Computing Optimal Checkpointing Strategies for Rollback and Recovery Systems

PIERRE L'ECUYER AND JACQUES MALENFANT

Abstract—**This paper presents a numerical approach for computing optimal dynamic checkpointing strategies for general rollback and recovery systems. The system is modeled as a Markov renewal decision process. General failure distributions, random checkpointing durations, and reprocessing dependent recovery times are allowed. The aim is to find a dynamic decision rule to maximize the average system availability over an infinite time horizon. We propose a computational approach to approximate such a rule. This approach is based on value iteration stochastic dynamic programming with spline or finite element approximation of the value and policy functions. Numerical illustrations are provided.**

Index Terms—**Availability, checkpointing, dynamic programming, Markov decision processes, numerical approximation, performance optimization, rollback and recovery.**

I. INTRODUCTION

Rollback and recovery with checkpointing is a commonly used technique in computer systems to ensure system integrity in spite of transient failures [6], [11], [12], [20]. Periodically, the state of the system is saved on stable storage; this operation is called *checkpointing*. Meanwhile, the sequence of actions performed by the system (or sufficient information to be able to redo these actions) is also recorded on stable storage (usually in a file called the *audit trail*). In the advent of a failure, the system reloads the state saved at the most recent checkpoint, and resumes execution (replays the audit trail) from that time on, in order to bring itself back to the correct state that immediately preceded the failure. This is called *error recovery*. In fact, the above description is somewhat simplified, but other details are not really necessary for our analysis.

The error recovery duration is usually dependent on the *reprocessing time*, which is defined as the time during which the system has been in production state since the most recent checkpoint. *Production state* is defined as the state in which the system is available to users, i.e. neither performing error recovery nor checkpointing.

The tradeoffs involved in choosing an appropriate checkpoint frequency are the following. Very frequent checkpoints cause high overhead due to the checkpointing durations, while too rare checkpoints also cause high overhead by giving rise to longer recovery periods. The problem of placing the checkpoints "optimally" in time has received considerable attention. The most frequently used objectives are to maximize system availability [4], [8], [9], [14], [16], [20] or to minimize the mean response time per transaction [1], [9], [16]. Major applications of the rollback and recovery technique include database system recovery [6], [20], checkpointing programs [12], [13], [21], [19], synchronizing parallel or distributed processors [11], and database reorganization. Various alternative optimization criteria are also suggested [13]. See [15] and [20] for more complete surveys. Transient error recovery algorithms for distributed systems are discussed in [11]. Most of the investigated models assume instantaneous failure detection and constant failure rate. However, experimental evidence [3], [5] shows that the constant failure rate assumption is not realistic.

Tantawi and Ruschitzka [20] have proposed a model with general interfailure time distributions, random checkpointing durations, and reprocessing dependent recovery times. In contrast to most previous models, failures are also allowed during either checkpointing or error recovery. Their model is a Markov decision process whose state transitions correspond to the failures that occur during either normal processing or checkpointing. At each transition, the system *state* is the reprocessing time at that point, and the *decision* to be made is the sequence of production times between the forthcoming successive checkpoints. This sequence should be followed until the next state transition, at which a new decision should be taken, and so on. A checkpointing *strategy* is a rule for selecting a decision, based on the current system state.

For such a model, one can obtain a general expression for system *availability*, where the availability is defined as the proportion of time the system is in production state, over an infinite horizon. Unfortunately, evaluating this expression is very demanding, since it requires computing an infinite number of imbedded integrals, and solving an integral equation. Computing an optimal policy with this expression is generally out of reach. The expression can be simplified by assuming that failures cannot occur during either error recovery or checkpointing, but even then, finding an optimal strategy still requires the solution of an infinite set of nonlinear equations, giving rise to a computationally intractable problem.

Instead of trying to compute an optimal strategy, Tantawi and Ruschitzka [20] analyze two restricted classes of strategies, under the additional assumption of no failures during either error recovery or checkpointing. These strategies, which are generally suboptimal, are called *equidistant* and *equicost*, respectively. The equidistant strategy assumes that the production time between successive checkpoints is constant, whereas the equicost strategy places the checkpoints so that the expected reprocessing time (cost) between any two successive checkpoints equals the mean checkpointing time (cost). The equicost strategy also forces checkpoints immediately after every error recovery, thus transforming the Markov process to a renewal process to ease the computations, but introducing still further suboptimality. Tantawi and Ruschitzka provide an iterative algorithm to compute an approximation of the best equicost strategy, and give a numerical illustration using the Weibull failure rate. They also derived a simple formula for the "optimal" value of the constant checkpointing interval for the case where one is restricted to the class of equidistant strategy. It depends on the interfailure and checkpointing duration distributions only through their means. Gelenbe [8] had previously shown that under Poisson failures, and assuming that no failure would occur during checkpointing or error recovery, the equidistant strategy is optimal. But these assumptions are not very realistic [3], [5] and in the general case, the best equidistant strategy is not optimal. For the example considered in [20], the best equidistant strategy is better than the best equicost strategy for some parameter values of the Weibull law (e.g., for the exponential case), and the reverse is true for other values.

