Estimating Software Rejuvenation Schedules in High-Assurance Systems

TADASHI DOHI¹, KATERINA GOŠEVA-POPSTOJANOVA² AND KISHOR TRIVEDI³

¹Department of Industrial and Systems Engineering, Hiroshima University, Higashi-Hiroshima 739-8527, Japan
²Lane Department of Computer Science and Electrical Engineering, West Virginia University, Morgantown, WV 26506-6109, USA
³Department of Electrical and Computer Engineering, Duke University, Durham, NC 27708-0294, USA
Email: dohi@gal.sys.hiroshima-u.ac.jp

Software rejuvenation is a preventive maintenance technique that has been extensively studied in recent literature. In this paper, we extend the classical result by Huang et al. (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 the viewpoints of system availability and economic justification, and examine asymptotic properties for the statistical estimation algorithms.

Received 2 November 2000; revised 2 May 2001

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 foster an urgent need for high-assurance systems. To achieve high assurance, developers must use consistent, rigorous methods throughout the development process, from requirement 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 [1]. 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. [1] 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 itself as gradual performance degradation. Software aging has also been observed in widely used software like Netscape and xrm. Perhaps the most vivid example of aging in safety-critical systems is the Patriot’s software [3], 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, the application user may not have 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 [4]. Since transient failures will disappear if the operation is retried later in a slightly different context, it is difficult to characterize their root origin. Therefore, the residual faults (sometimes called Heisenbugs [5]) have to be tolerated in the operational phase. Usual strategies to deal with failures in the 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 [1]. 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 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 [1, 2], 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 [6]. 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 occurring. 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. [1]. 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 reward rate [7, 8, 9, 10]. As 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 cannot 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 schedules 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 [11]. 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, 13]. The model-based approach is aimed at evaluating the effectiveness of software rejuvenation and determining the optimal schedules to perform rejuvenation. Next, we present a 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. [1] 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. [14] introduced into the model the idea of periodic rejuvenation. To deal with deterministic intervals between successive rejuvenations the system behaviour was represented through a Markov regenerative stochastic Petri net model. The subsequent work [15] involves arrival and queuing 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 [16] 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 systems presented in [17]. This model was recently generalized in [18] by considering multiple servers.

The fine grained software rejuvenation model presented in [19] takes a different approach to the characterization of 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 [20] 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 [6].

This paper results from an elaboration and extension of our earlier work [21, 22]. Of particular interest is how to determine the optimal schedule for performing 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. [1]. 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 rejuvenation to a real system operation.

3. MODEL DESCRIPTION

First, we introduce the basic software rejuvenation model proposed by Huang et al. [1]. 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 \( P_r[Z \leq t] = F_0(t) \) with finite mean \( \mu_0 (>0) \). 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 [1, 14]. The transition diagram for the semi-Markov model is depicted in Figure 1.

Define the failure time \( X \) (from State 1) and the repair time \( Y \), having the distribution functions \( P_r[X \leq t] = F_f(t) \) and \( P_r[Y \leq t] = F_a(t) \) with finite means \( \lambda_f (>0) \) and \( \mu_a (>0) \), 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 \( \mu_c (>0) \)), 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

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

We refer to \( t_0 (\geq 0) \) as the software rejuvenation schedule in this paper and \( U(\cdot) \) is the unit step function. The 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. [1].

4. SEMI-MARKOVIAN REWARD ANALYSIS

For the semi-Markov process described in Section 3, define the transition probability \( Q_{ij}(t) \) \((i, j = 0, \ldots, 3, i \neq j)\). Also, we define the Laplace–Stieltjes transform (LST) of the transition probability by

\[
q_{ij}(s) = \int_0^\infty \exp(-st)Q_{ij}(t)\,dt.
\]

