Estimating Software Rejuvenation Schedules in High Assurance Systems

TADASHI DOHI[1], KATERINA GOŠEVA-POPSTOJANOVA[2] AND KISHOR TRIVEDI[3]

[1]*Department of Industrial and Systems Engineering, Hiroshima University, Higashi-Hiroshima 739-8527, Japan*
[2]*Lane Department of Computer Science and Electrical Engineering, West Virginia University, Morgantown, WV 26506-6109, USA*
[3]*Department of Electrical and Computer Engineering, Duke University, Durham, NC 27708-0294, USA*
Email: dohi@gal.sys.hiroshima-u.ac.jp
Tel. +81 824 24 7697
Fax. +81 824 22 7025

Software rejuvenation is a preventive maintenance technique that has been extensively studied in the recent literature. In this paper, we extend the classical result by Huang, Kintala, Kolettis and Fulton (1995), and in addition propose a modified stochastic model to generate the software rejuvenation schedule. More precisely, the software rejuvenation models are formulated via the semi-Markov reward process, and the optimal software rejuvenation schedules are derived analytically in terms of the reward rate. In particular, we consider the two special cases; steady-state availability and expected cost per unit time in the steady state. Further, we develop non-parametric algorithms to estimate the optimal software rejuvenation schedules, provided that the statistically complete (unsensored) sample data of failure time is given. In numerical examples, we compare two models from both view points of system availability and economic justification, and examine asymptotic properties for the statistical estimation algorithms.

1. INTRODUCTION

Demands on software reliability and availability have increased tremendously due to the nature of present day applications. They impose stringent requirements in terms of cumulative downtime and failure free operation of software, since in many cases, the consequences of software failure can lead to huge economic losses or risk to human life. The disastrous consequences of such failures and an ever increasing dependence on computer systems fosters an urgent need for high-assurance systems. To achieve high assurance, developers must use consistent, rigorous methods throughout the development process, from requirements specification and design to implementation and assessment.

Recently, the phenomenon of *software aging*, one in which the state of the software system degrades with time, has been reported [17]. Actually, when software application executes continuously for long periods of time it ages as a result of 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. In high-assurance systems, software aging can cause outages that result in high cost. Huang *et al.* [17] report this phenomenon in telecommunications billing applications where over time the application experiences a crash or a hang failure. Avritzer and Weyuker [2] discuss aging in telecommunication switching software where the effect manifests as gradual performance degradation. Software aging has also been observed in widely-used software like Netscape and xrn. Perhaps the most vivid example of aging in safety critical systems is the Patriot's software [18], where the accumulated errors led to a failure that resulted in loss of human life.

Resource leaking and other problems causing software to age are due to the software faults whose fixing is not always possible because, for example, application user may not have the access to the source code. Furthermore, it is almost impossible to fully test and verify if a piece of software is fault-free. Testing software becomes harder if it is complex, and furthermore testing cycle times are often reduced due to smaller release time requirements. Common experience suggests that most software failures are transient in nature [15]. Since transient failures will disappear if the operation is retried later in slightly different context, it is difficult to characterize their root origin. Therefore, the residual faults (sometimes called Heisenbugs [14]) have to be tolerated in the operational phase. Usual strategies to deal with failures in operational phase are reactive in nature; they consist of action taken after failure. A complementary approach to handle transient software failures, called *software rejuvenation*, was proposed recently [17]. Software rejuvenation is a preventive and proactive (as opposite to being reactive) 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, *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, rejuvenation has been used in high assurance systems as well. For example, it has been used to avoid unplanned outages in continuously available systems, such as telecommunication systems [2, 17], where the cost of downtime is extremely high. Among typical applications of high-consequence systems, periodic software and system rejuvenation have been implemented for long-life deep-space missions [23]. Although the fault in the software still remains, performing rejuvenation periodically removes or minimizes potential error conditions due to that fault, thus preventing failures that might have unacceptable consequences.

Rejuvenation has the same motivation and advantages/disadvantages as preventive maintenance policies in hardware systems. Any rejuvenation typically involves an overhead, but, on the other hand, prevents more severe failures to occur. The application will of course be unavailable during rejuvenation, but since this is a scheduled downtime the cost is expected to be much lower than the cost of an unscheduled downtime caused by failure. Hence, an important issue is to determine the optimal schedule to perform software rejuvenation in terms of availability and cost.

In this paper, we extend the classical result by Huang *et al.* [17]. More precisely, the software rejuvenation models are formulated via the semi-Markov reward processes, and the optimal software rejuvenation schedules are derived analytically in terms of reward rate [13, 16, 19, 22]. As the important special cases, we consider both the availability model and the cost model. In the former, the optimal software rejuvenation schedule which maximizes the steady-state availability is derived. We also derive the cost optimal rejuvenation schedule which minimizes the expected cost per unit time in the steady state. Further, since the failure time distribution can not be easily estimated from a few data samples, we develop non-parametric statistical algorithms to estimate the optimal software rejuvenation schedules, provided that the statistically complete (unsensored) sample data of failure times is given. These can be useful in determining optimal rejuvenation schedule in the early segment of the operational phase. In numerical examples, we examine asymptotic properties of the statistical estimation algorithms.

2. RELATED WORK

In recent years, considerable attention has been devoted to the phenomenon of software aging. For an extensive survey, the reader is referred to [24]. The studies of aging-related failures are based on two approaches: measurement-based and model-based. The measurement-based approach deals with the detection and validation of the existence of software aging and estimating its effects on system resources [12, 25]. The model-based approach is aimed at evaluating the effectiveness of software rejuvenation and determining the optimal schedules to perform rejuvenation. Next, we present the brief overview of the previous work related to model-based approaches.

A great deal of research effort on modeling software aging and rejuvenation considers continuously running software systems. Huang *et al.* [17] used a continuous time Markov chain to model software rejuvenation. They considered the two-step failure model where the application goes from the initial robust (clean) state to a failure probable (degraded) state from which two actions are possible: rejuvenation or transition to failure state. Both rejuvenation and recovery from failure return the software system to the robust state. Garg *et al.* [11] introduced into the model the idea of periodic rejuvenation. To deal with deterministic interval between successive rejuvenations the system behaviour was represented through a Markov regenerative stochastic Petri net model. The subsequent work [8] involves arrival and queueing of jobs in the system and computes load and time dependent rejuvenation policy. The above models consider the effect of aging as crash/hang failure, referred to as hard failures, which result in unavailability of the software. However, due to the aging the software system can exhibit soft failures, that is, performance degradation. In [21] the performance degradation is modeled by the gradual decrease of the service rate. Both effects of aging, hard failures that result in an unavailability and soft failures that result in performance degradation, are considered in the model of transaction based software system presented in [10]. This model was recently generalized in [20] by considering multiple servers.

The fine grained software rejuvenation model presented in [4] takes a different approach to characterize the effect of software aging. It assumes that the degradation process consists of a sequence of additive random shocks; the system is considered to be out of service as soon as the appropriate parameter reaches an assigned threshold level.

Several studies considered software with a finite mission time. The work in [9] analyzes the effects of checkpointing and rejuvenation used together on the expected completion time of a software program. The use of preventive on-board maintenance that includes periodic software and system rejuvenation is proposed and analyzed in [23].

This paper results from an elaboration and extension of our earlier work [5, 6]. Of particular interest is how to determine the optimal schedule to perform software rejuvenation in terms of different criteria. The rejuvenation models considered in this paper have similar but somewhat generalized mathematical structure to that in Huang *et al.* [17]. However, the approaches taken to estimate the optimal software rejuvenation schedules are quite different. Note that in the above literature, the failure time distribution needs to be specified to derive the optimal rejuvenation schedule. This seems to be restrictive, since the determination of the theoretical distribution from the real data is rather troublesome, and needs both the goodness-of-fit test and the parameter estimation based on several candidate distribution functions. Although in the existing model-based approaches, the failure time distribution is fixed (e.g., the Weibull distribution or hypoexponential distribution), these have not yet been validated for software aging. By contrast, our approach does not depend on the kind of distribution function and can provide non-parametric estimators of the optimal software rejuvenation schedules. Thus we provide a very powerful approach for the application of the rejuvenation to a real system operation.

3. MODEL DESCRIPTION

First, we introduce the basic software rejuvenation model proposed by Huang *et. al* [17]. Although they formulated it as a simple continuous-time Markov chain, we extend their result in the more general mathematical framework. In particular, we regard the software rejuvenation models as continuous-time semi-Markov reward processes. Define the following four states:

State 0: highly robust state (normal operation state)

State 1: failure probable state

State 2: failure state

State 3: software rejuvenation state.