The model considered in this paper generalizes the model proposed in [20] by considering a component of the failure rate that depends on the current time in a cyclic way. More importantly, we take a different solution approach, based on stochastic dynamic programming, which permits the computation of an optimal strategy in the general case. Our model is a Markov Renewal Decision Process (MRDP), and the optimal strategy is computed using a heuristic extension of Schweitzer's algorithm [18], which solves the dynamic programming functional equations iteratively by successive approximations. Since the state and action spaces of our MRDP are continuous, the problem must be discretized. It is often suggested to approximate such a model by a Markov decision process with finite state and action spaces [14], [16]. But realistic approximations based

Manuscript received June 4, 1987; revised December 8, 1987. This work was supported by NSERC-Canada Grant A5463 and FCAR-Quebec Grant EQ2831 to P. L'Ecuyer.

P. L'Ecuyer is with the Département d'informatique, Université Laval, Ste-Foy, P.Q., Canada, G1K 7P4.

J. Malenfant was with the Département d'informatique, Université Laval, P.Q., Canada G1K 7P4. He is now with the Département d'informatique et de recherche opérationnelle, Université de Montréal, Montréal, P.Q., Canada H3C 1J7.

IEEE Log Number 8719339.

on that approach usually require these spaces to be very large, leading to an intractable computational problem. A more satisfactory approach consists in solving directly the continuous state and action MRDP, using numerical approximation techniques for the value and strategy functions. This kind of approach has been proposed and analyzed in [10] for MRDP's with discounting. Here, we suggest using spline [7] or finite element approximations of the value function (at each iteration), and of the retained strategy.

The general model is stated in Section II. In Section III, we give the dynamic programming formulation, propose a computational algorithm, and discuss some efficient heuristics to accelerate its convergence. In Section IV, we provide numerical illustrations, while in the conclusion, we discuss possible extensions and the practical usefulness of our work.

II. The Model

Our model is an undiscounted Markov Renewal Decision Process (MRDP) model [10], [18] with average reward criteria and with continuous state and action spaces. We first recall briefly the general definition of an MRDP, and then give a more precise description of our model. In an MRDP, the system is observed at random (discrete) points in time, called the *transition times*, denoted by $\tau_0 = 0, \tau_1, \tau_2, \cdots$. At time τ_i, a decision maker observes the *state* s_i of the system and picks a *decision* d_i from the set $D(s_i)$ of admissible decisions in state s_i. Let S be the set of all possible states, called the *state space*, and $D = \cup_{s \in S} D(s)$ be the set of all possible decisions, called the *decision space*.

The dynamics of the system are described by a family $\{p(\cdot \mid s, d) \mid s \in S, d \in D(s)\}$ of *probability laws* on $[0, \infty) \times S$, and rewards are cumulated according to a real-valued *reward function* q defined on $\{(s, d) \mid s \in S, d \in D(s)\}$. If the system is in state s_i at time τ_i and decision d_i is chosen, then the expected one-stage reward for stage i (from τ_i to τ_{i+1}) is $q(s_i, d_i)$, while the next state s_{i+1} and the time $\zeta_i = \tau_{i+1} - \tau_i$ until the next transition are generated according to the probability measure $p(\cdot \mid s_i, d_i)$. Let $T(s_i, d_i) = E[\zeta_i \mid s_i, d_i]$ denote the mean holding time until the next transition. A *policy* is a function μ which associates to each state $s \in S$ a decision $d = \mu(s) \in D(s)$. The aim of the decision maker is to find a policy to maximize the average reward per unit of time, over an infinite time horizon.

For the checkpointing–rollback–recovery model considered here, the *transition times* correspond to failure times, recovery completions, and checkpoint completions. Empirical evidence [3] suggests that for real-life systems, the *failure rate* has a cyclic behavior, related to the variation in time of the workload, and also varies (decreases) with the time since the last failure. Thus, we assume in our model that the failure rate is given by a function $\lambda(\tau, y)$ of the current time τ and of the time y since the last failure. This function depends on τ in a cyclic fashion: $\lambda(\tau, y) = \lambda((\tau \bmod K), y)$, where K is the period of the cycle. It is also assumed to be bounded, and bounded away from zero: $m \le \lambda(\tau, y) \le M$ for some constants $M \ge m > 0$. Let $\bar{F}(u \mid \tau, y)$ denote the corresponding conditional *reliability function*, which gives the probability of no failure during the next u units of time, given τ and y, and $f(u \mid \tau, y) = \lambda(\tau + u, y + u)\bar{F}(u \mid \tau, y)$ the corresponding conditional *failure density function*. We also have $\bar{F}(u \mid \tau, y) = \bar{F}(u \mid (\tau \bmod K), y)$ and $f(u \mid \tau, y) = f(u \mid (\tau \bmod K), y)$. Proper definitions of these quantities, in general, are given in most probability texts (see, e.g., [22]). As in most earlier works [1], [8], [9], [16], [20], we assume instantaneous failure detection. This implies that the state saved by a checkpoint is always correct. In practice, failure detection is not always instantaneous. A checkpoint may sometimes be contaminated, and the system may have to roll back to an earlier checkpoint. In principle, our model could be extended to take this possibility into account, but this would complicate the optimization problem significantly. We do not consider it in this paper.

Checkpoint durations are independent identically distributed (i.i.d.) random variables, with distribution $F_c(\cdot)$. The *error recovery* time after a failure is a random variable whose probability distribution $F_r(\cdot \mid x)$ depends on the reprocessing time x, i.e., the production time since the most recent checkpoint completion preceding the failure. The mathematical expectations of all these random variables are assumed to be finite and bounded away from zero: $m_1 \le \int_0^\infty c dF_c(c) < \infty$ and $m_2 \le \int_0^\infty r dF_r(r \mid x) < \infty$ for all $x \ge 0$, for some constants $m_1 > 0$ and $m_2 > 0$.

