Performance Evaluation 70 (2013) 917–933

Contents lists available at ScienceDirect

Performance Evaluation

journal homepage: www.elsevier.com/locate/peva

A comprehensive approach to optimal software rejuvenation

Jing Zhao [a,d,*], YanBin Wang [b], GaoRong Ning [c], Kishor S. Trivedi [d], Rivalino Matias Jr. [e], Kai-Yuan Cai [c]

[a] Computer Science and Tech. Department, Harbin Engineering University, Harbin, China
[b] Industrial Engineering Department, Harbin Institute of Technology, Harbin, China
[c] Department of Automatic control, BeiJing University of Aeronautics and Astronautics, BeiJing, China
[d] Electrical and Computer Eng. Department, Duke University, Durham, NC 27705, USA
[e] School of Computer Science, Federal University of Uberlandia, Uberlandia, Brazil

ARTICLE INFO

Article history:
Received 22 May 2012
Accepted 22 May 2013
Available online 16 July 2013

Keywords:
Accelerated life tests
Memory leaks
Non-parametric estimation
Optimal software rejuvenation
Semi-Markov process
Importance sampling

ABSTRACT

Software aging is caused by resource exhaustion and can lead to progressive performance degradation or result in a crash. We develop experiments that simulate an on-line bookstore application, using the standard configuration of TPC-W benchmark. We study application failures due to memory leaks, using the accelerated life testing (ALT). ALT significantly reduces the time needed to estimate the time to failure at normal level. We then select the Weibull time to failure distribution at normal level, to be used in a semi-Markov model so as to optimize the software rejuvenation trigger interval. Then we derive the optimal rejuvenation schedule interval by fixed point iteration and by an alternative non-parametric estimation algorithm. Finally, we develop a simulation model using importance sampling (IS) to cross validate the ALT experimental results and the semi-Markov model, and also we apply the non-parametric method to cross validate the optimized trigger intervals by comparing the availabilities obtained from the semi-Markov model and those from IS simulation using the non-parametric method.

1. Introduction

It is well known that failures of a computer system are more often due to software faults than due to hardware faults [1]. Software faults have been classified into three types according to potential manifestation characteristics: Bohrbugs, Mandelbugs, and aging-related bugs [2]. Aging-related bugs cause an increasing failure rate, gradual software performance degradation, and may eventually lead to a system hang or crash. Software aging is caused by the successive accumulation of the effects of aging-related fault activations. It leads to the exhaustion of system resources, e.g., due to memory leaks, unreleased locks, non-terminated threads, shared-memory pool latching, storage fragmentation, or similar causes [3–5]. This undesired phenomenon occurs not only in web and application servers, but also in critical applications that require high dependability. Software aging can cause great losses in safety-critical systems [6], including the loss of human lives [7]. To counteract software aging, a proactive technique called software rejuvenation (SR) has been proposed [4]. Rejuvenation has been implemented in various computing systems, such as billing data collection systems, telecommunication systems, transaction processing systems, and spacecraft systems [8–10]. It involves periodically terminating an application process, cleaning its internal state and restarting it in order to release system resources, so that the software performance is

* Corresponding author at: Computer Science and Tech. Department, Harbin Engineering University, Harbin, China.
E-mail addresses: jingzhao.duke@gmail.com (J. Zhao), wangyb9988@gmail.com (Y. Wang), ninggaorong@asee.buaa.edu.cn (G. Ning), kst@ee.duke.edu (K.S. Trivedi), rivalino@fc.ufu.br (R. Matias Jr.), kycai@buaa.edu.cn (K.-Y. Cai).

0166-5316/$ – see front matter © 2013 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.peva.2013.05.010

recovered. Several indicators of aging can capture the aging behavior [3,1,11]. Such indicators are measurable metrics of the target system likely to be influenced by software aging.

The phenomena of software aging are experimentally studied using two approaches: Non-Accelerated or Accelerated testing. The both involve monitoring software aging indicators. Most of the previous experimental research on software aging and rejuvenation employed the Apache web server as a test bed [12], and then used statistical methods to predict the time to resource exhaustion [13,4,14,15]. Others used Axis [16,15]. Accelerated testing consists of a variety of test methods for shortening the life of products, and the aim of such testing is to quickly obtain data for statistical analysis. Such tests save much time and money [17]. Accelerated testing has been used extensively in hardware life testing. A major difference between software faults and hardware faults is that software behavior is complex since the activation of software faults is affected by environmental factors such as concurrency, inferencing memory pool, etc.. Accelerated testing used in software is relatively new. Due to the difficulty in experimentally studying aging-related system failures by observation of failure times, recent studies have used the accelerated Life test(ALT) or accelerated degradation tests (ADT) to experimentally inject faults, and then obtain the time to failures (TTFs) at accelerated stress levels. Matias et al. developed a systematic approach to accelerate the aging effects at the experimental level [18]. They introduced the concept of aging factors and use different levels of accelerated workload to increase the system degradation. Based on the degradation data of selected system characteristics, captured through measurements, they apply the statistical technique of accelerated degradation tests (ADT) to estimate the time to failure in normal condition (without acceleration). In [11] the authors do not use degradation data, but directly observe failures obtained also under accelerated workloads. In this case, they use another technique called accelerated life tests (ALT) to estimate the time to failure in normal conditions. In both studies the system under test was based on the Apache web server. Zhao et al. develop experiments that simulate an on-line bookstore, following the standard TPC-W benchmark [19], to inject memory leaks to intensify memory consumption so as to accelerate application failures [20].

Most analytic models used for capturing software rejuvenation are based on the assumption that the distribution of time to failure due to software aging is known, and the aim is to determine the optimal times to trigger rejuvenation in order to maximize system availability or related measures [3,21,16]. The exception is Dohi's paper [22] that uses a non-parametric method of optimal rejuvenation scheduling. Whatever approach is used for rejuvenation scheduling, such as measurement based, analytic, or both, estimated time to failure should be obtained more efficiently. Accelerated testing can be used for estimating TTF due to software aging to reduce the time and costs. Hence we focus on using the ALT method to obtain the TTF to solve for the optimal rejuvenation problem.

Memory leaks are recognized to be one of the major software-aging causes leading to resource exhaustion problems in complex software. In [23], the authors focus on two types of memory problems (fragmentation and leakage) that cause software aging, presenting an experimental study on the cumulative effect of these problems in software systems [23,24]. Alonso et al. [25] inject memory leaks to intensify memory consumption to derive the nonlinear memory resource behavior, and then use machine-learning algorithms to predict whether software aging has reached a given threshold. The memory consumption rate is used to represent the accelerated stress level in our experiments [20]. Memory consumption rate may be affected by workload or input to the system, but it is reasonable to consider and employ the long term average memory consumption rate to represent the acceleration factor. In our experiments, we inject memory leaks to the test bed by explicitly appending objects, which cannot be recycled by the garbage collector, to obtain the acceleration factors denoted by average memory consumption rate. We then derive the estimate of time to failure (TTF) at different acceleration levels as well as in normal condition. Such an estimate is then used in an analytic semi-Markov model to determine the optimal rejuvenation trigger interval [20].

We estimate the optimal rejuvenation schedule by parametric and non-parametric methods. Most of the papers use the parametric estimation to obtain the optimal rejuvenation schedule [3,11,18,26]. The non-parametric estimation is a distribution free and dynamic method that does not rely on assumptions that the data is from a given probability distribution. This approach has been used to estimate the optimal software rejuvenation schedule by Dohi et al. [22]. However, to the best of our knowledge, no research paper has applied both parametric and non-parametric method to estimate the optimal rejuvenation schedule.

We further develop a detailed simulation model using the importance sampling (IS) technique of the ALT experiment. IS could greatly reduce the simulation time to obtain the Mean time to Failures (MTTFs) as well as the availabilities in the semi-Markov process (SMP) model. We cross validate among ALT experiments, simulation and SMP model. In the first phase we cross validate between ALT experiments and simulation, we obtain the TTF results at different accelerated levels from the simulation model and thence we obtain the non-accelerated TTFs. We then cross-validate the results of the simulation with measurements from the experiment. In the second phase, we cross validate the availability results estimated from the simulation model with rejuvenation, the availabilities estimated using non-parametric method from the simulation, and the semi-Markov availability model. We then go on to use the analytic semi-Markov model to determine the optimal rejuvenation trigger interval.

The main contributions of this paper are as follows. First, we combine experimentation, statistical analysis using ALT, probabilistic (semi-Markov) models, optimization, IS and simulation in a single effort. Second, we employ the parametric with non-parametric method to estimate the optimal software rejuvenation schedule. Third, we employ the IS in simulation, which greatly reduces the simulation time. Fourth, we apply the non-parametric method to estimate the availabilities as well as the optimized trigger intervals using the data sets derived by IS. Finally, we cross validate among the ALT experiment, simulation using IS, and SMP model.

The rest of the paper is organized as follows. In Section 2 we show how to use ALT in systems suffering from software aging. In Section 3, the experimental setup and data collection are explained, where we describe how memory leak is injected to derive the system TTF samples at different acceleration levels. In Section 4, we explain the use of the Weibull time to failure distribution along with a semi-Markov process model in order to optimize the software rejuvenation trigger interval with the system availability and operational cost as objective functions. In Section 5, fixed point iteration is used to numerically compute the optimal software rejuvenation trigger interval based on the parametric method of Section 4. Then the non-parametric method is used to calculate the optimal rejuvenation trigger interval. Subsequently the results of the parametric and the non-parametric methods are explained. In Section 6, we develop the simulation model using IS to cross validate the ALT experiment and the SMP model. Thence we apply the non-parametric method to estimate the optimized trigger intervals using data sets obtained from IS simulations, to cross validate the semi-Markov availability model. Finally, we present our conclusions in Section 7.