Suppose that all the states mentioned above are regeneration points. More specifically, let Z be the random time interval when the highly robust state changes to the failure probable state, having the common distribution function $\Pr\{Z \leq t\} = F_0(t)$ with finite mean μ_0 (> 0). Just after the state becomes the failure probable state, a system failure may occur with a positive probability. Without loss of generality, we assume that the random variable Z is observable during the system operation [11, 17]. The transition diagram for the semi-Markov model is depicted in Fig. 1.

Define the failure time X (from State 1) and the repair time Y, having the distribution functions $\Pr\{X \leq t\} = F_f(t)$ and $\Pr\{Y \leq t\} = F_a(t)$ with finite means λ_f (> 0) and μ_a (> 0), respectively. If the system failure occurs before triggering a software rejuvenation, then the repair is started immediately at that time and is completed after the random time Y elapses. Otherwise, the software rejuvenation is started. Note that the software rejuvenation cycle is measured from the time instant just after the system enters State 1 from State 0. Denote the distribution functions of the time to software rejuvenation trigger and its system overhead by $F_r(t)$ and $F_c(t)$ (with mean μ_c (> 0)), respectively. After completing the repair or the rejuvenation, the software system becomes as good as new, and the software age is initiated at the beginning of the next highly robust state. Consequently, we define the time interval from the beginning of the system operation to the next one as one cycle, and the same cycle is repeated again and again. If we consider the time to software rejuvenation trigger as a constant t_0, then it follows that

$$F_r(t) = U(t - t_0) = \begin{cases} 1 & \text{if } t \geq t_0 \\ 0 & \text{otherwise.} \end{cases} \tag{1}$$

We call t_0 (≥ 0) as *the software rejuvenation schedule* in this paper and $U(\cdot)$ is the unit step function. Underlying stochastic process is a semi-Markov process with four regeneration states. If we assume that the sojourn times in all states are exponentially distributed, the semi-Markov model under consideration is reduced to the one in Huang *et al.* [17].

4. SEMI-MARKOVIAN REWARD ANALYSIS

For the semi-Markov process described in Section 3, define the transition probability $Q_{ij}(t)$ $(i, j = 0, \cdots, 3, i \neq j)$. Also, we define the Laplace Stieltjes transform (LST) of the transition probability by $q_{ij}(s) = \int_0^\infty \exp\{-st\} dQ_{ij}(t)$. It is straightforward to obtain

$$q_{01}(s) = \int_0^\infty \exp\{-st\} dF_0(t), \tag{2}$$

$$q_{12}(s) = \int_0^\infty \exp\{-st\} \overline{F}_r(t) dF_f(t), \tag{3}$$

$$q_{13}(s) = \int_0^\infty \exp\{-st\} \overline{F}_f(t) dF_r(t), \tag{4}$$

$$q_{20}(s) = \int_0^\infty \exp\{-st\} dF_a(t), \tag{5}$$

$$q_{30}(s) = \int_0^\infty \exp\{-st\} dF_c(t), \tag{6}$$

where in general $\overline{\psi}(\cdot) = 1 - \psi(\cdot)$. Define the recurrence time distribution from State 0 to State 0 again by $H_{00}(t)$. Then the LST of the recurrence time distribution is

$$h_{00}(s) = \int_0^\infty \exp\{-st\} dH_{00}(t)$$

$$= q_{01}(s)q_{12}(s)q_{20}(s) + q_{01}(s)q_{13}(s)q_{30}(s). \quad (7)$$

Of our concern is the derivation of the transient probability to stay in State j ($j = 0, \cdots, 3$) at arbitrary time t (> 0), provided that the initial state at time $t = 0$ is State 0. Define the transient probability from State 0 to j ($j = 0, \cdots, 3$) and its LST by $P_{0j}(t)$ and $p_{0j}(s) = \int_0^\infty \exp\{-st\} dP_{0j}(t)$, respectively. After some manipulations, we have

$$p_{00}(s) = \overline{q}_{01}(s)/\overline{h}_{00}(s), \quad (8)$$

$$p_{01}(s) = q_{01}(s)\Big(\overline{q}_{12}(s) - q_{13}(s)\Big)/\overline{h}_{00}(s), \quad (9)$$

$$p_{02}(s) = q_{01}(s)q_{12}(s)\overline{q}_{20}(s)/\overline{h}_{00}(s), \quad (10)$$

$$p_{03}(s) = q_{01}(s)q_{13}(s)\overline{q}_{30}(s)/\overline{h}_{00}(s). \quad (11)$$

Next, we assign reward rate $\xi_j \in (-\infty, \infty)$ to each state j ($= 0, 1, 2, 3$) of the semi-Markov process. Following [19], we define the expected instantaneous reward rate:

$$PP(t) = \sum_{j=0}^{3} \xi_j P_{0j}(t), \quad (12)$$

Also, the expected steady-state reward rate is

$$\begin{aligned} PP &= \lim_{t\to\infty} PP(t) \\ &= \lim_{t\to\infty} \sum_{j=0}^{3} \xi_j P_{0j}(t) \\ &= \lim_{s\to 0} \sum_{j=0}^{3} \xi_j p_{0j}(s). \quad (13) \end{aligned}$$

From the familiar probabilistic argument, we can derive the expected steady-state reward rate as the function of t_0 for Model 1 as follows.

$$PP(t_0) = \frac{\xi_0\mu_0 + \xi_1 \int_0^{t_0} \overline{F}_f(t)dt + \xi_2\mu_a F_f(t_0) + \xi_3\mu_c\overline{F}_f(t_0)}{\mu_0 + \mu_a F_f(t_0) + \mu_c \overline{F}_f(t_0) + \int_0^{t_0} \overline{F}_f(t)dt} = K(t_0)/T(t_0). \quad (14)$$

To derive the optimal software rejuvenation schedule which maximizes the expected steady-state reward rate in Eq.(14), we examine the properties for the functions $PP(t_0)$. Suppose that the failure rate $r_f(t) = (dF_f(t)/dt)/\overline{F}_f(t_0)$ exists and is differentiable. Differentiating $PP(t_0)$ with respect to t_0 and setting equal to zero implies the non-linear equation $q_P(t_0) = 0$, where

$$\begin{aligned} q_P(t_0) &= \Big\{\xi_1 + (\xi_2\mu_a - \xi_3\mu_c) r_f(t_0)\Big\} T(t_0) \\ &\quad - \Big\{(\mu_a - \mu_c) r_f(t_0) + 1\Big\} K(t_0). \quad (15) \end{aligned}$$

Hence, if the first derivative of the function $q_P(t_0)$ is negative, then the expected steady-state reward rate $PP(t_0)$ is quasi-concave in t_0 [1]. The following result is useful to characterize the optimal software rejuvenation schedule maximizing the expected steady-state reward rate.

Theorem 4.1: Suppose that the lower limit of the expected steady-state reward rate $\inf_{0\le t_0<\infty} PP(t_0)$ for all t_0 is less than $(\xi_2\mu_a - \xi_3\mu_c)/(\mu_a - \mu_c)$. If the failure time distribution is strictly IFR (increasing failure rate), then $PP(t_0)$ is strictly quasi-concave, otherwise, quasi-convex in t_0.

In order to characterize the unique optimal software rejuvenation schedule, we have to examine the relationship between the lower limits $\inf_{0\le t_0<\infty} PP(t_0)$ and the parameter value $(\xi_2\mu_a - \xi_3\mu_c)/(\mu_a - \mu_c)$. Although it is straightforward to maximize the expected steady-state reward rate directly, giving the physical meaning to the necessary and sufficient conditions for the existence of the finite and non-zero software rejuvenation schedule is not so easy, because the expected steady-state reward rate is a rather general measure, but is an abstract concept. In other words, when the rejuvenation schedule is designed for the software system, the resulting scheduling policy must be checked whether it can be fitted to the actual operational circumstance. Then the problem is how to determine the reward rate or the weight ξ_j ($j = 0, 1, 2, 3$).

In the following sections, we discuss the two special cases of the expected steady-state reward rate: steady-state availability and the expected cost per unit time in the steady state. These measures belong to the subclass of the expected reward rate, but can give the physical or economical meanings to the resulting optimal schedules.