The *state* of the system at any transition time is described by a vector $s = (t, x, y, j)$, where t is the current time modulo K, x is the reprocessing time, y is the time since the last failure, and $j = 1, 2$, or 3 depending on whether this transition corresponds to a failure, a recovery completion, or a checkpoint completion. The *state space* is $S = S_1 \cup S_2 \cup S_3$, where

$$S_1 = \{(t, x, 0, 1) \mid 0 \le t \le K, 0 \le x\}$$

$$S_2 = \{(t, x, y, 2) \mid 0 \le t \le K, 0 \le x, 0 \le y\}$$

$$S_3 = \{(t, 0, y, 3) \mid 0 \le t \le K, 0 \le y\}. \tag{1}$$

When $j = 3$, then $x = 0$. When $j = 1$, then $y = 0$ and the only admissible decision is to initiate a recovery. We denote this decision by ρ. When $j = 2$ or 3, a decision d corresponds to the production time from now until the beginning of the next planned checkpoint. The set of admissible decisions in that case is $D(s) = [0, \infty)$. Thus, the *decision space* is $D = \{\rho\} \cup [0, \infty)$.

The possible transitions from any given state $s = (t, x, y, j)$ in S are given below. The next state is denoted by $s' = ((t + u) \bmod K, x', y', j')$, where u is the elapsed time before transiting to s', and c denotes the duration of the next checkpointing operation. If $s \in S_1$ (failure point), let r denote the value of the recovery time. In this case, the next transition may be triggered by a new failure during the recovery ($0 \le u < r$ and $(x', y', j') = (x, 0, 1)$), or correspond to the normal recovery completion ($u = r$ and $(x', y', j') = (x, u, 2)$). If $s \in S_2 \cup S_3$ (i.e., $j = 2$ or 3), then a decision $d \ge 0$ is chosen, and the next transition may correspond to a failure before the beginning of the next planned checkpoint ($0 \le u < d$ and $(x', y', j') = (x + u, 0, 1)$), or a failure during the checkpoint operation $d \le u < d + c$ and $(x', y', j') = (x + d, 0, 1)$), or the normal completion of the checkpoint ($u = d + c$ and $(x', y', j') = (0, y + u, 3)$).

The expected time $T(s, d) = T(t, x, y, j, d)$ until the next transition, which is the mathematical expectation of u given s and d, is given by

$$T(t, x, 0, 1, 0) = \int_0^\infty u f(u \mid t, 0)(1 - F_r(u \mid x))\, du + \int_0^\infty u \bar{F}(u \mid t, 0) dF_r(u \mid x) \tag{2}$$

$$T(t, x, y, j, d) = \int_0^\infty \left[\int_d^{d+c} u f(u \mid t, y)\, du + (d + c)\bar{F}(d + c \mid t, y) \right] dF_c(c) + \int_0^d u f(u \mid t, y)\, du \quad \text{if } j \ne 1. \tag{3}$$

In (2), the first term corresponds to a failure during recovery, while the second term corresponds to normal recovery completion. In (3), the last term corresponds to a failure during production (before the beginning of the checkpoint), and in the first term, the two parts correspond to a failure during the checkpointing operation and to a normal checkpoint completion, respectively. The one-stage expected reward $q(s, d)$ is the expected production time until the next transition, which is given by

$$q(t, x, 0, 1, 0) = 0 \tag{4}$$

$$q(t, x, y, j, d) = \int_0^d u f(u \mid t, y)\, du + d\bar{F}(d \mid t, y) \quad \text{if } j \ne 1. \tag{5}$$

When $j = 1$, the production time is zero, since the system is recovering. In (5), the first term corresponds to a failure during production, while the second term corresponds either to a failure during the next checkpointing operation or to normal checkpoint completion.

The decision maker seeks a *policy* μ to maximize the system's availability, i.e., the proportion of time, over an infinite horizon, that the system is in production state. At time τ_i, the system is in state $s_i = (t_i, x_i, y_i, j_i)$, and following policy μ, a decision $d_i = \mu(s_i) \in D(s_i)$ is taken. The time ξ_i until the next failure follows the distribution $\Pr(\xi_i \le \zeta) = 1 - \bar{F}(\zeta | t_i, y_i)$. The next transition occurs at $\tau_{i+1} = \tau_i + u_i$, where

$$u_i = \begin{cases} \min(\xi_i, r_i) & \text{if } j_i = 1; \\ \min(\xi_i, d_i + c_i) & \text{if } j_i = 2 \text{ or } 3 \end{cases} \tag{6}$$

and where r_i and c_i denote the respective values of the random recovery duration and random checkpoint duration. The next state s_{i+1} is generated according to the probability laws given above. The aim of the decision maker is to find a policy μ that maximizes

$$A(\mu) = E_\mu \left[\lim_{n\to\infty} \frac{1}{(\tau_n - \tau_0)} \sum_{i=0}^{n-1} q(s_i, d_i) \right] \tag{7}$$

where E_μ denotes the mathematical expectation, conditional on the use of policy μ. We assume that the above mathematical expectation does not depend on the initial state s_0. Let $A_* = \sup_\mu A(\mu)$ be its optimal value. A policy μ is called *optimal* if $A(\mu) = A_*$, and ϵ-optimal, for $\epsilon > 0$, if $A(\mu) \ge A_* - \epsilon$. In practice, one will try to compute an ϵ-optimal policy for a small enough value of ϵ.

