Transient Cost Analysis of Non-Markovian Software Systems with Rejuvenation

TADASHI DOHI*, HIROAKI SUZUKI* and SHUNJI OSAKI**
*Department of Information Engineering, Hiroshima University, 1-4-1 Kagamiyama, Higashi-Hiroshima 739-8527, Japan
**Department of Information and Telecommunication Engineering, Nanzan University, 27 Seirei-cho, Seto 480-0863, Japan, shunji@nanzan-u.ac.jp
(Received on March 31, 2006)

Abstract: In this paper, we perform the transient analysis of software cost models with periodic/non-periodic rejuvenation. We derive the Laplace-Stieltjes transforms of the ergodic probabilities for respective semi-Markov and Markov regenerative process models, and evaluate numerically the expected cumulative costs experienced by an arbitrary time and its time average by using the Laplace inversion technique, where an improved version of the classical Dubner and Abate's algorithm is used. Numerical examples suggest that the optimal software rejuvenation policy minimizing the expected cumulative cost shows quite different aspects from the steady-state solution which minimizes the long-run average cost.

Key Words: software aging, software rejuvenation, transient analysis, software cost models, Laplace inversion formula, non-Markovian models

1. Introduction

Present day applications impose stringent requirements in terms of software reliability and availability since in many cases the consequence 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. Regardless of development and testing effort, complex software still contains residual faults. When a software application executes continuously for long periods of time, some of the faults cause software to age due to the error conditions that accumulate with time and/or load. Software aging will affect the performance of the application and eventually cause it to fail [1]. Huang et al. [13] report this phenomenon in a telecommunication billing application where over time the application experiences a crash or a hang failure. Avritzer and Weyuker [2] discuss the 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 xim [3].

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 source code is not always available. Furthermore, it is impossible to fully test and verify if a piece of software is fault-free.

1 Communicating author’s email: dohi@rel.hiroshima-u.ac.jp
Testing software becomes harder if it is complex, and when the testing time is reduced due to tighter release time requirements. Common experience suggests that the most software failures are transient in nature [12]. Since transient failures will disappear if the operation is retried later in slightly different context, it is difficult to characterize their root origin. The time to find and deploy a fix to such faults can sometimes be intolerably long. 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.

A complementary approach to handle transient software failures, called software rejuvenation, is becoming popular [13]. Software rejuvenation is a preventive and proactive solution that is particularly useful for counteracting the phenomenon of software aging. It involves stopping the running software occasionally, cleaning its internal state and restarting it. Cleaning the internal state of software might involve garbage collection, flushing operating system kernel tables, reinitializing internal data structures, etc. An extreme, but well known example of rejuvenation which has been around as long as computers themselves is a hardware reboot. Apart from being used in an ad-hoc manner by almost all computer users, software rejuvenation has been used in high availability and mission critical systems. Although the fault in the program still remains, performing rejuvenation occasionally or periodically prevents failures due to that fault.

Software 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 under any criterion of optimality.

Considerable research has been carried out for the phenomenon of software aging and software rejuvenation. Huang et al. [13] introduce a continuous time Markov chain (CTMC) to model the transition behavior of the software system with rejuvenation. They consider a 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 system failure return the software system to the robust state. Garg et al. [11] introduce the idea of periodic rejuvenation (deterministic interval between successive rejuvenations) into the Huang et al.'s model [13] and represent the stochastic behavior by using a Markov regenerative stochastic Petri net. Dohi et al. [5] and Suzuki et al. [17] extend the seminal two-step software degradation models in Huang et al. [13] and Garg et al. [11], respectively, by using semi-Markov processes (SMPs) and Markov regenerative processes (MRGPs), and develop statistical estimation algorithms of the optimal software rejuvenation schedules from the complete sample of system failure data.

Later, these stochastic models are rather extended from a variety of points of view. Dohi et al. [6] and Danjou et al. [4] discuss the same models under the expected discounted cost criteria over an infinite time horizon. Dohi et al. [7] and Iwamoto et al. [14], [15] develop discrete software cost models with rejuvenation, and provide the similar non-parametric estimation procedures to the continuous models. Dohi et al. [8] introduce the different concept of average cost rate from the well-known long-run average cost in the software rejuvenation scheduling problem. Recently, Rinsaka and Dohi [16] analyze a fault-tolerant software system with redundant component and rejuvenation protocol, and evaluate its reliability and steady-state system availability. Xie et al. [18] propose the two-level software rejuvenation policy which
consists of the service-level rejuvenation and the box-level rejuvenation, and generalize the seminal works. For extensive surveys on this topic, the reader is referred to [19].