2. ALT method for software aging

Accelerated life tests (ALT) have been successfully applied in many engineering fields [17] to significantly reduce the experimentation time, in quantifying the life characteristics (e.g., mean time to failure) of a system under test (SUT). By applying controlled stresses to reduce the SUTs lifetime, the SUT is tested in an accelerated mode, and results are then adjusted to its normal operational condition. Thus, ALT uses the lifetime data obtained under accelerated stresses to estimate the lifetime distribution of the SUT for its normal condition. This systematic approach can be divided into four main steps: (1) selection of accelerating stress, (2) ALT planning and execution, (3) definition of the life-stress aging relationship, and (4) estimation of underlying life distribution or the pdf for the normal condition. The following sections will discuss each step in detail.

2.1. Selection of accelerating stress

A fundamental element during test planning is the definition of accelerating stress variable and its levels. Typical engineering accelerating stresses are temperature, vibration, humidity, voltage, and thermal cycling [17]. However, software reliability engineering does not have standards related to software accelerating stresses for ALT. Given the nature of aging related faults, we can determine suitable accelerating stresses based on experiments. Based on [18], we employ memory consumption rate as the stress factor and use constant stress loading scheme in this paper. We randomly inject memory leaks into a web server software to intensify its memory consumption rate.

2.2. ALT planning and execution

After selecting the acceleration factor, we can plan the ALT. This activity includes the following elements: number of stress levels, the amount of stress applied at each level, the allocation proportion in each level, and the sample size. In our approach to apply ALT for software components, the sample size is the number of test replications. According to the theory, the ALT test plans can be classified as: traditional, optimal, and compromise plans [17]. The traditional plans usually consist of three or four stress levels, with the same number of replications allocated at each level. The optimal plans specify only two levels of stress, high and low. The compromise plans usually work with three or four stress levels, and use an unequal allocation proportion. A more detailed description of the three plans can be found in [17]. In our approach, the traditional plan with four levels is used.

2.3. Life-stress aging relationship

Once the SUT is tested at the selected stress levels, the estimate of the mean time to failure (MTTF) at normal condition is to be obtained from the TTF samples obtained at different stress levels. Therefore, we need to build the relationship between life-stress at accelerated and normal levels. As an example, consider the life-stress model that is known as the Inverse Power Law (IPL) [17]:

$$L(s) = \frac{1}{k \cdot s^w} \tag{1}$$

where L represents an SUT life characteristic (e.g., mean time to failure), s is the stress level, k ($k > 0$) and w are model parameters to be determined from the observed failure time samples.

2.4. Lifetime distribution estimation

Assuming that the TTF sample is exponentially distributed, IPL yields the pdf of TTF as:

$$f(t, s) = ks^w e^{-ks^w t}. \tag{2}$$

The maximum-likelihood estimation (MLE) method can be applied to estimate the model parameters (k, w), and then use them to estimate the MTTF, i.e., $L(s)$, for the SUT under normal stress level.

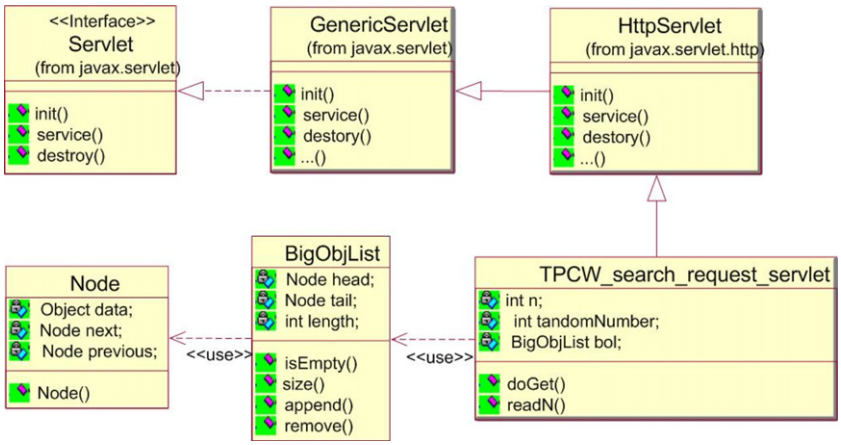


Fig. 1. Java servlet class.

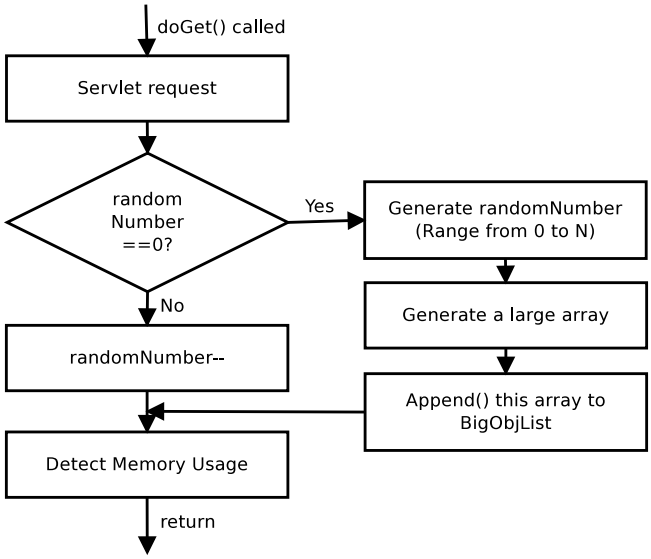


Fig. 2. Modification of *doGet()*.

3. Experimental study

The ALT experiment used in this paper is the same as in [20]. We study the aging effects of application failures caused by memory leaks based on a multitier e-commerce web site that simulates an on-line bookstore [19].

3.1. Injecting memory leaks

Our experiments use shopping transactions workload only [27]. We have modified the TPC-W implementation by changing the TPCW_search_request_servlet to inject memory leaks. The servlet class relationship including TPCW_search_request_servlet is shown in Fig. 1. Furthermore, we add a piece of code to the servlet so as to modify the *doGet()* method inside TPCW_search_request_servlet. The *doGet()* modification is illustrated in Fig. 2. A random number from 0 to *N* is generated, where *N* is specified in a configuration file. The *randomNumber* value determines how many requests can use the servlet before the next memory leak is injected. This number is decreased by one on each invocation of *doGet()*, i.e., on every visit of the search request page. Since the memory consumption rate would depend on the value of *N*, we can simulate this effect by varying *N*. Based on failure time samples collected under different stress loadings, the estimate of the time to failure distributions at different acceleration levels as well as at normal condition are obtained.

A JVM monitoring tool, jmap [28], is used to collect the JVM memory exhaustion data. We collect the usage of Young plus Old heaps used, at each five second interval. We also collect runtime memory used by JVM from the servlet perspective. Each TTF sample is the time from the beginning of a test to the time when the server's memory is exhausted. Then we use the runtime memory usage data to calculate the TTFs.

Table 1
Memory consumption rate and stress levels.

Memory consumption rate (kB/s) 0.0124	*N*	Memory consumption rate per replication *Normal level*
149.61	4	146.86, 149.22, 149.54, 157.04, 136.2, 153.44, 154.94
82.518	8	64.892, 84.042, 90.645, 88.752, 82.509, 85.847, 80.943
58.132	12	55.331, 58.284, 56.496, 59.213, 56.084, 55.148, 66.368
47.321	16	43.256, 48.649, 46.918, 50.081, 48.227,44.348, 49.768

Table 2
Sample of failure times (seconds).

TTF(S1)	TTF(S2)	TTF(S3)	TTF(S4)
731.101	1322.731	1769.281	2071.165
737.962	1342.926	1770.842	2188.907
738.284	1360.289	1898.677	2202.236
755.586	1382.329	1994.836	2266.199
764.761	1394.484	2014.693	2443.743
766.159	1419.333	2018.450	2558.261
862.257	1476.640	2098.482	2609.083

3.2. Analysis of experimental results

In our experiment, we design four acceleration levels (S1–S4) with N equal to 4, 8, 12, and 16, for each level, respectively. We run 7 replications at each acceleration level, thus 28 samples are obtained in total. Based on the experiment results, we calculate the mean memory consumption rate from runtime memory used, at each acceleration level. For each replication, the memory consumption rate is calculated using the Sen's slope estimate method [29]. These results are shown in Table 1. Next, we conduct the experiment removing acceleration factors so as to calculate the memory consumption rate at normal level. We observe that when the workload is equal to 100 Emulated Browsers (EBs), the total experimental time is 398,790 s, or 4.615625 days. The memory consumption rate of Young and Old heaps is approximately 0.0124 kB/s using Sen's slope method. The samples of failure times at each acceleration level are shown in Table 2.

