=1 \\ j\ne i+1}}^{n} u_j + u_{i+1}' ky_1 \prod_{\substack{j=2 \\ j\ne i+1}}^{n} u_j + \cdots + u_{i+1}' ky_{i-1} \prod_{j=i+2}^{n} u_j + ky_i' \prod_{j=i+2}^{n} u_j$$

$$\frac{\partial W}{\partial \tau_{i+1}} - \frac{\partial W}{\partial \tau_i} = 0 = (ky_i'u_{i+1} - ky_iu_{i+1}' - ky_{i+1}') \prod_{j=i+2}^{n} u_j + (u_i'u_{i+1} - u_iu_{i+1}')\left(hy_0 \prod_{\substack{j=1 \\ j\ne i,i+1}}^{n} u_j + ky_1 \prod_{\substack{j=2 \\ j\ne i,i+1}}^{n} u_j + \cdots + ky_{i-1} \prod_{j=i+2}^{n} u_j\right).$$

Substituting the expressions for u_i, y_i, u_i', and y_i', in the above equation produces

$$0 = \lambda pq(e^{\lambda\tau_i} - e^{\lambda\tau_{i+1}})\left(hy_0 \prod_{\substack{j=1 \\ j\ne i,i+1}}^{n} u_j + ky_1 \prod_{\substack{j=2 \\ j\ne i,i+1}}^{n} u_j + \cdots + ky_{i-1} \prod_{j=i+2}^{n} u_j\right) + kp\lambda(e^{\lambda\tau_i} - e^{\lambda\tau_{i+1}}) \prod_{j=i+2}^{n} u_j.$$

Since $p > 0$ by assumption,

$$\lambda pq\left(hy_0\prod_{\substack{j=1\\ j\neq i,i+1}}^{n}u_j+ky_1\prod_{\substack{j=2\\ j\neq i,i+1}}^{n}u_j+\cdots+ky_{i-1}\prod_{j=i+2}^{n}u_j\right)+kp\prod_{j=i+2}^{n}u_j>0.$$

It follows that $e^{\lambda\tau_i}-e^{\lambda\tau_{i+1}}=0$ and $\tau_i=\tau_{i+1}$, $1\le i\le n-1$. This then completes the proof. ■

Theorem 1 shows that the optimal intercheckpoint intervals in the basic model are equidistant except the first one. However, this result will only be true under the assumptions of perfect coverage of on-line detection mechanisms and Poisson fault occurrence process. (Theorem 1 "cannot" apply to the more general case to be discussed in the next section.)

To minimize W with respect to n, we express τ_i, $0\le i\le n$, as functions of n

$$\begin{cases}\tau_0=\tau^*+b\\ \tau_i=\tau^*, & 1\le i\le n\end{cases}\tag{4.18}$$

where

$$b=\frac{1}{\lambda}\log_e\left\{\frac{1+\lambda r}{1+\lambda s}\right\},\quad \tau^*=\frac{T-b}{n+1}+t_c.$$

From (4.12) and (4.18), we have, for $1\le i\le n$,

$$\begin{aligned}y_0&=e^{\lambda b}e^{\lambda\tau^*}-1\\ y_i&=y=e^{\lambda\tau^*}-1\\ u_i&=u=qe^{\lambda\tau^*}+p.\end{aligned}\tag{4.19}$$

Hence, (4.11) becomes

$$\begin{aligned}W&=hy_0\prod_{j=1}^{n}u+ky\prod_{j=2}^{n}u+\cdots+kyu+ky\\ &=hy_0u^n+ky\sum_{m=0}^{n-1}u^m.\end{aligned}\tag{4.20}$$

If $q=0$, $u=p=1$, while $u>1$ if $q>0$. Therefore,

$$W=\begin{cases}hy_0+nky & \text{if } q=0\\ u^n(kq^{-1}+hy_0)-kq^{-1} & \text{if } q>0.\end{cases}\tag{4.21}$$

Note that u, y, and y_0 are all functions of n, an integer to be determined. Although it is not possible to derive an explicit expression for the optimal n, the optimal value can be obtained by solving the equation $dW/dn=0$, and comparing the value of W to the two nearest integers of this solution.

Some numerical examples are shown in Figs. 3–5, where W is plotted as a function of n using (4.21). The unit of time-related variables is hour or per hour. Fig. 3 compares the curves for different values of the checkpointing time t_c. It is observed that a smaller t_c usually requires more checkpoints to attain the minimum W. The shape of the curve changes dramatically when t_c is changed. The curve for $t_c=0.5$ almost flattens out when $n>5$, while the curve for $t_c=2.0$ rises sharply beyond the minimum point. This shows that if it takes more time for checkpointing, the mean execution time becomes more sensitive to the number of checkpoints.

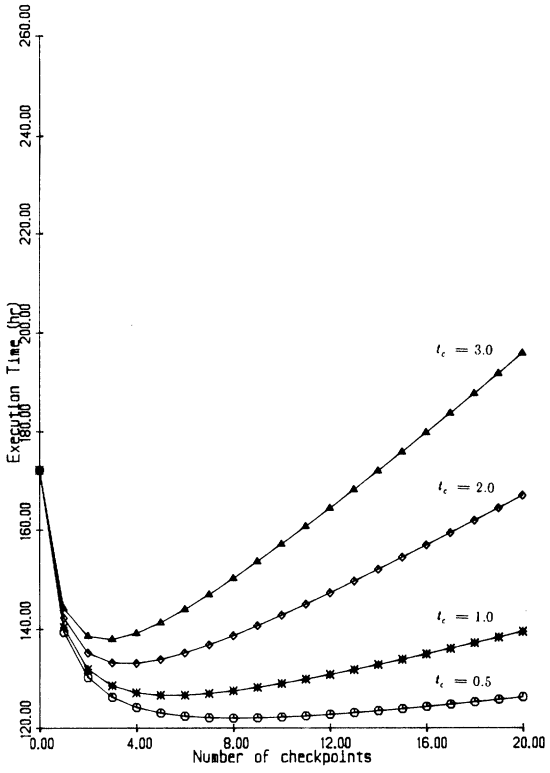


Fig. 3. W versus n when $\lambda=0.01$, $r=0.2$, $s=0.5$, $p=0.8$, $T=100$.

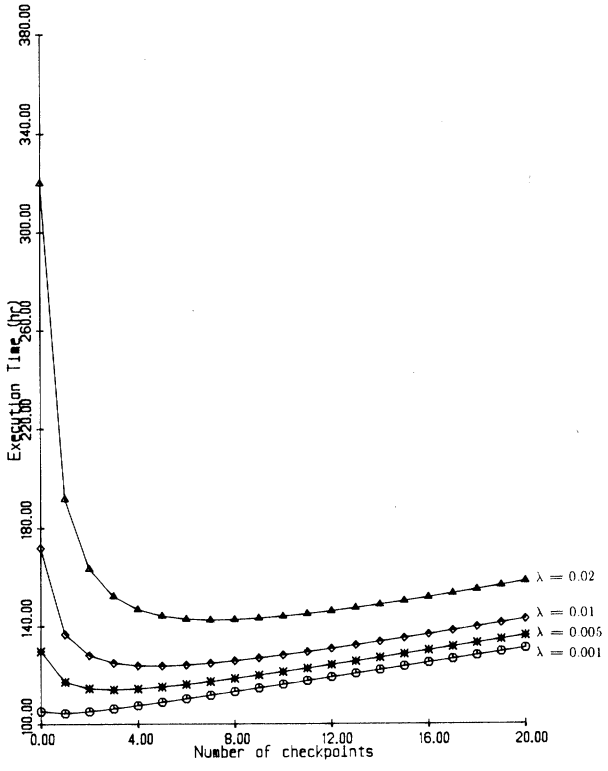


Fig. 4. W versus n when $r=0.1$, $s=0.3$, $t_c=1.5$, $p=0.9$, $T=100$.

