Optimization of Two-Granularity Software Rejuvenation Policy Based on the Markov Regenerative Process

Gaorong Ning, Jing Zhao, Yunlong Lou, Javier Alonso, Rivalino Matias Jr., Kishor S. Trivedi, *Fellow, IEEE*, Bei-Bei Yin, and Kai-Yuan Cai

***Abstract*—Software rejuvenation is a proactive software control technique that is used to improve a computing system performance when it suffers from software aging. In this paper, a two-granularity inspection-based software rejuvenation policy, which works as a closed-loop control technique, is proposed. This policy mitigates the negative impact of two-level software aging. The two levels considered are the user-level applications and the operating system. A Markov regenerative process model is constructed based on the system condition. We obtain the degradation rate of the application software and operating system from fault injection experiments. The diagnostic accuracy of the adopted monitor and analysis system, which is applied to inspect the application software and operating system, is considered as we provide the optimal rejuvenation strategies. Finally, the availability and the overall loss probability with their corresponding optimal inspection time intervals are obtained numerically based on the parameter values estimated from the experiments. Experimental results show that two-granularity software rejuvenation is much more effective than traditional single-level software rejuvenation. In our experimental study, when two-granularity software rejuvenation is used, the unavailability and the overall loss probability of the system were reduced by 17.9% and 2.65%, respectively, in comparison with the single-level rejuvenation.**

***Index Terms*—Diagnostic accuracy, Markov regenerative process (MRGP), multigranularity software aging, overall loss probability, software rejuvenation.**

Manuscript received May 7, 2015; revised January 9, 2015, January 14, 2016, and January 26, 2016; accepted March 3, 2016. Date of publication June 16, 2016; date of current version November 29, 2016. This work was supported in part by the CNPq under Grant Universal 487182/2013-8, the U.S. National Science Foundation under Grant CNS-1523994, the National Natural Science Foundation of China and Microsoft Research Asia under Grant 61572150, Grant 61272164, and Grant 61402027, the Spanish National Institute of Cybersecurity (INCIBE) according to rule 19 of the Digital Confidence Plan (Digital Agency of Spain) and the University of León under Contract X43, and the Beijing Higher Education Young Elite Teacher Project under Grant YETP1072. Associate Editor: C. Smidts.

G. Ning is with the Commanding Automation Technique D&R and Application Center, the Fourth Academy, China Aerospace Science and Industry Corporation, Beijing 102308, China (e-mail: ninggaorong@buaa.edu.cn).

J. Zhao and Y. Lou are with the Department of Computer Science and Technology, Harbin Engineering University, Harbin 150001 China (e-mail: zhaoj@hrbeu.edu.cn; 906196253@qq.com).

J. Alonso is with the Research Institute of Applied Sciences in Cybersecurity, University of León, 24004 León, Spain (e-mail: javier.alonso@unileon.es).

R. Matias, Jr., is with the School of Computer Science, Federal University of Uberlandia, Uberlandia MG-38400-902, Brazil (e-mail: rivalino@ufu.br).

K. S. Trivedi is with the Department of Electrical and Computer Engineering, Duke University, Durham, NC 27708-0294 USA (e-mail: ktrivedi@duke.edu).

B.-B. Yin and K.-Y. Cai are with the Department of Automatic Control, Beihang University, Beijing 100191, China (e-mail: yinbeibei@buaa.edu.cn; kycai@buaa.edu.cn).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TR.2016.2570539

Nomenclature

Acronyms

AS	Application software.
OS	Operating system.
MRGP	Markov regenerative process.
M&A	Monitor and analysis system.
MTrD	Mean time from robust to degradation state.
EMC	Embedded Markov chain.
EMRS	Embedded Markov renewal sequence.
ALT	Accelerated life test.
TTF	Time to failure.
CBMG	Customer behavior model graph.
EBs	Emulated browsers.
JVM	Java virtual machine.
MCR	Mean consumption rate.

Notation

T_{in}	Time to trigger inspection.
$\mathbf{K}(t)$	Global kernel matrix.
$\mathbf{E}(t)$	Local kernel matrix.
$\Gamma(t)$	Gamma distribution.
π	Steady-state probability of the MRGP.
A	Steady-state system availability.
L	Overall loss probability.
$\lfloor\cdot\rfloor$	Floor function returns an integer value of nearest rounded down integer.
$\lambda_{(\cdot)}$	Transition rate.

I. Introduction

THE software aging concept was introduced more than 20 years ago. It is about the phenomenon that software shows a performance degradation after executing for a long time uninterruptedly [1]. The cause of software aging is essentially residual defects (bugs) either in the software code or in design [2]. Commonly investigated causes of software aging include but are not limited to memory leaks, unreleased file-locks, and round-off errors [2], [3]. To counteract software aging, Huang *et al.* [1] introduced a software recovery technique known as software rejuvenation. This technique is used to proactively mitigate the software aging effects before the system crashes. It has been applied extensively in industry and academia [4]–[6].

Aging-related bugs are usually hard to find and fix [7], [8]. We note that there are tools available for finding and fixing such bugs. Nevertheless, many such bugs remain in complex software systems during operation. Hence, a complementary mitigation method needs to be proposed to deal with these remaining bugs during the operational phase. Cleaning the software environment during operation has been demonstrated effective to deal with aging-related bugs. These cleaning actions can be done in a reactive manner (after the occurrence of failures due to these bugs) or can be done in a proactive manner. Software rejuvenation is a cost-effective technique to proactively schedule a cleanup of the software environment (by means of techniques like a restart or a reboot) prior to the occurrence of a failure due to such bugs. Software rejuvenation does not remove software bugs. Instead of removing software bugs, software rejuvenation “refreshes” the operating environment by removing accumulated errors and freeing system resources when it is triggered.

Generally speaking, two main approaches are used to address software aging and rejuvenation challenges: analytical model-based approach and measurement-based approach [3], [9]. The analytical model-based approach uses probability models to estimate the software aging process and determine the optimal rejuvenation scheduling based on the estimated aging state [1], [10]–[12]. On the other hand, measurement-based approaches collect information directly from the system and estimates using statistical or machine learning methods the real state of the system to trigger the rejuvenation process accordingly [13]–[23]. Meanwhile, the former approach is relatively easy to generalize across different systems because it is based on a simplification of the system. The latter is not easy to generalize since usually it exploits some peculiar aspect related to the nature of the considered system (e.g., the fact that some particular resource exhibits seasonal or fractal patterns). Moreover, measurement-based approaches are not meant to estimate long-term dependability measures such as availability. Several attempts have been made to combine the benefits of both model-based and measurement-based approaches, by describing the phenomenon analytically, most often by Markov-based models, and determining the model’s parameters (and some case even the model structure) through measurements, i.e., via observed data [24]. These efforts are here referred to as hybrid, in that they combine aspects of the previous two approaches. Despite the practical importance of hybrid approaches, only a minority of studies has been made to exploit measures for feeding models.

To the best of our knowledge, most research interests in the literature only give careful consideration to the rejuvenation at a single level [5], [25]. Castelli *et al.* [6] first proposed a two-level software rejuvenation method, which was developed theoretically in [12].

An experimental study of four-granularity rejuvenation techniques was first carried out by Alonso, Matias, Vicente, Maria, and Trivedi [26], which covers application level, OS level, virtual machine level, and physical machine level. Ning, Trivedi, Hu, and Cai [9] gave a theoretical analysis on multigranularity rejuvenation based on continuous-time Markov chain.

To put the multigranularity software rejuvenation into use in a real system, there are several challenges to solve.

1) The first one is how to know which granularity the aging-related fault is at. To rightly ascertain the aged granularity is much more difficult than to know whether there is an aged granularity in the system.
2) The second challenge is how to analyze and model the multigranularity software system when there are several granularities need to consider. These granularities may depend on each other. So to obtain the optimal multigranularity rejuvenation strategy is much tougher than that in single-level cases.
3) The third challenge is how to experimentally or theoretically prove that multigranularity rejuvenation strategy is more effective than single-level software rejuvenation. Anyhow, rebooting the system is a valid method to counteract the software aging without identifying the aging granularity.

When the phenomenon of software aging is observed in a software system, if there is too little information to identify the location of the aging-related faults, then single-level rejuvenation of rebooting the OS may be an appropriate choice. If the aging can be located at the module level, then multigranularity is a suitable method. Using tools such as Leaky [27], LeakTracer [28], and Compuware DevPartner Java Edition [29], we usually can know which module or granularity the memory related bug is in. Furthermore, if the causes of aging can be located in the code, then fixing it may be possible. Unfortunately, aging-related bugs are commonly very hard to locate and fix in code [6], [30]–[32].

In this paper, we apply the paradigm of MRGP to study two-granularity software rejuvenation scheduling. We obtain the steady-state system availability and the overall loss probability as functions of the inspection interval. Then, the optimal inspection intervals are calculated accordingly by maximizing the availability and minimizing the loss probability.

The main contributions of this paper are the following:

1) an MRGP model for capturing the two-granularity software states;
2) obtaining the diagnostic accuracy of the M&A and its effects on the rejuvenation strategy;
3) optimizing the two-granularity rejuvenation scheduling to maximize the system availability and minimize the loss probability.

The rest of this paper is organized as follows. In Section II, the two-granularity software rejuvenation policy is explained. Then, in Section III, an MRGP model for the two-granularity software rejuvenation policy is constructed and the closed-form solution is obtained. Section IV is devoted to availability analysis. In Section V, we present the experimental setup and the methods used in our experimental study. The experimental results are analyzed in Section VI. Section VII presents the threats to validity. Finally, Section VIII presents the concluding remarks.

II. Two-Granularity Software Rejuvenation Policy

Time-based [13], [25] and inspection-based [9], [14], [18] are the two main types of software rejuvenation scheduling strategies. Time-based rejuvenation is used to recover the system at predetermined time intervals. It is usually an open-loop control technique. When inspection-based rejuvenation is applied,

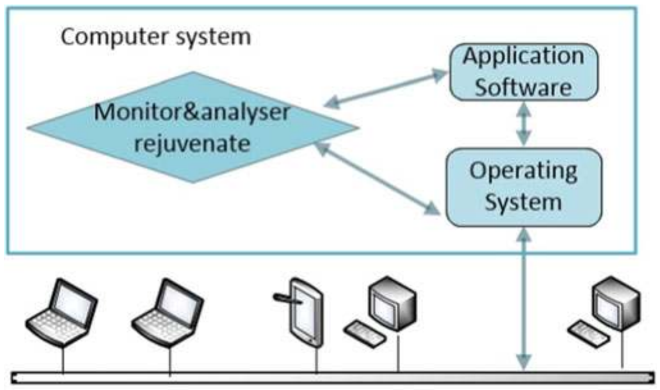


Fig. 1. Example of a two-level computer system.

it needs to periodically inspect the system to perceive software aging effects and then trigger rejuvenation accordingly. The inspection-based rejuvenation is, thus, a closed-loop control technique. Granularity of rejuvenation is another dimension of classification as proposed in [26]. This dimension identifies the location of aging effects and directly targets the aged granularity.

In this paper, we present an inspection-based model to analyze system availability, where two granularities are considered: AS level and OS level (see Fig. 1).

1) *AS-Level Granularity:* Application granularity software rejuvenation can be executed to mitigate the impacts of software aging at the AS level. This granularity rejuvenation implicates stopping and restarting the aged application. AS rejuvenation implies a proactive application restart that is useful in removing volatile in-memory application-specific aging effects [2], [6].
2) *OS-Level Granularity:* OS granularity is at a lower level than the AS granularity, whose effects are more complex than that of AS level. If the aging effects are caused by OS kernel, or that interacts between the AS and OS, then rejuvenation at the AS granularity is likely to be ineffective. Rejuvenation at the OS level would be required in this case.

A general model of software aging and rejuvenation was first presented by Huang *et al.* [1]. It is a four-state Markov model, including a robust state, a failure-prone state, a rejuvenation state, and a failure state, where the robust and the failure-prone states are available (up) states, and the failure state and the rejuvenation state are unavailable (down) states. Xie, Hong, and Trivedi [12] proposed a more detailed model by dividing the failure-prone state into “middle efficient state” and “low efficient state” and dividing the rejuvenation state into “partial rejuvenation state” and “full rejuvenation state” while studying inspection-based preventive maintenance in two-level systems. Alonso, Matias, Vicente, Maria, and Trivedi [26] proposed a comparative experimental analysis of four-granularity software and software rejuvenation overhead. In their paper, AS rejuvenation strategy is used to mitigate the aging effects first. If AS rejuvenation strategy fails, then the next higher

TABLE I
STATE DEFINITION OF THE SYSTEM

NO.	State	AS	OS	Keep?	Available?
0	Robust	Robust	Rob.	Yes	Yes
1	DA	Rej.	Rob.	Yes	No
2	DO	Rej.	Deg.	Yes	No
3	DV	Rej.	Rej.	Yes	No
4	DP	Failed	Rob.	Yes	No
5	FA	Failed	Deg.	Yes	No
6	FO	Failed	Failed	Yes	No
7	rD	Rob.	Deg.	Yes	Yes
8	Dr	Deg.	Rob.	Yes	Yes
9	DD	Deg.	Deg.	Yes	Yes
10	rR	Rob.	Rej.	No	–
11	rF	Rob.	Failed	No	–
12	DF	Deg.	Deg.	No	–
13	DR	Deg.	Rej.	No	–
14	RF	Rej.	Failed	No	–
15	FR	Failed	Rej.	No	–

level of granularity rejuvenation is executed till the aging is mitigated.

Rejuvenation strategies of different granularity treat different levels of software aging impact and have different costs. So it is necessary to choose the proper rejuvenation strategy to lessen the aging impacts at different levels. Being different from [26], our paper deals with how to obtain the optimal strategy and how to use those rejuvenation techniques presented by the above cited paper. Our method is to rejuvenate the aging part (or level) with the corresponding granularity rejuvenation technique, rather than rejuvenate the system from low level to high level.