5. SPECIAL CASE: AVAILABILITY MODEL

If we assign a reward rate 1 to operational states and zero to non-operational states the expected steady-state reward rate becomes the steady-state availability. In particular, putting $\xi_0 = \xi_1 = 1$ and $\xi_2 = \xi_3 = 0$ in Eq.(14), the steady-state availability becomes

$$\begin{aligned} A(t_0) &= \Pr\{\text{software system is operative in the steady state}\} \\ &= \lim_{t\to\infty}\Big\{P_{00}(t) + P_{01}(t)\Big\} \\ &= \lim_{s\to 0}\Big\{p_{00}(s) + p_{01}(s)\Big\} \\ &= \frac{\mu_0 + \int_0^{t_0} \overline{F}_f(t)dt}{\mu_0 + \mu_a F_f(t_0) + \mu_c\overline{F}_f(t_0) + \int_0^{t_0} \overline{F}_f(t)dt} \\ &= S(t_0)/T(t_0). \quad (16) \end{aligned}$$

The problem is to derive the optimal software rejuvenation schedule t_0^* which maximizes the steady-state availability $A(t_0)$.

We make the following assumption:

(A-1) $\mu_a > \mu_c$.

The assumption (A-1) states that the mean time to repair is strictly larger than the mean time to complete the software rejuvenation. This assumption is quite reasonable and intuitive. The following result gives the optimal software rejuvenation schedule.

Theorem 5.1: (1) Suppose that the failure time distribution is strictly IFR and that the assumption (A-1) holds. Define the following non-linear function:

$$q_A(t_0) = T(t_0) - \Big\{(\mu_a - \mu_c) r_f(t_0) + 1\Big\} S(t_0). \tag{17}$$

(i) If $q_A(0) > 0$ and $q_A(\infty) < 0$, then there exists a finite and unique optimal software rejuvenation schedule t_0^* $(0 < t_0^* < \infty)$ satisfying $q_A(t_0^*) = 0$, and the maximum steady-state availability is

$$A(t_0^*) = \frac{1}{(\mu_a - \mu_c) r_f(t_0^*) + 1}. \tag{18}$$

(ii) If $q_A(0) \le 0$, then the optimal software rejuvenation schedule is $t_0^* = 0$, *i.e.* it is optimal to start the rejuvenation immediately after entering the failure probable state, and the maximum steady-state availability is $A(0) = \mu_0/(\mu_0 + \mu_c)$.

(iii) If $q_A(\infty) \ge 0$, then the optimal rejuvenation schedule is $t_0^* \to \infty$, *i.e.* it is optimal not to carry out any rejuvenation, and the maximum steady-state availability is $A(\infty) = (\mu_0 + \lambda_f)/(\mu_0 + \mu_a + \lambda_f)$.

(2) Suppose that the failure time distribution is DFR (decreasing failure rate) and that the assumption (A-1) holds. Then, the system availability $A(t_0)$ is a quasi-convex function of t_0, and the optimal rejuvenation schedule is $t_0^* = 0$ or $t_0^* \to \infty$.

Proof: Differentiating $A(t_0)$ with respect to t_0 and setting it equal to 0 implies $q_A(t_0) = 0$. Further differentiation yields

$$\frac{dq_A(t_0)}{dt_0} = -(\mu_a - \mu_c) S(t_0) \frac{dr_f(t_0)}{dt_0}. \tag{19}$$

If $r_f(t_0)$ is strictly increasing, then the function $q_A(t_0)$ is strictly decreasing and the steady-state availability $A(t_0)$ is strictly quasi-concave in t_0 under the assumption (A-1). Further, if $q_A(0) > 0$ and $q_A(\infty) < 0$, then there exists a unique optimal software rejuvenation schedule t_0^* $(0 < t_0^* < \infty)$ satisfying $q_A(t_0^*) = 0$. If $q_A(0) \le 0$ or $q_A(\infty) \ge 0$, then the steady-state availability is monotonically increasing or decreasing in t_0, and the optimal policy becomes $t_0^* = 0$ or $t_0^* \to \infty$. On the other hand, if $r_f(t_0)$ is a decreasing function of t_0, then $A(t_0)$ is a quasi-convex function of t_0, and the optimal software rejuvenation schedule is $t_0^* = 0$ or $t_0^* \to \infty$. The proof is thus completed. ■

It is easy to check that the above result implies the result in Huang *et. al* [17], although they used the system downtime and its associated cost as criteria of optimality. As is clear from Theorem 5.1, when the failure time obeys the exponential distribution, the optimal software rejuvenation schedule becomes $t_0^* = 0$ or $t_0^* \to \infty$. It means that the rejuvenation should be performed as soon as the software enters the failure probable state ($t_0 = 0$) or should not be performed at all ($t_0 \to \infty$). Therefore, the determination of the optimal rejuvenation schedule based on the the steady-state availability is never motivated in such a situation. Since for a software system which ages it is more realistic to assume that failure time distribution is strictly IFR, our general setting is plausible and the result satisfies our intuition.

In this section, we derived the optimal software rejuvenation schedule that maximizes the steady-state availability. Note that the steady-state availability is the simplest case with $\xi_0 = \xi_1 = 1$ and $\xi_2 = \xi_3 = 0$ in the expected steady-state reward rate. In the following section, we consider the cost model. Actually, it is well known that the unplanned downtime for the running software system is more expensive than the planned downtime with preventive maintenance. In other words, the cost model is the more plausible reward model from the economic justification.

6. SPECIAL CASE: COST MODEL

Define the following cost components:

c_s (> 0)**:** repair cost per unit time

c_p (> 0)**:** rejuvenation cost per unit time.

Further, we make the following assumption:

(A-2) $c_s > c_p$.

Assumption (A-2) says that the cost of repair is larger than the cost of software rejuvenation. From the practical point of view, this assumption is also quite reasonable.

For the cost model we assign reward rates in terms of respective costs. Thus, putting $\xi_0 = \xi_1 = 0$, $\xi_2 = c_s$ and $\xi_3 = c_p$ in Eq.(14), the expected cost per unit time in the steady state becomes

$$\begin{aligned} C(t_0) &= \lim_{t\to\infty} \frac{\text{E[total cost during } (0,t)]}{t} \\ &= \lim_{t\to\infty} \Big\{ c_s P_{02}(s) + c_p P_{03}(s) \Big\} \\ &= \lim_{s\to 0} \Big\{ c_s p_{02}(s) + c_p p_{03}(s) \Big\} \\ &= \frac{c_s \mu_a F_f(t_0) + c_p \mu_c \overline{F}_f(t_0)}{\mu_0 + \mu_a F_f(t_0) + \mu_c \overline{F}_f(t_0) + \int_0^{t_0} \overline{F}_f(t) dt} \\ &= V(t_0)/T(t_0). \end{aligned} \tag{20}$$

The problem is to derive the optimal software rejuvenation schedule t_0^* which minimizes the expected cost per unit time in the steady state $C(t_0)$. The following result gives the optimal software rejuvenation schedule for the cost model.

Theorem 6.1: (1) Suppose that the failure time distribution is strictly IFR and that the assumptions (A-1) and (A-2) hold. Define the following non-linear function:

$$\begin{aligned} q_C(t_0) &= (c_s\mu_a - c_p\mu_c) r_f(t_0) S(t_0) \\ &\quad -\Big\{(\mu_a - \mu_c) r_f(t_0) + 1\Big\} V(t_0). \qquad (21) \end{aligned}$$

(i) If $q_C(0) < 0$ and $q_C(\infty) > 0$, then there exists a finite and unique optimal software rejuvenation schedule t_0^* $(0 < t_0^* < \infty)$ satisfying $q_C(t_0^*) = 0$, and the minimum expected cost is

$$C(t_0^*) = \frac{(c_s\mu_a - c_p\mu_c) r_f(t_0^*)}{(\mu_a - \mu_c) r_f(t_0^*) + 1}. \qquad (22)$$

(ii) If $q_C(0) \geq 0$, then the optimal software rejuvenation schedule is $t_0^* = 0$ and the minimum expected cost is $C(0) = c_p\mu_c/(\mu_0 + \mu_c)$.

(iii) If $q_C(\infty) \leq 0$, then the optimal software rejuvenation schedule is $t_0^* \to \infty$ and the minimum expected cost is $C(\infty) = (c_s\mu_a)/(\mu_0 + \mu_a + \lambda_f)$.

(2) Suppose that the failure time distribution is DFR and that the assumptions (A-1) and (A-2) hold. Then, the expected cost function $C(t_0)$ is a quasi-concave function of t_0, and the optimal software rejuvenation schedule is $t_0^* = 0$ or $t_0^* \to \infty$.