This paper is a continuation of the references [5] and [17], where the long-run average cost in the steady state is used as a criterion of optimality. Here we perform the transient analysis of software cost models with periodic/non-periodic rejuvenation based on the SMP [5] and MRGP [17]. More precisely, we derive the Laplace-Stieltjes transforms of the ergodic probabilities for the SMP and MRGP, and evaluate numerically the expected cumulative costs experienced by an arbitrary time and its time average by using the Laplace inversion technique, where an improved version of the classical Dubner and Abate’s algorithm [9], referred to as Durbin Method [10], is used. Numerical examples suggest that the optimal software rejuvenation policy minimizing the expected cumulative cost shows quite different aspects from the steady-state solution which minimizes the long-run average cost.

2. Two-Stage Failure Models

A. Model 1: Periodic Rejuvenation Model

Following Suzuki et al. [17], we consider the similar but somewhat different MRGP model with periodic software rejuvenation from Garg et al. [11]. Define the following five states:

State 0: highly robust state (normal operation state)
State 1: failure probable state
State 2: software rejuvenation state from failure probable state
State 3: failure state
State 4: software rejuvenation state from highly robust state.

Suppose that the software system starts for operation at time \( t = 0 \) from the highly robust state (State 0). Let \( Z_0 \) be the random time to reach the failure-probable state (State 1) from the highly robust state. Let \( \Pr\{Z_0 \leq t\} = F_0(t) \) with finite mean \( \mu_0 \) (\( > 0 \)). Just after the state becomes the failure-probable state, a system failure may occur with a positive probability. Let \( X \) be the time to failure from the failure-probable state having the probability distribution function \( \Pr\{X \leq t\} = F_f(t) \) with finite mean \( \mu_f \) (\( > 0 \)). If the system failure occurs before triggering a software rejuvenation, then the recovery operation from the system failure starts immediately. Then the time to complete the recovery operation, \( Y \), is also the positive random variable having the probability distribution function \( \Pr\{Y \leq t\} = F_a(t) \) with finite mean \( \mu_a \) (\( > 0 \)).

On the other hand, consider the case where the software rejuvenation is triggered before occurring the system failure, where two cases are further possible. First, the software rejuvenation may be triggered when the software system is still in the highly robust state. At the moment, let \( F_r(t) \) and \( F_f(t) \) be the probability distribution functions of the time to invoke software rejuvenation, \( T \), without failures and the time to complete software rejuvenation, \( Y \), having finite means \( t_0 \) (\( \geq 0 \)) and \( \mu_a \) (\( > 0 \)), respectively. After completing the software rejuvenation, the software system becomes as good as new, and the software age is initiated at the beginning of the next highly robust state. However, it should be noted that after completing the recovery operation the software system does not become as good as new yet, because the system has to be rejuvenated after the recovery operation. In [17], the authors distinguish a software recovery operation and the software rejuvenation, so that an additional rejuvenation may be needed after the software recovery from the system failure. For example, restarting the system after recovery might require some cleanup and resuming the process execution at the check point state. We call the above model Model 1 in this paper. Figure 1 illustrates the transition diagram for Model 1, where the states denoted by circles (0, 2, 3, 4) and square (1) are regeneration points and a non-regeneration point, respectively, in the MRGP.
B. Model 2: Non-Periodic Rejuvenation Model

The stochastic models with non-periodic rejuvenation are considered by Huang et al. [13] and Dohi et al. [5]. Define the following four states:

State 0: highly robust 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. Just after the state becomes the failure-probable state, a system failure may occur with a positive probability. It is assumed that the random variable $Z_0$ defined previously is observable during the system operation, i.e., we can know the posterior timing when the transition to the failure-probable state occurs. If the system failure occurs before triggering software rejuvenation, then the recovery operation is started immediately at that time and is completed after the random time $Y$ elapses. Otherwise, the software rejuvenation is performed after the time $T$ elapses in the failure-probable state. After completing 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. But, in a fashion similar to Model 1, the software rejuvenation is performed just after the completion of recovery as well as just after the failure probable state is entered, i.e., $\min\{Z_0+X+Y, Z_0+T\}$. Throughout both the models with periodic/non-periodic rejuvenation, we define the time interval from the beginning of the system operation (State 0) to the next one (State 0) as one cycle, and the same cycle repeats itself again and again over an infinite time span. The transition diagram for the model, called Model 2, is shown in Fig. 2.