We develop the aforementioned model proposed in [1] by dividing the software system into AS and OS software. We consider the case that the AS and the OS have their separated memory usage. In our model, for AS, there are four states: robust state, degradation state, rejuvenation state, and failed state, which are denoted by r, D, R, and F, respectively. Similarly, OS software has the same four states. So, there are total $4 \times 4 = 16$ states for the whole software system, which are depicted in Table I. These states are represented by two letters. The left one corresponds to the AS level and the right one corresponds to the OS level. For instance, state “rD” means AS is at robust state and OS is at degradation state. Out of 16 states, we need to consider only ten states, and the other six states can be ignored. Take the state “rF” for example, if the OS software has failed, the application will not be available. So, the state “rF” can be dropped.

An M&A is used to decide at which granularity rejuvenation should be carried out. The M&A consists of a monitor and a data analyzer. The monitor collects specific data from the AS and OS when the system is up. The data analyzer evaluates the data collected by the monitor and then diagnoses the state of the target system. Let (t) be the state reported by the M&A at time t. The maintenance actions of the system are determined by (t), which are given by Table II.

We have the following assumptions for our two-granularity software aging and rejuvenation model.

TABLE II
MAINTAIN ACTIONS AND THEIR CORRESPONDING TIME DETERMINED BY THE MONITOREDSTATE(t)

NO.	MonitoredState(t)	Action at AS level	Action at OS level	Time of the action
0	rr	no action	no action	-
4	Fr	Rea. Rest.	no action	T_{A4}
5	FD	Proa. Rest.	Proa. Rebo.	T_{O3}
6	FF	Rea. Rest.	Rea. Rebo.	T_{O4}
7	rD	Proa. Rest.	Proa. Rebo.	T_{O3}
8	Dr	Proa. Rest.	no action	T_{A3}
9	DD	Proa. Rest.	Proa. Rebo.	T_{O3}

1) The time to inspection trigger is a constant, T_{in}, which is actually the constant variable in this paper.
2) There is no delay in carrying out the inspection for the M&A, which means that the system makes instantaneous diagnoses after it is triggered.
3) When the AS fails, the M&A will be triggered immediately to detect the state of the OS.
4) After each rejuvenation of the AS or the OS, the timer of the M&A will be reset.
5) The holding time T_{A1} (or T_{O1}) of AS (or OS) from robust state to degradation state has the exponential distribution with parameter λ_{A1} (or λ_{O1}). The holding time T_{A2} (or T_{O2}) of the AS (or OS) from degradation state to failed state also has the exponential distribution with parameter λ_{A2} (or λ_{O2}).
6) The rejuvenation time T_{A3} (or T_{O3}) of the AS (or OS) from degradation state to robust state has a general distribution $F_{A3}(t)$ (or $F_{O3}(t)$). The reactive restart (or reboot) time T_{A4} (or T_{O4}) of the AS (or OS) also has a general distribution $F_{A4}(t)$ (or $F_{O4}(t)$).

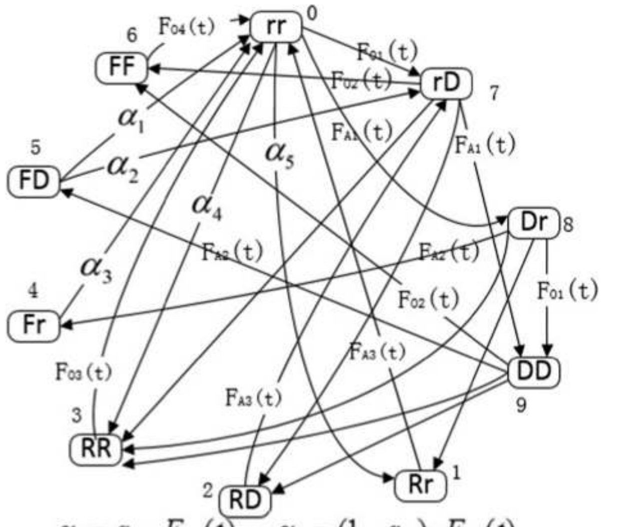
Let TrueState(t) be the true state of the system at time t. Assume that the M&A does not always make correct diagnoses. MonitoredState$(t) \neq$ TrueState(t) means that the M&A makes a misdiagnosis. Recall that the AS and the OS have their separated memory usage. Suppose that the M&A makes diagnoses independently for the AS and the OS. Let MonitoredState$_{\text{AS}}(t)$ and MonitoredState$_{OS}(t)$ be the diagnosis results of AS and OS, respectively. $(t) =$ 'rD' means that MonitoredState$_{\text{AS}}(t) =$ 'r' and MonitoredState$_{OS}(t) =$ 'D.' Suppose that if AS and OS are at one of the states in {failed, rejuvenation}, they can be rightly diagnosed by the M&A. Let

$$\begin{aligned} p_r &= Pr\{\text{MonitoredState}_{\text{AS}}(t) = \text{r} \,|\text{TrueState}_{\text{AS}}(t) = \text{r}\} \\ p_D &= Pr\{\text{MonitoredState}_{\text{AS}}(t) = \text{D} \,|\text{TrueState}_{\text{AS}}(t) = \text{D}\} \\ q_r &= Pr\{\text{MonitoredState}_{\text{OS}}(t) = \text{r} \,|\text{TrueState}_{\text{OS}}(t) = \text{r}\} \\ q_D &= Pr\{\text{MonitoredState}_{\text{OS}}(t) = \text{D} \,|\text{TrueState}_{\text{OS}}(t) = \text{D}\}. \end{aligned} \tag{1}$$

When and which rejuvenation technique should be executed depends on the diagnosis results from the M&A. For example, if TrueState(t) is "rD," there are four possible actions according to the inspection results MonitoredState(t):

$$\begin{cases} \text{no action} \rightarrow \text{rD}, & \text{if } (t) = \text{rr} \\ \text{Reju. OS} \rightarrow \text{rr}, & \text{if } (t) = \text{rD} \\ \text{Reju. AS} \rightarrow \text{rD}, & \text{if } (t) = \text{Dr} \\ \text{Reju. OS} \rightarrow \text{rr}, & \text{if } (t) = \text{DD} \end{cases}$$

by noting our assumption that if the AS and the OS are at one of the states in {failed, rejuvenation}, they can be rightly diagnosed by the M&A. The meaning of the above brace can be explained as follows. Suppose that TrueState(t) is "rD." If MonitoredState(t) is "rr," which means that the M&A makes a right report regarding the AS level and makes a wrong report regarding the OS level, then no maintenance action will be taken. The true state of the system will still be "rD." If MonitoredState(t) is "Dr," which means that the M&A makes wrong reports regarding both the AS level and the OS level, then the action of AS rejuvenation will be taken. The true state of the system will still be "rD" after AS rejuvenation. The true state transition path in this case is rD $\rightarrow$ RD $\rightarrow$ rD.



$$\alpha_1 = q_D \cdot F_{O3}(t), \quad \alpha_2 = (1-q_D)\cdot F_{A4}(t),$$
$$\alpha_3 = q_r \cdot F_{A4}(t) + (1-q_r)\cdot F_{O3}(t),$$
$$\alpha_4 = (1-q_r)\cdot F_{\text{in}}(t), \quad \alpha_5 = (1-p_r)\cdot q_r \cdot F_{\text{in}}(t).$$

Fig. 2. True state diagram for the two-granularity software aging and rejuvenation model.

Based on the rejuvenation scheduling defined above, the true state diagram is determined. Fig. 2 shows the state diagram of the ten states for our two-granularity software aging and rejuvenation model. In the figure, $F_{(\cdot)}(t)$ are the distribution functions of the corresponding sojourn times. According to our supposition, the holding time T_{A1} of the AS transition from robust state to degradation state follows exponential distribution $F_{A1}(t)$. Also the holding times T_{A2}, T_{O1}, and T_{O2} follow exponential distribution with the distribution functions $F_{A2}(t)$, $F_{O1}(t)$, and $F_{O2}(t)$, respectively. As shown in Fig. 2, because the time to trigger inspection is a deterministic value T_{in}, the stochastic process $Z(t)$ determined by the model is not a Markov process but an MRGP whose EMC is identified by states $0, 1, 2, 3, 4, 5,$ and 6. And the states $7, 8,$ and 9 are not regenerative state and hence are not in the EMC. The holding times of the transitions from one of the states in $\{7, 8, 9\}$ to one of the states in $\{1, 2, 3\}$ are not conditional independent. For example, the sojourn time distributions of the transition from state 7 to state 2 depend on the latest state to state 7. If the trajectory is from state 5 to state 7 and to state 2, the sojourn time at state 7 is T_{in}; if the trajectory is from state 1 to state 7 and to state 2, the sojourn time at state 7 is smaller than T_{in}.

A formalized description of the MRGP model will be given in the following section.

III. MRGP Model Analysis

Here, we study the steady-state probability of the MRGP model presented in Section II. We use $Z = \{Z(t); t \geq 0\}$ with state space $\Phi = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$ to represent the state of the system. Clearly, the stochastic process $Z(t)$ is not a homogeneous continuous-time Markov chain, because not all the sojourn times of the states are exponentially distributed. Furthermore, $Z(t)$ is not a semi-Markov process if the distribution $F_{\text{in}}(t)$ is not exponential. $Z(t)$ is called regenerative if there exist time points S_i at which the process probabilistically replicates itself [20]. The random times S_i are called regeneration time points of $Z(t)$. The sequence $\{S_i, i = 1, 2, \ldots\}$ of the regeneration points identifies an embedded renewal sequence. Therefore, at each regeneration point S_i, the process $Z(t)$ is independent of its past and has the same stochastic behavior as that $Z(t)$ has from $t = 0$.

$Z(t)$ is called an MRGP if it identifies an EMRS (X, S) with the further property that the distributions of $\{Z(t + S_n); t \geq 0\}$ supposed $\{Z(u); 0 \leq u \leq S_n, X_n = i, \ i \in \Omega \subseteq \Phi\}$, are the same as those of $\{Z(t), t \geq 0\}$ with $X_0 = i$, where $\Omega = \{0, 1, 2, 3, 4, 5, 6\}$. So we know that $\{Z(t); t \geq 0\}$ has Markov property at time points $(S_0, S_1, \ldots, S_n, \ldots)$ with the corresponding state sequence $(X_0, X_1, \ldots, X_n, \ldots)$, respectively. This also implies that the future of the process Z when $Z(t) = i \notin \Omega$ may do not have the Markov property, which means that state changes between two consecutive regeneration epochs S_i and S_{i+1} do not have regenerations. So, it might be possible that $Z(t)$ returns to Ω without passing these Markov regeneration epochs.

For such an MRGP, we can identify the matrix $\mathbf{V}(t) = [V_{i,j}(t)]$ as the conditional transition probabilities:

$$V_{i,j}(t) = Pr\{Z(t) = j \mid Z(0) = i\}.$$

At any instant t, the conditional transition probability matrix $V_{i,j}(t)$ defined by $Z(t)$ can be expressed as follows [8], [10]:

$$\begin{aligned} V_{i,j}(t) =\ & Pr\{Z(t) = j, S_1 > t \mid Z(0) = i\} \\ & + Pr\{Z(t) = j, S_1 \leq t \mid Z(0) = i\} \\ =\ & Pr\{Z(t) = j, S_1 > t \mid Z(0) = i\} \\ & + \sum_{k \in \Omega} \int_0^t dK_{i,k}(u) V_{k,j}(t-u) \end{aligned}$$

for all $i \in \Omega, j \in \Phi$, and $t \geq 0$. Let $\mathbf{E}(t) = [E_{i,j}(t)]$, where

$$E_{i,j}(t) = Pr\{Z(t) = j, S_1 > t \mid Z(0) = i\}.$$

Then, the integral equations for $V_{i,j}(t)$ identify a Markov renewal equation and can be written in the following matrix form:

$$\mathbf{V}(t) = \mathbf{E}(t) + \int_0^t d\mathbf{K}(u)\mathbf{V}(t-u). \qquad (2)$$

The matrix $\mathbf{K}(t)$ in (2) is called the global kernel matrix and $\mathbf{E}(t)$ is called the local kernel matrix of the MRGP, where $\mathbf{K}(t) = (\mathbf{K}_1(t),\ \mathbf{K}_2(t))$, and

$$\mathbf{K}_1(t) = \begin{pmatrix} 0 & k_{0,1}(t) & k_{0,2}(t) & k_{0,3}(t) \\ k_{1,0}(t) & 0 & 0 & 0 \\ 0 & 0 & 0 & k_{2,3}(t) \\ k_{3,0}(t) & 0 & 0 & 0 \\ k_{4,0}(t) & 0 & 0 & 0 \\ k_{5,0}(t) & 0 & k_{5,2}(t) & k_{5,3}(t) \\ k_{6,0}(t) & 0 & 0 & 0 \end{pmatrix}$$

$$\mathbf{K}_2(t) = \begin{pmatrix} k_{0,4}(t) & k_{0,5}(t) & k_{0,6}(t) \\ 0 & 0 & 0 \\ 0 & k_{2,5}(t) & k_{2,6}(t) \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & k_{5,6}(t) \\ 0 & 0 & 0 \end{pmatrix}.$$

Here, we present $k_{0,1}(t)$ and $k_{0,2}(t)$. The other $k_{i,j}(t)$ are given in the Appendix briefly.

