Journal of Electrical and Computer Engineering Innovations
(JECEI)
Journal homepage: http://www.jecei.sru.ac.ir

Research paper

On Multiple Objective of Software Rejuvenation Models with Several Policies

Z. Rahmani Ghobadi[1], H. Rashidi[2]*, S.H. Alizadeh[3]

[1]*Faculty of Computer and Information Technology Engineering, Qazvin Branch, Islamic Azad University, Qazvin, Iran.*
[2]*Department of Mathematics and Computer Science, Allameh Tabataba University, Tehran, Iran.*
[3]*ICT Research Institute, Iran Telecommunication Research Center, Tehran, Iran.*

Article Info

Article History:
Received 12 January 2021
Reviewed 25 February 2021
Revised 14 March 2021
Accepted 15 April 2021

Keywords:
Software rejuvenation
Performance
Availability
Cost

*Corresponding Author's Email Address:
hrashi@gmail.com

Abstract

Background and Objectives: Applications and systems software that are running constantly become obsolete due to the accumulation of error conditions or the depletion of resources like physical memory or performance degradation. In this regard, software rejuvenation has been proposed to deal with such a phenomenon and prevent software failure in the future. This paper proposes a multiple objective of software rejuvenation models with several policies. The purpose is to identify the right rejuvenation policy in practical situations.
Methods: We model software system with four policies using the Markov process. These policies are: (a) Software system without rejuvenation; (b) Software system with partial rejuvenation; (c) Software system with partial and full rejuvenation; and (d) Software system with four different types of rejuvenation. In the models and each policy, we consider assigning the level of performance on which the availability and operating costs are calculated.
Results: To evaluate the models with the four policies, many numerical experiments were performed. For each policy, we evaluated and compared three objectives, namely performance, availability and operating costs. The experimental results states that for Software System with the policy of four different type of rejuvenation have about 18 and 16 percent improvement in performance and availability, respectively, compared with those other policies. Moreover, the operating cost of the software system with partial rejuvenation policy is lower and more efficient than other policies.
Conclusion: According to the calculated objectives and the results of the policies, it can be concluded that in systems with lower operational costs, the most appropriate policy is the software system with four different types of rejuvenation because this policy bring the maximum possible value for the performance and availability. The result of this study showed that the combined method is not always a suitable method because its operating cost is higher than other methods and in systems that are more important in terms of cost, this policy is not appropriate.

Introduction

Given the accumulation of errors like memory leaks, numerical errors, fragmentation of storage space, system and application software can get obsolete during its continuous operation and aging. Moreover, their performance gradually declines and eventually breaks in

case of lack of maintenance [1]. Aging has been seen in various software systems, from critical systems like spacecraft systems [2] to commercial ones like web servers [3], embedded systems [4], billing software [5], transaction processing systems [6], and telecommunication switching [7]. Software rejuvenation is a commonly technique used to fight software aging and minimize system failure. This technique can remove the accumulated errors and frees operating system resources [8], also neutralize the effect of software aging and restore system function by occasionally stopping running software, cleaning the software's internal state, and restarting the system ([9], [10]).

The performance of software rejuvenation could drastically affect the rejuvenation program or policy. Particularly, system performance can be enhanced by frequent rejuvenation processes. In this respect, reaching optimal rejuvenation policy and enhancing the performance criteria of the system is of great significance [4].

In order to determine to what extent the rejuvenation policies can be applied in practice, this paper followed the research done in [10]. The rest of this paper is organized as follows. Next section reviews the related rejuvenation models. Then proposes the multiple objective software rejuvenation models with several policies and then shows the numerical experiments to evaluate and determine the best rejuvenation policy, and finally provides the conclusions of this research.

Literature Review

In this section, we review the latest studies dedicated to software rejuvenation. Generally, software aging refers to the escalation of failure rate or reduction of performance for a long-running time. More specifically, software aging effects can be associated with error accumulation and degrading resources that are leaked or corrupted states. Such impacts can be detected through aging indicators. In fact, system variables can be measured directly and can be associated with software aging [11]. Software rejuvenation has been defined as preemptive rollback of applications continuously running to prevent failures. Since an application might be unavailable during rejuvenation, it can exacerbate the downtime and lead to extra costs (e.g., financial losses). These costs, nonetheless, can be mitigated by rejuvenation scheduling during occasions when an application remains idle. Several studies have focused on the concepts of software aging and rejuvenation over the last few years. These studies have covered different aspects including the type of analysis, type of system, aging indicators, and rejuvenation techniques [9].

Regarding analysis, the rejuvenation methods are divided into three types of analysis: model-based, measurement-based, and hybrid. The model-based analyses involve a mathematical model, that includes states in which the system is correctly operating, states where the system is failure-proof, and states where software rejuvenation is underway. Several types of the model have been adopted for this purpose, including Markov Decision Processes and Stochastic Petri Nets ([12], [13]). The model-based method provides an abstract view of the system as well as a mathematical treatment. Capable of being grouped in time series analysis, the measurement-based method involves machine learning and other areas. The measurement-based rejuvenation approaches mainly serve to directly monitor system variables, e.g. aging indicators reflecting the onset of software aging, and predicting the incident of aging failures through analyzing the runtime data collected from a statistical perspective. This method provides accurate predictions on aging but requires direct monitoring and is difficult to generalize [14]. The hybrid method combines the benefits of model-based and measurement-based approaches. In this paper, we adopted the hybrid model as well [9]. An important problem concerns the type of system on which aging is analyzed. Moreover, it covers the rejuvenation actions implemented. It has been proven that software aging can affect numerous types of long-running software systems. Such systems are divided into three categories: safety-critical, non-safety-critical, and unspecified [15].

During the past 20 years, software rejuvenation has been extensively studied with the aim to design rejuvenation policies that optimize system availability and performance, mainly in terms of operational costs [16]. In [17] , refer to software availability modes with rejuvenation. Moreover, in [18], the authors presented a comprehensive availability model that considers failures and recovery behaviors of multiple virtual machines, various failure modes, and recovery behaviors and dependencies between different system subcomponents. A comprehensive model for availability evaluation of cloud computing with virtual machine monitor rejuvenation through virtual machine migration scheduling was presented in [19], where rejuvenation policies that maximize system availability through two migration approaches were proposed. A three-level rejuvenation policy under a Markov modeling framework was proposed for an active/standby cluster system in [20], where the steady-state availability and the downtime cost were the measures of interest.

To sum up, there are four policies for rejuvenation: (a) Check out completed applications [11]; (b) Application restart or partial rejuvenation (i.e. the whole application is restarted) [21], [22], [23], [24], (c) OS Reboot or full rejuvenation (i.e. it restarts the OS) [21], [22]; and (d) Turn off the system (at the level of the physical machine) [11].

The main idea in this paper is to evaluate the four policies of rejuvenation for a system that is always in service with lower performance. Hence, it aims to model the system's performance during software implementation from a powerful and robust mode to a failure mode. Moreover, some measures are suggested for preventing the complete failure of the system during the implementation by applying different types of rejuvenation.