Proof: Differentiating $C(t_0)$ with respect to t_0 and setting it equal to 0 implies $q_C(t_0) = 0$. Further differentiation yields

$$\begin{aligned} \frac{dq_C(t_0)}{dt_0} &= \frac{dr_f(t_0)}{dt_0}\Big[(c_s\mu_a - c_p\mu_c) S(t_0) \\ &\quad -(\mu_a - \mu_c) V(t_0)\Big]. \qquad (23) \end{aligned}$$

If the term in the bracket on the right hand side in Eq.(23) is negative, then $c_s + \mu_c(c_s - c_p)/(\mu_a - \mu_c) \leq C(t_0)$ for all $t_0 \in [0, \infty)$. Since $\mu_c(c_s - c_p)/(\mu_a - \mu_c) > 0$ from the assumptions (A-1) and (A-2), this contradicts the probability law even if the repair occurs in the steady state with probability one. Hence, it follows from the reduction argument that $c_s + \mu_c(c_s - c_p)/(\mu_a - \mu_c) > C(t_0)$.

If $r_f(t_0)$ is strictly increasing, then the function $q_C(t_0)$ is strictly increasing and the expected cost $C(t_0)$ is strictly quasi-convex in t_0. Further, if $q_C(0) < 0$ and $q_C(\infty) > 0$, then there exists a unique optimal software rejuvenation schedule t_0^* $(0 < t_0^* < \infty)$ satisfying $q_C(t_0^*) = 0$. If $q_C(0) \geq 0$ or $q_C(\infty) \leq 0$, then the expected cost is monotonically decreasing or increasing in t_0, and the optimal policy becomes $t_0^* = 0$ or $t_0^* \to \infty$. On the other hand, if $r_f(t_0)$ is a decreasing function of t_0, then $C(t_0)$ is a quasi-concave function of t_0, and the optimal software rejuvenation schedule is $t_0^* = 0$ or $t_0^* \to \infty$. The proof is thus completed. ■

In Sections 5 and 6, we derived the optimal software rejuvenation schedules which maximizes the steady-state availability or minimizes the expected cost per unit time in the steady state. It should be noted, however, that the optimal software rejuvenation schedule depends on the model parameters: μ_0, μ_c, μ_a, and the failure time distribution $F_f(t)$, as well as the reward rates c_s and c_p. Among these parameters the failure time distribution in the software operational phase is the most difficult to obtain. In the following section, we develop statistical non-parametric algorithms to estimate the optimal software rejuvenation schedules, provided that the statistical complete (unsensored) sample data of failure times is given.

7. STATISTICAL OPTIMIZATION ALGORITHMS

Before developing the statistical estimation algorithms for the optimal software rejuvenation schedules, we translate the underlying problems to graphical ones. Following Barlow and Campo [3], define the scaled total time on test (TTT) transform of the failure time distribution:

$$\phi(p) = (1/\lambda_f)\int_0^{F_f^{-1}(p)} \overline{F}_f(t)dt, \qquad (24)$$

where

$$F_f^{-1}(p) = \inf\{t_0; F_f(t_0) \geq p\}, \quad (0 \leq p \leq 1). \qquad (25)$$

It is well known [3] that $F_f(t)$ is IFR (DFR) if and only if $\phi(p)$ is concave (convex) on $p \in [0, 1]$. After a few algebraic manipulations, we have the following results.

Theorem 7.1: Suppose that the assumption (A-1) holds. Obtaining the optimal software rejuvenation schedule $t_0{}^*$ maximizing the steady-state availability $A(t_0)$ is equivalent to obtaining p^* $(0 \leq p^* \leq 1)$ such as

$$\max_{0 \leq p \leq 1} \frac{\phi(p) + \alpha_a}{p + \beta_a}, \qquad (26)$$

where

$$\alpha_a = \lambda_f + \mu_0, \qquad (27)$$

$$\beta_a = \mu_c(\mu_a - \mu_c). \qquad (28)$$

From Theorem 7.1 it can be seen that the optimal software rejuvenation schedule $t_0{}^* = F_f^{-1}(p^*)$ is determined by calculating the optimal point $p^*(0 \leq p^* \leq 1)$ maximizing the tangent slope from the point $(-\beta_a, -\alpha_a) \in (-\infty, 0) \times (-\infty, 0)$ to the curve $(p, \phi(p)) \in$

$[0,1]\times[0,1]$. Figure 2 illustrates the determination of the optimal software rejuvenation schedule on the two-dimensional graph, when the underlying failure time distribution is Weibull:

$$F_f(t) = 1 - e^{-(\frac{t}{\theta})^\beta} \tag{29}$$

with shape parameter $\beta = 2.34$ and scale parameter $\theta = 0.903$. In Fig. 2, since $p^* = 0.5240$ has the maximum slope, the optimal software rejuvenation schedule is $t_0^* = F_f^{-1}(0.5240) = 0.7950$. If the failure time distribution is the exponential, then it is seen that the optimal policy is $p^* = 0$ ($t_0^* = 0$) or $p^* = 1$ ($t_0^* \to \infty$).

Similar to the availability model, we consider the cost model discussed in Section 6.

Theorem 7.2: Suppose that both the assumptions (A-1) and (A-2) hold. Obtaining the optimal software rejuvenation schedule $t_0{}^*$ minimizing the expected cost per unit time in the steady state $C(t_0)$ is equivalent to obtaining p^* $(0 \le p^* \le 1)$ such as

$$\max_{0\le p\le 1} \frac{\phi(p)+\alpha_c}{p+\beta_c}, \tag{30}$$

where

$$\alpha_c = \frac{(c_s - c_p)\mu_c(\mu_0+\mu_a)}{\lambda_f(c_s\mu_a - c_p\mu_c)}, \tag{31}$$

$$\beta_c = \frac{c_p\mu_c}{c_s\mu_a - c_p\mu_c}. \tag{32}$$

Theorem 7.2 is the dual of Theorem 6.1. From this result, it is seen that the optimal software rejuvenation schedule $t_0{}^* = F_f^{-1}(p^*)$ is determined by calculating the optimal point $p^*(0 \le p^* \le 1)$ maximizing the tangent slope from the point $(-\beta_c, -\alpha_c) \in (-\infty,0)\times(-\infty,0)$ to the curve $(p,\phi(p)) \in [0,1]\times[0,1]$. Figure 3 depicts the determination of the optimal software rejuvenation schedule on the two-dimensional graph, when the underlying failure time distribution is Weibull as in Eq.(29). In Fig. 3, since $p^* = 0.1050$ has the maximum slope, the optimal software rejuvenation schedule is $t_0^* = F_f^{-1}(0.1050) = 0.1310$.

Next, suppose that the optimal software rejuvenation schedule is to be estimated from an ordered complete observation $0 = x_0 \le x_1 \le x_2 \le \cdots \le x_n$ of the failure times from an absolutely continuous distribution F_f, which is unknown. Then the scaled TTT statistics based on this sample are defined by $\phi_{nj} = \psi_j/\psi_n$, where

$$\psi_j = \sum_{k=1}^{j}(n-k+1)(x_k - x_{k-1}), \quad (j = 1,2,\cdots,n;\ \psi_0 = 0). \tag{33}$$

Since the empirical distribution function $F_n(x)$ corresponding to the sample data x_j $(j = 0,1,2,\cdots,n)$ is

$$F_n(x) = \begin{cases} j/n & \text{for} \quad x_j \le x < x_{j+1}, \\ 1 & \text{for} \quad x_n \le x \end{cases} \tag{34}$$

the resulting polygon by plotting the points $(F_n(x), \phi_{nj})$ $(j = 0,1,2,\cdots,n)$ and connecting them by line segments is called the *scaled TTT plot* [3, 7]. In other words, the scaled TTT plot can be regarded as a numerical counterpart of the scaled TTT transform.

The following results give non-parametric statistical estimation algorithms for the optimal software rejuvenation schedules.

Theorem 7.3: (i) Suppose that the optimal software rejuvenation schedule is to be estimated from n ordered complete sample $0 = x_0 \le x_1 \le x_2 \le \cdots \le x_n$ of the failure times from an absolutely continuous distribution F_f, which is unknown. Then, a non-parametric estimator of the optimal software rejuvenation schedule $\hat{t}_0^*$ which maximizes $A(t_0)$ is given by x_{j^*}, where

$$j^* = \left\{ j \mid \max_{0\le j\le n} \frac{\phi_{nj}+\alpha_a}{j/n+\beta_a} \right\} \tag{35}$$

and λ_f in Eq. (27) is replaced by $\sum_{k=1}^{n} x_k/n$.
(ii) The estimator given in (i) is strongly consistent, *i.e.* x_{j^*} converges to the optimal solution $t_0{}^*$ uniformly with probability one as $n \to \infty$, if a unique optimal software rejuvenation schedule exists.

