Available online at www.ijpe-online.com

International Journal of Performability Engineering

vol. 14, no. 2, February 2018, pp. 210-222
DOI: 10.23940/ijpe.18.02.p3.210222

Optimization of Software Rejuvenation Policy based on State-Control-Limit

Weichao Dang[a,b,*], Jianchao Zeng[b,c]

[a]*College of Electrical and Information Engineering, Lanzhou University of Technology, Lanzhou, 730050, China*
[b]*Division of Industrial and System Engineering, Taiyuan University of Science and Technology, Taiyuan, 030024, China*
[c] *School of Computer Science and Control Engineering, North University of China, Taiyuan, 030051, China*

Abstract

Software Rejuvenation is a proactive software control technique used to improve computing system performance when a system suffers from software aging. In this paper, a state-control-limit-based rejuvenation policy with periodical inspection has been proposed. The steady-state system availability model has been constructed based on the semi-renewal process. The steady-state probability density of degradation system stated as a function of the inspection interval and the rejuvenation threshold have been derived. The average unavailable time of when a soft failure occurs within an inspection cycle has been taken into account to calculate the steady-state system availability. Finally, the system availability with corresponding optimal inspection time interval and rejuvenation threshold have been obtained numerically.

Keywords: software aging; software rejuvenation; availability; soft failures; state-control-limit

(Submitted on November 1, 2017; Revised on December 10, 2017; Accepted on January 11, 2018)

1. Introduction

With advances in Internet technology, reliability of computer systems has emerged as a major concern for researchers. The failure of computer systems has severely affected system reliability. The failure of computer systems is usually caused by hardware or software failure, the latter being the more prominent source [13,23]. The concept of software aging has been proposed by Huang et al. [15], who point out that the performance of software that runs for a long time tends to deteriorate and the failure rate of software tends to increase, eventually causing the system to crash due to software faults, memory leaks, unreleased file locks, data corruption, storage fragments, and cumulative errors. Software aging has since been reported in UNIX workstations, OLTP DBMS servers, the Apache Web server, Sun JVM, spacecraft flight systems [7,8,12,14,24], Web application systems [18], and cloud computing infrastructures [2]. A well-known example of software aging is the failure of the US Patriot missile defense system after 100 hours of continuous operation due to the accumulation of rounding-off errors, with the system incapable of tracking and intercepting incoming Scud missiles [17].

As a significant cause of software aging, aging-related software faults are stochastic and difficult to eliminate in the software development and test phases. Therefore, errors triggered by specific user inputs occur in the operational phase [2]. Software rejuvenation has been proposed by Huang et al. [15] to prevent serious failures. Software rejuvenation terminates the running software at appropriate times and cleans the internal state of the software in order to restore the system to its pristine and relatively healthy intermediate state.

In comparison with passive repair techniques, the proactive and preventive techniques significantly reduce system unavailability time. The proactive and preventive techniques, however, may lose effectiveness because of incorrect scheduling and maintenance strategies. Therefore, a significant issue is when and how to trigger software rejuvenation.

* Corresponding author.
E-mail address: *dangweichao*@tyust.edu.cn

Two main approaches are generally used to address software aging and rejuvenation challenges: time-based approach and inspection-based approach [9].

The time-based approach assumes failure time distribution of the software system and models the state transition process of software deterioration by adopting Markov models [15], semi-Markov models [5,22,26], stochastic Petri nets [25], and other mathematical tools. It builds the rejuvenation model, establishing functions between cost and time, and between cost and workload in order to determine the optimal rejuvenation scheduling. As a black-box approach, the time-based approach does not focus on degradation mechanisms of the system [6].

The inspection-based approach determines the optimal rejuvenation scheduling through a collection of system data concerning resource usage and performance, time-series analysis, and machine learning in order to obtain the rate of resource consumption and performance degradation trends and to predict system short-term failures in accordance with actual load and resource consumption. The basic idea of inspection-based approach is to monitor system variables, namely aging indicators, which can denote the onset of software aging, and predict the occurrence of aging failures by analyzing the collected runtime data statistically. Most of the current methods have adopted the inspection-based approach to analyze some parameters of software aging and predict the time to resource exhaustion. Garg et al. [11] have proposed a metric ”estimated time to exhaustion”, which is calculated using a slope estimation technique, and this metric is used to detect and estimate the aging in the UNIX operating system. Jia et al. [16], after studying the effect of various performance indicators (e.g., response time, physical memory, and swap space) on software aging, have predicted the effect of a single parameter on software aging using principal component analysis. Avritzer et al. [3] have established a response time-based simulation model of software system performance degradation and proposed corresponding software rejuvenation algorithms according to three methods of response-time measurement.

Apart from studies where the software system degradation process is fitted by a stochastic system [3,11,16], a variety of studies have been conducted on rejuvenation modeling based on stochastic process. Meng et al. [19] have studied a periodical inspection rejuvenation policy for software systems, and assuming the probability of detected failure was a constant in any inspection interval. By contrast, we will calculate the probability distribution of the degradation state of the software system. Bobbio et al. [6] have proposed a fine-grained model by assuming that the degradation level of the system can be identified by monitoring an observable quantity. Specifically, they have presented two strategies, risk-level rejuvenation and alarm threshold, and established the steady-state unavailability of the software system by applying the theory of renewal reward processes. However, they do not take into account the effect of inspection on the system availability. Dieulle et al. [10] have studied inspection maintenance strategies for a deteriorating system, the process of which has been expressed by the Gamma process. However, they do not take into account the occurrences of the soft failure before the inspection epoch.

In this paper, we have applied the theorem of semi-renewal process to the study of software rejuvenation scheduling. We have obtained the steady-state system availability as a function of the inspection interval and rejuvenation threshold. The optimal inspection interval and rejuvenation threshold are calculated according to maximizing the system availability.

Our major contributions are as follows:

- The rejuvenation policy with periodical inspection and state-control limit has been proposed.
- The software system steady-state probability density function has been established on the basis of the semi-renewal processes.
- The probability that soft failures occur in an inspection cycle has been taken into account to model the system availability.

