Availability Optimization in Operational Software System with Aperiodic Time-Based Software Rejuvenation Scheme

Hiroyuki Okamura and Tadashi Dohi
Department of Information Engineering
Graduate School of Engineering, Hiroshima University
1–4–1 Kagamiyama, Higashi-Hiroshima 739–8527 Japan
Email: {okamu,dohi}@rel.hiroshima-u.ac.jp

***Abstract*—This paper discusses an aperiodic time-based rejuvenation policy maximizing the steady-state system availability in operational software system. Under a fixed periodic checkpoint schedule, we develop an algorithm to derive the optimal aperiodic rejuvenation times based on dynamic programming. In numerical examples, the optimal rejuvenation time and its maximum availability are illustrated in the case where the system failure obeys the Weibull distribution.**

I. Introduction

It has been recognized for a long time that the software system never deteriorates during operational phases. However, the phenomenon, called *software aging* [1], [2], is often observed in operating systems and widely-used applications. The software aging causes performance degradation of operational software system and eventually gives rise to software errors.

Huang et al. [3] reported the software aging in a telecommunication billing application where over time the application experiences a crash or a hang failure. The aging phenomenon in a telecommunication switching software was observed in Avritzer and Weyuker [4], where the effect manifests as gradual performance degradation. Garg et al. [5], Shereshevsky et al. [6], Vaidyanathan et al. [7] conducted empirical researches to measure memory resource exhaustion and aging phenomena in operational software systems.

In fact, software aging is caused by undetected software faults, and the aging phenomenon can be observed as performance degradation like memory leaks, heap corruption and data fragmentation. According to common experiences, it is seen that almost all of the errors due to the aging are transient [8].

As is well known, transient errors are reproducibly removed by changing the operational environment with retry, restart and reboot. In other words, since there is no reproducibility of transient errors, the faults that lead to transient errors are quite difficult to detect. Therefore, the software aging and its associated transient errors have to be tolerated in operational software system.

978-1-4244-3417-6/08/$25.00

A complementary approach to handle transient software errors is *software rejuvenation* [3]. Software rejuvenation is a preventive and proactive solution that is particularly useful for counteracting the software aging. It involves stopping the running software occasionally, cleaning its internal state and restarting it. Typical examples of cleaning the internal state of software are garbage collection, flushing operating system kernel tables, reinitializing internal data structures, etc. An extreme, but well-known example of rejuvenation is a hardware reboot. Apart from being used in an ad-hoc manner by almost all computer users, the software rejuvenation has been used in high availability and mission critical systems [9]. The most vivid example of aging in safety critical systems is the Patriot's software [10], where the accumulated errors led to a failure that resulted in loss of human life.

Software rejuvenation incurs time and load overhead to prevent more severe errors. In general, the rejuvenation operation needs to interrupt the running service, but the rejuvenation cost is relatively much lower than that of the recovery operation caused by unexpected errors. From this point of view, many authors discussed rejuvenation scheduling problem; when we should trigger the software rejuvenation.

Huang et al. [3] proposed a continuous-time Markov chain model with random software rejuvenation. Dohi et al. [11], [12] generalized the same model to semi-Markov models with different dependability measures and developed the computation methods of the optimal software rejuvenation schedule. As an alternative modeling approach, the work in Garg et al. [13] involved arrival and service of transactions in the system, and computed the load and time-based rejuvenation schedule by taking account of the effect of aging as crash/hang failure, referred to as hard failures, and performance degradation, referred to as soft failures.

This paper deals with a somewhat different rejuvenation scheduling problem from existing works. More specifically, here we consider an aperiodic time-based rejuvenation policy. In most of the other existing works, the rejuvenation policy is assumed to be stationary, i.e., all the policies can be categorized to periodic time-based or workload-based policies. However, when the rejuvenation takes place over a finite time

horizon, the optimal rejuvenation policy is no longer stationary. In such a case, we need aperiodic rejuvenation scheme to reduce rejuvenation and recovery overheads. Okamura et al. [14] considered the situation where one running process can be rejuvenated at several time points, and derived an aperiodic rejuvenation sequence to minimize total computation time based on dynamic programming. In this paper, we suppose that the software system has more priority to the rejuvenation than to the checkpointing. This assumption intuitively implies that the rejuvenation overhead is lower than that incurred by checkpointing. For example, when the rejuvenation is a process restart, the rejuvenation overhead is not so expensive. In such a case, this paper considers a different scheduling problem; how often we should do the software rejuvenation for a given checkpoint interval.