3. Behavioral Analysis

First we consider Model 1. For the MRGP, define the transition probability $Q_{ij}(t)$ ($i,j = 0, ..., 3$ or 4, $i \neq j$), so that $Q_{ij}(t)$ is the probability of starting from State $i$ and making a transition to State $j$ by time $t$. Let $q_{ij}(s) = \int_0^\infty \exp(-st) dQ_{ij}(t)$ be the Laplace Stieltjes transform (LST) of the transition probability. Further, define the probability that with the initial state $i$ the process
makes a transition to a non-regeneration state \( k (k \neq i, k \neq j) \) and finally moves State \( j \) by time \( t \) and its LST by \( Q_y^{(k)}(t) \) and \( q_y^{(k)} \), respectively.

![Fig. 2: Transition Diagram of Model 2](image)

For Model 1, it is straightforward to obtain

\[
\begin{align*}
q_{01}(s) &= \int_0^{\infty} \exp\{-st\} F_r(t) dF_0(t), \quad (1) \\
q_{02}^{(1)}(s) &= \int_0^{\infty} \exp\{-st\}[F_0 * F_r(t)] dF_c(t), \quad (2) \\
q_{03}^{(1)}(s) &= \int_0^{\infty} \exp\{-st\} F_r(t) d[F_0 * F_r(t)], \quad (3) \\
q_{04}(s) &= \int_0^{\infty} \exp\{-st\} F_0(t) dF_c(t), \quad (4) \\
q_{20}(s) &= \int_0^{\infty} \exp\{-st\} dF_c(t), \quad (5) \\
q_{22}(s) &= \int_0^{\infty} \exp\{-st\} dF_c(t), \quad (6) \\
q_{40}(s) &= \int_0^{\infty} \exp\{-st\} dF_c(t), \quad (7)
\end{align*}
\]

where in general \( \psi(\cdot) = 1 - \varphi(\cdot) \) and \( \ast \) denotes the Stieltjes convolution, i.e.

\[
A \ast B(t) = \int_0^t A(t - x) dB(x)
\]

for two continuous functions \( A(t) \) and \( B(t) \) with positive support. Suppose that the time to rejuvenate the software is a constant \( t_0 \). Then, the function \( F_r(t) \) is given by

\[
F_r(t) = U(t - t_0) - \begin{cases} 
1 : (t \geq t_0) \\
0 : (t < t_0),
\end{cases}
\]

where \( U(\cdot) \) is the unit step function. We call \( t_0 (\geq 0) \) the software rejuvenation schedule or simply the software rejuvenation policy in this paper. Define the recurrence time distribution from State 0 to State 0 by \( H_{00}(t) \). Then the LST of the recurrence time distribution is given by

\[
h_{00}(s) = \int_0^{\infty} \exp\{-st\} dH_{00}(t) = q_{02}^{(1)}(s)q_{20}(s) + q_{03}^{(1)}(s)q_{32}(s)q_{20}(s) + q_{04}(s)q_{40}(s).
\]

Of our concern is the derivation of the transient probability to stay in State \( j (j = 0, 1, \ldots, 4) \) at an arbitrary time \( t (\geq 0) \), provided that the initial state at time \( t = 0 \) is State 0. Define the
transient probability from State 0 to State j (j = 0, ..., 4) and its LST by $P_0(t)$ and $p_0(s) = \int_0^\infty \exp\{-st\}dP_0(t)$, respectively. After some manipulations, we have

$$p_{00}(s) = \frac{1 - q_{01}(s) - q_{03}(s)}{h_{00}(s)},$$  

(11)

$$p_{01}(s) = q_{01}(s) - \frac{q_{10}^{(1)}(s) - q_{13}^{(1)}(s)}{h_{00}(s)},$$  

(12)

$$p_{02}(s) = \frac{(q_{02}^{(1)}(s) + q_{13}^{(1)}(s)q_{20}(s))h_{20}(s)}{h_{00}(s)},$$  

(13)

$$p_{03}(s) = \frac{q_{03}(s)q_{32}(s)}{h_{00}(s)},$$  

(14)

$$p_{04}(s) = \frac{q_{04}(s)q_{40}(s)}{h_{00}(s)}.$$  

(15)

Thus, we can obtain the ergodic probabilities $P_{0i}(t_0) = \lim_{t_0 \to \infty} P_{0j}(s), (j = 0, 1, ..., 4)$ for Model 1.

