Received June 16, 2020, accepted July 15, 2020, date of publication July 24, 2020, date of current version August 5, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.3011813

Planning Optimal Rejuvenation Policy for Aging Software Systems via a Two-Layer Model

JINGWEI LI[1], YONG QI[1], (Member, IEEE), GUANGHUA WANG[2], AND JINWEI LIN[1]
[1]School of Electronic and Information Engineering, Xi'an Jiaotong University, Xi'an 710049, China
[2]Xi'an Satellite Control Center, Xi'an 710043, China

Corresponding authors: Jingwei Li (lijw66@163.com) and Yong Qi (qiy@mail.xjtu.edu.cn)

This work was supported by the National Natural Science Foundation of China under Grant 61672421.

ABSTRACT Software rejuvenation is a proactive fault management technique widely used for preventing performance degradation and failures due to software aging. It works by proactively terminating the system, cleaning the internal state of the system, and restarting the system. Since rejuvenation incurs extra system overhead and downtime cost, it is important to determine the optimal time to trigger rejuvenation. This paper proposes a two-layer model to characterize the system failure behavior and rejuvenation process under aging condition, planning a time-based rejuvenation policy maximizing the system availability and minimizing downtime cost. The lower layer is a failure model that uses an analytical model and runtime measurements to build the failure distribution of the aging system. The upper layer is a rejuvenation model that takes the failure distribution from the lower layer as input to formulate the availability function and downtime cost function. Taking these two functions as optimization targets, we can obtain the optimal rejuvenation time. Compared with the traditional analytical model, the two-layer model modeling software failure considers runtime measurements, which can describe the aging behavior more accurately. In the experimental part, we comprehensively evaluate the two-layer model by studying the aging of the web search system. The results show that the two-layer model reduces unavailability by 18.18% and reduces downtime cost by 31.22% in comparison with the traditional analytical model.

INDEX TERMS Software aging, proactive rejuvenation, two-layer model, availability, downtime cost.

ACRONYMS

p.d.f.	probability density function
c.d.f.	cumulative distribution function
CTMC	Continuous-time Markov Chain
MRGP	Markov Regenerative Processes
Cox model	Cox proportional hazard model
MLE	Maximum Likelihood Estimation
ALT	Accelerated Life Test
SLA	Service Level Agreement
min	minutes
avail-Mem	available Memory
AIC	Akaike Information Criterion

NOTATION

$h_w(\cdot)$	hazard function
$F(\cdot)$	failure distribution
τ	time to trigger rejuvenation
$A(\tau)$	availability function
$C(\tau)$	downtime cost function

The associate editor coordinating the review of this manuscript and approving it for publication was Michael Lyu.

I. INTRODUCTION

Software systems tend to suffer from performance degradation after a long period of uninterrupted operation. Continued performance degradation may eventually lead to service failure or system crash. This phenomenon is called software aging in software engineering [1]. Software aging has been observed in many systems, including web server [2], Linux operating system [3], virtual machine [4], cloud computing infrastructure [5], android mobile OS [6], etc. As cloud computing matures, more and more software systems are deployed in the cloud. In the cloud, various applications share platform resources, which increases the risk of system failure. The reasons of software aging are generally considered to be software bugs, including resource leak, unreleased file locks, data corruption, and accumulation of numerical errors [7], [8]. For instance, a common software bug is that the garbage collection mechanism in software programs fails to completely release the occupied resource. It is almost impossible to achieve fully zero-bug in a software system. Even high profiled software systems, such as Office and Windows systems also need to regularly update the system to improve performance.

There is no clear indicator indicating software aging. Software aging is hidden under performance monitoring and can only be discovered after a failure occurs. For e-commerce software, unplanned failures may lead to customer loss and revenue reduction. For mission-critical industrial software, it may cause catastrophic losses. To prevent performance degradation and failures caused by aging, software rejuvenation technique is widely used, which works by proactively freeing up system resources and removing accumulated error conditions [9]. Familiar rejuvenation methods include restarting processes, restarting virtual machines, or restarting software/hardware [10], [11]. For example, a software hosted on a virtual machine can take the form of rejuvenation by restarting the virtual machine or restarting software/ hardware. In fact, it is difficult to locate and remove aging-related bugs during system operation. Software rejuvenation is an effective way to cope with software aging. Instead of removing software bugs, it regularly cleans up the operating environment to restore the software to its initial state. However, since software rejuvenation must terminate the running service, frequent rejuvenation will affect the availability of the system and incur extra downtime costs [12]. In planning rejuvenation policy, an important research issue is to determine optimal times to trigger rejuvenation.

Reviewing the existing works, rejuvenation policies involve periodic time and aperiodic time triggered policies. Under the periodic policy, rejuvenation is triggered at a fixed time interval, where the time interval is generally determined by solving an optimization function (e.g., cost function). On the other hand, the aperiodic policy plans the rejuvenation based on the prediction of a certain system metric (e.g., resource metric). One example is [13], proposed a rejuvenation policy that schedules rejuvenation according to the predicted time when the critical memory utilization (CMU) reaches a preset threshold.

The methods to plan rejuvenation policy can be categorized into model-based and measurement-based approaches. Generally, the former approach plans the periodic rejuvenation time, and the latter approach plans aperiodic rejuvenation time. The model-based approach usually uses a probabilistic model to describe the aging evolution and rejuvenation process of the system, determining the optimal rejuvenation time in terms of an optimization goal formulated from the model (e.g., minimizing the cost). The measurement-based approach collects system attributes and applies statistical or machine learning methods to forecast the failure time of the system. This type of method determines rejuvenation time directly based on the prediction of the failure time. The model-based approach can be easily generalized across different systems. However, since the analytical model usually makes assumptions about the underlying distribution that used to describe the system, this may incur deviations in the estimated results. Conversely, the measurement-based approach is relatively effective but not easily generalized since the considered system exhibits its peculiar aging behavior (e.g., seasonal pattern, fractal pattern) and thus the specific methods are employed.

In this paper, we propose a two-layer model for the analysis of software rejuvenation considering runtime measurements, the availability function and cost function are respectively established and solved to obtain the optimal rejuvenation time. The lower layer is a failure model constructed based on Cox proportional hazard model. The failure model formulates the failure distribution function, which describes the probability of software failure in the presence of aging. The upper layer is a rejuvenation model that uses a three state semi-Markov process to characterize the rejuvenation process. The rejuvenation model takes the failure distribution from the lower layer as input to formulate the availability function and cost function. Based on the two-layer model, we plan the periodic rejuvenation policy in terms of maximizing availability and minimizing downtime cost. The Cox proportional hazard model is widely used in the analysis of survival data to estimate the effect of multiple risk factors on survival. We extend the classical semi-parametric Cox proportional hazard model to the fully parametric model to formulate the failure distribution of the aging system. Different from the previous analytical model method directly assumes that the failure distribution is subject to a specific distribution, we build the failure distribution according to the runtime measurements. Our approach can be said a combination of the analytical model and measurement method, which can be easily generalized to any type of software system. In the evaluation part, we evaluate the two-layer model on the web search system and compare it with the purely analytical model. In comparison with the purely analytical model, the two-layer model reduces unavailability by 18.18% and reduces downtime cost by 31.22%.