In Fig. 4, λ is changed from 0.001 to 0.2 which translates into changing from one failure every 10 tasks to 20 failures per

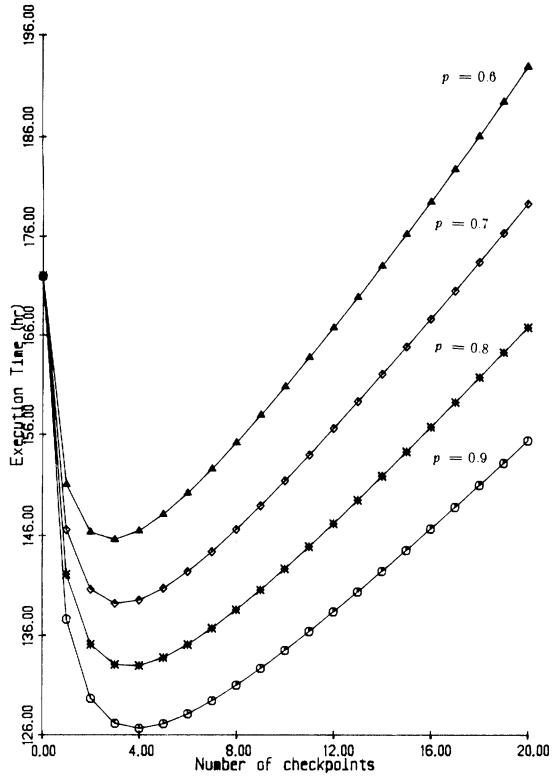


Fig. 5. W versus n when $\lambda = 0.01$, $r = 0.1$, $s = 0.3$, $t_c = 2.0$, $T = 100$.

task (when $T = 100$ is assumed for all tasks). When the failure rate is very low (such as $\lambda = 0.001$) the overhead of checkpointing will offset the saved execution time. But if failures occur more often (e.g., $\lambda = 0.2$), having one or more checkpoints can greatly reduce the execution time. In Fig. 5, the curves for different p are shown. It is clear that a lower probability of rollback requires fewer checkpoints.

Generally, more checkpoints are required to achieve minimum W when the failure rate λ is high, the checkpointing time t_c is small, and the rollback probability p is high.

V. Extended Model

The extended model takes into consideration more realistic coverages of both the on-line detection mechanism and the acceptance test. The constraint $E \leq E_{\text{spec}}$ in problem P now plays an important role, since, unlike in the basic model, $E \neq 0$. Again, closed-form expressions for the optimal intercheckpoint intervals and the number of checkpoints cannot be obtained because of their inherent complexity. However, a computation algorithm will be derived to determine approximate optimal solutions.

Let E_i, $0 \leq i \leq n$, denote the probability of an unreliable result when the task has progressed to the beginning of the interval τ_i. Let $E \equiv E_{n+1}$ and $F_j \equiv F_j(\tau_j)$, $0 \leq j \leq n$. Then, it is obvious that $E_0 = 0$ and

$$E_1 = F_0(1-d)(1-c) + F_0[d+(1-d)c]E_1. \tag{5.1}$$

In (5.1), the first term represents the case when failures occur in τ_0 but detected neither by the on-line detection mechanism nor by the acceptance test; the second term represents the case when a failure occurs and is detected within τ_0, triggering a restart recovery. The process of task execution renews after each restart recovery. Define the *combined failure coverage* $D \equiv d + (1-d)c$. Then, (5.1) becomes

$$E_1 = \frac{F_0(1-d)(1-c)}{1-F_0D}. \tag{5.2}$$

The derivation of E_j for $j > 1$ is similar to but more complicated than that of E_1. Let G_j be the conditional probability of an unreliable result at the end of τ_j, given that the system is free of failure at the beginning of τ_j. Following arguments similar to those for E_1, we can get

$$G_j = F_j(1-d)(1-c) + F_jD(pG_j + qE_{j+1})$$

where the process of task execution renews after a rollback or restart recovery. Simplifying the above expression yields

$$G_j = \frac{F_j(1-d)(1-c)}{1-pF_jD} + \frac{qF_jD}{1-pF_jD}E_{j+1}. \tag{5.3}$$

E_{j+1} can be calculated as the probability of sum of two events: 1) no latent failure exists at the beginning of τ_j and 2) some latent failures exist at the beginning of τ_j. Thus,

$$E_{j+1} = (1-E_j)G_j + E_j[(1-F_j)+F_j(1-d)](1-c) + E_j(1-F_j)cE_{j+1} + E_jF_jDE_{j+1}.$$

Replace G_j with (5.3) to get

$$E_{j+1} = \frac{F_j(1-d)(1-c) + (1-F_j)(1-c)E_j - (1-c)(1-F_jd)pF_jDE_j}{1-F_jD-c(1-F_j)E_j-(1-c)(1-F_jd)pF_jDE_j} \tag{5.4}$$

where $0 \leq j \leq n$.

We now derive an expression for the mean execution time W. A task has to be restarted with probability one whenever failures are detected in the interval τ_0, rollback is the same as restart in this case. For subsequent intervals τ_j, $j > 0$, the task may be free of failures with probability $1 - E_j$ or may have latent failures with probability E_j at the beginning of τ_j. For the former case, the last checkpoint must be correct at the time of its establishment, so the task can roll back or restart with probability p and q, respectively.

For the latter case, the last checkpoint may be correct or incorrect depending on whether or not the latent faults have induced error(s). In such a case, we assume that the last checkpoint is incorrect. (This is to err on the safe side.) If the system does detect a latent error[9] within τ_i either by the on-line detection mechanism or the acceptance test, it will roll back to the last checkpoint, resume execution, and then detect the

[9] Of course, at this point the system does not know whether it is latent or not.

same error again. The system may roll back several more times to see if the same occurs. Then, the last checkpoint can be declared to be incorrect and a restart recovery follows. Determining the number of rollbacks before restart is an issue of its own. As mentioned earlier, the time spent on unsuccessful rollback(s) is included in the restart setup time s. The expression for the mean execution time W can be expressed as follows:

$$\begin{aligned} W_1 &= (1-F_0)\tau_0 + F_0 d(s+\bar{z}_0+W_1) \\ &\quad + F_0(1-d)c(s+\tau_0+W_1) + F_0(1-d)(1-c)\tau_0 \\ &= (1-F_0 d)\tau_0 + F_0 d\bar{z}_0 + F_0 D(s+W_1). \end{aligned}$$

