Statistical Non-Parametric Algorithms to Estimate the Optimal Software Rejuvenation Schedule *

Tadashi Dohi
Dept. of Industrial and Systems Engineering
Hiroshima University
Higashi-Hiroshima 739-8527, Japan
dohi@gal.sys.hiroshima-u.ac.jp

Katerina Goševa-Popstojanova † Kishor S. Trivedi
Dept. of Electrical and Computer Engineering
Duke University
Durham, NC 27708-0294, USA
{katerina, kst}@ee.duke.edu

Abstract

In this paper, we extend the classical result by Huang, Kintala, Kolettis and Fulton (1995), and in addition propose a modified stochastic model to determine the software rejuvenation schedule. More precisely, the software rejuvenation models are formulated via the semi-Markov processes, and the optimal software rejuvenation schedules which maximize the system availabilities are derived analytically for respective cases. Further, we develop non-parametric statistical algorithms to estimate the optimal software rejuvenation schedules, provided that the statistical complete (unsensored) sample data of failure times is given. In numerical examples, we 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. However, these requirements are very difficult to design for and guarantee, particularly in applications of nontrivial complexity. When software application executes continuously for long periods of time, it ages due to the error conditions that accrue with time and/or load.

Software aging will affect the performance of the application and eventually cause it to fail. 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 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 [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, application developer 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 further more if testing and debugging 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 slightly different context, it is difficult to characterize their root origin. Therefore, the residual faults 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.

Recently, a complementary approach to handle transient software failures, called *software rejuvenation*, was proposed [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 a software might involve garbage collection, flushing operating system kernel tables, reinitial-

*This material is based upon the support by the Department of Defense, Alcatel, Telcordia and Aprisma Management Technologies via a CACC core project, NC, USA. Also, the first author is supported in part by Telecommunication Advancement Foundation, Tokyo, Japan.

†On leave of absence from the Department of Computer Science, Faculty of Electrical Engineering, University "Sv. Kiril i Metodij", Skopje, Macedonia.

izing 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 availability and mission critical systems [1, 2, 5, 6]. Although the fault in the program still remains, performing rejuvenation occasionally or periodically prevents failures due to that fault.

Rejuvenation has the same motivation and advantages/disadvantages as preventive maintenance policies in hardware systems. Any rejuvenation typically involves an overhead, but it may prevent more severe failures from 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 due to a 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], and in addition propose modified stochastic models to determine the optimal software rejuvenation schedule. More precisely, the software rejuvenation models are formulated via the semi-Markov process, and the optimal software rejuvenation schedules which maximize the system availabilities are derived analytically for respective cases. Further, we develop non-parametric statistical algorithms to estimate the optimal software rejuvenation schedules, provided that the statistical complete (unsensored) sample data of failure times is given. These can be useful in determining the optimal rejuvenation schedule in the early part of the operational phase, since the failure time distribution can not be easily estimated from a few data samples. In numerical examples, we examine asymptotic properties of the statistical estimation algorithms.

2. Previous work

In recent years, considerable attention has been devoted to the phenomenon of software aging. For an extensive survey, the reader is referred to [7]. First, 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.* [8] introduced the idea of periodic rejuvenation into the model. To deal with deterministic interval between successive rejuvenations the system behavior was represented through a Markov regenerative stochastic Petri net model.

The subsequent work [9] 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 [10] 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 [11]. This model was recently generalized in [12] by considering multiple servers.

The 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 and to calculate some reliability measures. This seems to be restrictive for use of rejuvenation in practice, 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 modeling-based approach, the failure time distribution is fixed, for instance, the Weibull distribution, this has 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 which maximize system availabilities. Thus, we provide a very powerful method for the application of rejuvenation to a real system operation.

