Journal of Computer Science 2 (6): 528-534, 2006
ISSN 1549-3636

Determining the Optimal Software Rejuvenation Schedule via Semi-Markov Decision Process

Hiroyuki Eto and Tadashi Dohi
Department of Information Engineering, Hiroshima University
1-4-1 Kagamiyama, Higashi-Hiroshima 739-8527, Japan

Abstract: Software rejuvenation is a preventive and proactive maintenance policy that is particularly useful for counteracting the phenomenon of software aging. In this study we consider an operational software system with multiple degradations and derive the optimal software rejuvenation policy minimizing the expected operation cost per unit time in the steady state, via the dynamic programming approach. Especially, we show analytically that the control-limit type of software rejuvenation policy is optimal. A numerical example is presented to make a decision table and to perform the sensitivity analysis of cost parameters.

Key words: Software aging, software rejuvenation, semi-Markov decision process, optimality

INTRODUCTION

Software faults should ideally have been removed during the debugging phase. Even if a piece of software has been thoroughly tested, it still may have some design faults that are yet to be revealed. Such software faults are called *bohrbugs* and may exist even in mature software such as commercial operating systems. Also, even mature software can be expected to have what are known as *heisenbugs*[1]. These are bugs in the software that are revealed only during specific collusions of events. For example, a sequence of operations may leave the software in a state that results in an error on an operation executed next. Simply retrying a failed operation, or if the application process has crashed, restarting the process might resolve the problem. Another type of fault observed in software systems is due to the phenomenon of resource exhaustion. Operating system resources such as swap space and free memory available are progressively depleted due to defects in software such as memory leaks and incomplete cleanup of resources after use. These faults may exist in operating systems, middleware and application software.

When software application executes continuously for long periods of time, some of the faults cause software to age due to the error conditions that accrue with time and/or load. *Software aging* will affect the performance of the application and eventually cause it to fail[2-4]. Software aging has also been observed in widely-used communication software like Internet Explorer, Netscape and xrn as well as commercial operating systems and middleware. A complementary approach to handle software aging and its related transient software failures, called *software rejuvenation*, is becoming popular[3-5]. Software rejuvenation is a preventive and proactive solution that is particularly useful for counteracting the phenomenon of software aging. It involves stopping the running software occasionally, cleaning its internal state and restarting it. Cleaning the internal state of a software might involve garbage collection, flushing operating system kernel tables, reinitializing internal data structures and hardware reboot.

In this study we consider an operational software system with multistage degradations and derive the optimal software rejuvenation policy minimizing the expected operation cost per unit time in the steady state, via the dynamic programming approach. This can be considered as an extension of the classical two-step failure models with time-based rejuvenation policies[5-8]. Vaidyanathan *et al.*[9] treat a multistep failure model, but do not discuss the optimal software rejuvenation policy from the analytical point of view. We suppose that the state of software system deteriorates stochastically and is described by a right-skip free continuous-time Markov chain (CTMC) with an absorbing state[10]. In the dynamic operation circumstance, we formulate the semi-Markov decision process and derive the optimal software rejuvenation policy minimizing the expected operation cost per unit time in the steady state. Especially, we show analytically that the control-limit type of software rejuvenation policy is optimal.

The rest part of this study is organized as follows. First, we summarize the related work and describe a multistage degradation software system with a CTMC and define the software rejuvenation scheme with the condition-based monitoring. Next, we formulate the cost minimization problem via the familiar

Corresponding Author : Tadashi Dohi, Professor, Department of Information Engineering, Hiroshima University, Japan

semi-Markov decision process and propose an iteration algorithm to derive the optimal software rejuvenation policy which minimizes the expected operation cost per unit time in the steady state. Thirdly, we study the optimality structure carefully and characterize some mathematical properties of the optimal software rejuvenation policy under several parametric assumptions. Numerical examples are presented to make a decision table and to perform the sensitivity analysis of cost parameters. Finally, the study is concluded with some remarks.

RELATED WORK

Huang *et al.*[5] consider a degradation phenomenon as a two-step stochastic process. From the clean state the software system jumps into a degraded state from which two actions are possible: rejuvenation with return to the clean state or transition to the complete failure state. They model a four-state process as a CTMC and derive the steady-state system availability and the expected operation cost per unit time in the steady state. Avritzer and Weyuker[11] discuss the aging in a telecommunication switching software where the effect manifests as gradual performance degradation. Garg *et al.*[7] introduce the idea of periodic rejuvenation (deterministic interval between successive rejuvenations) into the Huang *et al.*'s model[5] and represent the stochastic behavior by using a Markov regenerative stochastic Petri net. Dohi *et al.*[6] and Suzuki *et al.*[8] extend the seminal two-step software degradation models in Huang *et al.*[5] and Garg *et al.*[7], respectively, by using semi-Markov processes.