For $1 \le j \le n$,

$$W_{j+1} = W_j + V_j$$

$$\begin{aligned} V_j &= (1-E_j)[(1-F_j)+F_j(1-d)(1-c)]\tau_j \\ &\quad + (1-E_j)F_j d[p(r+\bar{z}_j+V_j)+q(s+\bar{z}_j+W_{j+1})] \\ &\quad + (1-E_j)F_j(1-d)c[p(r+\tau_j+V_j)+q(s+\tau_j+W_{j+1})] \\ &\quad + E_j F_j d(s+\bar{z}_j+W_{j+1}) \\ &\quad + E_j[F_j(1-d)+(1-F_j)]c(s+\tau_j+W_{j+1}) \\ &\quad + E_j[(1-F_j)(1-c)+F_j(1-d)(1-c)]\tau_j. \end{aligned}$$

After simplification, we get

Substituting (5.9) into (5.5) and (5.6) gives

$$W_1 = \frac{1-d}{1-F_0 D}\tau_0 + \frac{F_0 d}{1-F_0 D}\frac{1}{\lambda} + \frac{F_0 D}{1-F_0 D}s \tag{5.10}$$

$$\begin{aligned} W_{j+1} &= \frac{1-d}{1-p_j-q_j}\tau_j + \frac{F_j d}{1-p_j-q_j}\frac{1}{\lambda} \\ &\quad + \frac{p_j r+q_j s}{1-p_j-q_j} + \frac{1-p_j}{1-p_j-q_j}W_j. \end{aligned} \tag{5.11}$$

For a fixed n, it is clear from (5.4) and (5.11) that both E_{n+1} and W_{n+1} are functions of the $(n+1)$ variables, τ_i, $0 \le i \le n$. However, because of the constraint (4.13), one of the variables is dependent on the other variables. The choice of the dependent variable is arbitrary. In the discussion that follows, we will restrict ourselves to the dependency between any two variables. This can, of course, be easily generalized to the dependency between $(n+1)$ variables by a simple induction.

Definition: Define an operator ∇_{ij}, $0 \le i < j \le n$, on E_{n+1} and W_{n+1} as follows:

$$\nabla_{ij}(X) \equiv \lim_{\delta \to 0^+} \cdot \frac{X(\tau_0, \cdots, \tau_i-\delta, \cdots, \tau_j+\delta, \cdots, \tau_n) - X(\tau_0, \cdots, \tau_n)}{\delta}$$

where X is E_{n+1} or W_{n+1}.

Then, $\nabla_{ij}(E_{n+1})$ can be calculated by

$$\begin{aligned} \nabla_{ij}(E_{n+1}) &= \lim_{\delta \to 0^+} \frac{E_{n+1}(\tau_0, \cdots, \tau_i-\delta, \cdots, \tau_j+\delta, \cdots, \tau_n) - E_{n+1}(\tau_0, \cdots, \tau_i-\delta, \cdots, \tau_n)}{\delta} \\ &\quad - \lim_{\delta \to 0^+} \frac{E_{n+1}(\tau_0, \cdots, \tau_n) - E_{n+1}(\tau_0, \cdots, \tau_i-\delta, \cdots, \tau_n)}{\delta} \\ &= \frac{\partial E_{i+1}}{\partial \tau_j} - \frac{\partial E_{i+1}}{\partial \tau_i}. \end{aligned} \tag{5.12}$$

$$W_1 = \frac{1-F_0 d}{1-F_0 D}\tau_0 + \frac{F_0 d}{1-F_0 D}\bar{z}_0 + \frac{F_0 D}{1-F_0 D}s \tag{5.5}$$

$$\begin{aligned} W_{j+1} &= \frac{1-F_j d}{1-p_j-q_j}\tau_j + \frac{F_j d}{1-p_j-q_j}\bar{z}_j \\ &\quad + \frac{p_j r+q_j s}{1-p_j-q_j} + \frac{1-p_j}{1-p_j-q_j}W_j \end{aligned} \tag{5.6}$$

where p_j and q_j are defined as

$$p_j = p(1-E_j)DF_j \tag{5.7}$$

$$q_j = q(1-E_j)DF_j + E_j[c+(1-c)F_j d]. \tag{5.8}$$

From (4.3), $\bar{z}_j$ can be expressed as

$$\bar{z}_j = \frac{1}{\lambda} - \frac{\tau_j e^{-\lambda\tau_j}}{1-e^{-\lambda\tau_j}} = \frac{1}{\lambda} - \frac{1-F_j}{F_j}\tau_j. \tag{5.9}$$

Some important properties about E_{n+1} and W_{n+1} will be stated and proved below in Theorems 2 and 3. Based on these theorems, a numerical algorithm will be derived to obtain an approximate solution to the problem P.

Theorem 2: For any pair of integers i and j, $0 \le i < j \le n$, the inequality $\nabla_{ij}(E_{n+1}) > 0$ holds if the following conditions are true.

(C1) $E_k \approx 0$, $F_k \ll 1$, for $0 \le k \le n+1$,

(C2) $d(1+p) > 1$.

Proof: From the recursive formula of (5.4), E_{n+1} can be viewed as a function of τ_n and E_n, and E_n as a function of τ_{n-1} and E_{n-1}, and so on. Applying the chain rule in (5.12), we get

$$\begin{aligned} \nabla_{ij}(E_{n+1}) &= \frac{\partial E_{n+1}}{\partial E_n}\frac{\partial E_n}{\partial E_{n-1}}\cdots\frac{\partial E_{j+2}}{\partial E_{j+1}} \\ &\quad \cdot \left[\frac{\partial E_{j+1}}{\partial \tau_j} - \frac{\partial E_{j+1}}{\partial E_j}\cdots\frac{\partial E_{i+2}}{\partial E_{i+1}}\frac{\partial E_{i+1}}{\partial \tau_i}\right]. \end{aligned} \tag{5.13}$$

For $0 \le k \le n$, the following expressions can be derived.

$$\frac{\partial E_{k+1}}{\partial E_k} = \frac{(1-c)(1-F_k)(1-F_k d)(1-pF_k D)}{[1-F_k D - c(1-F_k)E_k - (1-c)(1-F_k d)pF_k DE_k]^2} \tag{5.14}$$

$$\frac{\partial E_{k+1}}{\partial \tau_k} = \frac{\lambda(1-F_k)(1-c)(1-d)\,[1-E_k(1-cF_j)]}{[1-F_k D - c(1-F_k)E_k - (1-c)(1-F_k d)pF_k DE_k]^2} - \frac{pDE_k(1-2F_k d)(1-F_k)(1-E_k)}{[1-F_k D - c(1-F_k)E_k - (1-c)(1-F_k d)pF_k DE_k]^2}. \tag{5.15}$$

Note that the fact $\partial F_k/\partial \tau_k = \lambda e^{-\lambda \tau_k} = \lambda(1 - F_k)$ has been used in deriving (5.14) and (5.15). In (5.14), $\partial F_{k+1}/\partial E_k > 0$ for all k, $0 \le k \le n$, since c, d, D, p, and F_k are all less than 1. Thus, from (5.13) the theorem will have been proved if

$$\frac{\partial E_{j+1}}{\partial \tau_j} - \frac{\partial E_{j+1}}{\partial E_j} \cdots \frac{\partial E_{i+2}}{\partial E_{i+1}} \frac{\partial E_{i+1}}{\partial \tau_i} > 0. \tag{5.16}$$