III. A Dynamic Programming Computational Procedure

In this section, we develop a numerical approximation method to approximate A_* and to compute an ϵ-optimal policy. It is based on dynamic programming. For a modern treatment of dynamic programming and Markovian decision problems, see Bertsekas [2]. Schweitzer [18] devised a successive approximation algorithm to solve an MRDP with finite state space $S = \{1, \cdots, N\}$ and finite decision space D. In this case, the law of transition from state to state is given by a set of transition probabilities: at any stage, $p(s' | s, d)$ is the probability that $s_{i+1} = s'$ given that $s_i = s$ and $d_i = d$. Schweitzer assumes that the mean holding time $T(s, d)$ is finite, and that there exists a constant $\gamma > 0$ such that for all admissible pairs (s, d), $0 < \gamma < T(s, d)/(1 - p(s|s, d))$. He also assumes that for each policy, the associated Markov chain has a unique subchain, plus possibly some transient states feeding this subchain.

Without loss of generality, let $s = N$ be a recurrent state. For every real-valued function V defined on S, let the constant $k(V)$ and the real-valued function $J(V)$ on S be defined, respectively, by

$$k(V) = \max_{d \in D(N)} \left[\frac{q(N, d) + \sum_{s'=1}^{N-1} V(s') p(s' | N, d)}{T(N, d)} \right] \tag{8}$$

$$J(V)(s) = V(s) + \gamma \max_{d \in D(s)} \cdot \left[\frac{q(s, d) + \sum_{s'=1}^{N-1} V(s') p(s' | s, d) - V(s) - k(V) T(s, d)}{T(s, d)} \right]. \tag{9}$$

Let $V_0 : S \to \mathbb{R}$ be a given initial function, and define recursively $k_n = k(V_{n-1})$ and $V_n = J(V_{n-1})$, for $n = 1, 2, 3, \cdots$. It is proven in [18] that 1) $J(V_*) = V_*$, where $V_* = \lim_{n\to\infty} V_n$, 2) $\lim_{n\to\infty} k_n = A_* = k(V_*)$, where A_* is the optimal average reward, and 3) the policy μ_* defined by

$$\mu_*(s) = \arg \max_{d \in D(s)} \left[\frac{q(s, d) + \sum_{s'=1}^{N-1} V_*(s') p(s' | s, d) - V_*(s)}{T(s, d)} \right] \tag{10}$$

is optimal. Here, "$\arg \max_{d \in D(s)}$" denotes a value of d for which the maximum is attained. Schweitzer's algorithm simply computes iteratively k_n and V_n, for $n = 1, 2, 3, \cdots$, until $|V_n(s) - V_{n-1}(s)|$ is smaller than a given ϵ for all s in S. The retained policy is the one obtained by replacing V_* by V_n in (10).

The above algorithm is in fact an iterative method for solving the functional equation $J(V_*) = V_*$. The interpretation of V_* is as follows. Let $R_*^{[0, t]}(s)$ be the total expected reward for period $[0, t)$, under an optimal policy, when s is the initial state at time 0. Then, it can be shown that $V_*(s) = \lim_{t\to\infty} (R_*^{[0, t]}(s) - tA_*)$. Thus, $V_*(s)$ represents a "differential gain" due to state s.

We now go back to our model of Section II. Schweitzer's assumptions do not hold for this model, since it has continuous state and action spaces. But our approach will be to generalize his algorithm in a heuristic way, replacing the set of transition probabilities by more general probability laws and the sums by integrals, putting aside all measurability issues. For practical purposes, we assume that for all policies of interest, each of the state variables x and y will always come back to zero, and that the expected duration between any two successive visits to zero is finite. This implies that the associated Markov chain has only one subchain.

Let γ be a positive constant such that

$$\gamma \le \inf \{ T(s, d) | s \in S, d \in D(s) \}, \tag{11}$$

and let $s_* = (t_*, 0, y_*, 3) \in S_3$ be a selected state. We replace (8) and (9) by

$$k(V) = \max_{d \in D} \left[\frac{H(V)(s_*, d)}{T(s_*, d)} \right] \tag{12}$$

$$J(V)(s) = V(s) + \gamma \max_{d \in D(s)} \cdot \left[\frac{H(V)(s, d) - V(s) - k(V) T(s, d)}{T(s, d)} \right] \tag{13}$$

where V is any real-valued function defined on S, and

$$\begin{aligned} &H(V)(t, x, 0, 1, \rho) \\ &\quad = q(t, x, 0, 1, \rho) + \int_0^\infty V((t+u) \bmod K, x, 0, 1) \\ &\qquad \cdot (1 - F_r(u|x)) f(u|t, 0)\, du \\ &\qquad + \int_0^\infty V((t+u) \bmod K, x, u, 2) \bar{F}(u|t, 0)\, dF_r(u|x) \end{aligned} \tag{14}$$

$$\begin{aligned} &H(V)(t, x, y, j, d) \\ &\quad = q(t, x, y, j, d) + \int_0^\infty \\ &\qquad \cdot \left[\int_d^{d+c} V((t+u) \bmod K, x+d, 0, 1) f(u|t, y)\, du \right. \\ &\qquad \left. + V((t+d+c) \bmod K, 0, y+d+c, 3) \bar{F}(d+c|t, y) \right] dF_c(c) \\ &\qquad + \int_0^d V((t+u) \bmod K, x+u, 0, 1) f(u|t, y)\, du \qquad \text{if } j \ne 1. \end{aligned} \tag{15}$$

The first integral in (14) corresponds to a failure during recovery, while the second one corresponds to recovery's completion. In (15), the last integral corresponds to a failure before the next checkpoint and in the first integral, the first term (inside integral) corresponds to a failure during checkpointing while the second one corresponds to a normal checkpoint completion. T and q are defined in (2)–(5).