The remainder of this paper is organized as follows: Section 2 introduces the characteristics of the software system, rejuvenation policy, and degradation processes. In Section 3, a steady-state availability model is established, and the steady-state probability density of the degradation state as a function of inspection interval and rejuvenation threshold is derived, the mean unavailable time in the inspection cycle is calculated, and a numerical solution to the probability density function is presented. In Section 4, the design and implementation of numerical experiments used to test the proposed system are described and the optimization model is solved by using a genetic algorithm. A sensitivity analysis is further conducted, and the effectiveness of the model is verified by the experimental results. Section 5 presents our concluding remarks.

2. System description

2.1. Characteristics of the software system

Fixed and variable sampling periods are adopted to detect performance indicators that reflect the system state. Statistical analyses of performance indicators can be used to determine the health of the system at different times. When the state of the system degrades so seriously that it can no longer satisfy user requests, a failure occurs, which is called a soft failure [4]. When a soft failure occurs, the degradation level of the system is defined as D_f. The characteristics of the software system are then described as follows:

a) The healthy state of the software system can be expressed as a random variable X_t, which is associated with the occupancy of computer resources, the value of which can be obtained by running a special inspection program. The subscript t represents the running time of the software system.

b) The inspection operation requires CPU time and other resources, resulting in performance degradation; therefore, the system cannot be inspected frequently. Inspection program is triggered periodically with the interval T. The execution duration of the inspection program is denoted by T_i.

c) The software system running in a pristine healthy state is represented by $X_0 = 0$. The performance of the software system gradually degrades, and its performance after running for a long time is denoted by $0 < X_t < D_f$. When $X_t \geq D_f$, system failure occurs. The failure can only be detected by inspection.

2.2. Rejuvenation policy

Performance indictors can be obtained by periodical inspection. After performance indicators are compared with the predetermined control limits, a different recovery can be employed, which is called the control-limit-based rejuvenation policy. The specific rejuvenation policy is as follows:

a) According to the results of inspection, different decisions are made: no maintenance is conducted if $0 < X_t < D_r$; rejuvenation is pre-activated if $D_r \leq X_t < D_f$; when $X_t \geq D_f$, system recovery is reactivated, where the rejuvenation threshold is denoted by D_r.

b) The system restores to the pristine healthy state after reactive recovery and proactive-preventive rejuvenation. The recovery duration and rejuvenation duration are represented by T_f and T_r respectively.

c) If the software system fails, reactive fault tolerance mechanisms, such as restart, recovery, rollback, reordering and replay, should be implemented wherever possible to ensure high availability and data integrity of long running applications [3]. The duration of reactive recovery is much greater than that of proactive rejuvenation, i.e. $T_f >> T_r$.

2.3. Degradation processes of software system

The software degradation process can be denoted by a non-decreasing continuous random process $\{X_t, t > 0\}$. When we suppose that the inspection program has no effect on degradation state, the given degradation state can be obtained [3]. The evolution of the system exhibits periodic characteristics if the restoring-as-good-as-new strategies described in Section 2.2 are adopted.

After inspection and rejuvenation at periodical discrete maintenance epochs $t_k = k\delta t$, the discrete-time stochastic process $X_k = X_{t_k}$ can be observed. The maintenance period δt is generally imposed in light of both system characteristics and the performance environment. Hereafter, this maintenance period δt is assigned the value 1 (in arbitrary time units). The interference of the evolution of the software system with inspection and rejuvenation behavior can be denoted by the stochastic process $\{X_k\}$, which is an embedded Markov chain in continuous state space. $X_k = 0$ implies that the system is in a new state at this point, called the regenerated point, and $X_k > 0$ means that the system is in the degradation state at another point, called the semi-regenerated point. The period between two successive inspection epochs is called the semi-renewal cycle, represented by S. Figure 1 shows the realization of a representative sample path of the software system state. The degradation

process of the software system can be approximated by a semi-renewal process. Since the repair decisions are different at the semi-regenerated point, the length of each semi-renewal cycle is similarly different:

a) If the degradation level does not exceed the rejuvenation threshold (denoted by event E_1), then no rejuvenation is performed. The length of the semi-renewal cycle is $T + T_i$, which hereafter is denoted by $S|E_1$;

b) If the degradation level exceeds rejuvenation threshold, then proactive rejuvenation is performed (denoted by event E_2). The length of the semi-renewal cycle is $T + T_i + T_r$, which hereafter is denoted by $S|E_2$;

c) If the degradation level exceeds failure threshold, then reactive recovery is performed (denoted by event E_3). The length of the semi-renewal cycle is $T + T_i + T_f$, which hereafter is denoted by $S|E_3$.

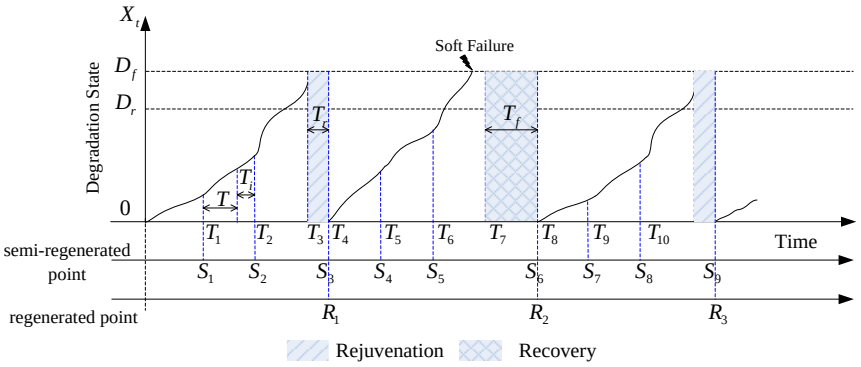


Figure 1. Degradation process of the software systems

3. Software system availability model based on periodical inspection and state control limit

3.1. Steady-state availability of the software system

$\{X_k, k = T_0, T_1, T_2, ..., T_n\}$ is a Markov chain. Each semi-regenerated point corresponds to an event associated with the repair decision ($E_i, i = 1, 2, 3$). When $n \to \infty$, there exists the long-range frequency of the event ($E_i, i = 1, 2, 3$), also called the steady-state probability $P\{E_i\}$, which satisfies as Equation (1):

$$\begin{aligned} P\{E_i\} &\approx \frac{\sum I\{E_i\}}{n} \\ \sum_{i=1}^{3} P\{E_i\} &= 1 \end{aligned} \tag{1}$$

where $I\{E_i\} = \begin{cases} 1 & if\ events\ E_i\ occurs \\ 0 & otherwise \end{cases}$, $\sum I\{E_i\}$ indicates the number of occurrences of the event $E_i, i = 1, 2, 3$.