The next step is to select the Lifetime distribution. The most used probability distributions in ALT experiments are from the location-scale family [30]. Examples of distributions from this family are Normal, Weibull, Lognormal, Logistic, LogLogistic, and Extreme Value distributions. Location-scale distributions have an important property in analyzing data from accelerated life tests, which is related to their cumulative distribution function (cdf). A random variable Y belongs to a location-scale family of distributions if its cdf can be written as:

$$P_r(Y \leq y) = F(y; \mu, \sigma) = \Phi\left(\frac{y-\mu}{\sigma}\right), \tag{3}$$

where $-\infty < \mu < \infty$ is a location parameter, $\sigma > 0$ is a scale parameter, and Φ does not depend on any unknown parameters. Appropriate substitution [30] shows that Φ is the cdf of $(Y-\mu)/\sigma$ when $\mu = 0$ and $\sigma = 1$. The importance of this family of distributions for ALT is due to the assumption that the location parameter, in (3), depends on the stress variable, s, that is $\mu(s)$, and the scale parameter, σ, is independent of s. This relationship is shown in (4).

$$Y = \log(t) = \mu(s) + \sigma\varepsilon, \tag{4}$$

where t is the time to failure, and ε is a probabilistic component modeling the time to failure sample variability. Essentially, we have a location-scale regression model to describe the effect that the explanatory variable, s, has on the time to failure. In this work, we evaluate density functions from location-scale family of distributions, and assume their scale parameter approximately constant (within the same CI) across the stress levels. According to [18], we test the probability distributions Weibull, Lognormal, and Exponential to identify the best fit. The criterion used to build the best-fit ranking is the log-likelihood function (Lk) [31]. The fitting results for these three models are shown in Table 3. They are consistent across all acceleration levels. The Lognormal distribution provides the best fitting results for the four acceleration levels S1–S4, followed closely by the Weibull distribution. Since Lognormal and Weibull presented very close results, we chose Weibull density to employ the preventive maintenance analytical approach presented by Chen and Trivedi in [26], which is based on Weibull distribution. We also chose Weibull with the IPL model to create our life-stress relationship as in the following section.

4. Optimal software rejuvenation

Based on the results discussed in Section 3.2, we use the preventive maintenance model in [26], with the Weibull time to failure distribution. We optimize the software rejuvenation trigger interval in order to maximize the system availability or minimize the operational cost. Fig. 3 shows this model consisting of three states: UP state, or state 0, in which the system

Table 3
Results of model fitting for accelerated failure times.

Accelerated level	Model	Lk	Best-fit ranking
4(S1)	Lognormal	−35.7755	1st
	Weibull	−37.5017	2nd
	Exponential	−53.4806	3rd
8(S2)	Lognormal	−36.9586	1st
	Weibull	−37.7957	2nd
	Exponential	−57.6369	3rd
12(S3)	Lognormal	−43.5112	2nd
	Weibull	−43.1158	1st
	Exponential	−59.9855	3rd
16(S4)	Lognormal	−46.6123	1st
	Weibull	−46.8550	2nd
	Exponential	−61.2881	3rd

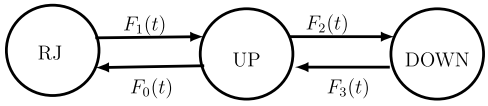


Fig. 3. Rejuvenation model.

is up; RJ state, or state 1, in which the system is undergoing software rejuvenation, and DOWN state, or state 2, in which the system is down and under reactive repair. State 0 is the only up state. From state 0 the system will enter state 1 with a general distribution function, $F_0(t)$, for the software rejuvenation trigger interval, or fail and enter state 2 with a general time to failure distribution $F_2(t)$. The distribution function for the duration of software rejuvenation (proactive repair) is $F_1(t)$, and the distribution function for the duration of reactive repair is $F_3(t)$.

We assume that the rejuvenation trigger interval is deterministic (t_0) and the mean time to carry out the rejuvenation and reactive repair are t_1 and t_2, respectively. We use $F(t)$ to represent $F_2(t)$ shown in Fig. 3. The two-parameter Weibull pdf for TTF is given by:

$$f(t) = \frac{\beta}{\eta}\left(\frac{t}{\eta}\right)^{\beta-1} e^{-\left(\frac{t}{\eta}\right)^{\beta}} \tag{5}$$

where,
$f(t) \geq 0, t \geq 0, \beta \geq 0, \eta \geq 0$,
η = scale parameter,
β = shape parameter (or slope).
The CDF of this Weibull distribution is given by:

$$F(t) = 1 - e^{-\left(\frac{t}{\eta}\right)^{\beta}}. \tag{6}$$

The sojourn time in UP state is then given by:

$$h_0 = \int_0^{t_0} (1 - F(t))\, dt = \frac{\eta}{\beta}\Gamma\left(\frac{1}{\beta}\right) G\left(\frac{1}{\eta^{\beta}} t_0^{\beta}, \frac{1}{\beta}\right) \tag{7}$$

where $G(x, \beta) = \frac{1}{\Gamma(\beta)} \int_0^x e^{-u} u^{\beta-1}\, du$ is the incomplete gamma function. Hence, we can get the steady state availability:

$$A_{weib} = \frac{h_0}{h_0 + (1 - F(t_0))t_1 + F(t_0)t_2}. \tag{8}$$

According to [30], the Weibull distribution may adopt the same parametrization structure shown in (3), where $\sigma = 1/\beta$ is the scale parameter, and $\mu = \log(\eta)$ is the location parameter. Hence, the assumption of same scale parameter across the stress levels must be evaluated on the estimated values of β after fitting the Weibull model to the four samples of failure times. We verified that the four beta values are inside the CI calculated for each sample, and their intervals satisfy the assumption of scale invariance. Table 4 presents the estimates for Weibull parameters, obtained through the maximum likelihood (ML) parameter estimation method [17].

From Table 4 we can see the value of the shape parameter of the fitted Weibull distribution is high. It means that the variance of TTF data is low at the same accelerated level. Our experimental study is related to memory-related aging failures, the property of which has a fixed upper bound for memory usage before observing failures. Each TTF sample is the time from the beginning of a test to the time when the servers memory is exhausted. The TTF is determined by the memory

Table 4
Parameter estimation of Weibull model.

Accelerated level	Parameter	ML estimate	90% CI	
			Lower	Upper
S1	η_1	787.015	753.944	821.536
	β_1	15.464	10.071	23.743
S2	η_2	1409.691	1376.925	1443.237
	β_2	28.116	17.881	44.209
S3	η_3	1992.147	1928.448	2057.949
	β_3	20.146	12.138	33.436
S4	η_4	2423.091	2308.777	2543.065
	β_4	13.631	8.368	22.202

Table 5
IPL-Weibull parameter.

Parameter	ML estimate	90% CI	
		Lower	Upper
β	17.3682	13.7767	21.8959
k	9E−6	8E−6	1.1E−5
w	0.9796	0.9397	1.0195

consumption rate, which is affected by two main factors: one is the frequency of visiting Search Request Page, and the other is the memory leak injection rate. The frequency of visiting Search Request Page depends on the TPC-W specification, while the memory leak injection rate is controlled by an integer N. Memory consumption rate have small changes among replications. Therefore, at each stress level, the variance of TTF data is low, and the shape parameter of the fitted Weibull distribution is high.

The IPL-Weibull model can be derived by setting $\eta = L(s)$, yielding the following IPL-Weibull pdf:

$$f(t, s) = \beta k s^w (k s^w t)^{\beta-1} e^{-(k s^w t)^\beta}. \tag{9}$$

This is a three-parameter model. The estimated IPL-Weibull parameters are listed in Table 5.

We obtain the MTTF at normal level as 7.6115E+6 s, or 126,858 min when memory consumption rate at normal level is 0.0124 kB/s. The 90% confidence interval of MTTF at normal level is (5.3730E+6, 1.0782E+7) s, that is, (89,550, 179,700) min. Correspondingly, for the parameter η confidence interval is denoted by $(\eta_{low}, \eta_{high})$ and is computed as (5.3730E+6, 1.0782E+7) s. Also, the parameter β confidence interval, denoted by $(\beta_{low}, \beta_{high})$ is (13.7767, 21.8959) as shown in Table 5.

Therefore, from Eq. (8) we derive the steady state availability $A = A_{weib}(\eta, \beta)$, and its confidence interval $A_{low} = A_{weib}(\eta_{low}, \beta_{low})$, $A_{high} = A_{weib}(\eta_{high}, \beta_{high})$. We assume that the mean duration for carrying out software rejuvenation, t_1, is 1 min, and the mean time for reactive repair, t_2, is 5 min. One objective is to maximize the steady-state availability, and we can get the optimal time to rejuvenation trigger, t_0, Another objective function is to minimize the expected cost. A cost of C_f per minute is incurred when the system is down due to system failure, and a cost of C_f' is incurred for each reactive repair carried out; a cost of C_p per minute is incurred when the system is down for carrying out software rejuvenation, and a cost of C_p' is incurred for each rejuvenation action carried out. The total expected cost per minute is thus

$$C = C_f \pi_2 + C_f' \pi_2 / t_2 + C_p \pi_1 + C_p' \pi_1 / t_1, \tag{10}$$

where π_2/t_2 and π_1/t_1 are the average number of reactive repairs and rejuvenation executions per minute, respectively.

We assume that $C_p = C_p' = 1/60$, $C_f = C_f' = 5/60$. Let $C = C(\eta, \beta)$, $C_{low} = C(\eta_{low}, \beta_{low})$, and $C_{high} = C(\eta_{high}, \beta_{high})$, so we derive the cost C, C_{low} and C_{high}.

5. Parametric and non-parametric estimation

In this section, we apply the parametric and non-parametric method to estimate the optimal software rejuvenation trigger interval.

5.1. Fixed point iteration

Taking the derivative of expression (8) with respect to t_0, we obtain Eq. (11) below. Successive substitution is used to solve Eq. (11) to obtain the optimized rejuvenation trigger interval maximizing the system availability. For minimizing the

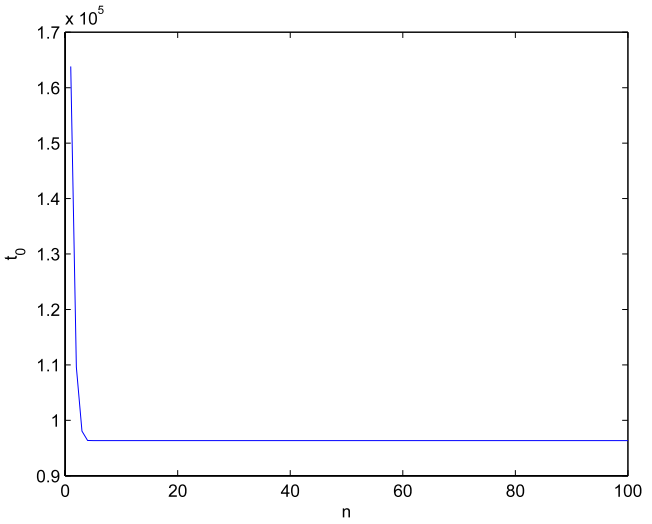


Fig. 4. Iterative behavior of t_0 to maximize system availabilities.