Implementing a variant of Schweitzer's algorithm with these equations requires numerical approximation. In practice, $J(V)$ can be computed only for a finite number of states at each iteration and, for an efficient implementation, that number should be reasonably small. That will not be the case if we adopt the commonly used naive discretization approach which consists in partitioning each of S and D into a finite class of subsets, selecting a representative state or decision in each subset, and defining an approximate finite state and action model. Under such an approach, the function V is in fact approximated by a piecewise constant function. A more satisfactory approach consists in using numerical approximation schemes like spline approximation, finite element methods, etc., to approximate V and μ in (12)–(15). Numerical integration techniques must also be used, and the maximization over d could usually be performed as a sequential search during the integration to approximate the optimum, followed by a refinement to locate it more precisely.

The number of states at which $J(V)$ is evaluated, the order of the approximation method, the order and step size of the integration method, and the required precision level in the optimal value of d may vary from iteration to iteration. It should be reasonable to start with coarse grids and simple (low-order) methods, gradually refining them as the iterative process is going on. Also, the algorithm could be modified in order to accelerate the convergence. A good idea is to start with a simple policy, like for instance the best *equidistant* strategy [9], [20], and keep it for a large number of iterations, to obtain a low-cost first approximation of V_*. Afterwards, instead of performing maximizations at every iteration, one could keep the same policy for a number of iterations, then maximize at one iteration to obtain a new policy, and so on. The optimal policy is never "attained" exactly, but is approached further and further as iterations and refinements go on. One stops when the current approximation is satisfactory for his needs.

In practice, the convergence rate of the algorithm is much affected by the value of γ. The smaller γ is, the slower it goes, and the relation is approximately linear. The terms in brackets in (13), multiplied by γ, represent the change in the value of V at the current iteration. An interesting heuristic to accelerate the convergence is to amplify this change, simply by replacing γ in (13) by: $\gamma' = \omega\gamma$, where $\omega \geq 1$ is a constant. This idea is called *overrelaxation* [17]. In practice, the value of ω may vary from iteration to iteration. Since this new value of γ usually does not satisfy (11), convergence is no more guaranteed. But in practice, large values of ω (e.g., 10 or 20) may sometimes be used without preventing convergence, and the computation time is thus divided approximately by ω. To guarantee convergence, one may use a decreasing sequence of values of ω converging to a value smaller than one. The approach we took in our experiments was more akin to interactive "trial-and-error:" increase ω (or keep it high) when the value of V is moving slowly but consistently in one direction, decrease it otherwise (i.e., when V starts to oscillate).

IV. Numerical Illustrations

In this section, we illustrate the proposed computational method with two examples, which are particular cases of the general model. In the first example, the failure rate depends on the time since the last failure, while in the second one, it depends only on the current time, in a cyclical fashion. Of course, these are only numerical illustrations, and cannot be used to draw general conclusions on the form of the optimal policy or on the goodness of the simpler suboptimal rules.

Example 1: Our first example is similar to the one considered by Tantawi and Ruschitzka [20], for which they were able to compute only a suboptimal policy. These authors assumed Weibull interfailure times with shape parameter 0.5 and a mean of 60 h. Their corresponding failure rate is $\lambda(y) = 0.5/\sqrt{30y}$, where y is the elapsed time since the last failure. Since that failure rate is neither bounded nor bounded away from zero, we just add to it the small constant 0.0001 everywhere and clamp it, for y near zero, to a large constant equal to $\lambda(0.001)$. Thus, we obtain

$$\lambda(y) = \begin{cases} 0.5/\sqrt{0.03} + 0.0001, & \text{if } y < 0.001, \\ 0.5/\sqrt{30y} + 0.0001, & \text{if } y \geq 0.001. \end{cases}$$

The error recovery time is assumed to be a (deterministic) linear function r of the reprocessing time: $r(x) = 0.5x + 0.1$, for $x \geq 0$. The checkpointing time is assumed to be constant at 1 min.

Data obtained from several computer systems [3] suggest that in the case of transient system errors, the Weibull distribution with a decreasing failure rate is a very good approximation to interfailure times. The error recovery is typically the sum of a fixed overhead time plus a variable time to reprocess the work that has been done since the last checkpoint completion. The choice of a fixed checkpointing time in this example is not due to a limitation of our method; random times can be handled as well (although increasing the computational effort).