The main contributions of this paper are the following:

- Introduce survival analysis method to study software aging and failure, and extend the semi-parametric Cox model to a full-parametric model to derive the failure distribution of the aging system.
- Propose a two-layer model that uses an analytical model and runtime measurements to plan an optimal rejuvenation policy for the aging system.
- Comprehensively evaluate the proposed approach on the web search system and illustrate its effectiveness.

The remainder of this paper is organized as follows: Section II overviews the relevant work and clarifies the differences in our work. In Section III, a failure model is presented to derive the failure distribution of the aging software system, which is used for constructing the rejuvenation model. Then, in Section IV, we describe the rejuvenation model and address the optimization problem of planning the optimal rejuvenation policy. Section V reports the evaluation results. Finally, in Section VI, we summarize the whole work.

II. RELEVANT WORK

Software aging and rejuvenation are important issues in software reliability research, and a lot of effort has been done to address how to characterize aging behavior and plan optimal rejuvenation policy.

The stochastic process is widely used in modeling software aging and rejuvenation. Huang *et al.* [7] first introduce the Markov approach to study rejuvenation, in which a four-state continuous-time Markov chain (CTMC) is constructed to describe normal, failure-probable, failure and rejuvenation states of the system. This work proves that rejuvenation can reduce downtime and cost. Subsequently, the Markov chain is applied to more complex software systems for rejuvenation analysis. Examples are Wang *et al.* [14] uses Markov chain to analyze the rejuvenation policy of cluster server systems and Ning *et al.* [15] use Markov chain to study the multi-granularity rejuvenation policy of system aging at four granularity levels.

Besides the Markov chain, the semi-Markov process and the Markov regenerative processes (MRGP) are widely used. Bao *et al.* [16] combine the Markov chain and semi-Markov process to study the rejuvenation policy of the system suffers aging due to resource leaks, where Markov chain is used to build the performance degradation model, and the semi-Markov process is employed to construct the rejuvenation model. The degradation model provides failure rate analysis to the rejuvenation model for deriving the availability function and determining the optimal rejuvenation time. Similar work is [9], uses two semi-Markov processes to plan optimal rejuvenation policy for the UNIX system, with one for building the rejuvenation model and with the other for obtaining the failure distribution related parameters required by the rejuvenation model. Bai *et al.* [38] uses a semi-Markov process to construct the analytical model for studying software rejuvenation in virtualized systems. Garg *et al.* [17] use the Markov regenerative stochastic petri net to model the rejuvenation behavior of the client-server type system, unavailability function is formulated based on the model. Then, the optimal rejuvenation interval is determined by minimizing the unavailability function. Zheng *et al.* [18] use Markov regenerative process to study the rejuvenation of the transaction system with a Markovian arrival process. Their model plans a periodic time of rejuvenation policy where the optimal rejuvenation time is determined by minimizing the loss probability and response time. Ning *et al.* [19] construct a Markov rejuvenation process model to plan a two-granularity rejuvenation policy for a software system that suffers aging from OS level and AS level. Recent work of [39] and [40] focus on the rejuvenation of software systems that perform computing tasks, and an event transition-based method is employed for deriving the optimal rejuvenation policy.

In addition to the model-based approaches mentioned above, there is also a considerable part of the work determines aperiodic rejuvenation time based on the measurement approach. In the early studies [20] and [21], they found that software aging is a chronic process manifests as gradual performance degradation and resource consumption. The measurement-based approach statistically analyzes performance data or resource data in order to detect the aging trend, forecast the resource exhaustion time, and plan rejuvenation. Time series analysis, machine learning method, and threshold method are widely used for measurement analysis. Zheng *et al.* [22] apply a modified Mann-Kendall & Sen's slope estimator on the time series data (response time and throughput) to detect the aging trend and forecast the time to failure of the web server system. Alonso *et al.* [23] observed that resource data exhibits nonlinear characteristic in the web J2EE application when it suffers aging, in order to capture the nonlinear behavior, a machine learning method (regression tree) is applied for forecasting the resource exhaustion time. Silva *et al.* [24] proposed a new rejuvenation approach based on self-healing techniques, which utilize virtualization to optimize self-recovery actions. The rejuvenation approach defines the threshold on the performance metric of the system (e.g., response time), and rejuvenation is triggered when the detected performance metric exceeds the threshold. Chen *et al.* [25] proposed an Entropy-based aging indicator to predict aging related failures by defining failure threshold on the aging indicator. Similar work is [26], the authors proposed a probabilistic aging indicator and they define the threshold on this aging indicator to predict the failures caused by aging.

This paper proposes a two-layer model that takes into account runtime measurements to model the rejuvenation process of the aging software system and plans rejuvenation policy in terms of maximizing the system availability and minimizing downtime cost. The two-layer model involves a failure model and a rejuvenation model, the former model is used as the input of the latter model to derive explicit optimization functions. We introduce a survival analysis method to study failure events of software aging and formulate a failure distribution using runtime measurements. Unlike previous analytical models that empirically assumes the failure distribution, our model estimates failure distribution by using multiple runtime measurements. Section V evaluates that our model is superior to the empirical model. The rejuvenation model is described as a three-state Markov process, including the available state, the failure state, and the rejuvenation state. The rejuvenation model takes the constructed failure distribution as the state transition distribution from the available state to the failure state to derive an availability function and a cost function. Then, the optimal rejuvenation time is determined by solving the optimization functions.

III. THE FAILURE MODEL

In this section, we develop a failure model to study the failure behavior of the aging software system. A failure distribution is formulated from the model, which describes the likelihood of system failure due to resource leak. The failure distribution will be used to construct the rejuvenation model in Section IV.

A. MODEL DESCRIPTION

Considering a runtime software system, intuitively the probability of system failure is affected by two factors:

1) On the one hand, it is affected by the time factor, since aging exists in the long-running software system, and it tends to degrade performance over time.

2) On the other hand, it is affected by system resource factors, especially Memory-related resources, since the fewer available resources, the more severe the aging and the greater the possibility of the system to fail.

For aging software systems, when the failure occurs is a random event. We use a probabilistic model to describe the likelihood of system failure. To account for both the time effect and resource factors in the system's failure rate, the Cox proportional hazard model (abbreviated to Cox model) is employed. In Cox model, the hazard function of a system consists of two multiplicative parts, the baseline hazard function and an exponential function including the effects of the monitoring variables. The hazard function is expressed as:

$$h\,(t|\mathbf{z}_t) = h_0(t)exp(\boldsymbol{\gamma}\mathbf{z}_t) \tag{1}$$

where $h_0(t)$ is the baseline hazard function related to the running time. $\mathbf{z}_t$ is a row vector, named as covariates that represent the monitoring variables at time t. $\boldsymbol{\gamma}$ is a column vector denotes the regression parameters corresponding to the monitoring variables. Cox model can be interpreted as: the hazard rate of the system at time t is affected by both the baseline hazard rate and the monitoring variables. $h_0(t)$ defines the hazard rate when the monitoring variables do not exist, while $\mathbf{z}_t$ defines the effect of monitoring variables on the hazard rate, making the hazard rate increases or decreases on the basis of the baseline hazard rate, and the coefficient $\boldsymbol{\gamma}$ controls the effect of each monitoring variables on the hazard rate. Cox model is a type of semi-parametric model since $h_0(t)$ has no concrete definition and its parameters cannot be estimated. To model the failure behavior of the software system, a concrete expression of the failure distribution is required. We extend the Cox model as a fully parametric model in order to derive the failure distribution. We assume that the $h_0(t)$ follows the Weibull distribution, and then the Cox model is extended as the Weibull Cox model. The hazard function is written as:

$$h_w\,(t|\mathbf{z}_t) = \frac{\beta}{\eta}(\frac{t}{\eta})^{\beta-1} exp(\boldsymbol{\gamma}\mathbf{z}_t) \tag{2}$$

where $\beta > 0$ and $\eta > 0$ are the shape and scale parameters of the Weibull distribution, respectively. Although we made assumptions about the baseline hazard function, since our model considers runtime measurements, the hazard rate estimates can be adjusted through the runtime measurements. The reason we choose Weibull distribution is due to its versatility of modeling a variety of life behaviors, which is widely used in reliability and life data analysis [27], also used in modeling failure distribution of software systems [28], [29].