For $k_{0,1}(t)$, there are two paths: $0 \to 1$ and $0 \to 8 \to 1$. Because the M&A is triggered at evenly spaced intervals T_{in} which is value to optimize in this paper, so we have

$$\begin{aligned} k_{0,1}(t) =\ & Pr\{Z(S_1) = 1, S_1 \leq t \mid Z(0) = 0\} \\ =\ & \sum_{k=1}^{\lfloor \frac{t}{T_{\text{in}}} \rfloor} (p_r q_r)^{k-1}(1-p_r) q_r e^{-(\lambda_{A1}+\lambda_{O1})kT_{\text{in}}} \\ & + p_D q_r \sum_{k=1}^{\lfloor \frac{t}{T_{\text{in}}} \rfloor} \int_0^{kT_{\text{in}}} \lambda_{A1} e^{-(\lambda_{A1}+\lambda_{O1})x} (p_r q_r)^{\lfloor \frac{x}{T_{\text{in}}} \rfloor} \\ & [(1-p_D)q_r]^{k - \lfloor \frac{x}{T_{\text{in}}} \rfloor - 1} e^{-(\lambda_{A1}+\lambda_{O1})(kT_{\text{in}}-x)} dx \end{aligned}$$

where the function $\lfloor x \rfloor$ finds the nearest integers less than or equal to x. So $\lfloor x/T_{\text{in}} \rfloor$ is the number of diagnoses before the state of the AS is degradation, and these diagnoses are rightly made. So the number of diagnoses is $k - \lfloor x/T_{\text{in}} \rfloor$, when the system is at state "Dr," of which the first $k - \lfloor x/T_{\text{in}} \rfloor - 1$ diagnoses are wrongly made and the last one is rightly made.

For $k_{0,2}(t)$, there are three paths: $0 \to 7 \to 2$, $0 \to 7 \to 9 \to 2$, and $0 \to 8 \to 9 \to 2$. So

$$k_{0,2}(t) = Pr\{Z(S_1) = 2, S_1 \leq t \mid Z(0) = 0\}$$

$$
\begin{aligned}
= {} & (1-p_r)(1-q_D)\sum_{k=1}^{\lfloor \frac{t}{T_{\text{in}}}\rfloor}\int_0^{kT_{\text{in}}}\lambda_{O1}e^{-(\lambda_{A1}+\lambda_{O1})x} \\
& (p_rq_r)^{\lfloor \frac{x}{T_{\text{in}}}\rfloor}[p_r(1-p_D)]^{k-\lfloor \frac{x}{T_{\text{in}}}\rfloor-1} \\
& e^{-(\lambda_{A1}+\lambda_{O1})(kT_{\text{in}}-x)}dx \\
& +p_D(1-q_D)\sum_{k=1}^{\lfloor \frac{t}{T_{\text{in}}}\rfloor}\int_0^{kT_{\text{in}}}\int_x^{kT_{\text{in}}}\lambda_{O1}e^{-(\lambda_{A1}+\lambda_{O1})x} \\
& \lambda_{A1}e^{-(\lambda_{A1}+\lambda_{O2})(y-x)}e^{-(\lambda_{A2}+\lambda_{O2})(kT_{\text{in}}-y)} \\
& (p_rq_r)^{\lfloor \frac{x}{T_{\text{in}}}\rfloor}[p_r(1-q_D)]^{\lfloor \frac{y}{T_{\text{in}}}\rfloor-\lfloor \frac{x}{T_{\text{in}}}\rfloor} \\
& [(1-p_D)(1-q_D)]^{k-\lfloor \frac{y}{T_{\text{in}}}\rfloor-1}dydx \\
& +p_D(1-q_D)\sum_{k=1}^{\lfloor \frac{t}{T_{\text{in}}}\rfloor}\int_0^{kT_{\text{in}}}\int_x^{kT_{\text{in}}}\lambda_{A1}e^{-(\lambda_{A1}+\lambda_{O1})x} \\
& \lambda_{O1}e^{-(\lambda_{O1}+\lambda_{A2})(y-x)}e^{-(\lambda_{A2}+\lambda_{O2})(kT_{\text{in}}-y)} \\
& (p_rq_r)^{\lfloor \frac{x}{T_{\text{in}}}\rfloor}[(1-p_D)q_r]^{\lfloor \frac{y}{T_{\text{in}}}\rfloor-\lfloor \frac{x}{T_{\text{in}}}\rfloor} \\
& [(1-p_D)(1-q_D)]^{k-\lfloor \frac{y}{T_{\text{in}}}\rfloor-1}dydx.
\end{aligned}
$$

Once $\mathbf{K}(t)$ is specified, we need to obtain the local kernel matrix $\mathbf{E}(t)$. Because an MRGP can change states between two consecutive Markov regeneration epochs, we need to capture these changes through the matrix $\mathbf{E}(t)$. Moreover, the cardinality of $Z(t)$ can be larger than the cardinality of the EMC state space and so, generally, $\mathbf{E}(t)$ is a rectangular matrix. In fact, in the present example, the EMC has seven states, while the MRGP has ten possible states. By a careful examination of the MRGP in Fig. 2, we have $\mathbf{E}(t) = \mathbf{E}_1(t),\ \mathbf{E}_2(t)$, where

$$\mathbf{E}_1(t) = \text{diag}\{E_{0,0}(t), E_{1,1}(t), \ldots, E_{6,6}(t)\}$$

and

$$\mathbf{E}_2(t) = \begin{pmatrix} E_{0,7}(t) & 0 & E_{2,7}(t) & 0 & 0 & E_{5,7}(t) & 0 \\ E_{0,8}(t) & 0 & 0 & 0 & 0 & 0 & 0 \\ E_{0,9}(t) & 0 & E_{2,9}(t) & 0 & 0 & E_{5,9}(t) & 0 \end{pmatrix}^T.$$

Here, we given $E_{0,0}(t)$ and $E_{0,7}(t)$. The others are given in the Appendix. For $E_{0,0}(t)$, we have

$$
\begin{aligned}
E_{0,0}(t) = {} & Pr\{Z(S_1)=1, S_1>t \mid Z(0)=0\} \\
= {} & \sum_{k=0}^{+\infty} Pr\{Z(S_1)=1, S_1>t \mid Z(0)=0, \\
& N_{\text{in}}(t)=k\}\cdot Pr\{N_{\text{in}}(t)=k\} \\
= {} & e^{-(\lambda_{A1}+\lambda_{O1})t}[(1-F_{\text{in}}(t)) \\
& +\sum_{k=1}^{\infty}(p_rq_r)^k\int_0^t(1-F_{\text{in}}(t-x))dF_{\text{in}}^{*k}(x)] \\
= {} & e^{-(\lambda_{A1}+\lambda_{O1})t}(p_rq_r)^{\lfloor \frac{t}{T_{\text{in}}}\rfloor}.
\end{aligned}
$$

For $E_{0,7}(t)$, we have

$$
\begin{aligned}
E_{0,7}(t) = {} & \int_0^t (p_rq_r)^{\lfloor \frac{x}{T_{\text{in}}}\rfloor}[p_r(1-q_r)]^{\lfloor \frac{t}{T_{\text{in}}}\rfloor-\lfloor \frac{x}{T_{\text{in}}}\rfloor} \\
& \lambda_{O1}e^{-(\lambda_{A1}+\lambda_{O1})x}e^{-(\lambda_{A1}+\lambda_{O2})(t-x)}dx.
\end{aligned}
$$

Once both the kernel matrices $\mathbf{K}(t)$ and $\mathbf{E}(t)$ are specified, the model can be analyzed. For the steady-state probabilities of the EMC, we define

$$\mathbf{K} := \lim_{t\to\infty}\mathbf{K}(t).$$

Solve the equation $\mathbf{v} = \mathbf{v}\cdot\mathbf{K}$, where the row vector is $\mathbf{v} = (v_0, v_1, \ldots, v_6)$; then, one of the solutions is

$$
\begin{aligned}
\mathbf{v} = {} & (\,1,\ k_{0,1},\ v_2,\ k_{0,2}+k_{0,5}k_{5,2}+v_2k_{2,5}k_{5,2}, \\
& k_{0,4},\ k_{0,5}+v_2k_{2,5}, \\
& k_{0,6}+k_{0,5}k_{5,6}+v_2(k_{2,6}+k_{2,5}k_{5,6})\,)
\end{aligned}
$$

where

$$v_2 = \frac{k_{0,2}+k_{0,5}k_{5,2}}{1-k_{2,5}k_{5,2}}.$$

So, the steady-state probability vector of the EMC is

$$\mathbf{v}^* = \frac{\mathbf{v}}{\sum_{i=0}^{6} v_i}.$$

To obtain the steady-state probabilities of the MRGP, we define the following terms:

$$\mu_i = \mathbb{E}[S_1 \mid X_0 = i] \quad \text{and} \quad \alpha_{m,n} = \int_0^\infty E_{m,n}dt$$

where μ_i are given in the Appendix, and $\alpha_{m,n}$ can be calculated with $E_{m,n}$, which are also given in the Appendix.

Then, the steady-state probabilities of the MRGP are given by

$$\pi_i = \sum_{k=0}^{6}\beta_k\frac{\alpha_{k,i}}{\mu_k}, \quad i\in\{0,1,2,\ldots,9\}$$

with

$$\beta_k = \frac{v_k\mu_k}{\sum_{i=0}^{6}v_i\mu_i}.$$

IV. Availability Analysis

Here, we provide more detailed analysis for the availability. The distributions $F_{A3}(t)$ and $F_{O3}(t)$ of proactive rejuvenation times are chosen as Weibull distributions, which occur in many situations of scientific interest, especially in man-made phenomena [33]. Here, the distributions $F_{A4}(t)$ and $F_{O4}(t)$ of reactive rejuvenation times are chosen as Pareto distributions. The distribution functions are given by

$$
\begin{aligned}
F_{A3}(t) = {} & 1-e^{-(t/a)^b},\ F_{O3}(t) = 1-e^{-(t/c)^d} \\
F_{A4}(t) = {} & 1-\frac{m^\sigma}{t},\ F_{O4}(t) = 1-\frac{n^\varrho}{t}.
\end{aligned} \tag{3}
$$

The expected times $\mathbb{E}(T_{A3})$, $\mathbb{E}(T_{O3})$, $\mathbb{E}(T_{A4})$, and $\mathbb{E}(T_{O4})$ in (3) can be rewritten as

$$\mathbb{E}(T_{A3}) = a\Gamma(1+1/b), \qquad \mathbb{E}(T_{O3}) = c\Gamma(1+1/d)$$
$$\mathbb{E}(T_{A4}) = \frac{m\sigma}{\sigma-1}, \qquad \mathbb{E}(T_{O4}) = \frac{n\varrho}{\varrho-1}.$$

Actually, our model can accommodate very general distributions. When the MRGP steady-state analysis is employed, we just need the mean sojourn time for each repair state. However, Weibull distribution is very popular in modeling the TTF and the time to repair of complex systems [33]. The use of Weibull and Pareto distributions can be seen as an example only. We highlight that due to the nature of MRGP models, any other distribution could be used, which makes the adopted approach very flexible.

To maximize system availability, different scenarios of software aging and corresponding rejuvenation policies have been considered [3]. Suppose that the system is up when it is at state in $i \in S_A = \{0, 7, 8, 9\}$. If the system state is not in S_A, it is not up. Then, the steady-state system availability as a function of the inspection time interval T_{in} is

$$A(T_{\text{in}}) = \sum_{i \in S_A} \pi_i(T_{\text{in}}).$$

Then, the optimal inspection interval T^*_{in} for maximizing the availability satisfies

$$\frac{d(A(T^*_{\text{in}}))}{dT^*_{\text{in}}} = 0.$$

In addition, we consider the loss probability L_i measured value as the reward of the model for the up state i in S_A. The reward rate of loss probability is 1 for the down states. Then, the overall loss probability of the system is

$$L(T_{\text{in}}) = \sum_{i \in S_A} L_i \cdot \pi_i(T_{\text{in}}) + \sum_{i \notin S_A} \pi_i(T_{\text{in}}).$$

The optimal inspection interval T^*_{in} for minimizing the overall loss probability satisfies

$$\frac{d(L(T^*_{\text{in}}))}{dT^*_{\text{in}}} = 0.$$

V. Experiments

Memory leaks are recognized to be one of the major causes of resource exhaustion problems in complex software, which represents one of the most serious causes of aging. Many papers have studied the effect of memory leaks in software [2], [15], [21], [32], [34]–[38].

The ALT approach has been successfully applied in many engineering fields [36], [37] to significantly reduce the experimentation time. In [36] and [37], the authors employed the ALT approach to solve the optimal rejuvenation problem. In this paper, we employ the injecting memory leaks approach to study the two-granularity strategies. We focus on the distributions of the TTF of the AS and the OS when they suffer from software aging.

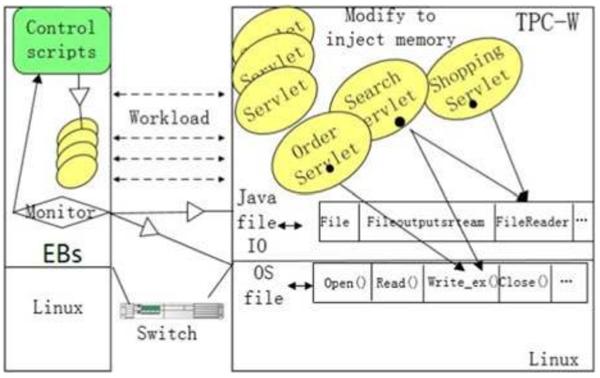


Fig. 3. Experiment test bed.

A. Experiment Setup

Our test bed is based on the settings used in [35], where it is composed of two physical machines which are connected by a switch. The two physical machines have exactly the same hardware facilities (CPU: Intel(R) Core(TM)2 Quad Q9400, RAM: 2 Gbyte, NIC: 1000 Mbps). One of the physical machines is used as a server provider, which is the system under test. The OS in this server is Fedora 13: 2.6.33.3-85.fc.i386.PAE. We chose TPC_W benchmark of Version 6.0.29 as the web server software [40], which also includes MySQL as the database server. The other physical machine is used to imitate the clients generating requests to the server provider. It is composed of a set of EBs. The number of EBs is 100 throughout the experiment.

In TPC-W standard, there are 14 different web pages including home page, best selling page, new books page, search page, shopping cart, order status page, etc. According to the functions, the 14 pages can be divided into browse and order categories. The order pages typically have greater processing requirements than the browse pages due to the high frequency of database accesses and secure transactions. The users' navigation pattern is emulated based on a CBMG. The CBMG shows how the users navigate, how to use the functions, and how to control the transition frequency of these functions. In a user session, an EB sends a sequence of logically connected requests to the web application server. Before the EB sends the next requests, there is a "thinking time." We can obtain the controlled execution percentage of each page by enforcing the transition probabilities for the CBMG.

