Turkish Journal of Electrical Engineering & Computer Sciences

http://journals.tubitak.gov.tr/elektrik/

Research Article

Turk J Elec Eng & Comp Sci
(2021) 29: 730 – 744
© TÜBİTAK
doi:10.3906/elk-2003-159

A software availability model based on multilevel software rejuvenation and markov chain

Zahra RAHMANI GHOBADI[1], **Hassan RASHIDI**[2*]
[1]Faculty of Computer and Information Technology Engineering,Qazvin Branch,Islamic Azad University, Qazvin, Iran
[2]Faculty of Statistics, Mathematics and Computer Science, Allameh Tabataba'i University, Tehran, Iran

Received: 27.03.2020 • **Accepted/Published Online:** 01.12.2020 • **Final Version:** 30.03.2021

Abstract: Increasing use of software, rapid and unavoidable changes in the operational environment bring many problems for software engineers. One of these problems is the aging and degradation of software performance. Software rejuvenation is a proactive and preventive approach to counteract software aging. Generally, when software is initiated, amounts of memory are allocated. Then, the body of software is executed for providing a service and when the software is terminated, the allocated memory is released. In this paper, a rejuvenation model based on multilevel software rejuvenation and Markov chain presented. In this model, the system performance as a result of degraded physical memory and memory usage is divided into four equal levels by services. Hence, we offer four types of policies for software rejuvenation. In addition, the system availability is determined, and a cost function for the model is introduced. The cost function includes the time of performing rejuvenation, the number of system services at any time, and the number of rejuvenation actions. To validate the proposed model, a case study in the banking system in Iran has been studied. Due to the differences in the use of the system over time, it is better to perform the four different policies with regard to the use of the system. The numerical results show that the proposed model is convenient for the system so that the costs are reduced per day.

Key words: Software rejuvenation, aging, availability, markov chain,cost function

1. Introduction

Given the rapid growth in modern technologies, industries are becoming increasingly dependent on computers. In terms of performance, these computer systems are affected by hardware and software failures, which in turn can lead to detrimental results for a computer system and even humans. Nowadays, engineers are making great efforts to produce hardware and software that offer a high level of confidence. In the literature around reliability, the aging software has been reported in software with long execution time. This phenomenon intensifies the risk of failure rates or performance degradation and eventually crash failures during run-times [1].

Software aging has been detected in numerous types of systems, such as critical systems, telecommunication switching, and billing software programs [2], Android smartphones[3], virtual machine[4], networked UNIX workstations[5], Openstack cloud computing platform[6], Apache web server[7], spacecraft flight systems [8], and real-time systems[9]. Also, software aging could cause great losses in the safety-critical systems, including the loss of human lives [10].

In the last few years, several researched devoted on this field, including a nonhomogeneous cyclic Markovian chain model [11], software rejuvenation for resources [12], evaluating periodic and rejuvenation

*Correspondence: hrashi@gmail.com

schemes in operational software applications [13], techniques and opportunities for rejuvenation in distributed software systems that are currently aging [14], and effects of time-triggered system rejuvenation policies on the availability of service adopting a queuing model. Numerous other studies have been devoted to system performance [15, 16] including reliability [17–19], reliability analysis of an embedded cluster system [1], and accessibility (See [20–23]). Escheikh in [24, 25] analyzed the concept of power management performability in order to evaluate the effect on performance and energy consumption in virtualized systems. Basically, software aging and rejuvenation are two of the most critical issues in software systems.

Software rejuvenation is a proactive approach when it comes to the reversal of software aging. Software rejuvenation is defined as follows: restarting an application to a clean internal state periodically. It can also be described as the preemptive periodic rollback of applications continuously running in order to prevent potential failures [26]. Traditional strategies for software rejuvenation are classified into time-based and inspection-based [27] . The former approach builds the rejuvenation model, establishing functions between time and cost as well as between workload and cost in an effort to specify the optimal rejuvenation scheduling. The latter approach determines the optimal rejuvenation scheduling through a collection of system data regarding resource usage and performance [28].

Due to the exhaustion of operating system resources, fragmentation, and accumulation of errors, software aging causes problems in all organizations. One of these organizations is banks in which many people are doing many electronic and paper-based transactions over 24 h. These transactions must be managed with more speed, availability, reliability, and security. Additionally, the bank systems are using more efforts to manage these transactions with several types of passwords, such as static and dynamic passwords and recently one-time-password.

In this paper, we propose a novel model for software rejuvenation based on availability optimization through the Markov chain. In this model, the volume of free physical memory is employed as a measure for degradation of resources. When rejuvenation does not take place, the free physical memory of a system with running applications decreases due to memory leaks and other software failures. In our proposed model, rejuvenation is conducted when the volume of free physical memory arrives at certain critical levels. There will be four categories of rejuvenation conducted.

The next sections of this paper are organized as follows: Section 2 intends to review the relevant works. Section 3 describes the problem in detail; Section 4 presents the new method; Section 5 shows the experiments and numerical results. Finally, Section 6 provides the summary and conclusions.

2. 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 [29].

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 [30]. 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[30].

With respect to analysis, the rejuvenation methods are divided into three types namely 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 [31, 32]. 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 [33]. The hybrid-method combines the benefits of model-based and measurement-based approaches. In this paper, we adopted the hybrid model as well [30].

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, nonsafety-critical, and unspecified [34].

Aging indicators have become an essential area of focus in academic studies, since they are highly practical for detecting whenever the system state is susceptible to aging-related failures. In this procedure, aging indicators typically monitor systems during execution. Aging indicators can cover resource usage and performance. In this paper, memory consumption is included as an indicator of aging. It is easy to implement rejuvenation based on aging indicators, but it is unique to specific systems [30]. There are two policies for rejuvenation: application restart (i.e. the whole application is restarted) and OS reboot (i.e. it restarts the OS). In this paper, we have also proposed two policies for rejuvenation to complement those offered in[30].

During the past 20 years, software rejuvenation has been extensively studied with the aim to design rejuvenation policies that optimize system availability, reliability, and performance, mainly in terms of operational cost. In another paper, Koutras and Platis propose to model software systems' overall performance capacity by assigning a performance capacity level at each of the possible states that it can be in, using a continuous-time Markov process. A performance capacity indicator for all possible rejuvenation models incorporating partial, full, or both rejuvenation actions is defined and evaluated in the transient, and the steady-state phase and the impact of various rejuvenation policies on it are further examined [35].