Equation (2) estimates the instantaneous failure rate. To estimate the failure probability, we derive the probability density function (p.d.f.) and the cumulative distribution function (c.d.f.). Let T be a non-negative random variable denotes the time to failure. The distribution of T can be characterized by its p.d.f. and c.d.f. The c.d.f. of T is giving by:

$$\begin{aligned} F\,(t|\mathbf{z}_t) &= Pr\,(T \geq t, \mathbf{z}_t) \\ &= \int_0^t f\,(t\,|\,\mathbf{z}_t)\,dt \end{aligned} \tag{3}$$

represents the probability that failure has occurred by time t. In survival analysis, the survival function is given by:

$$\begin{aligned} S\,(t|\mathbf{z}_t) &= Pr(T < t, \mathbf{z}_t) \\ &= 1 - F(t|\mathbf{z}_t) \end{aligned}$$

represents the probability that a subject can survive beyond time t, in other words, the failure has not occurred by duration time t. According to the principle of survival analysis [30], the survival function is estimated as:

$$S\,(t|\mathbf{z}_t) = exp(-\int_0^t h_w\,(t|\mathbf{z}_t)\,dt)$$

Hence, the c.d.f. is derived as:

$$\begin{aligned} \mathrm{F}\,(t|\mathbf{z}_t) &= 1 - \exp(-\int_0^t \mathrm{h}_w\,(t|\mathbf{z}_t)\,dt) \\ &= 1 - exp\left(-\left(\frac{t}{\eta}\right)^{\beta} exp(\boldsymbol{\gamma}\mathbf{z}_t)\right) \end{aligned} \tag{4}$$

Equation (4) will be used as input to the rejuvenation model to derive the availability and downtime cost functions, the details will be described in Section IV.

B. MODEL SOLUTION

In (4), parameters η, β, and $\boldsymbol{\gamma}$ need to be estimated. Given a known form model and a set of observation data, the unknown parameters of the model can be estimated by using the maximum likelihood estimation (MLE) method. MLE is a statistical method that estimates the parameters of a probability distribution by maximizing a likelihood function, so that under the considered model the observed data is most probable. In MLE method, the likelihood function is defined as follows:

$$l\,(\boldsymbol{\theta}) = f\,(\mathbf{y};\boldsymbol{\theta}) = \prod\nolimits_{i=1}^{m} f(y_j;\boldsymbol{\theta})$$

where $\theta = [\theta_1,\theta_2,\ldots,\theta_k]^T$ is the unknown parameters of a model, $f(\mathbf{y};\boldsymbol{\theta})$ is the joint probability density of random variables $\{y_1, y_2, \ldots\}$, $\mathbf{y} = \{y_1, y_2, \ldots, y_m\}$ is a given set of observation data. For independent and identically distributed random variables, $f(\mathbf{y};\boldsymbol{\theta})$ will be the product of univariate density functions. The likelihood function can be interpreted as: given a model and a set of observation results, MLE is to make inferences about the model parameters that is most likely to have generated the observation results.

According to (4), the p.d.f. of the failure model can be derived as:

$$f(t, z_t) = \frac{dF(t, z_t)}{dt} = \frac{\beta}{\eta}\left(\frac{t}{\eta}\right)^{\beta-1} exp(\gamma z_t)\, exp\left(-\left(\frac{t}{\eta}\right)^{\beta} exp(\gamma z_t)\right)$$

We assume a set of observation data $\boldsymbol{D} = \{t_1|\mathbf{z_1}, t_2|\mathbf{z_2}, t_3|\mathbf{z_3}, \ldots, t_m|\mathbf{z_m}\}$. For each observation i, where t_i is the failure time and $\mathbf{z_i}$ is the monitored variables at time t_i. We can then define the likelihood function as:

$$\begin{aligned} l(\eta, \beta, \boldsymbol{\gamma}) &= f(\boldsymbol{D}; \eta, \beta, \boldsymbol{\gamma}) \\ &= \prod_{i=1}^{m} f(t_i|\mathbf{z_i}; \eta, \beta, \boldsymbol{\gamma}) \\ &= \prod_{i=1}^{m} \frac{\beta}{\eta}\left(\frac{t_i}{\eta}\right)^{\beta-1} exp(\boldsymbol{\gamma}\mathbf{z}_i)\, exp\left(-\left(\frac{t_i}{\eta}\right)^{\beta} \times exp(\boldsymbol{\gamma}\mathbf{z}_i)\right) \end{aligned}$$

To facilitate the solution, the log-likelihood function is used, which is expressed as:

$$\begin{aligned} ln[l(\eta, \beta, \boldsymbol{\gamma})] &= ln\prod_{i=1}^{m} f(t_i|\mathbf{z}_i; \eta, \beta, \boldsymbol{\gamma}) \\ &= m\, ln\left(\frac{\beta}{\eta}\right) + \sum_{i=1}^{m} ln\left(\frac{t_i}{\eta}\right)^{\beta-1} + \sum_{i=1}^{m} \boldsymbol{\gamma}\mathbf{z}_i - \sum_{i=1}^{m}\left(\frac{t_i}{\eta}\right)^{\beta} exp(\boldsymbol{\gamma}\mathbf{z}_i) \end{aligned}$$

The log-likelihood function is used as the objective function of maximization. Differentiating the objective function with respect to the parameters $\eta, \beta, \boldsymbol{\gamma}$, we get the equations as follows:

$$\frac{\partial ln[l(\eta, \beta, \gamma)]}{\partial \eta} = 0$$
$$\frac{\partial ln[l(\eta, \beta, \gamma)]}{\partial \beta} = 0$$
$$\frac{\partial ln[l(\eta, \beta, \gamma)]}{\partial \gamma} = 0$$

The optimal estimation of $\hat{\eta}, \hat{\beta}, \hat{\boldsymbol{\gamma}}$ can be obtained by solving the above equations. In this paper, we employ the Newton-Raphson method to solve the above nonlinear equations.

IV. THE REJUVENATION MODEL AND OPTIMAL REJUVENATION POLICY

In this section, we develop a proactive rejuvenation model to plan the optimal rejuvenation policy. The optimization functions about availability and downtime cost functions are formulated and solved for determining the optimal rejuvenation time.