Condition C1 is used to approximate the denominators of both (5.14) and (5.15) as $(1 - F_k D)^2$. Further approximation on the numerator of (5.15) then leads to

$$\frac{\partial E_{k+1}}{\partial \tau_k} \approx \frac{\lambda(1-F_k)(1-c)(1-d)}{(1-F_k D)^2}.$$

This results in

$$\begin{aligned}
&\frac{\partial E_{j+1}}{\partial \tau_j} - \frac{\partial E_{j+1}}{\partial E_j} \cdots \frac{\partial E_{i+2}}{\partial E_{i+1}} \frac{\partial E_{i+1}}{\partial \tau_i} \\
&= \frac{\lambda(1-F_j)(1-c)(1-d)}{(1-F_j D)^2} - \frac{\lambda(1-F_i)(1-c)(1-d)}{(1-F_i D)^2} \\
&\quad \cdot \left\{ \prod_{k=i+1}^{j} \frac{(1-c)(1-F_k)(1-F_k d)(1-pF_k D)}{(1-F_k D)^2} \right\} \\
&= \frac{\lambda(1-F_j)(1-c)(1-d)}{(1-F_j D)^2} \left\{ 1 - \frac{(1-F_j d)(1-pF_j D)}{(1-F_i d)(1-pF_i D)} \right. \\
&\quad \left. \cdot \prod_{k=i}^{j-1} \frac{(1-c)(1-F_k)(1-F_k d)(1-pF_k D)}{(1-F_k D)^2} \right\}.
\end{aligned}$$

The inequality (5.16) holds if

$$\frac{(1-F_j d)(1-pF_j D)}{(1-F_i d)(1-pF_i D)} \cdot \prod_{k=i}^{j-1} \frac{(1-c)(1-F_k)(1-F_k d)(1-pF_k D)}{(1-F_k D)^2} < 1,$$

the left-hand side of which can also be expressed as

$$(1-c)^{j-i} \left\{ \prod_{k=i+1}^{j-1} \frac{(1-pF_k D)(1-F_k d)(1-F_k)}{(1-F_k D)^2} \right\} \cdot \frac{(1-pF_j D)(1-F_j d)(1-F_i)}{(1-F_i D)^2}. \tag{5.17}$$

Since $F_i, F_j \ll 1$ by C1, the value of the last factor in (5.17) is approximately 1, and hence (5.17) is dominated by the other remaining factors. It is easy to show that $[(1 - pF_k D)(1 - F_k d)(1 - F_k)]/(1 - F_k D)^2] < 1$ for all $0 \le k \le n + 1$ if $d(1 + p) > 1$. Hence, the fact $(1 - c)^{j-i} < 1$ makes the whole product less than 1. ■

Usually the values of E_k must be very small for any real-time system. This is the rationale behind the conditions C1 and C2 of Theorem 2. To produce a low probability of having an unreliable result, the system requires a reasonably high fault coverage and a high probability of rollback when failures occur (i.e., $d(1 + p) > 1$), and a low failure occurrence rate (thus, $F_k \ll 1$). If the system cannot meet C1 and C2, the probability of an unreliable result will be high. In that case, checkpointing is not a useful technique at all to improve the system's reliability, and it would be better to employ other schemes such as triplicated voting.

Theorem 2 states that if the length of an interval τ_j is increased at the expense of decreasing a preceding interval τ_i, $i < j$, E_{n+1} will always increase. Consequently, E_{n+1} can be decreased by stretching earlier intervals against later intervals. When two adjacent intervals are considered, stretching the earlier interval means delaying the establishment of the checkpoint in between. That is, E_{n+1} can be reduced by moving any checkpoints to the right on the time axis in Fig. 1. Theorem 2 also verifies the fact that if all the checkpoints are inserted near the end of the task, the execution result will become very reliable, since the task has to pass all the acceptance tests near the end of the task.

Theorem 3: For any pair of integers i and j, $0 \le i < j \le n$, $\nabla_{ij}^2(W_{n+1}) \equiv \nabla_{ij}(\nabla_{ij}(W_{n+1})) > 0$ if C1 and the following condition hold: (C3) $Dp > 0.5$.

Proof: First simplify the recursive formula of (5.11) by using C1 to get

$$\begin{aligned}
W_{j+1} &= \frac{(1-d)\tau_j}{1-F_j D} + \frac{F_j d\lambda^{-1}}{1-F_j D} + \frac{prF_j D + qsF_j D}{1-F_j D} + \frac{1-pF_j D}{1-F_j D} W_j \\
&= (1-d)\frac{\tau_j}{1-F_j D} + (d\lambda^{-1} + prD + qsD + qDW_j) \cdot \frac{F_j}{1-F_j D} + W_j.
\end{aligned} \tag{5.18}$$

In (5.18), W_{j+1} can be viewed as a function of τ_j and W_j, and hence for all j,

$$\frac{\partial W_{j+1}}{\partial W_j} = \frac{1-pF_j D}{1-F_j D} \tag{5.19}$$

$$\frac{\partial W_{j+1}}{\partial \tau_j} = \frac{A_j}{(1-F_j D)^2} - \frac{B_j F_j}{(1-F_j D)^2} + \frac{C(1-F_j)\tau_j}{(1-F_j D)^2} \tag{5.20}$$

where

$$A_j = 1 + \lambda prD + \lambda qsD + \lambda qDW_j \tag{5.21}$$

$$B_j = d + (1-d)D + \lambda prD + \lambda qsD + \lambda qDW_j \quad (5.22)$$

$$C = (1-d)D\lambda. \quad (5.23)$$

Now W_{n+1} is a function of τ_n and W_n, W_n is a function of τ_{n-1} and W_{n-1}, and so on. Using the chain rule and (5.12) as in the proof of Theorem 2, we get

$$\nabla_{ij}(W_{n+1}) = \frac{\partial W_{n+1}}{\partial W_n}\frac{\partial W_n}{\partial W_{n-1}}\cdots\frac{\partial W_{j+2}}{\partial W_{j+1}} \cdot \left[\frac{\partial W_{j+1}}{\partial \tau_j} - \frac{\partial W_{j+1}}{\partial W_j}\cdots\frac{\partial W_{i+2}}{\partial W_{i+1}}\frac{\partial W_{i+1}}{\partial \tau_i}\right]$$
$$= H_1\left[\frac{\partial W_{j+1}}{\partial \tau_j} - \frac{1-pF_jD}{1-F_jD}H_2\frac{\partial W_{i+1}}{\partial \tau_i}\right] \quad (5.24)$$

where

$$H_1 = \prod_{k=j+1}^{n}\left(\frac{1-pF_kD}{1-F_kD}\right) \quad (5.25)$$

$$H_2 = \prod_{k=i+1}^{j-1}\left(\frac{1-pF_kD}{1-F_kD}\right) \quad (5.26)$$

are functions independent of τ_i and τ_j. It is easy to see that $H_1 > 0$ and $H_2 > 0$, since p, F_k, and D are all less than 1.