As other examples, it is interesting to consider both effects of aging as crash/hang failure, referred to as *hard failure* and of aging as *soft failure* that can lead to performance degradation. Pfening *et al.*[12] model a performance degradation process by the gradual decrease of the processing rate in a non-stationary Markovian queueing system and formulate a determination problem of the optimal software rejuvenation schedule by a Markov decision process. Garg *et al.*[13] consider a transaction-based software system, which involves arrival and queueing of jobs and analyze both effects of aging; hard failures that result in an unavailability and soft failures that result in performance degradation. Park and Kim[14] carry out the availability analysis for active/standby cluster systems with rejuvenation. Li *et al.*[15] analyze an aging phenomenon in a real web server application. Liu *et al.*[16] and Vaidyanathan *et al.*[17] model a cable modem termination system and a cluster software system with rejuvenation, respectively. Recently, Xie *et al.*[18] develop a two-level software rejuvenation scheme with service-level rejuvenation and box-level rejuvenation.

Fujio *et al.*[19] and Okamura *et al.*[20] also formulate the control-limit type of rejuvenation policies and compare them with the corresponding time-based policies[13,21] numerically, where the control-limit policy triggers the software rejuvenation at the time instant when the system state reaches to a threshold level, while the time-based policy does at a pre-specified time. As expected intuitively, it is shown in the references[19,20] that the control-limit type of rejuvenation policies can provide better performance than the time-based ones. The main purpose of this study is to prove that the control-limit type of software rejuvenation policy is the best policy among all the Markovian policies in a simple multistage degradation model. In real time applications, actually, the static models[5-8] are difficult for use, because the decision making whether the software rejuvenation should be triggered or not is impossible at an arbitrary timing. On the other hand, the dynamic rejuvenation policy in Pfening *et al.*[12] would be useful to trigger the software rejuvenation sequentially as observing the system state, although they never take account of an event of hard (system) failure. In other words, Pfening *et al.*[12] represent a performance degradation process by the decreasing processing rate in a Markovian queueing system, but we model it by the right-skip free CTMC for a non-transaction based software system. Our approach in this study can be classified into a condition-based preventive maintenance with observation of system state[22].

MULTISTAGE DEGRADATION SOFTWARE SYSTEM

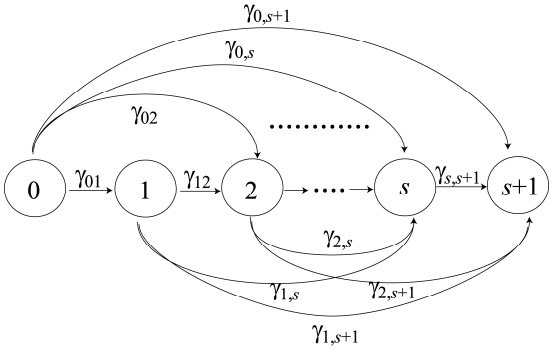

Fig. 1: Markovian transition diagram of software degradation level

Consider an operational software system which deteriorates with time. State of the software system deteriorates stochastically and changes from i to j, where states 0 and $s + 1$ are the normal (robust) state and the system down state, respectively. Without any loss of generality, the level j is degradated more severely than the level i ($< j$). Suppose that the state of software at time t, $\{N(t), t \geq 0\}$, is described by a right right-skip free CTMC with state space $I = \{0, 1, \ldots, s + 1\}$ and that the transition rate from i to j ($i, j = 0, \ldots, s +$

1, $i < j$) is given by $\gamma_{i,j}$ (> 0), where $\sum_{j=i+1}^{s+1} \gamma_{i,j} = \Gamma_i$ for all i (= 0, 1, …, s) (Fig. 1). That is, it is assumed that the system state with degradation level i can make a transition to all the upper levels j (> i). When the system failure occurs, then the system is down ($j = s + 1$) and the recovery operation immediately starts, where the time to complete the recovery operation is an independent and identically distributed (i.i.d.) random variable having the cumulative distribution function (c.d.f.) $H_{s+1}(x)$ and mean $1/\omega_{s+1}$ (> 0).

On the other hand, one makes a decision whether to trigger the software rejuvenation at the time instant when the state of software system changes from i to j (= $i + 1$, $i + 2$, …, s). If one decides to continue operation, the state is monitored until the next change of state, otherwise, the software rejuvenation is preventively triggered, where the time to complete the rejuvenation is also an i.i.d. random variable with the c.d.f. $H_i(x)$ and mean $1/\omega_i$ (> 0), depending on the state i (= 0, 1, …, s). Let x_1 (> 0) and x_2 (> 0) be the rejuvenation cost per unit time and the recovery cost per unit time, respectively. In both periods of rejuvenation and recovery operation, the system operation is stopped. Also, it is assumed that the state-dependent cost a_i (> 0) is incurred per unit operation time for i = 0, 1, …, s.