A. MODEL DESCRIPTION

We use a 3-state semi-Markov process to describe the rejuvenation process of the software system in the presence of aging. Figure 1 depicts the evolution of the three states, where

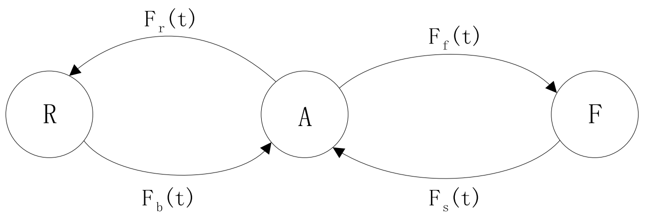


FIGURE 1. **The rejuvenation model that is described with a 3-state semi-Markov process.**

A represents the available state, R represents the rejuvenation state and F represents the failure state. The initial start-up system is in state A. The system runs continuously over time and gradually suffers from performance degradation due to aging. If the system fails, the system transitions from state A to state F, while if the rejuvenation operation is performed, the system transitions from state A to state R. In state F, after reactive recover is completed, the system is restored and transition to state A. In state R, after the rejuvenation operation is completed, the system returns to the initial state of A. The system is not available in state R and state F. In Figure 1, $F_r(t)$, $F_f(t)$, $F_b(t)$, $F_s(t)$ are respectively denote the distribution function of time to rejuvenation, time to failure, duration of the rejuvenation operation, duration of the recovery operation. The correlation between the failure model and the rejuvenation model is the failure state F. In the failure model, we have formulated the distribution function of the time to failure. Hence, $F_f(t) = \mathrm{F}(t|\mathbf{z}_t)$. The forms of $F_b(t)$ and $F_s(t)$ could be identified and estimated using standard statistical methods. Some work gives suggestions in the form of the two functions, such as [19] suggests Weibull distribution and Pareto distribution for $F_b(t)$ and $F_s(t)$ respectively. Generally, duration of the rejuvenation operation and duration of the recovery operation are specified according to software maintenance guidelines [12], [19].

We plan a time-based rejuvenation policy, that is, perform the rejuvenation after the system runs a period τ from its startup, i.e., $F_r(t) = \tau$ for t $>$ 0. The optimal rejuvenation time interval τ is determined by maximizing the system availability and minimizing the downtime cost.

B. AVAILABILITY AND DOWNTIME COST

We use a transition matrix to describe the transitions of the rejuvenation model, which is expressed as:

$$\boldsymbol{P} = \begin{bmatrix} 0 & P_{AR} & P_{AF} \\ P_{RA} & 0 & 0 \\ P_{FA} & 0 & 0 \end{bmatrix}$$

where P_{AR} represents the transition probability from state A to state R, P_{AF} represents the transition probability from state A to state F. P_{RA} and P_{FA} have a similar meaning. The transition probability can be estimated by the distribution function, thus we have:

$$p_{AF} = F_f(\tau)$$
$$p_{AR} = 1 - p_{AF} = 1 - F_f(\tau)$$

Let $\boldsymbol{\pi} = [\pi_A, \pi_R, \pi_F]$ is the steady state probability. According to the following properties:

$$\begin{aligned} &P_{AR} + P_{AF} = 1 \\ &P_{RA} = 1 \\ &P_{FA} = 1 \\ &\boldsymbol{\pi} = \boldsymbol{\pi} \boldsymbol{P} \\ &\pi_A + \pi_R + \pi_F = 1 \end{aligned}$$

Then, we have:

$$\pi_A = \frac{1}{2} \quad \pi_R = \frac{P_{AR}}{2} \quad \pi_F = \frac{P_{AF}}{2}$$

Let $d_A d_R$ and d_F respectively denote the sojourn time of the system in state A, R and F, where d_R and d_F are also representing the time required to complete the rejuvenation operation and the time required to complete the recovery after a failure. In practice, failure occurs unprepared may lead to great losses. After a failure occurs, a reactive recovery is triggered that also needs some tricky operations, such as data recovery. Hence, the time and cost of reactive recovery is much greater than proactive rejuvenation [17]. d_R and d_F are generally considered as the mean of distribution functions $F_b(t)$ and $F_s(t)$ in Figure 1. The sojourn time of state A is given by [31]:

$$d_A = \int_0^{\tau} (1 - F_f(t))dt$$

It can be interpreted as: the system did not fail during the period before the rejuvenation was performed.

As shown in Figure 1, the system is only available in state A. we derive the steady state availability as follows:

$$\begin{aligned} A(\tau) &= \frac{\pi_A d_A}{\pi_A d_A + \pi_R d_R + \pi_F d_F} \\ &= \frac{d_A}{d_A + P_{AR} d_R + P_{AF} d_F} \\ &= \frac{\int_0^{\tau} (1 - F_f(t))dt}{\int_0^{\tau} (1 - F_f(t))dt + (1 - F_f(\tau)) d_R + F_f(\tau) d_F} \end{aligned} \tag{5}$$

By the failure distribution $F_f(t) = \mathrm{F}(t|\mathbf{z}_t)$, and according to (4), then we have:

$$F_f(t) = 1 - exp\left(-\left(\frac{t}{\eta}\right)^{\beta} exp(\boldsymbol{\gamma}\mathbf{z}_t)\right) \tag{6}$$

In (5), substitute for $F_f(t)$ by (6), we obtain the concrete expression of the availability function, which is a nonlinear function about the rejuvenation time τ. Function (5) is used as the objective function of maximization. We differentiate (5) with respect to variable τ and set the derivative function equal to zero, then we employ the bisection method to find the root of the derivative equation, and this value is the optimal rejuvenation time τ^*. Later, section V will illustrate that the availability function is a concave function (Fig.5), so there must be a value of τ^* that maximizes the availability.

Availability and downtime cost are two different concerns. To determine the optimal rejuvenation time can also be considered from the perspective of minimizing downtime cost. Let c_F and c_R denote the cost of failure and the cost of rejuvenation per unit time respectively, then the downtime cost per unit time can be expressed as:

$$\begin{aligned} C(\tau) &= \frac{\pi_R d_R C_R + \pi_F d_F C_F}{\pi_A d_A + \pi_R d_R + \pi_F d_F} \\ &= \frac{P_{AR} d_R c_R + P_{AF} d_F c_F}{d_A + P_{AR} d_R + P_{AF} d_F} \\ &= \frac{\left(1 - F_f(\tau)\right) d_R c_R + F_f(\tau) d_F c_F}{\int_0^{\tau} (1 - F_f(t))dt + \left(1 - F_f(\tau)\right) d_R + F_f(\tau) d_F} \end{aligned} \tag{7}$$

In (7), substitute for $F_f(t)$ by (6), we obtain the concrete expression of the downtime cost function, which is a nonlinear function of the rejuvenation time τ. Function (7) is used as the objective function of minimization. Similar to before, we differentiate (7) with respect to τ and set the derivative function equal to zero, then we employ the bisection method to find the root of the derivative equation, and this value is the optimal rejuvenation time τ^*. In section V, we illustrate that the downtime cost function is a convex function (Figure 6), so there must be a value of τ^* that minimizes the downtime cost.

V. CASE STUDY

In this section, we comprehensively evaluate the failure model and the rejuvenation model by studying the aging of the web search system. The experimental results show that the proposed model can be used as a reliable method to plan the optimal rejuvenation time.

When experimenting to study the software aging, long periods of runtime are required to observe system failures. To reduce the observation time, the Accelerated life test (ALT) approach proposed in [32] and [33] is adopted to accelerate the software aging. The mechanism of the ALT method is to activate aging-related bugs so that they result in software aging. For example, activation of Memory-related bugs can result in a Memory leak. Memory leak has been demonstrated to be one of the main causes of software aging and it occurs frequently in running software systems [34]. Several papers have studied the effects of Memory leaks on software aging [23], [35]. In this paper, we employ the inject Memory leak method to accelerate software aging.