The Proposed Multiple Objective Software Rejuvenation Models

In this section, the multiple objective software rejuvenation models are proposed. In the models, we consider four policies with leveling the performance of systems. System performance could be perceived as the ability of the system to provide services at an acceptable level. This ability that is known as performance was modeled and evaluated in the present study. The modeling shows the change in the performance of the system through destruction from a robust state to a state of failure.

It is supposed that the system will go into a state with lower performance during the rejuvenation activity. In this state, a performance is defined, followed by estimating the effect of various rejuvenation policies on it. Hence, we have considered the assumptions to implement rejuvenation policies, as follows:

- First, the system works in a full performance state. Due to aging, the level of performance diminishes with a decrease in the resources including memory usage. The policy used includes partial [23], full [25], and policy with four different types of rejuvenation [11]. One of the rejuvenation types will be activated when the system performance decreases and reaches 80%, 60%, 40% and 20%. In the case of a fault, the performance level decreases to 0%.
- While the rejuvenation activity is running, the level of software performance declines due to the nature and the type of rejuvenation. For partial rejuvenation, the performance rate is assumed to be 30%. This reduction in performance level during the partial rejuvenation activity involves stopping and restarting the program process. Meanwhile, the system can still provide services, although the levels are decreased. This number is 30% of an indicator level and is used to indicate the effect of partial rejuvenation on system performance.
- Full rejuvenation includes restarting the operating system. All running programs must be stopped before the operating system restart. Thus, a full rejuvenation can be considered a two-part operation: The first part is the proper termination of all running programs and the second restarting is the operating system. Here, it is assumed that the system performance during the full rejuvenation is 10%.
- Four different types of rejuvenation may be used. The first type is a partial restart of services during which some services are down and unused but still take up memory space. Instead of shutting down the system completely, these services shut down completely and free up the memory they had. As the programs running in these conditions continue to run, no change occurs in the level of system performance during this type of rejuvenation. In the second type, in addition to the terminated services, the service running is stopped and memory is freed; thus, it is supposed that system performance will be 30% by doing so. In the third type, a total restart happens; i.e., the termination of the terminated services, the running service, and the operating system. Accordingly, a large amount of memory is freed and the system is transferred to a more efficient mode. The performance of the system will be 10% while performing this type of rejuvenation. The fourth approach is the operation that occurs at the level of the physical machine. This approach, which is the most common method used, is based on turning on and off. This rejuvenation mode is the most expensive one, as well, and transfers the system to a powerful and robust state. The performance of the system is assumed 0% during this type of rejuvenation.

In the policies presented, each policy corresponds to a level of performance where a software system can serve. Software rejuvenation has been proposed to deal with errors that can lead to possible malfunctions. Four types of policies have been assumed to examine the effects of all possible software rejuvenation: (a) Software system without rejuvenation (SSWR) policy;(b) Software system with partial rejuvenation (SSPR) policy; (c) Software system with partial and full rejuvenation (SSPFR) policy;and (d) Software system with four different types of rejuvenation (SS4DTR) policy.

Each of these policies is modeled separately in the next subsections. To investigate this rejuvenation policies effectiveness, it is necessary to calculate some criteria. Considering that most of the articles, including [26], [27] and [28] reviewed and discussed the criteria of performance, availability and cost, we also in this study has evaluated performance, availability, and operating cost. The time Markov process {Z (t), t≥0} used to evaluate the above criteria. Markov process is {Z (t), t≥0}, m ∈ {Policy1, Policy2, Policy3, Policy4} be used to describe the evaluation of any policy. Abbreviations provide the definitions needed to calculate performance, reliability, and cost for any policy.

System availability is defined, the probability of the system working at moment t.

To mathematically define availability, we must first divide the system state-space S_P into two subsets of operational states (U_P) and down states (D_P) such that $S_P=U_P \cup D_P$, $U_P \cap D_P = 0$.

The U_P = {1,…, r} is the set of active states of the system and D_P = {r + 1,…, s} is the set of inactive states of the system. Availability for each policies expressed as (1) [29].

$$A(t) = \alpha_P e^{t \cdot Q_P} \cdot 1_{s.r}.\ t \geq 0 \tag{1}$$

Although rejuvenation can delay the failure, it can be costly. If partial or full, or a four types rejuvenation is a plan, it costs less than unplanned failure. In the second case (i.e., unplanned failure), a repair process begins that is costly and time-consuming. However, not all rejuvenation policiescan benefit from a software system [28].

A. Policy 1: Software System without Rejuvenation (SSWR)

In this software policy, no rejuvenation activity occurs. Figure 1 shows the state transition diagram for the SSWR policy. At first, the system starts in a robust state, fully capable of providing services. In this case, it is assumed that the system's amount of free memory is at its highest level, that is 100%, and the system is operating at the highest possible performance. This state, as shown in Fig. 1, is indicated by the state S_0. Due to memory consumption, the system experiences other states before failure. To model the performance drop due to performance degradation, four states of S_H, S_M, S_L, and S_U have considered which the system performance levels are assumed to be 80%, 60%, 40%, and 20%, respectively. State F indicates a failure state in which the system can't provide any service, and its performance level is 0%.

The exponential distribution time in each system state assumes that λ_H is the constant transition rate from the robust state S_0 to the degraded state S_H. Correspondingly, λ_M is the fixed transition rate from the state S_H to the medium performance state S_M, and λ_L is the fixed transition rate from the S_M mode to the low-performance state S_L. λ_U is the constant transition rate from the state S_L to the unstable state S_U. Finally, the system with λ_F rate reaches a failure in the state F. After a failure occurs, the repair process begins to restore the system to a robust shape. The corresponding constant transition rate is express by μ_{rep}. Note after the repair is complete, the system is as good as the new system.

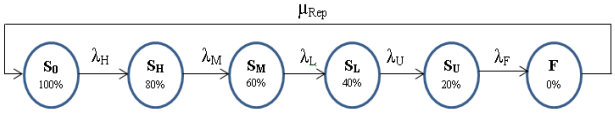

Fig. 1: The state transition diagram for SSWR policy.

To evaluate the system's performance, we must define the level of performance in each case in which the system may be located and examine each of the proposed policies (Policies 1, 2, 3, and 4). The system performance level is defined as (2):

$$PL_P = [PL_P(i), i \in S_P], \quad S_P \in \{Policy1, Policy2, Policy3, Policy4\} \tag{2}$$

PL_P is the performance level of the Policy P. Therefore, the indicator that calculates the overall performance of the software system at any given time is as (3):

$$PI_P(t) = \alpha_P e^{t.Q_P} (PL_P)^T, \ t \geq 0, \quad P \in \{Policy1, Policy2, Policy3, Policy4\} \tag{3}$$

where the Q_P is the transition rate matrix for the Markov process, and α_P is the initial probability distribution of the Policy P. PI_P (t) indicates the system performance at any time t. It indicates whether the system is enabled (U_P) or disabled (D_P) mode.

