数理解析研究所講究録 1457 巻 2005 年 200-207

Behavioral Analysis of a Fault-Tolerant Software System with Rejuvenation

林坂弘一郎, 土肥 正
Koichiro Rinsaka and Tadashi Dohi
Department of Information Engineering, Graduate School of Engineering,
Hiroshima University, Japan

Abstract In recent years, considerable attention has been devoted to continuously running software systems whose performance characteristics are smoothly degrading in time. Software aging often affects the performance of a software system and eventually causes it to fail. A novel approach to handle transient software failures due to software aging is called software rejuvenation, which can be regarded as a preventive and proactive solution that is particularly useful for counteracting the aging phenomenon. In this paper, we focus on a high assurance software system with fault-tolerance and preventive rejuvenation, and analyze the stochastic behavior of such a highly critical software system. More precisely, we consider a fault-tolerant software system with two-version redundant structure and random rejuvenation schedule, and evaluate quantitatively a dependability measure like the steady-state system availability based on the familiar Markovian analysis. In numerical examples, we examine the dependence of two system diversity techniques; design and environment diversity techniques, on the system dependability measures.

1. Introduction

Present day applications impose stringent requirements in terms of software dependability 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. In general, the software dependability techniques can be classified into two approaches: design and environment diversity techniques. The former corresponds to the redundant-software architecture such as recovery block, N version programming and N self-check programming [16,23], the latter to diversify the software operating circumstance temporarily. The typical examples of environment diversity technique are progressive retry, rollback roll-forward recovery with checkpointing, restart, hardware reboot, *etc.* In recent years, considerable attention has been devoted to continuously running software systems whose performance characteristics are smoothly degrading in time. That is to say, when software application executes continuously for long periods of time, some of the faults cause software appear to age due to the error conditions that accrue with time and/or load. This phenomenon is called *software aging* and can be observed in many real software systems [1,5,7,21,28].

Huang *et al.* [12] report this phenomenon in the real telecommunication billing application where over time the application experiences a crash or a hang failure. Avritzer and Weyuker [3] discuss aging in a 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 [5]. Perhaps the most vivid example of aging in safety critical systems is the Patriot's software [17], where the accumulated errors led to a failure that resulted in loss of human lives. 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. Our common experience suggests that most software failures are transient in nature [11]. Since transient failures will likely not recur even 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 are often tolerated in the operational phase. Usual strategies to deal with failures in operational phase are reactive in nature; they consist of action taken after the occurrence of the failure.

A novel approach to handle transient software failures is called *software rejuvenation* which can be regarded as 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 a 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. The studies of aging-related failures are based on two approaches: measurement-based and model-based. The measurement-based approach concentrates on the detection and validation of the existence of software aging and estimating its effects on system resources [10,13,24,29,30]. On the other hand, the model-based approach aims at evaluating the effectiveness of software rejuvenation and determining the optimal schedule to perform it.

Huang, *et al.* [12] consider a model-based approach where the degradation is described by a two step process. From the clean state the software system makes a transition into a degraded state from which two actions are possible: rejuvenation with return to the clean state or transition to the complete failure state. They model the four-state process as a continuous-time Markov chain (CTMC) and derive the steady-state availability and the expected cost per unit time in the steady state. Garg *et al.* [8] introduce the idea of periodic rejuvenation into the Huang *et al.* model [12], and represent the system behavior through a Markov regenerative stochastic Petri net. Dohi *et al.* [6], Suzuki *et al.* [25, 26] extend the Huang *et al.* model [12] to semi-Markov models and further develop nonparametric algorithms to estimate the optimal software rejuvenation schedule. Tai *et al.* [27] also discuss the concept of on-board preventive maintenance which is an analogous to software rejuvenation and maximize the mission reliability. Garg *et al.* [9] develop a preventive maintenance model with two kinds of environment diversity techniques and examine the effects of checkpointing and rejuvenation for the expected completion time of a software program. Liu *et al.* [14] and Park and Kim [22] evaluate the cable modem termination system and the active/standby cluster systems with rejuvenation, respectively. Aung [2] and Liu *et al.* [15] treat the determination problem of software rejuvenation schedule from the view point of software survivability. Bao *et al.* [4] develop an adaptive software rejuvenation scheme based on the statistical observation of system failure time.