Software failure not only refers to system crash, but also includes that the software system is still running but cannot provide normal functions or services, for example, violation of the Service level agreement (SLA). The latter kind of failure is common in practice. For the web search system, its normal service requires that the response time no more than 300 ms [36], [37]. Hence, we define that the web search system fails when its response time exceeds 300 ms.

TABLE 1. Configurations of the server machine and client machine.

Configuration	Server machine	Client machine
CPU	Intel(R) Xeon(R) CPU E3-1231 v3 @ 3.40GHz	Intel(R) Core(TM) i5-4460 CPU @ 3.20GHz
RAM	7.8Gbyt	7.7Gbyt
NIC	100Mbps	100Mbps
OS	Linux server 4.2.0-27-generic #32~14.04.1-Ubuntu	Linux osv-05 3.16.0-30-generic #40~14.04.1-Ubuntu

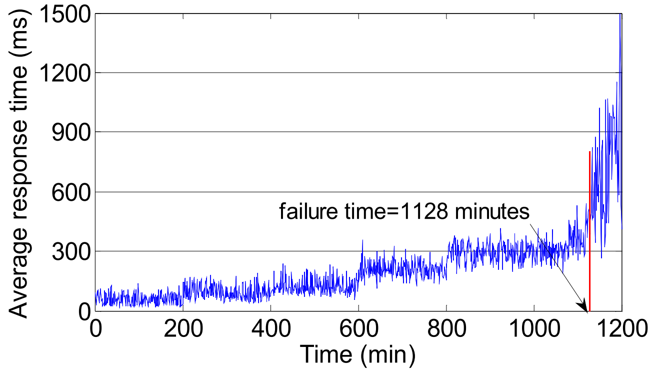


FIGURE 2. The monitored performance data of web search during 1200 minutes of continuous operation.

A. EXPERIMENT SETUP

The experimental platform composed of two physical machines, one as a server and the other as a client, and the two physical machines are connected by a switch. Table 1 lists the configurations of the two machines. We deploy the Cloudsuite's Web search to the platform, where the server machine contains an index of the text and fields found in a set of crawled websites and, the client machine sends random keyword query requests to the server end. The Cloudsuite's web search relies on the Apache Solr search engine framework, and the client machine uses the Faban load generator to simulate real-world clients that send requests to the server node. The server machine is the test system, so we study the aging of the server end during the system operation.

In order to accelerate software aging, we inject Memory leaks on the server machine by using Lookbusy (version: 1.4). Lookbusy is a simple load generator that can occupy a certain amount of Memory according to the user's need so that the occupied Memory is not available for other software systems.

During the web search operation, we leverage the Vmstate tool (version: procps-ng 3.3.9) to collect the resource metrics and parse system logs to collect performance metric. As an example, Figure 2 and Figure 3 depict the collected performance data and resource data of web search during 1200 minutes of continuous operation, in which Memory leaks gradually increases from 500 MB to 1500 MB. As the software system works in a highly dynamic environment, the collected data has great fluctuations. In the time series data, we identify the failure point as the first

TABLE 2. Resource metrics used for studying the aging of the web server system.

Metric	Description
Swpd	The amount of virtual Memory used
Free	The amount of idle Memory
Buffer	The amount of Memory used as buffers
Cache	The amount of Memory used as cache

TABLE 3. Failure data and its corresponding monitoring variables.

ID	Failure time (minutes)	Swpd (MB)	Avail-Mem (MB)
1	1106	51928	4702848
2	1128	43816	4495744
3	1106	51928	4702848
4	805	44320	3925684
5	1081	182188	3629836
6	1106	51928	4702848
7	620	289664	3203504
8	801	1121768	2996772
9	721	1159612	2768960
10	601	1694548	2588328

point of 10 consecutive points with response time exceeding 300ms. Failure time is the time from the beginning of system operation to system failure. As shown in Figure 2, the failure occurs after the system runs for 1128 min. That is to say, from 1128 min, at least 9 consecutive detection points have a response time of more than 300ms. Figure 3 shows the resource data collected by Vmstate. 4 different types of resource metrics are collected, all of which are Memory-related metrics. Table 2 explains the meaning of resource metrics.

According to (4), the failure distribution is estimated in terms of the failure time data and its corresponding monitoring variables. In this case study, we choose Swpd and available Memory (short for avail-Mem) as candidates for monitoring variables, where the avail-Mem is the sum of Free, Buffer, and Cache. In the next section, we will analyze the candidate variables and model the optimal failure model.

Table 3 lists the 10 sets of failure data and its corresponding monitoring variables obtained by the experiment. It is worth noting that since fluctuations in the monitoring variables (As shown in Figure 3), we take the average value of the monitoring variables of five points before and after the failure point as the monitoring variables. For example, when the failure time is 1128 min, values of Swpd and avail-Mem respectively are the average of 10 monitored values from 1124 to 1133 min. For the two monitoring variables, the more consumption of Swpd, the greater the probability of system failure, but the less the avail-Mem, the more likelihood of system failure. The Swpd and avail-Mem are denoted as z_1 and z_2, respectively. Since the monitored data value is large, it will increase the computational complexity of the model. We use the normalization method to process the data, it is executed as follows:

$$z_i = \frac{z_i - z_{imin}}{z_{imax} - z_{imin}}, \quad i = 1, 2$$

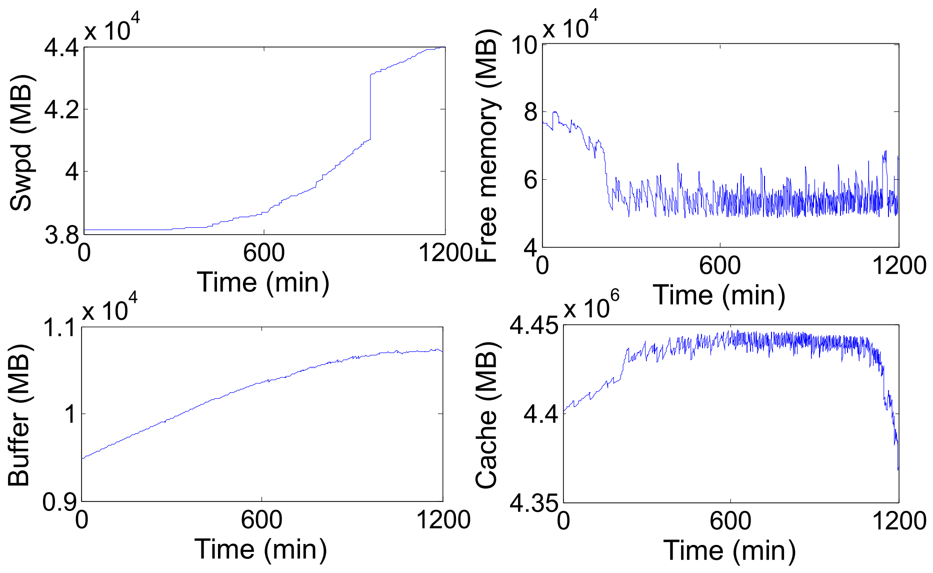


FIGURE 3. **The monitored resource data of web search during 1200 minutes of continuous operation.**

where $z_{1max} = 1815808$ and $z_{1min} = 38116$, $z_{2max} = 4736088$ and $z_{2min} = 2450316$ are respectively the maximum and minimum values for z_1 and z_2. When training the Cox model to estimate the model parameters, the number of samples needed is generally 5-10 times of model variables. In our model, two parameters are considered, so 10 sets of data are sufficient.

