Journal of Advances in Computer Research
Quarterly ISSN: 2008-6148
Sari Branch, Islamic Azad University, Sari, I.R.Iran
(Vol. 11, No. 2, May 2020), Pages: 59-69
www.jacr.iausari.ac.ir

A Model for Software Rejuvenation Based On Availability Optimization

Zahra Rahmani Ghobadi[1], Hasan Rashidi✉[2], Sasan H.Alizadeh[3]

1) Faculty of Computer and Information Technology Engineering, Qazvin Branch, Islamic Azad University, Qazvin, Iran
2) Departments of Mathematics and Computer Science, Allameh Tabataba'i University, Tehran, Iran
3) ICT Research Institute (Iran Telecommunication Research Center),Tehran,Iran

z.rahmani@qiau.ac.ir; hrashi@atu.ac.ir; s.alizadeh@itrc.ac.ir

Received: 2020/01/26; Accepted: 2020/05/22

Abstract

In this paper, we focus on the availability of software systems. Software systems with long-running execution may lead to software aging. This phenomenon affects the performance of software system and may eventually cause them to crash or failures. To counteract the phenomenon aging, software rejuvenation is implemented to prevent severe software failures. 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. We propose a software rejuvenation model based on degraded physical memory. This model is implemented with a Markov chain and the system performance due to memory usage, by services, divided into four equal level. Hence, we offer four types of policies for software rejuvenation. The goal of the proposed method is to optimize availability of system. The results we achieve show the superiority of proposed method.

Keywords: *Software Rejuvenation, Aging, Availability, Markov Chain*

1. Introduction

With the growth of modern technology, all the industries are dependent on the computer systems. The performance of these computer systems are affected by hardware and software failures. These failures can create adverse results for a computer system or even to humans. Today, engineers are putting many efforts to produce hardware and software with a high level of confidence. In reliability literatures, aging phenomenon has been reported in software with long execution time, this phenomenon leads to increased failure rates or performance degradation during runs and eventually a crash failure[1].

Software aging has been observed in many kinds of systems, such as critical systems, telecommunication switching and billing software[2], networked UNIX workstations[3], web services[3], Android smartphones[3], spacecraft flight systems [6], Open stack cloud computing platform [3], virtual machine[3], real time system[8]. In addition, software aging could cause great losses in the safety-critical systems, including the loss of human lives[10].

Over the past years, many researchers have contributed in this field. Such as a cyclic non homogeneous Markovian chain model [11], Software rejuvenation for resource

[12], the evaluation of a periodic check pointing and rejuvenation schemes in operational software systems [13], The methods and opportunities for rejuvenation in ageing distributed software systems [3], the effect of time-trigged system rejuvenation policies on the service availability using a queuing model [15], and many another researches. System performance [3], including reliability [19] reliability analysis of an embedded cluster system [1], and availability [20,21] Escheikh in [22],[23]analyze power management performability, evaluating the impact on performance and energy consumption in virtualized systems, then, software aging and rejuvenation is one of the most important issues in software systems.

Software rejuvenation is a proactive approach to counteract software aging. Software rejuvenation is the concept of periodically restarting an application at a clean internal state. Define it as ‘‘the periodic pre-emptive rollback of continuously running applications to prevent failures [24]. Traditional software rejuvenation strategies are categorized into time-based and inspection-based. [25]The time-based approach builds the rejuvenation model, establishing functions between cost and time and between cost and workload in order to determine the optimal rejuvenation scheduling. The inspection-based approach determines the optimal rejuvenation scheduling through a collection of system data concerning resource usage and performance[26].

Several studies [27,28] have reported that one of the causes of performance degradation and unplanned software outages is the software aging phenomenon. Software aging observed in software as long-running software systems suffer from abnormal states, performance degradation, and even hang or failure. The reasons for software aging are consumption of operating system resources, data corruption, and rounding error accumulation, which could be accompanied by memory leaks, data fragments, unreleased file locks, and database connections. Software rejuvenation is one of the most commonly used approaches to handle issues caused by software aging. This process removes the accumulated errors and frees operating system resources [29].