Theorem 7.4: (i) Suppose that the optimal software rejuvenation schedule is to be estimated from n ordered complete sample $0 = x_0 \le x_1 \le x_2 \le \cdots \le x_n$ of the failure times from an absolutely continuous distribution F_f, which is unknown. Then, a non-parametric estimator of the optimal software rejuvenation schedule $\hat{t}_0^*$ which minimizes $C(t_0)$ is given by x_{j^*}, where

$$j^* = \left\{ j \mid \max_{0\le j\le n} \frac{\phi_{nj}+\alpha_c}{j/n+\beta_c} \right\} \tag{36}$$

and λ_f in Eq. (31) is replaced by $\sum_{k=1}^{n} x_k/n$.
(ii) The estimator given in (i) is strongly consistent if a unique optimal software rejuvenation schedule exists.

It is straightforward to prove the above results (i) in Theorem 7.3 and Theorem 7.4 from Theorem 7.1 and Theorem 7.2, respectively. The uniform convergence property in (ii) follows from the Glivenko-Cantelli lemma [3] and the strong law of large numbers. The graphical procedure proposed here has an educational value for better understanding of the optimization problem and it is convenient for performing sensitivity analysis of the optimal software rejuvenation policy when different values are assigned to the model parameters. Of special interest is to estimate the optimal schedule without specifying the failure time distribution. Although some typical theoretical distribution functions such as the Weibull distribution and the gamma distribution are assumed in the reliability/performability analysis, our non-parametric estimation algorithm can generate the optimal software rejuvenation schedule using the on-line knowledge about the observed failure times.

Figure 4 shows the estimation results of the optimal software rejuvenation schedule under the availability criterion, where the failure time data is synthetically generated by the Weibull distribution with shape parameter $\beta = 4.0$ and scale parameter $\theta = 0.9$. For 100 failure data points, the estimates of the optimal rejuvenation schedule and the maximum system availability are $\hat{t}_0^* = 0.56592$ and $A(\hat{t}_0^*) = 0.987813$, respectively. In Fig. 5, we show the estimation results of the cost model, where the failure time data is generated by the same Weibull distribution as Fig. 4. In Fig. 5, the estimates of the optimal rejuvenation schedule and the minimum expected cost are $\hat{t}_0^* = 0.401221$ and $C(\hat{t}_0^*) = 0.000131508$, respectively. We would like to emphasize that the optimal rejuvenation schedule that minimizes the expected cost per unit time in the steady state depends on the particular values of the repair cost c_s and rejuvenation cost c_p. However, in general, it will be different from the optimal software rejuvenation schedule that maximizes the steady-state availability. So, the optimal rejuvenation schedule should be chosen accordingly to the most appropriate criterion for a given application.

In this section, we translated the underlying algebraic problem to a geometrical one to seek the maximum tangent slope. This graphical idea was used to develop a statistical estimation algorithm to find the optimal rejuvenation schedule. In the following section, we carry out the sensitivity analysis of model parameters and examine asymptotic properties for the statistical estimation algorithms using the simulation data.

8. NUMERICAL EXAMPLES

8.1. Sensitivity analysis

We carry out the sensitivity analysis on the model parameters. Figure 6 shows the behaviour of the steady-state availability, where the failure time distribution is given by Eq.(29). Define the MTTF (Mean Time to Failure) by $\lambda_f = \theta\Gamma(1 + 1/\beta)$, where $\Gamma(\cdot)$ denotes the standard gamma function. As the MTTF becomes larger for a fixed shape parameter, the optimal software rejuvenation schedule which maximizes the steady-state availability takes a larger value for each case. Also, dependences of MTTF on the optimal software rejuvenation time and its associated availability are investigated in Figs. 7 and 8, respectively. As the MTTF gets larger, both the rejuvenation schedule and its associated steady-state availability become monotonically larger. Next, consider the expected cost criterion. Figure 9 illustrates the behaviour of the expected cost, where the failure time distribution is also Weibull as in Eq.(29). As the MTTF becomes larger for a fixed shape parameter, the optimal software rejuvenation schedule which minimizes the expected cost takes larger value for each case. Dependences of the optimal software rejuvenation time and its associated cost value on the MTTF are also investigated in Figs. 10 and 11, respectively. As the MTTF gets larger, the rejuvenation schedule becomes monotonically larger, but the minimum expected cost becomes smaller.

Table 1 presents the dependence of the rejuvenation schedule on the ratio μ_a/μ_c in the availability model. As the ratio μ_a/μ_c increases, the rejuvenation schedule monotonically decreases, and the maximum availability also decreases. Thus, higher the mean repair time relative to the mean time to perform rejuvenation more frequently the rejuvenation should be performed. This monotone tendency can also be observed for other parameters. In Table 2, we examine the dependence of the rejuvenation schedule on the ratio λ_f/μ_0. If the MTTF becomes larger for a fixed value of μ_0, *i.e.*, the system tends to be more reliable, the resulting optimal rejuvenation schedule can take rather large value, that is, the rejuvenation should be performed less and less frequently.

Table 3 presents the dependence of the rejuvenation schedule on the cost ratio c_s/c_p in the cost model. As the cost ratio c_s/c_p increases, the rejuvenation time monotonically decreases, but the cost value increases. In Table 4, we examine the dependence of the rejuvenation schedule on the ratio μ_a/μ_c. If the mean repair time becomes larger for fixed time to complete rejuvenation, the resulting optimal rejuvenation schedule takes smaller values. The final example in Table 5 presents the relationship between λ_f/μ_0 and the optimal rejuvenation schedule. As λ_f takes larger value with respect to μ_0, *i.e.*, the system tends to be more reliable, t_0^* becomes larger.

8.2. Asymptotic behaviour

Next, we examine the asymptotic properties of the estimators developed in Section 7. One of the most important problem in practical applications is the speed of convergence of the estimates for the optimal software rejuvenation schedules. In other words, since large number of sample failure time data points are not available in the early part of the operational phase, it is important to investigate the number of data points at which one can estimate the optimal software rejuvenation schedule accurately without specifying the failure time distribution. Figures 12 and 13 illustrate the asymptotic behaviour of the estimates for the optimal software rejuvenation schedule and its associated maximum steady-state availability. The failure time data are generated by the Weibull distribution with shape parameter $\beta = 2.34$ and scale parameter $\theta = 0.903$. In the figures, the horizontal lines denote the real optimal rejuvenation schedule and the maximum system availability, respectively.

In Fig. 13, the maximum availability $A(\hat{t}_0^*)$ is calculated in accordance with the estimation algorithm in

Theorem 7.3, where the sample mean $\hat{\lambda}_f = \sum_{k=1}^{n} x_k/n$ changes as the failure time data is observed. From Figs. 12 and 13, it is seen that the estimate of the optimal rejuvenation schedule fluctuates until the number of observations is about 40. These results enable us to use the non-parametric algorithm proposed here to estimate precisely the optimal software rejuvenation schedule under the incomplete knowledge of the failure time distribution.

Next, we examine the asymptotic properties of the estimators developed in the cost models. Figures 14 and 15 illustrate the asymptotic behaviour of the estimates for the optimal software rejuvenation schedule and its associated minimum expected cost, respectively, where the failure time data are generated from the Weibull distribution with shape parameter $\beta = 4.0$ and scale parameter $\theta = 0.9$. In these figures, estimates of the optimal policy $\hat{t}_0^*$ and the corresponding minimum cost value $C(\hat{t}_0^*)$ are calculated in accordance with the estimation algorithm in Theorem 7.4. From Fig. 14 it is seen that the estimate of the optimal rejuvenation schedule fluctuates until the number of observations is about 50. On the other hand, it is found that the expected cost per unit time in the steady state can be estimated accurately after the number of observations becomes about 20. These results will be useful to design the high assurance software system under the incomplete knowledge of the failure phenomenon.

9. CONCLUSION

In this paper, we have analyzed a generalized software rejuvenation model under two different criteria and developed the statistical non-parametric algorithms to estimate the optimal software rejuvenation schedules. In particular, we have considered two special cases: steady-state availability and expected cost per unit time in the steady state. The resulting estimators for the optimal software rejuvenation schedules have quite nice convergence properties and are useful to apply to a real software operation without specifying the underlying failure time distribution. In fact, the measurement-based approach [12, 25] to perform the effective software rejuvenation requires much effort to measure the physical characteristics for the system. Also, the model-based approaches studied in the literature [4, 8, 10, 20, 21] can not explain the software aging phenomenon completely, since the underlying failure time distribution is unknown in many cases. The statistical approach developed in this paper is simple, but can guarantee the real optimal software rejuvenation schedule if the number of failure time data points becomes large. Such an on-line estimation algorithm should be applied to other complex software systems, as the transaction-based software systems. In the future, we plan to develop non-parametric estimation algorithms for other high assurance software systems with rejuvenation.