We wrote a perl script to work as the monitor which collects the memory usage of the two levels and other information from the web server. In our experiment, the monitor is executed every 7 s.

Fig. 3 illustrates the experimental environment used in this paper.

B. Separation of the AS and the OS

To make sure that the memory usage of the AS and the OS is independent as the initial assumption, we configure the JVM in which Tomcat and TPC_W servlets run. The initial and maximum heap size is set to be 25% of the total available physical

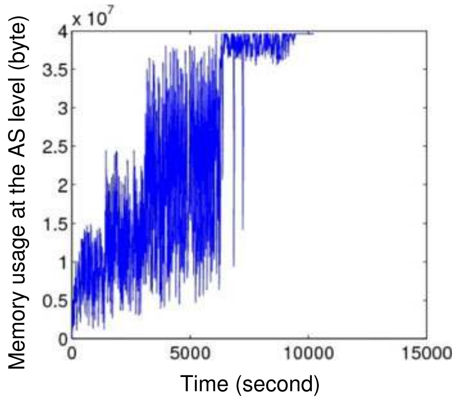


Fig. 4. AS memory used after memory separation.

memory. Therefore, the maximum size of the Heap will be allocated to it once the JVM is started. In this way, the memory allocated to the TPC_W would not change throughout the experiment, and thus, memory usage will work independently, and therefore, we will able to collect the memory usage of each level separately [41].

Fig. 4 shows that the heap size is fixed after memory usage separation.

C. Injecting Memory Leaks

To emulate the software aging effects of memory exhaustion, we inject memory leaks into the AS and the OS by modifying the source code of TPC_W Benchmark and an OS kernel module, respectively.

1) Injecting Memory Leaks Into the AS: We modify the two pages TPC_W_search_request_Servlet and TPC_W_Shopping_cart_interaction. If the request_Servlet or Shopping_cart_interaction is requested by the EBs, then a bug is triggered and a random number N, which is the size of the injected memory leak, is generated on each invocation of doGet(). In this way, we embed two bugs into the pages of TPC_W_search_request_servlet and TPC_W_shopping_cart_interaction. Fig. 5 illustrates how memory leaks are injected at the AS level.

2) Injecting Memory Leaks Into the OS: A kernel module is written to inject memory leaks into the OS to emulate the memory exhaustion effects at the OS level. In this module, we add a new function write_ex() which has the same function as the kernel system call write(). When TPC_W_search_request_Servlet or TPC_W _order_status_Servlet is requested, the added kernel module is then triggered. In this case, the routine write_ex() is executed, which makes memory leak inside the kernel. Note that the other software will not call write_ex(), because this is exclusive for our use only. If TPC_W_search_request_Servlet is visited, both the AS level and the OS level are injected with memory leaks. The process of injecting memory leaks into the OS is similar to that into the AS. The process is presented in Fig. 6.

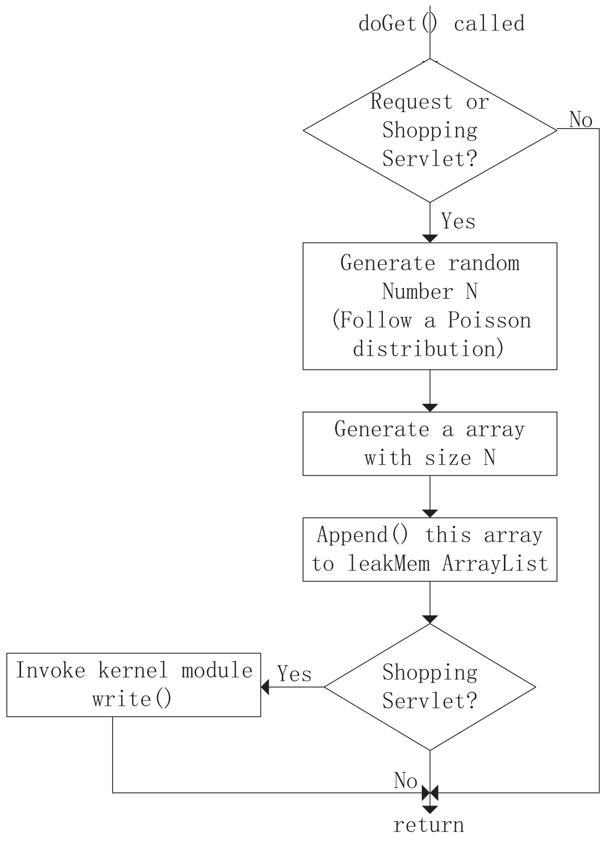

doGet() called
Request or Shopping Servlet?
No
Yes
Generate random Number N (Follow a Poisson distribution)
Generate a array with size N
Append() this array to leakMem ArrayList
Shopping Servlet?
Yes
Invoke kernel module write()
No
return

Fig. 5. AS injecting memory leak process.

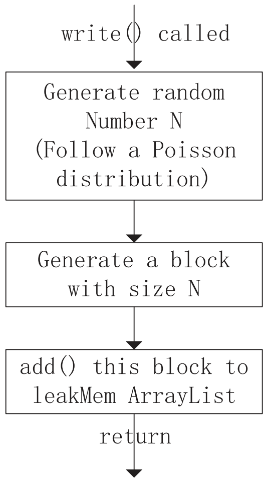

write() called
Generate random Number N (Follow a Poisson distribution)
Generate a block with size N
add() this block to leakMem ArrayList
return

Fig. 6. Memory leak injection at the OS level.

D. System State Definition and Diagnostic Accuracy of Monitor

Before we get the sojourn time of each state, we need to define the states of AS and OS. In this paper, we use a full path method to define the states of AS and OS, which is used to obtain the misdiagnosis ratio of the M&A. The advantage of the full path method is that we need only a few sensors and the diagnosis can be done offline. Fig. 7 presents the monitoring process. First, it

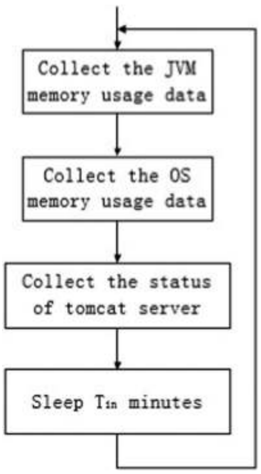

Collect the JVM memory usage data
Collect the OS memory usage data
Collect the status of tomcat server
Sleep Tin minutes

Fig. 7. Monitor.

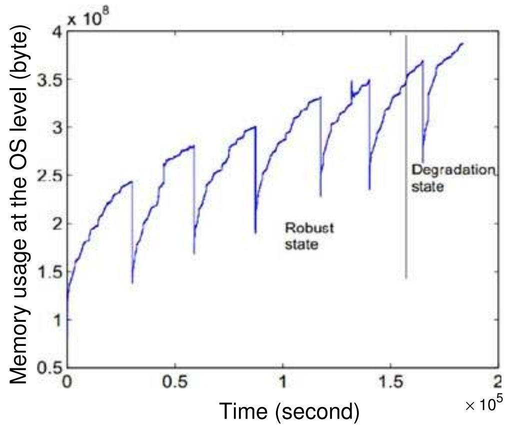


Fig. 9. State definition of OS.

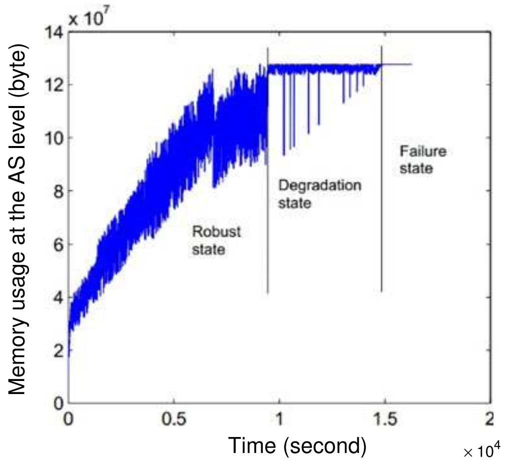


Fig. 8. State definition of AS.

invokes JVM monitoring tool Jmap to obtain the memory usage of Young, Old, and Permanent spaces. Second, it invokes the system program "free" to obtain the memory usage of the OS. Last, it invokes a script written in bash to obtain the status of the Tomcat server by visiting the admin page of Tomcat server.

Let t_{fA} be the time that AS goes down; then, we define the critical time point t_{DA} as

$$t_{\mathrm{DA}} = \min_{t^*}\{t^* \mid \text{the average memory usage of AS in the time interval } [t^*,\ t_{\mathrm{fA}}] \text{ equals } 90\% \text{ of the total memory of AS}\}.$$

Then, we consider that $\mathrm{TrueState}_{\mathrm{AS}}(t)$ of AS is degradation state when $t \in [t_{\mathrm{DA}},\ t_{\mathrm{fA}}]$, and $\mathrm{TrueState}_{\mathrm{AS}}(t)$ is robust state when $t \in [0,\ t_{\mathrm{DA}}]$. We define the states of the OS just in the same way as that of the AS (see Figs. 8 and 9).

A question is why we use 90% of the total memory as the critical value to define the degradation state of AS. Actually, when the memory usage of AS is about 90% of its total memory, it is expected that the swap space starts being used and thus the system performance is affected.

After $\mathrm{TrueState}_{\mathrm{AS}}(t)$ being defined, we give the definition of $\mathrm{MonitoredState}_{\mathrm{AS}}(t)$. To improve the diagnostic accuracy of the M&A, we set 2% absolute error tolerance. So if the M&A reports that $\mathrm{MonitoredState}_{\mathrm{AS}}(t_k)$ is robust state, then the M&A reports that $\mathrm{MonitoredState}_{\mathrm{AS}}(t_{k+1})$ at time $(k+1)\tau$ is degradation state if and only if the AS memory usage inspected by the M&A is more than 92% of its total memory. If the M&A reports that $\mathrm{MonitoredState}_{\mathrm{AS}}(t_k)$ is degradation state, then the M&A reports that $\mathrm{MonitoredState}_{\mathrm{AS}}(t_{k+1})$ is robust state if and only if the AS memory usage inspected by the M&A is less than 88% of the total memory.

If $\mathrm{MonitoredState}_{\mathrm{AS}}(t) \neq \mathrm{TrueState}_{\mathrm{AS}}(t)$ in an inspection, then we say that the M&A makes a misdiagnosis in this inspection.

Another question is why we set 2% absolute error tolerance when we determine the diagnostic accuracy of the M&A. Actually, if we set a larger value for the absolute error tolerance, then the M&A becomes more insensitive when the available memory changes. If we set a smaller value, then the M&A becomes much more sensitive when the available memory changes. If the M&A is insensitive, then type II error in our statistical hypothesis testing occurs with larger probability. While if the M&A is sensitive, then type I error occurs with larger probability. Of course, a reasonable value for absolute error tolerance is likely to depend on the change of memory usage. Our experimental data show that 2% absolute error tolerance makes a compromise between the two types of statistical errors.

To obtain the misdiagnosis probability of the M&A to the AS and OS, we need to record memory usage at the AS and OS levels. The misdiagnosis probability of the M&A to the AS under the condition that the $\mathrm{TrueState}_{\mathrm{AS}}(t)$ is robust is estimated by

$$1 - p_r = \frac{N_{m,\mathrm{AS},r}}{N_{t,\mathrm{AS},r}}$$

where $N_{m,\mathrm{AS},r}$ is the number of misdiagnosis times of the M&A to the AS and $N_{t,\mathrm{AS},r}$ is the diagnosis times when the real state of AS is robust. We use a preliminary experiment to get data of $N_{m,\text{AS},r}$ and $N_{t,\text{AS},r}$. In the experiments, the memory usage of the AS and the OS level is recorded about every 7.1 s till AS crashes. The time interval from the AS restarts to AS crashes is [0 s, 14 320 s]. In the time interval [0 s, 9341 s] ⊂ [0 sec, 14 320 s], the AS is at robust state, and in the time interval (9341 s, 14 320 s], the AS is at degradation state. When the AS is at robust state in this time interval [0 s, 9341 s], the diagnosis times when the AS is at robust state as 1315 units. According to the state definition of AS, we have right diagnosis times is 1310. So we obtained $N_{m,\text{AS},r} = 5$ and $N_{t,\text{AS},r} = 1315$. Then, the estimation of the misdiagnosis probability of the M&A to the AS under the condition of the AS being at robust state is

$$1 - p_r = \frac{5}{1315} = 0.0038$$

and the rightly diagnosis probability is 0.9962.

When AS is at degradation state in the time interval [9341 s, 14 320 s], the misdiagnosis probability of the M&A to AS is estimated by

$$1 - p_D = \frac{N_{m,\text{AS},D}}{N_{t,\text{AS},D}}$$

where $N_{m,\text{AS},D}$ is the number of misdiagnosis times of the M&A to the AS and $N_{t,\text{AS},D}$ is the total times of inspection. We obtain the estimation of the conditional misdiagnosis probability as

$$1 - p_D = \frac{6}{712} = 0.0085.$$

Similarly, we obtain the estimations of the misdiagnosis probabilities of the M&A to the OS when the real state of the OS is robust or degradation, respectively:

$$1 - q_r = \frac{905}{22\,789} = 0.0397$$

$$1 - q_D = \frac{346}{2543} = 0.1361.$$

Because the environment of these experiments to obtain the right diagnosis probability of the M&A are similar to the environment of the other experiments to obtain to the distributions of the random variables such as T_{A1}, T_{A2} in the following section, so we conclude that the right diagnosis probability of the M&A is the same as that in the experiments in Section VI.

Actually, the right diagnosis probability also depends on the environment, in which the AS and the M&A work. How to develop the right diagnosis probability of the M&A should be considered in the further research.