Note that the system state can be described by only the index j ($0 < j \le s + 1$). At each time instant when the state changes from i to j, one has an option to choose Action 1 (rejuvenation) or Action 2 (continuation of processing). When the system failure occurs, *i.e.* the state of system becomes $j = s + 1$, the recovery operation (Action 3) is taken. Let $Q^{(\delta)}(i, j)$ denote the probability that the state changes from i to j under Action δ (= 1, 2, 3). Then it is seen that

(i) Case 1:

$$Q^{(1)}(i,0) = \int_0^\infty dH_i(t) = 1, \quad i = 0, 1, \cdots, s, \tag{1}$$

where the mean rejuvenation time (overhead) is given by

$$h_i = \int_0^\infty t dH_i(t). \tag{2}$$

(ii) Case 2:

$$Q^{(2)}(i, j) = \gamma_{i,j} / \Gamma_i, \quad i, j = 0, 1, \cdots, s+1; i < j. \tag{3}$$

(iii) Case 3:

$$Q^{(3)}(s+1,0) = \int_0^\infty dH_{s+1}(t) = 1, \tag{4}$$

where the mean recovery time (overhead) is given by

$$h_{s+1} = \int_0^\infty t dH_{s+1}(t). \tag{5}$$

After completing rejuvenation and recovery operations, the state of software system becomes as good as new, *i.e.* $j = 0$ in Eqs. (1) and (4) and the same cycle repeats again and again over an infinite time horizon. We define the time interval from the initial point to the completion of rejuvenation or recovery operation whichever occurs first, as one cycle.

SEMI-MARKOV DECISION PROCESS

Observing the state of software system, we sequentially determine the optimal timing to trigger the software rejuvenation so as to minimize the expected operation cost per unit time in the steady state. Define the following cost component:

$V(i)$: value function at state $i \in I$,

$M(i)$: minimum expected cumulative cost when Action 1 is taken at state i (< $s + 1$),

$W(i)$: minimum expected cumulative cost when Action 2 is taken at state i (< $s + 1$),

$\mathbf{z}$: expected operation cost per unit time in the steady state, where $\mathbf{z}^*$ denotes the minimum one,

$D(i)$: action space at state i:

$$D(i) = \begin{cases} 1: & M(i) \le W(i), i < s+1 \\ 2: & M(i) > W(i), i < s+1 \\ 3: & i = s+1. \end{cases} \tag{6}$$

From the preliminary above, the Bellman equation based on the principle of optimality[10] is given by

$$V(i) = \min\{M(i), W(i)\}, \tag{7}$$

where

$$V(s+1) = x_2 h_{s+1} + V(0) - \mathbf{z}^* h_{s+1}, \tag{8}$$

$$M(i) = x_1 h_i + V(0) - \mathbf{z}^* h_i, \tag{9}$$

$$W(i) = a_i / \Gamma_i + \sum_{j=i+1}^{s+1} \frac{\gamma_{i,j} V(j)}{\Gamma_i} - \mathbf{z}^* / \Gamma_i. \tag{10}$$

It is well known that the software rejuvenation policy satisfying Eq. (7) is the best policy among all the Markovian policies[10]. To solve the above functional equation numerically, we can easily develop the well-known value iteration algorithm for the semi-Markov decision process. Define

$A(n) = \min_{i \in I}\{V^n(i) - V^{n-1}(i)\}$,

$B(n) = \max_{i \in I}\{V^n(i) - V^{n-1}(i)\}$,

ε : tolerance level,

τ : design parameter satisfying $0 \le \tau / h_i$ for all i, $\tau \Gamma_i$ and $\tau / h_{s+1} \le 1$ (Tijms[10]).

In general, $V^n(i)$ denotes the value function at n-th iteration. Then, the value iteration algorithm is given in the following:

Value Iteration Algorithm:

Step 1:

$$n := 0, \; V^0(i) := M^0(i) := W^0(i) := 0. \tag{11}$$

Step 2:

$$W^{n+1}(i) := a_i + \sum_{j=0}^{s+1} \frac{\tau \gamma_{i,j} V^n(j)}{\Gamma_i} + \left(1 - \sum_{j=0}^{s+1} \frac{\tau \gamma_{i,j}}{\Gamma_i}\right) V^n(i), \tag{12}$$

$$M^{n+1}(i) := x_1 + \left(\frac{\tau}{h_i}\right) V^n(0) + \left(1 - \frac{\tau}{h_i}\right) V^n(i), \tag{13}$$

$$V^{n+1}(i) := \min\{M^{n+1}(i), W^{n+1}(i)\}, \quad (14)$$

$$V^{n+1}(s+1) := x_2 + \left(\frac{\tau}{h_{s+1}}\right)V^n(0) + \left(1-\frac{\tau}{h_{s+1}}\right)V^n(s+1). \quad (15)$$

Step 3: If $0 \le B(n) - A(n) \le \varepsilon A(n)$, then stop the procedure, otherwise, $n := n + 1$ and go to **Step 1**.