3. Problem description and modeling

In a software rejuvenation technique, as the application efficiency lowers to a certain level, the recovery system initiates, cleans its internal states, restores efficiency and restarts services. In fact, software rejuvenation involves terminating an application gracefully and restarting it immediately at a clean internal state. This technique enhances system availability to a significant degree. During rejuvenation, the system is unavailable, and sometimes rejuvenation may give rise to system downtime. However, rejuvenation scheduling and performance during service idle, the downtime is expected to decrease[36].

Consisting of four states, Figure 1 illustrates the state diagram for software rejuvenation technique and transition rate between states. On the outset, the system is in an appropriately safe state (S_0). When the system encounters performance degradation over time, it will go to the state prone to failure (S_L). In this state, the system is better to arrive in the failure state (F) where a crash occurs, transient to rejuvenation state (R) until a clean internal state is achieved and the system returns to the safe state [36].

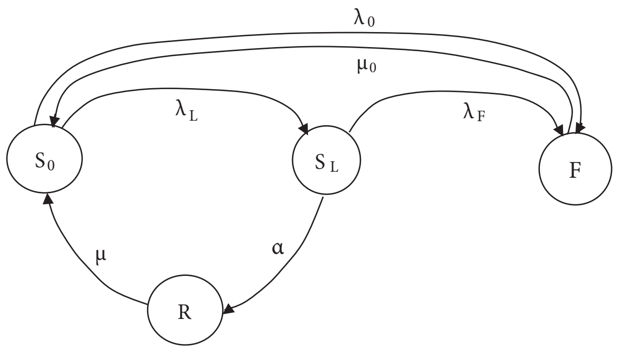

Figure 1. State transition diagram.

As a practical technique, software rejuvenation is conducted on software programs in order to prevent any potential failures and degradations. With regard to downtime, however, the cost can be immense. When running in rejuvenation state (R) and failure state (F), the system is unavailable. Therefore, the availability of the system is determined by Equation1:

$$A(\infty) = 1 - (P(R) + P(F)). \tag{1}$$

The transition rates are represented by λ_1, λ_2, λ_3, μ_1, μ_2, and α where λ_1 is the transition rate from state S_0 to state S_L, λ_2 is the transition rate from state S_L to F, and λ_3 represents the transition rate from state F to state S_0. When the system is kept down to complete rejuvenation, it implies that the system will not be available for a certain period of time.

4. The newly proposed model and cost function

In the new model, we insert l_i, where i = L, M, H, U, and each i represents one free memory level (Figure 2). There are four types of policies for software rejuvenation provided in our model; therefore, we have four levels for free memory. The critical threshold for free memory is represented by l_U. When free memory due to leakage is lower than l_U, system crash will occur and repair operation is initiated to retain the primary level of free memory. When memory reaches l_U, the system will return to low-efficiency, medium-efficiency, or high-efficiency states through rejuvenation. In this procedure, the system rebounds to its initial state, where it runs in a healthy, robust mode.

The efficiency of any system decreases over a period of constant operation, leading to instability. This is where rejuvenation comes into play. However, regarding a rejuvenation process that actually takes place, many questions need to be answered: What is the best time for rejuvenation? How much rejuvenation is required?, and so on.

We proposed four types of rejuvenation policies and their availability are compared so that the most suitable policy can be adopted. These policies are listed below:

-Rejuvenation policy-I: In this policy, system services are partially restarted. Some services might still take up space after being terminated. Instead of halting the entire system, such services are forcefully closed, and the related memory is freed up. As a result, the system state is changed from unstable to low efficiency. This type of rejuvenation does not result in significant improvements.

-Rejuvenation policy-II: This policy of rejuvenation is performed when the system is in an unstable state. In this case, all terminated and running services are stopped, and the system's state changes into medium efficiency.

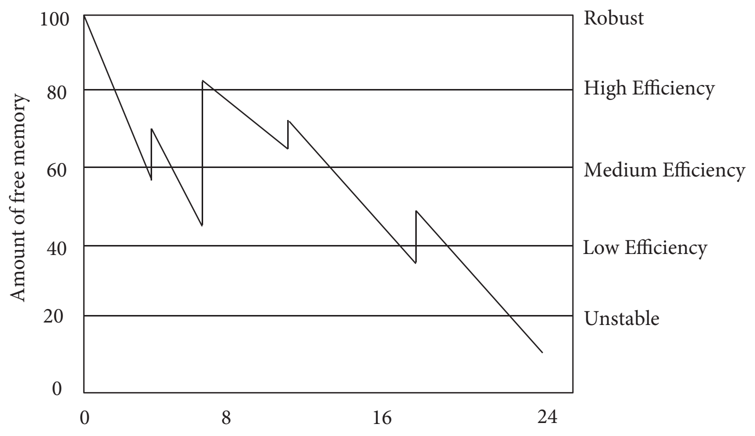

Figure 2. Amount of free memory.

-Rejuvenation policy-III: When the system is in the unstable state, performing a general restart – i.e. stopping the terminated services, the running services, and the operating system – will free up a considerable amount of space, and the system will go to high-efficiency state.

-Rejuvenation policy-IV: This policy of rejuvenation is performed at the level of the physical machine and is the most typical policy of rejuvenation. It simply turns the system off and then on again. It is the costliest type of rejuvenation, but it will transfer the system to a healthy and robust state. Figure 3 shows the four rejuvenation types. The proposed Markov model has ten states. Figure 3 shows the ten states and the transitions between them. L_U represents the rejuvenation threshold. A specific type of rejuvenation is selected based on the conditions. As Figure 3 shows, rejuvenation policy-I restores state R_L and rejuvenation policy-II restores state R_M. Rejuvenation policy-III restores state R_H, and finally, rejuvenation policy-IV restores state R.

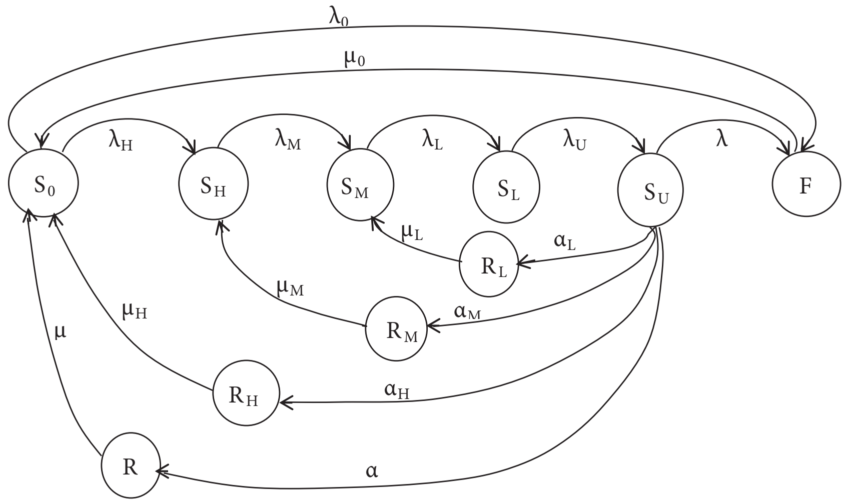

Figure 3. State transition diagram.