In Section 2, we describe a stochastic model and formulate the steady-state availability under an aperiodic rejuvenation policy. Section 3 describes the dynamic programming approach to obtain the optimal rejuvenation time sequence maximizing the system availability. Numerical examples are presented in Section 4. In these examples, we illustrate the aperiodic optimal rejuvenation time sequence in the case where the system failure follows the Weibull distribution. Finally the paper is concluded with some remarks in Section 5.

II. Software Availability Modeling

A. Model Description

Consider an operational software system with software aging phenomenon. Suppose that the system generates checkpoints at every fixed time interval to prevent data or computation loss. Without loss of generality, we assume that the last checkpoint is placed at $t = 0$.

Let $F(t)$ be the system failure time distribution which has the probability density function and the mean are given by $f(t)$ and μ_f (> 0), respectively. Since the system or software resources deteriorate in the software system with aging, we suppose that $F(t)$ is IFR (Increasing Failure Rate) and that the failure rate $r(t) = f(t)/\overline{F}(t)$ is increasing in t, where in general $\overline{\psi}(\cdot) = 1 - \psi(\cdot)$. The system is as good as new at $t = 0$. This implies that the checkpointing behaves like software rejuvenation as well. Let T (> 0) be a time interval of checkpoints. If the software system does not fail until the time T, the checkpointing is performed and the software system is also rejuvenated with the overhead μ_c.

For the software system, we deal with optimal placement of rejuvenation points; the time when the rejuvenation operation starts without overhead. Let $\boldsymbol{\pi} = \{t_1, t_2, \ldots, t_N\}$ be a sequence of rejuvenation points for one time interval of checkpoints, where N (> 0) is the number of rejuvenation points allowable to be placed at the time interval. Each rejuvenation needs time overhead μ_r (> 0) and the system is as good as new after each completion of the rejuvenation. In this paper, we assume that the rejuvenation is performed by a simple operation like process restarting, so that the rejuvenation overhead is not so expensive.

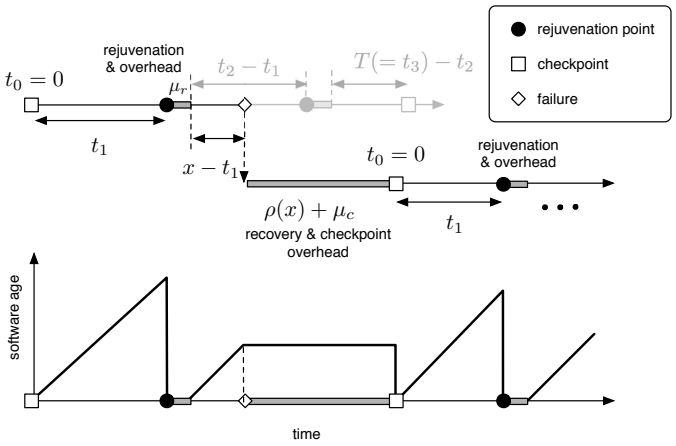

Fig. 1. Possible realization of software system and its age behavior.

When the system failure occurs, the system immediately undergoes rollforward recovery with the information on the system status stored at the last checkpoint. More specifically, if the failure occurs at the time when the elapsed time from the last checkpoint is $x \in (0, T)$, the restoration (recovery) overhead is given by $\rho(x) = \alpha x + \beta$, where αx denotes the time needed to re-execute for the lost computation and the second term is a fixed overhead. Even after the completion of restoration, the system also generates a checkpoint to prevent additional overheads caused by additional system failures.

Figure 1 illustrates relationship between transition of system states and system aging. We define time length from $t = 0$ to T as one cycle. The same cycle repeats again and again over an infinite time horizon. Of course, during the rejuvenation, restoration and checkpointing, the processes are not operated.

B. Formulation of System Availability

Of our concern here is the formulation of steady-state system availability which is defined as the probability that the software system is operative in the steady state. For this purpose, we define the renewal points at which the checkpoints are placed, and focus on the probabilistic behavior between two successive renewal points, i.e., during one cycle. From the familiar renewal reward argument [15], the steady-state system availability with the rejuvenation schedule $\boldsymbol{\pi}$ and the checkpoint time interval T is represented by

$$Ass(\boldsymbol{\pi}, T) = \lim_{t\to\infty} \frac{\mathrm{E}[\text{system operation time during } [0,t)]}{t} = \frac{\mathrm{E}[\text{system operation time during one cycle}]}{\mathrm{E}[\text{time length of one cycle}]}, \quad (1)$$

where E denotes the mathematical expectation operator. Then, the problem is to find the optimal maintenance schedule $(\boldsymbol{\pi}^*, T^*)$ maximizing the steady-state system availability $Ass(\boldsymbol{\pi}, T)$.