It would be possible to derive the optimal software rejuvenation schedule by applying the above value iteration algorithm, if there exists a unique optimal solution. However, it is worth noting that an analytical approach to characterize the optimal rejuvenation policy, without solving the Bellman equation directly, is possible by making some parametric (but reasonable) assumptions. In the following section, we investigate some mathematical properties for the optimal software rejuvenation policy and prove its optimality.

OPTIMALITY OF CONTROL-LIMIT POLICY

We prove the optimality of the control-limit type of policy. We make the following assumptions:

(A-1) Γ_i is monotonically increasing in i (=0,1, …, s+1).
(A-2) For an arbitrary increasing function f_j, $\Sigma_{j=i+1}^{s+1}\gamma_{i,j}f_j/\Gamma_i$ is monotonically increasing in i (= 0, 1, …, j - 1).
(A-3) For an arbitrary x, $\overline{H}_{s+1}(x) > \overline{H}_i(x)$, where in general $\overline{H}(\cdot) = 1 - H(\cdot)$.
(A-4) a_i/Γ_i is monotonically increasing in i (= 0, 1, …, s).
(A-5) $a_i/\Gamma_i - x_1h_i$ in monotonically increasing in i (= 0, 1, …, s).
(A-6) $x_2 - \mathbf{z} \ge x_1 - \mathbf{z} \ge 0$.

The assumption (A-1) implies that the mean sojourn time in each state decreases, as the system deteriorates. The assumption (A-2) seems to be somewhat technical, but is intuitively reasonable. For instance, let f_j be any cost parameter depending on state j. In this case, the expected cost incurred when the system state makes a transition, $\Sigma_{j=i+1}^{s+1}f_j(\gamma_{i,j}/\Gamma_i)$, tends to increase as the degraded level i progresses. In the assumption (A-3), one expects in the sense of probability that the recovery time from system failure is strictly greater than the rejuvenation overhead. The assumption (A-4) means that the operation cost increases, but according to (A-5) the advantage of triggering rejuvenation of the software increases gradually, as the software system deteriorates. In the assumption (A-6), we require that the recovery cost is greater than the rejuvenation cost, where the both cost parameters, x_1 and x_2, are greater than the expected operation cost per unit time in the steady state, say, $\mathbf{z}$. In fact, if $\mathbf{z} > x_2 > x_1$, the system is down in the steady state with probability one. That is, due to the deductive argument, the assumption (A-6) has to necessarily hold.

We give the main results of this study.

Lemma 4.1: The function $V(i)$ is increasing in i.

Proof: It is evident from *(A-3)* and *(A-6)* to show that

$$M(s) = x_1h_s + V(0) - \mathbf{z}h_s, \quad (16)$$

$$V(s+1) = x_2h_{s+1} + V(0) - \mathbf{z}h_{s+1}. \quad (17)$$

Hence we have $V(s) \le V(s+1)$ immediately. Supposing that $V(i+1) \le V(i+2) \le \ldots \le V(s) \le V(s+1)$ for an arbitrary i, from *(A-2)*, it can be seen that

$$\sum_j \frac{\gamma_{i,j}}{\Gamma_i}V(j) \le \sum_j \frac{\gamma_{i+1,j}}{\Gamma_{i+1}}V(j). \quad (18)$$

In Case 1 with $D(i+1) = 1$, from the assumption *(A-6)* we have

$$V(i+1) - V(i) \ge x_1(h_{i+1} - h_i) - \mathbf{z}(h_{i+1} - h_i) \ge 0. \quad (19)$$