In this paper, we propose a new model for software rejuvenation based on availability optimization with markov chain. In this model, the amount of free physical memory is used as a measure for resource degrades. Without rejuvenation, with performing applications, the free physical memory of a system reduces due to memory leaks and other software faults. In our proposed method, rejuvenation is performed when the amount of free physical memory reaches some critical levels. There will be four kinds of rejuvenation performed.

The organization of the paper is as follows. Section 2 reviews related works. Section 3 discuss about Problem description. Section 4 presents the proposed method. Section 5 shows some examples of experimental results. Finally, Sect. 6 provides the conclusions of the paper.

2. Literature Review

In general, the Software Aging phenomenon consists in the increase of failure rate and/or decrease of performance of a long-running software system. Aging effects are the results of error accumulation, in terms of leaked resources or corrupted state these effects can be detected by means of aging indicators, that is, system variables that can be directly measured and that can be related to Software Aging phenomena. [30]

Software Rejuvenation was defined as the preemptive rollback of continuously running applications to prevent failures in the future. Since an application may be

unavailable during rejuvenation, rejuvenation can increase the downtime and incur in some costs (e.g., costs due to the loss of business). However, these costs can be minimized by scheduling rejuvenation during the idlest times of an application.[31]

Many studies on Software Aging and Rejuvenation have been done in recent years. These studies have addressed different dimensions; these dimensions include type of analysis, type of system, aging indicators and Rejuvenation techniques.[31]

In terms of analysis type is divided to model-based method, measurements-based method and hybrid method.

In model-based analyses, a mathematical model of the system is considered, that include states in which the system is correctly working, states in which the system is failure-prone, and states in which software rejuvenation is taking place. Several kinds of model have been considered for this purpose, such as Markov Decision Processes and Stochastic Petri Nets [32],[33]. Model-based method provides an abstract view of system and a mathematical treatment. Measurements-based method that can be grouped in time series analysis, machine learning and etc. The basic idea of measurement-based rejuvenation approaches is to directly monitor system variables, namely aging indicators that can denote the onset of software aging, and predict the occurrence of aging failures by analyzing the collected runtime data statistically. This method has accurate prediction about aging but it need direct monitoring and not be easily generalizable. [14]

Hybrid method has been made to combine the benefits of both model-based and measurement-based approaches. In this paper also used this model. [31]

An important issue regards the type of system on which aging is analyzed, and/or rejuvenation actions are implemented. Software aging has been shown to affect many kinds of long-running software systems. Then types of system divide to three classes: safety critical systems, non-safety-critical systems, unspecified.[34]

Aging indicators are an important area of studies, since they are instrumental for detecting when the system state is prone to aging-failures, by monitoring them during the system execution. Aging indicators can be indicators of resource usage and performance indicators. In our article, memory consumption is considered as an indicator of aging. Rejuvenation is easy to implement based on aging indicators, but it is specific to certain systems. [31]

There are two techniques for rejuvenation; Application Restart (The whole application is restarted) and OS Reboot (restarts the OS). In this paper, we have also proposed two other types of rejuvenation techniques.[31]

3. Problem Description and Modeling

In software rejuvenation technique, when the application efficiency is reduced to a certain degree, recovery system and clean its internal states, restored efficiency and restart service. In other words, software rejuvenation is the concept of gracefully terminating an application and immediately restarting it at a clean internal state. This technique, increase system availability significantly. During the perform of rejuvenation, system is not available then, Sometimes rejuvenation may increase system downtime, but if schedule the rejuvenation and perform during when the service is idle, expected downtime is decrease[35].

The state diagram for software rejuvenation technique and transition rate between states is presented in Fig. 1, which consists of 4 states. In begin, the system is in a right and safe state (state S_0), but with reduced performance over time, the system will go to

the state prone to failure (state S_L), in this state, it is better to before go to failure state (state F) and system crash, transient to rejuvenation state (state R) until clean internal state and return to safe state [35].