B. EVALUATION OF THE FAILURE MODEL

To analyze whether Swpd and avail-Mem are useful to construct the failure model, we use F-test to verify the significance of the two variables on the failure model. According to the F-test's theory, we propose the following hypothesis:

H_0: The failure model without considering monitoring variables, i.e. the hazard function of the failure model is simplified to the baseline hazard function, which is expressed as:

$$h_w(t) = \frac{\beta}{\eta}\left(\frac{t}{\eta}\right)^{\beta-1}$$

H_1: The failure model considering monitoring variables z_1 and z_2, i.e., the hazard function of the failure model is expressed as:

$$h_w(t|\mathbf{z_t}) = \frac{\beta}{\eta}\left(\frac{t}{\eta}\right)^{\beta-1} \exp(\gamma_1 z_1 + \gamma_2 z_2)$$

H_0 is the null hypothesis, H_1 is the alternative hypothesis. At the significance level of $\alpha = 0.05$, the computed p-value is 0.017, since p-value $< \alpha$, it indicates less than 5% probability the null hypothesis is correct. Therefore, we reject the null hypothesis and accept the alternative hypothesis. That is to say, the monitoring variables z_1 and z_2 have a significant effect on the failure model.

The F-test verifies that the monitoring variables have a significant effect on the failure, but it does not mean that both z_1 and z_2 have significant contributions to the failure model. To analyze the effect of each monitoring variable on the failure model, we use the Akaike information criterion (AIC) to estimate which variable can build the optimal failure model. AIC deals with the trade-off between the goodness of fit of the model and the simplicity of the model, which can be estimated by:

$$AIC = -2lnL + 2K$$

where L is the maximum value of the likelihood function for the model, K is the number of the monitoring variables in the model. Given a set of candidate models, the preferred model is the one with the minimum AIC value. For the two candidate monitoring variables, there will be three candidate models, as shown in Table 4. The first two columns list the candidate models and their descriptions, and the last three columns list the $\ln L$, K, and AIC values of each candidate model. The results show that failure model-2 with the minimum AIC value. Therefore, failure model-2 is the optimal failure model that will be used to construct the rejuvenation model. The failure model-2 has the following failure distribution:

$$F(t|z_2) = 1 - \exp\left(-\left(\frac{t}{\eta}\right)^{\beta} \exp(\gamma_2 z_2)\right) \quad (8)$$

We estimate the parameters in failure model-2 using the MLE method described in Section III.B, and results are list in Table 5. It can be seen from the table that the coefficient γ_2 is a negative value, indicating that the avail-Mem has a negative and significant effect on the failure, that is, the less available Memory, the more likelihood of failure.

When not considering runtime measurements (monitoring variables), the failure model-2 is simplified as the baseline model. We compare the failure model-2 with the baseline model in terms of the goodness of fit. For a probabilistic

TABLE 4. Selection of the optimal failure model based on AIC.

Candidate model	Description	*lnL*	*K*	*AIC*
failure model-1	failure model with monitoring variable z_1	-61.64	1	125.28
failure model-2	failure model with monitoring variable z_2	-61.13	1	124.26
failure model-3	failure model with monitoring variables $z_1$and z_2	-60.47	2	124.95

TABLE 5. The parameters of failure model-2 estimated by the MLE method.

Parameter	β	η	γ_2
Estimated value	8.40	689.45	-4.46

TABLE 6. Comparison of the goodness of fit between failure model-2 and baseline model.

Failure model	Log-likelihood
failure model-2	-61.13
baseline model	-67.24

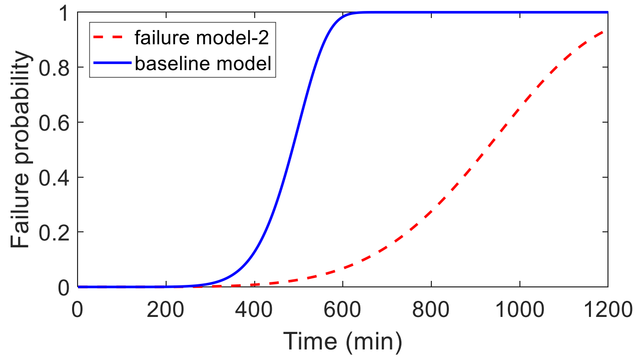


FIGURE 4. Comparison of failure probability between failure model-2 and baseline model, where failure model-2 is the failure model with monitoring variable avail-Mem and baseline model is the failure model without monitoring variable.

model, we use log-likelihood to measure the goodness of fit. Since the log-likelihood is what we are maximizing in the model training, so the larger the log-likelihood value, the better the model fitting effect. Table 6 lists the comparison results of the two models. We can see that failure model-2 is superior to the baseline model. Figure 4 illustrates the failure distributions of the two models, where the solid line denotes the c.d.f. of the failure model-2 and the dotted line denotes the c.d.f. of the baseline model. It can be seen from the figure that failure model-2 has a higher probability of failure than the baseline model, indicates that the system with Memory leak is more likely to fail than the system without Memory leak.

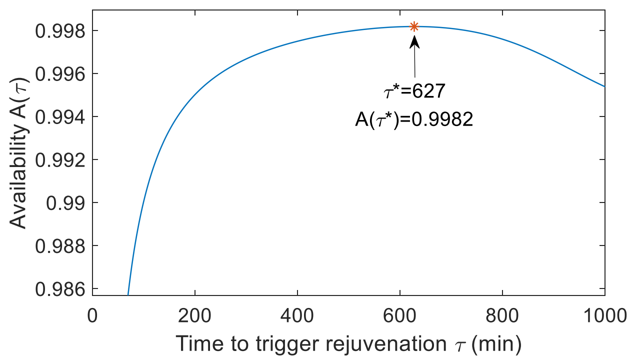


FIGURE 5. Availability versus rejuvenation trigger time.

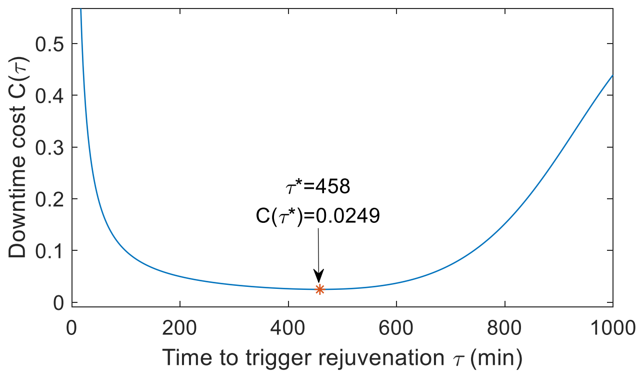


FIGURE 6. Downtime cost versus rejuvenation trigger time.

C. EVALUATION OF THE REJUVENATION MODEL

According to Section IV.B, we derive the optimal rejuvenation time by maximizing the availability function and minimizing the downtime cost function. Specifically, the failure model $F_f(t)$ is replaced by the optimal failure model-2. The form of the failure model-2 is shown in (8), and the model parameters are list in Table 5. In (5) and (7), substitute for $F_f(t)$ by (8), we can estimate the optimal rejuvenation time.