It can be derived from (5.24) that

$$\nabla_{ij}^2(W_{n+1}) = H_1\frac{\partial^2 W_{j+1}}{\partial \tau_j^2} - 2H_1\lambda qD\frac{1-F_j}{(1-F_jD)^2}H_2\frac{\partial W_{i+1}}{\partial \tau_i} + H_1\frac{1-pF_jD}{1-F_jD}H_2\frac{\partial^2 W_{i+1}}{\partial \tau_i^2}$$
$$= H_1\frac{\partial^2 W_{j+1}}{\partial \tau_j^2} + H_1H_2\left[\frac{1-pF_jD}{1-F_jD}\frac{\partial^2 W_{i+1}}{\partial \tau_i^2} - \frac{2\lambda qD(1-F_j)}{(1-F_jD)^2}\frac{\partial W_{i+1}}{\partial \tau_i}\right]. \quad (5.27)$$

Note that $A_j > B_j$ [both defined in (5.21) and (5.22)] for $0 \le j \le n$. So from (5.20), we get

$$\frac{\partial^2 W_{j+1}}{\partial \tau_j^2} = \frac{\lambda(1-F_j)}{(1-F_jD)^3}[2DA_j - (1+F_jD)B_j + (2D-1-F_jD)\tau_jC]$$
$$> \frac{\lambda(1-F_j)}{(1-F_jD)^3}[2DA_j - (1+F_jD)A_j + (2D-1-F_jD)\tau_jC]$$
$$= \frac{\lambda(1-F_j)}{(1-F_jD)^3}(A_j + C)(2D-1-F_jD) > 0 \quad \text{if } 2D-1-F_jD > 0. \quad (5.28)$$

Plugging (5.28) into (5.27) and examining the terms inside the brackets of (5.27), we get

$$\frac{1-pF_jD}{1-F_jD}\frac{\partial^2 W_{i+1}}{\partial \tau_i^2} - 2\lambda qD\frac{1-F_j}{(1-F_jD)^2}\frac{\partial W_{i+1}}{\partial \tau_i} = \frac{\lambda}{(1-F_iD)^3(1-F_jD)^2}. \quad (5.29)$$

$$\{(1-pF_jD)(1-F_jD)(1-F_i)[2DA_i - (1+F_jD)B_i + (2D-1-F_jD)\tau_jC] - 2qD(1-F_iD)(1-F_j)\cdot[A_i - F_iB_i + (1-F_i)\tau_iC]\}$$
$$> \frac{\lambda(1-F_i)(A_i+\tau_iC)}{(1-F_iD)^3(1-F_jD)^2}[(1-pF_jD)(1-F_jD)\cdot(2D-1-F_iD) - 2qD(1-F_iD)(1-F_j)]$$
$$\approx \frac{\lambda(1-F_i)(A_i+\tau_iC)}{(1-F_iD)^3(1-F_jD)^2}(2D-1-2qD) > 0 \quad \text{if } 2D-1-2qD > 0.$$

Note that $2D - 1 - 2qD > 0$ and $2D - 1 - F_jD > 0$, if C3 holds. Hence, from (5.27), (5.28), and (5.29), $\nabla_{ij}(\nabla_{ij}(W_{n+1})) > 0$. ■

Again, to make any checkpointing meaningful, the system must have $D > 0.7$ and $p > 0.7$, thereby satisfying the condition C3.[10]

By applying Theorem 3 to every pair of intervals, it is possible (although very time consuming) to find the global minimum of W_{n+1}. However, even if the minimum of W_{n+1} is found, the interval combination which yields this minimum may not lead to the probability of an unreliable result satisfying the constraint $E_{n+1} \le E_{\text{spec}}$. Therefore, the following algorithm is proposed to solve problem P.

Algorithm A:

A1. Set $n := 1$.
If $E_1(T + t_c) \le E_{\text{spec}}$ *then* set $W_{\min} := W_0(T + t_c)$ *else* set $W_{\min}$ to a large value.

A2. Construct a finite sequence of interval vectors $(\tau_0^k, \cdots, \tau_n^k)$, $k = 1, \cdots, K$ by a systematic way such that $E_{n+1}(\tau_0^k, \cdots, \tau_n^k) > E_{n+1}(\tau_0^{k+1}, \cdots, \tau_n^{k+1})$ for $1 \le k \le K$. Such a sequence is called a *search path*.

A3. Find κ such that

$$E_{n+1}(\tau_0^\kappa, \cdots, \tau_n^\kappa) \ge E_{\text{spec}} > E_{n+1}(\tau_0^{\kappa+1}, \cdots, \tau_n^{\kappa+1}).$$

If no such κ can be found then set $n := n + 1$ and go to A2.

A4. Find $W_{n+1}^*(\tau_0^*, \cdots, \tau_n^*) = \min_{\kappa \le k \le K} W_{n+1}(\tau_0^k, \cdots, \tau_n^k)$

A5. *If* $W_{n+1}^* < W_{\min}$ *then* set $W_{\min} := W_{n+1}^*$, $n := n + 1$, and go to A2 *else* stop.

Algorithm A starts by calculating E and W for the case of no-checkpointing, i.e., $E_1(T + t_c)$ and $W_0(T + t_c)$. If $E \le$

[10] Otherwise, checkpointing should not be used as mentioned earlier.

E_{spec}, no-checkpointing is a legitimate candidate, so W_{min} is set to $W_0(T + t_c)$. If $E > E_{spec}$, no-checkpointing cannot be a solution of problem P, and W_{min} is set to an arbitrary large value to indicate that no solution has been found yet. Steps A2–A4 will find a solution of problem P if it exists for each given n, the number of checkpoints. If no solution exists for a given n, the algorithm repeats those steps after incrementing n. Since the task execution result becomes increasingly trustable as the number of checkpoints increases, for any E_{spec} there is always an integer m such that $E_{n+1} \leq E_{spec}$ for all $n \geq m$. This guarantees the existence of κ in step A2 for some n. The algorithm terminates in step A5 when there is an n such that $W^*_{n+1} \geq W_{min}$. This terminating condition must hold eventually, since the checkpointing overhead increases linearly with n while the saving of execution time from rollback recovery is limited.

The search path needed in step A2 can be determined by Theorem 2. We can use any search path that has the effect of moving one or more checkpoints to the right in the time axis while increasing k, since Theorem 2 has shown that E_{n+1} on such a path is decreasing. There are many ways of constructing such a search path. In practice, the choice of a search path depends heavily on the convenience of checkpoint implementation and/or the physical limitation in a particular system. Two simple approaches of constructing search paths are conceivable: the *common ration approach* and the *common difference approach*. The common ratio approach considers intervals with the relation $\tau_{j+1} = \rho\tau_j$, $0 \leq j \leq n$, where ρ is a constant ratio between two adjacent intervals. The search path is obtained by decreasing ρ in discrete steps within some given range, e.g., $0.8 \leq \rho \leq 1.2$. Similarly, the common difference approach considers intervals with the relation $\tau_{j+1} = \tau_j - \delta$, $0 \leq j \leq n$, where δ is the common difference. The search path is obtained by decreasing δ in discrete steps within some given range, e.g., $-4 \leq \delta \leq 4$. The range of the common ratio or the common difference is determined primarily by the physical limitation of the system.