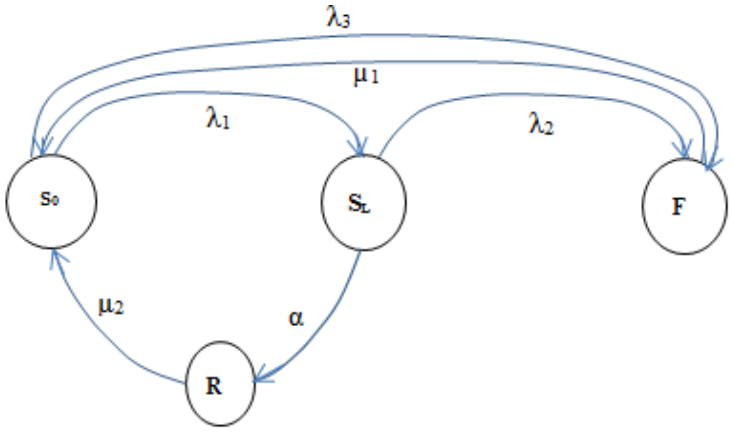

Fig.1. State transition diagram

Software rejuvenation is a useful action performed on software systems to prevent future software degradations and failures, but the cost in terms of downtime can be considerable. When the system is in rejuvenation state (state R) and failure state (state F), system is unavailable. Then availability of system is determined using

$$A(\infty) = 1 - (P(R) + P(F)) \qquad (1)$$

The transition rates are denoted by λ_1, λ_2, λ_3, μ_1, μ_1 and α, that λ_1 denotes the transition rate from state S_0 to state S_L, λ_2 denoting the transition rate from state S_L to state F and $\lambda 3$ denoting the transition rate from state F to state S_0.

Having the system down in order to perform rejuvenation, means that the system is not available for a time period.

4. The Proposed Method

In proposed model we define l_i, where i=L, M, H, U, and each i represents one free memory level [10] (Fig. 2). Due to we offer four types of policies for software rejuvenation, therefor we have four levels for free memory. The critical level of free memory is denoted by l_U. When free memory due to memory leakage is less than l_U, the system will crash and a repair will occur to restore the initial level of free memory. When system memory reaches l_U, the system will return to either of low efficiency, medium efficiency, or high efficiency states through rejuvenation. In this way, the system will go back to its initial state, which is the healthy and powerful state of the system.

A system's efficiency is reduced over a period of constant operation, and the system may become instable. This is where rejuvenation comes into play. However, a rejuvenation process is actually performed; many questions need to be answered: what is the best time for rejuvenation? How much rejuvenation is required? Etc

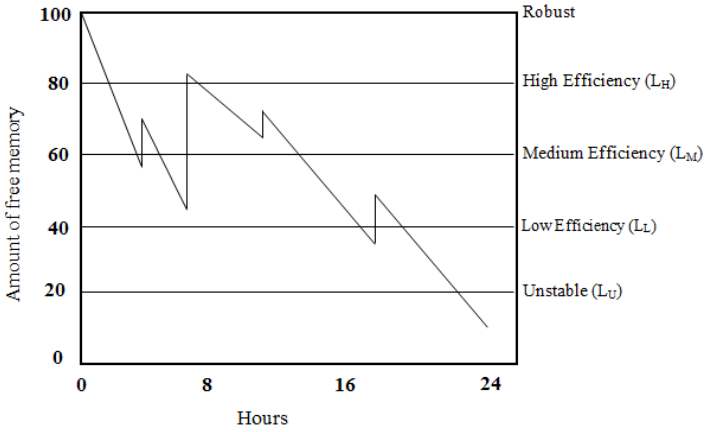

Fig.2. Amount of free memory

Below, a few rejuvenation methods are proposed and their availability is compared, so that the most suitable policy can be adopted.

- **Rejuvenation Type-I**: In this method, 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, system state is changed from unstable to low efficiency. This type of rejuvenation does not result in significant improvements.

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

- **Rejuvenation Type-III**: When the system is in the unstable state, performing a general restart – 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 Type**-IV: This type of rejuvenation is performed at the level of physical machine and is the most typical type of rejuvenation. It is simply turning the system off and then on again. It is the costlier type of rejuvenation, but it will transfer the system to a healthy and powerful state. Fig. 3 shows the four rejuvenation types.