Since the software systems work continuously in runtime, the relevant criterion for the policy P, according to the above equation, is evaluated as (4):

$$PI_P = \pi_P (PL_P)^T \tag{4}$$

where the π_P is the constant state probability distribution for the policy P obtained by solving the linear (5):

$$\pi_P \cdot Q_P = 0, \quad \sum_{i \in S_P} \pi_{P.i} = 1 \tag{5}$$

For the SSWR policy, the corresponding state-space is S_{P1}={S_0, S_H, S_M, S_L, S_U, F}. The state transition matrix for this policy, which is required for PI_P evaluation, is formulated as (6):

$$Q_{P1} = \begin{bmatrix} -\lambda_H & \lambda_H & 0 & 0 & 0 & 0 \\ 0 & -\lambda_M & \lambda_M & 0 & 0 & 0 \\ 0 & 0 & -\lambda_L & \lambda_L & 0 & 0 \\ 0 & 0 & 0 & -\lambda_U & \lambda_U & 0 \\ 0 & 0 & 0 & 0 & -\lambda_F & \lambda_F \\ \mu_{Rep} & 0 & 0 & 0 & 0 & -\mu_{Rep} \end{bmatrix} \tag{6}$$

Based on the state-space of S_{P1} and Q_{P1} matrix and according to (3), the performance index for the SSWR policy can evaluate as (7):

$$PI_{P_1}(t) = \alpha_{P_1} e^{t \cdot Q_{P1}} (PL_{P_1})^T . t \geq 0 \tag{7}$$

Assuming that the software system starts from state S_0, that is, the performance level is 100%, then PI_{P1} (t) is calculated as (8):

$$\begin{aligned} PI_{P_1}(t) &= [1 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0] e^{t \cdot Q_{P1}} \\ &\quad [1 \quad 0.80 \quad 0.60 \quad 0.40 \quad 0.20 \quad 0]^T \\ &= \pi_{P1.0}(t) + 0.80\pi_{P1.H}(t) + 0.60\pi_{P1.M}(t) \\ &\quad + 0.40\pi_{P1.L}(t) + 0.20\pi_{P1.U}(t);\ t \geq 0 \end{aligned} \tag{8}$$

By solving linear equations in (5) and with using (4), the software system performance can be expressed as (9):

$$PI_{P1} = \pi_{P1.0} + 0.80\pi_{P1.H} + 0.60\pi_{P1.M} + 0.40\pi_{P1.L} + 0.20\pi_{P1.U} \quad (9)$$

To model the availability in the SSWR policy, the state-space $S_{P1} = \{S_0, S_H, S_M, S_L, S_U, F\}$ can be divided into $U_{P1} = \{S_0, S_H, S_M, S_L\}$ and $D_{P1} = \{S_U, F\}$ (see Fig. 1). Note that the states S_0, S_H, S_M, S_L are the operating states, and the states S_U, F are the unstable and failure states of the system in which the system does not work properly. Based on the state-space $S_{P1}$and the Q_{P1} matrix in (6), the availability of the SSWR policies evaluated according to (10):

$$A_{P1}(t) = \alpha_{P1} e^{t.Q_{P1}} 1_{6.4} \,.\, t \geq 0 \quad (10)$$

Availability is evaluated according to the (11) and (12):

$$A_{P1}(t) = [1 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0] e^{t.Q_{P1}} [1 \quad 1 \quad 1 \quad 1 \quad 0 \quad 0]^T = \pi_{P1.0}(t) + \pi_{P1.H}(t) + \pi_{P1.M}(t) + \pi_{P1.L}(t);\ t \geq 0 \quad (11)$$

$$A_{P1} = \pi_{P1.0} + \pi_{P1.H} + \pi_{P1.M} + \pi_{P1.L} \quad (12)$$

The system designer must carefully decide on the rejuvenation policy, to determine whether rejuvenation is beneficial in terms of operational cost, at first, the cost for the SSWR policy is evaluated. Therefore, the cost of downtime for a SSWR policy is defined as (13):

$$C_{P1}(i) = \begin{cases} C_{rep}, if\ i \in F \\ 0 \quad , \quad else \end{cases} \quad (13)$$

where C_{rep} is the system repair cost. The total downtime cost per unit time for the SSWR policies calculated as (14):

$$TEOC_{P1} = C_{rep}.\pi_{P1.F} \quad (14)$$

B. Policy 2: Software System with Partial Rejuvenation (SSPR)

The partial rejuvenation can counteract system performance degradation [22], [23]. The partial rejuvenation has minimal effect and can make running applications more usable [22].

Fig. 2 shows the state transition diagram for the SSPR policy.

The system initially starts in a robust state. This state is expressed by S_0, where the system performance level is 100%. Over time, with performance degradation, the system state change and reaches the level of performance 80% and then 60% and then 40%, and finally to the unstable state, that is 20% of the performance level. If the performance degradation continues, system failure will occur.

In each of these states, the partial rejuvenation can occur and bring the system into a better situation. The change of situation between the states occurs when the partial rejuvenation is applied, for example, from S_H to S_{HPR} with transition rate r_p.

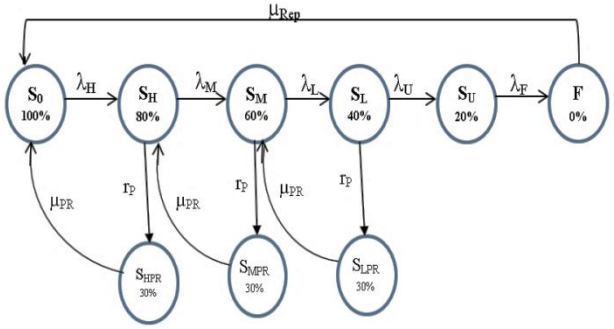

Fig. 2: The state transition diagram for the SSPR policy.

In the software system where only the partial rejuvenation is active, the corresponding state-space is $S_{P2} = \{S_0, S_H, S_M, S_L, S_U, F, S_{HPR}, S_{MPR}, S_{LPR}, S_{UPR}\}$ which states S_{HPR}, S_{MPR}, S_{LPR}, S_{UPR} are rejuvenation states. The performance in each case can obtain according to Fig. 2. The state transition matrix for this policy is defined as (15):

$$Q_{P2} = \begin{bmatrix} -\lambda_H & \lambda_H & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -(\lambda_M + r_p) & \lambda_M & 0 & 0 & 0 & r_p & 0 & 0 & 0 \\ 0 & 0 & -(\lambda_L + r_p) & \lambda_L & 0 & 0 & 0 & r_p & 0 & 0 \\ 0 & 0 & 0 & -(\lambda_U + r_p) & \lambda_U & 0 & 0 & 0 & r_p & 0 \\ 0 & 0 & 0 & 0 & -(\lambda_F + r_p) & \lambda_F & 0 & 0 & 0 & r_p \\ \mu_{Rep} & 0 & 0 & 0 & 0 & -\mu_{Rep} & 0 & 0 & 0 & 0 \\ \mu_{PR} & 0 & 0 & 0 & 0 & 0 & -\mu_{PR} & 0 & 0 & 0 \\ 0 & \mu_{PR} & 0 & 0 & 0 & 0 & 0 & -\mu_{PR} & 0 & 0 \\ 0 & 0 & \mu_{PR} & 0 & 0 & 0 & 0 & 0 & -\mu_{PR} & 0 \\ 0 & 0 & 0 & \mu_{PR} & 0 & 0 & 0 & 0 & 0 & -\mu_{PR} \end{bmatrix} \quad (15)$$