Markov state space at time t is specified by Equation 2:

$$X(t) = [Si; i = 0, H, M, L, U] \cup [(Rj); j = 0, H, M, L] \cup [F]. \quad (2)$$

According to [37], let $S_j(t)$ be transition probability function of continuous-time Markov process and the q_j be transition rate. Kolmogorov forward equation is defined by Equation 3:

$$\frac{dp_{ij}(t)}{dt} = \sum_{k=0}^{N} s_{ik}(t) q_{kj}, i, j = 0, L, M, H. \tag{3}$$

By letting $S(t)$ be the matrix of transition probability function and Q be the matrix of transition rate function, 3 is expressed in matrix format equation 4:

$$S'(t) = S(t)Q; S(0) = I \tag{4}$$

where, the I is the unit matrix and the Q is a square matrix.

In the software rejuvenation model depicted in Figure 3, the Markov chain has ten states. The transition rate matrix for the continuous-time Markov process Z can be easily derived as Matrix 5 according to the Markov's rules. In the Matrix Q, the beginning states are in the columns on the left-side, while the transition states are in the rows of the right-side.

Because of memory leakage, the system entails five degraded states called low, medium, and high-efficiency states as well as unstable and failure states. The transition rate of state to other states is exponentially distributed with rates λ_H, λ_M, λ_L, λ_U, and λ_F, respectively. In addition,it is assumed that the lifetime and repair time of the active state is distributed exponentially with failure rate λ_0 and repair rate μ_0. The repair rate for all states is represented by μ_k, since four types of rejuvenation are proposed, then $\mu_k = \mu_L, \mu_M, \mu_H, \mu$, and rejuvenation rates shown by α_k, and, since four types of rejuvenation are proposed, then $\alpha_k = \alpha_L, \alpha_M, \alpha_H, \alpha$.

$$\begin{array}{c} \\ S_0 \\ S_H \\ S_M \\ S_L \\ S_U \\ F \\ R_L \\ R_M \\ R_H \\ R \end{array} \begin{array}{c} \begin{array}{cccccccccc} S_0 & S_H & S_M & S_L & S_U & F & R_L & R_M & R_H & R \end{array} \\ \begin{pmatrix} d_1 & \lambda_H & 0 & 0 & 0 & \lambda_0 & 0 & 0 & 0 & 0 \\ 0 & d_2 & \lambda_M & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & d_3 & \lambda_L & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & d_4 & \lambda_U & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & d_5 & \lambda_F & \alpha_L & \alpha_M & \alpha_H & \alpha \\ \mu_0 & 0 & 0 & 0 & 0 & d_6 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \mu_L & 0 & 0 & d_7 & 0 & 0 & 0 \\ 0 & 0 & \mu_M & 0 & 0 & 0 & 0 & d_8 & 0 & 0 \\ 0 & \mu_H & 0 & 0 & 0 & 0 & 0 & 0 & d_9 & 0 \\ \mu & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & d_{10} \end{pmatrix} \end{array} \tag{5}$$

$$\begin{aligned} & d_1 = -(\lambda_0 + \lambda_H), d_2 = -\lambda_M, d_3 = -\lambda_L, d_4 = -\lambda_U, d_5 = -(\lambda_F + \alpha_L + \alpha_M + \alpha_H + \alpha) \\ & d_6 = -\mu_0, d_7 = -\mu_L, d_8 = -\mu_M, d_9 = -\mu_H, d_{10} = -\mu \end{aligned} \tag{6}$$

$$\begin{aligned} & (\lambda_0 + \lambda_H)S_0 = \mu.R + \mu_0.F; \lambda_M.S_H = \lambda_H.S_0 + \mu_H.R_H; \lambda_L.S_M = \lambda_M.S_H + \mu_M.R_M \\ & \lambda_U.S_L = \lambda_L.S_M + \mu_L.R_L; (\lambda_F + \alpha_L + \alpha_M + \alpha_H + \alpha)S_U = \lambda_U.S_L; \mu_0.F = \lambda_F.S_U + \lambda_0.S_0 \\ & \mu.R = \alpha.S_U; \mu_H.R_H = \alpha_H.S_U; \mu_M.R_M = \alpha_M.S_U; \mu_L.R_L = \alpha_L.S_U; \mu_L.R_L = \alpha_L.S_U \\ & S_0 + S_H + S_M + S_L + S_U + R + R_H + R_M + R_L + F = 1 \end{aligned} \tag{7}$$

By solving the above formulas, i.e. Equation 7, recursively, the probability state is obtained by Equation 8, as follows:

$$\begin{aligned}
&S_0 = \frac{\alpha.S_U + \mu_0.F}{\lambda_H + \lambda_0}, S_H = \frac{\alpha_H.S_U + \lambda_H.S_0}{\lambda_M}, S_M = \frac{\alpha_M.S_U + \lambda_M.S_H}{\lambda_L}, S_L = \frac{\alpha_L.S_U + \lambda_L.S_M}{\lambda_U}, \\
&S_U = \frac{\lambda_0.S_0 - \mu_0.F}{\lambda_F}, R = \frac{\alpha}{\mu}S_U, R_H = \frac{\alpha_H}{\mu_H}S_U, R_M = \frac{\alpha_M}{\mu_M}S_U, R_L = \frac{\alpha_L}{\mu_L}S_U, F = \frac{(\lambda_0 + \lambda_H)S_0 - \mu.R}{\mu_0}
\end{aligned} \tag{8}$$

The system is unavailable in rejuvenation states and failure states. Then, availability is determined by Equation 9 as follows:

$$A(\infty) = 1 - (R + R_H + R_M + R_L + F). \tag{9}$$

It is assumed that the process is initially in state S_0. At S_0, all the entire memory is free; therefore,

$$PS_0(0) = 1, PS_H(0) = 0, PS_M(0) = 0, PS_L(0) = 0, PS_U(0) = 0, F(0) = 0. \tag{10}$$

The availability function and failure probability are determined by Equation 11 as follows:

$$\begin{aligned}
&A_F(t) = S_0 + S_H + S_M + S_L \\
&f(t) = \lambda_0.S_0 + \lambda_F.S_U
\end{aligned} \tag{11}$$

For cost function, we consider a banking system that can run and serve continuously during each day. It is observed, the workload on the system reaches its peak at specific times of some days. This causes degradation of system resources, which results in system aging, and eventually, failure. It is clear that rejuvenation prevents failure; however, it increases the costs due to system unavailability during the rejuvenation process. The rejuvenation rates should be functions of time because the number of transactions that are carried out in the system and that are affected by rejuvenation policies varies from time to time. Obviously, the number of transactions is low during late-night hours. Therefore, the 24 h of the day were divided into four periods: the first period from 1:00 to 07:00 a.m. (1 to 7), the second period from 07:00 a.m. to 1:00 p.m. (7 to 13), the third period from 1:00 to 7:00 p.m. (13 to 19), and finally, the fourth period from 7:00 p.m. to 1:00 a.m. of the next day (19 to midnight+1). The workload is minimum in the first period, maximum in the second period, and average in the other two periods.