In this paper, we focus on a high assurance software system with fault-tolerance and preventive rejuvenation, and analyze the stochastic behavior of such a highly critical software system. More precisely, we consider a fault-tolerant software system with two-version redundant structure and random rejuvenation schedule, and evaluate quantitatively a dependability measure like the steady-state system availability based on the familiar Markovian analysis. Two version software system is perhaps the most popular fault tolerant software system with redundancy. In the hardware fault tolerance (*e.g.* see Osaki [19]), two independent components are assumed in the computer architecture. However, it is almost impossible to develop statistically independent software programs. Hence, when two version software system is modeled mathematically, the correlation on the failure property between two software systems has to be represented by bivariate probability distribution. In this paper we develop a CTMC model with redundancy and rejuvenation, by taking account of the failure correlation of two software systems. Then, the bivariate exponential distribution in the sense of Marshall and Olkin [18] is introduced to represent the correlation between two software systems. Also, in order to model the deterioration process due to software aging, we apply the two step failure model similar to Huang *et al.* [12]. This point should be distinguished from the existing result in the hardware fault tolerant literature (*e.g.* Osaki [20]), though this paper is a continuation of earlier work [12].

The rest of this paper is organized as follows. In Section 2, we describe a Markov software rejuvenation model discussed by Huang *et al.* [12] and summarize their results on the steady-state system availability. In Section 3, we model a two version software system with rejuvenation and derive analytically the steady-state system availability. Section 4 is devoted to numerical examples, where we examine the dependence of two system diversity techniques; design and environment diversity techniques, on the system dependability measures. Finally, the paper is concluded with some remarks.

2. Single Version Software System with Rejuvenation

Suppose that a single version software system is started for operation at time $t = 0$ with the highly robust state (normal operation state). The system can smoothly degrade in time. As assumed in the existing literature [6, 8, 12, 25, 26], we focus on the two step failure process to model the telecommunication billing application in AT & T, *i.e.*, the highly robust state changes to the failure probable state at random timing. Just after the state becomes the failure probable state, a system failure may occur with a positive probability. If the system failure occurs before triggering a software rejuvenation, then the recovery operation is started immediately at that time and is completed with a random recovery overhead. Otherwise, the

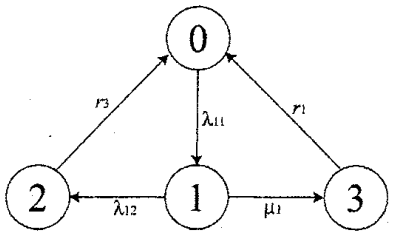

Figure 1: Markovian transition diagram for a single version software system with rejuvenation.

software rejuvenation is triggered and the software system becomes as good as new with the rejuvenation overhead. Then, the software age is initialized to zero at the beginning of the next highly robust state. We define the time interval from the beginning of the system operation to the completion of recovery operation or software rejuvenation as one cycle, and assume that the same cycle is repeated again and again.

Huang *et al.* [12] introduce the simple software rejuvenation model based on the CTMC. Define the following four states:

State 0: highly robust state (normal operation state)

State 1: failure probable state

State 2: system failure state

State 3: software rejuvenation state.

Figure 1 depicts the Markovian transition diagram of Huang *et al.* model [12], where λ_{11} (>0), λ_{12} (>0), μ_1 (>0), r_1 (>0) and r_3 (>0) denote the transition rates from the highly robust state to the failure probable state, the system failure rate from the failure probable state, the transition rate to trigger the software rejuvenation from the failure probable state, the recovery rate from the system failure state and the recovery rate from the software rejuvenation state, respectively. In this model setting, the opportunity to trigger the software rejuvenation arrives at the system according to the homogeneous Poisson process with rate μ_1. This assumption does not seem to be restrictive, because the preventive rejuvenation is not always possible at pre-scheduled time in continuously running software systems.