As mentioned, the Markov model has 10 states. Fig. 3 shows these 10 states and the transitions between them. L_U is rejuvenation threshold. A specific type of rejuvenation is selected based on the conditions. As Fig. 3 shows, rejuvenation Type-I restores state R_L, rejuvenation Type-II restores state R_M, rejuvenation Type-III restores state R_H, and finally, rejuvenation Type-IV restores state R.

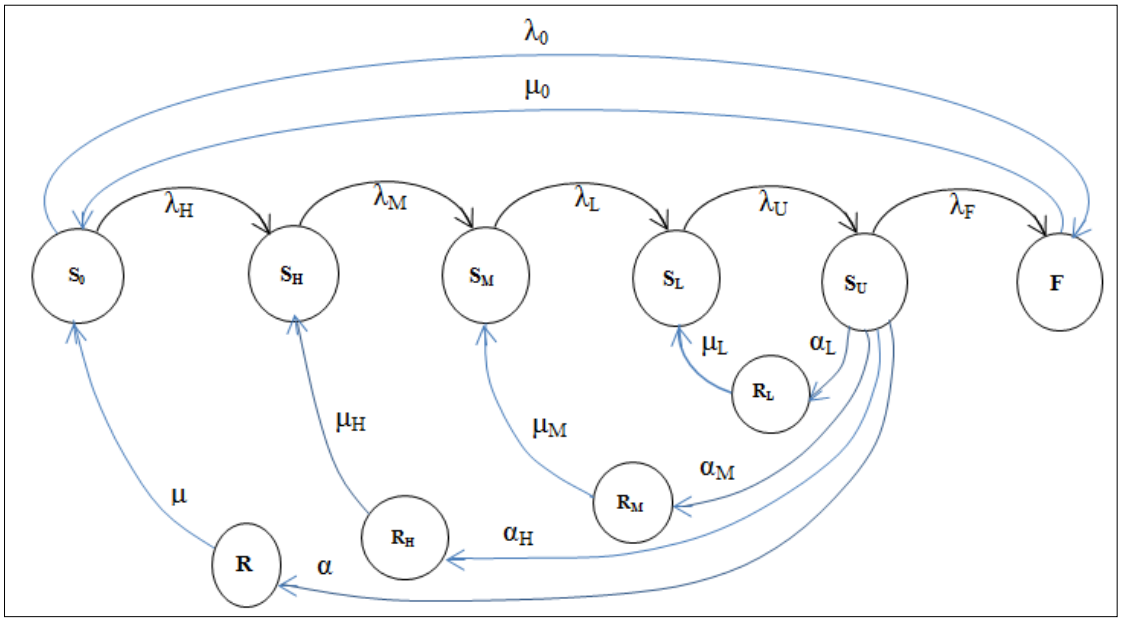

Fig.3. State transition diagram

Markov state space at time t is as follows:

$$X(t) = [\{S_i\ ;\ i = 0, H, M, L, U\} \cup \{(Rj); j = L, M, H\} \cup \{F\}] \tag{2}$$

Markov chain has 10 states in this model. Q is a square matrix, which represents the transition probability (Equation 3). The probability beginning states are in left columns of matrix and probability of transition states presented in rows of matrix.

$$Q = \begin{array}{c} \\ S_0 \\ S_H \\ S_M \\ S_L \\ S_U \\ F \\ R_L \\ R_M \\ R_H \end{array} \begin{array}{c} \begin{array}{cccccccccc} S_0 & S_H & S_M & S_L & S_U & F & R_L & R_M & R_H & \end{array} \\ \begin{bmatrix} d_1 & \lambda_H & 0 & 0 & 0 & \lambda_0 & 0 & 0 & 0 & 0 \\ 0 & d_2 & \lambda_M & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & d_3 & \lambda_L & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & d_4 & \lambda_U & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & d_5 & \lambda_F & \alpha_L & \alpha_M & \alpha_H & \alpha \\ \mu_0 & 0 & 0 & 0 & 0 & d_6 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \mu_L & 0 & 0 & d_7 & 0 & 0 & 0 \\ 0 & 0 & \mu_M & 0 & 0 & 0 & 0 & d_8 & 0 & 0 \\ 0 & \mu_H & 0 & 0 & 0 & 0 & 0 & 0 & d_9 & 0 \\ \mu & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & d_{10} \end{bmatrix} \end{array} \tag{3}$$