3. Model description

3.1. Basic model

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 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

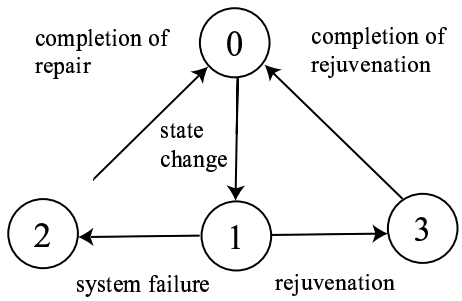

Figure 1. Semi-Markovian diagram of Model 1

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 [1, 8]. The transition diagram for Model 1 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 in the basic model referred to as Model 1 the software rejuvenation cycle is measured from the time instant just after the system enters State 1 from State 0. Denote the distribution function of the time to invoke the software rejuvenation and the distribution of the time to complete software rejuvenation 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 time 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 rejuvenative schedule* in this paper and $U(\cdot)$ is the unit step function. Hence, the underlying stochastic process is a semi-Markov process with four regeneration states. Note that under the assumption that the sojourn times in all states are exponentially distributed, Model 1 is reduced to the one in Huang *et al.* [1].

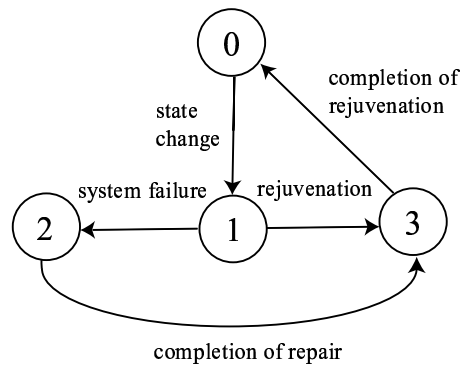

Figure 2. Semi-Markovian diagram of Model 2

3.2. Modified model

The next model is a modification of the basic model. When the repair is completed after the system failure, is the software system really renewed? Probably, the answer is "NO" in many cases. If one distinguishes software repair and the software rejuvenation, an additional rejuvenation might be needed after the software repair. For example, restarting the system after repair might require some cleanup and resuming the process execution at the checkpointed state. Such an additional rejuvenation policy has not been considered in the literature in spite of the practical need. Figure 2 is the transition diagram for Model 2. In this model, the software rejuvenation is performed just after the completion of repair as well as at the constant time t_0 after the failure probable state is entered, *i.e.*, $\min\{Z + X + Y, Z + t_0\}$.

4. Availability analysis

In this section, we formulate the system availabilities for respective models. Applying the standard technique of semi-Markov processes, the system availability for Model 1 becomes

$$\begin{aligned} A_1(t_0) &= \Pr\{\text{software system is operative in the steady} - \text{state}\} \\ &= \frac{\mu_0 + \int_0^{t_0} \overline{F}_f(t)dt}{\mu_0 + \mu_a F_f(t_0) + \mu_c \overline{F}_f(t_0) + \int_0^{t_0} \overline{F}_f(t)dt} \\ &= S_1(t_0)/T_1(t_0). \end{aligned} \tag{2}$$

Also, the system availability for Model 2 is

$$\begin{aligned} A_2(t_0) &= \frac{\mu_0 + \int_0^{t_0} \overline{F}_f(t)dt}{\mu_0 + \mu_c + \mu_a F_f(t_0) + \int_0^{t_0} \overline{F}_f(t)dt} \\ &= S_2(t_0)/T_2(t_0). \end{aligned} \tag{3}$$

The problem is to derive the optimal software rejuvenation schedule t_0^* which maximizes the system availability in the steady-state $A_i(t_0)$ $(i = 1, 2)$.

We make the following assumption:

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

The assumption (A-1) means 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 results give the optimal software rejuvenation schedule for Model i $(i = 1, 2)$. The results are given without proof for brevity.

Theorem 1: For Model 1, (1) suppose that the failure time distribution is strictly IFR (increasing failure rate) under the assumption (A-1). Define the following non-linear function:

$$q_1(t_0) = T_1(t_0) - \Big\{(\mu_a - \mu_c)r_f(t_0) + 1\Big\}S_1(t_0), \quad (4)$$

where $r_f(t) = (dF_f(t)/dt)/\overline{F}_f(t)$ is the failure rate.