It is straightforward to obtain

\[
q_{01}(s) = \int_0^\infty \exp(-st)\,dF_0(t),
\]

\[
q_{12}(s) = \int_0^\infty \exp(-st)\,dF_f(t),
\]

\[
q_{13}(s) = \int_0^\infty \exp(-st)\,dF_r(t),
\]

\[
q_{20}(s) = \int_0^\infty \exp(-st)\,dF_a(t),
\]

\[
q_{30}(s) = \int_0^\infty \exp(-st)\,dF_c(t),
\]

where, in general, \( \psi(\cdot) = 1 - \psi(\cdot) \). 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

\[
h_{00}(s) = \int_0^\infty \exp(-st)\,dH_{00}(t)
= q_{01}(s)q_{12}(s)q_{20}(s) + q_{01}(s)q_{13}(s)q_{30}(s).
\]

We are interested in the derivation of the transient probability to stay in State \( j \) \((j = 0, \ldots, 3)\) 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 \( j \) \((j = 0, \ldots, 3)\) and its LST by \( P_{0j}(t) \) and \( p_{0j}(s) = \int_0^\infty \exp(-st) dP_{0j}(t) \), respectively. After some manipulation, we have

\[
\begin{align*}
p_{00}(s) &= q_{10}(s)/\overline{T}_0(s), \quad (8) \\
p_{01}(s) &= q_{10}(s)(\overline{T}_{12}(s) - q_{13}(s))/\overline{T}_0(s), \quad (9) \\
p_{02}(s) &= q_{01}(s)q_{12}(s)/\overline{T}_{20}(s)/\overline{T}_0(s), \quad (10) \\
p_{03}(s) &= q_{01}(s)q_{13}(s)/\overline{T}_{30}(s)/\overline{T}_0(s). \quad (11)
\end{align*}
\]

Next, we assign reward rate \( \xi_j \in (-\infty, \infty) \) to each state \( j \) \((=0, 1, 2, 3)\) of the semi-Markov process. Following [9], we define the expected instantaneous reward rate as

\[
PP(t) = \sum_{j=0}^3 \xi_j p_{0j}(t). \quad (12)
\]

Also, the expected steady-state reward rate is

\[
PP = \lim_{t \to \infty} PP(t) = \lim_{t \to \infty} \frac{\sum_{j=0}^3 \xi_j p_{0j}(t)}{T(t)} = \lim_{s \to 0} \frac{\sum_{j=0}^3 \xi_j p_{0j}(s)}{T_0(s)} \quad (13)
\]

From the familiar probabilistic argument, we can derive the expected steady-state reward rate as the function of \( t_0 \) for this model as follows.

\[
\begin{align*}
PP(t_0) &= \frac{\xi_0\mu_0 + \xi_1 \int_0^{t_0} F_f(t) \, dt + \xi_2\mu_a F_f(t_0) + \xi_3\mu_c F_f(t_0)}{\mu_0 + \mu_a F_f(t_0) + \mu_c F_f(t_0) + \int_0^{t_0} F_f(t) \, dt} \\
&= K(t_0)/T(t_0). \quad (14)
\end{align*}
\]

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

\[
q_p(t_0) = \{\xi_1 + (\xi_2\mu_a - \xi_3\mu_c) r_f(t_0)} \} T(t_0) - \{( \mu_a - \mu_c) r_f(t_0) + 1 \} K(t_0). \quad (15)
\]

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 \( t_0 \) [23]. The following result is useful for characterizing 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 \( \inf_{0 \leq t_0 < \infty} PP(t_0) \) 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 \( \inf_{0 \leq t_0 < \infty} PP(t_0) \) 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 result of selecting 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 \( \xi_j \) \((j = 0, 1, 2, 3)\).

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 sub-class of the expected reward rate, but can give 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 0 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 (14), the steady-state availability becomes

\[
A(t_0) = \Pr\{\text{software system is operative} \mid \text{in the steady state} \} = \lim_{s \to 0} \{p_{00}(s) + p_{01}(s)\} = \frac{\mu_0 + \mu_a F_f(t_0) + \mu_c F_f(t_0) + \int_0^{t_0} F_f(t) \, dt}{\mu_0 + \mu_a F_f(t_0) + \mu_c F_f(t_0) + \int_0^{t_0} F_f(t) \, dt} = S(t_0)/T(t_0). \quad (16)
\]

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:

**A1.** \( \mu_a > \mu_c \).

The assumption (A1) 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 assumption (A1) holds. Define the following nonlinear function:

\[
q_A(t_0) = T(t_0) - ((\mu_a - \mu_c) r_f(t_0) + 1)S(t_0). \quad (17)
\]

(i) If \( q_A(0) > 0 \) and \( q_A(\infty) < 0 \), then there exists a finite and unique optimal software rejuvenation schedule \( t_0^* \).
\begin{equation}
A(t_0^*) = \frac{1}{(\mu_a - \mu_c) r_f(t_0^*) + 1}.
\end{equation}

(ii) If \( q_A(0) \leq 0 \), 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 \( q_A(\infty) \geq 0 \), 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 assumption (A1) 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
\begin{equation}
\frac{dq_A(t_0)}{dt_0} = -\frac{(\mu_a - \mu_c) S(t_0) dr_f(t_0)}{d\theta_0}.
\end{equation}

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 assumption (A1). 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 \( q_A(0) \leq 0 \) or \( q_A(\infty) \geq 0 \), 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 complete.

It is easy to check that the above result implies the result in Huang et al. [1], 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 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:

(A2) \( c_s > c_p \).

Assumption (A2) 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 (14), the expected cost per unit time in the steady state becomes
\begin{equation}
C(t_0) = \lim_{t \to \infty} \int_{0}^{t} E[\text{total cost during } (0, t)] dt
= \lim_{t \to \infty} \left[ c_s P_{02}(s) + c_p P_{03}(s) \right]
= \lim_{t \to 0} \left[ c_s P_{02}(s) + c_p P_{03}(s) \right]
= \frac{c_s \mu_a F_f(t_0) + c_p \mu_c F_f(t_0)}{\mu_0 + \mu_c F_f(t_0) + \mu_c F_f(t_0) + \int_{0}^{t_0} F_f(t) dt}
= V(t_0)/T(t_0).
\end{equation}

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 assumptions (A1) and (A2) hold. Define the following nonlinear function:
\begin{equation}
q_C(t_0) = (c_s \mu_a - c_p \mu_c) S(t_0) - [(\mu_a - \mu_c) r_f(t_0)] V(t_0).
\end{equation}

(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
\begin{equation}
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}.
\end{equation}

(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 assumptions (A1) and (A2) 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

\[
\frac{d^2 C(t_0)}{dt_0^2} = \frac{d^2 f(t_0)}{dt_0^2} + \left[ (c_i \mu_a - c_p \mu_c) S(t_0) - (\mu_a - \mu_c) V(t_0) \right].
\]

(23)

If the term in the bracket on the right-hand side in (23) is negative, then \( c_i + \mu_c (c_i - c_p)/(\mu_a - \mu_c) \leq C(t_0) \) for all \( t_0 \in [0, \infty) \). Since \( \mu_c (c_i - c_p)/(\mu_a - \mu_c) > 0 \) from assumptions (A1) and (A2), 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 \( c_i + \mu_c (c_i - c_p)/(\mu_a - \mu_c) > C(t_0) \).

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 complete.

In Sections 5 and 6, we derived the optimal software rejuvenation schedules which maximize the steady-state availability or minimize 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: \( \mu_0, \mu_c, \mu_a \), and the failure time distribution \( F_f(t) \), as well as the reward rates \( c_i \) 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 [24], define the scaled total time on test (TTT) transform of the failure time distribution:

\[
\phi(p) = \frac{1}{\lambda_f} \int_0^{F_f^{-1}(p)} \frac{F_f(t)}{F_f(t)} dt,
\]

(24)

where

\[
F_f^{-1}(p) = \inf\{t_0: F_f(t_0) \geq p\} \quad (0 \leq p \leq 1).
\]

(25)

It is well known [24] 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 assumption (A1) 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 that

\[
\max_{0 \leq p \leq 1} \phi(p) + \alpha_a = \frac{\alpha_a}{p + \beta_a},
\]

(26)

where

\[
\alpha_a = \lambda_f + \mu_0,
\]

(27)

\[
\beta_a = \mu_c (\mu_a - \mu_c).
\]