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

$$\begin{aligned} &(\lambda_0 + \lambda_H)S_0 = \mu R + \mu_0 F \\ &\lambda_M S_H = \lambda_H S_0 + \mu_H R_H \\ &\lambda_L S_M = \lambda_M S_H + \mu_M R_M \\ &\lambda_U S_L = \lambda_L S_M + \mu_L R_L \\ &(\lambda_F + \alpha_L + \alpha_M + \alpha_H + \alpha)S_U = \lambda_U S_L \\ &\mu_0 F = \lambda_F S_U + \lambda_0 S_0 \\ &\mu R = \alpha S_U \\ &\mu_H R_H = \alpha_H S_U \\ &\mu_M R_M = \alpha_M S_U \\ &\mu_L R_L = \alpha_L S_U \end{aligned} \tag{4}$$

$$S_0 + S_H + S_M + S_L + S_U + R + R_H + R_M + R_L + F = 1$$

Due to memory leakage, in the system has five degraded states namely high, medium, low efficiency states and unstable and failure states. The transition rate of state to another states are exponentially distributed with rates λ_H, λ_M, λ_L, λ_U, λ_F respectively. Also, the life time and repair time of the active state is assumed to be exponentially distributed with failure rate λ_o and repair rate μ_0. Repair rate for all state shown by μ_k, since four type rejuvenation proposed, then $\mu_k=\mu_L,\mu_M,\mu_H,\mu$, and rejuvenation rates shown by α_k and since four type rejuvenation proposed, then $\alpha_k=\alpha_L,\alpha_M,\alpha_H,\alpha$.

By solving the above formulas recursively, the probability of each of the states is obtained as follows:

$$S_0 = \frac{\alpha S_U+\mu_0 F}{\lambda_H+\lambda_0}\text{،}\ S_H = \frac{\alpha_H S_U+\lambda_H S_0}{\lambda_M}\text{،}\quad S_M = \frac{\alpha_M S_U+\lambda_M S_H}{\lambda_L}\text{،}\ S_L = \frac{\alpha_L S_U+\lambda_L S_M}{\lambda_U}$$

$$S_U = \frac{\lambda_0 S_0-\mu_0 F}{\lambda_F}\text{،}\ \ R = \frac{\alpha}{\mu}S_U\text{،}\ \ R_H = \frac{\alpha_H}{\mu_H}S_U\text{،}\quad R_M = \frac{\alpha_M}{\mu_M}S_U\text{،}\quad R_L = \frac{\alpha_L}{\mu_L}S_U \qquad (5)$$

$$F = \frac{(\lambda_0+\lambda_H)S_0-\mu R}{\mu_0}$$

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

$$A(\infty) = 1-(R+R_H+R_M+R_L+F) \qquad (6)$$

It is assumed that the process is initially in S_0 state. At S_0 state all the memory is free; therefore, $PS_0(0)=1$ ، $PS_H(0)=0$ ،$PS_M(0)=0$ ،$PS_L(0)=0$ ،$PS_U(0)=0$ و $F(0)=0$.

The availability function and failure probability are as follows:

$$A_F(t) = \{S_0+S_H+S_M+S_L\}$$
$$f(t) = \{\lambda_0 S_0+\lambda_F S_U\} \qquad (7)$$

5. Experiments and Numerical Results

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

In this section, the results of 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. The coding was done in MATLAB software. The system availability is calculated by considering the probability of being in each state. The transition probability between the states is shown in Tables 1 and 2, $\mu_1$is probability of repair and $\mu_2$is probability of rejuvenation. In order to calculate the performance indices, we set λ=1 and α=1. System availability values for each case are displayed in Figs 4 and 5.