In this example, since the recovery time is deterministic, the decision d that has to be taken at the end of a recovery may be taken in advance at the last failure point preceding that recovery. Hence, it is not necessary to observe the system at recovery completions, so that the *transition times* may correspond only to failure times and checkpoint completions. Also, it is not necessary to keep the current time t in the state description. The *state space* becomes $S = S_1 \cup S_3$, where $S_1 = \{(x, 0, 1) | 0 \leq x\}$ and $S_3 = \{(0, y, 3) | 0 \leq y\}$. Now, when $j = 1$, the decision maker takes a decision $d \geq 0$, which corresponds to the production time from the end of the recovery to the beginning of the next planned checkpoint, provided that no failure occurs during that recovery. The possible state transitions from any given state s in S are

$$s = (x, 0, 1) \rightarrow \begin{cases} s' = (x, 0, 1), & \text{if } 0 \leq u < r(x) \\ & \text{(failure during the recovery)} \\ s' = (x + u - r(x), 0, 1), & \\ & \text{if } r(x) \leq u \leq r(x) + d \\ & \text{(failure during production)} \\ s' = (x + d, 0, 1), & \\ & \text{if } r(x) + d < u < r(x) + d + c \\ & \text{(failure during checkpointing)} \\ s' = (0, r(x) + d + c, 3), & \\ & \text{if } u = r(x) + d + c \\ & \text{(normal checkpoint completion)} \end{cases}$$

$$s = (0, y, 3) \rightarrow \begin{cases} s' = (u, 0, 1), & \text{if } 0 \leq u \leq d \\ & \text{(failure during production)} \\ s' = (d, 0, 1), & \text{if } d < u < d + c \\ & \text{(failure during checkpointing)} \\ s' = (0, y + d + c, 3), & \\ & \text{if } u = d + c \\ & \text{(normal checkpoint completion).} \end{cases}$$

Equations (2)-(5), (14), and (15) are easy to specialize accordingly. Since the state space is the union of two one-dimensional half-spaces, the function V is not very hard to approximate. We select two grids: $0 = x_1 < x_2 < \cdots < x_\alpha$, and $0 = y_1 < y_2 < \cdots < y_\beta$, where α and β are two positive integers, and evaluate $J(V)(s)$ only for s in $S' = S_1' \cup S_3'$, where $S_1' = \{(x_i, 0, 1), 1 \le i \le \alpha\}$ and $S_3' = \{(0, y_k, 3), 1 \le k \le \beta\}$.

Spline approximation was used to approximate $J(V)$ on S_1 and S_3, yielding two spline functions V_1' and V_3'. For $x > x_\alpha$ or $y > y_\beta$, we defined $V_1'(x) = V_1'(x_\alpha)$ and $V_3' = V_3'(y_\beta)$. These two approximating functions were then used for the computation of $H(V)$ at the next iteration. Simpson's integration rule was used to compute $H(V)$, and the maximization with respect to d was done during the integration. For the examples described here, we used an overrelaxation factor $\omega = 50$ for the first 200 iterations, and $\omega = 10$ afterwards.

A *Pascal* program has been written on a VAX-11/780 to solve this numerical example. The functions V_1' and V_3' were piecewise linear for a number of iterations, and then splines of order 4 (cubic splines). We had $\gamma = 0.015$ and $s_* = (0, 3, 3)$. We computed the availability under an optimal policy, and also the availability under the equidistant strategy with a checkpointing interval of 118.9 min (which is the best equidistant strategy [20]). We obtained a system availability of 0.9836 for the optimal case, and 0.9823 for the equidistant case. These figures differ from those obtained by Tantawi and Ruschitzka [20], who obtained 0.9833 for equicost and 0.9818 for equidistant; but these authors were assuming that no failures occur during error recovery or checkpointing, and their time since last failure did not include the time spent performing these activities. We also computed the optimal policy under these assumptions, and obtained an availability of 0.9836. Hence, for this particular example, we see that the *equicost* strategy is almost optimal. Also, the difference in performance between the best equidistant strategy and the optimal strategy is relatively small (here, this difference corresponds to about 11 h per year for a system which operates 24 h a day). But this need not be always the case. For Weibull distributions with very small or very large shape parameters, the difference could possibly be larger (see [20]).

At optimality, the function $V_*(x, 0, 1)$ decreases in x almost linearly, while $V_*(0, y, 3)$ increases in y in a log-like fashion. The curve $\mu_*(x, y, j)$ represents the optimal value of the production time d until the next planned checkpoint. When $j = 1$ (and $y = 0$), $\mu_*(x, 0, 1)$ decreases almost linearly in x from a value of approximately 0.78 at $x = 0$, to 0 at $x = 0.66$ h. It is zero for $x > 0.66$ h. Hence, it is not always optimal to perform a checkpoint immediately after a failure recovery. When $j = 3$ (and $y = 0$), $\mu_*(0, y, 3)$ increases in y (in a log-like fashion), in accordance with the decreasing failure rate assumption.

Example 2: We now consider a failure rate which depends on the current time t in a cyclic fashion, with period $K = 24$ h. It may be interpreted as the sum of a constant component and a time-varying component, which depends on the system's load. The failure rate function is given by

$$\lambda(t) = \begin{cases} 1/100 & \text{if } 0 \le t < 8; \\ 1/40 & \text{if } 8 \le t < 12; \\ 1/60 & \text{if } 12 \le t < 13; \\ 1/50 & \text{if } 13 \le t < 17; \\ 1/100 & \text{if } 17 \le t < 24. \end{cases}$$

The recovery time function and checkpointing time are the same as in example 1.

In this example, the current time modulo K must be kept in the state description, but it is not necessary to keep the time y since the last failure. The state space is $S_1 \cup S_3$, where $S_1 = \{(t, x, 1) | 0 \le t < 24, 0 \le x\}$ and $S_3 = \{(t, 0, 3) | 0 \le t < 24\}$. Transition times are defined as in example 1. At any transition time, the decision maker takes a decision $d \ge 0$, which corresponds to the production time until the next planned checkpoint. The largest value of γ that satisfies (11) is $\gamma = 0.0166$. S_3 is a one-dimensional line segment and S_1 is a rectangle in the plane.