Let $\{X(t) = i, t \geq 0\}$, $(i = 0, 1, 2, 3)$ be the system state at time t with the transition probability $Q_{0,j}(t) = \Pr\{X(t) = j | X(0) = 0\}$ $(t \geq 0,\ j = 0, 1, 2, 3)$. From an elementary probabilistic argument, the Kolmogorov's differential equations which the transition probabilities have to satisfy are given by

$$\frac{dQ_{0,0}(t)}{dt} = -\lambda_{11}Q_{0,0}(t) + r_3Q_{0,2}(t) + r_1Q_{0,3}(t), \quad (1)$$

$$\frac{dQ_{0,1}(t)}{dt} = -(\lambda_{12} + \mu_1)Q_{0,1}(t) + \lambda_{11}Q_{0,0}(t), \quad (2)$$

$$\frac{dQ_{0,2}(t)}{dt} = -r_3Q_{0,2}(t) + \lambda_{12}Q_{0,1}(t), \quad (3)$$

$$\frac{dQ_{0,3}(t)}{dt} = -r_1Q_{0,3}(t) + \mu_1Q_{0,1}(t) \quad (4)$$

with the initial conditions:

$$Q_{0,0}(0) = 1, \quad Q_{0,1}(0) = Q_{0,2}(0) = Q_{0,3}(0) = 0. \quad (5)$$

Suppose that the limiting transition probability p_j $(j = 0, 1, 2, 3)$ exists, *i.e.*

$$p_j = \lim_{t\to\infty} Q_{0,j}(t), \quad j = 0, 1, 2, 3. \quad (6)$$

By taking the limitation, since the Kolmogorov's differential equations in Eqs. (1)-(4) are reduced to the algebraic equations:

$$-\lambda_{11}p_0 + r_3p_2 + r_1p_3 = 0, \quad (7)$$

$$-(\lambda_{12} + \mu_1)p_1 + \lambda_{11}p_0 = 0, \quad (8)$$

$$-r_3p_2 + \lambda_{12}p_1 = 0, \quad (9)$$

$$-r_1p_3 + \mu_1p_1 = 0, \quad (10)$$

$$p_0 + p_1 + p_2 + p_3 = 1, \quad (11)$$

we obtain the steady-state system availability as follows:

$$A = p_0 + p_1 = \frac{\frac{1}{\lambda_{11}} + \frac{1}{\lambda_{12}+\mu_1}}{\frac{1}{\lambda_{11}} + \frac{1}{\lambda_{12}+\mu_1} + \frac{\lambda_{12}}{(\lambda_{12}+\mu_1)r_3} + \frac{\mu_1}{(\lambda_{12}+\mu_1)r_1}}. \quad (12)$$

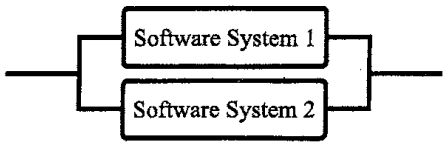

Figure 2: Configuration of two version software system.

3. Fault-Tolerant Software System with Rejuvenation

3.1 Availability Analysis

Next, we consider a fault-tolerant software system with redundancy and preventive rejuvenation. Suppose that two software programs are running in parallel. Figure 2 indicates the configuration of two version software system. In a fashion similar to Huang *et al.* model [12], it is assumed that

each software deteriorates and reaches to the system failure state if no software rejuvenation is triggered after entering to the failure probable state. More specifically, consider a two-unit parallel redundant system with deterioration on a lattice. In the two-unit hot-standby parallel redundant system, each unit has two stage deterioration levels, say failure probable state and system failure state, and the transition from the normal operation (deterioration) level to the deterioration (system failure) level is occurred following an exponential distribution. Let X and Y be the deterioration times for Software System 1 and Software System 2, respectively, and denote the non-negative random variables having the following marginal distribution functions;

$$F_X(x) = 1 - \exp\{-\lambda_1 x\}, \quad x > 0, \ \lambda_1 > 0, \quad (13)$$
$$F_Y(y) = 1 - \exp\{-\lambda_2 y\}, \quad y > 0, \ \lambda_2 > 0. \quad (14)$$

If the failure property of two software systems is statistically independent, then the joint distribution function is given by

$$\begin{aligned} F_{XY}(x,y) &= \Pr\{X \le x, Y \le y\} \\ &= 1 - \{1 - F_X(x)\}\{1 - F_Y(y)\} \\ &= 1 - \exp\{-\lambda_1 x - \lambda_2 y\}. \end{aligned} \quad (15)$$

Since it is impossible to develop the completely independent software systems with same functions, however, the correlation between the failure properties for each software system should be taken into consideration.