Table 1. Rejuvenation and repair rates values for first method

Exp1	Exp2	Exp3	Exp4	Exp5	Exp6	Exp7	Exp8
μ_1=0.5	μ_1=0.75	μ_1=1	μ_1=1.25	μ_1=1.5	μ_1=1.75	μ_1=2	μ_1=2.25
μ_2=10	μ_2=11	μ_2=12	μ_2=13	μ_2=14	μ_2=15	μ_2=16	μ_2=17

Table 2. Rejuvenation and repair rates values for proposed method

Exp1	Exp2	Exp3	Exp4	Exp5	Exp6	Exp7	Exp8
μ_0=1	μ_0=1	μ_0=1	μ_0=1	μ_0=1	μ_0=1	μ_0=1	μ_0=1
μ_L=10	μ_L=11	μ_L=12	μ_L=13	μ_L=14	μ_L=15	μ_L=16	μ_L=17
μ_M=9	μ_M=10	μ_M=11	μ_M=12	μ_M=13	μ_M=14	μ_M=15	μ_M=16
μ_H=8	μ_H=9	μ_H=10	μ_H=11	μ_H=12	μ_H=13	μ_H=14	μ_H=15
μ =4	μ =5	μ =6	μ =7	μ =8	μ =9	μ =10	μ =11

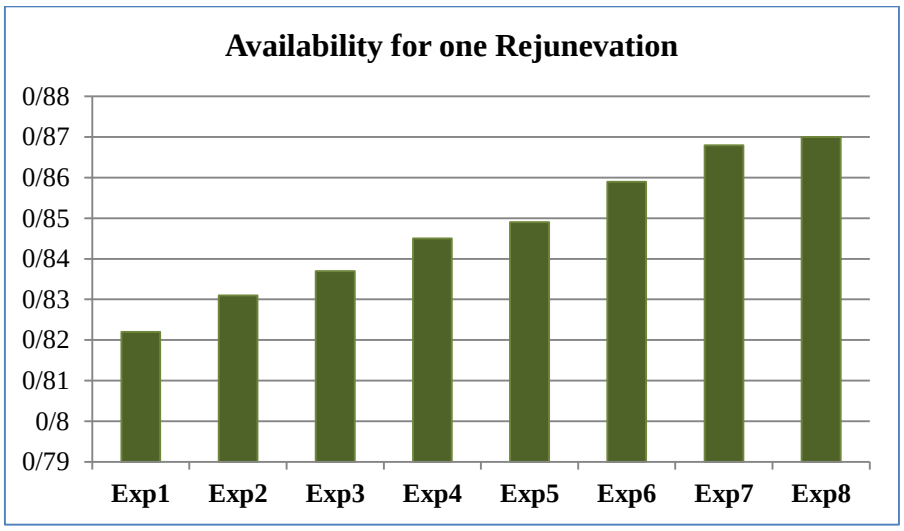

Fig.4. Availability for first method

In Fig. 4, the variation of the availability versus the variation of the different rejuvenation rate μ_1 and μ_2 is shown. Availability is increased when the value of μ_1 and μ_2 is increased.