VI. Result Analysis

The degradation rate of a given software usually depends on the running environment, which involves the type of work and its respective intensity [25], [37]. To obtain the measurements of MTrD, and the mean time from degradation to failed state (MTDF) of AS and OS, respectively, a two-stage random number generating method is used. In the first stage, two series of random number values $(X_1, X_2, \ldots, X_6)$ and $(Y_1, Y_2, \ldots, Y_6)$ are generated following an exponential distribution with the parameter λ equaling to $1kb$, which are used as the average size of each memory leak injected at the AS level and the OS level in each experiment, respectively. We use $(X_1, X_2, \ldots, X_6)$ and $(Y_1, Y_2, \ldots, Y_6)$ to represent the complexity of different types of work the sever system provides, which makes the MTrD and MTDF exponentially distributed. In the second stage, in each experiment for the AS and the OS, the injected leak size of each visit follows a Poisson distribution, which represents the strength of the work. For example, in the ith experiment for the AS level, the injected leak size of each visit to the page TPC_W_search_request_page as well as TPC_W_shopping_cart_interaction follows the Poisson distribution Poisson(X_i) at the AS level, and the injected leak size of each visit to the TPC_W_search_request_page and TPC_W_order_status_Servlet also follow a Poisson distribution, Poisson(Y_i), at the OS level. We conducted a total of 12 experiments to obtain six samples of MTrD and six samples of MTDF for the AS and OS respectively, which are shown in Table III. The six experiments for the AS and that for the OS were executed independently.

In each experiment, we end the experiment run after the memory resource is exhausted; then, we obtained the MTrD and MTDF and the memory consumption rate in each experiment, which are shown in Table III. Therefore, we obtained the state transition rates of Table II, which are given in Table IV.

TABLE III
Memory Consumption Rate and TTF of AS and OS

Group	Amount of Memory Injection (kbyte)	AS From Health To Degradation (minute)	AS Experiment TTF (minute)	AS From Degradation To Dead (minute)	MCR of AS Level (byte/minute)
1 grp	1.94	168.1	252.8	84.7	413 226.630
2 grp	1.22	217.7	494.1	276.4	220 982.776
3 grp	1.32	91.1	186.9	95.8	579 525.056
4 grp	0.84	178.0	410.1	232.1	257 752.463
5 grp	0.70	417.6	596.7	179.1	145 538.356
6 grp	1.02	111.4	324.7	213.3	275 094.530
Group	Amount of Memory Injection (kbyte)	OS From Health To Degradation (minute)	OS Experiment TTF (minute)	OS From Degradation To Dead (minute)	MCR of OS Level (byte/minute)
1 grp	1.95	145.6	294.0	148.4	585 963.462
2 grp	0.22	527.7	1580.0	1052.3	102 752.829
3 grp	0.72	189.6	538.3	348.7	279 518.242
4 grp	0.044	1603.0	4204.0	2601.0	45 399.326
5 grp	1.85	148.2	296.5	148.3	625 204.966
6 grp	2.33	114.5	206.7	92.2	807 782.578

TABLE IV
Parameters Obtained from the Experiment

λ_{A1}	λ_{A2}	λ_{O1}
0.0051 /min	0.0055 /min	0.0022 /min
λ_{O2}	L_{rr}	L_{Dr}
0.0043 /min	0.0044	0.0196
L_{rD}	L_{DD}	
0.0178	0.0211	

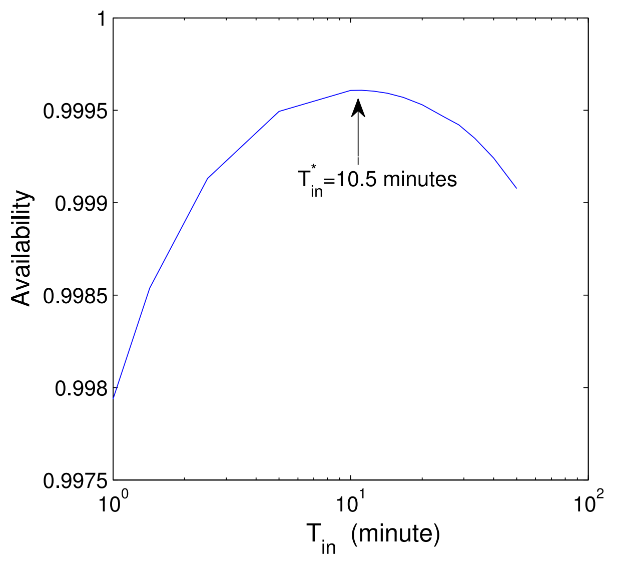


Fig. 10. Availability of the system versus the optimal inspection time interval.

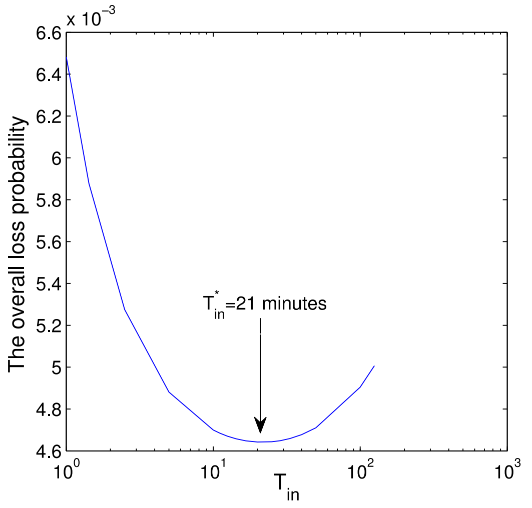


Fig. 11. Overall loss probability of the system versus the optimal inspection time interval.

In addition, we record the loss probabilities L_{rr}, L_{rD}, L_{Dr}, and L_{DD} when the system is at "rr," "rD," "Dr," and "DD" states (see Table IV).

Suppose that the average repair times of the AS are $\mathbb{E}(T_{A3}) = 0.5$ min and $\mathbb{E}(T_{A4}) = 1$ min. The average repair times of the OS are $\mathbb{E}(T_{O3}) = 1.5$ min and $\mathbb{E}(T_{O4}) = 2$ min, which are the common time for software maintenance [26]. Then, the availability of the system versus T_{in} is given in Fig. 10, where the optimal availability found was 0.99968 (which means that the unavailability is $1 - 0.99968 = 0.00032$) and the corresponding optimal inspection interval was $T_{\mathrm{in}}^* = 10.5$ min. The overall loss probability of the system versus T_{in} is given in Fig. 11. From the figure, we know that the minimum overall loss probability is 4.7461×10^{-3} and the corresponding optimal T_{in}^* for the overall loss probability in this case is 21 min.

The numerical results show that the percentage of rejuvenation at the AS level is 60.19%, and the percentage of rejuvenation at the OS level is $1 - 60.19\% = 39.81\%$.

To comparatively analyze the effects of two-granularity software rejuvenation and single-level software rejuvenation, we suppose that there is no AS proactive or reactive restart, which is the common case when the traditional single-level rejuvenation is used. In this scenario, if the AS is at aging state, we rejuvenate the OS level (not just the AS level); if the OS is at aging state, we also rejuvenate the OS level. Similarly, if the AS is at crash state, we reactively restart the OS level (not the AS level); if the OS is at crash state, we also reactively restart the OS level. In our model, we correspondingly replaced the RD and Rr states in Fig. 2 with RR and RR states. Then, the sojourn times of the corresponding new states have the distribution F_{O3}. The other states and their sojourn times did not change. In this case, numerical results show that the unavailability and overall loss probability of the system are 0.00039 and 4.8753×10^{-3}, respectively.

If two-granularity software rejuvenation is used, the unavailability of the system is reduced by

$$\frac{0.00039 - 0.00032}{0.00039} = 17.9\%$$

in comparison when the single-level rejuvenation is used. Similarly, the overall loss probability of the system is reduced by

$$\frac{4.8753 \times 10^{-3} - 4.7461 \times 10^{-3}}{4.8753 \times 10^{-3}} = 2.65\%.$$

VII. Threats to Validity

Every empirical study suffers from methodological shortcomings [31], [32]. Basically, the following four categories of validity are the most concerned issues by researchers in this respect.

1) *Conclusion validity:* Conclusion validity refers to the statistical relationship between the treatment and the outcome.

   In this paper, we use a preliminary experiment to obtain the misdiagnosis probability of the M&A. If the event $\text{MonitoredState}_{\mathrm{AS}}(t) \neq \text{TrueState}_{\mathrm{AS}}(t)$ is observed in an inspection, then we say that the M&A makes a misdiagnosis in this inspection. A threat to the misdiagnosis probability is that the event above has different rates during a state of the AS. For example, if the AS is at robust state, the event occurrence rate is smaller rate when the system has enough memory, and the event occurrence rate becomes larger when the time approaches the critical time t_{DA} which is defined previously in this paper. Besides, when repeating the preliminary experiment, deviation occurs to the misdiagnosis probability.

   Another threat to validity is that the independent variables such as TTF of the AS and the OS are not known to correlate substantially. Besides, the JVM garbage collection impacts on the memory usage measurement in the experiment, which makes the parameters such as λ_{DA} and λ_{DO} vary.

2) *Internal validity:* Internal validity refers to the relationship between the treatment and the outcome. A threat to the internal validity might result from the dependent variables in an experiment, especially when the researchers are not aware of the relationships among dependent variables. In this paper, we used memory usage as the indicator of software aging. There are some other software aging indicators such as CPU usage rate, reply time delay, and so on. If we use different aging indicators, then the variables such as T_{DA} and T_{DO} measured are different. In this case, an issue would be to select low quality aging indicators, that is, aging indicators with a low precision to capture the aging effects or even that show high false-positive or false-negative rates. A good source of discussion on this problem would be [38].

   Another weak threat to validity is the other AS on the OS. In this paper, we focus on the web server AS and OS, but they are more or less affected by the other supporting AS.
3) *External validity:* External validity is concerned with the generalizability of the results for the other people outside the investigated case.

   In this paper, we considered the case that the AS and the OS are independent by separating memory usage of the AS and the OS levels. Our method cannot be applied to the other case when the AS and the OS shares the memory usage. Thus, this is one threat to external validity.
4) *Construct validity:* Construct validity is about the extent to which the specific variables of an empirical study are consistent with the intended constructs in the theoretical model.

   Commonly investigated causes of software aging include but are not limited to memory leaks, unreleased file-locks, and round-off errors [2], [8]. Memory usage is one of the most used measurements of software aging [38]. This paper uses memory usage as the measurement of software aging. However, the other measurements cannot totally be ignored in a software system. They are potential threats to validity.

   The software aging-related bugs are the root causes of software aging. In this paper, software rejuvenation strategy was optimized to counteract the impacts of software aging-related bugs. Therefore, the bugs unrelated to software aging are threats to validity.

VIII. Conclusion

In this paper, we have discussed the two-level software aging behavior of software systems. The aim of this study is to apply an overall optimal rejuvenation strategy to the system that suffers from two-granularity software aging. According to the software states identified, four software maintenance techniques, AS proactive rejuvenation, OS proactive rejuvenation, AS reactive restart, and OS reactive restart, are used to counteract the negative impact of the software aging effects.

We have proposed a Markov regeneration process model based on the states identified in the Apache Tomcat server system. We have validated the analytical solution of the Markov regeneration process model. The aging behavior-related parameters of the software were obtained from controlled experiments. Therefore, the corresponding optimal inspection interval was given from the analytical solution.

The experimental results show that two-granularity software rejuvenation was much more effective to counteract the software aging than single (OS) level rejuvenation.

Note that multilevel rejuvenation can be applied only when we know which part (or level) the aging is at. So, the main limitation of this paper is the difficulty to locate the aged part (level) by using different metrics for software aging detection in real applications. While if the traditional single (OS) level rejuvenation is used, this limitation can be redeemed by applying the metric of system performance [38]. So, it is important to investigate appropriate metrics and mechanisms to measure the aging effects in different levels of computing systems, which will allow us to apply easier our proposal approach.

The diagnostic accuracy of the M&A has been considered when we gave the optimal rejuvenation scheduling in this paper. For this purpose, a full path method is used. How to improve the diagnostic accuracy of the M&A is an interesting extension of this study. With a fine-grained software observability, high dependable automatic software rejuvenation technique will be closer to real systems.