The mean semi-renewal cycle of the software system is expressed as Equation (2):

$$\begin{aligned} E(S) &= \sum_{i=1}^{3} E(S|E_i)P\{E_i\} \\ &= E(T + T_i)P\{E_1\} + E(T + T_i + T_r)P\{E_2\} + E(T + T_i + T_f)P\{E_3\} \\ &= T(P\{E_1\} + P\{E_2\} + P\{E_3\}) + E(T_i)(P\{E_1\} + P\{E_2\} + P\{E_3\}) \\ &+ E(T_r)P\{E_2\} + E(T_f)P\{E_3\} \\ &= T + E(T_i) + E(T_r)P\{E_2\} + E(T_f)P\{E_3\} \end{aligned} \tag{2}$$

The last step of Equation (2) is simplified according to Equation (1). In Equation (2), S represents the length of a semi-renewal cycle; $E(S)$ represents the expectation of S; $E(S|E_i, i = 1, 2, 3)$ represents the mean length of the semi-renewal cycles corresponding to the three different repair decisions respectively.

In a semi-renewal cycle, the system is not available in the following scenarios:

a) The system is not available during rejuvenation and failure recovery.

b) Inspection program results in unavailability of the system to a certain extent due to CPU time and some resources occupying during its running. The degree of effect is represented by the impact factor $a(0 < a < 1)$.

c) At the end of the inspection program running, if $X_k \geq D_f$, which means that the degradation level of the system at $T_{k'}$ $(T_{k-1} < T_{k'} \leq T_k)$ moment has reached D_f, then the system is also considered unavailable in the interval $[T_{k'}, T_k]$. The unavailable duration is represented by $T - T_{k'}$.

Thus, the average unavailable time of the system in the semi-renewal cycle is expressed as Equation (3):

$$\begin{aligned} E(D) &= \sum_{i=1}^{3} E(D|E_i)P\{E_i\} \\ &= E(aT_i)P\{E_1\} + E(aT_i + T_r)P\{E_2\} + E(T - T_{k'} + aT_i + T_f)P\{E_3\} \\ &= aE(T_i) + E(T_r)P\{E_2\} + (E(T_f) + E(T - T_{k'}))P\{E_3\} \end{aligned} \tag{3}$$

where D is the length of the unavailable time in a semi-renewal cycle and $E(D)$ indicates the mean unavailable time length of the semi-renewal cycle given that the steady-state probability of each event is $P\{E_i\}$.

According to the theorem of semi-renewal processes, the steady-state availability of the system is approximated by its availability in a semi-renewal cycle as Equation (4):

$$A_\infty \approx A(S) = 1 - \frac{E(D)}{E(S)} = 1 - \frac{aE(T_i) + E(T_r)P\{E_2\} + (E(T_f) + E(T - T_{k'}))P\{E_3\}}{T + E(T_i) + E(T_r)P\{E_2\} + E(T_f)P\{E_3\}} \tag{4}$$

Obviously, the steady-state availability of the system is affected by the inspection interval T and the rejuvenation threshold D_r. In the rejuvenation decision-making process, if the inspection interval T is too short, the inspection program runs frequently and system unavailability time increases. With a short T, the probability that the system state exceeds the rejuvenation threshold ($P\{E_2\}$) increases. Correspondingly, the probability that the system state exceeds the failure threshold ($P\{E_3\}$) decreases. On the contrary, if the inspection interval T is too long, the probability that failure occurs ($P\{E_3\}$) increases. If the rejuvenation threshold D_r is set too low, rejuvenation decision becomes frequent, and the unavailable time of the system increases, whereas the probability that the system fails ($P\{E_3\}$) and the unavailable time increase if the rejuvenation threshold D_r is set too high.

It has been found that the inspection interval and the rejuvenation threshold affect the probability of rejuvenation and recovery as well as the availability of the system. The best strategy, therefore, is to choose the optimal inspection interval and rejuvenation threshold to maximize the steady-state availability of the system.

If the steady-state probability density function of the degradation state can be obtained, then the probability that different maintenance decisions are made, i.e., $P\{E_2\}$ and $P\{E_3\}$ in Equation (4), can be derived by integrating the probability density function. Moreover, the mean unavailability time, i.e., $E(T - T_{k'})$, needs to be calculated when a soft failure occurs in an inspection cycle. The different scenarios are analyzed at the semi-renewal points, also called the maintenance decision points, and hence a steady-state probability density function of the software degradation processes is obtained in Section 3.2. Since a soft failure of the system can only be observed after inspection, the length of the mean unavailability time in the inspection cycle needs to be determined. The details are described in Section 3.3.

3.2. The steady-state probability density function of the software

As described in Section 2.3, the state of the system at the semi-renewal point is only related to the previous point; therefore, $\{X_k, k = T_0, T_1, T_2, ..., T_n\}$ is an embedded Markov chain in continuous state space. According to the steady-state probability density formula of the irreducible, ergodic Markov chain $\pi(x) = \int \pi(y)K(y,x)\mathrm{d}y$, the steady-state probability density function of the degradation state can be derived. $K(y,x)$, also called the transition probability kernel ([20], p.65), is the

conditional probability density from the previous semi-renewal point to the given one. The state at the beginning of the semi-renewal cycle is denoted by y and the state at the end of the semi-renewal cycle by x. The steady-state probability density function of the degradation state is represented by $\pi(x)$. The state at the beginning of the semi-renewal cycle consists of two scenarios. In one scenario, if $y \geq D_r$, then rejuvenation or recovery needs to be carried out, and thus the system is restored to its pristine healthy state. In the other scenario, if $0 < y < D_r$, then no maintenance is needed. The two scenarios are analyzed as follows:

It is assumed that the increase in degradation of the software system in unit time is represented by a random variable $\triangle X$, the probability density function of which is $f(x)$. According to the different rejuvenation decision-making of the software system at the semi-renewal points, there are three different semi-renewal cycles of the system, i.e.$T + T_i$, $T + T_i + T_r$ and $T + T_i + T_f$. In the latter two cycles, there is no degradation during the rejuvenation and failure recovery of the system. Therefore, only the length of the degradation in the semi-renewal cycle is considered, the length being $T + T_i$ time units. The probability density function of $\triangle X$ in the $T + T_i$ time units is the $T + T_i$-th convolution of $f(x)$, i.e.$f^{(T+T_i)}(x)$.

a) The software system can be restored to the pristine state by rejuvenation or by failure recovery under the rejuvenation strategy mentioned in section 2.2. The probability that the software system is in the pristine state at the beginning of the semi-renewal cycle is the sum of the probabilities of these two events (rejuvenation and failure recovery), i.e. $P(y = 0) = P\{E_2\} + P\{E_3\} = 1 - P\{E_1\} = 1 - \int_0^{D_r} \pi(y)\mathrm{d}y$. The conditional probability density of the state x at the end of semi-renewal cycle is $f^{(T+T_i)}(x)$, where $f^{(T+T_i)}(x)$ is the $T + T_i$-th convolution of $f(x)$.

b) If $0 < y < D_r$ at the beginning of the semi-renewal cycle, then the degradation state x at the end of the semi-renewal cycle has two scenarios. One scenario is $x < D_r$, i.e. $0 < y < x < D_r$, and the other scenario is $x > D_r$. The probability of the first scenario is expressed as $\int_0^x \pi(y)dy$ and the probability of the second is expressed as $\int_0^{D_r} \pi(y)\mathrm{d}y$. The probability of $0 < y < D_r$, therefore, is expressed as $\int_0^{min(x,D_r)} \pi(y)\mathrm{d}y$. The conditional probability density of the state x at the end of semi-renewal cycle is $f^{(T+T_i)}(x - y)$, where $f^{(T+T_i)}(x - y)$ is the $T + T_i$-th convolution of $f(x - y)$.

Integrating the above-mentioned two scenarios, the steady-state probability density function of the degradation state can be derived as Equation (5):

$$\pi(x) = \left[1 - \int_0^{D_r} \pi(y)\mathrm{d}y\right] f^{(T+T_i)}(x) + \int_0^{min(x,D_r)} \pi(y) f^{(T+T_i)}(x - y)\mathrm{d}y \tag{5}$$

The steady-state probability function of the rejuvenation (E_2) and the failure recovery (E_3) can be obtained by integrating the steady-state probability density function $\pi(x)$ as Equation (6) and Equation (7).

$$P\{E_2\} = \int_{D_r}^{D_f} \pi(x)\mathrm{d}x \tag{6}$$

$$P\{E_3\} = \int_{D_f}^{\infty} \pi(x)\mathrm{d}x \tag{7}$$

3.3. Mean unavailable time in the inspection cycle

If the degradation level reaches D_f at the epoch $T_{k'}$ in the inspection cycle, then the unavailable time is $T - T_{k'}$. The probability that the event occurs is represented by $Pr(X_{T_{k'}} = D_f)$. The mean unavailable time in the inspection cycle is expressed as Equation (8):

$$E(T - T_{k'}) = \sum_{T_{k'}=1}^{T-1} (T - T_{k'}) Pr(X_{T_{k'}} = D_f) \tag{8}$$

The event $Pr(X_{T_{k'}} = D_f)$ is equivalent to the composite event: the system runs during the first $T_{k'}$ time units of the cycle and fails between t_k and t_{k+1}. The system remains in the failure state during the last $T - T_{k'}$ periods of the cycle. Let

$H(T_{k'}|z;T)$ be the probability that the available time equals $T_{k'}$ time units given that the deterioration value after maintenance equals z and the semi-renewal cycle is T time units. $H(T_{k'}|z;T)$ can be expressed by the following Equation (9):

$$H(T_{k'}|z;T) = f^{(T_{k'})}(D_f - z), 0 < T_{k'} < T \tag{9}$$

According to the analysis of Section 3.2, the probability that the software system is in the pristine state at the beginning of the semi-renewal cycle is represented as Equation (10):

$$P(y=0) = P\{E_2\} + P\{E_3\} = 1 - P\{E_1\} = 1 - \int_0^{D_r} \pi(y)\mathrm{d}y \tag{10}$$

The probability that the software system is in the degradation state at the beginning of the semi-renewal cycle is represented as Equation (11):

$$P(0 < y < D_r) = P\{E_1\} = \int_0^{D_r} \pi(y)\mathrm{d}y \tag{11}$$

According to the theorem of total probability, we can write Equation (12)

$$\begin{aligned} &Pr(X_{T_{k'}} = D_f) = Pr(X_{T_{k'}} = D_f, y = 0) + Pr(X_{T_{k'}} = D_f, 0 < y < D_r) \\ &= P(y=0)Pr(X_{T_{k'}} = D_f | y = 0) + P(0 < y < D_r)Pr(X_{T_{k'}} = D_f | 0 < y < D_r) \\ &= P(y=0)H(T_{k'}|0;T) + P(0 < y < D_r)H(T_{k'}|y;T) \\ &= [1 - \int_0^{D_r} \pi(y)\mathrm{d}y] f^{(T_{k'})}(D_f) + \int_0^{D_r} \pi(y) f^{(T_{k'})}(D_f - y)\mathrm{d}y \end{aligned} \tag{12}$$

The average unavailable time in the inspection cycle is represented as Equation (13):

$$E(T - T_{k'}) = \sum_{T_{k'}=1}^{T-1} (T - T_{k'})([1 - \int_0^{D_r} \pi(y)\mathrm{d}y] f^{(T_{k'})}(D_f) + \int_0^{D_r} \pi(y) f^{(T_{k'})}(D_f - y)\mathrm{d}y) \tag{13}$$

After substitution of Equation (5) into Equation (6) and Equation (7), the probability of the rejuvenation ($P\{E_2\}$) and failure recovery ($P\{E_3\}$) can be obtained respectively in the semi-renewal cycle. After substitution of Equation (13) together with Equation (6) and Equation (7) into Equation (4), the steady-state availability A_∞ of the software system can be derived.