In this paper, we use the bivariate exponential distribution in the sense of Marshall and Olkin [18] to represent both deterioration/failure phenomena for two software systems. The main reason to apply the Marshall and Olkin distribution is that it is easy to represent the simultaneous deterioration/failure in the framework and apply it to the CTMC analysis [20]. For the deterioration times X and Y, the deterioration time distribution is given by the following bivariate exponential:

$$\begin{aligned} F(x,y) &= \Pr\{X \le x, Y \le y\} \\ &= 1 - \exp\{-\lambda_1 x - \lambda_2 y - \lambda_3 \max(x,y)\}, \end{aligned} \quad (16)$$

where λ_3 (≥ 0) denotes the correlation parameter. When $\lambda_3 = 0$, then two software systems are independent from each other, the system failure time distribution is given in Eq. (16).

Define the following 15 states:

State 0: Both systems are operating

State 4: Both systems deteriorate

State 8: Both systems are down

State 1 (3): System 1 (2) deteriorates but System 2 (1) is operating

State 5 (7): System 1 (2) fails but System 2 (1) deteriorate

State 6 (2): System 2 (1) fails but System 1 (2) is operating

State 9 (10): System 1 (2) is rejuvenated but System 2 (1) is operating

State 11 (12): System 1 (2) is rejuvenated but System 2 (1) deteriorates

State 13 (14): System 1 (2) is rejuvenated but System 2 (1) fails.

Based on the definition above, the system failure is corresponding to States 8, 13 and 14.

Figure 3 depicts the Markovian transition diagram for fault-tolerant software system with rejuvenation in renewal case, where λ_{11}, λ_{12}, μ_1, r_1 and r_3 (λ_{21}, λ_{22}, μ_2, r_2 and r_4) denote the transition rates from the highly robust state to the failure probable state, the system failure rate from the failure probable state, the transition rate to trigger of software rejuvenation from the failure probable state, the recovery rate from the system failure state and the recovery rate from the software rejuvenation state for Software System 1 (Software System 2), respectively. Further, λ_{31} denotes the transition rate that both software systems deteriorate simultaneously, λ_{32} (λ_{33}) denotes the transition rate that Software System 1 (2) fails and Software System 2 (1) deteriorates simultaneously, and λ_{34} denotes the simultaneous system failure rate from the failure probable state.

Let $\{X(t) = i, t \ge 0\}$ ($i = 0, 1, \cdots, 14$) be the CTMC to represent the stochastic behavior of the two version software system with rejuvenation. In a fashion similar to the Markovian argument in Section 2, we obtain the pointwise system availability:

$$\begin{aligned} A(t) \quad = \quad & Q_{0,0}(t) + Q_{0,1}(t) + Q_{0,2}(t) + Q_{0,3}(t) \\ & + Q_{0,4}(t) + Q_{0,5}(t) + Q_{0,6}(t) \\ & + Q_{0,7}(t) + Q_{0,9}(t) + Q_{0,10}(t) \\ & + Q_{0,11}(t) + Q_{0,12}(t), \end{aligned} \quad (17)$$

where the stationary transition probabilities $Q_{0,j}(t)$, ($j = 0, 1, \cdots, 14$) are the solutions of the Kolmogorov's differential equations in Eqs.(36)-(50) (see Appendix). If there exist the limiting probabilities

$$p_j = \lim_{t \to \infty} Q_{0,j}(t) \quad (j = 0, 1, \cdots, 14), \quad (18)$$

then we derive the following simultaneous (algebraic) equations:

$$-(\lambda_{11} + \lambda_{21} + \lambda_{31})p_0 + r_3 p_2 + r_4 p_6 + r_1 p_9$$

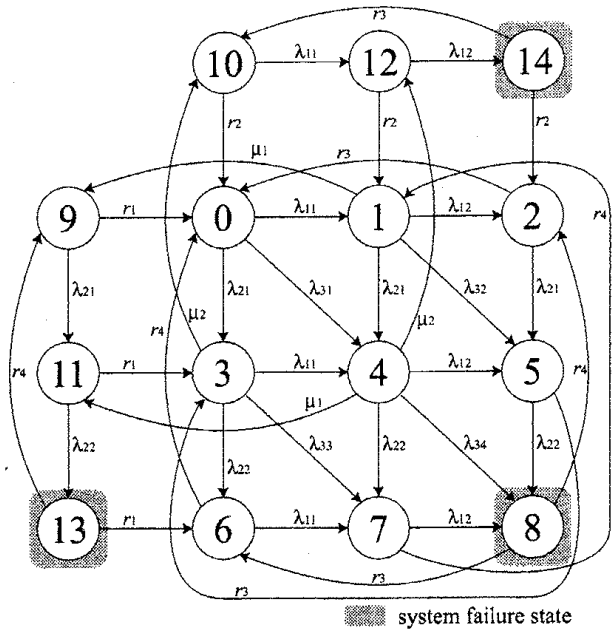

Figure 3: Markovian transition diagram for fault-tolerant software system with rejuvenation.