Like the SSWR policy, the software system starts from state S_0, which the performance is 100%. The performance for the SSPR policies calculated as (16):

$$PI_{P2}(t) = [1 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0] e^{t.Q_{P2}} [1 \quad 0.80 \quad 0.60 \quad 0.40 \quad 0.20 \quad 0 \quad 0.30 \quad 0.30 \quad 0.30]^T = \pi_{P2.0}(t) + 0.80\pi_{P2.H}(t) + 0.60\pi_{P2.M}(t) + 0.40\pi_{P2.L}(t) + 0.30(\pi_{P2.HPR}(t) + \pi_{P2.MPR}(t) + \pi_{P2.LPR}(t)) + 0.20\pi_{P2.U}(t);\ ;\ t \geq 0 \quad (16)$$

By solving the linear equations, the software system performance is expressed as (17):

$$PI_{P2} = \pi_{P2.0} + 0.80\pi_{P2.H} + 0.60\pi_{P2.M} + 0.40\pi_{P2.L} + 0.30(\pi_{P2.HPR} + \pi_{P2.MPR} + \pi_{P2.LPR}) + 0.20\pi_{P2.U} \quad (17)$$

To model the availability in the SSPR policy according to Fig. 2, the state-space is S_{P2}= {S_0, S_H, S_M, S_L, S_U, F, SHPR, SMPR, SLPR, SUPR} and the state-space can be divided into active U_{P2}={S_0, S_H, S_M, S_L} and down D_{P2}={ S_U, F,S_{HPR}, S_{MFR}, S_{UPR} , S_{LFR}}. Although during system rejuvenation, the system's performance is 30%, they are considered down states. The availability of the SSPR policies expressed as (18):

$$A_{P2} = \pi_{P2.0} + \pi_{P2.H} + \pi_{P2.M} + \pi_{P2.L} \quad (18)$$

In the case of partial rejuvenation, the system can continue the service at a low level of performance, and the total downtime cost is calculated as (19):

$$C_{P2}(i) = \begin{cases} C_{rep}, & if\ i \in \{F\} \\ C_{PR}, & if\ i \in \{S_{HPR}, S_{MPR}, S_{LPR}\} \\ 0, & else \end{cases} \quad (19)$$

where C_{rep} is the system repair cost and C_{PR} is the cost of performing partial rejuvenation. Therefore, the total cost of downtime per unit time for SSPR policies expressed as (20):

$$TEOC_{P2} = C_{rep}.\pi_{P2.F} + C_{PR}.(\pi_{P2.HPR} + \pi_{P2.MPR} + \pi_{P2.LPR}) \quad (20)$$

C. Policy 3: Software system with the partial and full rejuvenation (SSPFR)

Fig. 3 shows the state transition diagram for the SSPER policy. Here, the system initially starts in a robust state. This state is expressed as S_0, where the performance level is 100%. Over time, the system's performance degradation increases, and the system performance level is assumed to be reduced to 80% and then 60% and 40%, and finally, to the unstable state, that is, 20%.

If the performance degradation continues, the system may experience a software failure. In each of these states, the partial or full rejuvenation can occur and improve the system's state depending on the performance level. As shown in Fig. 3, when the system reaches the state S_H, it is better to perform a partial rejuvenation activity to move the system state to the optimal condition. In states with performance levels 60%, 40%, and 20%, it is better to have a full rejuvenation to optimize the system.

The partial and full rejuvenation occurs with the rate r_p and r_F, respectively. In full rejuvenation, the system will have a low performance for a more extended period. When the rejuvenation activity is performed, the system will go into a state of low performance.

$$Q_{P3} = \begin{bmatrix} -\lambda_H & \lambda_H & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -(\lambda_M + r_p) & \lambda_M & 0 & 0 & 0 & r_p & 0 & 0 & 0 \\ 0 & 0 & -(\lambda_L + r_F) & \lambda_L & 0 & 0 & 0 & r_F & 0 & 0 \\ 0 & 0 & 0 & -(\lambda_U + r_F) & \lambda_U & 0 & 0 & 0 & r_F & 0 \\ 0 & 0 & 0 & 0 & -(\lambda_F + r_p) & \lambda_F & 0 & 0 & 0 & r_p \\ \mu_{Rep} & 0 & 0 & 0 & 0 & -\mu_{Rep} & 0 & 0 & 0 & 0 \\ \mu_{PR} & 0 & 0 & 0 & 0 & 0 & -\mu_{PR} & 0 & 0 & 0 \\ \mu_{FR} & 0 & 0 & 0 & 0 & 0 & 0 & -\mu_{FR} & 0 & 0 \\ \mu_{PR} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -\mu_{FR} & 0 \\ 0 & 0 & 0 & \mu_{PR} & 0 & 0 & 0 & 0 & 0 & -\mu_{PR} \end{bmatrix} \quad (21)$$

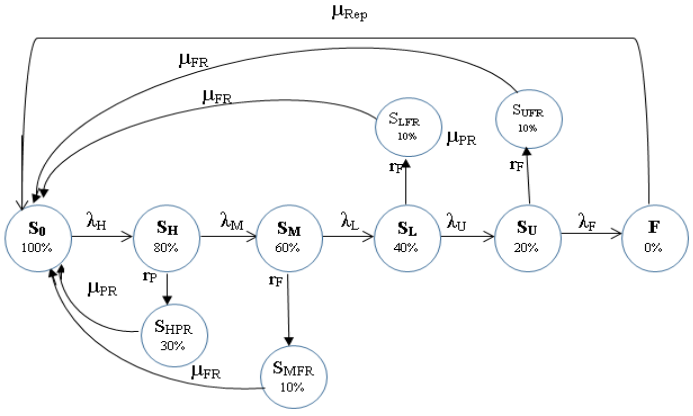

Fig. 3: The state transition diagram for the SSPFR policy.

In the SSPFR policy, the corresponding state-space is S_{P3}= {S_0, S_H, S_m, S_L, S_U, F, S_{HPR}, S_{MFR}, S_{LFR}, S_{UPR}}, which the states S_{HPR} and S_{UPR} are for partial rejuvenations and the states S_{MFR} and S_{LFR} are for full rejuvenations. The performance can be obtained according to Fig. 3. The state transition matrix for this policy is defined as (21).