3.4. Numerical solution to the steady-state probability density function

Equation (5), which expresses steady-state probability density, is an implicit integral equation, the analytical solution to which is difficult to obtain. Numerical methods are used to obtain the approximate solution.

Let $g(x) = f^{(T+T_i)}(x)$, Equation (5) can be rewritten as

$$\pi(x) = g(x) - g(x)\int_0^{D_r} \pi(y)\mathrm{d}y + \int_0^{min(x,D_r)} \pi(y)g(x-y)\mathrm{d}y \tag{14}$$

According to the numerical solutions to the integral equation and the quadrature rule:

$$\int_a^b y(x)\mathrm{d}x = \sum_{j=1}^{N} y(x_j)\delta x \tag{15}$$

When we apply Equation (15), the numerical expression of Equation (14) becomes Equation (16):

$$\pi(x) = g(x) - g(x)h \sum_{j=1}^{D_r/h} \pi(jh) + h \sum_{j=1}^{min(x/h, D_r/h)} \pi(jh)g(x - jh) \tag{16}$$

When we define $p = x/h, q = D_r/h$, the approximate equation at the quadrature point can be expressed as Equation (17):

$$\pi(ph) = g(ph) - g(ph)h \sum_{j=1}^{q} \pi(jh) + h \sum_{j=1}^{min(p,q)} \pi(jh)g(ph - jh) \tag{17}$$

After transposition, the following equation can be obtained:

$$\pi(ph) + hg(ph) \sum_{j=1}^{q} \pi(jh) - h \sum_{j=1}^{min(p,q)} \pi(jh)g(ph - jh) = g(ph) \tag{18}$$

In Equation (18), D_{max} is the censored number replacing ∞, and $i_{max} = \lfloor D_{max}/h \rfloor$ is the maximum number of numerical integrations. The matrix notation of Equation (18) becomes

$$\pi(\mathbf{I} + \mathbf{K^1} - \mathbf{K^2}) = \mathbf{g} \tag{19}$$

Solving Equation (19), we can derive

$$\pi = \mathbf{g}(\mathbf{I} + \mathbf{K^1} - \mathbf{K^2})^{-1} \tag{20}$$

π in Equation (20) is the vector to be solved. In Equation (20), $\pi = (\pi(h), \pi(2h), ..., \pi(i_{max}h))'$ and $\mathbf{f^{(N)}} = (f(h), f(2h), ..., f(i_{max}h))'$, where superscript $'$ represents the vector transpose. $\mathbf{I}$ in Equation (20) represents the $i_{max} \times i_{max}$-dimensional identity matrix. $\mathbf{K^1}$ and $\mathbf{K^2}$ in Equation (20) represent the matrices $\left\{K^1_{pj}\right\}_{i_{max} \times i_{max}}$ and $\left\{K^2_{pj}\right\}_{i_{max} \times i_{max}}$ respectively.

where $K^1_{pj} = \begin{cases} h * g(ph) & p = 1, ..., i_{max} \\ 0 & else \end{cases}$, $K^2_{pj} = \begin{cases} h * g(ph - jh) & p = 1, ..., i_{max}, j = 1, ..., max(p, q) \\ 0 & else \end{cases}$.

3.5. The rejuvenation decision model

The above-mentioned analyses show that the inspection interval and rejuvenation threshold need to be adjusted in order to maximize the steady-state availability of the system, hence influencing the probability of rejuvenation and recovery. The rejuvenation problem under discussion, therefore, can be modelled as Equation (21), which describes a single-objective optimization problem with constraints, namely:

$$\begin{aligned} &max A(T, D_r) \\ &s.t. T \in \mathbb{R}^+ \\ &\quad 0 < D_r < D_f \end{aligned} \tag{21}$$

As shown in Equation (21), the non-linear single-objective optimization model with constraints can be approximately solved by the heuristic algorithm. The genetic algorithm (GA) can be used to handle any objective function and constraints, whether it is linear or non-linear, discrete or continuous [1]. Therefore, the GA is chosen to solve the maintenance decision model of the system. The approximate optimal solution to the model can be solved through the joint optimization of the decision variables.

4. Numerical experiments

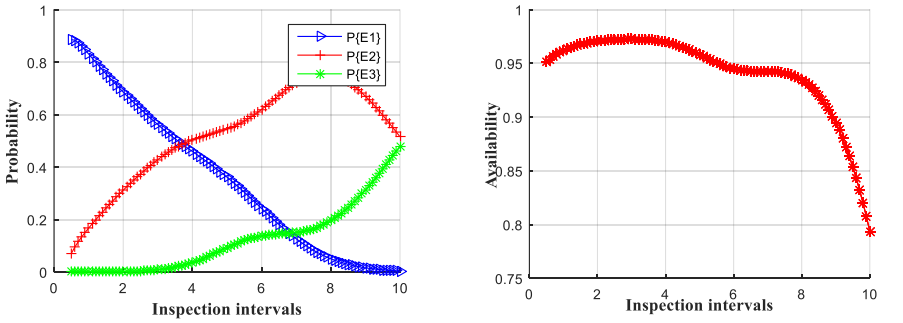
The degradation process of the software system can be modeled as a Markov process. It can also be modeled as a gamma process with independent, non-negative increments with a gamma distribution [6,10,21]. We chose gamma distribution as the

experimental distribution of the software system degradation processes. If we suppose that the software system degradation increment $\triangle_{(k,k+1)}X$ accords with $\Gamma(\alpha,\beta)$ within unit time and the pdf of $\triangle_{(k,k+1)}X$ is $f(x|\alpha,\beta)=\frac{\beta^{\alpha}}{\Gamma(\alpha)}x^{\alpha-1}e^{-\beta x},x>0$, where $\Gamma(\alpha)=\int_0^{\infty}x^{\alpha-1}e^{-x}dx,\alpha>0$, then the degradation increment accords with $\Gamma(n\alpha,\beta)$ within n time units. Gamma distribution shape parameters $\alpha=2$, $\alpha=4$ and scale parameter $\beta=1$ are selected in this study. The differences in the values of α/β reflect different degradation rates of the system. The failure threshold of the system is denoted by $D_f=20$; the mean running time of inspection program is denoted by $E(T_i)=0.02$; the impact factor is denoted by $a=0.8$; and the mean rejuvenation and recovery time are denoted by $E(T_r)=0.14$ and $E(T_f)=1$ respectively.