Next, consider Model 2. In a fashion similar to Model 1, define the LSTs of the transition probabilities for Model 2 by

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

(16)

$$q_{12}(s) = \int_0^\infty \exp\{-st\}F_1(t)dF_2(t),$$  

(17)

$$q_{13}(s) = \int_0^\infty \exp\{-st\}F_1(t)dF_3(t),$$  

(18)

$$q_{23}(s) = \int_0^\infty \exp\{-st\}dF_2(t),$$  

(19)

$$q_{30}(s) = \int_0^\infty \exp\{-st\}dF_3(t).$$  

(20)

Also, the LST of the recurrent time distribution is

$$h_{00}(s) = q_{01}(s)q_{12}(s)q_{23}(s)q_{30}(s) + q_{01}(s)q_{13}(s)q_{30}(s).$$  

(21)

Then, we have

$$p_{00}(s) = \frac{q_{01}(s)}{h_{00}(s)},$$  

(22)

$$p_{01}(s) = \frac{q_{01}(s)(q_{12}(s) - q_{13}(s))}{h_{00}(s)},$$  

(23)

$$p_{02}(s) = \frac{q_{01}(s)q_{12}(s)q_{23}(s)}{h_{00}(s)},$$  

(24)

$$p_{03}(s) = \frac{(q_{01}(s)q_{12}(s)q_{23}(s)q_{30}(s) + q_{01}(s)q_{13}(s)q_{30}(s))}{h_{00}(s)}.$$  

(25)

It is easily possible to obtain the ergodic probabilities $P_{0i}(t_0) = \lim_{t_0 \to \infty} P_{0j}(s), (j = 0, 1, ..., 3)$ for Model 2, by taking the limitation.
4. Optimal Software Rejuvenation Policies

A. Cost Analysis

Define the following cost components:

\( c_s (\geq 0) \): recovery cost per unit time
\( c_p (\geq 0) \): rejuvenation cost per unit time.

For Model 1, the long-run average cost in the steady state [5] is given by

\[
C_1(t_0, t) = c_s P_{03}(t_0) + c_p \{P_{04}(t_0) + P_{02}(t_0)\}.
\]

On the other hand, we obtain

\[
C_2(t_0, t) = c_s P_{02}(t_0) + c_p P_{03}(t_0, t),
\]

as the long-run average cost in Model 2. We are now interested in the derivation of the expected total costs up to an arbitrary time \( t \), \( C_j(t_0, t) \) (\( j = 1, 2 \)) with the pre-specified rejuvenation schedule \( t_0 (0 \leq t_0 < \infty) \). For Model \( j \) (\( j = 1, 2 \)), the instantaneous expected costs at time \( t \) are given by

\[
C_{i,j}(t_0, t) = c_s P_{03}(t_0, t) + c_p \{P_{04}(t_0) + P_{02}(t_0, t)\} \quad (i = 0, 1, \ldots, 3 \text{ or } 4).
\]

From the preliminary above, the expected total costs up to an arbitrary time \( t \), \( C_j(t_0, t) \) (\( j = 1, 2 \)), are given by

\[
C_1(t_0, t) = \int_0^t \left\{ c_s P_{03}(t_0, x) + c_p \{P_{04}(t_0) + P_{02}(t_0, x)\} \right\} dx,
\]

\[
C_2(t_0, t) = \int_0^t \left\{ c_s P_{02}(t_0, x) + c_p P_{03}(t_0, x) \right\} dx
\]

for Model 1 and Model 2, respectively. It is evident that

\[
\lim_{t \to \infty} \frac{C_1(t_0, t)}{t} = c_s P_{03}(t_0) + c_p \{P_{04}(t_0) + P_{02}(t_0)\},
\]

\[
\lim_{t \to \infty} \frac{C_2(t_0, t)}{t} = c_s P_{02}(t_0) + c_p P_{03}(t_0).
\]

Then the problem is to obtain the optimal software rejuvenation policy \( t_0^* \) which minimizes the expected total cost up to an arbitrary time \( t \), \( C_j(t_0, t_0^*) \), or equivalently, its time average \( C_j(t_0, t_0^*)/t \) for Model \( j \) (\( j = 1, 2 \)). Since it is hard to get the explicit forms of \( C_j(t_0, t_0^*) \), we numerically evaluate and minimize the expected total cost up to time \( t \).