(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:

\[
F_f(t) = 1 - e^{-\left(\frac{t}{\beta_a}\right)^{\theta}}
\]

(29)

with shape parameter \( \beta = 2.34 \) and scale parameter \( \theta = 0.903 \). In Figure 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 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 assumptions (A1) and (A2) 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 \( p^* \) \( (0 \leq p^* \leq 1) \) such that

\[
\max_{0 \leq p \leq 1} \phi(p) + \alpha_c = \frac{\alpha_c}{p + \beta_c},
\]

(30)

where

\[
\alpha_c = \frac{(c_i - c_p) \mu_c (\mu_0 + \mu_a)}{\lambda_f (c_i \mu_a - c_p \mu_c)},
\]

(31)

\[
\beta_c = \frac{c_p \mu_c}{c_i \mu_a - c_p \mu_c}.
\]

(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 \( p^* \) \( (0 \leq p^* \leq 1) \) 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
Determination of the optimal software rejuvenation schedule minimizing the expected cost per unit time in the steady state 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: $\lambda = 0.8013$, $\lambda_0 = 0.7950$, $A(t_0) = 0.9676$.

FIGURE 2. Determination of the optimal software rejuvenation schedule maximizing the steady-state availability on the two-dimensional graph: $\theta = 0.031 \times 10^{-1}$, $\beta = 2.34$, $\mu_0 = 4.0 \times 10^{-1}$, $\mu_a = 4.0 \times 10^{-2}$, $\mu_c = 3.0 \times 10^{-1}$.

The time distribution is Weibull as in (29). In Figure 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 $0 = x_0 \leq x_1 \leq x_2 \leq \ldots \leq x_n$ 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 $t_0^*$ which maximizes $A(t_0)$ is given by $x_j^*$, where

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

and $\lambda_f$ in (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 an ordered complete sample $0 = x_0 \leq x_1 \leq x_2 \leq \ldots \leq x_n$ 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 $t_0^*$ which minimizes $C(t_0)$ is given by $x_j^*$, where

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

and $\lambda_f$ in (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 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 [24] and the strong law of large numbers. The graphical procedure...
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: \( \theta = 9.0 \times 10^{-1}, \beta = 4.0, \mu_0 = 2.0, \mu_a = 4.0 \times 10^{-2}, \mu_c = 3.0 \times 10^{-2}, \).

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 \( A(\hat{t}^*_0) = 0.9875813 \), respectively. In Figure 5, we show the estimation results of the cost model, where the failure-time data is generated by the same Weibull distribution as Figure 4. In Figure 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.

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

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 (29). Define the mean time to failure (MTTF) 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 Figures 7 and 8, respectively. As the MTTF gets larger, both the rejuvenation schedule and its associated steady-state availability become monotonically decreasing.
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 (29). As the MTTF becomes larger for a fixed shape parameter, the optimal software rejuvenation schedule which minimizes the expected cost takes a larger value for each case. Dependences of the optimal software rejuvenation time and its associated cost value on the MTTF are also investigated in Figures 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 $\mu_a/\mu_c$ in the availability model. As the ratio $\mu_a/\mu_c$ increases, the rejuvenation schedule monotonically decreases, and the maximum availability also decreases. Thus, the higher the mean repair time relative to the mean time to perform rejuvenation the 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 $\lambda_f/\mu_0$. If the MTTF becomes larger for a fixed value of $\mu_0$, i.e. the system tends to be more reliable, the resulting optimal rejuvenation schedule can take a rather large value, that is, the rejuvenation should be performed less and less frequently.
TABLE 1. Effect of the ratio $\mu_a/\mu_c$ on the optimal rejuvenation schedule in the availability model: $\mu_c = 3.0 \times 10^{-1}$, $\mu_0 = 2.4 \times 10^2$, $\lambda_f = 2.16 \times 10^3$, $\beta = 2.0$.