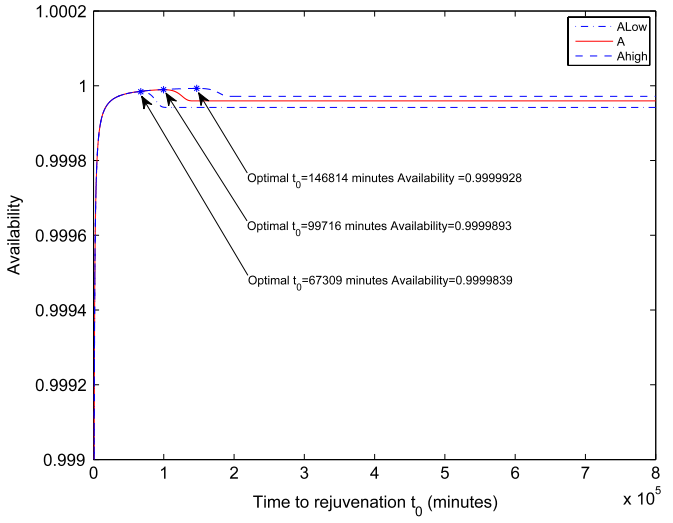


Fig. 5. Steady-state availability *vs.* time to rejuvenation t_0.

operational cost we take the derivative of Eq. (10) to obtain Eq. (12) below.

$$[t_2 + (t_1 - t_2)e^{-\left(\frac{t_0}{\eta}\right)^{\beta}}] + h_0(t_0) \cdot \frac{\beta}{\eta^{\beta}} \cdot (t_1 - t_2) \cdot t_0^{\beta-1} = 0 \tag{11}$$

$$[Dh_0(t_0) + Dt_1 - (t_2 - t_1) \cdot t_1] \cdot \frac{\beta}{\eta^{\beta}} \cdot t_0^{\beta-1} - [D \cdot (1 - e^{-\left(\frac{t_0}{\eta}\right)^{\beta}} + t_1)] = 0 \tag{12}$$

where, $D = t_2C_f + C'_f - t_1C_p - C'_p$.

Using successive substitution to solve formulas (11) and (12), respectively, we derive the optimized rejuvenation trigger interval t_0 of availability A_{low}, A and A_{high}, considering to maximize the system availability. We obtain 67,309, 99,716, and 146,814 min to t_0, respectively, and the corresponding availabilities for A, A_{low} and A_{high} are 0.9999839, 0.9999893, and 0.9999928. The optimized rejuvenation trigger interval t_0 for cost C_{low}, C and C_{high} that minimize the operational cost is 61,456, 92,776, 138,649, respectively. Behavior of the sequence of iteratives solving for optimal t_0 is shown in Fig. 4.

Steady-state availability *vs.* time to rejuvenation trigger, t_0, is shown in Fig. 5. The optimal time to trigger rejuvenation and the corresponding availability are marked in this figure. In this case, the optimal choice of rejuvenation trigger interval could clearly accrue availability improvement.

The average cost *vs.* time to rejuvenation t_0 is shown in Fig. 6. The optimal software rejuvenation intervals for the cost models are as short as 61,456–138,649 min, while the optimal intervals for the availability models are 67,309–146,814 min.

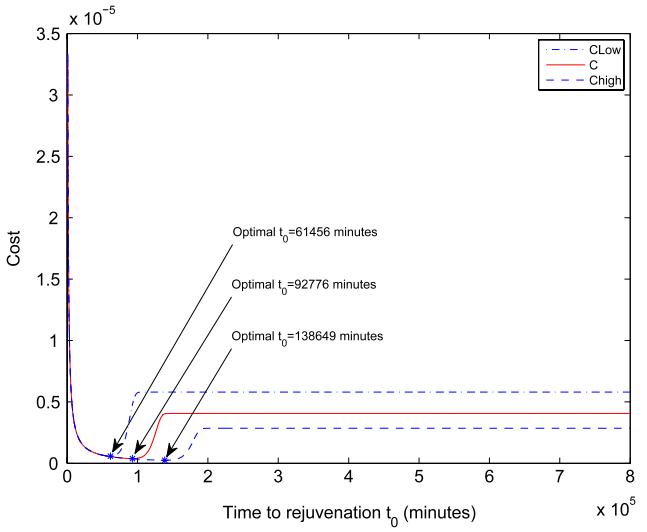


Fig. 6. Average cost *vs.* time to rejuvenation t_0.

5.2. Non-parametric estimation

We use the statistical non-parametric algorithm to estimate the optimal software rejuvenation trigger interval, given the sample failure time data. We employ the approach of translating the underlying problem of maximizing system availability to a graphic one presented by Dohi et al. in [22]. Following Barlow and Campo [32], define the scaled total time on test (TTT) transform of the failure time distribution:

$$\phi(p) = (1/\eta) \int_0^{F^{-1}(p)} \overline{F}(t)\, dt \tag{13}$$

where, η is the mean of the time to failure distribution $F(t)$, and

$$F^{-1}(p) = \inf\{t_0; F(t_0) \geq p\}, \quad (0 \leq p \leq 1). \tag{14}$$

The optimal software rejuvenation trigger interval t_0 maximizing the system A_{weib} is equivalent to obtaining $p^*(0 \leq p^* \leq 1)$ such as

$$\max \frac{\phi(p)}{p + t_1/(t_2 - t_1)}. \tag{15}$$

From Eq. (15), the optimal rejuvenation trigger interval $t_0 = F^{-1}(p)$ is determined by calculating the optimal point $p^*(0 \leq p^* \leq 1)$ maximizing the tangent slope from the point $(-t_1/(t_2 - t_1), 0)$ to the curve $(p, \phi(p)) \in [0, 1] \times [0, 1]$.

Suppose an ordered complete observation $0 = x_0 \leq x_1 \leq x_2 \leq \cdots \leq x_n$ of the failure times from a continuous distribution F, which is unknown. Then the scaled TTT statistics based on this sample are defined by $\phi_{nj} = \psi_j/\psi_n$, where

$$\psi_j = \sum_{i=1}^{j} (n - i + 1)(x_i - x_{i-1}), \quad (j = 1, 2, \ldots, n; \psi_0 = 0). \tag{16}$$

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

$$F_n(x) = \begin{cases} j/n & \text{for } x_j \leq x < x_{j+1} \\ 1 & \text{for } x_n \leq x. \end{cases} \tag{17}$$

By plotting the points $(j/n, \phi_{nj})$ $(j = 0, 1, 2, \ldots, n)$ and connecting them by line segments is called the scaled TTT plot. Then, a non-parametric estimator of the optimal software rejuvenation trigger interval $\hat{t_0}$ that maximizes A_{weib} is given by x_{j^*}, where

$$j^* = \left\{ j | \max \frac{\phi_{nj}}{j/n + t_1/(t_2 - t_1)} \right\}. \tag{18}$$

Fig. 7 shows the estimation results of the optimal rejuvenation trigger interval on the two-dimensional graph for A. The failure data are generated by the Weibull distribution with shape parameters of β, β_{low}, β_{high} and scale parameters of η, η_{low},

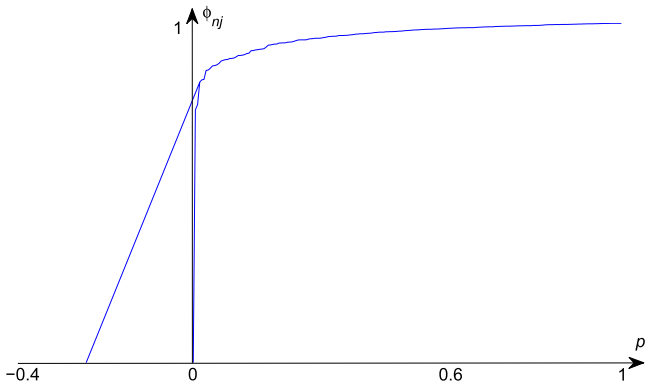


Fig. 7. Estimation of software rejuvenation trigger interval on the two-dimensional graph.

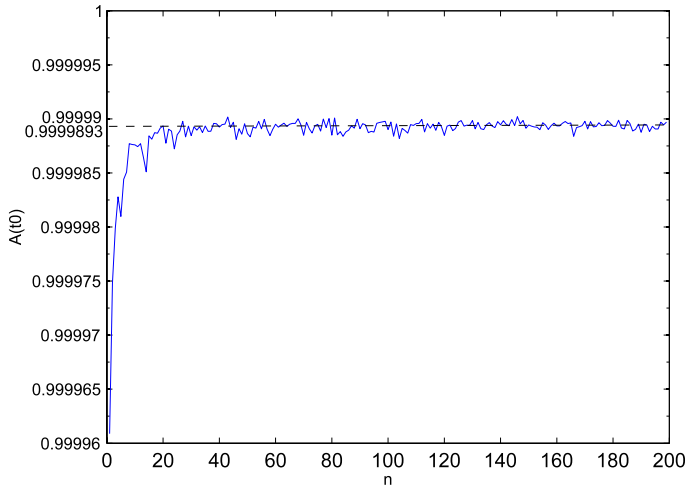


Fig. 8. Asymptotic behavior of the maximum system availability.