Appendix

The global kernel matrix $K(t)$:

For $k_{0,1}(t)$, there are two paths: $0 \to 1$, and $0 \to 8 \to 1$.

$$\begin{aligned} k_{0,1}(t) = {} & Pr\{Z(S_1) = 1, S_1 \le t \mid Z(0) = 0\} \\ = {} & \sum_{k=1}^{\lfloor \frac{t}{T_{\mathrm{in}}} \rfloor} (p_r q_r)^{k-1}(1-p_r) q_r e^{-(\lambda_{A1}+\lambda_{O1})kT_{\mathrm{in}}} \\ & + p_D q_r \sum_{k=1}^{\lfloor \frac{t}{T_{\mathrm{in}}} \rfloor} \int_0^{kT_{\mathrm{in}}} \lambda_{A1} e^{-(\lambda_{A1}+\lambda_{O1})x} (p_r q_r)^{\lfloor \frac{x}{T_{\mathrm{in}}} \rfloor} \\ & [(1-p_D) q_r]^{k-\lfloor \frac{x}{T_{\mathrm{in}}} \rfloor - 1} e^{-(\lambda_{A1}+\lambda_{O1})(kT_{\mathrm{in}}-x)} dx. \end{aligned}$$

For $k_{0,2}(t)$, there are three paths: $0 \to 7 \to 2$, $0 \to 7 \to 9 \to 2$ and $0 \to 8 \to 9 \to 2$. So

$$\begin{aligned} k_{0,2}(t) = {} & Pr\{Z(S_1) = 2, S_1 \le t \mid Z(0) = 0\} \\ = {} & (1-p_r)(1-q_D) \sum_{k=1}^{\lfloor \frac{t}{T_{\mathrm{in}}} \rfloor} \int_0^{kT_{\mathrm{in}}} \lambda_{O1} e^{-(\lambda_{A1}+\lambda_{O1})x} \\ & (p_r q_r)^{\lfloor \frac{x}{T_{\mathrm{in}}} \rfloor} [p_r(1-q_D)]^{k-\lfloor \frac{x}{T_{\mathrm{in}}} \rfloor - 1} \\ & e^{-(\lambda_{A1}+\lambda_{O1})(kT_{\mathrm{in}}-x)} dx \\ & + p_D(1-q_D) \sum_{k=1}^{\lfloor \frac{t}{T_{\mathrm{in}}} \rfloor} \int_0^{kT_{\mathrm{in}}} \int_x^{kT_{\mathrm{in}}} \lambda_{O1} e^{-(\lambda_{A1}+\lambda_{O1})x} \\ & \lambda_{A1} e^{-(\lambda_{A1}+\lambda_{O2})(y-x)} e^{-(\lambda_{A2}+\lambda_{O2})(kT_{\mathrm{in}}-y)} \\ & (p_r q_r)^{\lfloor \frac{x}{T_{\mathrm{in}}} \rfloor} [p_r(1-q_D)]^{\lfloor \frac{y}{T_{\mathrm{in}}} \rfloor - \lfloor \frac{x}{T_{\mathrm{in}}} \rfloor} \end{aligned}$$

$$[(1-p_D)(1-q_D)]^{k-\lfloor \frac{y}{T_{\mathrm{in}}}\rfloor-1}dydx$$
$$+p_D(1-q_D)\sum_{k=1}^{\lfloor \frac{t}{T_{\mathrm{in}}}\rfloor}\int_0^{kT_{\mathrm{in}}}\int_x^{kT_{\mathrm{in}}}\lambda_{A1}e^{-(\lambda_{A1}+\lambda_{O1})x}$$
$$\lambda_{O1}e^{-(\lambda_{O1}+\lambda_{A2})(y-x)}e^{-(\lambda_{A2}+\lambda_{O2})(kT_{\mathrm{in}}-y)}$$
$$(p_rq_r)^{\lfloor \frac{x}{T_{\mathrm{in}}}\rfloor}[(1-p_D)q_r]^{\lfloor \frac{y}{T_{\mathrm{in}}}\rfloor-\lfloor \frac{x}{T_{\mathrm{in}}}\rfloor}$$
$$[(1-p_D)(1-q_D)]^{k-\lfloor \frac{y}{T_{\mathrm{in}}}\rfloor-1}dydx.$$

For $k_{0,3}(t)$, there are five paths: $0\rightarrow 3, 0\rightarrow 7\rightarrow 3, 0\rightarrow 8\rightarrow 3$, $0\rightarrow 7\rightarrow 9\rightarrow 3$, and $0\rightarrow 8\rightarrow 9\rightarrow 3$. So

$$k_{0,3}(t) = Pr\{Z(S_1)=3, S_1\le t \mid Z(0)=0\}$$
$$= \sum_{k=1}^{\lfloor \frac{t}{T_{\mathrm{in}}}\rfloor} e^{-(\lambda_{A1}+\lambda_{O1})kT_{\mathrm{in}}}(p_rq_r)^{k-1}(1-q_r)$$
$$+q_D\sum_{k=1}^{\lfloor \frac{t}{T_{\mathrm{in}}}\rfloor}\int_0^{kT_{\mathrm{in}}}\lambda_{O1}e^{-(\lambda_{A1}+\lambda_{O1})x}$$
$$(p_rq_r)^{\lfloor \frac{x}{T_{\mathrm{in}}}\rfloor}[p_r(1-q_D)]^{k-\lfloor \frac{x}{T_{\mathrm{in}}}\rfloor-1}$$
$$e^{-(\lambda_{A1}+\lambda_{O2})(kT_{\mathrm{in}}-x)}dx$$
$$+(1-q_r)\sum_{k=1}^{\lfloor \frac{t}{T_{\mathrm{in}}}\rfloor}\int_0^{kT_{\mathrm{in}}}\lambda_{A1}e^{-(\lambda_{A1}+\lambda_{O1})x}$$
$$(p_rq_r)^{\lfloor \frac{x}{T_{\mathrm{in}}}\rfloor}[q_r(1-p_D)]^{k-\lfloor \frac{x}{T_{\mathrm{in}}}\rfloor-1}$$
$$e^{-(\lambda_{A2}+\lambda_{O1})(kT_{\mathrm{in}}-x)}dx$$
$$+q_D\sum_{k=1}^{\lfloor \frac{t}{T_{\mathrm{in}}}\rfloor}\int_0^{kT_{\mathrm{in}}}\int_x^{kT_{\mathrm{in}}}\lambda_{O1}e^{-(\lambda_{A1}+\lambda_{O1})x}$$
$$\lambda_{A1}e^{-(\lambda_{A1}+\lambda_{O2})(y-x)}e^{-(\lambda_{A2}+\lambda_{O2})(kT_{\mathrm{in}}-y)}$$
$$(p_rq_r)^{\lfloor \frac{x}{T_{\mathrm{in}}}\rfloor}[p_r(1-q_D)]^{\lfloor \frac{y}{T_{\mathrm{in}}}\rfloor-\lfloor \frac{x}{T_{\mathrm{in}}}\rfloor}$$
$$[(1-p_D)(1-q_D)]^{k-\lfloor \frac{y}{T_{\mathrm{in}}}\rfloor-1}dydx$$
$$+q_D\sum_{k=1}^{\lfloor \frac{t}{T_{\mathrm{in}}}\rfloor}\int_0^{kT_{\mathrm{in}}}\int_x^{kT_{\mathrm{in}}}\lambda_{A1}e^{-(\lambda_{A1}+\lambda_{O1})x}$$
$$\lambda_{O1}e^{-(\lambda_{O1}+\lambda_{A2})(y-x)}e^{-(\lambda_{A2}+\lambda_{O2})(kT_{\mathrm{in}}-y)}$$
$$(p_rq_r)^{\lfloor \frac{x}{T_{\mathrm{in}}}\rfloor}[(1-p_D)q_r]^{\lfloor \frac{y}{T_{\mathrm{in}}}\rfloor-\lfloor \frac{x}{T_{\mathrm{in}}}\rfloor}$$
$$[(1-p_D)(1-q_D)]^{k-\lfloor \frac{y}{T_{\mathrm{in}}}\rfloor-1}dydx.$$

For $k_{0,4}(t)$, there is one path: $0\rightarrow 8\rightarrow 4$. So

$$k_{0,4}(t)$$
$$= Pr\{Z(S_1)=4, S_1\le t \mid Z(0)=0\}$$
$$= \int_0^t\int_0^{t-x}\lambda_{A1}\lambda_{A2}e^{-(\lambda_{A1}+\lambda_{O1})x}e^{-(\lambda_{A2}+\lambda_{O1})y}$$
$$(p_rq_r)^{\lfloor \frac{x}{T_{\mathrm{in}}}\rfloor}[(1-p_D)q_r]^{\lfloor \frac{x+y}{T_{\mathrm{in}}}\rfloor-\lfloor \frac{x}{T_{\mathrm{in}}}\rfloor}dydx.$$

For $k_{0,5}(t)$, there are two paths: $0\rightarrow 8\rightarrow 9\rightarrow 5$, $0\rightarrow 7\rightarrow 9\rightarrow 5$. So

$$k_{0,5}(t)$$
$$= Pr\{Z(S_1)=5, S_1\le t \mid Z(0)=0\}$$
$$= \int_0^t\int_x^t\int_y^t\lambda_{O1}e^{-(\lambda_{A1}+\lambda_{O1})x}$$
$$\lambda_{A1}e^{-(\lambda_{A1}+\lambda_{O2})(y-x)}\lambda_{A2}e^{-(\lambda_{A2}+\lambda_{O2})(z-y)}$$
$$(p_rq_r)^{\lfloor \frac{x}{T_{\mathrm{in}}}\rfloor}[p_r(1-q_D)]^{\lfloor \frac{y}{T_{\mathrm{in}}}\rfloor-\lfloor \frac{x}{T_{\mathrm{in}}}\rfloor}$$
$$[(1-p_D)(1-q_D)]^{\lfloor \frac{z}{T_{\mathrm{in}}}\rfloor-\lfloor \frac{y}{T_{\mathrm{in}}}\rfloor}dzdydx$$
$$+\int_0^t\int_x^t\int_y^t\lambda_{A1}e^{-(\lambda_{A1}+\lambda_{O1})x}$$
$$\lambda_{O1}e^{-(\lambda_{A1}+\lambda_{O2})(y-x)}\lambda_{A2}e^{-(\lambda_{A2}+\lambda_{O2})(z-y)}$$
$$(p_rq_r)^{\lfloor \frac{x}{T_{\mathrm{in}}}\rfloor}[q_r(1-p_D)]^{\lfloor \frac{y}{T_{\mathrm{in}}}\rfloor-\lfloor \frac{x}{T_{\mathrm{in}}}\rfloor}$$
$$[(1-p_D)(1-q_D)]^{\lfloor \frac{z}{T_{\mathrm{in}}}-\frac{y}{T_{\mathrm{in}}}\rfloor}dzdydx.$$

For $k_{0,6}(t)$, there are three paths: $0\rightarrow 7\rightarrow 6, 0\rightarrow 7\rightarrow 9\rightarrow 6$, $0\rightarrow 8\rightarrow 9\rightarrow 6$.

So

$$k_{0,6}(t)$$
$$= Pr\{Z(S_1)=6, S_1\le t \mid Z(0)=0\}$$
$$= \int_0^t\int_0^{t-x}\lambda_{O1}\lambda_{O2}e^{-(\lambda_{A1}+\lambda_{O1})x}e^{-(\lambda_{A1}+\lambda_{O2})y}$$
$$(p_rq_r)^{\lfloor \frac{x}{T_{\mathrm{in}}}\rfloor}[(1-q_D)p_r]^{\lfloor \frac{x+y}{T_{\mathrm{in}}}\rfloor-\lfloor \frac{x}{T_{\mathrm{in}}}\rfloor}dydx$$

$$\int_0^t\int_x^t\int_y^t\lambda_{O1}e^{-(\lambda_{A1}+\lambda_{O1})x}$$
$$\lambda_{A1}e^{-(\lambda_{A1}+\lambda_{O2})(y-x)}\lambda_{O2}e^{-(\lambda_{A2}+\lambda_{O2})(z-y)}$$
$$(p_rq_r)^{\lfloor \frac{x}{T_{\mathrm{in}}}\rfloor}[p_r(1-q_D)]^{\lfloor \frac{y}{T_{\mathrm{in}}}\rfloor-\lfloor \frac{x}{T_{\mathrm{in}}}\rfloor}$$
$$[(1-p_D)(1-q_D)]^{\lfloor \frac{z}{T_{\mathrm{in}}}\rfloor-\lfloor \frac{y}{T_{\mathrm{in}}}\rfloor}dzdydx$$
$$+\int_0^t\int_x^t\int_y^t\lambda_{A1}e^{-(\lambda_{A1}+\lambda_{O1})x}$$
$$\lambda_{O1}e^{-(\lambda_{A1}+\lambda_{O2})(y-x)}\lambda_{O2}e^{-(\lambda_{A2}+\lambda_{O2})(z-y)}$$
$$(p_rq_r)^{\lfloor \frac{x}{T_{\mathrm{in}}}\rfloor}[q_r(1-p_D)]^{\lfloor \frac{y}{T_{\mathrm{in}}}\rfloor-\lfloor \frac{x}{T_{\mathrm{in}}}\rfloor}$$
$$[(1-p_D)(1-q_D)]^{\lfloor \frac{z}{T_{\mathrm{in}}}-\frac{y}{T_{\mathrm{in}}}\rfloor}dzdydx.$$

$k_{1,0}(t) = F_{A3}(t)$.

When $k_{2,3}(t)$ is considered, we need to note that there are two rings: $2\rightarrow 7\rightarrow 2$ and $2\rightarrow 7\rightarrow 9\rightarrow 2$. Let

$$F_{272}(t) = \sum_{k=1}^{\lfloor \frac{t}{T_{\mathrm{in}}}\rfloor} e^{-(\lambda_{A1}+\lambda_{O2})(kT_{\mathrm{in}})}[(1-p_r)(1-q_D)]$$
$$[p_r(1-q_D)]^{k-1}F_{A3}(t-kT_{\mathrm{in}}),$$

$$\begin{aligned}F_{2792}(t) = {} & \sum_{k=1}^{\lfloor \frac{t}{T_{\rm in}}\rfloor} \int_0^{kT_{\rm in}} [p_r(1-q_D)]^{\lfloor \frac{x}{T_{\rm in}}\rfloor}[p_D(1-q_D)] \\ & [(1-p_D)(1-q_D)]^{k-\lfloor \frac{x}{T_{\rm in}}\rfloor-1} e^{-(\lambda_{A1}+\lambda_{O2})(x)} \\ & \lambda_{A1} e^{-(\lambda_{A2}+\lambda_{O2})(kT_{\rm in}-x)} dx \cdot F_{A3}(t-kT_{\rm in}).\end{aligned}$$