Similar to the previous policy, the software system starts from the state S_0, which the performance is 100%. The performance for the SSPFR is calculated as (22):

$$\begin{aligned} PI_{P3}(t) &= [1 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0] e^{t \cdot Q_{P3}} \\ &\quad [1 \;\; 0.80 \;\; 0.60 \;\; 0.40 \;\; 0.20 \;\; 0 \;\; 0.30 \;\; 0.10 \;\; 0.10 \;\; 0.10]^T \\ &= \pi_{P3.0}(t) + 0.80\pi_{P3.H}(t) + 0.60\pi_{P3.M}(t) \\ &\quad + 0.40\pi_{P3.L}(t) + 0.30\pi_{P3.HPR}(t) \\ &\quad + 0.20\pi_{P3.U}(t) + 0 \\ &\quad \cdot 10\big(\pi_{P3.MFR}(t) + \pi_{P3.LFR}(t) \\ &\quad + \pi_{P3.UPR}(t)\big);\; t \geq 0 \end{aligned} \tag{22}$$

By solving the linear equations, the performance of software system is expressed as (23):

$$\begin{aligned} PI_{P3} &= \pi_{P3.0} + 0.80\pi_{P3.H} + 0.60\pi_{P3.M} \\ &\quad + 0.40\pi_{P3.L} + 0.30\pi_{P3.HPR} \\ &\quad + 0.20\pi_{P3.U} \\ &\quad + 0 \cdot 10(\pi_{P3.MFR} + \pi_{P3.LFR} + \pi_{P3.UPR}) \end{aligned} \tag{23}$$

When both partial and full rejuvenation are used, the state-space divided two subsets: up U_{P3} = {S_0, S_H, S_M, S_L} and down D_{P3} = {S_U, F, S_{HPR}, S_{MFR}, S_{UPR}, S_{LFR}}. The availability of the SSPFR policies expressed as (24):

$$A_{P3} = \pi_{P3.0} + \pi_{P3.H} + \pi_{P3.M} + \pi_{P3.L} \tag{24}$$

In the SSPFR policy, reduced the level of performance, the costs imposed on the system is considered as (25):

$$C_{P3}(i) = \begin{cases} C_{rep}, & if\ i \in \{F\} \\ C_{FR} & if\ i \in \{S_{LFR}, S_{MFR}, S_{UfR}\} \\ C_{PR}, & if\ i \in \{S_{HPR}\} \\ 0\,, & else \end{cases} \tag{25}$$

Therefore, the total cost of downtime per unit time can be calculated as (26):

$$\begin{aligned} TEOC_{P3} &= C_{rep}.\pi_{P3.F} \\ &\quad + C_{FR}.(\pi_{P3.MFR} + \pi_{P3.LFR} \\ &\quad + \pi_{P3.UFR}) + C_{PR}.(\pi_{P3.HPR}) \end{aligned} \tag{26}$$

D. Policy 4: Software system with four different types of rejuvenation (SS4DTR)

This policy keeps the system in its best situation by applying four rejuvenation types at four different levels.

Fig. 4 shows the state transition diagram for the SS4DTR policy.

At first, the system starts in a robust state, where the performance level is 100%; this state is shown as S_0.Over time, the performance level degrades to 80% and then 60% and 40%, finally, to the unstable state 20%. If the performance degradation continues, the system will eventually go to failure. Each of the four types of rejuvenation can occur in these states and improve the system's condition depending on the performance levels.

When the system reaches the state S_H, which is a performance level 80%, it is better to have a type I rejuvenation, which will have the lowest downtime cost rejuvenation activity, as shown in Fig. 4. In the state SP_{1R}, the system runs and continues to work without reducing the level of performance. After the rejuvenation activity, the system is transferred to the optimal state with a performance level 100%.

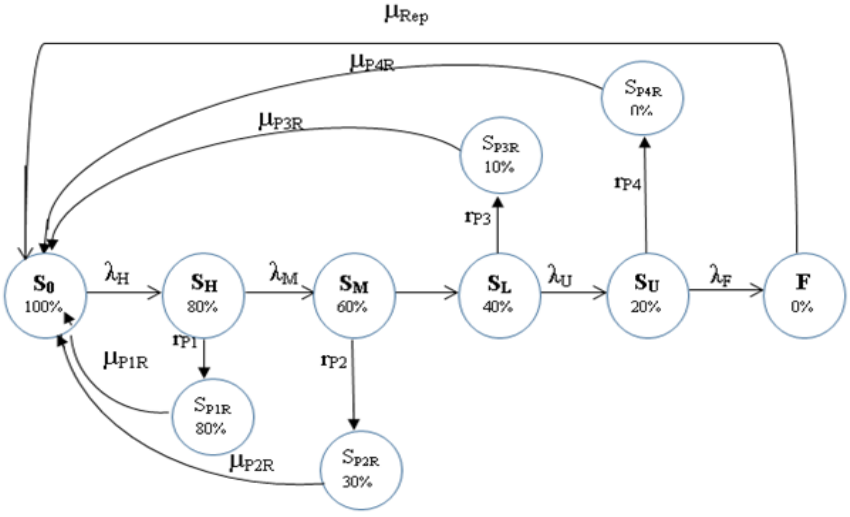

Fig. 4: The state transition diagram for the SS4DTR policy.

When the system reaches S_M state with a performance level 60%, it can apply type II rejuvenation. During this type of rejuvenation, the system performance level will be 50% (the state S_{P2R}) and after completion, the system is transferred to the optimal state 100%. When the system reaches S_L state with a performance level 40%, type III rejuvenation can be performed. During this type's rejuvenation, the system performance level will be 10% (the state S_{P3R}), and after rejuvenation completion, the system is going to the optimal state (performance level 100%).Type IV rejuvenation is the best choice when the system reaches an unstable state with a performance level 20%.

However, the system downtime increase, and the performance level is 0% during this type of rejuvenation but compared to the time it takes to repair, this rejuvenation will be more appropriate. It will eventually bring the system back to the optimal situation. Finally, when the system fails (the state F), the only solution is to repair, which has a high operational cost and downtime.

In the SS4DTR policy, the state-space is $S_{P4}=\{S_0, S_H, S_M, S_L, S_U, F, S_{P1R}, S_{P2R}, S_{P3R}, S_{P4R}\}$, which each of states S_{P1R}, S_{P2R}, S_{P3R}, S_{P4R} are related to the first, second, third and fourth type of rejuvenation, respectively. The performance can be obtained according to Fig. 4. The state transition matrix for this policy is defined as (27).

$$Q_{P4}=\begin{bmatrix} -\lambda_H & \lambda_H & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -(\lambda_M+r_{p1}) & \lambda_M & 0 & 0 & 0 & r_{P1} & 0 & 0 & 0 \\ 0 & 0 & -(\lambda_L+r_{p2}) & \lambda_L & 0 & 0 & 0 & r_{P2} & 0 & 0 \\ 0 & 0 & 0 & -(\lambda_U+r_{p3}) & \lambda_U & 0 & 0 & 0 & r_{P3} & 0 \\ 0 & 0 & 0 & 0 & -(\lambda_F+r_{p4}) & \lambda_F & 0 & 0 & 0 & r_{P4} \\ \mu_{Rep} & 0 & 0 & 0 & 0 & -\mu_{Rep} & 0 & 0 & 0 & 0 \\ \mu_{P1R} & 0 & 0 & 0 & 0 & 0 & -\mu_{P1R} & 0 & 0 & 0 \\ \mu_{P2R} & 0 & 0 & 0 & 0 & 0 & 0 & -\mu_{P2R} & 0 & 0 \\ \mu_{P3R} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -\mu_{P3R} & 0 \\ \mu_{P4R} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -\mu_{P4R} \end{bmatrix} \quad (27)$$