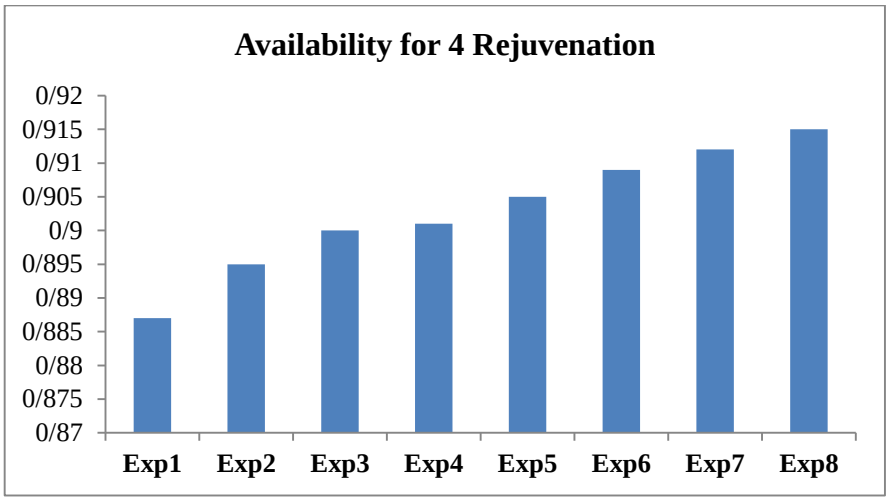

Fig.5. Availability for proposed method

In Fig. 5, the variation of the availability versus the variation of the different rejuvenation rate μ_1, μ_2, μ_3 and μ_4 is shown. Availability is increased when the value of rejuvenation rate is increased. As can be seen in Figs 4 and 5, there is a difference between availability when only one rejuvenation method is used and when the four proposed rejuvenation types are used.

6. Conclusion

In this paper, two different model of the software rejuvenation are described and studied. The first model consists of performing software rejuvenation with 4 states. This technique has been also studied in past papers. After studying this technique extensively, we infer that when the rejuvenation action is performed, the system is not available for a long time. Furthermore, we propose a new type of modeling for the resource degradation, consisting of performing software rejuvenation of ‘‘partial restart’’ type at four different degradation levels, and we considered four levels for free memory of system. Moreover, a case study is presented and we compared the two different rejuvenation techniques together. Finally, we evaluated the availability of the both of models, our study showed that the proposed model optimizes availability of the system.

References

[1] M. Jain, T. Manjula, R. Gulati, “Software Rejuvenation Policies for Cluster System,” Springer, the National Academy of Sciences, 2016.

[2] W. Jifang, G. Donghua, E. Qing, H. Zhenyu, “Research on Rejuvenation of Software,” United States of America Journal of Computational and Theoretical Nanoscience, 2016.

[3] F. Salfner, K. Wolter, “Analysis of service availability for time-trigged rejuvenation policies,” The Journal of Systems and Software, 2010.

[4] J. Ghayathri, S. Pannirselvam, “Categorization of web services based on QOS constraint using decision tree classifier,” International Journal of Innovative Technology and Creative Engineering 2016.

[5] Y. Qiao, Zh. Zheng, F.Q. Fang, F. Qin, K.S.Trivedi, K.Y. Cai, "Two-Level Rejuvenation for Android Smartphones and Its Optimization," IEEE Trans. Reliability, 2019.

[6] A. Avritzer, R. Cole, E. Weyuker, "Methods and Opportunities for rejuvenation in aging distributed software systems," The Journal of Systems and Software, 2010.

[7] D. Sun, G. Chang, Q. Guo, C. Wang, "A dependability model to enhance security of cloud environment using systemlevel virtualization techniques," In: Proceeding of 1st Conference on Pervasive Computing, Signal Processing and Applications, 2010.

[8] P. Kulkarni, "Software rejuvenation and workload distribution in virtualized system," International Journal of Innovated Research in Computer and Communication Engineering. 2015.

[9] G. Levitin, L. Xing, H. Huang, "Optimization of partial software rejuvenation policy," Reliability Engineering and System Safety, 2019.

[10] V.P. Koutras, N. Agapios, A. George, G.A. Gravvanis, "On the optimization of free resources using non-homogeneous Markov chain software rejuvenation model," Reliability Engineering and System Safety, 2007.

[11] V.P. Koutras, A.N. Platis, A. George, G.A. Gravvanis, "Software rejuvenation for resource optimization based on explicit approximate inverse precondition," Appl Math Comput, 2007.

[12] M. Grottke, L. Li, K. Vaidyanathan, K.S. Trivedi, "Analysis of software aging in a web server," IEEE Trans Reliab, 2006.

[13] Y. Hong, D. Chen, L. Li, K.S. Trivedi, "Closed Loop Design for Software Rejuvenation," In Proc. of the Workshop on self-healing, adaptive and self-managed systems, 2002.