For the sake of simplicity, we rewrite the optimal maintenance schedule as $\tilde{\boldsymbol{\pi}} = \{t_0, t_1, \ldots, t_N, t_{N+1}\}$, where $t_0 = 0$ and $t_{N+1} = T$. During the interval between two successive rejuvenation points $[t_{i-1}, t_i)$, we derive the expected up

time (expected operative time) $A(t_i|t_{i-1})$ and the expected total time $S(t_i|t_{i-1})$ during $[t_{i-1}, t_i)$ including rejuvenation, restoration and checkpoint overheads by

$$A(t_i|t_{i-1}) = \int_0^{t_i - t_{i-1}} x dF(x) + (t_i - t_{i-1})\overline{F}(t_i - t_{i-1}), \quad (2)$$

$$S(t_i|t_{i-1}) = \int_0^{t_i - t_{i-1}} \{x + \rho(t_{i-1} + x) + \mu_c\} dF(x) + (t_i - t_{i-1} + \mu_r)\overline{F}(t_i - t_{i-1}), \quad (3)$$

respectively. At the last time period $[t_N, t_{N+1})$, it is noted that the checkpointing is carried out unless the system failure occurs and that it can be regarded as a boundary time period. Then, the expected up time and the expected total time are given by

$$A(t_{N+1}|t_N) = \int_0^{t_{N+1} - t_N} x dF(x) + (t_{N+1} - t_N)\overline{F}(t_{N+1} - t_N), \quad (4)$$

$$S(t_{N+1}|t_N) = \int_0^{t_{N+1} - t_N} \{x + \rho(t_N + x) + \mu_c\} dF(x) + (t_{N+1} - t_N + \mu_c)\overline{F}(t_{N+1} - t_N), \quad (5)$$

respectively. Based on the above results, the steady-state system availability is formulated as a fraction of time when the software system is up during one cycle, and is straightforwardly given by

$$Ass(\tilde{\boldsymbol{\pi}}) = \frac{\sum_{i=1}^{N+1} \prod_{j=1}^{i-1} \overline{F}(t_j - t_{j-1}) A(t_i|t_{i-1})}{\sum_{i=1}^{N+1} \prod_{j=1}^{i-1} \overline{F}(t_j - t_{j-1}) S(t_i|t_{i-1})}. \quad (6)$$

III. DP Algorithm

Since the steady-state system availability is given as a function of $\tilde{\boldsymbol{\pi}}$, the problem is reduced to a non-linear maximization problem $\max_{\tilde{\boldsymbol{\pi}}} Ass(\tilde{\boldsymbol{\pi}})$, provided that the number of rejuvenation points N is given. It is worth noting that there is no effective algorithm to find the optimal pair $(\tilde{\boldsymbol{\pi}}^*, N^*)$ simultaneously, so that the number of rejuvenation points must be carefully adjusted according to a heuristic manner. For a fixed N, the most popular method to find the optimal $\tilde{\boldsymbol{\pi}}^*$ would be the Newton's method or its iterative variant. However, since the Newton's method is a general-purpose non-linear optimization algorithm, it may not often function better to solve the maximization problem with many parameter constraints. In our maximization problem, the decision variables $\tilde{\boldsymbol{\pi}}$ are restricted. For such a sequential optimization problem, it is well known that the dynamic programming (DP) can be used effectively.