This implies that $V(i+1) \ge V(i)$. In Case 2 with $D(i+1)=2$, we obtain

$$\begin{aligned} V(i+1) - V(i) &\ge a_{i+1}/\Gamma_{i+1} + \sum_j \frac{\gamma_{i+1,j}}{\Gamma_{i+1}}V(j) - \mathbf{z}/\Gamma_{i+1} \\ &\quad - a_i/\Gamma_i - \sum_j \frac{\gamma_{i,j}}{\Gamma_i}V(j) + \mathbf{z}/\Gamma_i \\ &= (a_{i+1}/\Gamma_{i+1} - a_i/\Gamma_i) + (\mathbf{z}/\Gamma_i - \mathbf{z}/\Gamma_{i+1}) \\ &\quad + \left\{\sum_j \frac{\gamma_{i+1,j}}{\Gamma_{i+1}}V(j) - \sum_j \frac{\gamma_{i,j}}{\Gamma_i}V(j)\right\} \\ &\ge 0, \end{aligned} \quad (20)$$

which is due to *(A-1)*, *(A-2)* and *(A-4)*. Thus, it can be shown that $V(i) \le V(i+1)$. From the inductive argument, it can be proved that $V(i) \le V(i+1)$ for an arbitrary i.

Theorem 4.2: There exists the optimal control limit N^* + 1 satisfying

$$D(i) = \begin{cases} 1: & otherwise \\ 2: & i \le N^* \end{cases} \quad (21)$$

under the assumptions *(A-1)-(A-6)*.

Proof:

$$\begin{aligned} W(i) - M(i) &= a_i/\Gamma_i + \sum_j \frac{\gamma_{i,j}}{\Gamma_i}V(j) - \mathbf{z}/\Gamma_i - x_1h_i - V(0) + \mathbf{z}h_i \\ &= (a_i/\Gamma_i - x_1h_i) + \sum_j \frac{\gamma_{i,j}}{\Gamma_i}V(j) \\ &\quad - \mathbf{z}/\Gamma_i - V(0) + \mathbf{z}h_i. \end{aligned} \quad (22)$$

From *(A-1)*, *(A-2)*, *(A-5)* and Lemma 4.1, it is seen that $W(i) - M(i)$ is a monotonically increasing function of i. Hence, the proof is completed.

From Theorem 4.2, the problem can be reduced to obtain the optimal control-limit N^* + 1 so as to minimize the expected operation cost per unit time in the steady state. In fact, this type of rejuvenation policy

is essentially the same as the workload-based rejuvenation policy in the transaction-based software system[20]. In other words, even for our non-queueing system framework, the control-limit type of rejuvenation policy is better than any time-based one[13,21].

Next, we formulate the expected operation cost as a function of N, *i.e.* $\mathbf{z} = \mathbf{z}(N)$. Define

$$R_{i,j} = \begin{cases} 1 & i = j, \\ \sum_{k=i+1}^{j} \gamma_{i,k} R_{k,j} / \Gamma_i & \text{otherwise}, \end{cases} \tag{23}$$

$$t_i = \begin{cases} \sum_{j=i+1}^{N} R_{i,j} / \Gamma_j & i \le N, \\ 0 & i > N, \end{cases} \tag{24}$$

where $R_{i,j}$ is the transition probability from the state i to the state j and t_i denotes the mean time to trigger the software rejuvenation. Define the first passage time:

$$T^* = \inf\{t \ge 0 : N(t) \ge N^* + 1\}, \tag{25}$$

so that the random variable T^* is the first time when $N(t)$ is greater than the level $N^* + 1$. Using the above notation, we can get the following result without the proof.

Theorem 4.3: The optimal software rejuvenation time is given by the first passage time T^*, where the optimal threshold level N^* is the solution of $\min_{0 \le N < \infty} \mathbf{z}(N)$ and

$$\mathbf{z}(N) = \frac{\sum_{j=1}^{N} \frac{R_{0,j}}{\Gamma_i} \left\{ a_j + \sum_{k=N+1}^{s} x_1 \gamma_{j,k} h_k + x_2 \gamma_{j,s+1} h_{s+1} \right\}}{t_0 + \sum_{j=1}^{N} \sum_{k=N+1}^{s} \frac{R_{0,j}}{\Gamma_i} \{ \gamma_{j,k} h_k + \gamma_{j,s+1} h_{s+1} \}}. \tag{26}$$

In Eq. (26), the function $\mathbf{z}(N)$ is formulated as the expected operation cost for one cycle divided by the mean time length of one cycle. The minimization problem of $\mathbf{z}(N)$ with respect to N (= 0, 1, 2, …) is trivial. Define the difference of $\mathbf{z}(N)$ by $\phi(N) = \mathbf{z}(N+1) - \mathbf{z}(N)$. If $\phi(N+1) - \phi(N) > 0$, then the function $\mathbf{z}(N)$ is strictly convex in N. Further, if $\phi(N^*-1) > 0$ and $\phi(N^*) \le 0$, then there exist (al least one, at most two) optimal threshold level N^* which minimizes $\mathbf{z}(N)$, *i.e.*, $\mathbf{z}^* = \mathbf{z}(N^*)$. In fact, the convex property of the function $\mathbf{z}(N)$ can be easily checked numerically.