(i) If $q_1(0) > 0$ and $q_1(\infty) < 0$, then there exists a finite and unique optimal software rejuvenation schedule t_0^* $(0 < t_0^* < \infty)$ satisfying $q_1(t_0^*) = 0$, and the maximum system availability is

$$A_1(t_0^*) = \frac{1}{(\mu_a - \mu_c)r_f(t_0^*) + 1}. \quad (5)$$

(ii) If $q_1(0) \leq 0$, then the optimal software rejuvenation schedule is $t_0^* = 0$, *i.e.* it is optimal to start the rejuvenation just after entering the failure probable state, and the maximum system availability is $A_1(0) = \mu_0/(\mu_0 + \mu_c)$.

(iii) If $q_1(\infty) \geq 0$, then the optimal rejuvenation schedule is $t_0^* \to \infty$, *i.e.* it is optimal not to carry out the rejuvenation, and the maximum system availability is $A_1(\infty) = (\mu_0 + \lambda_f)/(\mu_0 + \mu_a + \lambda_f)$.

(2) Suppose that the failure time distribution is DFR (decreasing failure rate) under the assumption (A-1). Then, the system availability $A_1(t_0)$ is a convex function of t_0, and the optimal rejuvenation schedule is $t_0^* = 0$ or $t_0^* \to \infty$.

It is easy to check that the result above 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 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 system 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.

Similarly, consider Model 2. In this case, the assumption (A-1) is not necessarily needed.

Theorem 2: For Model 2, (1) suppose that the failure time distribution is strictly IFR. Define the following non-linear function:

$$q_2(t_0) \quad = \quad T_2(t_0) - \Big\{\mu_a r_f(t_0) + 1\Big\}S_2(t_0). \quad (6)$$

(i) If $q_2(0) > 0$ and $q_2(\infty) < 0$, then there exists a finite and unique optimal software rejuvenation schedule t_0^* $(0 < t_0^* < \infty)$ satisfying $q_2(t_0^*) = 0$, and the maximum system availability is

$$A_2(t_0^*) = \frac{1}{\mu_a r_f(t_0^*) + 1}. \quad (7)$$

(ii) If $q_2(0) \leq 0$, then the optimal software rejuvenation schedule is $t_0^* = 0$, and the maximum system availability is $A_2(0) = \mu_0/(\mu_0 + \mu_c)$.

(iii) If $q_2(\infty) \geq 0$, then the optimal software rejuvenation schedule is $t_0^* \to \infty$, and the maximum system availability is $A_2(\infty) = (\mu_0 + \lambda_f)/(\mu_0 + \mu_a + \mu_c + \lambda_f)$.

(2) Suppose that the failure time distribution is DFR. Then, the system availability $A_2(t_0)$ is a convex function of t_0, and the optimal software rejuvenation schedule is $t_0^* = 0$ or $t_0^* \to \infty$.

In this section, we derived the optimal software rejuvenation schedules that maximize the system availabilities in the steady-state. It should be noted, however, that the optimal software rejuvenation schedule has to be determined from model parameters; μ_0, μ_c, μ_a and the failure time distribution $F_f(t)$. In other words, it is not easy in general to specify the failure time distribution in the software operational phase. 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.

5. Statistical estimation algorithms

Before developing the statistical estimation algorithms for the optimal software rejuvenation schedules, we translate the underlying problems $\max_{0 \leq t_0 < \infty} A_i(t_0)$ $(i = 1, 2)$

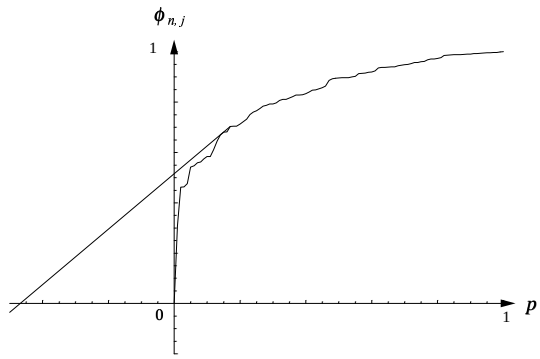

Figure 3. Estimation of the optimal software rejuvenation schedule on the two-dimensional graph (Model 1): $\theta = 0.9$**,** $\beta = 4.0$**,** $\mu_0 = 2.0$**,** $\mu_a = 0.04$**,** $\mu_c = 0.03$