4.1. Verification of the optimization model

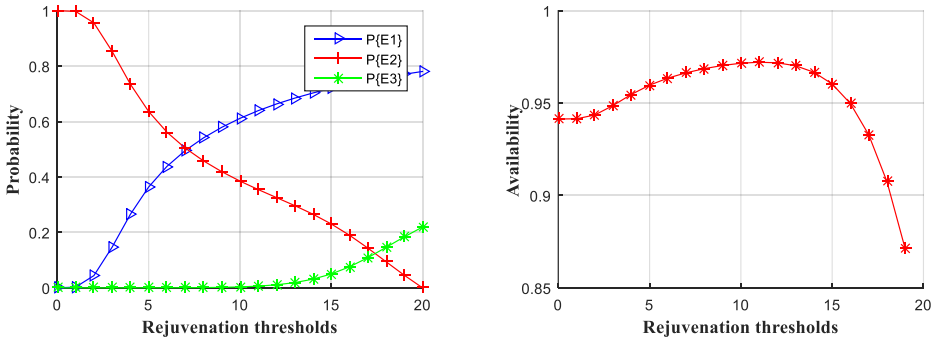
System availability is calculated when no rejuvenation strategies are adopted. If $\alpha=2$, the mean degradation rate can be derived as $\alpha/\beta=2$ and the mean time to failure (MTTF) as 10, which shows that the steady-state availability of the system was $10/11=0.909091$; if $\alpha=4$, the mean degradation rate can be derived as $\alpha/\beta=4$ and the MTTF as 5, which shows that the steady-state availability of the system is $5/6=0.833333$.

The effect of each decision variable on the optimization objectives is analyzed to show that for each decision variable, there exists an optimal value that optimizes the objective. In this way, the correctness of the maintenance optimization model is verified. Figure 2(a) shows the probability trends of three maintenance decisions with varying inspection intervals (T). Figure 2(b) shows the trends of steady-state availability with varying inspection intervals (T) on the condition of rejuvenation threshold $D_r=10.2$. Obviously, the steady-state availability of the system has a maximum value with gradual increase in the inspection interval. Due to the effects of the inspection interval on availability, the system is inspected frequently if the inspection interval is too short, which results in lower system availability. With the increase in T, the probability of rejuvenation and recovery increases. Trade-offs between the probability of rejuvenation and recovery and the probability of the inspection are obtained at a special time, when the steady-state availability of the system is maximal. As T continues to increase, the probability of recovery increases accordingly; meanwhile, the steady-state availability decreases. Figure 3(a) shows the probability trends of occurrence of the three maintenance decisions with varying rejuvenation thresholds D_r. Figure 3(b) shows the trends of steady-state availability with varying rejuvenation thresholds in the inspection interval $T=2.5$. The smaller the value of D_r, the higher the probability of rejuvenation and the lower the probability of recovery, and the greater the availability of the system; As D_r increases, the probability of rejuvenation decreases, whereas the probability of recovery increases, and steady-state availability decreases drastically.



(a) Effects of inspection intervals on repair decision (b) Effects of inspection intervals on availability

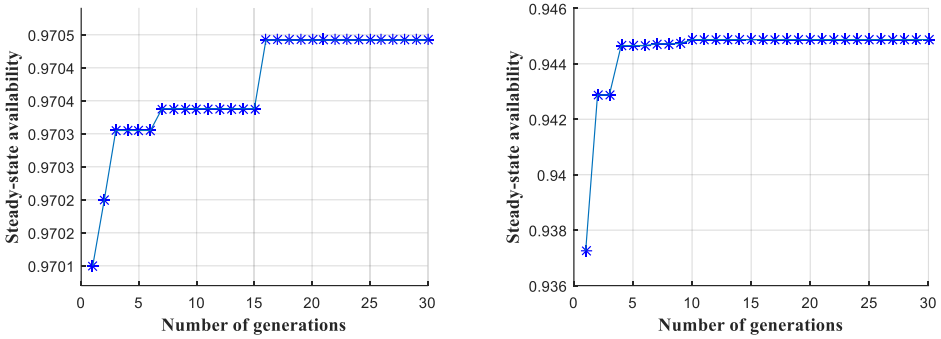
Figure 2. Effects of inspection intervals on repair decision and availability



(a) Effects of rejuvenation thresholds on repair decision (b) Effects of rejuvenation thresholds on availability

Figure 3. Effects of rejuvenation thresholds on repair decision and availability

The probability density function (see Equation (20)) is solved by applying the numerical solutions proposed in Section 3.4. $D_{max} = 10D_f$ represents censored number replacing ∞, which is the upper bound of gamma distribution pdf. If the interval $[0, 10D_f]$ is equally divided into 2,000 parts then $h = 0.1$. The parameters of the genetic algorithm are configured as follows: the number of individuals is 30; the maximum number of generations is 30; the generation gap is 0.8; the mutation probability is 0.2; and the crossover probability is 0.8. The upper bound of the inspection interval $T_{max} = 10$. Figure 4(a) and Figure 4(b) show examples of the optimization result of GA when $\alpha = 2, \beta = 1$ and when $\alpha = 4, \beta = 1$. If $\alpha = 2$, the approximate optimal solutions obtained are $T^* = 2.391666$ and $D_r^* = 10.197959$. Accordingly, the maximum availability is 0.970442; If $\alpha = 4$, the approximate optimal solutions obtained are $T^* = 1.197978$ and $D_r^* = 10.974221$. Accordingly, the maximum availability is 0.944874. Obviously, the steady-state availability of the system greatly increases by adopting the above-mentioned periodical inspection and rejuvenation policy. It has been found that if the degradation rate of the system increases, the inspection interval significantly decreases to increase the probability of rejuvenation and reduce the probability of recovery. Accordingly, greater availability is obtained.