η_{high}, respectively. For 200 failure data points, the estimates of optimal rejuvenation trigger interval for A, A_{low}, and A_{high} are $\hat{t_0}$ =101664, 68885, 149308, and the corresponding availabilities are 0.9999839, 0.9999893, 0.9999927.

We examine the asymptotic properties of the estimators developed above, to investigate the number of data points at which one can estimate the optimal software rejuvenation schedule accurately without fitting the data to a failure time distribution. The failure data are synthetically generated by the Weibull distribution with shape parameter β and scale parameter η. Fig. 8 illustrates the asymptotic behavior of the estimates for the system availabilities. The maximum availabilities are calculated with the estimation algorithm, we change the sample failure data observed by experiment in a single step from two sample failure data, so that the sample mean $\hat{\eta} = \sum_{k=1}^{n} \frac{x_k}{n}$ changes as the failure data is observed. It can be seen that the system availability can be estimated accurately after the number of observations becomes about 20. Fig. 9 illustrates the asymptotic behavior of the estimate for the optimal software rejuvenation schedule. It is seen that the estimate of the optimal rejuvenation schedule fluctuates until the number of observations is about 50.

From Fig. 4, we can conclude that parametric estimation will give the fixed optimal value, while from Figs. 8 and 9, we see that non-parametric estimation of optimal value changes with number of sample failure observation, and converges at the number 20 or 50 with respect to estimated availability or optimal rejuvenation trigger interval t_0. The availabilities or optimized trigger intervals t_0 are very close to those obtained from the fixed point estimations. It is also clear that the non-parametric approach is adaptive and distribution free.

6. Simulation approach in ALT

We develop a discrete-event simulation model of ALT using C++, which is modeled after the TPC-W experiment in Section 3, to validate the analytical results derived from the aforementioned IPL models in Section 4. Fig. 10 shows the queueing model of the program. In this simulation model, each client represents one EB in our experiment, and simulates the EB's actions of sending requests and receiving responses. Following the experimental setup presented in Section 3, we select the workload to be 100 clients. Requests that arrive while the server is busy are placed into a queue. Considering

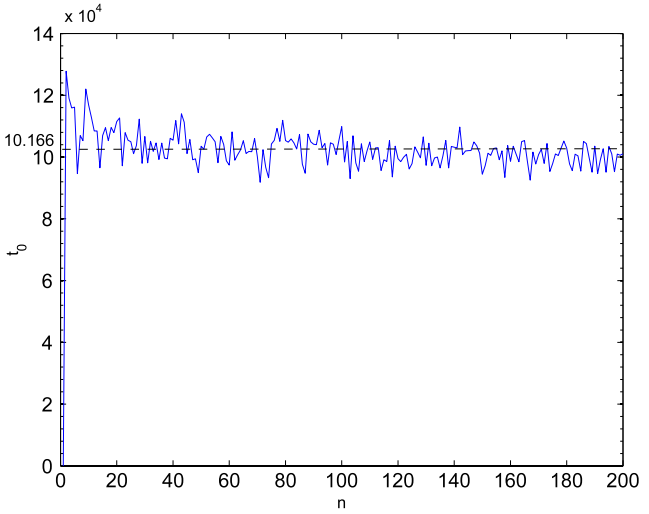


Fig. 9. Asymptotic behavior of the optimal software rejuvenation schedule for A.

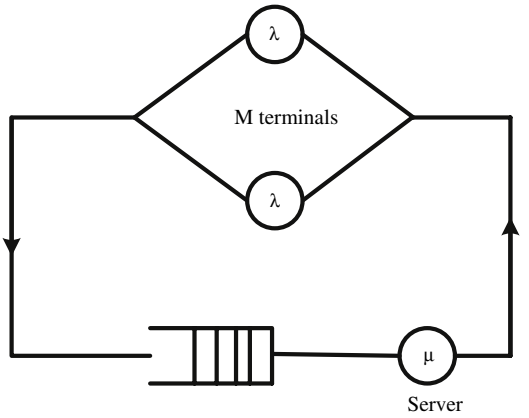


Fig. 10. Queueing model of the simulation program.

that the server in our experiment has only one CPU, we configure the server in the simulation program to work as an FCFS (First-come, First-served) node, and the length of the queue is set to the sum of Tomcat's buffer length (acceptCount = 100) and max threads number (maxThreads = 200) [33].

The TPC-W benchmark™ defines three distinct mix of web interactions: Browsing, Shopping and Ordering. The term web interaction refers to a complete process of requesting for one of the 14 different pages in the e-commerce web site. It includes one or more HTTP requests for HTML documents, images files or other web objects. Each EB or client starts from requesting the Home Page, and then randomly selects the next navigation option according to the current mix of web interactions [27]. We select the Shopping mix in our experiment, and also it is used in our simulation.

6.1. Importance sampling method in simulation

We employ the importance sampling (IS) technique to reduce the simulation time. There are total of 14 pages for TPC-W benchmark, in which each page is taken as the task module, such as $m_1, m_2, \ldots, m_{14}$. Assume that the execution time of each module is $\tau_1, \tau_2, \ldots, \tau_{14}$. Let $X_t = X(t)$ represent the running task module at time t. Since a task module transfers to the next task module according to the transition probability matrix of TPC-W specification, thus the next state relates to the current state. Accordingly, for $t_0 < t_1 < \cdots < t_{k+1} < t$, the conditional probability mass function (pmf) of $X(t)$ satisfies the Eq. (19).

$$\begin{cases} P(X(t_{k+1}) = i_{k+1} \mid X(t_0) = i_0, X(t_1) = i_1, X(t_k) = i_k) = P(X(t_{k+1}) = i_{k+1} \mid X(t_k) = i_k) \\ i_k \in \{m_1, m_2, \ldots, m_{14}\}, k = 0, 1, \ldots. \end{cases} \tag{19}$$

Therefore, $X(t_i)$ is a discrete time Markov chain(DTMC). Let $p_{ij} = P(X(t_{k+1}) = j \mid X(t_k) = i)$, We can obtain the transition probability matrix of $X(t_i)$ as the Eq. (20).

$$P = [p_{ij}] = \begin{bmatrix} p_{1,1} & p_{1,2} & \cdots & p_{1,14} \\ p_{2,1} & p_{2,2} & \cdots & p_{2,14} \\ \cdots & \cdots & \cdots & \cdots \\ p_{14,1} & p_{14,2} & \cdots & p_{14,14} \end{bmatrix}. \tag{20}$$

Let $\pi = [\pi_1, \pi_2, \ldots, \pi_{14}]$ represent $X(t_i)$ steady state probability vector, which satisfies $\pi = \pi P$ and $\sum_{i=1}^{14} \pi_i = 1$. Let $E(\tau_i) = a_i$ and assume that the average running time of the ith task module is a_i. Then $X(t)$ is a semi-Markov process, with the steady state probability

$$p_i = \frac{\pi_i \cdot a_i}{\sum_{j=1}^{14} \pi_j \cdot a_j}.$$

The IS approach is used to reduce the simulation time. We change the $[p_{ij}] \to [p'_{ij}]$,

$$P' = [p'_{ij}] = \begin{bmatrix} p'_{1,1} & p'_{1,2} & \cdots & p'_{1,14} \\ p'_{2,1} & p'_{2,2} & \cdots & p'_{2,14} \\ \cdots & \cdots & \cdots & \cdots \\ p'_{14,1} & p'_{14,2} & \cdots & p'_{14,14} \end{bmatrix}. \tag{21}$$

Let π' be the steady state probability vector of the new transition probability matrix. Thus

$$p'_i = \frac{\pi'_i \cdot a_i}{\sum_{j=1}^{14} \pi'_j \cdot a_j}$$

is the steady state probability of semi-Markov chain $X(t)$ at the new transition probability matrix.

Let the aging rate (memory consumption rate) of module i be v_i. The average aging rate with the transition probability matrix P is

$$\bar{v} = v_1 p_1 + v_2 p_2 + \cdots + v_{14} p_{14}.$$

The average aging rate with the new transition probability matrix, P', is

$$\bar{v'} = v_1 p'_1 + v_2 p'_2 + \cdots + v_{14} p'_{14}.$$

Thus, we define the acceleration factor of P to P' for IS simulation, $s(P, P')$, that is,

$$s(P, P') = \frac{\bar{v'}}{\bar{v}}. \tag{22}$$

Assuming that the average aging rate of different task modules satisfies the following,

$$v_1 \geq v_2 \geq v_3 \geq \cdots \geq v_{14},$$

we increase the visiting probability of the TPCW_search_request_servlet, which belongs to the SREQ page according to TPC-W specification. Also we reduce the visiting probability of an unimportant page, such as the Home page. For example,

$$[p'_{ij}] = \begin{bmatrix} p_{1,1} + \epsilon & p_{1,2} & \cdots & p_{1,13} & p_{1,14} - \epsilon \\ p_{2,1} + \epsilon & p_{2,2} & \cdots & p_{2,13} & p_{2,14} - \epsilon \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ p_{14,1} + \epsilon & p_{14,2} & \cdots & p_{14,13} & p_{14,14} - \epsilon \end{bmatrix}. \tag{23}$$

We obtain the π' with $\pi'_1 > \pi_1$, $\pi'_{14} < \pi_{14}$ so that the system aging rate becomes larger than that of the original system. Therefore, the system simulation time will be reduced using the IS approach.