$$+ r_2 p_{10} = 0, \quad (19)$$

$$-(\lambda_{12} + \lambda_{21} + \lambda_{32} + \mu_1)p_1 + \lambda_{11}p_0 + r_4 p_7 + r_2 p_{12} = 0, \quad (20)$$

$$-(\lambda_{21} + r_3)p_2 + \lambda_{12}p_1 + r_4 p_8 + r_2 p_{14} = 0, \quad (21)$$

$$-(\lambda_{11} + \lambda_{22} + \lambda_{33} + \mu_2)p_3 + \lambda_{21}p_0 + r_3 p_5 + r_1 p_{11} = 0, \quad (22)$$

$$-(\lambda_{12} + \lambda_{22} + \lambda_{34} + \mu_1 + \mu_2)p_4 + \lambda_{31}p_0 + \lambda_{21}p_1 + \lambda_{11}p_3 = 0, \quad (23)$$

$$-(\lambda_{22} + r_3)p_5 + \lambda_{32}p_1 + \lambda_{21}p_2 + \lambda_{12}p_4 = 0, \quad (24)$$

$$-(\lambda_{11} + r_4)p_6 + \lambda_{22}p_3 + r_3 p_8 + r_1 p_{13} = 0, \quad (25)$$

$$-(\lambda_{12} + r_4)p_7 + \lambda_{33}p_3 + \lambda_{22}p_4 + \lambda_{11}p_6 = 0, \quad (26)$$

$$-(r_3 + r_4)p_8 + \lambda_{34}p_4 + \lambda_{22}p_5 + \lambda_{12}p_7 = 0, \quad (27)$$

$$-(\lambda_{21} + r_1)p_9 + \mu_1 p_1 + r_4 p_{13} = 0, \quad (28)$$

$$-(\lambda_{11} + r_2)p_{10} + \mu_2 p_3 + r_3 p_{14} = 0, \quad (29)$$

$$-(\lambda_{22} + r_1)p_{11} + \mu_1 p_4 + \lambda_{21}p_9 = 0, \quad (30)$$

$$-(\lambda_{12} + r_2)p_{12} + \mu_2 p_4 + \lambda_{11}p_{10} = 0, \quad (31)$$

$$-(r_1 + r_4)p_{13} + \lambda_{22}p_{11} = 0, \quad (32)$$

$$-(r_2 + r_3)p_{14} + \lambda_{12}p_{12} = 0, \quad (33)$$

$$\sum_{j=0}^{14} p_j = 1. \quad (34)$$

Finally, by solving the above equations numerically, we can get the steady-state system availability:

$$A = p_0 + p_1 + p_2 + p_3 + p_4 + p_5 + p_6 + p_7 + p_9 + p_{10} + p_{11} + p_{12}. \quad (35)$$

4. Numerical Illustrations

Here, we compare the present model with Huang *et al.* [12] and investigate the effect of redundancy in terms of dependability measures. To simplify the analysis, it is assumed that $\lambda_3 = \lambda_{31} = \lambda_{32} = \lambda_{33} = \lambda_{34}$. Also, we assume the parametric circumstance mentioned in Table 1. Figure 4 illustrates the dependence of software rejuvenation rate μ_1 on the steady-state system availability for Huang *et al.* model [12]. From this figure, for a single version software system, the steady-state system availability can be improved from 0.997 to 0.998 (0.1003%) by adjusting the software rejuvenation rate from $\mu_1 = 0.0458$ to 0.02665. On the other hand, in Figs. 5 and 6, the behavior of the steady-state system availability with varying software rejuvenation rate μ_1 is plotted in two cases: $\lambda_3 = 1/500$ and $\lambda_3 = 0$, where $\lambda_3 = 0$ implies that two software systems are completely independent in terms of system failure occurrence. Comparing Fig.5 with Fig.6, it can be observed that the correlation parameter λ_3 strongly influences to the steady-state system availability, *i.e.* as λ_3 increases much more, two software systems have greater correlation on the system failure from each other and the system availability decreases.