Figure 2 depicts the behavior of the expected operation cost with respect to N, where the model parameters used here are given by $\gamma_{0,1} = \gamma_{0,2} = \gamma_{1,2} = 0.5$, $\gamma_{0,3} = \gamma_{0,6} = \gamma_{2,6} = 0.3$, $\gamma_{0,4} = \gamma_{0,5} = \gamma_{1,5} = 0.2$, $\gamma_{1,3} = \gamma_{1,4} = 0.6$, $\gamma_{1,6} = \gamma_{5,5} = 0.1$, $\gamma_{2,3} = \gamma_{2,4} = \gamma_{3,6} = 0.7$, $\gamma_{3,4} = 0.4$, $\gamma_{3,5} = 0.9$, $\gamma_{4,5} = \gamma_{4,6} = 1$, $\gamma_{5,6} = 1.9$, $x_1 = 8$ (\$), $x_2 = 15$ (\$), $a_0 = 1$ (\$), $a_1 = 3$ (\$), $a_2 = 5$ (\$), $a_3 = 7$ (\$), $a_4 = 9$ (\$), $a_5 = 11$ (\$), $\omega_0 = 2.0$ (hr^{-1}), $\omega_1 = 1.5$ (hr^{-1}), $\omega_2 = 1.2$ (hr^{-1}), $\omega_3 = 1.0$ (hr^{-1}), $\omega_4 = 0.7$ (hr^{-1}), $\omega_5 = 0.5$ (hr^{-1}), $\omega_6 = 0.2$ (hr^{-1}). It can be shown that there is a unique optimal threshold level $N^* = 1$ in this example.

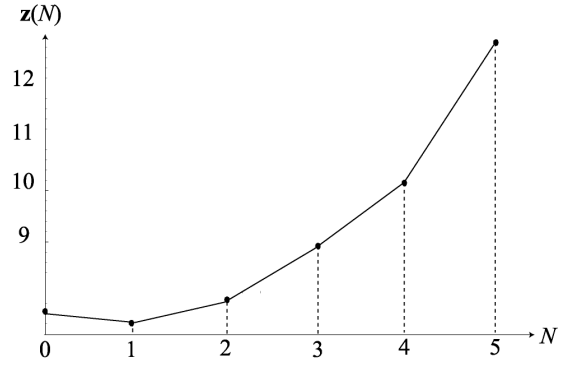

Fig. 2: Behavior of the expected operation cost with respect to N

NUMERICAL EXAMPLES

Here we give an illustrative example to determine the optimal rejuvenation schedule. Huang *et al.*[5] and Dohi *et al.*[6,8] suppose that the number of system states is only 3: normal operation, deterioration (failure probable state) and system down (failure). In this section we consider a generalized Markovian deterioration process with 3 degradation levels (totally, 5 states). Figure 3 illustrates the Markovian transition diagram with absorption, representing the deterioration process of the software system, where each transition rate is assigned on each arc. The transition probability that the system state changes from k to l at time t, denoted by $T_{k,l}(t)$, satisfies the Chapman-Kolmogorov equation. Since the system state follows a right-skip free CTMC, the transition probability $T_{k,l}(t)$ can be derived analytically. For instance, when $(k, l) = (0, 0)$, we have the following differential equation:

$$\frac{d}{dt} T_{0,0}(t) = -T_{0,0}(t). \tag{27}$$

Solving the above equation with the initial condition $T_{k,k}(0) = 1$, we get $T_{0,0}(t) = e^{-t}$. Similarly, we have: $T_{0,1}(t) = 0.4te^{-t}$, $T_{0,2}(t) = 0.3te^{-t} + 0.08t^2e^{-t}$, $T_{0,3}(t) = 0.2te^{-t} + 0.17t^2e^{-t} + 0.016t^3e^{-t}$, $T_{0,4}(t) = -e^{-t} - 0.9te^{-t} - 0.25t^2e^{-t} - 0.016t^3e^{-t} + 1$, $T_{1,1}(t) = e^{-t}$, $T_{1,2}(t) = 0.4te^{-t}$, $T_{1,3}(t) = 0.2te^{-t} + 0.12t^2e^{-t}$, $T_{1,4}(t) = -0.8e^{-t} - 0.6te^{-t} - 0.12t^2e^{-t} + 0.8$, $T_{2,3}(t) = 0.6te^{-t}$, $T_{2,4}(t) = -e^{-t} - 0.6te^{-t} + 1$, $T_{3,4}(t) = e^{-t} + 1$.