The fundamental idea of DP algorithm is to solve recursively the optimality equations which are typical functional equations. Hence, it seems to be straightforward to give the optimality equations which the optimal maintenance schedule $\tilde{\boldsymbol{\pi}}^*$ must satisfy. Suppose that there exists the unique maximum steady-state system availability ξ. From the principle of optimality [16], we obtain the following optimality equations for the maximization problem of the steady-state system availability:

$$h_i = \max_{t_i} W(t_i|t_{i-1}^*, h_1, h_{i+1}), \qquad i = 1, \ldots, N, \quad (7)$$

$$h_{N+1} = \max_{t_{n+1}} W(t_{N+1}|t_N^*, h_1, h_1), \quad (8)$$

where the function $W(t_i|t_{i-1}, h_1, h_{i+1})$ is given by

$$W(t_i|t_{i-1}, h_1, h_{i+1}) = A(t_i|t_{i-1}) - \xi S(t_i|t_{i-1}) + h_1 F(t_i - t_{i-1}) + h_{i+1}\overline{F}(t_i - t_{i-1}) \quad (9)$$

and the function h_i, $i = 1, \ldots, N+1$, are called the *relative value functions*.

Since Eqs. (7) and (8) are necessary and sufficient conditions of the optimal maintenance schedule, the next step is to solve the above optimality equations. In the long history of the DP research, there are a couple of algorithms to solve the optimality equations. In this paper, we apply the *policy iteration* scheme [17] to derive the optimal maintenance scheduling. Our policy iteration consists of two steps; the optimization of the maintenance schedule under a given relative value function and the computation of the relative value function based on the updated maintenance schedule. These steps are repeatedly executed until the resulting maintenance schedule converges to the optimal value.

In the optimization phase, it should be noted that the functions

$$W(t_i|t_{i-1}, h_1, h_{i+1}), \quad i = 1, \ldots, N, \quad (10)$$

are not always concave with respect to the decision variables t_i. Our problem is the case. This fact leads to the difficulty for maximizing the steady-state system availability. In order to overcome this problem, we define the following composite function:

$$W(t_i|t_{i-1}, h_1, W(t_{i+1}|t_i, h_1, h_{i+2})), \quad i = 1, \ldots, N \quad (11)$$

and propose to use it instead of $W(t_i|t_{i-1}, h_1, h_{i+1})$. Because the above composite function is a concave function, it is possible to find the optimal maintenance schedule in each iteration phase by maximizing the composite function for $i = 1, \ldots, N$.

In the computation phase of the relative value function, on the other hand, we solve the following linear system:

$$\boldsymbol{M}\boldsymbol{x} = \boldsymbol{b}, \quad (12)$$

where for a given maintenance schedule, the relative value functions h_i and the maximum steady-state system availability ξ must satisfy where

$$[\boldsymbol{M}]_{i,j} = \begin{cases} -\overline{F}(t_i - t_{i-1}) & \text{if } i = j \text{ and } j \neq N+1, \\ 1 & \text{if } i = j+1, \\ S(t_i|t_{i-1}) & \text{if } j = N+1, \\ 0 & \text{otherwise,} \end{cases} \quad (13)$$

$$\boldsymbol{x} = (h_2, \ldots, h_N, h_{N+1}, \xi)', \quad (14)$$

$$\boldsymbol{b} = (A(t_1|t_0), \ldots, A(t_N|t_{N-1}), A(t_{N+1}|t_N))', \quad (15)$$

where $[\cdot]_{i,j}$ denotes the (i,j)-element of matrix, and the prime ($\prime$) represents transpose of vector. The above results come from the direct application of the optimality equations (7) and (8). Note that $h_1 = 0$, since we are here interested in the relative value function h_i and ξ

Finally, we given the DP algorithm to derive optimal rejuvenation point sequence as follows.

DP Algorithm

- **Step 1:** Give initial values

$$k := 0,$$
$$t_0 := 0,$$
$$\tilde{\boldsymbol{\pi}}^{(0)} := \{t_1^{(0)}, \ldots, t_N^{(0)}, t_{N+1}^{(0)}\}.$$

- **Step 2:** Compute $h_1^{(k)}, \ldots, h_{N+1}^{(k)}, \xi^{(k)}$ for the linear system (12) with the optimal maintenance schedule $\tilde{\boldsymbol{\pi}}^{(k)}$.
- **Step 3:** Solve the following optimization problems:

$$t_i^{(k+1)} := \underset{t_{i-1}^{(k)} \le t \le t_{i+1}^{(k)}}{\text{argmax}} W(t|t_{i-1}^{(k)}, 0, W(t_{i+1}^{(k)}|t_i, 0, h_{i+2}^{(k)})),$$
$$\text{for } i = 0, 1, \ldots, N-1,$$
$$t_N^{(k+1)} := \underset{t_{N-1}^{(k)} \le t \le t_{N+1}^{(k)}}{\text{argmax}} W(t|t_{N-1}^{(k)}, 0, W(t_{N+1}^{(k)}|t, 0, 0)),$$
$$t_{N+1}^{(k+1)} := \underset{t_N^{(k)} \le t < \infty}{\text{argmax}} W(t|t_N^{(k)}, 0, 0).$$

- **Step 4:** For all $i = 1, \ldots, N+1$, if $|t_i^{(k+1)} - t_i^{(k)}| < \delta$, stop the algorithm, where δ is an error tolerance level. Otherwise, let $k := k + 1$ and go to Step 2.