A *Pascal* program was run to solve this example. $J(V)$ was approximated by order 2 splines (piecewise linear) functions on S_3, and by piecewise bilinear functions on rectangular finite elements covering S_1. As the iterations went on, the number of evaluation points went from 5 to 241 in t on S_3, and from 5×5 to 21×49 in (x, t) on S_1. The computed optimal availability was 0.9836.

The optimal policy is relatively complex. For instance, $\mu_*(t, 0, 3)$ is piecewise decreasing, but with upward jumps between the pieces. For fixed x, $\mu_*(t, x, 1)$ behaves similarly, while for fixed t, $\mu_*(t, x, 1)$ decreases in x until it reaches zero, and is equal to zero for larger values of x.

V. Conclusion

It is still possible to consider further generalizations of the model, such as allowing checkpoints during error recovery [15] (this might be worthwhile since the failure rate just after a failure is often very high) or coping with distributed systems [11]. Other performance measures than the availability could also be considered, like a time-varying utility of the availability, the average waiting time per transaction [15], or maximizing the probability of finishing a given task before a given deadline [13].

Dynamic programming need not be used only to compute policies, it could also be used, sometimes, to prove useful theoretical properties. For instance, it could be interesting to obtain sufficient conditions under which the optimal policy μ_* would be monotonic, e.g., increasing in y and decreasing in x. We have been unable to obtain such conditions which are realistic and easy to verify, so this remains an open problem.

For highly complex models, for which the state description includes more than two or three continuous variables at the same time, it becomes very difficult to approximate $J(V)$ properly, and the required amount of computation soon becomes overwhelming, leading to a practically intractable problem. In such cases, using suboptimal rules is probably the only way to go in practice. These suboptimal rules may be evaluated through computer simulation, which is another highly time consuming activity, but it does not tell how much suboptimal they are. To evaluate the degree of suboptimality of a policy, we must be able to compute or approximate the optimal value of the objective function. Thus, even if the algorithm proposed in this paper is computationally unattractive for overly complex models, it could be used to evaluate the degree of suboptimality of simpler control policies for sufficiently realistic models.

Acknowledgment

The authors wish to thank Dr. A. Mili, B. Pinta, O. Roux, and T. Vo-Dai for their remarks and suggestions. A discussion with Dr. A. N. Tantawi of IBM was also very helpful. M. Mayrand wrote part of the software and performed the numerical experiments for Section IV.

References

[1] F. Bacelli, "Analysis of a service facility with periodic checkpointing," *Acta Informatica,* vol. 15, pp. 67-81, Jan. 1981.

[2] D. P. Bertsekas, *Dynamic Programming: Deterministic and Stochastic Models.* Englewood Cliffs, NJ: Prentice-Hall, 1987.

[3] X. Castillo, S. R. McConnel, and D. P. Siewiorek, "Derivation and calibration of a transient error reliability model," *IEEE Trans. Comput.,* vol. C-31, pp. 658-671, July 1982.

[4] K. M. Chandy, "A survey of analytic models of rollback and recovery strategies," *Computer,* vol. 8, pp. 40-47, May 1975.

[5] L. H. Crow and N. D. Singpurwalla, "An empirically developed Fourier series model for describing software failures," *IEEE Trans. Reliability,* vol. R-33, pp. 176-183, June 1984.

[6] C. J. Date, *An Introduction to Database Systems, Vol. II.* Reading, MA: Addison-Wesley, 1983.

[7] C. DeBoor, *A Practical Guide to Splines.* Berlin, Germany: Springer-Verlag, 1978.

[8] E. Gelenbe, "On the optimum checkpoint interval," *J. Ass. Comput. Mach.,* vol. 26, pp. 259-270, Apr. 1979.

[9] E. Gelenbe and D. Derochette, "Performance of rollback and recovery

systems under intermittent failures," *Commun. Ass. Comput. Mach.*, vol. 21, pp. 493–499, June 1978.
[10] A. Haurie and P. L'Ecuyer, "Approximation and bounds in discrete event dynamic programming," *IEEE Trans. Automat. Contr.*, vol. AC-31, pp. 227–235, Mar. 1986.
[11] R. Koo and S. Toueg, "Checkpointing and rollback-recovery for distributed systems," *IEEE Trans. Software Eng.*, vol. SE-13, pp. 23–31, Jan. 1987.
[12] I. Koren, Z. Koren, and S. Y. H. Su, "Analysis of a class of recovery procedures," *IEEE Trans. Comput.*, vol. C-35, pp. 703–712, Aug. 1986.
[13] C. M. Krishna, K. G. Shin, and Y-H. Lee, "Optimization criteria for checkpoint placement," *Commun. Ass. Comput. Mach.*, vol. 27, pp. 1008–1012, Oct. 1984.
[14] J. M. Magazine, "Optimality of intuitive checkpointing policies," *Inform. Processing Lett.*, vol. 17, pp. 63–66, Aug. 1983.
[15] J. Malenfant, "Modélisation du rétablissement lors des pannes dans les bases de données par des processus de renouvellement markoviens commandés," Rep. DIUL-RR-8701, Dép. d'informatique, Univ. Laval, Jan. 1987.
[16] V. F. Nicola and F. J. Kylstra, "A model of checkpointing and recovery with a specified number of transactions between checkpoints," in *PERFORMANCE '83*, A. K. Agrawala and S. K. Tripathi, Eds. Amsterdam, The Netherlands: North-Holland, 1983, pp. 83–100.
[17] E. L. Porteus and J. C. Totten, "Accelerated computation of the expected discounted return in a Markov chain," *Oper. Res.*, vol. 26, pp. 350–358, 1978.
[18] P. J. Schweitzer, "Iterative solution of the functional equations of undiscounted Markov renewal programming," *J. Math. Anal. Appl.*, vol. 34, pp. 495–501, 1971.
[19] K. G. Shin, T.-H. Lin, and Y.-H. Lee, "Optimal checkpointing of real-time tasks," *IEEE Trans. Comput.*, vol. C-36, Nov. 1987.
[20] A. Tantawi and M. Ruschitzka, "Performance analysis of checkpointing strategies," *ACM Trans. Comput. Syst.*, vol. 2, pp. 123–144, May 1984.
[21] S. Toueg and Ö. Babaoglu, "On the optimum checkpoint selection problem," *SIAM J. Comput.*, vol. 13, pp. 630–649, Aug. 1984.
[22] K. S. Trivedi, *Probability and Statistics with Reliability, Queueing, and Computer Science Applications*. Englewood Cliffs, NJ: Prentice-Hall, 1982.