In step A4, W^*_{n+1} is obtained once a local minimum is found, since Theorem 3 has shown that W_{n+1} is concave with respect to the operator ∇_{ij}. If no local minimum is found, the minimum would occur either at $k = \kappa$ or $k = K$.

Some examples are shown in Figs. 6–8 using the common ratio approach. The solid line represents the curve for W_{n+1} and the dashed line represents the curve for E_{n+1}. It is observed that if the failure coverages are high (Fig. 6), the minimum W occurs around $\rho = 1.0$, while if the failure coverages are low (Fig. 7), the minimum W occurs at $\rho > 1.0$. This result is expected since for high failure coverages, the extended model will be close to the basic model where the optimal intercheckpoint interval is equidistant, and for low failure coverages, the probability of restart is high so that more frequent checkpointing at the beginning of the task and less frequent checkpointing near the end of the task are required to reduce the time wasted in restart recovery. The following procedure is derived immediately from Algorithm A to get the optimal solution to P using the common ratio approach once the graphs similar to those in Figs. 6–8 have been obtained.

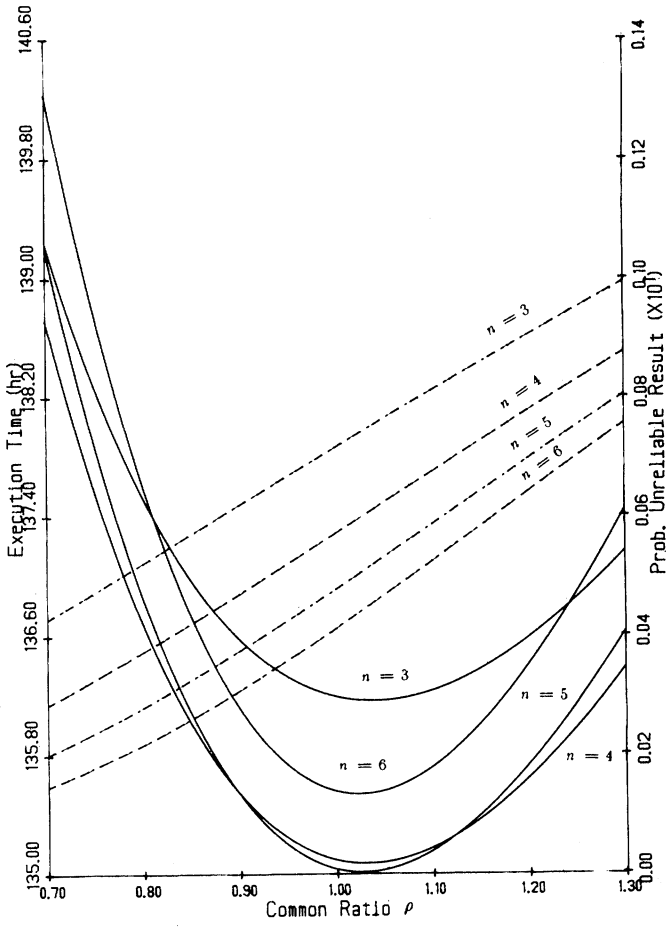


Fig. 6. Diagram of W and E versus common ratio when $c = 0.8$, $d = 0.9$, $p = 0.8$, $\lambda = 0.01$, $r = 0.4$, $s = 0.7$, $t_c = 1.5$, $T = 100$.

S1. Draw a horizontal line $E_{n+1} = E_{spec}$ on the graph.
S2. The intersecting points of this line with the dashed lines give the maximum allowable common ratios for different values of n.
S3. For each n, find the minimum W_{n+1} in the region of the allowable common ratios.
S4. Choose the value of n which yields the minimum W_{n+1}.

Two examples below illustrate the use of this procedure. Example 1 uses Fig. 6, while using Fig. 7 for Example 2.

Example 1: Let $T = 100$ h, $\lambda = 0.01$ per hour, $c = 0.8$, $d = 0.9$, $r = 0.4$ h, $s = 0.7$ h, $p = 0.8$, $q = 0.2$, and $t_c = 1.5$ h. Three cases are considered, i.e., $E_{spec} = 0.002$, 0.003, 0.005 for $1 \leq n \leq 11$. The results are given in Tables I–III.

From Table I, the minimum W with $E_{spec} = 0.002$ occurs when $n = 7$ and $\rho = 0.83$. Similarly, the smallest W with $E_{spec} = 0.003$ occurs when $n = 6$ and $\rho = 0.89$, and the smallest W with $E_{spec} = 0.005$ occurs when $n = 5$ and $\rho = 1.02$. We can see that a lower E_{spec} will produce a solution with more checkpoints and a smaller ratio.

Example 2: Let $T = 100$ h, $\lambda = 0.01$ per hour, $c = 0.6$, $d = 0.7$, $r = 0.4$ h, $s = 0.7$ h, $p = 0.8$, $q = 0.2$, and $t_c = 1.5$ h. Two possible values of E_{spec} are considered, i.e., 0.02, 0.04 for $1 \leq n \leq 11$. The results are given in Tables IV and V.

The smallest W with $E_{spec} = 0.02$ occurs when $n = 7$ and ρ

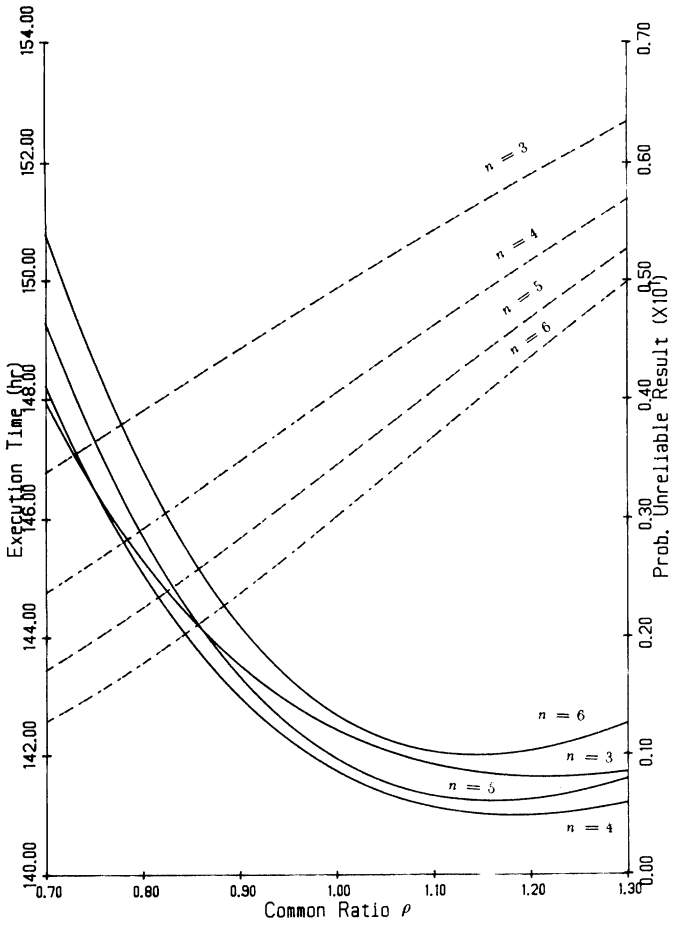


Fig. 7. Diagram of W and E versus common ratio when $c = 0.6$, $d = 0.7$, $p = 0.8$, $\lambda = 0.01$, $r = 0.4$, $s = 0.7$, $t_c = 1.5$, $T = 100$.