In the newly proposed model, the amount of free physical memory is used as a measure of resources that degrade. The rejuvenation actions, performed to counteract aging, are triggered based on measured free memory levels, as shown in Figure 3 four different types of rejuvenation are proposed in order to improve the performance of this model. To demonstrate that rejuvenation rates are functions of time, these rates are defined as $\mu_L(n)$, $\mu_M(n)$, $\mu_H(n)$, $\mu(n)$, where n represents time. This study attempted to derive a cost function so as to model an effective rejuvenation policy with optimum cost. To obtain the cost function, the steady-state probabilities are needed. According to Markov chain rules, the steady-state probabilities at the beginning of the first period are estimated through Equation 12.

$$sQ = 0, \sum_{i=1}^{n} s_i = 1 \tag{12}$$

where $Q = [q_{ij}]$ is a n*n matrix that contains transition rates from state i to state j and n is the number of states. To compute downtime costs, we should consider different components of cost according to system states.

Suppose C_f is the cost of failure per time unit and $C_{R,j}$ is the rejuvenation cost for level j, where j=H, M, L. It is obvious that $C_F > C_{R,j}$ and $C_F > \sum_{j=R_L}^{R} C_{R,j}$ and $C_{RL} < C_{RM} < C_{RH} < C_R$. The downtime cost function is calculated as follows:

$$DCF = [C_F.F + C_{R_L}.R_L + C_{R_M}.R_M + C_{R_H}.R_H + C_R.R] \tag{13}$$

As mentioned in Section 3, the steady-state probabilities that are involved in the problem are F, R_L, R_M, and R_H, and the cost function is obtained from the Equation14. The downtime cost function for each period is determined as follows:

$$\begin{aligned} DCF_1 &= [C_f.F_1 + C_{R_L}.R_{1,L} + C_{R_M}.R_{1,M} + C_{R_H}.R_{1,H} + C_R.R_1] \\ DCF_2 &= [C_f.F_2 + C_{R_L}.R_{2,L} + C_{R_M}.R_{2,M} + C_{R_H}.R_{2,H} + C_R.R_2] \\ DCF_3 &= [C_f.F_3 + C_{R_L}.R_{3,L} + C_{R_M}.R_{3,M} + C_{R_H}.R_{3,H} + C_R.R_3] \\ DCF_4 &= [C_f.F_4 + C_{R_L}.R_{4,L} + C_{R_M}.R_{4,M} + C_{R_H}.R_{4,H} + C_R.R_4] \end{aligned} \tag{14}$$

Where DCF_1 denotes the cost function regarding the first period, and DCF_2 , DCF_3 , and DCF_4 are the cost functions for the second, third, and fourth periods, respectively. Therefore, F, $R_{f,L}$, $R_{f,M}$, $R_{f,H}$, and R_f refer to the steady-state probabilities for repair, rejuvenation 1, rejuvenation 2, rejuvenation 3, and rejuvenation 4 at the r-th period, where r =1, 2, 3, 4. In order to determine the optimal cost-effective rejuvenation policy, the rejuvenation rates µ1, µ2, µ3, and µ4 need to be explored. For this purpose, the system load in terms of time has to be covered. It is obvious that rejuvenation prevents failure but it is also costly since the system does not provide any services during rejuvenation. Therefore, an increase in rejuvenation rates increases the cost. The proposed strategy for this case study is to use the method that frees up the maximum amount of memory (rejuvenation Type 4) when the system load is at its minimum level, i.e. the first period is from 1:00 to 07:00 a.m. The long system unavailability in this method can be ignored since the workload is also insignificant. When there is a significant workload in the second period. i.e. 7:00 a.m. to 1:00 p.m. (7 to 13), a method should be used that results in minimum unavailability, which is rejuvenation Type 1. Similarly, during the third period, i.e. 1:00 to 7:00 p.m. (13 to 19), there is a partial decrease in the number of services. In this period, the most appropriate method is rejuvenation Type 3. In the fourth period, i.e. from 7:00 p.m. to 1:00 a.m. of the next day (19 to midnight+1), the number of services will be even less than that in the third period, which indicates that the most appropriate method is rejuvenation Type 4. Considering the workload of the system, it is clear that the more the number of services, the more the cost of the model. A new cost function is formulated below:

$$DCF = k(\mu_L(n) + \mu_M(n) + \mu_H(n) + \mu(n) + \beta(t)) \tag{15}$$

Where $k(\mu_L(n) + \mu_M(n) + \mu_H(n) + \mu(n))$ denotes that the total cost is directly related to the rejuvenation rate (k is a positive constant) and $\beta(t)$, t=1,2, …,24, shows the number of transactions in the system at time t. n is the time period (n=1,2,3,4). Therefore, the total cost is obtained through the following cost function:

$$DCF(t) = \begin{cases} DCF_1 + k(\mu_L(1) + \mu_M(1) + \mu_H(1) + \mu(1) + \beta(t)) \\ DCF_2 + k(\mu_L(2) + \mu_M(2) + \mu_H(2) + \mu(2) + \beta(t)) \\ DCF_3 + k(\mu_L(3) + \mu_M(3) + \mu_H(3) + \mu(3) + \beta(t)) \\ DCF_4 + k(\mu_L(4) + \mu_M(4) + \mu_H(4) + \mu(4) + \beta(t)) \end{cases} \tag{16}$$

5. Experiments and Results

Software rejuvenation is a useful process that prevents software degradation and failure in software systems. However, it is a significantly costly process. A system becomes unavailable during rejuvenation, which implies that the system goes unavailable for a given period of time. On the other hand, ignoring rejuvenation will cause degradation and eventually failure, which in turn imposes significant costs on the system. Therefore, it is necessary that rejuvenation policies be carefully examined before deciding on the execution time of rejuvenation and the number of rejuvenations.