Similar to the previous policy, the software system starts from the state S_0, which the performance is 100%. The performance for SS4DTR policy is calculated as (28):

$$\begin{aligned} PI_{P4}(t) &= [1 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0] e^{t\cdot Q_{M4}} \\ &\quad [1 \;\; 0.80 \;\; 0.60 \;\; 0.40 \;\; 0.20 \;\; 0 \;\; 0.80 \;\; 0.50 \;\; 0.10 \;\; 0]^T \\ &= \pi_{P4.0}(t) + 0.80\big(\pi_{P4.H}(t) + \pi_{P4.P1R}(t)\big) \\ &\quad + 0.60\pi_{P4.M}(t) + 0.50\pi_{P4.P2R}(t) \\ &\quad + 0.40\pi_{P4.L}(t) + 0.20\pi_{P4.U}(t) \\ &\quad + 0\cdot 10\pi_{P4.P3R}(t)\;;\; t \geq 0 \end{aligned} \quad (28)$$

By solving the linear equations, the software system performance is expressed as (29):

$$\begin{aligned} PI_{P4} &= \pi_{P4.0} + 0.80(\pi_{P4.H} + \pi_{P4.P1R}) + 0.60\pi_{P4.M} \\ &\quad + 0.50\pi_{P4.P2R} + 0.40\pi_{P4.L} + 0.20\pi_{P4.U} + 0 \\ &\quad \cdot 10\pi_{P4.P3R} \end{aligned} \quad (29)$$

when SS4DTR policy is used, the state-space is divided into two up and down subsets, $U_{P4}=\{S_0, S_H, S_M, S_L, S_{P1R}, S_{P2R}\}$and $D_{P4}=\{S_U, F, S_{P4R}, S_{P3R}\}$.

The availability of the SS4DTR policy is expressed as (30):

$$\begin{aligned} A_{P4} &= \pi_{P4.0} + \pi_{P4.H} + \pi_{P4.M} + \pi_{P4.L} + \pi_{P4.P1R} \\ &\quad + \pi_{P4.P2R} \end{aligned} \quad (30)$$

In this policy, four different types of rejuvenation have applied, and the costs imposed on the system are considered as (31):

$$C_{P4}(i)=\begin{cases} C_{rep}, & if\ i \in \{F\} \\ C_{4R}, & if\ i \in \{S_{P4R}\} \\ C_{3R}, & if\ i \in \{S_{P3R}\} \\ C_{2R}, & if\ i \in \{S_{P2R}\} \\ C_{1R}, & if\ i \in \{S_{P1R}\} \\ 0, & else \end{cases} \quad (31)$$

Therefore, the total cost of downtime per unit time is calculated as (32):

$$\begin{aligned} TEOC_{P4} &= C_{rep}.\pi_{P4.F} + C_{4R}.\pi_{P4.P4R} \\ &\quad + C_{3R}.\pi_{P4.P3R} + C_{2R}.\pi_{P4.P2R} \\ &\quad + C_{1R}.\pi_{P4.P1R} \end{aligned} \quad (32)$$

Numerical Experiments

In this section, we intend to provide a comparison between the four policies with numerical experiments. The data used are present in Table 1, do not come out of real-life software systems, but it is under the relevant software rejuvenation literature [17], [30].

Figs. 5, 6, and 7 shows the system's performance, availability, and operational cost for SSWR, SSPR, SSPFR, and SS4DTR policies in 48 hours' period. As can be seen, for SSWR policy, the system performance and availability decrease with the operational time since without any proactive actions, performance and availability degradation cannot be avoided either an eventual failure. The high probability of an eventual failure affects

the cost indicator, which increases with the operational time.

Table 1: Parameter values for comparing policies

Parameter	Value
λ_H	8 h^{-1}
λ_M	16 h^{-1}
λ_L	24 h^{-1}
λ_U	32 h^{-1}
λ_F	40 h^{-1}
μ_{rep}	6 h^{-1}
$\mu_{PR} = \mu_{P3R}$	4 h^{-1}
$\mu_{FR} = \mu_{P4R}$	5 h^{-1}
$r_P = r_{P3}$	4 h^{-1}
$r_F = r_{P4}$	20 h^{-1}
r_{P1}	0.5 h^{-1}
r_{P2}	2 h^{-1}
μ_{P1R}	1 h^{-1}
μ_{P2R}	2 h^{-1}
C_{rep}	50 cost units per hour
$C_{PR} = C_{P3R}$	5 cost units per hour
$C_{FR} = C_{P4R}$	10 cost units per hour
C_{P2R}	2 cost units per hour
C_{P1R}	1 cost units per hour

In the SSPR policy, the rejuvenation policy does every 4 h; as seen in the previous policy, over time, the performance of the system will decrease, but in this policy due to the partial rejuvenation is expected to occur at a slower rate, and often expected to provide a higher level of performance. It should also consider that rejuvenation activities will incur costs, including unavailability of the system during software rejuvenation, so that rejuvenation activities will reduce availability. Fig. 6 shows the performance, availability, and operational costs for the SSPR policy.

In the SSPFR policy, the partial rejuvenation does once every 4 h and the full rejuvenation does once every 20h. Over time, system performance will decrease, but given the partial and full rejuvenation enabled, this performance reduction be expected to occur at a slower rate, as shown in Fig. 3 when performance is 80%, the partial rejuvenation performed. When performance is 60% or 40% or 20%, the full rejuvenation performed. According to the values of Table 1 and the equations obtained in the previous section, the performance, availability, and operating cost diagrams are present in Fig. 5, 6, and 7.

In the SS4DTR policy, as shown in Fig. 4, when the performance is 80%, the first type of rejuvenation, in the case of 60% performance, the second type of rejuvenation, in the case of 40% performance, the third type and in the case with 20% performance, the fourth type is rejuvenated. According to the values of Table 2 and the equations obtained in the previous section, the performance, availability, and operating cost diagrams are present in Fig. 5, 6, and 7.

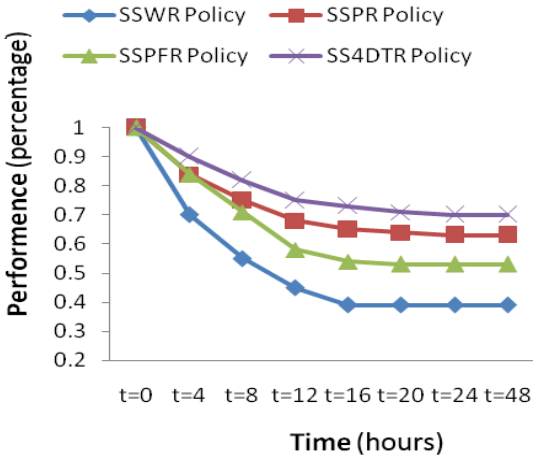

Fig 5. The performance of the policies.