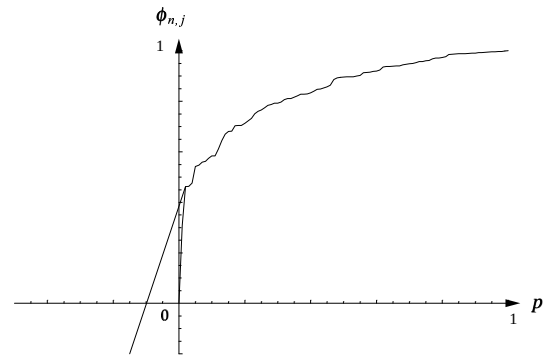

Figure 4. Estimation of the optimal software rejuvenation schedule on the two-dimensional graph (Model 2): $\theta = 0.9$**,** $\beta = 4.0$**,** $\mu_0 = 2.0$**,** $\mu_a = 0.04$**,** $\mu_c = 0.03$

to graphical ones. Following Barlow and Campo [13], define the scaled total time on test (TTT) transform of the failure time distribution:

$$\phi(p) = (1/\lambda_f) \int_0^{F_f^{-1}(p)} \overline{F}_f(t)dt, \tag{8}$$

where

$$F_f^{-1}(p) = \inf\{t_0; F_f(t_0) \geq p\}, \quad (0 \leq p \leq 1). \tag{9}$$

It is well known [13] 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 result.

Theorem 3: For Model i $(i = 1, 2)$, obtaining the optimal software rejuvenation schedule ${t_0}^*$ maximizing the system availability $A_i(t_0)$ is equivalent to obtaining p^* $(0 \leq p^* \leq 1)$ such as

$$\max_{0 \leq p \leq 1} \frac{\phi(p) + \alpha_i}{p + \beta_i}, \tag{10}$$

where $\alpha_1 = \lambda_f + \mu_0$, $\alpha_2 = \mu_0/\lambda_f$, $\beta_1 = \mu_c(\mu_a - \mu_c)$ and $\beta_2 = \mu_c/\mu_a$.

From Theorem 3 it is to see 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 $(-\beta_i, -\alpha_i) \in (-\infty, 0) \times (-\infty, 0)$ to the curve $(p, \phi(p)) \in [0, 1] \times [0, 1]$.

Next, suppose that the optimal software rejuvenation schedule has to be estimated from an ordered complete observation $0 = x_0 \leq x_1 \leq x_2 \leq \cdots \leq x_n$ 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

$$\psi_j = \sum_{k=1}^{j} (n-k+1)(x_k - x_{k-1}), \quad (j = 1, 2, \cdots, n;\ \psi_0 = 0). \tag{11}$$

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

$$F_n(x) = \begin{cases} j/n & \text{for} \quad x_j \leq x < x_{j+1}, \\ 1 & \text{for} \quad x_n \leq x, \end{cases} \tag{12}$$

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

The following result gives non-parametric statistical estimation algorithms for the optimal software rejuvenation schedules.

Theorem 4: (i) Suppose that the optimal software rejuvenation schedule has to be estimated from n ordered complete sample $0 = x_0 \leq x_1 \leq x_2 \leq \cdots \leq x_n$ 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 $\hat{t}_0^*$ which maximizes $A_i(t_0)$ $(i = 1, 2)$ is given by x_{j^*}, where

$$j^* = \left\{ j \mid \max_{0 \leq j \leq n} \frac{\phi_{nj} + \alpha_i}{j/n + \beta_i} \right\} \tag{13}$$

and λ_f in Eq. (10) 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.

It is straightforward to prove the above result in (i) from Theorem 3. The uniform convergence property in (ii) is due to the Glivenko-Cantelli lemma (*e.g.* see [13]) 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, of course, 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 often assumed in the reliability analysis, our non-parametric estimation algorithm can generate the optimal software rejuvenation schedule using the online knowledge about the observed failure times.

Figures 3 and 4 show the estimation results of the optimal software rejuvenation schedule for Model 1 and Model 2, respectively, where the failure time data is 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 $A_1(\hat{t}_0^*) = 0.987813$, respectively, in Model 1. On the other hand, in Model 2, one estimates $\hat{t}_0^* = 0.377739$ and $A_2(\hat{t}_0^*) = 0.987027$.