Then, we have

$$\begin{aligned} & k_{2,3}(t) \\ = {} & Pr\{Z(S_1)=3, S_1 \le t \mid Z(0)=2\} \\ = {} & \sum_{N=0}^{+\infty}\sum_{k=0}^{N} C_N^k \int_0^t \int_0^{t-x} \sum_{m=0}^{\lfloor \frac{t-x-y}{T_{\rm in}}\rfloor-1} [p_r(1-q_D)]^m q_D \\ & e^{-(\lambda_{A1}+\lambda_{O2})(m+1)T_{\rm in}} dF_{A3}(y) dF_{272}^{*k} * F_{2792}^{*(N-k)}(x) \\ & + \sum_{N=0}^{+\infty}\sum_{k=0}^{N} C_N^k \int_0^t \int_0^{t-x} \int_0^{t-x-y} \lambda_{A1} e^{-(\lambda_{A1}+\lambda_{O2})(z)} \\ & [p_r(1-q_D)]^{\lfloor \frac{z}{T_{\rm in}}\rfloor} \sum_{m=0}^{\lfloor \frac{t-x-y}{T_{\rm in}}\rfloor-\lfloor \frac{z}{T_{\rm in}}\rfloor-1} [(1-p_D)(1-q_D)]^m \\ & e^{-(\lambda_{A2}+\lambda_{O2})(m+1)T_{\rm in}} q_D dz dF_{A3}(y) \\ & dF_{272}^{*k} * F_{2792}^{*(N-k)}(x). \\ & k_{2,5}(t) \\ = {} & Pr\{Z(S_1)=5, S_1 \le t \mid Z(0)=2\} \\ = {} & \sum_{N=0}^{+\infty}\sum_{k=0}^{N} C_N^k \int_0^t \int_0^{t-x} \int_0^{t-x-y} \int_0^{t-x-y-z} \\ & \lambda_{A1} e^{-(\lambda_{A1}+\lambda_{O2})(z)} \lambda_{A2} e^{-(\lambda_{A2}+\lambda_{O2})(u)} \\ & [p_r(1-q_D)]^{\lfloor \frac{z}{T_{\rm in}}\rfloor} [(1-p_D)(1-q_D)]^{\lfloor \frac{z+u}{T_{\rm in}}\rfloor-\lfloor \frac{z}{T_{\rm in}}\rfloor} \\ & du dz dF_{A3}(y) dF_{272}^{*k} * F_{2792}^{*(N-k)}(x). \\ & k_{2,6}(t) \\ = {} & \sum_{N=0}^{+\infty}\sum_{k=0}^{N} C_N^k \int_0^t \int_0^{t-x} \int_0^{t-x-y} \lambda_{O2} e^{-(\lambda_{A1}+\lambda_{O2})(z)} \\ & [p_r(1-q_D)]^{\lfloor \frac{z}{T_{\rm in}}\rfloor} dz dF_{A3}(y) dF_{272}^{*k} * F_{2792}^{*(N-k)}(x). \\ & + \sum_{N=0}^{+\infty}\sum_{k=0}^{N} C_N^k \int_0^t \int_0^{t-x} \int_0^{t-x-y} \int_0^{t-x-y-z} \\ & \lambda_{A1} e^{-(\lambda_{A1}+\lambda_{O2})(z)} \lambda_{O2} e^{-(\lambda_{A2}+\lambda_{O2})(u)} \\ & [p_r(1-q_D)]^{\lfloor \frac{z}{T_{\rm in}}\rfloor} [(1-p_D)(1-q_D)]^{\lfloor \frac{z+u}{T_{\rm in}}\rfloor-\lfloor \frac{z}{T_{\rm in}}\rfloor} \\ & du dz dF_{A3}(y) dF_{272}^{*k} * F_{2792}^{*(N-k)}(x). \end{aligned}$$

$$\begin{aligned} k_{3,0}(t) = {} & F_{O3}(t). \\ k_{4,0}(t) = {} & (1-q_r)F_{O3}(t) + q_r F_{A4}(t). \\ k_{5,0}(t) = {} & q_D F_{O3}(t). \end{aligned}$$

To obtain $k_{5,2}(t)$, we let

$$\begin{aligned} & F_{5795}(t) \\ = {} & (1-p_4) \int_0^t \int_0^{t-x} \int_0^{t-x-y} \lambda_{A1} e^{-(\lambda_{A1}+\lambda_{O2})y} \\ & [p_r(1-q_D)]^{\lfloor \frac{y}{T_{\rm in}}\rfloor} \lambda_{A2} e^{-(\lambda_{A2}+\lambda_{O2})z} \\ & [(1-p_D)(1-q_D)]^{\lfloor \frac{y+z}{T_{\rm in}}\rfloor-\lfloor \frac{y}{T_{\rm in}}\rfloor} dz dy dF_{A4}(x). \end{aligned}$$

Then

$$\begin{aligned} & k_{5,2}(t) \\ = {} & Pr\{Z(S_1)=2, S_1 \le t \mid Z(0)=5\} \\ = {} & \sum_{k=0}^{+\infty} \int_0^t \int_0^{t-x} \sum_{m=0}^{\lfloor \frac{t-x-y}{T_{\rm in}}\rfloor-1} [p_r(1-q_D)]^m \\ & e^{-(\lambda_{A1}+\lambda_{O2})[(m+1)T_{\rm in}]} [(1-p_r)(1-q_D)] \\ & dF_{A4}(y) dF_{5795}^{*k}(x) \\ & + \sum_{k=0}^{+\infty} \int_0^t \int_0^{t-x} \int_0^{t-x-y} \lambda_{A1} e^{-(\lambda_{A1}+\lambda_{O2})z} \\ & [p_r(1-q_D)]^{\lfloor \frac{z}{T_{\rm in}}\rfloor} \sum_{m=0}^{\lfloor \frac{t-x-y-z}{T_{\rm in}}\rfloor-1} [(1-p_D)(1-q_D)]^m \\ & e^{-(\lambda_{A2}+\lambda_{O2})[(m+1)T_{\rm in}]} [p_D(1-q_D)] \\ & dz dF_{A4}(y) dF_{5795}^{*k}(x). \\ & k_{5,3}(t) \\ = {} & Pr\{Z(S_1)=3, S_1 \le t \mid Z(0)=5\} \\ = {} & \sum_{k=0}^{+\infty} \int_0^t \int_0^{t-x} \sum_{m=0}^{\lfloor \frac{t-x-y}{T_{\rm in}}\rfloor-1} [p_r(1-q_D)]^m \\ & e^{-(\lambda_{A2}+\lambda_{O2})[(m+1)T_{\rm in}]} q_D dF_{A4}(y) dF_{5795}^{*k}(x) \\ = {} & \sum_{k=0}^{+\infty} \int_0^t \int_0^{t-x} \int_0^{t-x-y} \lambda_{A1} e^{-(\lambda_{A1}+\lambda_{O2})z} \\ & [p_r(1-q_D)]^{\lfloor \frac{z}{T_{\rm in}}\rfloor} \sum_{m=0}^{\lfloor \frac{t-x-y-z}{T_{\rm in}}\rfloor-1} [(1-p_D)(1-q_D)]^m \\ & \cdot e^{-(\lambda_{A2}+\lambda_{O2})[(m+1)T_{\rm in}]} q_D dz dF_{A4}(y) dF_{5795}^{*k}(x). \\ & k_{5,6}(t) \\ = {} & Pr\{Z(S_1)=6, S_1 \le t \mid Z(0)=5\} \\ = {} & \sum_{k=0}^{+\infty} \int_0^t \int_0^{t-x} \int_0^{t-x-y} \lambda_{O2} e^{-(\lambda_{A1}+\lambda_{O2})z} \\ & [p_r(1-q_D)]^{\lfloor \frac{z}{T_{\rm in}}\rfloor} dz dF_{A4}(y) dF_{5795}^{*k}(x) \\ & + \sum_{k=0}^{+\infty} \int_0^t \int_0^{t-x} \int_0^{t-x-y} \lambda_{A1} e^{-(\lambda_{A1}+\lambda_{O2})z} \end{aligned}$$

$$[p_r(1-q_D)]^{\lfloor \frac{z}{T_{\text{in}}}\rfloor} \sum_{m=0}^{\lfloor \frac{t-x-y-z}{T_{\text{in}}}\rfloor -1} [(1-p_D)(1-q_D)]^m$$
$$e^{-(\lambda_{A2}+\lambda_{O2})[(m+1)T_{\text{in}}]} dz dF_{A4}(y) dF_{5795}^{*k}(x).$$
$$k_{6,0}(t) = F_{A4}(t).$$

The local kernel matrix $E(t)$:

$$\begin{aligned}
E_{0,0}(t) &= e^{-(\lambda_{A1}+\lambda_{O1})t}[(1-F_{\text{in}}(t)) \\
&\quad + \sum_{k=1}^{+\infty}(p_r q_r)^k \int_0^t (1-F_{\text{in}}(t-x)) dF_{\text{in}}^{*k}(x)] \\
&= e^{-(\lambda_{A1}+\lambda_{O1})t}(p_r q_r)^{\lfloor \frac{t}{T_{\text{in}}}\rfloor}. \\
E_{1,1}(t) &= 1 - F_{A3}(t). \\
E_{2,2}(t) &= 1 - F_{A3}(t) + \sum_{N=1}^{+\infty}\sum_{k=0}^{N} C_N^k \\
&\quad \int_0^t (1-F_{A3}(t-x)) dF_{272}^{*k} * F_{2792}^{*(N-k)}(x). \\
E_{3,3}(t) &= 1 - F_{O3}(t). \\
E_{4,4}(t) &= 1 - F_{A4}(t). \\
E_{5,5}(t) &= [1 - q_D F_{O3}(t) - (1-q_D)F_{A4}(t)] \\
&\quad + \sum_{k=0}^{+\infty}\int_0^t [1 - q_D F_{O3}(t-x) \\
&\quad -(1-q_D)F_{A4}(t-x)] dF_{5795}^{*k}(x). \\
E_{6,6}(t) &= 1 - F_{O4}(t).
\end{aligned}$$

$$\begin{aligned}
&E_{0,7}(t) \\
&= \int_0^t \lambda_{O1} e^{-(\lambda_{A1}+\lambda_{O1})x} e^{-(\lambda_{A1}+\lambda_{O2})(t-x)} \\
&\quad (p_r q_r)^{\lfloor \frac{x}{T_{\text{in}}}\rfloor} [p_r(1-q_D)]^{\lfloor \frac{t}{T_{\text{in}}}\rfloor - \lfloor \frac{x}{T_{\text{in}}}\rfloor} dx. \\
&E_{0,8}(t) \\
&= \int_0^t \lambda_{A1} e^{-(\lambda_{A1}+\lambda_{O1})x} e^{-(\lambda_{A2}+\lambda_{O1})(t-x)} \\
&\quad (p_r q_r)^{\lfloor \frac{x}{T_{\text{in}}}\rfloor} [(1-p_D)q_r]^{\lfloor \frac{t}{T_{\text{in}}}\rfloor - \lfloor \frac{x}{T_{\text{in}}}\rfloor} dx. \\
&E_{0,9}(t) \\
&= \int_0^t \int_x^t \lambda_{O1} e^{-(\lambda_{A1}+\lambda_{O1})x} (p_r q_r)^{\lfloor \frac{x}{T_{\text{in}}}\rfloor} \\
&\quad \lambda_{A1} e^{-(\lambda_{A1}+\lambda_{O2})(y-x)} [p_r(1-q_D)]^{\lfloor \frac{y}{T_{\text{in}}}\rfloor - \lfloor \frac{x}{T_{\text{in}}}\rfloor} \\
&\quad e^{-(\lambda_{A2}+\lambda_{O2})(t-y)} \\
&\quad [(1-p_D)(1-q_D)]^{\lfloor \frac{t}{T_{\text{in}}}\rfloor - \lfloor \frac{y}{T_{\text{in}}}\rfloor} dy dx \\
&\quad + \int_0^t \int_x^t \lambda_{A1} e^{-(\lambda_{A1}+\lambda_{O1})x} (p_r q_r)^{\lfloor \frac{x}{T_{\text{in}}}\rfloor} \\
&\quad \lambda_{O1} e^{-(\lambda_{A1}+\lambda_{O2})(y-x)} [q_r(1-p_D)]^{\lfloor \frac{y}{T_{\text{in}}}\rfloor - \lfloor \frac{x}{T_{\text{in}}}\rfloor} \\
&\quad e^{-(\lambda_{A2}+\lambda_{O2})(t-y)} \\
&\quad [(1-p_D)(1-q_D)]^{\lfloor \frac{t}{T_{\text{in}}}\rfloor - \lfloor \frac{y}{T_{\text{in}}}\rfloor} dy dx. \\
&E_{2,7}(t) \\
&= \sum_{N=0}^{+\infty}\sum_{k=0}^{N} C_N^k \int_0^t \int_0^{t-x} e^{-(\lambda_{A1}+\lambda_{O2})(t-x-y)} \\
&\quad [p_r(1-q_D)]^{\lfloor \frac{t-x-y}{T_{\text{in}}}\rfloor} dF_{A3}(y) dF_{272}^{*k} * F_{2792}^{*(N-k)}(x). \\
&E_{2,9}(t) \\
&= \sum_{N=0}^{+\infty}\sum_{k=0}^{N} C_N^k \int_0^t \int_0^{t-x} \int_0^{t-x-y} \lambda_{A1} e^{-(\lambda_{A1}+\lambda_{O2})(z)} \\
&\quad e^{-(\lambda_{A2}+\lambda_{O2})(t-x-y-z)} [p_r(1-q_D)]^{\lfloor \frac{z}{T_{\text{in}}}\rfloor} \\
&\quad [(1-p_D)(1-q_D)]^{\lfloor \frac{t-x-y}{T_{\text{in}}}\rfloor - \lfloor \frac{z}{T_{\text{in}}}\rfloor} \\
&\quad dz dF_{A3}(y) dF_{272}^{*k} * F_{2792}^{*(N-k)}(x).
\end{aligned}$$

$$\begin{aligned}
E_{5,7}(t) &= \sum_{k=0}^{+\infty}\int_0^t \int_0^{t-x} (1-q_D)^{k+1} \\
&\quad e^{-(\lambda_{A1}+\lambda_{O2})(t-x-y)}(1-F_{\text{in}}(t-x-y)) \\
&\quad dF_{A4}(y) dF_{5795}^{*k}(x). \\
E_{5,9}(t) &= \sum_{k=0}^{+\infty}\int_0^t \int_0^{t-x} \int_0^{t-x-y} (1-q_D)^{k+1} \\
&\quad \lambda_{A1} e^{-(\lambda_{A1}+\lambda_{O2})z} e^{-(\lambda_{A2}+\lambda_{O2})(t-x-y-z)} \\
&\quad (1-F_{\text{in}}(t-x-y)) dz dF_{A4}(y) dF_{5795}^{*k}(x).
\end{aligned}$$

The mean sojourn time at state $i \in \Omega$:

$$\begin{aligned}
\mu_0 &= \int_0^{+\infty} E_{0,0}(t) + E_{0,7}(t) + E_{0,8}(t) + E_{0,9}(t) dt. \\
\mu_i &= \int_0^{+\infty} E_{i,i}(t) dt, \text{ for } i = 1,3,4,6. \\
\mu_2 &= \int_0^{+\infty} E_{2,2}(t) + E_{2,7}(t) + E_{2,9}(t) dt. \\
\mu_5 &= \int_0^{+\infty} E_{5,5}(t) + E_{5,7}(t) + E_{5,9}(t) dt.
\end{aligned}$$

References

[1] Y. Huang, C. Kintala, N. Kolettis, and N. D. Fulton, “Software rejuvenation: Analysis, module, and applications,” in *Proc. 25th Int. Symp. Fault-Tolerance Comput.*, 1995, pp. 381–390.