In this section, the results of the experiments are presented. A comparison has been made between downtime overhead when only one rejuvenation type is used and when the four proposed rejuvenation methods are used. This application is implemented in a specific banking area and the coding was done in MATLAB (MathWorks, Inc., Natick, MA, USA). The system availability is calculated by considering the rate at which the system can be found in each state. The transition rate between the states is shown in Table 1 and Table 2, μ_1 is rate of repair and, μ_2 is rate of rejuvenation (per hour). In order to calculate the performance indices, we set λ=1 and α=1. The system availability values for each case is displayed in Figure 4 and Figure 5.

Table 1. Rejuvenation and repair rates values for the first method.

Exp1	Exp2	Exp3	Exp4	Exp5	Exp6	Exp7	Exp8
μ1=0.5	μ1=0.75	μ1=1	μ1=1.25	μ1=1.5	μ1=1.75	μ1=2	μ1=2.25
μ2=10	μ2=11	μ2=12	μ2=13	μ2=14	μ2=15	μ2=16	μ2=17

Table 2. Rejuvenation and repair rates values for the proposed method.

Exp1	Exp2	Exp3	Exp4	Exp5	Exp6	Exp7	Exp8
μ0 = 1	μ0 = 1	μ0 = 1	μ0 = 1	μ0 = 1	μ0 = 1	μ0 = 1	μ0 = 1
μL = 10	μL = 11	μL = 12	μL = 13	μL = 14	μL = 15	μL = 16	μL=17
μM = 9	μM = 10	μM = 11	μM = 12	μM = 13	μM = 14	μM = 15	μM=16
μH = 8	μH = 9	μH = 10	μH = 11	μH = 12	μH = 13	μH = 14	μH=15
μ = 4	μ = 5	μ = 6	μ = 7	μ = 8	μ = 9	μ = 10	μ = 11

Figure 4 displays the variation of the availability versus the variation of the different rejuvenation rate μ_1 and μ_2 . Availability increase when the value of μ_1 and μ_2 grow.

Figure 5 displays the variation of the availability versus those of the different rejuvenation rates μ_1, μ_2, μ_3 and μ_4. Availability is increased when the value of the rejuvenation rate increases. As can be seen in Figure 4 and Figure 5, there is a difference between availability when only one rejuvenation method is used and when the four newly proposed rejuvenation types are used. In the following, the new method has been further discussed.

In Figure 6A, rejuvenation rate μ_L changes and rejuvenation rates μ_M, μ_H and μ are set to be constant. Hence, the availability value refers only to a function of the rejuvenation rate μ_L only. In other words, the availability increases when the value of μ_L grows. Moreover, Figure 6B indicates that rejuvenation rate μM changes, while rejuvenation rates μ_L , μ_H, and μ are set to be constant. In this case, availability increases but such increases are less than the previous one. In Figure 6C and Figure 6D, the availability for the different rates of third and fourth rejuvenation types has been obtained, according to the Figs, the availability first

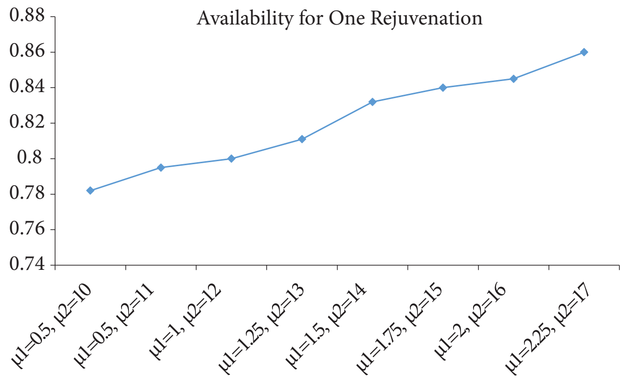

Figure 4. Availability for the first method.

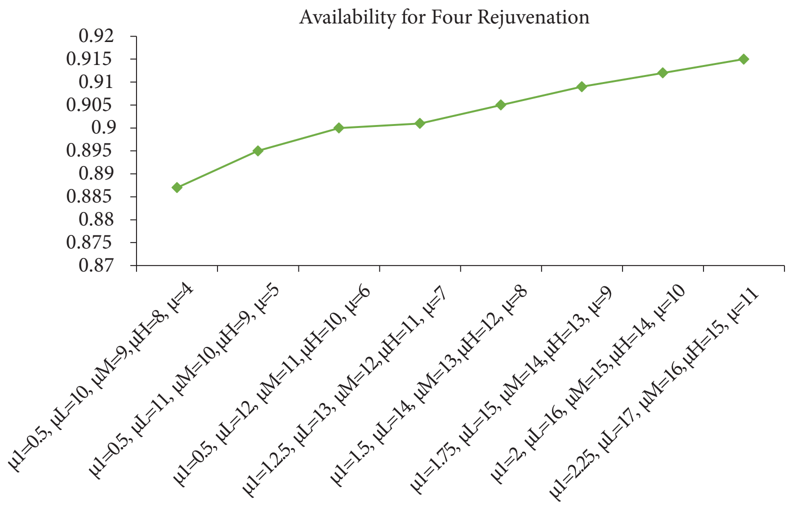

Figure 5. Availability for the proposed method.

increases, and then decreases. That is because the system is not available during rejuvenation, and the time of unavailability for third and fourth rejuvenation types is long, thus reducing availability. When rejuvenation is performed at large μ_L andμ_M rates, the rejuvenation procedure of this category takes place more often and the availability increases. Therefore, it is preferred to have an higher availability by conducting rejuvenation with small μ_H and μ rates.

In order to examine the proposed model, we consider one of the biggest banks in Iran (known as Keshavarzi bank, Agriculture Bank in English). This bank is a major Iranian banking establishment offering retail and commercial services for farmers and industrial agriculture. Currently, the bank serves as the only specialized financial institution in the agricultural sector that holds over 1800 branches nationwide and finances nearly 70% of the Iranian agricultural sector. While established in Tehran, the bank operates throughout the nation with over 16,000 employees and 1800 branches. The bank currently specializes in providing credit facilities for agricultural development and other rural development activities. Before applying the model, we had to obtain a model for the operational environment in this bank. The number of services in each period at this bank are shown in Table 3. The number of services in the first period is very small and is at its lowest level. This number increases significantly in the second period and reaches its maximum level. In the third period, the number of services decreases, and in the fourth period, it decreases even more. We did a regression to obtain the number of services at the four periods of time and obtained Equation 17:

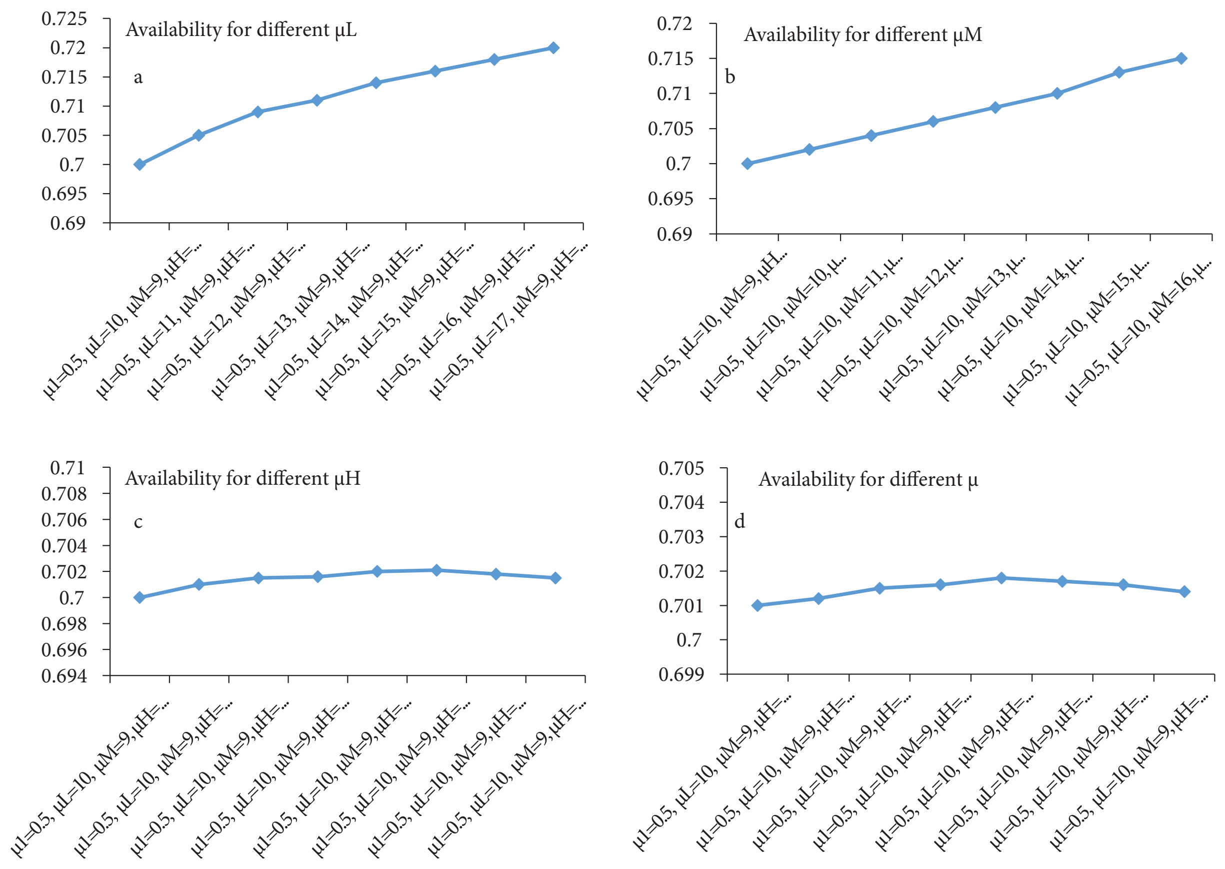

Figure 6. Availability for different rates of rejuvenation, Fig.A. (Different μ_L), Fig.B. (Different μ_M), Fig.C. (Different μ_H), Fig.D. (Different μ).

Table 3. Number of services at different time of day.

Time	Num of service	Time	Num of service	Time	Num of service	Time	Num of service
t = 1	71439	t = 7	230321	t = 13	100001	t = 19	97599
t = 2	61584	t = 8	260000	t = 14	100016	t = 20	95904
t = 3	49375	t = 9	294481	t = 15	100081	t = 21	93439
t = 4	34464	t = 10	334256	t = 16	100256	t = 22	90000
t = 5	16479	t = 11	379841	t = 17	100625	t = 23	85359
t = 6	4976	t = 12	431776	t = 18	101269	t = 24	79264

$$\beta(t) = \begin{cases} N-(12+t)^4, 1 \le t < 7 \\ N+(12+t)^4, 7 \le t < 13 \\ N+(12-t)^4, 13 \le t < 19 \\ N-(12-t)^4, 19 \le t < 1(Midnight+1) \end{cases} \tag{17}$$

, where the N represents the average number of transactions in the bank (N=100,000) and t is the period over 24 h of each day. In practice, the cost of each period is assumed to corresponding values in Table 4 according to the values obtained in[12].

Table 4. Costs for rejuvenation and repair.

Cost	C_f	C_R	$C_{R,H}$	$C_{R,M}$	$C_{R,L}$
Value	500	100	50	30	10

According to the Equation 16 and 17, the total cost of the model is calculated through the following equation:

$$DCF = \begin{cases} 180600 - (12+t)^4, 1 \leq t < 7 \\ 152300 + (12+t)^4, 7 \leq t < 13 \\ 194560 + (12-t)^4, 13 \leq t < 19 \\ 196250 - (12-t)^4, 19 \leq t < 1(Midnight + 1) \end{cases} \tag{18}$$

The proposed rejuvenation policy, in this case, is to perform rejuvenation Type-III and Type-IV in the first period (1:00–7:00 a.m.), rejuvenation Type-I in the second period (7:00 a.m. to 1:00 p.m.), rejuvenation Type-I and Type-II in both of third period (1:00–7:00 p.m.) and fourth period (7:00 p.m. to 1:00 a.m.). In the first period (1:00–7:00 a.m.), due to the low number of transactions, the cost function for each of the 4 rejuvenation operations is approximately the same, as shown in the Figure 7A. Therefore, it is more appropriate to perform Type III and IV rejuvenation to create more free space.

The second period (07:00 a.m. to 1:00 p.m.), due to the high number of transactions, the cost function for the first type of rejuvenation is lower than all other transactions, so it is suggested that the rejuvenation of the first type be repeated frequently in order to avoid the cost of system failure. As can be seen in the Figure 7B. The third period (1:00–7:00 p.m.), the number of transactions is more balanced, as shown in the Figure 7C, and the cost function for performing the first and second type rejuvenation is lower, so it is recommended to perform the first and second type rejuvenation at this time. The fourth period (7:00 p.m. to 01:00 a.m. of the next day), The number of transactions is reduced, as shown in the Figure 7D, the cost function for performing the first and second type rejuvenation is lower than others, so it is suggested to perform the first and second type rejuvenation in this period.