Finally, it is concluded that the software dependability can be controlled with two model parameters based on design and environment diversity techniques. Namely, if the alternative software version is designed from the standpoint of dependability measures, the correlation parameters like λ_3

Table 1: Parameter Values

Parameter	Value
λ_{11}^{-1}	240 hrs.
λ_{12}^{-1}	50 hrs.
λ_{21}^{-1}	240 hrs.
λ_{22}^{-1}	50 hrs.
r_1	6 /hr.
r_2	6 /hr.
r_3	1 /hr.
r_4	1 /hr.

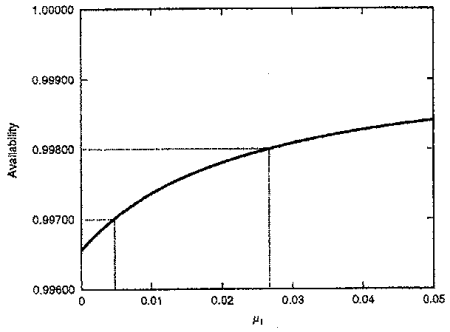

Figure 4: Dependence of software rejuvenation rate on the steady-state system availability for single version software system.

plays a significant role to attain the target dependability level. On the other hand, after a sufficient cost was spent in the design of redundant structure, the environment diversity technique such as software rejuvenation should be carried out and the software rejuvenation rates like μ_1 should be determined to trigger the preventive maintenance of degraded software systems due to aging phenomenon.

5. Conclusion

In this paper we have dealt with a fault-tolerant software system with alternative redundant version and random rejuvenation, and investigated its stochastic behavior. Based on the simple CTMC approach, we have numerically derived the steady-state system availability, and have examine the effect of two diversity techniques; design and environment diversity techniques. In practice, it is well known that much effort and cost are needed in the design phase to develop the effective two version software system with lower correlation. However, we have shown that by combining two diversity techniques effectively, the dependability level can be improved. In the next step, we will consider the case with the deterministic software rejuvenation schedule and extend the present CTMC model

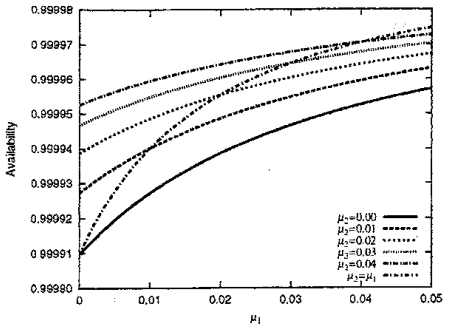

Figure 5: Dependence of software rejuvenation rate on the steady-state system availability for two version software system: $\lambda_3^{-1} = 500$ (hrs).

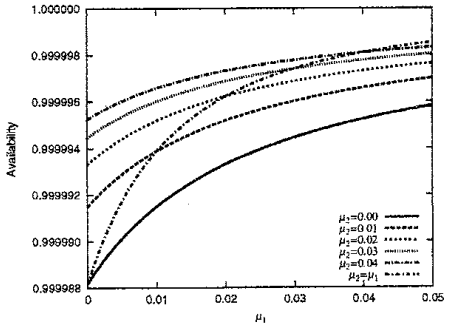

Figure 6: Dependence of software rejuvenation rate on the steady-state system availability for two version software system: $\lambda_3 = 0$.

to the semi-Markov one. Then, the other bivariate family *e.g.* such as bivariate gamma distribution should be applied to model the system failure phenomenon for the fault-tolerant software system under several restrictive assumptions. If two software systems are expected to be statistically independent, it is not so hard to formulate the optimal software rejuvenation scheduling problem. However, as tried in this paper, the modeling of correlated software systems will involve several technical problems to be overcome.