Let T represent the MTTF at normal use level and the T' is the MTTF at the accelerated level, as a result we obtain the following formula,

$$T = s(P, P') * T' = \frac{\bar{v'}}{\bar{v}} * T'. \tag{24}$$

6.2. IS parameter settings in simulation

A web interaction in the TPC-W experiment can be divided into two phases. In the first phase, EB sends an HTTP request to the server asking for the HTML document, and parses the HTML code to get the URLs of other web objects, which are all

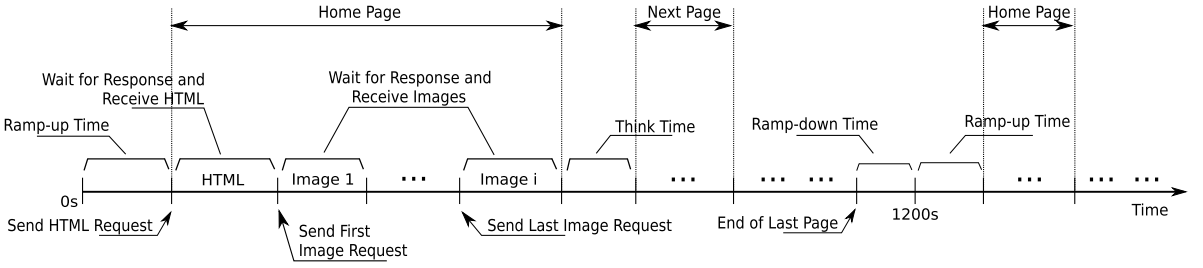


Fig. 11. Work flow of one client in the simulation program.

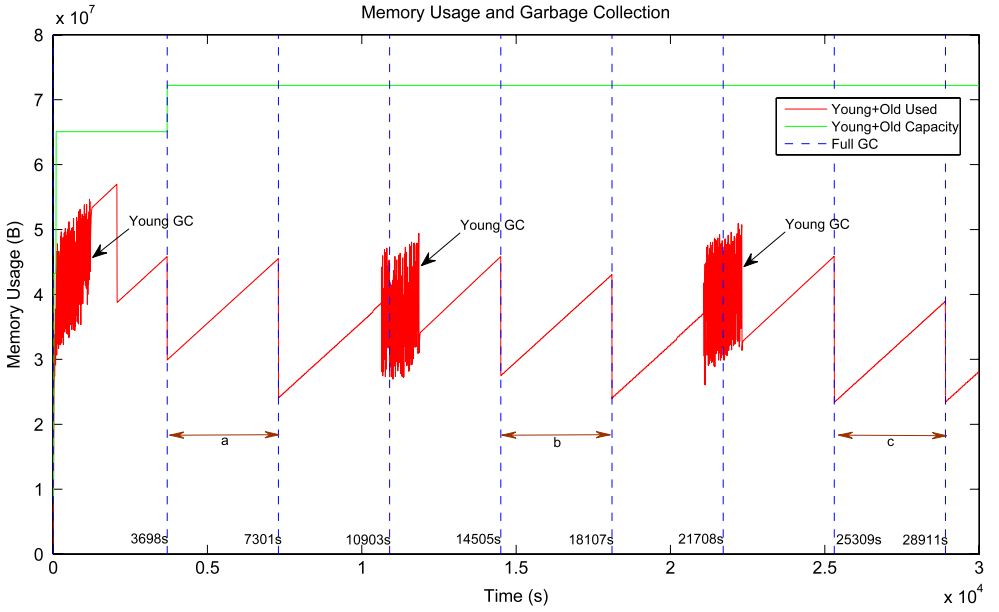


Fig. 12. Memory usage and garbage collection.

image files in our case. In the second phase, EB sends HTTP requests for these image files one after another, a web interaction completes when all of the image files are received or a timeout occurs. The quantities of image files on different pages can be different, so are the numbers of HTTP requests the client sends for different pages. When a web interaction finishes, EB enters a "Think" phase. "Think Time" is sampled from an exponential distribution with mean being varied from 7 to 8 s [27]. The work flow of one EB is shown in Fig. 11.

In our simulation program, we use this response time of non-queueing case as the service time of each request. The initial occupancy of the server's memory is 19,726,336 bytes, and the capacity is 127,729,664 bytes. They are obtained from the Young plus Old heap memory of Tomcat in our experiment. We leave the Permanent zone out because the usage and capacity of this zone is kept substantially unchanged during our experiment.

Fig. 12 shows the memory usage and garbage collection in the first 30,000 s of our non-accelerated experiment, in which 100 EBs are used. We observed that the full garbage collection runs approximately every one hour in our experiment, however, the young generation collections show no obvious regularity. The memory consumed by processing one request is defined as $(t_j - t_i) \cdot v$, where t_j is the time when the current request finished, t_i is the time when the last request finished, and $v = 2462.9\,\mathrm{B/s}$. These memory objects will be marked as recyclable after the request is processed, and then be collected by the subsequent full garbage collection. We made no distinction between the Young and Old heap memory in our simulation program, so we did not add the young generation collection. The memory leak injection method in the simulation is the same as the one we described in Section 3.1. Depending on a *randomNumber* between 0 and N, we randomly inject about 1-megabyte of memory leak to the server when the Search Request Page is processed. The injected memory will not be added to the recyclable memories, so it will never be collected by the garbage collector. When the unrecyclable memory reaches the capacity of the server, the server becomes invalid, and must be restarted to get back to work again.

Using the above IS in simulation program, we increase the visiting probabilities of the SREQ page, in which the TPCW_search_request_servlet will be invoked, and in the meantime we reduce the visiting probabilities of the Home page

Table 6
$RE_{j,Si}$ values using IS in simulation.

IS group	S1	S2	S3	S4
1	0.0611742	0.0535918	0.0649777	0.0264433
2	0.0618430	0.0267465	0.0635911	0.0109086
3	0.0454371	0.0399137	0.0265406	0.0403616
4	0.0369092	0.0407316	0.0155297	0.0553655
5	0.0740658	0.0439704	0.0169784	0.0119429
6	0.0441900	0.0650622	0.0490446	0.0447249

Table 7
Simulation results in use level ($N = 69000$) and relative errors.

IS group	MTTFs	CIs	$RE_{j,normal}$
1	7,464,491	(7,296,383, 7,632,599)	0.0119426
2	7,459,400	(7,264,551, 7,654,248)	0.0126530
3	7,590,238	(7,430,392, 7,750,084)	0.0046836
4	7,539,277	(7,339,427, 7,739,126)	0.0019211
5	7,454,041	(7,231,326, 7,676,756)	0.0133798
6	7,526,793	(7,365,989, 7,687,597)	0.0034650

and the SHOP page. The 14 pages from the SREQ to SHOP page represent the state $i_1, i_2, \ldots, i_{14}$ in Eq. (19). Based on the original transition probability matrix, specified by TPC-W benchmark, we choose 6 different groups to change the ϵ_i values to run the IS simulation, and also 12 replications at each group.

1. $\epsilon_1 = 0.00690$, $\epsilon_2 = 0.00310$, $\epsilon_3 = 0.05351$, the values of the column of "SREQ" add ϵ_k, $k = 1, 2, 3$, while the values of the column of "HOME page" reduce ϵ_k, $k = 1, 2, 3$. If the values of the column of "HOME page" are less than those of ϵ_k, $k = 1, 2, 3$, the values of the same row of the HOME and the SREQ page remain unchangeable. The following groups obey the same rules for the column of the SREQ page adding ϵ_k, $k = 1, 2, \ldots$, and the column of the HOME page reducing ϵ_k, $k = 1, 2, \ldots$, if no further explanations.
2. $\epsilon_1 = 0.00690$, $\epsilon_2 = 0.00310$, $\epsilon_3 = 0.05351$, $\epsilon_4 = 0.01390$, $\epsilon_5 = 0.02260$.
3. $\epsilon_1 = 0.00690$, $\epsilon_2 = 0.00310$, $\epsilon_3 = 0.05351$, $\epsilon_4 = 0.01390$, $\epsilon_5 = 0.02260$, $\epsilon_6 = 0.02260$.
4. $\epsilon_1 = 0.00690$, $\epsilon_2 = 0.00310$, $\epsilon_3 = 0.05351$, $\epsilon_4 = 0.01390$, $\epsilon_5 = 0.02260$, $\epsilon_6 = 0.02260$, $\epsilon_7 = 0.07251$.
5. $\epsilon_1 = 0.00690$, $\epsilon_2 = 0.00310$, $\epsilon_3 = 0.05351$, $\epsilon_4 = 0.01390$, $\epsilon_5 = 0.02260$, $\epsilon_6 = 0.02260$, $\epsilon_7 = 0.07251$, $\epsilon_8 = 0.47504$.
6. $\epsilon_1 = 0.00690$, $\epsilon_2 = 0.00310$, $\epsilon_3 = 0.05351$, $\epsilon_4 = 0.01390$, $\epsilon_5 = 0.02260$, $\epsilon_6 = 0.02260$, $\epsilon_7 = 0.07251$, $\epsilon_8 = 0.47504$, $\epsilon_9 = 0.25453$, $\epsilon_{10} = 0.06950$, $\epsilon_{11} = 0.07021$. The values of the column of "HOME page" reduce ϵ_k, $k = 1, 2, 3, \ldots, 9$, and the values of the column of "SHOP page" reduce ϵ_k, $k = 10, 11$, and the values of "SREQ page" add ϵ_k, $k = 1, 2, \ldots, 11$.

The obtained acceleration factor using Eq. (24), at each group, is 1.0683, 1.08778, 1.3068, 1,38846, 1,653468, and 1.8652, respectively.