In this section, we transformed 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.

6. Numerical illustrations

6.1. Availability comparison

In this section, we compare two availability models and carry out the sensitivity analysis on the model parameters. Figures 5 and 6 show the behavior of the system availabilities for Model 1 and Model 2, respectively. The failure time distribution is the Weibull distribution:

$$F_f(t) = 1 - e^{-(t/\theta)^\beta} \tag{14}$$

with mean time to failure (MTTF) given 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 system 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 system availability become monotonically larger. In particular, it is found that both the rejuvenation time and the availability for Model 1 are always larger than for Model 2. Hence, if restarting the system after repair requires some cleanup and resuming the process execution at the checkpointed state, the rejuvenation schedule should be set at a smaller value than in the case without rejuvenation after the repair.

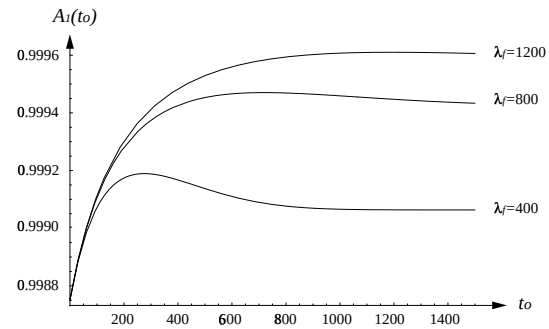

Figure 5. Behavior of the system availability for Model 1: $\mu_a = 0.6$, $\mu_c = 0.3$, $\mu_0 = 240$, $\beta = 2$

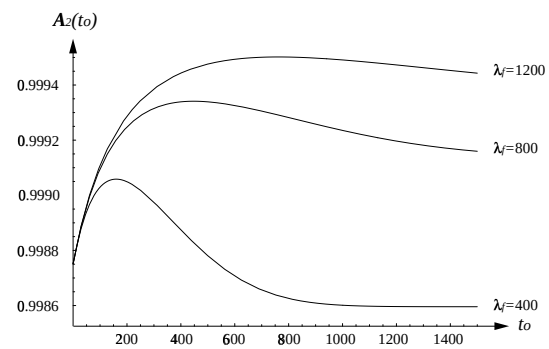

Figure 6. Behavior of the system availability for Model 2: $\mu_a = 0.6$, $\mu_c = 0.3$, $\mu_0 = 240$, $\beta = 2$

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

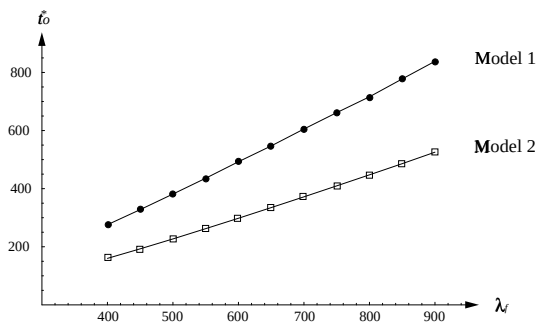

Figure 7. Comparison of two models with respect to optimal rejuvenation schedule: : $\mu_a = 0.6$**,** $\mu_c = 0.3$**,** $\mu_0 = 240$**,** $\beta = 2$

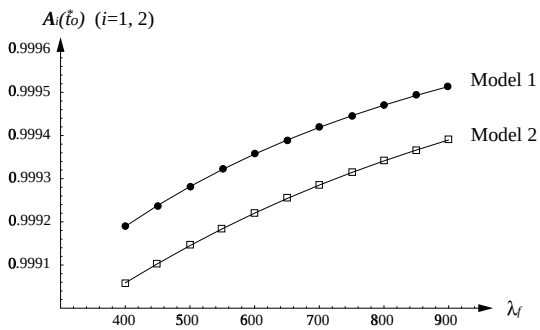

Figure 8. Comparison of two models with respect to system availability: $\mu_a = 0.6$**,** $\mu_c = 0.3$**,** $\mu_0 = 240$**,** $\beta = 2$