We assume that the mean time to recover from a failure is 5 min, i.e., $d_F = 5$ min, and the mean time required for the rejuvenation operation is 1 min, i.e., $d_R = 1$ min. We also assume that the cost of failure is \$100 per min, i.e., $c_F = \$100/\text{min}$, and the cost of rejuvenation is \$10 per min, i.e., $c_R = \$10/\text{min}$. Figure 5 shows the rejuvenation trigger time versus the availability of the system, where the maximum availability found is $A(\tau^*) = 0.9982$ and the corresponding optimal rejuvenation time interval is $\tau^* = 627$ min. Figure 6 plots the rejuvenation trigger time versus the downtime cost of the system, where the minimum cost found is $C(\tau^*) = 0.0249$ and the corresponding optimal rejuvenation time interval is $\tau^* = 458$ min.

D. COMPARISON

Compared with the traditional analytical model, the effectiveness of the two-layer model proposed in this paper is reflected in the integration of runtime measurements to correct the

TABLE 7. Comparison of rejuvenation optimization between the two-layer model with and without monitoring variables.

Model	Availability		Downtime cost	
	τ^*	$A(\tau^*)$	τ^*	$C(\tau^*)$
Two-layer model with monitoring variable	627 min	0.9982	458 min	0.0249
Two-layer model without monitoring variable	569 min	0.9978	344 min	0.0362

analytical model. We compare the two cases with and without system measurement, i.e., the two-layer model with monitoring variables and the two-layer model without monitoring variable. Table 7 lists the comparison results. It can be seen that compared to the model without considering the monitoring variables, our model reduces the unavailability by:

$$\frac{(1-0.9978)-(1-0.9982)}{1-0.9978} \times 100\% = 18.18\%$$

similarly, our model reduces the downtime cost by:

$$\frac{0.0362-0.0249}{0.0362} \times 100\% = 31.22\%$$

The comparison results indicate that runtime measurement can improve the effectiveness of the two-layer model.

VI. CONCLUSION

Long-running software systems inevitably suffer aging, leading to gradual performance degradation and eventually service failure. Unplanned software failures negatively affect system reliability. Software rejuvenation is an effective fault management technique that is widely adopted to counteract the aging effect and prevent failure. The benefit and effectiveness of software rejuvenation can be greatly affected by the rejuvenation policy. A frequent rejuvenation will affect the system availability and incurs extra downtime costs. The research on determining optimal rejuvenation time is an important issue. In this paper, we derive the optimal time-based rejuvenation policy maximizing the system availability and minimizing the downtime cost. A two-layer model is proposed that includes a failure model and a rejuvenation model works together to derive the optimal rejuvenation time. The failure model is constructed based on the Cox proportional hazard model, using runtime measurements to derive a failure distribution of the system. The rejuvenation model is built based on the semi-Markov process, which takes the failure distribution from the failure model as input to derive an availability function and a downtime cost function. Then, optimal rejuvenation time is derived in terms of the two functions. The case study conducted on the web server system have shown that our model can be used as a reliable method for planning the optimal rejuvenation policy.

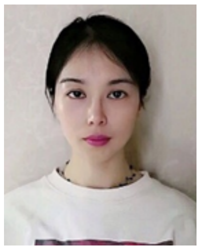
ACKNOWLEDGMENT

The authors are very grateful to anonymous reviewers for their constructive comments.

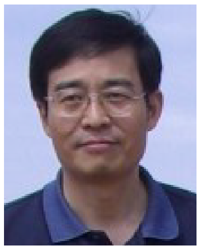
REFERENCES

[1] D. L. Parnas, "Software aging," in *Proc. 16th Int. Conf. Softw. Eng. (ICSE)*, 1994, pp. 279–287.
[2] S. Jia, C. Hou, and J. Wang, "Software aging analysis and prediction in a Web server based on multiple linear regression algorithm," in *Proc. IEEE 9th Int. Conf. Commun. Softw. Netw. (ICCSN)*, May 2017, pp. 1452–1456.
[3] D. Cotroneo, R. Natella, R. Pietrantuono, and S. Russo, "Software aging analysis of the linux operating system," in *Proc. IEEE 21st Int. Symp. Softw. Rel. Eng. (ISSRE)*, Nov. 2010, pp. 71–80.
[4] L. Cui, B. Li, J. Li, J. Hardy, and L. Liu, "Software aging in virtualized environments: Detection and prediction," in *Proc. IEEE 18th Int. Conf. Parallel Distrib. Syst. (ICPADS)*, Dec. 2012, pp. 718–719.
[5] J. Araujo, R. Matos, P. Maciel, and R. Matias, "Software aging issues on the eucalyptus cloud computing infrastructure," in *Proc. IEEE Int. Conf. Syst., Man, Cybern. (ICSMC)*, Oct. 2011, pp. 1411–1416.
[6] D. Cotroneo, F. Fucci, A. K. Iannillo, R. Natella, and R. Pietrantuono, "Software aging analysis of the Android mobile OS," in *Proc. IEEE 27th Int. Symp. Softw. Rel. Eng. (ISSRE)*, Oct. 2016, pp. 478–489.
[7] Y. Huang, C. Kintala, N. Kolettis, and N. D. Fulton, "Software rejuvenation: Analysis, module and applications," in *Proc. 25th Int. Symp. Fault-Tolerant Comput. (FTCS)*, 1995, pp. 381–390.
[8] M. Grottke, R. Matias, and K. S. Trivedi, "The fundamentals of software aging," in *Proc. IEEE Int. Conf. Softw. Rel. Eng. Workshops (ISSRE Wksp)*, Nov. 2008, pp. 1–6.
[9] K. Vaidyanathan and K. S. Trivedi, "A comprehensive model for software rejuvenation," *IEEE Trans. Dependable Secure Comput.*, vol. 2, no. 2, pp. 124–137, Feb. 2005.
[10] K. S. Trivedi and K. Vaidyanathan, "Software aging and rejuvenation," in *Wiley Encyclopedia of Computer Science and Engineering*. Chichester, U.K.: Wiley, Dec. 2007.
[11] D. Bruneo, S. Distefano, F. Longo, A. Puliafito, and M. Scarpa, "Workload-based software rejuvenation in cloud systems," *IEEE Trans. Comput.*, vol. 62, no. 6, pp. 1072–1085, Jun. 2013.
[12] J. Alonso, R. Matias, E. Vicente, A. Maria, and K. S. Trivedi, "A comparative experimental study of software rejuvenation overhead," *Perform. Eval.*, vol. 70, no. 3, pp. 231–250, Mar. 2013.
[13] J. Araujo, R. Matos, P. Maciel, F. Vieira, R. Matias, and K. S. Trivedi, "Software rejuvenation in eucalyptus cloud computing infrastructure: A method based on time series forecasting and multiple thresholds," in *Proc. IEEE 3rd Int. Workshop Softw. Aging Rejuvenation (WoSAR)*, Nov. 2011, pp. 38–43.
[14] D. Wang, W. Xie, and K. S. Trivedi, "Performability analysis of clustered systems with rejuvenation under varying workload," *Perform. Eval.*, vol. 64, no. 3, pp. 247–265, Mar. 2007.
[15] G. Ning, K. S. Trivedi, H. Hu, and K.-Y. Cai, "Multi-granularity software rejuvenation policy based on continuous time Markov chain," in *Proc. IEEE 3rd Int. Workshop Softw. Aging Rejuvenation (WoSAR)*, Nov. 2011, pp. 32–37.
[16] Y. Bao, X. Sun, and K. S. Trivedi, "A workload-based analysis of software aging, and rejuvenation," *IEEE Trans. Rel.*, vol. 54, no. 3, pp. 541–548, Sep. 2005.
[17] S. Garg, A. Puliafito, M. Telek, and K. S. Trivedi, "Analysis of software rejuvenation using Markov regenerative stochastic Petri net," in *Proc. 6th Int. Symp. Softw. Rel. Eng. (ISSRE)*, 1995, pp. 180–187.
[18] J. Zheng, H. Okamura, L. Li, and T. Dohi, "A comprehensive evaluation of software rejuvenation policies for transaction systems with Markovian arrivals," *IEEE Trans. Rel.*, vol. 66, no. 4, pp. 1157–1177, Dec. 2017.
[19] G. Ning, J. Zhao, Y. Lou, J. Alonso, R. Matias, K. S. Trivedi, B.-B. Yin, and K.-Y. Cai, "Optimization of two-granularity software rejuvenation policy based on the Markov regenerative process," *IEEE Trans. Rel.*, vol. 65, no. 4, pp. 1630–1646, Dec. 2016.
[20] L. Bernstein, "Innovative technologies for preventing network outages," *AT&T Tech. J.*, vol. 72, no. 4, pp. 4–10, Jul. 1993.
[21] S. Garg, A. van Moorsel, K. Vaidyanathan, and K. S. Trivedi, "A methodology for detection and estimation of software aging," in *Proc. 9th Int. Symp. Softw. Rel. Eng. (ISSRE)*, 1998, pp. 283–292.
[22] P. Zheng, Y. Qi, Y. Zhou, P. Chen, J. Zhan, and M. R. Lyu, "An automatic framework for detecting and characterizing performance degradation of software systems," *IEEE Trans. Rel.*, vol. 63, no. 4, pp. 927–943, Dec. 2014.