<table>
<thead>
<tr>
<th>$\mu_a/\mu_c$</th>
<th>$t^*_0$</th>
<th>$A(t^*_0)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>2364.0</td>
<td>0.99976</td>
</tr>
<tr>
<td>3</td>
<td>1538.5</td>
<td>0.99969</td>
</tr>
<tr>
<td>4</td>
<td>1207.4</td>
<td>0.99963</td>
</tr>
<tr>
<td>5</td>
<td>1013.6</td>
<td>0.99959</td>
</tr>
<tr>
<td>6</td>
<td>885.3</td>
<td>0.99955</td>
</tr>
<tr>
<td>7</td>
<td>788.7</td>
<td>0.99952</td>
</tr>
<tr>
<td>8</td>
<td>715.8</td>
<td>0.99949</td>
</tr>
<tr>
<td>9</td>
<td>657.7</td>
<td>0.99947</td>
</tr>
<tr>
<td>10</td>
<td>609.6</td>
<td>0.99945</td>
</tr>
<tr>
<td>11</td>
<td>569.0</td>
<td>0.99943</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>$\lambda_f/\mu_0$</th>
<th>$t^*_0$</th>
<th>$A(t^*_0)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>359.7</td>
<td>0.99927</td>
</tr>
<tr>
<td>3</td>
<td>629.8</td>
<td>0.99943</td>
</tr>
<tr>
<td>4</td>
<td>1180.2</td>
<td>0.99961</td>
</tr>
<tr>
<td>5</td>
<td>1484.7</td>
<td>0.99966</td>
</tr>
<tr>
<td>6</td>
<td>1780.7</td>
<td>0.99971</td>
</tr>
<tr>
<td>7</td>
<td>2053.3</td>
<td>0.99974</td>
</tr>
<tr>
<td>8</td>
<td>2364.1</td>
<td>0.99976</td>
</tr>
<tr>
<td>9</td>
<td>2643.8</td>
<td>0.99978</td>
</tr>
<tr>
<td>10</td>
<td>2966.7</td>
<td>0.99980</td>
</tr>
<tr>
<td>11</td>
<td>3193.7</td>
<td>0.99982</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>$c_s/c_p$</th>
<th>$t^*_0$</th>
<th>$C(t^*_0)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>1207.10</td>
<td>0.1829</td>
</tr>
<tr>
<td>3</td>
<td>883.35</td>
<td>0.2230</td>
</tr>
<tr>
<td>4</td>
<td>715.56</td>
<td>0.2529</td>
</tr>
<tr>
<td>5</td>
<td>609.20</td>
<td>0.2767</td>
</tr>
<tr>
<td>6</td>
<td>534.35</td>
<td>0.2968</td>
</tr>
<tr>
<td>7</td>
<td>478.16</td>
<td>0.3139</td>
</tr>
<tr>
<td>8</td>
<td>434.07</td>
<td>0.3288</td>
</tr>
<tr>
<td>9</td>
<td>398.37</td>
<td>0.3420</td>
</tr>
<tr>
<td>10</td>
<td>368.74</td>
<td>0.3538</td>
</tr>
<tr>
<td>11</td>
<td>343.67</td>
<td>0.3645</td>
</tr>
</tbody>
</table>

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 $\mu_a/\mu_c$. 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 $\lambda_f/\mu_0$ and its associated maximum steady-state availability. The failure-time data are generated by the Weibull distribution.

8.2. Asymptotic behaviour

Next, we examine the asymptotic properties of the estimators developed in Section 7. One of the most important problems in practical applications is the speed of convergence of the estimates for the optimal software rejuvenation schedules. In other words, since large numbers 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.

THE COMPUTER JOURNAL, Vol. 44, No. 6, 2001


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 Figure 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 Figures 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 Figure 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 for designing high assurance software systems under 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 for applying to real software operations without specifying the underlying failure-time distribution. In fact, the measurement-based approach [12, 13] for performing effective software rejuvenation requires much effort to measure the physical characteristics for the system. Also, the model-based approaches studied in the literature [15, 16, 17, 18, 19] cannot 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. In the future, we plan to develop non-parametric estimation algorithms for other high assurance software systems with rejuvenation.

ACKNOWLEDGEMENTS

This work has been supported 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.

REFERENCES