From the behavior of the cost function, the optimization policy is clarified (i.e., performing low-cost short-run rejuvenation in the second period, costly time-consuming rejuvenation in the first period, and rejuvenations Type-I and Type-II in the third and fourth periods), results in a normal cost for all periods of the day.

6. Summary and conclusion

One of the key challenges in software engineering involves software aging, which may lead to failure. To respond to this challenge, software rejuvenation is used. This paper modeled a special type of software rejuvenation, which consists of performing software rejuvenation of ”partial restart” type at four different degradation levels, while defining a cost function based on rejuvenation rates. Moreover, a case study is presented and the outcome of the cost policy was that rejuvenation should be performed with regard to different times during a day. For example, the first type of rejuvenation has a short run-time and is more suitable for the second period, in which the workload is maximum and system availability is critically important. The experimental and numerical results show that the proposed model is convenient for the banking system so that the costs are reduced per day. Because of this effective result, the applicability of the proposed model to other industrial cases can be performed in future research.

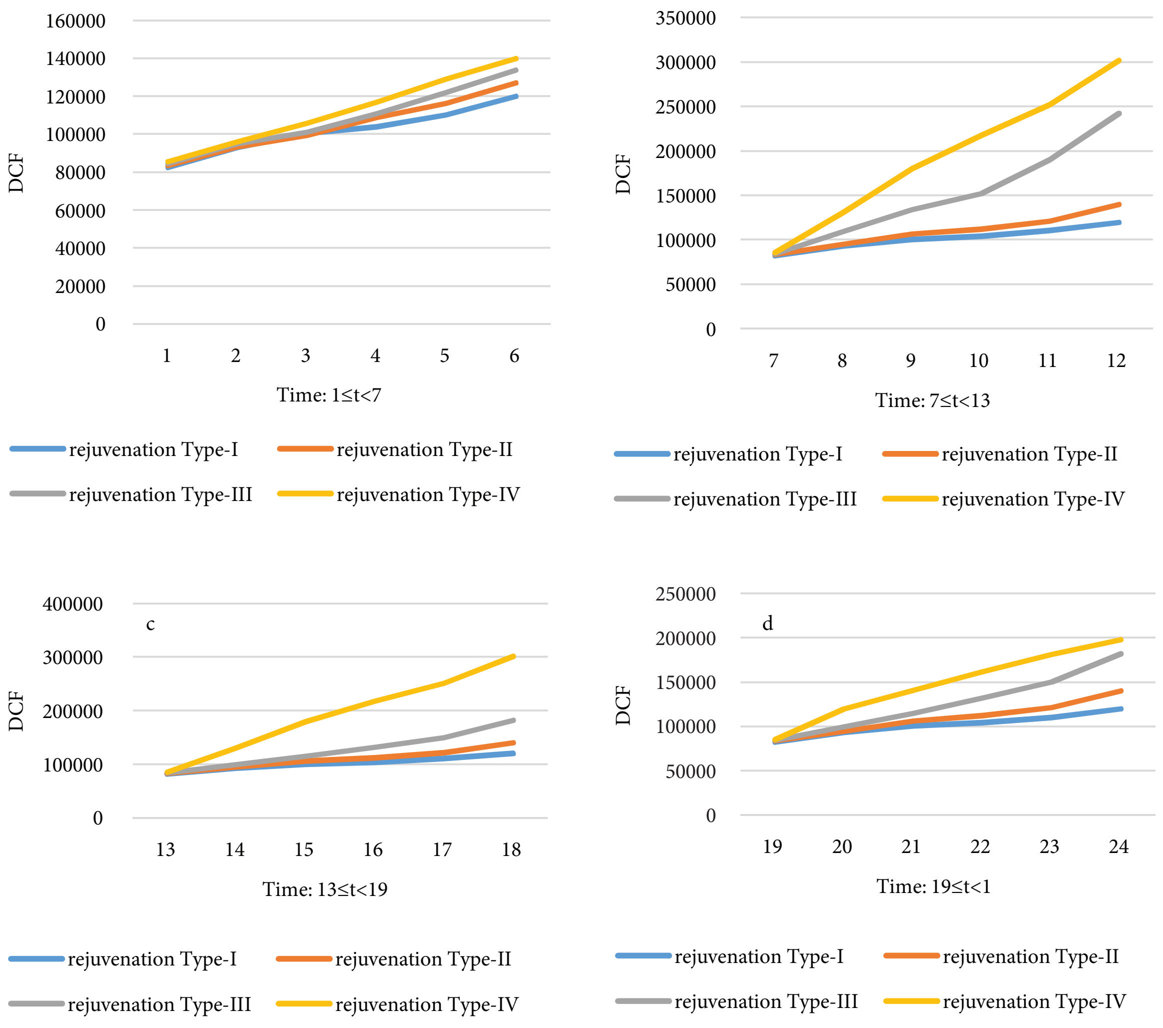

Figure 7. Downtime cost for different rejuvenation rate in different period. Fig.A.(first period), Fig.B.(second period), Fig.C. (third period), Fig.D. (fourth period).

References

[1] Jain M, Manjula T, Gulati R. Software rejuvenation policies for cluster system. The National Academy of Sciences 2016; 86 (3): 339-346. doi: 10.1007/s40010-016-0273-1.

[2] Jifang W, Donghua G, Qing E, Zhenyu H. Research on rejuvenation of software. United States of America Journal of Computational and Theoretical Nanoscience 2016; 13 (5): 3366-3373. doi: 10.1166/jctn.2016.5000.

[3] Qiao Y, Zheng Zh, Fang FQ, Qin F, Trivedi KS et al. Two-level rejuvenation for Android Smartphones and its optimization. IEEE Transactions on Reliability 2019; 68 (2): 633-652. doi: 10.1109/TR.2018.2881306.

[4] Torquato M, Umesh IM, Maciel P. Models for availability and power consumption evaluation of a private cloud with VMM rejuvenation. The Journal of Supercomputing 2018; 74 (9): 4817-4841. doi: 10.1007/s11227-018-2485-4.

[5] Salfner F, Wolter K. Analysis of service availability for time-trigged rejuvenation policies. The Journal of Systems and Software 2010; 83 (9): 1579-1590. doi: 10.1016/j.jss.2010.05.022.

[6] Melo C, Araujo J, Alves V, Maciel P. Investigation of Software aging effects on the OpenStack cloud computing platform. JSW journal 2017; 12(2): 125-137. doi: 10.17706/jsw.12.2.125-137.

[7] Okamura H, Dohi T. Comprehensive evaluation of a periodic check pointing and rejuvenation schemes in operational software system. The Journal of Systems and Software 2010; 83 (1): 1591-1604. doi: 10.1016/j.jss.2009.06.058.