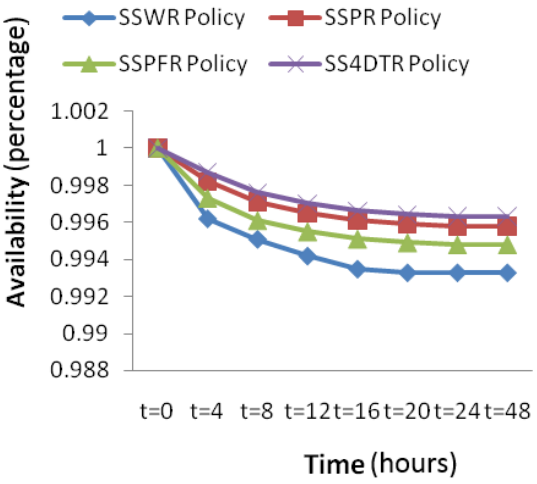

Fig 6: The availability of the policies.

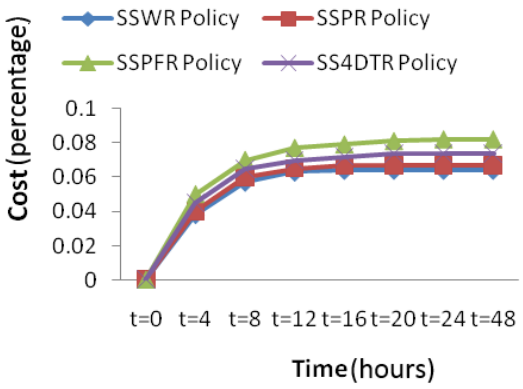

Fig 7: The cost of the policies.

Comparison of Numerical Results Expressed Policies

According to the obtained results, the policies can be compared in terms of performance, availability, and

operating costs. According to Fig. 5, it seems that the SS4DTR policy provides the highest performance among all policies.

As shown in Fig. 6, SS4DTR policy is also more efficient in terms of availability, and this is since the system in situations where the first and second type of rejuvenation be performed with a slight reduction in performance, Still available.

Fig. 7 shows the operating cost of each policy. As can be seen, in contrast to the performance and availability that was optimal for the SS4DTR policy, the operating cost for the SSPR policy is proportional. It is lower and more efficient than other policies.

Conclusion

This paper followed the research done in [10] that its main purpose was to find out the best policy in the rejuvenation models for software systems and in order to determine to what extent the existing polices can be applied in practice. For this purpose, we consider three objectives, namely performance, availability and, operating costs. In the rejuvenation models, we investigated four different policies, including software system without rejuvenation (SSWR), with the partial rejuvenation (SSPR), with the partial and full rejuvenation (SSPFR), and system with four different types of rejuvenation (SS4DTR). For each policy, the performance, availability, and operational cost are calculated. According to the presented policies, calculated objectives, and the values of each of these objectives, we saw that SS4DTR policy in terms of performance and availability works better than other policies. On the other hand, applying SS4DTR policy caused more costs, and the SSPR policy had a lower operating costs than other policies. Therefore, it can be concluded that in systems with lower operational costs, the most appropriate policy is the SS4DTR, because it has the maximum possible value in the performance and availability. The result of this study showed that the combined method is not always a suitable method because its operating cost is higher than other methods and in systems that are more important in terms of cost, this policy is not appropriate.

Author Contributions

Z. Rahmani Ghobadi, H. Rashidi, and S. Hoseinalizadeh presented the multiple objective of software rejuvenation models with four policies. Z. Rahmani Ghobadi examined each policy and wrote the manuscript. H. Rashidi and S. Hoseinalizadeh interpreted the results and improved the structure of paper.

Acknowledgment

The authors gratefully thank the anonymous reviewers and the editor of JECEI for their useful comments and suggestions.

Conflict of Interest

The authors declare no potential conflict of interest regarding the publication of this work. In addition, the ethical issues including plagiarism, informed consent, misconduct, data fabrication and, or falsification, double publication and, or submission, and redundancy have been completely witnessed by the authors.

Abbreviations

E_P	State-Space for the policy P
$Q_P = (q_{ij})_{i,j\epsilon E_P}$	It is a square matrix whose elements represent the probability transition matrix between the states.
$\alpha_P = (\alpha_P(i))_{i\epsilon E_P}$	The initial probability distribution of the Z_P (t) process for the policy P
$l_{s,r} = (1 \quad \dots \quad 1 \quad 0 \quad \dots \quad 0)^T$	The column vector s is the next in which r is the first element equal to 1, and the remainder s-r is equal to zero.
$l_r = (1 \quad \dots \quad 1)^T$	It is a next r column vector whose elements have a value of 1.
λ_H	constant transition rate from the robust state S_0 to the degraded state S_H
λ_M	fixed transition rate from the S_H state to the medium performance state S_M
λ_L	fixed transition rate from the S_M mode to the low-performance state S_L
λ_U	constant transition rate from S_L to unstable state S_U
λ_F	rate reaches a failure in state F
μ_{rep}	repair rate for restore the system to a robust shape
$\mu_{PR} = \mu_{P3R}$	constant transition rate for partial rejuvenation
$\mu_{FR} = \mu_{P4R}$	constant transition rate for full rejuvenation
$r_P = r_{P3}$	Probability transition from S_L to S_{P3R}
$r_F = r_{P4}$	Probability transition from S_U to S_{P4R}
r_{P1}	Probability of partial rejuvenation

r_{P2}	Probability of full rejuvenation
μ_{P1R}	constant transition rate for type I rejuvenation
μ_{P2R}	constant transition rate for type II rejuvenation
C_{rep}	Cost of repair
$C_{PR} = C_{P3R}$	Cost of partial rejuvenation
$C_{FR}= C_{P4R}$	Cost of full rejuvenation
C_{P1R}	Cost of type I rejuvenation
C_{P2R}	Cost of type II rejuvenation

References

[1] M. Grottke, R. Matias, K. &Trivedi, "The fundamentals of software aging," in Proc. IEEE First International Workshop on Software Aging and Rejuvenation, Washington, DC, USA, 2008.

[2] T. Tai, S. N. Chau, L. Alkalaj, H. Hecht, "On-Board Preventive Maintenance: Analysis of Effectiveness and Optimal Duty Period," in Proc. Third Int'l Workshop Object Oriented Real-Time Dependable Systems, CA, 1997.

[3] L. Lei, K. Vaidyanathan, K.S. Trivedi, "An approach for estimation of software aging in a web server," in Proc. International Symp.on Empirical Software Engineering, 2002.

[4] C.M. Kintala, "Software rejuvenation in embedded systems," J. Autom., Lang. , 14: 63–73, 2009.