B. Laplace Inversion Technique

The essential problem is to invert the LST effectively and calculate

\[
\int_0^t P_{i,j}(t_0, x) dx \quad (i = 0, 1, \ldots, 3 \text{ or } 4).
\]

Here we apply an improved version of the classical Dubner and Abate's algorithm [9], referred to as Durbin Method [10], although it is the well-known Laplace inversion transform technique (not Laplace Stieljes inversion transform technique). By inverting \( p_i(s)/s^2 \) (\( i = 0, 1, \ldots, 3 \) or 4) with the Durbin method, the cumulative (transient) probability \( \int_0^t P_{i,j}(t_0, x) dx \) can be evaluated. The main reason to use the Durbin method is that its Mathematica package (free software) is available on [20]. More precisely, for an arbitrary function \( \psi(t) \) and its first derivative \( \psi'(t) \), the Laplace transform of \( \psi(t) \), \( \varphi(s) = \int_0^\infty \exp(-st)\psi(t) dt \), satisfies the following formula:
\[ \psi'(t) + \text{ERROR}(a, t, S) = \frac{e^{\alpha t}}{S} \left[ \frac{1}{2} \text{Re}\{\phi(a)\} + \sum_{k=1}^{\infty} \text{Re}\{\phi(a + \frac{ik\pi}{S})\} \right] \times \cos \frac{k\pi t}{S} - \sum_{k=0}^{\infty} \text{Im}\{\phi(a + \frac{ik\pi}{S})\} \sin \frac{k\pi t}{S} \right]. \] (31)

where \( \text{ERROR}(a, t, S) \) is the error function defined in the literature [10]. \( S \) and \( a \) are both arbitrary design parameters. In the Dubner and Abate’s algorithm [9], it is known that the effective time range of the Laplace inversion transform is given by \( 0 < t < T/2 \), but that the Durbin method [10] can provide more wide range \( 0 < t < 2T \). Also, the error function in the Dubner and Abate’s algorithm increases exponentially in \( t \). However, the error function in the improved version is independent of \( t \), and has an upper bound with respect to \( t \). To minimize the expected cumulative cost numerically, we apply a heuristic binary search technique to find the optimal software rejuvenation schedule, instead of the numerical differentiation method.

5. Numerical Examples

A. Parameter Set

We derive the optimal software rejuvenation schedule, \( t^*_0 \), minimizing the expected cumulative cost at time \( t \). Suppose that the time to shift from the highly robust state to the failure probable state, \( Z_0 \), and the system failure time from the failure probable state, \( X \), obey the following gamma distributions:

\[ f_0(t) = \frac{\lambda i^{k_1 - 1}\exp[-\lambda_1 t]}{\Gamma(k_1)}, \]

\[ f_j(t) = \frac{\lambda j^{k_2 - 1}\exp[-\lambda_2 t]}{\Gamma(k_2)} \]  

with parameters \( \lambda_i \) and \( k_i (i = 1, 2) \), where \( \Gamma(\cdot) \) is the standard gamma function. It is easily shown that the mean values of \( Z_0 \) and \( X \) are given by \( k_i/\lambda_i (i = 1, 2) \). Here, we assume that \( \lambda_1 = 1, k_2 = 2, \lambda_4 = 1/240, \lambda_3 = 1/1080 \), that is, \( F_0(t) \) and \( F_j(t) \) are given by the simple gamma distributions with order two. Then, we have \( \mu_0 = k_1/\lambda_1 = 240 \) and \( \mu_i = k_j/\lambda_2 = 2160 \). This means that the expected total time to system failure is given by 2400 (hr). For the other probability distributions, \( F_a(t) \) and \( F_c(t) \), we assume the exponential distributions with parameter \( \mu_c = 1/6 \) and \( \mu_a = 1/3 \), respectively.

We perform a preliminary experiment to determine the design parameters \( S \) and \( a \) and determine \( S = 4t \) and \( a = 2/t \) following [9]. Also, we set the truncation level for calculating an infinite series in Eq. (31) as \( k = 2000 \). Then, the Durbin method can be rewritten as

\[ \psi'(t) = \frac{e^{\alpha t}}{4t} \left[ \frac{1}{2} \text{Re}\{\phi(\frac{2}{t})\} + \sum_{k=1}^{2000} \text{Re}\{e^{ik\pi/4}\phi(\frac{2}{t} + \frac{4k\pi}{4t})\} \right]. \] (34)