A. Appendix

A.1 Stationary transition probabilities

The Kolmogorov differential equations which the stationary transition probabilities $Q_{0,j}(t)$ $(j = 0, 1, \cdots, 14)$ satisfy are given by

$$\begin{aligned}\frac{dQ_{0,0}(t)}{dt} &= -(\lambda_{11}+\lambda_{21}+\lambda_{31})Q_{0,0}(t) \\ &\quad + r_3 Q_{0,2}(t) + r_4 Q_{0,6}(t) \\ &\quad + r_1 Q_{0,9}(t) + r_2 Q_{0,10}(t), \end{aligned} \quad (36)$$

$$\frac{dQ_{0,1}(t)}{dt} = -(\lambda_{12}+\lambda_{21}+\lambda_{32}+\mu_1)Q_{0,1}(t) + \lambda_{11}Q_{0,0}(t) + r_4Q_{0,7}(t) + r_2Q_{0,12}(t), \quad (37)$$

$$\frac{dQ_{0,2}(t)}{dt} = -(\lambda_{21}+r_3)Q_{0,2}(t) + \lambda_{12}Q_{0,1}(t) + r_4Q_{0,8}(t) + r_2Q_{0,14}(t), \quad (38)$$

$$\frac{dQ_{0,3}(t)}{dt} = -(\lambda_{11}+\lambda_{22}+\lambda_{33}+\mu_2)Q_{0,3}(t) + \lambda_{21}Q_{0,0}(t) + r_3Q_{0,5}(t) + r_1Q_{0,11}(t), \quad (39)$$

$$\frac{dQ_{0,4}(t)}{dt} = -(\lambda_{12}+\lambda_{22}+\lambda_{34}+\mu_1+\mu_2) \times Q_{0,4}(t) + \lambda_{31}Q_{0,0}(t) + \lambda_{21}Q_{0,1}(t) + \lambda_{11}Q_{0,3}(t), \quad (40)$$

$$\frac{dQ_{0,5}(t)}{dt} = -(\lambda_{22}+r_3)Q_{0,5}(t) + \lambda_{32}Q_{0,1}(t) + \lambda_{21}Q_{0,2}(t) + \lambda_{12}Q_{0,4}(t), \quad (41)$$

$$\frac{dQ_{0,6}(t)}{dt} = -(\lambda_{11}+r_4)Q_{0,6}(t) + \lambda_{22}Q_{0,3}(t) + r_3Q_{0,8}(t) + r_1Q_{0,13}(t), \quad (42)$$

$$\frac{dQ_{0,7}(t)}{dt} = -(\lambda_{12}+r_4)Q_{0,7}(t) + \lambda_{33}Q_{0,3}(t) + \lambda_{22}Q_{0,4}(t) + \lambda_{11}Q_{0,6}(t), \quad (43)$$

$$\frac{dQ_{0,8}(t)}{dt} = -(r_3+r_4)Q_{0,8}(t) + \lambda_{34}Q_{0,4}(t) + \lambda_{22}Q_{0,5}(t) + \lambda_{12}Q_{0,7}(t), \quad (44)$$

$$\frac{dQ_{0,9}(t)}{dt} = -(\lambda_{21}+r_1)Q_{0,9}(t) + \mu_1Q_{0,1}(t) + r_4Q_{0,13}(t), \quad (45)$$

$$\frac{dQ_{0,10}(t)}{dt} = -(\lambda_{11}+r_2)Q_{0,10}(t) + \mu_2Q_{0,3}(t) + r_3Q_{0,14}(t), \quad (46)$$

$$\frac{dQ_{0,11}(t)}{dt} = -(\lambda_{22}+r_1)Q_{0,11}(t) + \mu_1Q_{0,4}(t) + \lambda_{21}Q_{0,9}(t), \quad (47)$$

$$\frac{dQ_{0,12}(t)}{dt} = -(\lambda_{12}+r_2)Q_{0,12}(t) + \mu_2Q_{0,4}(t) + \lambda_{11}Q_{0,10}(t), \quad (48)$$

$$\frac{dQ_{0,13}(t)}{dt} = -(r_1+r_4)Q_{0,13}(t) + \lambda_{22}Q_{0,11}(t), \quad (49)$$

$$\frac{dQ_{0,14}(t)}{dt} = -(r_2+r_3)Q_{0,14}(t) + \lambda_{12}Q_{0,12}(t) \quad (50)$$

with the initial conditions:

$$Q_{0,0}(0) = 1, \quad Q_{0,j}(0) = 0 \quad j = 1, 2, \cdots, 14. \quad (51)$$

References

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

[2] Aung, K.-M.-M. (2004), The optimum time to perform software rejuvenation for survivability, *Proc. 7th IASTED Int'l Conf. on Software Eng.*, 292–296.