REFERENCES

[1] Avriel, M. (1976) *Nonlinear Programming: Analysis and Methods*. Prentice-Hall, Inc., Englewood Cliffs, NJ.

[2] Avritzer, A. and Weyuker, E. J. (1997) Monitoring smoothly degrading systems for increased dependability. *Empirical Software Engineering*, **2**, 159–77.

[3] Barlow, R. E. and Campo, R. (1975) Total time on test processes and applications to failure data analysis. In Barlow, R. E., Fussell, J. and Singpurwalla, N. D. (eds.), *Reliability and Fault Tree Analysis*. SIAM, Philadelphia, 451–481.

[4] Bobbio, A. and Sereno, M. (1998) Fine grained software rejuvenation models. *Proc.* 3rd *IEEE Int'l Computer Performance & Dependability Symp.*, IEEE Computer Society Press, Los Alamitos, CA, 4–12.

[5] Dohi, T., Goševa-Popstojanova, K. and Trivedi, K.S. (2000) Analysis of software cost models with rejuvenation. *Proc. 5th IEEE Int'l Symp. High Assurance Systems Engineering*, IEEE Computer Society Press, Los Alamitos, CA, 25–34.

[6] Dohi, T., Goševa-Popstojanova, K. and Trivedi, K.S. (2000) Statistical non-parametric algorithms to estimate the optimal software rejuvenation schedule. *Proc. 2000 Pacific Rim Int'l Symp. Dependable Comput.*, IEEE Computer Society Press, Los Alamitos, CA, 77–84.

[7] Dohi, T., Kaio, N. and Osaki, S. (1999) Optimal checkpointing and rollback strategies with media failures: statistical estimation algorithms. *Proc. 1999 Pacific Rim Int'l Symp. Dependable Comput.*, IEEE Computer Society Press, Los Alamitos, CA, 161–168.

[8] Garg, S., Huang, Y., Kintala, C. and Trivedi, K. S. (1995) Time and load based software rejuvenation: policy, evaluation and optimality. *Proc.* 1st *Fault-Tolerant Symp.*, 22–25.

[9] Garg, S., Huang, Y., Kintala, C. and Trivedi, K. S. (1996) Minimizing completion time of a program by checkpointing and rejuvenation *Proc. 1996 ACM SIGMETRICS Conf.*, ACM, Cambridge, MA, 252–261.

[10] Garg, S., Pfening, S., Puliafito, A., Telek, M. and Trivedi, K. S. (1998) Analysis of preventive maintenance in transactions based software systems. *IEEE Trans. Computers*, **47**, 96–107.

[11] Garg, S., Puliafito, A., Telek, M. and Trivedi, K. S. (1995) Analysis of software rejuvenation using Markov regenerative stochastic Petri net. *Proc.* 6th *Int'l Symp. on Software Reliability Eng.*, IEEE Computer Society Press, Los Alamitos, 24–27.

[12] Garg, S., Van Moorsel, A., Vaidyanathan, K. and Trivedi, K. S. (1998) A methodology for detection and estimation of software aging. *Proc.* 9th *Int'l Symp. on Software Reliability Eng.*, IEEE Computer Society Press, Los Alamitos, 282–292.

[13] Giardo, G., Marie, R. A., Sericola, B. and Trivedi, K. S. (1990) Performability analysis using semi-Markov reward processes. *IEEE Trans. Computers*, **39**, 1251–1264.

[14] Gray, J. (1986) Why do computers stop and what can be done about it? *Proc.* 1986 *Symp. on Reliability in Distributed Software and Database Systems*, 3–12.

[15] Gray, J. and Siewiorek, D. P. (1991) High-availability computer systems. *IEEE Computer*, **9**, 39–48.

[16] Hsueh, M. C., Iyer, R. S. and Trivedi, K. S. (1988) Performability modeling based on real data: a case study. *IEEE Trans. Computers*, **37**, 478–484.

[17] Huang, Y., Kintala, C., Kolettis, N. and Funton, N. D. (1995) Software rejuvenation: analysis, module and applications. *Proc. 25th IEEE Int'l Symp. on Fault Tolerant Computing*, IEEE Computer Society Press, Los Alamitos, CA, 381–390.

[18] Marshall, E. (1992) Fatal error: how Patriot overlooked a Scud. *Science*, **255**, 1347.

[19] Meyer, J. F. (1980) On evaluating the performability of degradable computing systems. *IEEE Trans. Computers*, **29**, 720–731.

[20] Okamura, H., Fujimoto, A., Dohi, T., Osaki, S. and Trivedi, K. S. (2000) The optimal preventive maintenance policy for a software system with multi server station. *Proc. 6th ISSAT Int'l Conf. Reliability and Quality in Design*, 275–279.

[21] Pfening, S., Garg, S., Puliafito, A., Telek, M. and Trivedi, K.S. (1996) Optimal rejuvenation for tolerant-ing soft failure. *Performance Evaluation*, **27/28**, 491–506.

[22] Smith, R. M., Trivedi, K. S. and Ramesh, A. V. (1988) Performability analysis: measures, an algorithm, and a case study. *IEEE Trans. Computers*, **37**, 406–417.

[23] Tai, A. T., Alkalai, L. and Chau, S. N. (1998) On-board preventive maintenance for long-life deep-space missions: a model-based analysis. *Proc. 3rd IEEE Int'l Computer Performance & Dependability Symp.*, IEEE Computer Society Press, Los Alamitos, CA, 196–205.

[24] Trivedi, K. S., Vaidyanathan, K. and Goševa-Postojanova, K. (2000) Modeling and analysis of software aging and rejuvenation. *Proc. 33rd Annual Simulation Symp.*, IEEE Computer Society Press, Los Alamitos, CA, 270–279.

[25] Vaidyanathan, K. and Trivedi, K. S. (1999) A measurement-based model for estimation of resource exhaustion in operational software systems. *Proc. 10th Int'l Symp. on Software Reliability Eng.*, IEEE Computer Society Press, Los Alamitos, 84–93.

ACKNOWLEDGEMENTS

This material is based upon the support by Grant-in-Aid for Scientific Research from the Ministry of Education, Science and Culture of Japan under Grant Nos. 13780367 and 13480109, the Department of Defense, Alcatel, Telcordia and Aprisma Management Technologies, via a CACC core project, NC, USA, and Jet Propulsion Laboratory, CA, USA under award JPL REE 1216658.

TABLE 1. Effect of the ratio μ_a/μ_c on the optimal rejuvenation schedule in availability model: $\mu_c = 3.0 \times 10^{-1}$, $\mu_0 = 2.4 \times 10^2$, $\lambda_f = 2.16 \times 10^3$, $\beta = 2.0$.

μ_a/μ_c	t_0^*	$A(t_0^*)$
2	2364.0	0.99976
3	1538.5	0.99969
4	1207.4	0.99963
5	1013.6	0.99959
6	885.3	0.99955
7	788.7	0.99952
8	715.8	0.99949
9	657.7	0.99947
10	609.6	0.99945
11	569.0	0.99943

TABLE 2. Effect of the ratio λ_f/μ_0 on the optimal rejuvenation schedule in availability model: $\mu_a = 6.0 \times 10^{-1}$, $\mu_c = 3.0 \times 10^{-1}$, $\beta = 2.0$, $\mu_0 = 2.4 \times 10^2$.