In Step 3, an arbitrary optimization technique has to be applied. Since the composite function is concave in the range $[t_{i-1}, t_{i+1})$, it is relatively easy to calculate the optimal rejuvenation sequence and the optimal preventive maintenance time. In fact, the golden section method [18] would be effective to find the solution. In the following section, we give some numerical examples to calculate the optimal maintenance schedule based on the proposed DP algorithm.

IV. Numerical Illustration

Suppose that the system failure time obeys the Weibull distribution;

$$F(t) = 1 - \exp\left\{-\left(\frac{t}{\eta}\right)^{\phi}\right\}, \tag{16}$$

where η (> 0) and ϕ (> 0) are scale and shape parameters. If $\phi > 1$, then the system failure time distribution is IFR. The mean time to failure (MTTF) and the failure rate for the Weibull distribution are given by $\mu_f = \eta\Gamma(1 + 1/\psi)$ and $r(t) = \phi t^{\phi-1}/\eta^{\phi}$, respectively, where $\Gamma(\cdot)$ is the standard gamma function. In this example, we set $\phi = 2.0$ and adjust the scale parameter so that MTTF equals 1.0. To simplify the analysis, we assume $\alpha = 1$ and $\beta = 0$, and then the restoration overhead is represented by $\rho(x) = x$. This means that the restoration operations require the exactly same time amount as the processing time since the last checkpoint, i.e., the restoration is based on a re-execution. Also, we set the other model parameters $\mu_r = 0$ and $\mu_c = 0.01$, i.e., no rejuvenation overhead is assumed, to clarify the parameter sensitivity. For the purpose to calculate the optimal maintenance schedule, we make a computation program written by C language with GSL (GNU Scientific Library).

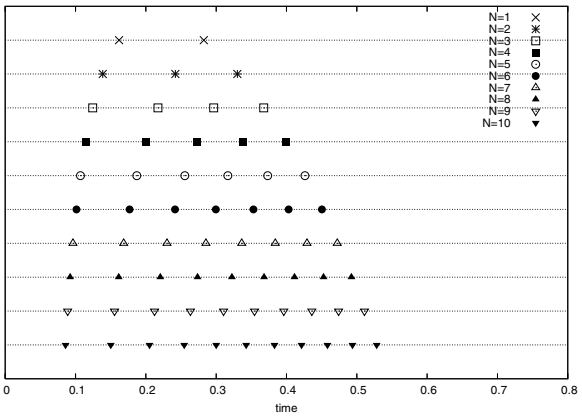

Fig. 2. Optimal maintenance schedule in Weibull failure case ($\mu_r = 0$).

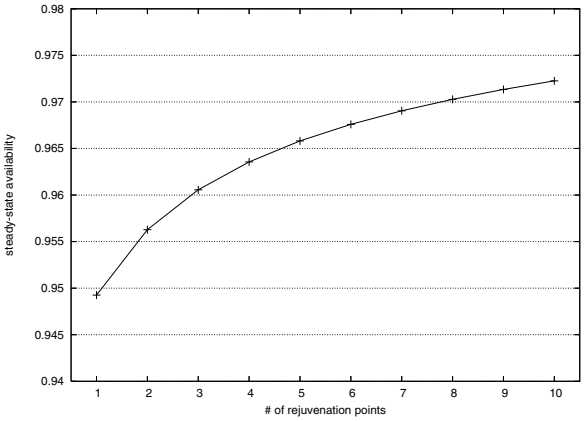

Fig. 3. Steady-state system availabilities under the optimal maintenance schedules in Weibull failure case ($\mu_r = 0$).

Figure 2 illustrates the optimal maintenance schedule (rejuvenation points and the next checkpoint). In the figure, the horizontal lines indicate the cases where the numbers of rejuvenation points allowable to be placed for one successive checkpoints are $N = 1, \ldots, 10$. Since the x-axis represents the elapsed time without rejuvenation overheads from the last checkpoint, each point (dot) is the rejuvenation point. On the other hand, the last dot indicates the next checkpoint time, which equivalently means the checkpoint time interval.