(a) $\alpha = 2, \beta = 1$ (b) $\alpha = 4, \beta = 1$

Figure 4. An example of optimization result of GA

4.2. Sensitivity analysis

There are four parameters in the optimization model: the mean rate of degradation, the inspection impact factor, the rejuvenation time, and the recovery time. These parameters are increased and reduced respectively in our experiments. The sensitivity of the optimization results is analyzed, and the optimization results are shown in Tables 1 through 4.

Table 1 shows that with an increase in degradation rate, the probability of system failure increases; therefore, the inspection interval and the rejuvenation threshold are supposed to be reduced, and the probability of rejuvenation is supposed to increase to obtain the maximum steady-state availability. However, with the inspection interval and rejuvenation threshold decreasing, the probability of the inspection and the probability of the rejuvenation increase, and the optimized system availability decrease as a consequence.

Table 1. Effects of average rate of degradation on the optimal repair policy
($a = 0.8, E(T_i) = 0.02, E(T_r) = 0.14, E(T_f) = 1, D_f = 20$)

α, β	T^*	D_r^*	A_∞
1,1	3.699000	10.360515	0.983971
1.2,1	3.260328	10.437265	0.981192
1.4,1	2.890060	10.577313	0.978479
2,1	**2.391666**	**10.197959**	**0.970442**
4,1	1.162979	8.975536	0.923211

Table 2 shows that with increasing inspection impact factor, the effect of inspection on system availability increases; therefore, the inspection interval is supposed to increase to reduce the frequency of inspection. However, the decrease in inspection frequency results in an increase in the probability of failure occurrences. Therefore, the rejuvenation threshold is supposed to decrease in order to increase the probability of rejuvenation, and the optimized system availability is finally obtained.

Table 2. Effects of impact factors on the optimal repair policy
($\alpha = 2, \beta = 1, E(T_i) = 0.02, E(T_r) = 0.14, E(T_f) = 1, D_f = 20$)

a	T^*	D¤r	A1
0.2	1.291020	12.525384	0.977366
0.4	1.965817	10.715765	0.974226
0.8	**2.391666**	**10.197959**	**0.970442**
1	2.471960	9.850069	0.968797

Table 3 shows that with the increase in rejuvenation time and its gradually approaching recovery time, the optimal steady-state availability is obtained only by reducing the inspection interval and increasing the rejuvenation threshold to reduce the probability of occurrence of recovery.

Table 3. Effects of rejuvenation time on the optimal repair policy
($\alpha = 0.8, \beta = 2, E(T_i) = 1, E(T_r) = 0.02, E(T_f) = 1, D_f = 20$)

Tr	T¤	D¤r	A1
0.07	2.787900	8.231529	0.980962
0.14	**2.391666**	**10.197959**	**0.970442**
0.28	1.799883	12.326432	0.952641
0.56	1.283085	14.768429	0.922753

Table 4 shows that with increasing recovery time, the optimal steady-state availability is obtained only by reducing the probability of recovery. The rejuvenation threshold is supposed to decrease in order to increase the probability of rejuvenation. It has been found that with the increase in recovery time, the inspection interval does not change significantly, and optimal availability is obtained mainly by adjusting the rejuvenation threshold. The effectiveness of rejuvenation policy with state-control limit has hence been verified.

Table 4. Effects of failure repair time on the optimal repair
($\alpha = 0.8, \beta = 2, E(T_i) = 1, E(T_r) = 0.02, E(T_f) = 0.14, D_f = 20$)

Tf	T¤	D¤r	A1
0.5	2.298362	10.864307	0.971103
1	**2.391666**	**10.197959**	**0.970442**
1.5	2.190316	10.380928	0.969982
2	2.280267	9.718824	0.969490

5. Conclusions

In software systems, there are software faults, called aging-related bugs, which are transient, non-deterministic, and difficult to characterize. Performances of software systems tend to degrade if they run for a long time. Software rejuvenation is an effective approach to counteract software aging. A system fails because it ages over time. Inspection programs assess the state of the software by collecting system data based on resource usage and performance, and by conducting statistical analyses. Software systems evolve from healthy operation in pristine states to gradual degradation and final failure, which can be modeled by a Gamma process. The degradation level of the system can be obtained by inspection. The incremental degradation of the system accords with a gamma distribution. If the degradation of the system exceeds the rejuvenation threshold, rejuvenation programs are initiated automatically; therefore, the software system is recovered to its pristine healthy state.

The following has been discussed in this study:

1. Rejuvenation policy with periodical inspection and state-control-limit has been proposed according to performance degradation characteristics of the software system. The state evolution process, interfered by inspection and rejuvenation behavior, has been modeled as an embedded Markov chain in continuous state space; hence, the analytical expression of steady-state availability has been derived.

2. The steady-state probability density function of the system state under varying inspection intervals and rejuvenation thresholds has been derived, and a numerical solution to the function has been proposed.

3. The mean unavailability time has been derived in the inspection cycle when soft failures occur, which makes the availability model of the software system more realistic and more practical.

Software rejuvenation policy for stand-alone systems has been studied in this paper. Further research on software rejuvenation policy can be extended to multi-server systems, such as cluster and cloud computing infrastructures, by integrating redundancy with virtualization technology.

Acknowledgements

This research was supported in part by the Chinese National Natural Science Foundation under Grant No. 61573250, the Key Research and Development Program of Shanxi Province under Grant No. 201703D121042-1, the Key Science and Technology Program of Shanxi Province under Grant No. 20130321006-01, the Youth Foundation of Shanxi Province under Grant No. 201601D021065 and the PhD Research Startup Foundation of TYUST under Grant No. 20152021.
Availability