[23] J. Alonso, J. Torres, J. L. Berral, and R. Gavalda, "Adaptive on-line software aging prediction based on machine learning," in *Proc. IEEE/IFIP Int. Conf. Dependable Syst. Netw. (DSN)*, Jun. 2010, pp. 507–516.
[24] L. M. Silva, J. Alonso, and J. Torres, "Using virtualization to improve software rejuvenation," *IEEE Trans. Comput.*, vol. 58, no. 11, pp. 1525–1538, Nov. 2009.
[25] P. Chen, Y. Qi, X. Li, D. Hou, and M. Rung-Tsong Lyu, "ARF-predictor: Effective prediction of aging-related failure using entropy," *IEEE Trans. Dependable Secure Comput.*, vol. 15, no. 4, pp. 675–693, Aug. 2018.
[26] J. Li, Y. Qi, and L. Cai, "A hybrid approach for predicting aging-related failures of software systems," in *Proc. IEEE Symp. Service-Oriented Syst. Eng. (SOSE)*, Mar. 2018, pp. 96–105.
[27] J. Zhao, Y. Wang, G. Ning, K. S. Trivedi, R. Matias, Jr., and K.-Y. Cai, "A comprehensive approach to optimal software rejuvenation," *Perform. Eval.*, vol. 70, no. 11, pp. 917–933, Nov. 2013.
[28] A. Kumar and M. Saini, "Cost-benefit analysis of a single-unit system with preventive maintenance and Weibull distribution for failure and repair activities," *J. Appl. Math., Statist. Informat.*, vol. 10, no. 2, pp. 5–19, Jan. 2015.
[29] G. Levitin, L. Xing, and H. Ben-Haim, "Optimizing software rejuvenation policy for real time tasks," *Rel. Eng. Syst. Saf.*, vol. 176, pp. 202–208, Aug. 2018.
[30] L. Tian and R. Olshen. Survival Analysis. Stanford University, Stanford, CA, USA. Accessed: Jan. 2020. [Online]. Available: https://web.stanford.edu/~lutian/coursepdf/slideweek1.pdf
[31] D. Chen and K. S. Trivedi, "Analysis of periodic preventive maintenance with general system failure distribution," in *Proc. Pacific Rim Int. Symp. Dependable Comput. (PRDC)*, 2001, pp. 103–107.
[32] R. Matias, P. A. Barbetta, K. S. Trivedi, and P. J. F. Filho, "Accelerated degradation tests applied to software aging experiments," *IEEE Trans. Rel.*, vol. 59, no. 1, pp. 102–114, Mar. 2010.
[33] J. Zhao, Y. Jin, K. S. Trivedi, and R. Matias, Jr., "Injecting memory leaks to accelerate software failures," in *Proc. IEEE 22nd Int. Symp. Softw. Rel. Eng. (ISSRE)*, Nov. 2011, pp. 260–269.
[34] D. Cotroneo, R. Natella, R. Pietrantuono, and S. Russo, "A survey of software aging and rejuvenation studies," *ACM J. Emerg. Technol. Comput. Syst.*, vol. 10, no. 1, pp. 1–34, Jan. 2014.
[35] G. Carrozza, D. Cotroneo, R. Natella, A. Pecchia, and S. Russo, "Memory leak analysis of mission-critical middleware," *J. Syst. Softw.*, vol. 83, no. 9, pp. 1556–1567, Sep. 2010.
[36] R. Rojas-Cessa, Y. Kaymak, and Z. Dong, "Schemes for fast transmission of flows in data center networks," *IEEE Commun. Surveys Tuts.*, vol. 17, no. 3, pp. 1391–1422, 3rd Quart., 2015.
[37] B. Vamanan, J. Hasan, and T. N. Vijaykumar, "Deadline-aware datacenter tcp (D2TCP)," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 42, no. 4, pp. 115–126, Aug. 2012.
[38] J. Bai, X. Chang, F. Machida, K. S. Trivedi, and Z. Han, "Analyzing software rejuvenation techniques in a virtualized system: Service provider and user views," *IEEE Access*, vol. 8, pp. 6448–6459, 2020.
[39] G. Levitin, L. Xing, and L. Luo, "Joint optimal checkpointing and rejuvenation policy for real-time computing tasks," *Rel. Eng. Syst. Saf.*, vol. 182, pp. 63–72, Feb. 2019.
[40] G. Levitin, L. Xing, and Y. Xiang, "Optimizing software rejuvenation policy for tasks with periodic inspections and time limitation," *Relia. Eng. Sys. Saf.*, vol. 197, pp. 1–10, May 2020.



JINGWEI LI received the B.S. and M.S. degrees in computer science from Lanzhou University, China, in 2010 and 2013, respectively. She is currently pursuing the Ph.D. degree with the Department of Computer Science and Technology, Xi'an Jiaotong University, China. Her research interests include performance optimization, big data analytics, and efficient resource management.



YONG QI (Member, IEEE) received the Ph.D. degree from Xi'an Jiaotong University, China. He is currently a Full Professor with the Department of Computer Science and Technology, Xi'an Jiaotong University. His research interests include operating systems, distributed systems, cloud computing and big data systems, as well as system security and application. He is a member of the ACM.

GUANGHUA WANG received the B.S. and M.S. degrees in computer science from Northeast Forestry University, in 2011 and 2014, respectively. He currently works as an Engineer at the Satellite Control Center, Xi'an, China. His research interest is network operation and maintenance.

JINWEI LIN received the B.S. degree in computer science from Xi'an Jiaotong University, China, in 2018, where he is currently pursuing the master's degree with the Department of Computer Science and Technology. His research interests include cloud computing and power camping.