6.3. Cross validation results among ALT, simulation and SMP model

We carried out 7 repeated simulation experiments on acceleration levels S1–S4, in which N equals 4, 8, 12 and 16, respectively, totaling 6 groups. Thus, there are $6 \times 7 \times 4$ simulation replications. From the simulation, we obtained the TTF samples in each replication. We go on to obtain the $\text{MTTF}_{j,Si}$, at each group j, where $j \in 1, 2, \ldots, 6$, and at each acceleration level Si, where $i \in 1, 2, 3, 4$. We calculate the relative errors between the $\text{MTTF}_{j,Si}$ and the MTTF_{Si}, as $RE_{j,Si} = |\text{MTTF}_{j,Si} - \text{MTTF}_{Si}|/\text{MTTF}_{Si}$. The $RE_{j,Si}$ values at each group are shown in Table 6.

Accordingly, we carried out 14 repeated simulation when $N = 69000$, comparing with the normal level experiment in Section 3, at each group using different ϵ_i values. We obtain the MTTFs with its 90% CIs at each group j. We define the relative error $RE_{j,normal}$ at normal level between $\text{MTTF}_{j,sim}$, the MTTF from the simulation and MTTF_{normal}, IPL-Weibull in Section 4, as $RE_{j,normal} = |(\text{MTTF}_{j,sim} - \text{MTTF}_{normal})|/\text{MTTF}_{j,sim}$. The calculated $RE_{j,normal}$ values as well as MTTFs with their 90% CIs are shown in Table 7.

From Tables 6 and 7, it can be seen that the 90% CIs of MTTF at normal level, obtained from the simulation results, fall into the CIs obtained from the experimental results, thus we consider that the simulation model represents the experimental testbed adequately. We can see that the 90% CIs of TTF are close to what we obtained from the IPL-Weibull model in Section 4.

Next, we use the acceleration factor equals to 1.8652 in IS, to introduce both reactive repair and rejuvenation in our simulation program to cross-validate the availabilities computed from the semi-Markov model. We take 9 different rejuvenation trigger intervals in our simulation model with reactive repair at non-accelerated level, and use 12 replications at each rejuvenation trigger interval of 4,038,540, 5,982,960, 7,306,314, 7,553,784, 7,801,254, 8,400,000, 8,808,840, 9,600,000 and 10,800,000 s, among which 5,982,960 is the optimal trigger interval obtained from the semi-Markov model.

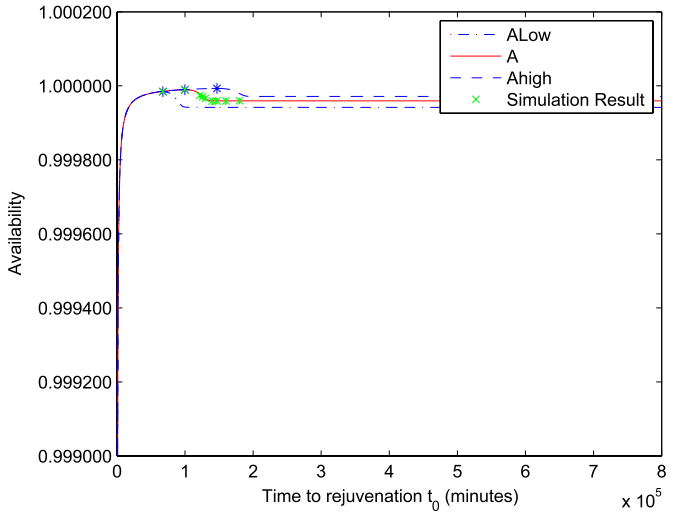


Fig. 13. Availability comparisons between semi-Markov model and simulation model.

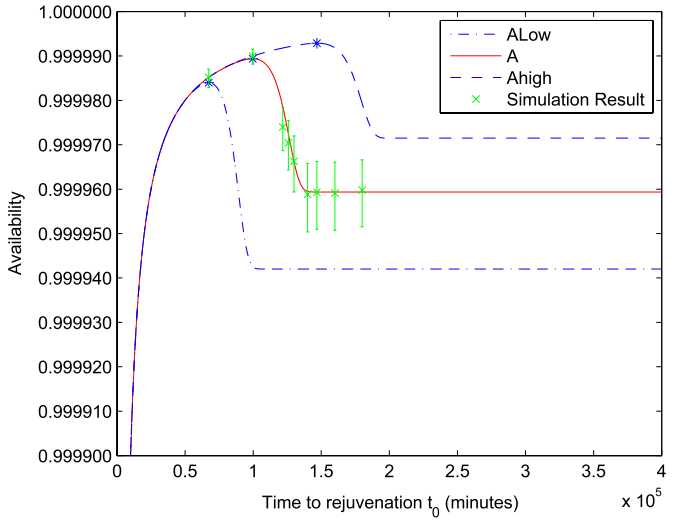


Fig. 14. Zoomed: availability comparisons between semi-Markov model and simulation model.

Then we estimate the system availability at each rejuvenation trigger and the 90% CI applying the F-distribution from the simulation model with reactive repair. In Fig. 13, we plot the simulation estimated availability results against the semi-Markov model, Furthermore, in a zoom-in of Fig. 14, we show nine discrete availability points as well as their 90% CIs. From these figures we can see that the simulation availability results have a reasonably good match with those from the semi-Markov model. The IS approach greatly reduces the simulation time. When the acceleration factor of the 8th group equals 1.8652, the simulation time using IS is about half of that without using IS.

From Section 5.2, we can see that the obtained availabilities using non-parametric method stabilize at 25 samples. Since the IS approach may provide the TTFs at normal level of ALT in a tolerably short time, then we apply the non-parametric method to estimate the optimized trigger interval maximizing the availability. We use the observed TTFs at normal level of ALT obtained from IS simulation, and there are total 50 $\times$ 10 data sets, where each group includes 50 TTFs, and 10 groups. Using above TTFs estimated by non-parametric method, we obtain the optimized availabilities as 0.99999015, 0.99999021, 0.99998961, 0.99998984, 0.99998958, 0.99999005, 0.99998990, 0.99998963, 0.99999043, and 0.99999057. The corresponding optimized trigger intervals are 111,897, 112,583, 106,035, 108,499, 105,786, 110,709, 109,069, 106,303, 115,179, and 116,822 min. Thus, we successfully cross validate the results of SMP model (in Section 4) by IS simulations.

7. Conclusion

In this paper, we obtain the accelerated life test results by injecting memory leaks. Then, Weibull time to failure distribution of the rejuvenation model is used in a semi-Markov model, to optimize the software rejuvenation trigger interval so

as to maximize the availability or minimize the operational cost. Secondly, the parametric and the non-parametric methods are utilized to estimate the optimal software rejuvenation trigger interval. The results show that optimal rejuvenation t_0 by parametric method is stable, while optimal t_0 and availabilities obtained by non-parametric estimation adapt to changes and eventually converge. Finally, we develop a simulation model using IS to cross validate the ALT experimental results and the SMP model. The IS approach greatly reduces the simulation time. We apply the non-parametric method to estimate the optimized trigger intervals by comparing the availabilities obtained from the SMP model and those from IS simulation using non-parametric method.

Acknowledgments

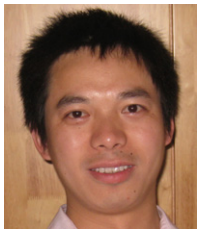
This work was supported in part by the National Natural Science Foundation of China under Grant No. 60873036. This work was also supported in part by Brazilian CNPq, Fundamental Research Funds for the Central Universities (award number HEUCF100601, HEUCFT1007), and Fundamental Research Funds for Harbin Engineering University.