λ_f/μ_0	t_0^*	$A(t_0 *)$
2	359.7	0.99927
3	629.8	0.99943
4	1180.2	0.99961
5	1484.7	0.99966
6	1780.7	0.99971
7	2053.3	0.99974
8	2364.1	0.99976
9	2643.8	0.99978
10	2966.7	0.99980
11	3193.7	0.99982

TABLE 3. Effect of the cost ratio c_s/c_p on the rejuvenation schedule in cost model: $\mu_a = 6.0 \times 10^{-1}$, $\mu_c = 3.0 \times 10^{-1}$, $\mu_0 = 2.4 \times 10^2$, $\beta = 2.0$, $\lambda_f = 2.16 \times 10^3$, $c_p = 5.0 \times 10^2$.

c_s/c_p	t_0^*	$C(t_0^*)$
2	1207.10	0.1829
3	883.35	0.2230
4	715.56	0.2529
5	609.20	0.2767
6	534.35	0.2968
7	478.16	0.3139
8	434.07	0.3288
9	398.37	0.3420
10	368.74	0.3538
11	343.67	0.3645

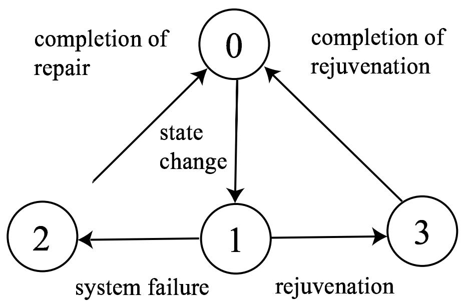

FIGURE 1. State transition diagram.

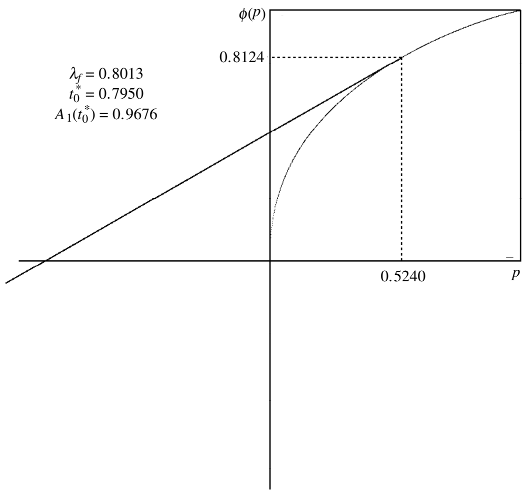

FIGURE 2. Determination of the optimal software rejuvenation schedule maximizing the steady-state availability on the two-dimensional graph: $\theta = 9.03 \times 10^{-1}$, $\beta = 2.34$, $\mu_0 = 4.0 \times 10^{-1}$, $\mu_a = 4.0 \times 10^{-2}$, $\mu_c = 3.0 \times 10^{-1}$.

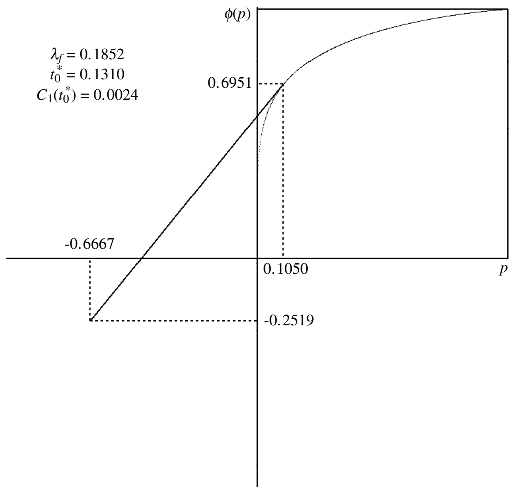

FIGURE 3. Determination of the optimal software rejuvenation schedule minimizing the expected cost per unit time in the steady state on the two-dimensional graph: $\theta = 2.0 \times 10^{-2}$, $\beta = 5.2$, $\mu_0 = 2.0 \times 10^{-2}$, $\mu_a = 5.0 \times 10^{-2}$, $\mu_c = 4.0 \times 10^{-2}$, $c_s = 2.0 \times 10^{-2}$, $c_p = 1.0 \times 10^{-2}$.

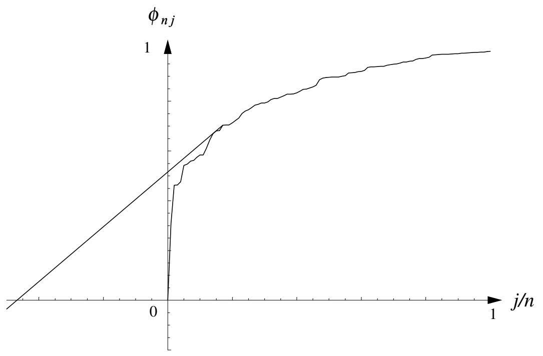

FIGURE 4. Estimation of the optimal software rejuvenation schedule maximizing the steady-state availability on the two-dimensional graph: $\theta = 9.0 \times 10^{-1}$, $\beta = 4.0$, $\mu_0 = 2.0$, $\mu_a = 4.0 \times 10^{-2}$, $\mu_c = 3.0 \times 10^{-2}$.

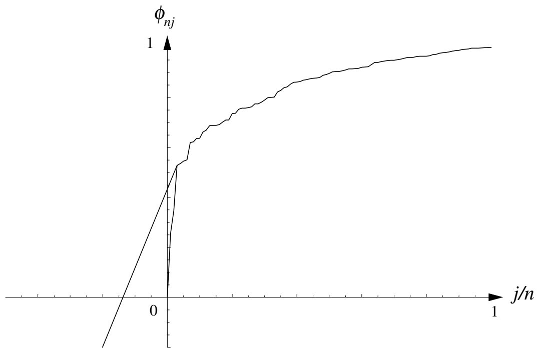

FIGURE 5. Estimation of the optimal software rejuvenation schedule minimizing the expected cost per unit time in the steady state on the two-dimensional graph: $\theta = 9.0 \times 10^{-1}$, $\beta = 4.0$, $\mu_0 = 2.0$, $\mu_a = 4.0 \times 10^{-2}$, $\mu_c = 3.0 \times 10^{-2}$, $c_s = 2.0 \times 10^{-2}$, $c_p = 1.0 \times 10^{-2}$.

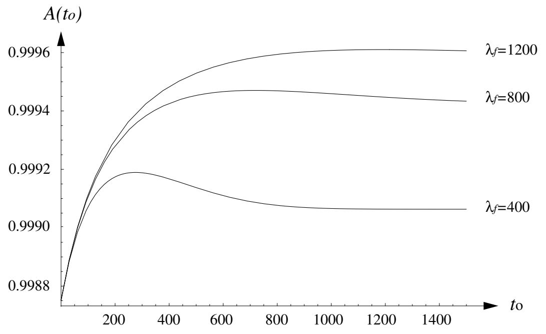

FIGURE 6. Behaviour of the steady-state availability: $\mu_a = 6.0 \times 10^{-1}$, $\mu_c = 3.0^{-1}$, $\mu_0 = 2.4 \times 10^{2}$, $\beta = 2.0$.

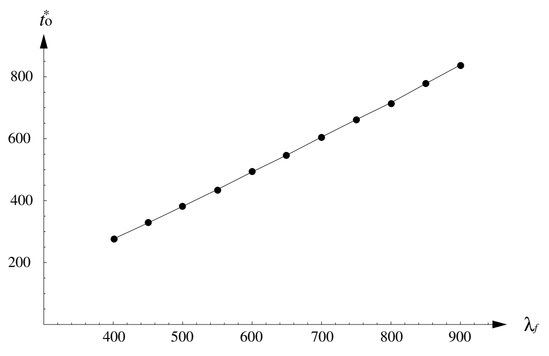

FIGURE 7. Effect of the MTTF on the optimal software rejuvenation time in the availability model: $\mu_a = 6.0\times10^{-1}$, $\mu_c = 3.0 \times 10^{-1}$, $\mu_0 = 2.4 \times 10^2$, $\beta = 2.0$.

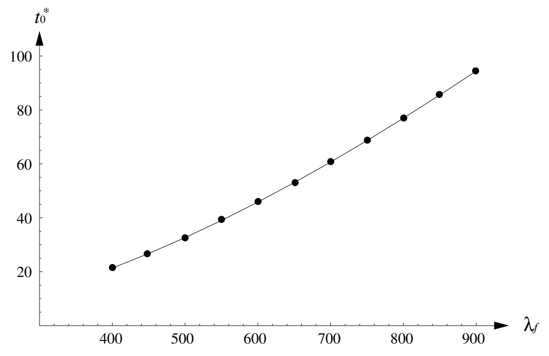

FIGURE 10. Effect of the MTTF on the optimal software rejuvenation time in the cost model: $\mu_a = 6.0 \times 10^{-1}$, $\mu_c = 3.0 \times 10^{-1}$, $\mu_0 = 2.4 \times 10^2$, $\beta = 2.0$, $c_s = 5.0 \times 10^3$, $c_p = 5.0 \times 10^2$.