Table 1: Decision table

i	0	1	2	3	4
$D(i)$	2	2	1	1	3

Suppose that the rejuvenation cost and the recovery cost from system failure are given by $x_1 = 8$ (\$) and $x_2 = 15$ (\$), respectively. Also, it is assumed that $a_0 = 1$ (\$), $a_1 = 3$ (\$), $a_2 = 5$ (\$), $a_3 = 7$ (\$), $\omega_0 = 2.0$ (hr^{-1})\$, $\omega_1 =$

1.5 (hr^{-1}), ω_2 = 1.2 (hr^{-1}), ω_3 = 1.0 (hr^{-1}), ω_4 = 0.2 (hr^{-1}). In Table 1, we obtain the so-called *decision table*

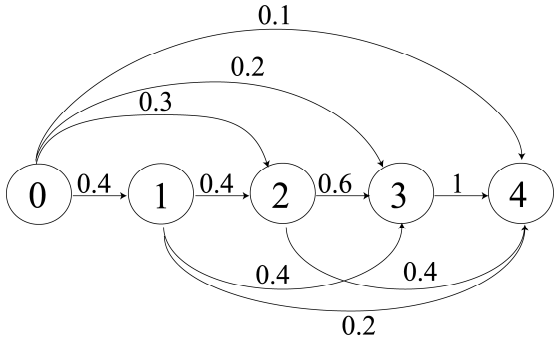

Fig. 3: An illustrative example with three degradation levels

Table 2: Dependence of cost ratio x_1/x_2 on the optimal rejuvenation policy and its associated expected operation cost

x_1/x_2	$i = N^* + 1$	$\mathbf{z}(N^*)$
7/15	2	4.021
8/15	2	4.255
9/15	2	4.489
10/15	2	4.722
11/15	3	4.956
12/15	3	5.168
13/15	3	5.364
14/15	3	5.543
15/15	3	5.675

to characterize the optimal software rejuvenation policy. From this result, it is optimal to trigger the software rejuvenation at the first time when the system state reaches to $i = N^* + 1 = 2$. Then the associated minimum expected operation cost per unit time in the steady state is given by $\mathbf{z}^* = \mathbf{z}(N^*) = \mathbf{z}(1) = 4.255$ ($).

Next, we examine the dependence of the cost ratio x_1/x_2 on the optimal software rejuvenation policy. Table 2 presents the dependence of x_1/x_2 on the expected operation cost, where the corresponding optimal threshold level N^* is insensitive to the change of x_1/x_2 in the both ranges of 7/15 ~ 10/15 and 11/15 ~ 15/15. This is because the optimal threshold level is given by an integer value. As the rejuvenation cost per unit time is relatively larger with a fixed recovery cost parameter, the expected operation cost also increases monotonically for larger rejuvenation cost.

CONCLUSION

In this study, we have considered a dynamic rejuvenation policy for a multistage degradation software system. We have formulated the underlying optimization problem by a semi-Markov decision process and proved the optimality of control-limit type of software rejuvenation policy. A numerical example has been presented to illustrate the dynamic rejuvenation policy and its associated expected operation cost per unit time in the steady state. Here we have derived the decision table to characterize the optimal policy and performed the sensitivity analysis of cost parameters on it.

The result can be applied to the preventive maintenance problem with garbage collection for an application software, if the degradation level can be quantified by the total amount of memory leak. Based on the optimality of control-limit policy, the software user monitors the level of resource exhaustion and can trigger the garbage collection at the best timing in terms of the cost minimization. Of course, it is essentially important to estimate the transition rate from the field data with higher accuracy. If one fails to collect such data on the degradation time, the applicability of our stochastic model to the real software fault management can not be limited. Also, it is worth noting that the optimality of control-limit policy can not be guaranteed if the assumptions *(A-1)-(A-6)* do not hold, so that these parametric assumptions have to be checked carefully in practice.

In future, we will consider an adaptive software rejuvenation policy in the same modeling framework as this study. In practice, it is not so easy for an arbitrary software user to model software aging phenomena (number of degradation states and transition architecture) and to estimate the transition rates from his or her operational experience. To challenge the above issue, adaptive algorithms like non-parametric statistics and reinforcement learning should be applied to design an autonomic rejuvenation protocol with adaptive prediction ability.

ACKNOWLEDGMENTS

This study is an extended version of the reference [23]. This research was partially supported by the Ministry of Education, Science, Sports and Culture: Grant-in-Aid for Exploratory Research, Grant No.15651076 (2003-2005).