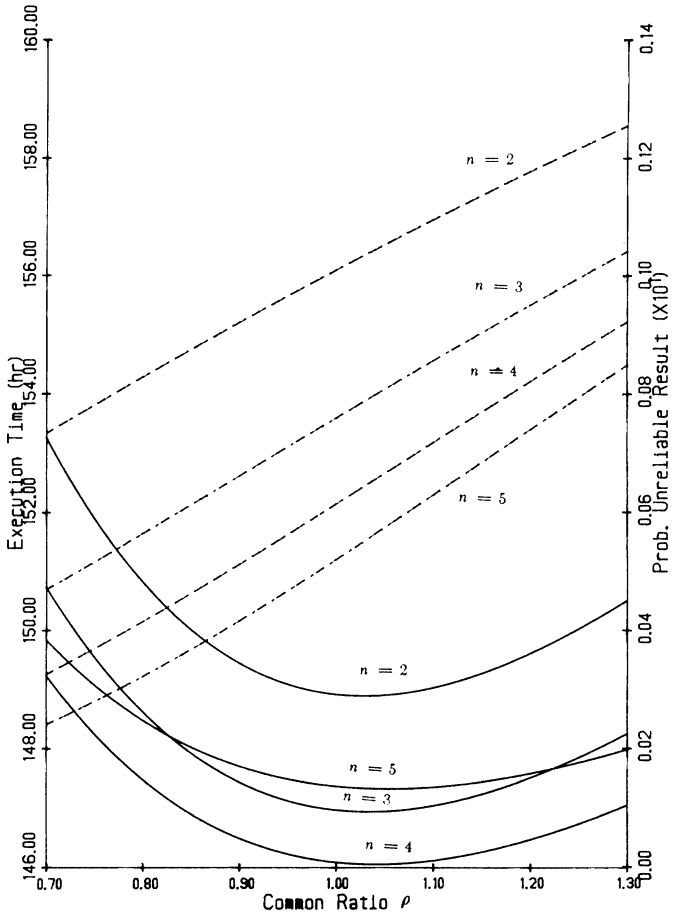


Fig. 8. Diagram of W and E versus common ratio when $c = 0.8$, $d = 0.9$, $p = 0.8$, $\lambda = 0.01$, $r = 0.4$, $s = 0.7$, $t_c = 3.0$, $T = 100$.

$= 0.89$, but that with $E_{spec} = 0.04$ occurs when $n = 5$ and $\rho = 1.08$. One can see again that a lower E_{spec} will produce a solution with more checkpoints and a smaller ratio. This implies that a more reliable system requires more checkpoints and must perform checkpointing more frequently towards the end of the task.

Note that the checkpointing time t_c is related to the coverage of acceptance tests; high coverage will require longer t_c. Although we did not assume any relation between them for our analysis, the ratio, t_c/T, in all examples is made to lie in the range of 1 percent to represent the situation of short checkpointing time with moderate coverage. This represents a more practical case than the one of long checkpointing time with high coverage.

VI. Conclusion

Taking a task-oriented view, we have developed the basic and extended models to design and evaluate the performance of checkpointing of real-time tasks. The performance criterion used in this paper is the real (mean) execution time W for a task subject to the specified probability of an unreliable result E at the completion of the task.

The basic model hinges on the assumption of a perfect coverage of the on-line detection mechanism, i.e., any failure will be detected upon its occurrence. An optimal checkpointing strategy under this setting is shown to be equidistant for

TABLE I
OPTIMUM SOLUTIONS FOR EXAMPLE 1 WITH $E_{spec} = 0.002$.

n	ratio	minimum W	E
2	0.11	175.798315	0.001970
3	0.42	149.987954	0.001996
4	0.58	142.656881	0.001937
5	0.69	139.498282	0.001928
6	0.77	138.267395	0.001947
7	0.83	138.096742	0.001977
8	0.87	138.659892	0.001960
9	0.90	139.601276	0.001940
10	0.93	140.717164	0.001983
11	0.95	142.076324	0.001981

TABLE II
OPTIMUM SOLUTIONS FOR EXAMPLE 1 WITH $E_{spec} = 0.003$.

n	ratio	minimum W	E
2	0.26	160.682392	0.002924
3	0.55	143.680655	0.002963
4	0.71	138.466707	0.002933
5	0.82	136.511351	0.002985
6	0.89	136.213365	0.002986
7	0.94	136.695475	0.002994
8	0.97	137.666222	0.002932
9	1.00	138.877896	0.002965
10	1.01	140.295496	0.002840
11	1.03	141.831054	0.002524

TABLE III
OPTIMUM SOLUTIONS FOR EXAMPLE 1 WITH $E_{spec} = 0.005$.

n	ratio	minimum W	E
2	0.49	147.385574	0.004020
3	0.77	137.953693	0.004928
4	0.93	135.336271	0.004999
5	1.02	135.099280	0.004989
6	1.01	135.540305	0.004230
7	1.01	136.429802	0.003747
8	1.01	137.562550	0.003376
9	1.01	138.866307	0.003080
10	1.01	140.295496	0.002840
11	1.00	141.831054	0.002524

TABLE IV
OPTIMUM SOLUTIONS FOR EXAMPLE 2 WITH $E_{spec} = 0.02$.

n	ratio	minimum W	E
3	0.400000	163.700223	0.019825
4	0.620000	151.669977	0.019767
5	0.750000	147.325267	0.019755
6	0.830000	145.891900	0.019480
7	0.890000	145.592264	0.019553
8	0.930000	146.122613	0.019403
9	0.960000	147.085528	0.019294
10	0.990000	148.241936	0.019731
11	1.010000	149.694418	0.019847

TABLE V
OPTIMUM SOLUTIONS FOR EXAMPLE 2 WITH $E_{spec} = 0.04$.

n	ratio	minimum W	E
2	0.470000	157.783088	0.039808
3	0.810000	145.076028	0.039520
4	0.990000	141.816575	0.039909
5	1.080000	141.383623	0.039456
6	1.130000	141.990891	0.038819
7	1.120000	143.086989	0.035350
8	1.110000	144.414490	0.032446
9	1.100000	145.909046	0.029926
10	1.090000	147.530024	0.027675
11	1.090000	149.236187	0.026473

any given number of checkpoints. The extended model includes imperfect coverages of both the on-line detection mechanism and the acceptance test. We have shown that under imperfect coverages, E can always be reduced by moving checkpoints to the right on the time axis. Using this property, an algorithm is derived which, assuming that the checkpoint intervals must maintain a common ratio, determines the minimum W subject to $E \leq E_{spec}$. The significant finding from the algorithm is in that if a task requires a high probability of correct execution results, we must do checkpointing more frequently towards the end of the task. This result is a departure from the conventional assumption that checkpoints should be equally distributed throughout the task. The basic reason of using unequal checkpoint intervals is that the requirements on program's correctness outweigh those on program's execution speed.