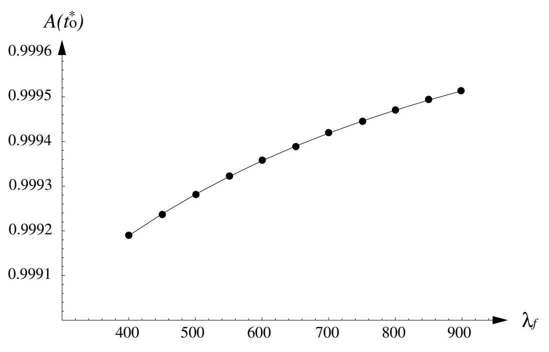

FIGURE 8. Effect of the MTTF on the maximum steady-state availability: $\mu_a = 6.0 \times 10^{-1}$, $\mu_c = 3.0 \times 10^{-1}$, $\mu_0 = 2.4 \times 10^2$, $\beta = 2.0$.

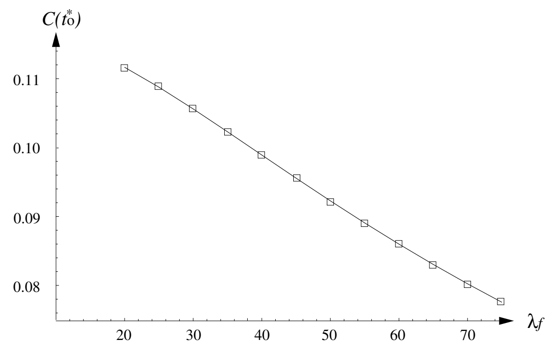

FIGURE 11. Effect of the MTTF on the minimum expected cost: $\mu_a = 6.0\times10^{-1}$, $\mu_c = 3.0\times10^{-1}$, $\mu_0 = 2.4\times10^2$, $\beta = 2.0$, $c_s = 5.0 \times 10^3$, $c_p = 5.0 \times 10^2$.

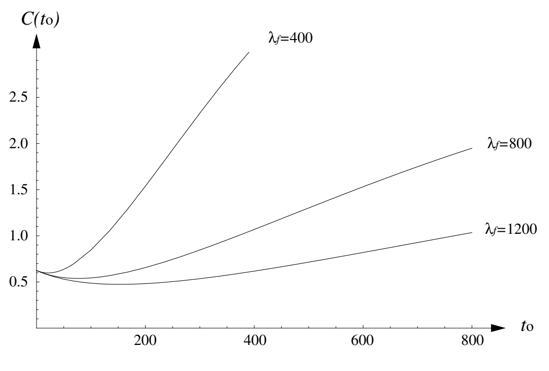

FIGURE 9. Behaviour of the expected cost: $\mu_a = 6.0 \times 10^{-1}$, $\mu_c = 3.0\times10^{-1}$, $\mu_0 = 2.4\times10^2$, $\beta = 2.0$, $c_s = 5.0\times10^3$, $c_p = 5.0 \times 10^2$.

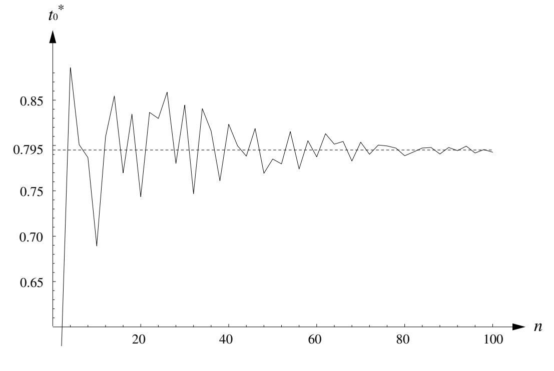

FIGURE 12. Asymptotic behaviour of the optimal software rejuvenation schedule under the steady-state availability criterion: $\theta = 9.03 \times 10^{-1}$, $\beta = 2.34$, $\mu_0 = 4.0 \times 10^{-1}$, $\mu_a = 4.0 \times 10^{-2}$, $\mu_c = 3.0 \times 10^{-2}$.

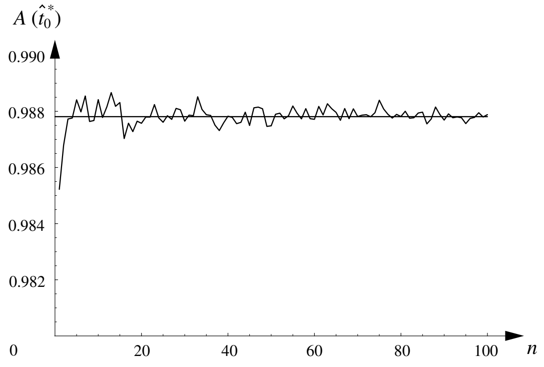

FIGURE 13. Asymptotic behaviour of the maximum steady-state availability: $\theta = 9.03 \times 10^{-1}$, $\beta = 2.34$, $\mu_0 = 4.0 \times 10^{-1}$, $\mu_a = 4.0 \times 10^{-2}$, $\mu_c = 3.0 \times 10^{-2}$.

TABLE 4. Effect of the ratio μ_a/μ_c on the rejuvenation schedule in cost model: $\mu_c = 3.0 \times 10^{-1}$, $\mu_0 = 2.4 \times 10^2$, $\beta = 2.0$, $\lambda_f = 2.16 \times 10^3$, $c_s = 5.0 \times 10^3$, $c_p = 5.0 \times 10^2$.

μ_a/μ_c	t_0^*	$C(t_0^*)$
2	368.74	0.3538
3	272.29	0.3988
4	218.11	0.4295
5	182.81	0.4523
6	157.76	0.4700
7	138.98	0.4843
8	124.33	0.4960
9	112.55	0.5058
10	102.86	0.5142
11	94.74	0.5215

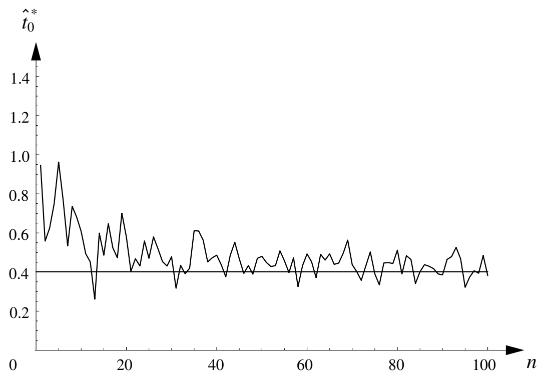

FIGURE 14. Asymptotic behaviour of the optimal software rejuvenation schedule under the expected cost criterion: $\theta = 9.0 \times 10^{-1}$, $\beta = 4.0$, $\mu_0 = 2.0$, $\mu_a = 4.0 \times 10^{-2}$, $\mu_c = 3.0 \times 10^{-2}$, $c_s = 2.0 \times 10^{-2}$, $c_p = 1.0 \times 10^{-2}$.

TABLE 5. Effect of the ratio λ_f/μ_0 on the rejuvenation schedule in cost model: $\mu_a = 6.0 \times 10^{-1}$, $\mu_c = 3.0 \times 10^{-1}$, $\mu_0 = 2.4 \times 10^2$, $\beta = 2.0$, $c_s = 5.0 \times 10^3$, $c_p = 5.0 \times 10^2$.

λ_f/μ_0	t_0^*	$C(t_0^*)$
2	30.23	0.5873
3	63.82	0.5511
4	105.42	0.5121
5	152.51	0.4741
6	203.34	0.4390
7	256.78	0.4073
8	312.08	0.3790
9	368.74	0.3538
10	426.41	0.3314
11	484.85	0.3114

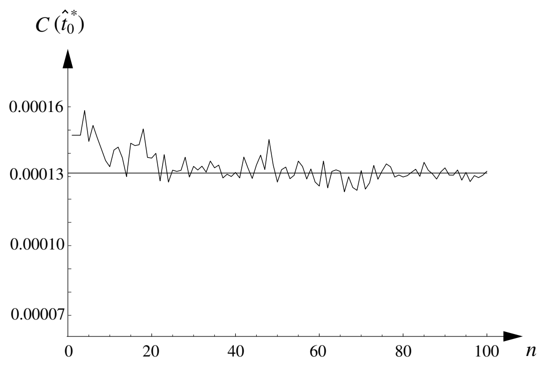

FIGURE 15. Asymptotic behaviour of the minimum expected cost: $\theta = 9.0 \times 10^{-1}$, $\beta = 4.0$, $\mu_0 = 2.0$, $\mu_a = 4.0 \times 10^{-2}$, $\mu_c = 3.0 \times 10^{-2}$, $c_s = 2.0 \times 10^{-2}$, $c_p = 1.0 \times 10^{-2}$.