[14] M. Shereshevsky, J. Crowell, B. Cukic, V. Gandikota AND Y. Liu, "Software aging and multifractality of memory resources," In Dependable Systems and Networks, 2003.

[15] S. Garg, A. Puliafito, M. Telek, K.S. Trivedi, "Analysis of preventive maintenance in transactions based software systems," IEEE Trans Comput, 1998.

[16] E. Torres, G. Callou, Andrade E. A hierarchical approach for availability and performance analysis of private cloud storage services. Computing 2018; 100 (6): 621-644

[17] Yang M, Min G, Yang W, Li Z. Software rejuvenation in cluster computing systems with dependency between nodes. Computing 2014; 96: 503-526.

[18] J. Dantas, R. Matos, J. Araujo, P. Maciel, "Eucalyptus-based private clouds: availability modeling and comparison to the cost of a public cloud," Computing 2015.

[19] J.D. Esary, H. Ziehms, "Reliability analysis of phased missions, Proceedings of the Conference on Reliability and Fault Tree Analysis," SIAM, 1975.

[20] M. Torquato, I.M. Umesh, P. Maciel, "Models for availability and power consumption evaluation of a private cloud with VMM rejuvenation enabled by VM live Migration," Journal of Supercomputing, 2018.

[21] Z. Rahmani Ghobadi, H. Rashidi, "Availability analysis and improvement with software rejuvenation," Third International Conference on Contemporary Issues in Computer and Information Sciences, 2012.

[22] M. Escheikh, Z. Tayachi, K. Barkaoui, "Workload-dependent software aging impact on performance and energy consumption in server virtualized systems,"27th International Symposium on Software Reliability Engineering Workshops (ISSREW), 2016.

[23] M. Escheikh, K. Barkaoui, H. Jouini, "Versatile workload-aware power management performability analysis of server virtualized systems," Journal of Systems and Software, 2017.

[24] A. Avritzer, J. Weyuk, "Monitoring smoothly degrading systems for increased dependability," EmpirSoftwEng journal, 1991.

[25] Y. Fang, B. Yin, G. Ning, Zh. Zheng, K. Cai, "A Rejuvenation Strategy of Two-Granularity Software Based on Adaptive Control," IEEE 22nd International Symposium on Dependable Computing, 2017.

[26] W. Dang, J. Zeng, "Optimization of Software Rejuvenation Policy based on State-Control-Limit," International Journal of Performability Engineering, 2018.

[27] E. Torres, G. Callou, E. Andrade, "A hierarchical approach for availability and performance analysis of private cloud storage services," Computing 2018.

[28] M. Yang, G. Min, W. Yang, Z. Li, "Software rejuvenation in cluster computing systems with dependency between nodes," Computing 2014.

[29] G. Levitin, L. Xing, H. Huang, "Optimization of partial software rejuvenation policy," Reliability Engineering & System Safety, 2019.

[30] M. Grottke, R. Matias, and K.S. Trivedi, "The fundamentals of software aging," IEEE Int'l. Conf Workshops in Software Reliability Engineering, 2008..

[31] D. Cotroneo, R. Natella, R. Pietrantuono, S. Russo, "A Survey of Software Aging and Rejuvenation Studies," ACM Journal on Emerging Technologies in Computing Systems, 2014.

[32] Y. Bao, X. Sun, and K. Trivedi, "A workload-based analysis of software aging, and rejuvenation," IEEE Transactions on Reliability, 2005.

[33] S. Garg, A. Puliafito, M. Telek and K. Trivedi, "Analysis of software rejuvenation using markov regenerative stochastic petri net," Sixth Int'l. Symp in Software Reliability Engineering, 1995.

[34] D. Cotroneo, R. Natella, R. Pietrantuono, AND S. Russo, "Software aging and rejuvenation: Where we are and where we are going," IEEE Third International Workshop on Software Aging and Rejuvenation, 2011.

[35] Y. Huang, C. Kintala, N. Kolettis, N. Fulton, "Software rejuvenation: analysis, module and application," Proceedings of IEEE international symposium on fault tolerant computing, 1995.