B. Asymptotic Behavior

We examine the asymptotic behavior of the optimal software rejuvenation schedule which minimizes the expected cumulative total cost \( C_i(t^*_0, t) \) for Model \( i \). In Fig. 3, we plot the transient behavior of the optimal software rejuvenation schedule. The optimal solution \( t^*_0 \) fluctuates first in the earlier phase, and converges to the steady-state solution (560.50 or 266.97), which minimizes \( \lim_{t \to \infty} C_i(t^*_0, t) / t \), as the operation time \( t \) monotonically increases. Figures 4 and 5 illustrate the time-dependent behavior of the cumulative expected costs for Model 1 and Model 2, respectively. As seen from the figures, both expected cumulative cost behave like a
Transient Cost Analysis of Non-Markovian Software Systems with Rejuvenation

linear function of $t$ and increase monotonically. However, by checking the time-average behavior, the expected cost per unit time at time $t$, $C_i(t_0, t)/t$ ($i = 1, 2$) approached to a constant level from the bottom. From these graphical results, it is seen that the software rejuvenation schedules based on the long-run average cost in the steady state [5], [17] tend to underestimate or overestimate the real solutions for the relatively shorter operation time. Also, the long-run average cost always overestimates the transient expected cost rate $C_i(t_0^*, t)/t$ ($i = 1, 2$).

Fig. 3: Transient Behavior of the Optimal Rejuvenation Schedule, $t_0^*$: $\theta = 2437.00$, $\beta = 2.00$, $\mu_0 = 240.00$, $\mu_a = 0.33$, $\mu_c = 0.16$.

Fig. 4: Time-Dependent Behavior of Expected Cumulative Cost and its Time-Average with the Optimal Rejuvenation Schedule $T_0^*$ (Model 1): $\Theta = 2437.00$, $B = 2.00$, $\mu_0 = 240.00$, $\mu_a = 0.33$, $\mu_c = 0.16$.

Fig. 5: Time-Dependent Behavior of Expected Cumulative Cost and its Time-Average with the Optimal Rejuvenation Schedule $T_0^*$ (Model 2): $\Theta = 2437.00$, $B = 2.00$, $\mu_0 = 240.00$, $\mu_a = 0.33$, $\mu_c = 0.16$.
C. Sensitivity Analysis

To evaluate the difference between the ergodic and transient solutions quantitatively, we carry out the sensitivity analysis of model parameters. Table 1 presents the dependence of the optimal software rejuvenation schedule on the parameter ratio \(c_s/c_p\). From this result, it is found that as the rejuvenation cost relatively decreases to the recovery cost, the resulting software rejuvenation schedule shortens and the corresponding expected cost decreases. It is noted that the optimal rejuvenation schedule for Model 1 is rather bigger than that for Model 2, because two rejuvenation times have the different meanings from each other. That is, the software rejuvenation schedule for Model 1 is measured from the initial point for operation but Model 2 does the software rejuvenation from the beginning of the failure probable state. Also, it is evident that the expected cost for Model 2 is always cheaper than that for Model 1, since one can identify the failure probable state in Model 2. Of course this assumption is rather strong and questionable in practice. On the other hand, even if one can not know whether the current state is the failure probable state or not, it would be possible to trigger the software rejuvenation in Model 1.

Table 1: Dependence of the Optimal Software Rejuvenation Policy on the Parameter Ratio

<table>
<thead>
<tr>
<th>(c_s/c_p)</th>
<th>(t_0^*)</th>
<th>(C_2(t_0^*, t))</th>
<th>(t_0^*)</th>
<th>(C_1(t_0^*, t))</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>0.7889</td>
<td>880.80</td>
<td>0.8071</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>0.6206</td>
<td>851.52</td>
<td>0.6575</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>0.5157</td>
<td>640.78</td>
<td>0.5575</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>0.4430</td>
<td>560.86</td>
<td>0.4925</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>0.3892</td>
<td>500.82</td>
<td>0.4434</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>0.3476</td>
<td>459.05</td>
<td>0.4073</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>0.3143</td>
<td>457.16</td>
<td>0.3768</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>0.2871</td>
<td>419.08</td>
<td>0.3517</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>0.2643</td>
<td>385.22</td>
<td>0.3305</td>
<td></td>
</tr>
</tbody>
</table>

Acknowledgments

The authors were partially supported by the Ministry Education, Science, Sports and Culture, Grant-in-Aid for Exploratory Research; Grant No.15651076 (2003-2005) and Scientific Research (B); Grant No.16310116 (2004-2006), and the Nanzan University Pache Research Subsidy for 2005.

References


[20]. http://pumpjack.tamu.edu/~valko/Nil/