LIST OF VARIABLES

- T Total fault-free computation time for a task.
- T_i Computation time at which the ith checkpoint is placed.
- I_i Computation time between the ith and $(i + 1)$th checkpoints.
- t_c Time for establishing a checkpoint (checkpointing time).
- n Total number of checkpoints for a task.
- τ_i Sum of I_i and t_c, except for $i = n$.
- r Time for setting up a rollback recovery.
- s Time for setting up a restart recovery.
- p Probability of rollback recovery upon detecting a failure.
- q Probability of restart recovery upon detecting a failure.
- W Mean execution time for a task.
- W_i Mean execution time after establishing the ith checkpoint.
- V_i Mean execution time between ith and $(i + 1)$th checkpoints.
- z_i Computation time between a failure occurrence and the ith checkpoint.
- $F_i(\cdot)$ Distribution function of z_i.
- λ Failure occurrence rate.
- E_j Probability of an unreliable result just before the jth checkpoint.
- d Coverage of the on-line detection mechanism.
- c Coverage of an acceptance test.
- D Combined failure coverage, i.e. $D = d + (1 - d)c$.

ACKNOWLEDGMENT

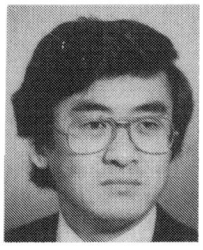
The authors would like to thank the anonymous referees for useful comments on the first draft of this paper.

REFERENCES

[1] C. H. C. Leung and Q. H. Choo, "On the execution of large batch programs in unreliable computing systems," *IEEE Trans. Software Eng.*, vol. SE-10, pp. 444–450, July 1984.
[2] A. Duda, "The effects of checkpointing on program execution time," *Inform. Proc. Lett.*, vol. 16, pp. 221–229, June 1983.
[3] J. W. Young, "A first order approximation to the optimum checkpoint interval," *Commun. ACM,* vol. 17, pp. 530–531, Sept. 1974.
[4] K. M. Chandy, J. C. Browne, C. W. Dissly, and W. R. Uhrig, "Analytic models for rollback and recovery strategies in data base systems," *IEEE Trans. Software Eng.*, vol. SE-1, pp. 100–110, Mar. 1975.
[5] E. Gelenbe, "On the optimum checkpoint interval," *J. Ass. Comput. Mach.*, vol. 26, pp. 259–270, Apr. 1979.
[6] E. Gelenbe and D. Derochette, "Performance of rollback recovery systems under intermittent failures," *Commun. ACM,* vol. 21, pp. 493–499, June 1978.
[7] F. Baccelli, "Analysis of a service facility with periodic checkpointing," *Acta Informatica,* vol. 15, pp. 67–81, Jan. 1981.
[8] A. N. Tantawi and M. Ruschitzka, "Performance analysis of checkpointing strategies," *ACM Trans. Comput. Syst.*, vol. 2, pp. 123–144, May 1984.
[9] T. B. Smith and J. H. Lala, "Development and evaluation of a fault-tolerant multiprocessor (FTMP) computer Vol. I: FTMP principles of operation," NASA Contr. Rep. 166071, May 1983.
[10] A. L. Hopkins *et al.*, "FTMP—A highly reliable fault tolerant multiprocessor for aircraft," *Proc. IEEE,* vol. 66, pp. 1221–1239, Oct. 1978.
[11] J. H. Wensley *et al.*, "SIFT: Design and analysis of a fault-tolerant computer for aircraft control," *Proc. IEEE,* vol. 66, pp. 1240–1255, Oct. 1978.
[12] B. Randell, "System structure for software fault tolerance," *IEEE Trans. Software Eng.*, vol. SE-1, pp. 220–232, June 1975.
[13] 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.
[14] H. Hecht and M. Hecht, "Use of fault trees for the design of recovery blocks," in *Dig. Papers, FTCS-12,* 1982, pp. 134–139.
[15] K. G. Shin and Y.-H. Lee, "Error detection process—Model, design,

and its impact on computer performance," *IEEE Trans. Comput.*, vol. C-33, pp. 529–540, June 1984.

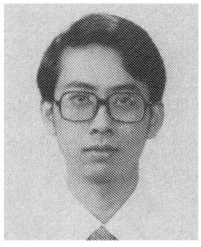
[16] A. Avizienis, "The four-universe information system model for the study of fault-tolerance," in *Dig. Papers, FTCS-12,* 1982, pp. 6–13.



Kang G. Shin (S'75–M'78–SM'83) received the B.S. degree in electronics engineering from Seoul National University, Seoul, Korea in 1970, and the M.S. and Ph.D. degrees in electrical engineering from Cornell University, Ithaca, NY, in 1976 and 1978, respectively.

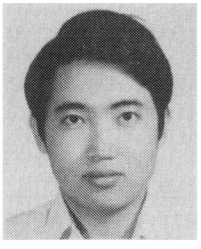
He is a Professor in the Department of Electrical Engineering and Computer Science, The University of Michigan, Ann Arbor, which he joined in 1982. He has been very active and authored/coauthored over 120 technical papers in the areas of fault-tolerant real-time computing, computer architecture, and robotics and automation. In 1986, he founded the Real-Time Computing Laboratory, where he and his students are currently building a 19-node hexagonal mesh multiprocessor to validate various architectures and analytic results in the area of distributed real-time computing. From 1970 to 1972 he served in the Korean Army as an ROTC officer and from 1972 to 1974 he was on the Research Staff of the Korea Institute of Science and Technology, Seoul, Korea, working on the design of VHF/UHF communication systems. From 1978 to 1982 he was an Assistant Professor at Rensselaer Polytechnic Institute, Troy, NY. He was also a Visiting Scientist at the U.S. Airforce Flight Dynamics Laboratory in Summer 1979 and at Bell Laboratories, Holmdel, NJ, in Summer 1980. He was a Guest Editor of the 1987 August special issue of IEEE TRANSACTIONS ON COMPUTERS on Real-Time Systems.

Dr. Shin was the Program Chairman of the 1986 IEEE Real-Time Systems Symposium (RTSS), the General Chairman of the 1987 RTSS. He is a member ACM, Sigma Xi, and Phi Kappa Phi.



Tein-Hsiang Lin (S'83) received the B.S. degree from the National Taiwan University, Taipei, Taiwan, in 1980, and the M.S. degree from the Iowa State University, Ames, in 1984, both in electrical engineering.

Currently, he is working towards the Ph.D. degree in computer, information, and control engineering at the University of Michigan, Ann Arbor. His research interests include multiprocessor and distributed system performance evaluation and fault-tolerant computing.



Yann-hang Lee (S'81–M'84) received the B.S. degree in engineering science and the M.S. degree in electrical engineering from National Cheng Kung University, in 1973 and 1978, respectively, and the Ph.D. degree in computer, information, and control engineering from the University of Michigan, Ann Arbor, in 1984.

Since December 1984, he has been with IBM Thomas J. Watson Research Center, Yorktown Heights, NY. His current research interests include distributed computer systems, database processing, fault-tolerant computing, performance evaluation, and stochastic decision theory.