[8] Avritzer A, Cole R, Weyuker E. Methods and opportunities for rejuvenation in aging distributed software systems. The Journal of Systems and Software 2010; 83 (9): 1568-1578. doi: 10.1016/j.jss.2010.05.026

[9] Levitin G, Xing L, Huang HZh. Optimization of partial software rejuvenation policy. Reliability Engineering and System Safety 2019; 188 (1): 289-296. doi: 10.1016/j.ress.2019.03.011.

[10] Koutras VP, Agapios N, George A, Gravvanis GA. On the optimization of free resources using non-homogeneous Markov chain software rejuvenation model. Reliability Engineering and System Safety 2007; 92 (12): 1724-1732. doi: 10.1016/j.ress.2006.09.017.

[11] Grottke M, Li L, Vaidyanathan K, Trivedi KS. Analysis of software aging in a web server. IEEE Transactions on Reliability 2006; 55 (3): 411-420. doi: 10.1109/TR.2006.879609.

[12] Koutras VP, Platis AN, George A, Gravvanis GA. Software rejuvenation for resource optimization based on explicit approximate inverse preconditiong. Applied Mathematics and Computation 2007; 189 (1): 163-177. doi: 10.1016/j.amc.2006.11.056.

[13] Hong Y, Chen D, Li L, Trivedi KS. Closed loop design for software rejuvenation. In: Proceedings of the Workshop on Self-healing, Adaptive and Self-managed Systems; 2002. pp. 56-68.

[14] Tai A, Chau S, Alkalaj L, Hect H. On-board preventive maintenance: a design-oriented analytic study for long-life applications. Performance Evaluation 1999; 35 (5): 215-232. doi: 10.1016/S0166-5316(99)00006-1.

[15] Barlow RE, Hunter LC. Reliability analysis of a one-unit system. Operational Research 1961; 9 (2): 200-208. doi:10.1287/opre.9.2.200

[16] Esary JD, Ziehms H. Reliability analysis of phased missions. Proceedings of the Conference on Reliability and Fault Tree Analysis (SIAM); 1975. pp. 213-236.

[17] Cui LR, Huang JB, Li Y. Degradation models with Wiener diffusion processes under calibrations. IEEE Transactions on Reliability 2016; 65 (2): 615-623. doi: 10.1109/TR.2015.2484075.

[18] Garg S, Puliafito A, Telek M, Trivedi KS. Analysis of preventive maintenance in transactions based software systems. IEEE Transactions on Computers 1998; 47 (1): 96-107. doi: 10.1109/12.656092.

[19] Garro A, Groß J. Reliability analysis of an Attitude determination and control system (ADCS) through the RAMSAS method. Journal of Computational Science 2014; 5 (3): 439-449. doi: 10.1016/j.jocs.2013.06.003.

[20] Ekyay BC, Özekici S. Mean time to failure and availability of semi-Markov missions with maximal repair. European Journal of Operational Research 2010; 207 (3): 1442-1454. doi: 10.1016/j.ejor.2010.07.019.

[21] Lee Y, Kim H. Availability analysis of systems with time-based software rejuvenation. Reliability Engineering and System Safety 2019; 23 (2): 201-206. doi:10.6109/jkiice.2019.23.2.201

[22] Rahmani Ghobadi Z, Rashidi H. Availability analysis and improvement with software rejuvenation. Third International Conference on Contemporary Issues in Computer and Information Sciences; 2012. pp. 213-218.

[23] Rahmani Ghobadi Z, Rashidi H. software rejuvenation technique-an improvement in applications with multiple versions. Computers and Intelligent Systems Journal 2010; 2(1): 22-26.

[24] Escheikh M, Tayachi Z, Barkaoui K. Workload-dependent software aging impact on performance and energy consumption in server virtualized systems. 27th International Symposium on Software Reliability; 2016. pp. 111-118.

[25] Escheikh M, Barkaoui K, Jouini H. Versatile workload-aware power management performability analysis of server virtualized systems. Journal of Systems and Software 2017; 125 (1): 365-379. doi: 10.1016/j.jss.2016.12.037.

[26] Avritzer A, Weyuk J. Monitoring smoothly degrading systems for increased dependability. Empirical Software Engineering Journal 1997; 2 (1): 59-77. doi: 10.1023/A:1009794200077

[27] Fang Y, Yin B, Ning G, Zheng Zh, Cai K. A rejuvenation strategy of two-granularity software based on adaptive control. IEEE 22nd Pacific RimInternational Symposium on Dependable Computing; 2017. pp. 104-109.

[28] Dang W, Zeng J. Optimization of software rejuvenation policy based on state-control-limit. International Journal of Performability Engineering 2018; 14 (2): 210-222. doi: 10.23940/ijpe.18.02.p3.210222.

[29] Grottke M, Matias R, Trivedi K. The fundamentals of software aging. IEEE International Conference on Software Reliability Engineering Workshops; 2008. pp. 1-6.

[30] Cotroneo D, Natella R, Pietrantuono R, Russo S. A survey of software aging and rejuvenation studies. ACM Journal on Emerging Technologies in Computing Systems 2014; 10 (1): 1-34. doi: 10.1145/2539117

[31] Bao Y, Sun X, Trivedi K. A workload-based analysis of software aging, and rejuvenation. IEEE Transactions on Reliability 2005; 54 (3): 541-548. doi: 10.1109/TR.2005.853442

[32] Garg S, Puliafito A, Telek M, Trivedi KS. Analysis of software rejuvenation using markov regenerative stochastic petri net. Sixth International Symposium on Software Reliability Engineering; 1995. pp. 180-187.

[33] Shereshevsky M, Crowell J, Cukic B, Gandikota V, Liu Y. Software aging and multifractality of memory resources. International Conference on Dependable Systems and Networks; 2003. pp. 721-730.

[34] Cotroneo D, Natella R, Pietrantuono R, Russo S. Software aging and rejuvenation: Where we are and where we are going. IEEE Third International Workshop 2011. pp. 1-6.

[35] Koutras V.P, Platis A.N. On the performance of software rejuvenation models with multiple degradation levels. Software Quality Journal; 2020 28 (1): 135-171.

[36] Huang Y, Kintala C, Kolettis N, Fulton N. Software rejuvenation: analysis, module and application. Twenty-Fifth International Symposium on Fault-Tolerant Computing; 1995. pp. 381-390.

[37] Yong Q, Haining M, Di H, Ying CH. A Study on software rejuvenation model of application server cluster in two-dimension state space using Markov process. Information Technology Journal; 2008 7 (1): 98-104.