[2] M. Grottke, R. Matias, and K. S. Trivedi, “The fundamentals of software aging,” in *Proc. IEEE Int. Conf. Softw. Rel. Eng. Workshops.*, 2008, pp. 1–6.

[3] K. Vaidyanathan and K. S. Trivedi, “A comprehensive model for software rejuvenation,” *IEEE Trans. Dependable Secure Comput.*, vol. 2, no. 2, pp. 124–136, Apr.–Jun. 2005.

[4] J. Alonso, M. Grottke, A. Nikora, and K. S. Trivedi, “The nature of the times to flight software failure during space missions,” in *Proc. IEEE Int. Conf. Softw. Rel. Eng., Workshops*, 2012, pp. 331–340.

[5] C. Kintala, “Software rejuvenation in embedded systems,” *J. Autom., Lang. Combinatorics*, vol. 14, pp. 63–73, 2009.

[6] V. Castelli *et al.*, “Proactive management of software aging,” *IBM J. Res. Develop.*, vol. 45, pp. 311–332, 2001.

[7] W. Yurcik and D. Doss, “Achieving fault-tolerant software with rejuvenation and reconfiguration,” *IEEE Softw.*, vol. 18, no. 4, pp. 48–52, Jul./Aug. 2001.

[8] M. Grottke and K. S. Trivedi, “Fighting bugs: Remove, retry, replicate, and rejuvenate,” *Computer*, vol. 40, no. 2, pp. 107–109, Feb. 2007.

[9] G. Ning, K. S. Trivedi, H. Hu, and K.-Y. Cai, “Multi-granularity software rejuvenation policy based on continuous time Markov chain,” in *Proc. IEEE 3rd Int. Workshop Softw. Aging Rejuvenation*, 2011, pp. 32–37.

[10] T. Dohi, K. G. Popstojanova, and K. S. Trivedi, “Analysis of software cost models with rejuvenation,” in *Proc. Int. Symp. High Assurance Syst. Eng.*, 2000, pp. 25–34.

[11] T. Dohi, K. G. Popstojanova, and K. S. Trivedi, “Estimating software rejuvenation schedules in high assurance systems,” *Comput. J.*, vol. 44, pp. 473–485, 2001.

[12] W. Xie, Y. Hong, and K. S. Trivedi, “Analysis of a two-level software rejuvenation policy,” *Rel. Eng. Syst. Safety*, vol. 87, pp. 13–22, 2005.

[13] T. Dohi, K. G. Popstojanova, and K. S. Trivedi, “Statistical non-parametric algorithms to estimate the optimal software rejuvenation schedule,” in *Proc. Pacific Rim Int. Symp. Dependable Comput.*, 2000, pp. 77–84.

[14] Y. Bao, X. Sun, and K. S. Trivedi, “A workload-based analysis of software aging, and rejuvenation,” *IEEE Trans. Rel.*, vol. 54, no. 3, pp. 102–114, Sep. 2005.

[15] M. Grottke, L. Li, K. Vaidyanathan, and K. S. Trivedi, “Analysis of software aging in a web server,” *IEEE Trans. Rel.*, vol. 55, no. 3, pp. 411–420, Sep. 2006.

[16] Y. Hong, D. Chen, L. Li, and K. S. Trivedi, “Closed loop design for software rejuvenation,” presented at the Workshop Self-Healing, Adaptive Self-Managed Syst., New York, NY, USA, 2002.

[17] W. Qin and Q. Wang, “Feedback performance control for computer systems: An LPV approach,” in *Proc. Amer. Control Conf.*, 2005, pp. 4760–4765.

[18] A. Bobbio, A. Sereno, and C. Anglano, “Fine grained software degradation models for optimal rejuvenation policies,” *Perform. Eval.*, vol. 46, pp. 45–62, 2001.

[19] S. Garg, Y. Huang, C. Kintala, and K. S. Trivedi, “Time and load based software rejuvenation: Policy, evaluation and optimality,” in *Proc. 1st Fault-Tolerant Symp.*, 1995, pp. 22–25.

[20] S. Garg, A. Van Moorsel, K. Vaidyanathan, and K. S. Trivedi, “A methodology for detection and estimation of software aging,” in *Proc. 9th Int. Symp. Softw. Rel. Eng.*, 1998, pp. 283–292.

[21] M. Shereshevsky, J. Crowell, B. Cukic, V. Gandikota, and Y. Liu, “Software aging and multifractality of memory resources,” in *Proc. Int. Conf. Dependable Syst. Netw.*, 2003, pp. 721–730.

[22] Y. Jia, X. Chen, and K. -Y. Cai, “Chaotic analysis of software aging in a web server,” in *Proc. 2th Int. Workshop Service-Oriented Syst. Eng.*, 2006, pp. 117–120.

[23] K. Xue, L. Su, Y. Jia, and K. -Y. Cai, “A neural network approach to forecasting computing-resource exhaustion with workload,” in *Proc. 9th Int. Conf. Quality Softw.*, 2009, pp. 315–324.

[24] K. Vaidyanathan and K. S. Trivedi, “A comprehensive model for software rejuvenation,” *IEEE Trans. Dependable Secure Comput.*, vol. 2, no. 2, pp. 124–136, Apr.–Jun. 2005.

[25] R. Matias and P. J. Freitas Filho, “An experimental study on software aging and rejuvenation in Web servers,” in *Proc. 30th Annu. Int. Comput. Softw. Appl. Conf.*, pp. 189–196, vol. 1, 2006.

[26] J. Alonso, R. Matias, E. Vicente, A. Maria, and K. S. Trivedi, “A comparative experimental study of software rejuvenation overhead,” *Perform. Eval.*, vol. 70, pp. 231–250, 2013.

[27] Linux. Player. c, Linux memory leak detection tool, [EB/OL]. [Online]. Available: http://www.linuxidc.com/Linux/2014-09/106299.htm

[28] Frederic Germain,Trace and analyze memory leaks in C++ programs, [EB/OL]. [Online]. Available: http://www.andreasen.org/LeakTracer/

[29] Compuware Corporation, Compuware Ships DriverStudio 3.0, [EB/OL]. [Online]. Available: http://www.prnewswire.com/news-releases/compuware-improves-the-quality-of-enterprise-java-applications-with-devpartner-java-edition-33-54093837.html

[30] C. Wohlin, P. Runeson, M. Höst, M. Ohlsson, B. Regnell, and A. Wesséln, *Experimentation in Software Engineering*. New York, NY, USA: Springer, 2012.

[31] F. Huang, B. Liu, Y. Song, and S. Keyal, “The links between human error diversity and software diversity: Implications for fault diversity seeking,” *Sci. Comput. Program.*, vol. 89, pp. 350–373, 2014.

[32] F. Huang, B. Liu, S. Wang, and Q. Li, “The impact of software process consistency on residual defects,” *J. Softw.: Evol. Process*, vol. 27, 2015.

[33] A. Kumar and M. Saini, “Cost-benefit analysis of a single-unit system with preventive maintenance and Weibull distribution for failure and repair activities,” *J. Appl. Math., Statist. Informat.*, vol. 10, pp. 5–19, 2014.

[34] K. Vaidyanathan, D. Selvamuthu, and K. S. Trivedi, “Analysis of inspection-based preventive maintenance in operational software systems,” in *Proc. 21st Int. Symp. Rel. Distrib. Syst.*, 2002, pp. 286–295.

[35] J. Alonso, J. Torres, J. Berral, and R. Gavalda, “Adaptive on-line software aging prediction based on machine learning,” in *Proc. IEEE/IFIP Int. Conf. Dependable Syst. Netw.*, 2010, pp. 507–516.

[36] J. Zhao, Y. Wang, G. Ning, K. S. Trivedi, R. Matias, and K. -Y. Cai, “A comprehensive approach to optimal software rejuvenation,” *Perform. Eval.*, vol. 70, pp. 917–933, 2013.

[37] R. Matias, P. A. Barbetta, and K. S. Trivedi, “Accelerated degradation tests applied to software aging experiments,” *IEEE Trans. Rel.*, vol. 59, no. 1, pp. 102–114, Mar. 2010.

[38] R. Matias, A. Andrzejak, F. Machida, D. Elias, and K. Trivedi, “A systematic differential analysis for fast and robust detection of software aging,” in *Proc. 33rd IEEE Int. Symp. Rel. Distrib. Syst.*, 2014, pp. 311–320.

[39] S. Garg, A. Puliafito, M. Telek, and K. S. Trivedi, “Analysis of preventive maintenance in transactions based software systems,” *IEEE Trans. Comput.*, vol. 47, no. 1, pp. 96–107, Jan. 1998.

[40] M. Lipasti, Tpc-W Java version, [EB/OL]. (2009). [Online]. Available: http://www.ece.wisc.edu/ pharm/

[41] Oracle, Understanding Memory Management, [EB/OL]. (2010). [Online]. Available: http://docs.oracle.com/cd/E13150_01/jrockit_jvm/jrockit/geninfo/diagn os/garbage_collect.html

Gaorong Ning received the Ph.D. degree in navigation guidance and control from the Department of Automatic Control, Beihang University, Beijing, China, in 2016.

His main research interests include software reliability engineering and network information warfare.

Jing Zhao received the Ph.D. degree in computer science and technology from the Harbin Institute of Technology of China, Harbin, China, in 2006.

In 2010, she was with the Department of Electrical and Computer Engineering, Duke University, Durham, NC, USA, as a Postdoctoral Researcher under the supervision of Dr. K. Trivedi. She is currently a Professor with the School of Computer Science, Harbin Engineering University of China. Her research interests include reliability engineering, software aging theory, and dependability modeling.

Yunlong Lou is currently working toward the M.S. degree in the Department of Computer Science and Technology, Harbin Institute of Technology of China, Harbin, China.

His research interests include software reliability and network information warfare.

Javier Alonso received the master’s degree in computer science and Ph.D. degree from the Technical University of Catalonia (Universitat Politecnica de Catalunya, UPC), Barcelona, Spain, in 2004 and 2011, respectively.

From 2006 to 2011, he was an Assistant Lecturer with the Computer Architecture Department, UPC. From 2011 to 2014, he was a Postdoctoral Associate under the mentoring of Prof. K. S. Trivedi, with the Department of Electrical and Computer Engineering, Duke University, Durham, NC, USA. He is currently the Research Manager and Acting Research Director with the Research Institute of Applied Sciences in Cybersecurity, University of León, León, Spain. He is also a Visiting Assistant Professor with Duke University. His research interests include software engineering focusing on high-performance and high-available large-scale distributed software systems as well as mobile/cloud computing.

Rivalino Matias, Jr., received the B.S. degree in informatics from Minas Gerais State University, Belo Horizonte, Brazil, in 1994, and the M.S. and Ph.D. degrees in computer science, and industrial and systems engineering from the Federal University of Santa Catarina, Santa Catarina, Brazil, in 1997 and 2006, respectively.

He is currently an Associate Professor with the Computer School, Federal University of Uberlândia, Uberlândia, Brazil. His research interests include dependability applied to computing systems, software aging theory, and diagnosis protocols for computing systems.

Kishor S. Trivedi (M'86–SM'87–F'92) received the B.S.E.E. degree from Indian Institute of Technology, India, in 1968, and the M.S. and Ph.D. degrees from the University of Illinois in 1972 and 1974, respectively.

Dr. Trivedi holds the Hudson Chair with the Department of Electrical and Computer Engineering, Duke University, Durham, NC, USA. He has been on the Duke faculty since 1975. He is the author of a well-known book entitled *Probability and Statistics with Reliability, Queuing and Computer Science Applications* (Prentice-Hall, 1982); a thoroughly revised second edition (including its Indian edition) of this book has been published by Wiley. He has also published two other books entitled *Performance and Reliability Analysis of Computer Systems* (Kluwer, 1996) and *Queueing Networks and Markov Chains* (Wiley, 1998). His research interests include reliability, availability, performance, performability, and survivability modeling of computer and communication systems.

Dr. Trivedi is a Golden Core Member of the IEEE Computer Society. He received the IEEE Computer Society Technical Achievement Award for his research on software aging and rejuvenation.

Bei-Bei Yin received the Ph.D. degree from Beihang University (Beijing University of Aeronautics and Astronautics), Beijing, China, in 2010.

She has been a Lecturer with Beihang University since 2010. Her main research interests include software testing, software reliability, and software cybernetics.

Kai-Yuan Cai received the B.S., M.S., and Ph.D. degrees from Beihang University (Beijing University of Aeronautics and Astronautics), Beijing, China, in 1984, 1987, and 1991, respectively.

He has been a Full Professor at Beihang University since 1995. He is a Cheung Kong Scholar (Chair Professor), jointly appointed by the Ministry of Education of China and the Li Ka Shing Foundation of Hong Kong in 1999. His main research interests include software testing, software reliability, reliable flight control, and software cybernetics.