[5] K. Iwamoto, T. Dohi, H. Okamura, N. Kaio, "Discrete-time cost analysis for a telecommunication billing application with rejuvenation," Comput. Math. Appl., 51(2): 335-344, 2006.

[6] J. Zhao, Y. Wang, G. Ning, K. Trivedi, R. Matias, K. Cai, "A comprehensive approach to optimal software rejuvenation," Perform. Eval., 70(11): 917-933, 2013.

[7] A. Avritzer, E.J. Weyuker, "Monitoring Smoothly Degrading Systems for Increased Dependability," Empirical Software Eng., 2(1): 59-77, 1997.

[8] G. Levitin, L. Xing, H. Huang, "Optimization of partial software rejuvenation policy," Reliab. Eng. Syst. Saf., 182: 63-72, 2019.

[9] D. Cotroneo, R. Natella, R. Pietrantuono and S. Russo, "A survey of software aging and rejuvenation studies," J. Emerg. Technol. Comput. Syst, 10(1), 2014.

[10] G. Rahmani, H. Rashidi, "software availability model based on multilevel software rejuvenation and markov chain," Turk. J. Elec. Eng. & Comp. Sci., 29: 730-744, 2021.

[11] Y. Bao, X. Sun, K. Trivedi, "A workload-based analysis of software aging, and rejuvenation," IEEE Trans. Reliab., 54(3): 541-548, 2005.

[12] S. Garg, A. Pulia, M. Telek, K. Trivedi, "Analysis of software rejuvenation using markov regenerative stochastic petri net," in Proc. Sixth International Symposium on Software Reliability Engineering, 1995.

[13] M. Shereshevsky, J. Crowell, B. Cukic, V. Gandikota, Y. Liu, "Software aging and multifractality of memory resources," in Proc. International Conference on Dependable Systems and Networks, 2003.

[14] D. Cotroneo, R. Natella, R. Pietrantuono, S. Russo, "Software aging and rejuvenation: Where we are and where we are going," in Proc. IEEE Third International Workshop, 2011.

[15] H. Okamura, K. Yamamoto, T. Dohi, "Transient analysis of software rejuvenation policies in virtualized system: Phase-type expansion approach," Qual. Technol. Quant. Manage., 11(3): 336-351, 2014.

[16] G. Ning, J. Zhao, Y. Lou, J. Alonso, R. Matias, et al. "Optimization of two-granularity software rejuvenation policy based on the Markov regenerative process," IEEE Trans. Reliab., 65(4): 1630–1646, 2016.

[17] T. Dohi, H. Okamura, "Dynamic software availability model with rejuvenation," J. Oper. Res. Soc. Jpn., 59(4): 270–290, 2016.

[18] T.A. Nguyen, D. Kim, J. Park, "A comprehensive availability modeling and analysis of a virtualized servers system using stochastic reward nets," Sci. World J., 2014: 1-18, 2014.

[19] M. Torquato, I.M. Umesh, P.J. Maciel, "Models for availability and power consumption evaluation of a private cloud with VMM rejuvenation enabled by VM live migration," J. Supercomput., 74(9): 1–25, 2018.

[20] J.M. Preeti, "Availability analysis of software rejuvenation in active/standby cluster system," Int. J. Ind. Syst. Eng., 19(1): 75–93, 2015.

[21] W. Xie, Y. Hong, K.S. Trivedi, "Analysis of a two-level software rejuvenation policy," Reliab. Eng. Syst. Saf., 87(1): 13-22, 2005.

[22] V.P. Koutras, "Two-level software rejuvenation model with increasing failure rate degradation," Berlin Heidelberg: Springer-Verla, 97: 101– 115, 2011.

[23] Y. Fang, B. Yin, G. Ning, Z. Zheng, K. Cai, "A rejuvenation strategy of two-granularity software based on adaptive control," in Proc. IEEE 22nd Pacific Rim International Symposium on dependable computing (PRDC), Christchurch, 2017.

[24] V.P. Koutras, A.N. Platis, N. Limnios, "Availability and reliability estimation for a system undergoing minimal, perfect and failed rejuvenation," in Proc. IEEE International Conference on Software Reliability Engineering Workshops, Seattle, 2008.

[25] A. Sadek, N. Limnios, "Nonparametric estimation of reliability and survival function for continuoustimefinite Markov processes," J. Stat. Plann. Inference , 133(1): 1–21, 2005.

[26] Y. Lee, H. Kim, "Availability analysis of systems with time-based software rejuvenation," Reliab. Eng. Syst. Saf., 23(2): 201-206, 2019.

[27] Q. Qiu, L. Cui, "Availability analysis for general repairable systems with repair time threshold," Commun. Stat.- Theory Methods, 48(3): 628-64, 2019.

[28] V.P. Koutras, A.N. Platis, "On the performance of software rejuvenation models with multiple degradation levels," Software Qual. J., 28(1): 1-37, 2020.

[29] Y. Huang, C. Kintala, N. Kolettis, N.D. Fulton, "Software rejuvenation: Analysis, module andapplications," in Proc. Twenty-Fifth International Symposium on Fault-Tolerant Computing. Digest of Papers, 1995.

[30] X. Hua, C. Guo, H. Wu, D. Lautner, S. Ren, "Schedulability analysis for real-time task set onresource with performance degradation and dual-level periodic rejuvenations," IEEE Trans. Comput., 66(3): 553-559, 2017.

Biographies

Zahra Rahmani Ghobadi received the B.Sc. degree in Software Engineering from the Shomal University, Amol, Iran, the M.Sc. degree in Software Engineering from the Islamic Azad University, Qazvin, Iran, and she is attending the Ph.D. degree in Software Engineering from the Islamic Azad University, Qazvin, Iran. In 2015, she joined the department of, Computer and Information Technology, the Islamic Azad University, Ramsar, Iran. Her current research interests include software engineering, software quality, and software availability.

Hassan Rashidi is a Professor in Department of Mathematics and Computer Science of Allameh Tabataba'i University. He received the B.Sc. degree in Computer Engineering and M.Sc. degree in Systems Engineering and Planning, both from the Isfahan University of Technology, Iran. He obtained Ph.D. from Computer Science and Electronic System Engineering department of University of Essex, UK. His research interests include software engineering, software testing, and scheduling algorithms. He has published many research papers in International conferences and Journals.

Sasan H. Alizadeh is an Assistant Professor in IRAN Telecommunication Research Center. He received the B.Sc. degree in Computer from the Shiraz University, Shiraz, Iran, the M.Sc. degree in Computer from the Amirkabir University, Tehran, Iran, and He obtained Ph.D. from Computer Science from the Amirkabir University, Tehran, Iran.

How to cite this paper:

Z. Rahmani Ghobadi, H. Rashidi, S.H. Alizadeh, "On multiple objective of software rejuvenation models with several policies," J. Electr. Comput. Eng. Innovations, 10(1): 25-36, 2022.

DOI: 10.22061/JECEI.2021.7905.448

URL: https://jecei.sru.ac.ir/article_1551.html