Table 1. Dependence of μ_a/μ_c **on the optimal rejuvenation schedule:** $\mu_c = 0.3$**,** $\mu_0 = 240$**,** $\lambda_f = 2160$**,** $\beta = 2$

	Model 1		Model 2	
μ_a/μ_c	t_0^*	$A_1(t_0^*)$	t_0^*	$A_2(t_0^*)$
2	2364.0	0.99976	1538.5	0.99969
3	1538.5	0.99969	1207.4	0.99963
4	1207.4	0.99963	1013.6	0.99959
5	1013.6	0.99959	885.3	0.99955
6	885.3	0.99955	788.7	0.99952
7	788.7	0.99952	715.8	0.99949
8	715.8	0.99949	657.4	0.99947
9	657.4	0.99947	609.6	0.99945
10	609.6	0.99945	569.0	0.99943
11	569.0	0.99943	535.1	0.99941

6.2. Asymptotic behavior

Next, we examine the asymptotic properties of the estimators developed in Section 5. 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 earlier 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 9 and 10 illustrate the asymptotic behavior of the estimates for the system availabilities. The failure time data are generated by the Weibull distribution with shape parameter $\beta = 4.0$ and scale parameter $\theta = 0.9$. In the figures, the horizontal lines denote the real maximum system availabilities for respective models.

Table 2. Dependence of λ_f/μ_0 **on the optimal rejuvenation schedule:** $\mu_a = 0.6$**,** $\mu_c = 0.3$**,** $\beta = 2$**,** $\mu_0 = 240$

	Model 1		Model 2	
λ_f/μ_o	t_0^*	$A_1(t_0 *)$	t_0^*	$A_2(t_0^*)$
2	359.7	0.99927	213.1	0.99913
3	629.8	0.99943	387.0	0.99930
4	1180.2	0.99961	571.5	0.99942
5	1484.7	0.99966	761.1	0.99950
6	1780.7	0.99971	953.8	0.99957
7	2053.3	0.99974	1148.8	0.99962
8	2364.1	0.99976	1345.0	0.99967
9	2643.8	0.99978	1538.5	0.99969
10	2966.7	0.99980	1742.5	0.99972
11	3193.7	0.99982	1937.6	0.99974

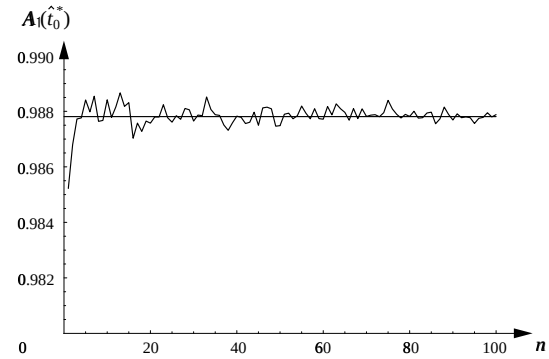

Figure 9. Asymptotic behavior of the maximum system availability for Model 1: $\theta = 0.9$**,** $\beta = 4.0$**,** $\mu_0 = 2.0$**,** $\mu_a = 0.04$**,** $\mu_c = 0.03$

In these figures, the miximum availabilities $A_i(\hat{t}_0^*)$ $(i = 1, 2)$ are calculated in accordance with the estimation algorithm in Theorem 4, where the sample mean $\hat{\lambda}_f = \sum_{k=1}^{n} x_k/n$ changes as the failure time data is observed. From Figs. 9 and 10, it is seen that the estimate of the optimal rejuvenation schedule fluctuates until the number of observations is about 20. This result enables 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.