References

[1] M. Grottke, L. Li, K. Vaidyanathan, K.S. Trivedi, Analysis of software aging in a web server, IEEE Transactions on Reliability 55 (3) (2006) 411–420.
[2] M. Grottke, A.P. Nikora, K.S. Trivedi, An empirical investigation of fault types in space mission system software, in: 2010 IEEE/IFIP International Conference on Dependable Systems & Networks (DSN), IEEE, 2001 L Street, NW. Suite 700 Washington, DC 20036-4910, USA, 2010, pp. 447–456.
[3] Y. Huang, C. Kinatla, N. Kolertis, Software rejuvenation: analysis, module and applications, in: 25th Symposium on Fault Tolerant Computing, IEEE, 2001 L Street, NW. Suite 700 Washington, DC 20036-4910, USA, 1995, pp. 381–390.
[4] S. Garg, A.V. Moorsel, K. Vaidyanathan, K. Trivedi, A methodology for detection and estimation of software aging, in: 1998 9th International Symposium on Software Reliability Engineering, IEEE, 2001 L Street, NW. Suite 700 Washington, DC 20036-4910, USA, 1998, pp. 283–292.
[5] M. Grottke, R. Matias, K.S. Trivedi, The fundamentals of software aging, in: 2008 IEEE First International Workshop on Software Aging and Rejuvenation (WoSAR), IEEE, 2001 L Street, NW. Suite 700 Washington, DC 20036-4910, USA, 2008, pp. 1–6.
[6] Y.F. Jia, L. Zhao, K.-Y. Cai, A nonlinear approach to modeling of software aging in a web server, in: 15th Asia-Pacific Software Engineering Conference, IEEE, 2001 L Street, NW. Suite 700 Washington, DC 20036-4910, USA, 2008, pp. 77–84.
[7] E. Marshall, Fatal error: how patriot overlooked a scud, Science 255 (5050) (1992) 1347.
[8] K.J. Cassidy, K.C. Gross, A. Malekpour, Advanced pattern recognition for detection of complex software aging in online transaction processing servers, in: 2010 International Conference on Dependable Systems and Networks, IEEE, 2001 L Street, NW. Suite 700 Washington, DC 20036-4910, USA, 2010, pp. 478–482.
[9] X. Zhang, H. Pham, Predicting operational software availability and its applications to telecommunication systems, International Journal of Systems Science 33 (11) (2002) 923–930.
[10] K.-Y. Cai, Software reliability and control, Journal of Computer Science and Technology 21 (5) (2006) 697–707.
[11] R. Matias, K.S. Trivedi, P.R. Maciel, Using accelerated life tests to estimate time to software aging failure, in: ISSRE 2010, IEEE, 2001 L Street, NW. Suite 700 Washington, DC 20036-4910, USA, 2010, pp. 211–219.
[12] ASF, Apache web server, 2011. http://httpd.apache.org.
[13] S. Garg, A. Puliafito, M. Telek, K.S. Trivedi, Analysis of preventive maintenance in transactions based software systems, IEEE Transactions on Computers 47 (1) (1998) 96–107.
[14] Y.J. Bao, X. Sun, K.S. Trivedi, A workload-based analysis of software aging and rejuvenation, IEEE Transactions on Reliability 54 (3) (2005) 541–548.
[15] J. Alonso, L. Silva, A. Andrzejak, P. Silva, J. Torres, High-available grid services through the use of virtualized clustering, in: 2007 Intl. Conf. on Grid Computing, IEEE, 2001 L Street, NW. Suite 700 Washington, DC 20036-4910, USA, 2007, pp. 34–41.
[16] L. Silva, H. Madeira, J.G. Silva, Software aging and rejuvenation in a soap-based server, in: 2006 Intl. Symposium on Network Computing and Applications, IEEE, 2001 L Street, NW. Suite 700 Washington, DC 20036-4910, USA, 2006, pp. 56–65.
[17] B. Nelson, Accelerated Testing: Statistical Method, Test Plans, and Data Analysis, Wiley, New Jersey, 2004.
[18] R. Matias, P.A. Barbetta, K.S. Trivedi, Accelerated degradation tests applied to software aging experiments, IEEE Transactions on Reliability 59 (1) (2010) 102–114.
[19] T. Bezenek, T. Cain, R. Dickson, T. Heil, M. Martin, C. McCurdy, R. Rajwar, E. Weglarz, C. Zilles, M. Lipasti, TPC-W benchmark java version, 2011. http://pharm.ece.wisc.edu/tpcw.shtml.
[20] J. Zhao, Y.L. Jin, K.S. Trivedi, R. Matias, Injecting memory leak to accelerate software failures, in: ISSRE 2011, IEEE, 2001 L Street, NW. Suite 700 Washington, DC 20036-4910, USA, 2011, pp. 260–269.
[21] K. Vaidyanathan, K.S. Trivedi, A comprehensive model for software rejuvenation, IEEE Transactions on Dependable and Secure Computing 2 (2) (2005) 124–137.
[22] T. Dohi, K. Goseva-Popstojanova, K.S. Trivedi, Statistical non-parametric algorithms to estimate the optimal software rejuvenation schedule, in: 2000 Pacific Rim Int'l Symp. Dependable Computing PRDC, IEEE, 2001 L Street, NW. Suite 700 Washington, DC 20036-4910, USA, 2000, pp. 77–84.
[23] A. Macedo, T.B. Ferreira, R. Matias, The mechanics of memory-related software aging, in: 2010 IEEE Second International Workshop on Software Aging and Rejuvenation (WoSAR), IEEE, 2001 L Street, NW. Suite 700 Washington, DC 20036-4910, USA, 2010, pp. 1–5.
[24] R. Matias, I. Beicker, B. Leitao, P. Maciel, Measuring software aging through os kernel instrumentation, in: 2010 IEEE Second International Workshop on Software Aging and Rejuvenation (WoSAR), IEEE, 2001 L Street, NW. Suite 700 Washington, DC 20036-4910, USA, 2010, pp. 1–6.
[25] J. Alonso, J. Berral, R. Gavald, J. Torres, Adaptive on-line software aging prediction based on machine learning, in: 2010 Intl. Conf. Dependable Systems and Networks DSN 2010, IEEE, 2001 L Street, NW. Suite 700 Washington, DC 20036-4910, USA, 2010, pp. 507–516.
[26] D.Y. Chen, K.S. Trivedi, Analysis of periodic preventive maintenance with general system failure distribution, in: 2001 Pacific Rim International Symposium on Dependable Computing, IEEE, 2001 L Street, NW. Suite 700 Washington, DC 20036-4910, USA, 2001, pp. 103–107.
[27] TPC, TPC benchmark™ W specification version 1.8, 2002. http://www.tpc.org/tpcw/spec/tpcw_V1.8.pdf.
[28] Oracle, Jstat—java virtual machine statistics monitoring tool, 2010. http://download.oracle.com/javase/1.5.0/docs/tooldocs/share/jstat.html.
[29] P. Sen, Estimates of the regression coefficient based on kendall's tau, Journal of the American Statistical Association 63 (4) (1968) 1379–1389.
[30] W.Q. Meeker, L.A. Escobar, Statistical Methods for Reliability Data, Wiley, New York, 1998.
[31] K.S. Trivedi, Probability and Statistics With Reliability, Queuing, and Computer Science Applications, second ed., John Wiley & Sons, New York, 2002.
[32] R.E. Barlow, R. Campo, Total time on test processes and applications to failure data analysis, in: R.E. Barlow, J. Fussell, N.D. Singpurwalla (Eds.), Reliability and Fault Tree Analysis, SIAM, Philadelphia, 1975, pp. 451–481.
[33] ASF, Apache tomcat configuration reference—the HTTP connector, 2011. http://tomcat.apache.org/tomcat-6.0-doc/config/http.html.

Jing Zhao received Ph.D. (2006) degree in Computer Science and Technology from the Harbin Institute of Technology of China. In 2010 he was with the Department of Electrical and Computer Engineering at Duke University, Durham, NC, working as a Postdoc under supervision of Dr. Kishor Trivedi. She is currently a Professor in the School of Computer Science at Harbin Engineering University of China. Her research interests include reliability engineering, software aging theory, and dependability modeling.

YanBin Wang received the Ph.D. (2006) degree in Industrial Engineering from the Harbin Institute of Technology of China. In 2010 he was with the Department of Electrical and Computer Engineering at Duke University, Durham, NC, working as a research associate. He is currently an Associate Professor in the Industrial Engineering Department of the Harbin institute of Technology of China. His research interests include quality management, scheduling and optimization.



GaoRong Ning received the B.S. and M.S. degrees in Applied Mathematics from Capital Normal University, in 2006 and 2010. Now he is a sixth-year doctoral candidate in the School of Automation Science and Electrical Engineering, Beihang University, Beijing. His research interests include software reliability and differential game.

Kishor S. Trivedi holds the Hudson Chair in the Department of Electrical and Computer Engineering at Duke University, Durham, NC. He has been on the Duke faculty since 1975. He is the author of a well known text entitled, Probability and Statistics with Reliability, Queuing and Computer Science Applications, published by Prentice-Hall; a thoroughly revised second edition (including its Indian edition) of this book has been published by John Wiley. He has also published two other books entitled, Performance and Reliability Analysis of Computer Systems, published by Kluwer Academic Publishers and Queueing Networks and Markov Chains, John Wiley. He is a Fellow of the Institute of Electrical and Electronics Engineers. He is a Golden Core Member of the IEEE Computer Society. He has published over 490 articles and has supervised 44 Ph.D. dissertations. He is the recipient of the IEEE Computer Society Technical Achievement Award for his research on Software Aging and Rejuvenation. His research interests in are in reliability, availability, performance and survivability of computer and communication systems and in software dependability. He works closely with industry in carrying our reliability/availability analysis, providing short courses on reliability, availability, and in the development and dissemination of software packages such as SHARPE, SREPT and SPNP.

Rivalino Matias Jr. received his B.S. (1994) in Informatics from the Minas Gerais State University, Brazil. He earned his M.S. (1997) and Ph.D. (2006) degrees in Computer Science and Industrial and Systems Engineering from the Federal University of Santa Catarina, Brazil, respectively. In 2008 he was with Department of Electrical and Computer Engineering at Duke University, Durham, NC, working as a research associate under supervision of Dr. Kishor Trivedi. He also works for IBM Research Triangle Park in a research related to embedded system availability and reliability analytical modeling. He is currently an Associate Professor in the School of Computer Science at Federal University of Uberlândia, Brazil. Dr. Matias has served as reviewer for IEEE Transactions on Dependable and Secure Computing, Journal of Systems and Software, and several international conferences. His research interests include operating systems, reliability engineering, software aging theory, and dependability modeling.

Kai-Yuan Cai 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. He has been a Full Professor at Beihang University (Beijing University of Aeronautics and Astronautics) since 1995. He was born in April 1965 and entered Beihang University as an undergraduate student in 1980. He received his B.S. degree in 1984, M.S. degree in 1987, and Ph.D. degree in 1991, all from Beihang University. He was a research fellow at the Centre for Software Reliability, City University, London, a visiting scholar at Purdue University (USA), and visiting professorial fellow at the University of Wollongong (Australia).

Dr. Cai has published numerous research papers. He is the author of three books: Software Defect and Operational Profile Modeling (Kluwer, Boston, 1998); Introduction to Fuzzy Reliability (Kluwer, Boston, 1996); Elements of Software Reliability Engineering (Tsinghua University Press, Beijing, 1995, in Chinese). He serves on the editorial board of the international journal Fuzzy Sets and Systems and was the editor of the Kluwer International Series on Asian Studies in Computer and Information Science. He served as program committee co-chair for the Fifth International Conference on Quality Software, the First International Workshop on Software Cybernetics and General co-chair for The Second International Symposium on Service-Oriented System Engineering. He also served as guest Editor for Fuzzy Sets and Systems (1996), the International Journal of Software Engineering and Knowledge Engineering (2006), the Journal of Systems and Software (2006), and IEEE Transactions on Reliability (2010). His main research interests include software reliability and testing, reliable flight control, and software cybernetics.