REFERENCES

1. Gray, J., 1986. Why do computers stop and what can be done about it? Proc. 5th Intl. Symp. on Reliab. Distributed Software and Database Systems, pp: 3-12.
2. Adams, E., 1984. Optimizing preventive service of the software products. IBM J. Res. & Develop., 28: 2-14.
3. Castelli, V., R.E. Harper, P. Heidelberger, S.W. Hunter, K.S. Trivedi, K.V. Vaidyanathan and W.P. Zeggert, 2001. Proactive management of software aging. IBM J. Res. & Develop., 45: 311-332.
4. Dohi, T., K. Goševa-Popstojanova, K. Vaidyanathan, K.S. Trivedi and S. Osaki, 2003. Software Rejuvenation Modeling and Applications. Handbook of Reliability Engineering (Ed. H. Pham), pp: 245-268. Springer-Verlag, London.
5. Huang, Y., C. Kintala, N. Kolettis and N.D. Fulton, 1995. Software Rejuvenation: Analysis, Module and Applications. Proc. 25th Intl. Symp. on Fault Tolerant Computing, pp: 381-390.

6. Dohi, T., K. Goševa-Popstojanova and K.S. Trivedi, 2001. Estimating software rejuvenation schedule in high assurance systems. The Computer J., 44: 473-485.
7. Garg, S., M. Telek, A. Puliafito and K.S. Trivedi, 1995. Analysis of software rejuvenation using Markov regenerative stochastic Petri net. Proc. 6th Intl. Symp. on Software Reliab. Eng., pp: 24-27.
8. Suzuki, H., T. Dohi, K. Goševa-Popstojanova and K.S. Trivedi, 2002. Analysis of Multi Step Failure Models with Periodic Software Rejuvenation. In: Advances in Stochastic Modelling (Eds. J.R. Artalejo and A. Krishnamoorthy), pp: 85-108. Notable Publications, Neshanic Station.
9. Vaidyanathan, K., D. Selvamuthu and K.S. Trivedi, 2002. Analysis of inspection-based preventive maintenance in operational software systems. Proc. 21st IEEE Symp. on Reliable Distributed Systems, pp: 286-295.
10. Tijms, H.C., 1994. Stochastic Models: An Algorithmic Approach. John Wiley & Sons, New York.
11. Avritzer, A. and E.J. Weyuker, 1997. Monitoring smoothly degrading systems for increased dependability. Empirical Software Eng., 2: 59-77.
12. Pfening, S., S. Garg, A. Puliafito, M. Telek and K.S. Trivedi, 1996. Optimal rejuvenation for tolerating soft failure. Performance Evaluation, 27/28: 491-506.
13. Garg, S., S. Pfening, A. Puliafito, M. Telek and K.S. Trivedi, 1998. Analysis of preventive maintenance in transactions based software systems. IEEE Trans. on Computers, 47: 96-107.
14. Park, K. and S. Kim, 2002. Availability analysis and improvement of active/standby cluster systems using software rejuvenation. J. Systems and Software, 61: 121-128.
15. Li, L., K. Vaidyanathan and K.S. Trivedi, 2002. An approach for estimation of software aging in a web server. Proc. 2002 Intl. Symp. on Empirical Software Eng., pp: 91-100.
16. Liu, Y., Y. Ma, J.J. Han, H. Levendel and K.S. Trivedi, 2002. Modeling and analysis of software rejuvenation in cable modem termination System. Proc. 13th Intl. Symp. on Software Reliab. Eng., pp: 159-170.
17. Vaidyanathan, K., R.E. Harper, S.W. Hunter and K.S. Trivedi, 2001. Analysis of software rejuvenation in cluster systems. Proc. ACM SIGMETRICS 2001/Performance, pp: 62-71.
18. Xie, W., Y. Hong and K.S. Trivedi, 2005. Analysis of a two-level software rejuvenation policy. Reliab. Eng. Sys. Safety, 87: 13-22.
19. Fujio, H., H. Okamura and T. Dohi, 2003. Fine-grained shock models to rejuvenate software systems. IEICE Trans. on Information and Systems (D), E86-D: 2165-2171.
20. Okamura, H., S. Miyahara, T. Dohi and S. Osaki, 2001. Performance evaluation of workload-based software rejuvenation scheme. IEICE Trans. on Information and Systems (D), E84-D: 1368-1375.
21. Bobbio, A., M. Sereno and C. Anglano, 2001. Fine grained software degradation models for optimal rejuvenation policies. Performance Evaluation, 46: 45-62.
22. Chen, D. and K.S. Trivedi, 2005. Optimization for condition-based maintenance with semi-Markov decision process. Reliab. Eng. Sys. safety, 90: 25-29.
23. Eto, H. and T. Dohi, 2005. Optimality of control-limit type of software rejuvenation policy. Proc. IEEE 11th Intl. Conf. on Parallel and Distributed Systems, II: 483-487.