From this result, it is seen that the resulting rejuvenation intervals are decreasing time sequence. This is because the restoration overhead linearly increases as the elapsed time from the last checkpoint becomes large. The penalty at late time is larger than the early time, and therefore more rejuvena-

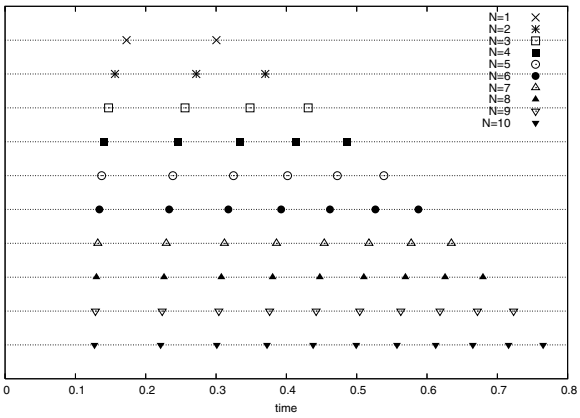

Fig. 4. Optimal maintenance schedule in Weibull failure case ($\mu_r = 0.002$).

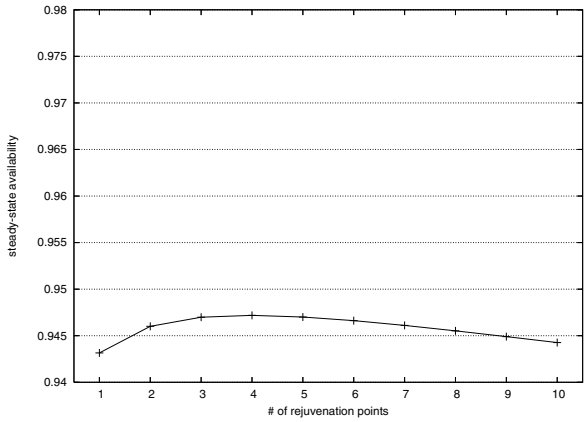

Fig. 5. Steady-state system availabilities under the optimal maintenance schedules in Weibull failure case ($\mu_r = 0.002$).

tion points are needed as time elapses. On the other hand, since no rejuvenation overhead is assumed, the optimal checkpoint interval also gets larger as the number of rejuvenation points increases. However, the optimal checkpoint time interval tends to converge to a certain point (around 0.6 in this example).

Figure 3 presents the maximum steady-state availability under the optimally scheduled maintenance policy. The increase tendency of the maximum availability is similar to that of the optimal checkpoint time interval in Fig. 2. When the number of rejuvenation points is large so that the optimal checkpoint time converges, the maximum availability is also expected to converge to its upper limit.

Next, we examine the dependence of rejuvenation overhead on the optimal maintenance schedule, where $\mu_r = 0.002$ is assumed. Figure 4 presents the respective optimal maintenance schedule in the case of $\mu_r = 0.002$. Likewise the previous example with $\mu_r = 0.0$, it is seen that the optimal rejuvenation intervals form decreasing time sequence. Figure 5 plots the maximized values of the steady-state system availability with respect to the number of rejuvenation points. As different points from Figs. 2 and 3, the optimal checkpoint time intervals are much longer than those in the case of no rejuvenation overhead. On the other hand, the maximum steady-state availability gets an optimal value when the number of rejuvenation points is 4. From these observations, we find that there exists the optimal number of rejuvenation points if the rejuvenation overhead exists.

V. Conclusions

In this paper, we have considered an operational software system with aperiodic rejuvenation scheme. We have proposed a dynamic programming algorithm to determine the optimal maintenance schedule maximizing steady-state system availability. In numerical examples, dependence of the optimal maintenance schedule on several model parameters has been examined. Lessons learned from the numerical examples are that (i) the optimal rejuvenation time forms a decreasing sequence, (ii) the optimal checkpoint interval is bounded even if the rejuvenation cost is zero, (iii) there exists the optimal number of rejuvenation points in the case where the rejuvenation overhead is not zero.