[3] Avritzer, A. and Weyuker, E. J. (1997), Monitoring smoothly degrading systems for increased dependability, *Empirical Software Eng.*, **2**, 59–77.

[4] Bao, Y., Sun, X. and Trivedi, K. S. (2003), Adaptive software rejuvenation: degradation model and rejuvenation scheme, *Proc. Inte'l Conf. on Dependable Systems and Networks*, 241–248, IEEE CS Press.

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

[6] Dohi, T., Goševa-Popstojanova, K. and Trivedi, K. S. (2001), Estimating software rejuvenation schedule in high assurance systems, *The Computer Journal*, **44**, 473–485.

[7] Dohi, T., Goševa-Popstojanova, K., Vaidyanathan, K., Trivedi, K. S. and Osaki, S. (2003), Software rejuvenation – modeling and applications, *Springer Reliability Engineering Handbook* (H. Pham, ed.), Springer-Verlag, 245–263.

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

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

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

[11] Gray, J. (1986), Why do computers stop and what can be done about it?, *Proc. 5th Int'l Sympo. on Reliab. Distributed Software and Database Systems*, 3–12, IEEE CS Press.

[12] Huang, Y., Kintala, C., Kolettin, N. and Funton, N. D. (1995), Software rejuvenation: analysis, module and applications, *Proc. 25th Int'l Symp. on Fault Tolerant Computing*, 381-390, IEEE CS Press.

[13] Lei, L., Vaidyanathan, K. and Trivedi, K. S. (2002), An approach for estimation of software aging in a web server, In *Proc. 2002 Int'l Symp. on Empirical Software Eng.*, 91–100, IEEE CS Press.

[14] Liu, Y., Ma, Y., Han, J. J., Levendel, H. and Trivedi, K. S. (2002), Modeling and analysis of software rejuvenation in cable modem termination system, *Proc. 13th Int'l Symp. on Software Reliab. Eng.*, 159–170, IEEE CS Press.

[15] Liu, Y., Mendiratta, V. and Trivedi, K. (2004), Survivability analysis of telephone access network, *Proc. 15th Int'l Symp. on Software Reliab. Eng.* (in press), IEEE CS Press.

[16] Lyu, M. R. (1995), *Software Fault Tolerance*, John Wiley & Sons.

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

[18] Marshall, A. W. and Olkin, I. (1967), A multivariate exponential distribution, *J. American Statist. Assoc.* **62**, 30–40.

[19] Osaki, S. (1980), *Reliability Evaluation of Some Fault-Tolerant Computer Architectures*, Lecture Notes in Computer Science **97**, Springer-Verlag.

[20] Osaki, S. (1980), Two-unit parallel redundant system with bivariate exponential lifetimes, *Microelectron. Reliab.* **20**, 521–523.

[21] Parnas, D. L. (1994), Software aging, *Proc. 16th Int'l Conf. on Software Eng.*, 279–287, IEEE CS Press.

[22] Park, K. and Kim, S. (2002), Availability analysis and improvement of active/standby cluster systems using software rejuvenation, *J. Systems and Software*, **61**, 121–128.

[23] Pullum, L. L. (2001), *Software Fault Tolerance: Techniques and Implementation*, Artech House.

[24] Shereshevsky, M., Cukic, B., Crowel, J. and Candikota, V. (2003), Software aging and multifractality of memory resources, *Proc. Int'l Conf. on Dependable Systems and Networks*, 721-730, IEEE CS Press.

[25] Suzuki, H., Dohi, T., Goševa-Popstojanova, K. and Trivedi, K. S. (2002), Analysis of multistep failure models with periodic software rejuvenation, *Advances in Stochastic Modelling* (J. R. Artalejo and A. Krishnamoorthy, eds.), Notable Publications, Inc., 85–108.

[26] Suzuki, H., Dohi, T., Kaio, N. and Trivedi, K. S. (2003), Maximizing interval reliability in operational software system with rejuvenation, *Proc. 14th Int'l Symp. on Software Reliab. Eng.*, 246–256, IEEE CS Press.

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

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

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

[30] Vaidyanathan, K., Harper, R. E., Hunter, S. W. and Trivedi, K. S. (2001), Analysis of software rejuvenation in cluster systems, *Proc. ACM SIGMETRICS 2001/Performance 2001*, 62–71, ACM.