References

1. T. Amudha and B. L. Shivakumar, "Parameter Optimization in Genetic Algorithm and Its Impact on Scheduling Solutions." Springer India, 2015
2. J. Araujo, R. Matos, V. Alves, P. Maciel, F.V.D. Souza, R. Matias, and K.S. Trivedi, "Software Aging in the Eucalyptus Cloud Computing Infrastructure: Characterization and Rejuvenation," *ACM Journal on Emerging Technologies in Computing Systems*, vol. 10, no. 1, pp. 1557-1564, 2014
3. A. Avritzer, A. Bondi, M. Grottke, K.S. Trivedi, and E.J. Weyuker, "Performance Assurance via Software Rejuvenation: Monitoring, Statistics and Algorithms," in *International Conference on Dependable Systems and Networks, DSN 2006*, pp. 435-444, 2006
4. A. Avritzer and E. J. Weyuker, "Monitoring Smoothly Degrading Systems for Increased Dependability," *Empirical Software Engineering*, vol. 2, no. 2, pp. 59-77, 1997
5. Y. Bao, X. Sun, and K. S. Trivedi, "A Workload-based Analysis of Software Aging, and Rejuvenation," *IEEE Transactions on Reliability*, vol. 54, no. 3, pp. 541-548, 2005
6. A. Bobbio, M. Sereno, and C. Anglano, "Fine Grained Software Degradation Models for Optimal Rejuvenation Policies," *Performance Evaluation*, vol. 46, no. 1, pp. 45-62, 2001
7. K. J. Cassidy, K. C. Gross, and A. Malekpour, "Advanced Pattern Recognition for Detection of Complex Software Aging Phenomena in Online Transaction Processing Servers," in *Proceedings of International Conference on Dependable Systems and Networks, DSN 2002*, pp. 478-482, 2002
8. D. Cotroneo, S. Orlando, R. Pietrantuono, and S. Russo, "A Measurement-based Ageing Analysis of the JVM," *Software Testing Verification & Reliability*, vol. 23, no. 3, pp. 199–239, 2013
9. D. Cotroneo, R. Pietrantuono, S. Russo, and K. Trivedi, "How Do Bugs Surface? A Comprehensive Study on the Characteristics of Software Bugs Manifestation," *Journal of Systems & Software*, vol. 113, no. 27-43, 2016
10. L. Dieulle, C. Berenguer, A. Grall, and M. Roussignol, "Continuous Time Predictive Maintenance Scheduling for a Deteriorating System," *IEEE Transactions on Reliability* vol. 51, no. 2, pp. 150-155, 2001
11. S. Garg, A. Puliafito, M. Telek, and K. Trivedi, "Analysis of Preventive Maintenance in Transactions Based Software Systems," *IEEE Transactions on Computers*, vol. 47, no. 1, pp. 96-107, 1998
12. S. Garg, A. van Moorsel, K. Vaidyanathan, and K.S. Trivedi, "A Methodology for Detection and Estimation of Software Aging," in *Proceedings of Ninth International Symposium on Software Reliability Engineering*. pp. 283-292, 1998
13. J. Gray and D.P. Siewiorek, "High-Availability Computer Systems," *Computer*, vol. 24, no. 9, pp. 39-48, 1991
14. M. Grottke, L. Li, K. Vaidyanathan, and K.S. Trivedi, "Analysis of Software Aging in a Web Server," *Discussion Papers*, vol. 55, no. 3, pp. 411 - 420, 2005
15. Y. Huang, C. Kintala, N. Kolettis, and N.D. Fulton, "Software Rejuvenation: Analysis, Module and Applications," in *proceedings of 25 International Symposium, Fault-Tolerant Computing, FTCS-25*. pp. 381-390, 1995
16. Y.F. Jia, X.E. Chen, L. Zhao, and K.Y. Cai, "On the Relationship between Software Aging and Related Parameters," in *International Conference on Quality Software*, pp. 241-246, 2008
17. E. Marshall, "Fatal Error: How Patriot Overlooked a Scud," *Science*, vol. 255, no. 5050, pp. 1347-1347, 1992
18. R. Matias, A. Andrzejak, F. Machida, D. Elias, and K. Trivedi, "A Systematic Differential Analysis for Fast and Robust Detection of Software Aging," 2014
19. H. Meng, J. Liu, X. Hei, H. Meng, J. Liu, and X. Hei, "Modeling and optimizing periodically inspected software rejuvenation policy based on geometric sequences," *Reliability Engineering & System Safety*, vol. 133, no. 133, pp. 184-191, 2015
20. S. Meyn and R. L. Tweedie, "Markov Chains and Stochastic Stability." Springer-Verlag, 1993
21. J. M. V. Noortwijk, "A Survey of the Application of Gamma Processes in Maintenance," *Reliability Engineering & System Safety*, vol. 94, no. 1, pp. 2-21, 2009
22. K. Rinsaka and T. Dohi, "Toward High Assurance Software Systems with Adaptive Fault Management," *Software Quality Journal*, vol. 24, no. 1, pp. 1-21, 2016
23. M. Sullivan and R. Chillarege, "Software Defects and Their Impact on System Availability-A Study of Field Failures in Operating

Systems," in *Proc. Twenty-First International Symposium on Fault-Tolerant Computing, FTCS-21*, pp. 2-9, 1991
24. A. T. Tai, L. Alkalai, and S.N. Chau, "On-board Preventive Maintenance: A Design-oriented Analytic Study for Long-life Applications," *Performance Evaluation*, vol. 35, no. 3–4, pp. 215-232, 1999
25. D. Wang, W. Xie, and K.S. Trivedi, "Performability Analysis of Clustered Systems with Rejuvenation under Varying Workload," *Performance Evaluation*, vol. 64, no. 3, pp. 247-265, 2007
26. T. H. Zhao, Q. I. Yong, J. Y. Shen, D. Hou, and X.M. Zheng, "Application Server Multi-State Aging Model and Optimal Rejuvenation Strategy Research," *Journal of System Simulation*, vol. 19, no. 8, pp. 1705-1709, 2007

Weichao Dang obtained a M.S. degree in Computer Application Technology in 2004 from Taiyuan University of Science and Technology, China. He is an associate professor of Taiyuan University of Science and Technology. He is now pursuing his PhD in Control Theory and Control Engineering at Lanzhou University of Technology. His current research interests include software reliability, maintenance scheduling and Cloud computing.

Jianchao Zeng is a Professor and a Tutor of PhD student at North University of China. He received his PhD in System Engineering from Xi'an Jiaotong University in 1990. He is now the Vice President of North University of China. His current research focuses on modeling and control of complex systems, intelligent computation, swarm intelligence and swarm robotics.