In future, we will compare between the results of aperiodic checkpointing scheme [19] and the proposed one, and will examine the effectiveness of checkpointing and rejuvenation in a unified system availability model. In addition, we will refine the proposed DP-based algorithm by using a discretization technique with respect to computational speed, and develop an on-line maintenance algorithm to control rejuvenation, restoration and checkpointing simultaneously. Furthermore we will explore possibility of an on-line control scheme based on Bayesian learning.

Acknowledgment

This research was partially supported by the Ministry of Education, Science, Sports and Culture, Grant-in-Aid for Scientific Research (C), Grant Nos. 18510138 (2006-2008), 19510148 (2007-2008) and Young Scientists (B), Grant No. 19700065 (2007-2008).

References

[1] E. Adams, "Optimizing preventive service of the software products," *IBM J. Research & Development*, vol. 28, pp. 2–14, 1984.

[2] V. Castelli, R. E. Harper, P. Heidelberger, S. W. Hunter, K. S. Trivedi, K. Vaidyanathan, and W. P. Zeggert, "Proactive management of software aging," *IBM J. Research & Development*, vol. 45, pp. 311–332, 2001.

[3] Y. Huang, C. Kintala, N. Kolettis, and N. D. Fulton, "Software rejuvenation: analysis, module and applications," in *Proc. 25th Int'l Sympo. Fault Tolerant Computing*, 1995, pp. 381–390.

[4] A. Avritzer and E. J. Weyuker, "Monitoring smoothly degrading systems for increased dependability," *Empirical Software Engineering*, vol. 2, pp. 59–77, 1997.

[5] S. Garg, A. van Moorsel, K. Vaidyanathan, and K. S. Trivedi, "A methodology for detection and estimation of software aging," in *Proc. 9th Int'l Sympo. Software Reliab. Eng.*, 1998, pp. 282–292.

[6] M. Shereshevsky, B. Cukic, J. Crowel, and V. Candikota, "Software aging and multifractality of memory resources," in *Proc. Int'l Conf. on Dependable Systems and Networks*, 2003, pp. 721–730.

[7] K. Vaidyanathan and K. S. Trivedi, "A measurement-based model for estimation of resource exhaustion in operational software system," in *Proc. 10th Int'l Sympo. Software Reliab. Eng.*, 1999, pp. 84–93.

[8] J. Gray and D. P. Siewiorek, "High-availability computer systems," *IEEE Computer*, vol. 24, pp. 39–48, 1991.

[9] A. T. Tai, L. Alkalai, and S. N. Chau, "On-board preventive maintenance: a design-oriented analytic study for long-life applications," *Performance Evaluation*, vol. 35, pp. 215–232, 1999.
[10] E. Marshall, "Fatal error: how Patriot overlooked a Scud," *Science*, vol. 3, p. 1347, 1992.
[11] T. Dohi, K. Goševa-Popstojanova, and K. S. Trivedi, "Estimating software rejuvenation schedule in high assurance systems," *The Computer Journal*, vol. 47, pp. 473–485, 2001.
[12] T. Dohi, H. Suzuki, and K. S. Trivedi, "Comparing software rejuvenation policies under different dependability measures," *IEICE Trans. on Information and Systems*, vol. E87-D, pp. 2078–2085, 2004.
[13] S. Garg, S. Pfening, A. Puliafito, M. Telek, and K. S. Trivedi, "Analysis of preventive maintenance in transactions based software systems," *IEEE Transactions on Computers*, vol. 47, pp. 96–107, 1998.
[14] H. Okamura, K. Iwamoto, and T. Dohi, "A dynamic programming algorithm for software rejuvenation scheduling under distributed computation circumstance," *Journal of Computer Science*, vol. 2, pp. 505–512, 2006.
[15] R. E. Barlow and F. Proschan, *Mathematical Theory of Reliability*. New York: John Wiley & Sons, 1965.
[16] R. Bellman, *Dynamic Programming*. Princeton University Press, 1957.
[17] M. Puterman, *Markov Decision Processes*. John Wiley & Sons, 1994.
[18] B. S. Gottfried, "A stopping criterion for the golden-ratio search," *Operations Research*, vol. 23, no. 3, pp. 553–555, 1975.
[19] H. Okamura and T. Dohi, "Analysis of a software system with rejuvenation, restoration and checkpointing," in *Service Availability: 5th Int'l Service Availability Sympo., Lecture Notes in Computer Science*, T. Nanya, F. Maruyama, A. Pataricza, and M. Malek, Eds. Springer-Verlag, 2008, vol. 5017, pp. 110–128.