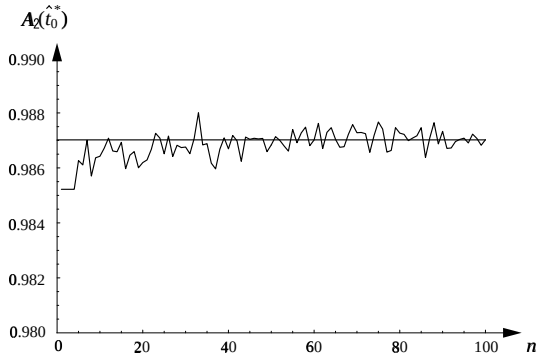

Figure 10. Asymptotic behavior of the maximum system availability for Model 2: $\theta = 0.9$**,** $\beta = 4.0$**,** $\mu_0 = 2.0$**,** $\mu_a = 0.04$**,** $\mu_c = 0.03$

7. Conclusion

In this paper, we have analyzed two generalized software rejuvenation models and developed a statistical nonparametric algorithm to estimate the optimal software rejuvenation schedules which maximize the system availabilities. The resulting estimators for the optimal software rejuvenation schedules have quite nice convergence properties and are useful in applying to a real software in operation without specifying the underlying failure time distribution. In fact, the measurement-based approach in the literature [7] to perform the effective software rejuvenation requires much effort to measure the physical characteristics of the system. Also, the modeling-based approaches studied in the literature 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, and can guarantee the real optimal software rejuvenation schedule if the number of failure time data becomes large. Such an on-line estimation algorithm should be applied to complex software systems, such as the transaction-based software systems.

References

[1] Y. Huang, C. Kintala, N. Kolettis and N. D. Funton, "Software Rejuvenation: Analysis, Module and Applications", *Proc.* 25th *IEEE Int'l Symp. on Fault Tolerant Computing*, IEEE Computer Society Press, Los Alamitos, CA, 1995, pp. 381–390.

[2] A. Avritzer and E. J. Weyuker, "Monitoring Smoothly Degrading Systems for Increased Dependability", *Empirical Software Engineering*, **2**, 1987, pp. 59–77.

[3] E. Marshall, "Fatal Error: How Patriot Overlooked a Scud", *Science*, **3**, 1992, p. 1347.

[4] J. Gray and D. P. Siewiorek, "High-Availability Computer Systems", *IEEE Computer*, **9**, 1991, pp. 39–48.

[5] B. O. A. Grey, "Making SDI Software Reliable Through Fault-Tolerant Techniques", *Defense Electronics*, **8**, 1987, pp. 77–80.

[6] A. T. Tai, L. Alkalai and S. N. Chau, "On-Board Preventive Maintenance for Long-Life Deep-Space Missions: A Model-Based Analysis", *Proc.* 3rd *Annual IEEE Int'l Computer Performance & Dependability Symposium*, IEEE Computer Society Press, Los Alamitos, CA, 1998, pp. 196–205

[7] K. S. Trivedi, K. Vaidyanathan and K. Goševa-Postojanova, "Modeling and Analysis of Software Aging and Rejuvenation", *Proc.* 33rd *Annual Simulation Symp.*, IEEE Computer Society Press, Los Alamitos, CA, 2000, pp. 270–279.

[8] S. Garg, A. Puliafito, M. Telek and K. S. Trivedi, "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, CA, 1995, pp. 24–27.

[9] S. Garg, Y. Huang, C. Kintala and K. S. Trivedi, "Time and Load Based Software Rejuvenation: Policy, Evaluation and Optimality", *Proc.* 1st *Fault-Tolerant Symp.*, 1995, pp. 22–25.

[10] S. Pfening, S. Garg, A. Puliafito, M. Telek and K. S. Trivedi, "Optimal Rejuvenation for Toleranting Soft Failure", *Performance Evaluation*, **27/28**, 1996, pp. 491–506.

[11] S. Garg, S. Pfening, A. Puliafito, M. Telek and K. S. Trivedi, "Analysis of Preventive Maintenance in Transactions Based Software Systems", *IEEE Trans. Computers*, **47**, 1998, pp. 96–107.

[12] H. Okamura, A. Fujimoto, T. Dohi, S. Osaki and K. S. Trivedi, "The Optimal Preventive Maintenance Policy for a Software System with Multi Server Station", *Proc.* 6th *ISSAT Int'l Conf. Reliability and Quality in Design*, 2000, pp. 275–279.

[13] R. E. Barlow and R. Campo, "Total Time on Test Processes and Applications to Failure Data Analysis", *Reliability and Fault Tree Analysis* (R. E. Barlow, J. Fussell and N. D. Singpurwalla, eds.), SIAM, Philadelphia, 1975, pp. 451–481.