Pseudoexhaustive Test Pattern Generator with Enhanced Fault Coverage

P. GOLAN, O. NOVÁK, AND J. HLAVIČKA

Abstract—**This paper describes a new method of pseudoexhaustive test pattern generation suitable above all for circuits using random access scan (RAS). Two linear feedback shift registers (LFSR) are used to generate scan addresses and test patterns to be scanned into these addresses. It is shown that the suggested testing method gives better results than random testing.**

Index Terms—**Built-in self-test equipment, linear code, linear feedback shift register, pseudoexhaustive testing, random testing, scan design, simplex code, test generation.**

Manuscript received October 11, 1986; revised November 10, 1987.

P. Golan and O. Novák are with the Research Institute for Mathematical Machines, Prague 1, Czechoslovakia.
J. Hlavička is with the Department of Computers, Czech Technical University, Prague 2, Czechoslovakia.

IEEE Log Number 8719340.

I. Introduction

Design for testability [13] has become a necessity for VLSI circuits. Although the separation of combinational circuits from memory simplified the test generation, the amount of computation needed for the fault modeling still remains unacceptably large. Therefore, the pseudoexhaustive testing, i.e., exhaustive testing of inputs feeding one output, has been suggested [5]. The pseudoexhaustive test sets can be generated, e.g., by a built-in test pattern generator (TPG) using LFSR [1].

Tang and Chen showed in [3] the relationship between the output sequences generated by an m-bit LFSR and cyclic codes with minimum distance $d_{\min}$, which allowed the generation of pseudoexhaustive test sets. They are characterized by the fact that within the set of codewords, all 2^r possible vectors, i.e., the exhaustive test set, can be found on any r-tuple of bits for $r \leq d_{\min} - 1$. The biggest drawback of the method [3] is a large number of different LFSR seeds needed for larger $d_{\min}$. This drawback was removed by Wang and McCluskey [4], but only at the expense of a much longer LFSR, and by Akers [6] whose method is applicable only to a small number of UUT inputs. [2] and [9] suggest other methods of pseudoexhaustive test set generation, but their implementation is much more difficult than using an LFSR. McCluskey described in [5] a table-driven algorithm for manual pseudoexhaustive test set generation. In [7] Lempel and Cohn presented a method similar to [3], based on concatenation of test matrices corresponding to different $(2^m - 1, m)$ cyclic simplex codes, so-called maximum-length shift-register codes, whose codewords are generated by m-bit LFSR's. They assume that the corresponding generator polynomials would be chosen at random. From this assumption, they deduced the probability of pseudoexhaustive test set existence. The number of tests needed to create with a given probability a pseudoexhaustive test set is very high, much higher than for random test set.

In [15], Dervisoglu made a more optimistic estimation of the probability of pseudoexhaustive test set existence for the method [7]. He assumed that the selection of several different primitive polynomials for LFSR implies random, equally probable permutations of columns of the test matrices. However, it was shown in [7] that there exist some m-tuples of columns which cannot be covered exhaustively even after having used all primitive polynomials of degree m. Thus, the permutations of columns cannot be considered as equally probable and the formula derived in [15] for the method of [7] is not true.

Specifically, at the application to the level-sensitive scan design (LSSD) are aimed methods [8] and [10]. Bardell and McAnney [8] solve to some extent the problem of series–parallel generation of test patterns and removal of some structural dependencies (i.e., the existence of identical bit streams on different UUT inputs). This problem was fully solved in [16].

II. General Method of Fault Coverage Enhancement

Our goal is to generate a pseudoexhaustive test set for an s-input, t-output combinational circuit in which every output depends on at most r inputs ($r < s$). The subcircuit with r inputs containing all gates which feed signals into one output is called a *cone* or, if we want to show the number of its inputs, *r-cone*.

Definition 1: Binary matrix $T^*(s, r)$ with s columns in which every submatrix of r columns contains all $2^r - 1$ nonzero binary row vectors of length r is called *completely (s, r)-exhaustive test matrix*.

A completely (s, r)-exhaustive test matrix describes (after adding a row of zeros if necessary) an *(s, r)-exhaustive test set* in which exhaustive test sets for all r-tuples of inputs are contained (we assume that every matrix column represents data fed into one UUT input). The addition of a row of zeros is necessary if the TPG does not generate the all-zero vector on the respective r-tuple of bits. As the remedy is very simple, we will not discuss this point any further and