Proactive Software Rejuvenation Solution for Software Fault Tolerance System

Khin Moh Moh Lwin
Myanmar Maritime University
khinmohlwin@gmail.com

Abstract

Software failures are now known to be a dominant source of system outages. The cost of unplanned outages is higher than the cost of planned outages. Software systems often experience aging related defects which cause unexpected outages and affecting its availability. A system should have the capability enough to tolerate the faults and operate continuously in spite of faults and interruptions. A proactive fault management method to deal with the software aging phenomenon is software rejuvenation. In this paper, we focus on a high assurance software system with fault-tolerance and proactive software rejuvenation and analyze the stochastic behavior of such a software system. More precisely, we consider a software fault tolerance system with redundant and evaluate the steady-state system availability based on the familiar Markovian analysis through the use of numerical analysis.

Keywords: Availability, Software fault tolerance, Software Aging, Software rejuvenation, Stochastic modelling

1. Introduction

It is well known that, currently, computer system outages are more often due to software faults, than hardware faults [3, 8]. Several studies have reported that one of the causes of unplanned software outages is the software aging phenomenon. Software aging [7], a recent phenomenon under intense investigation, is caused by aging-related defects that lead to the accumulation of errors. The aging related defects accumulate due to memory leaks, data corruption, fragmentation, round-off errors [6, 9]. Software aging is gaining in significance because of the growing economic importance of software, and software is the "capital" of many high-tech firms. Aging occurs because software is extremely complex and never wholly free of errors. It is almost impossible to fully test and verify that a piece of software is bug-free. Under aging conditions, the state of the software degrades gradually with time, inevitably resulting in undesirable consequences.

Software rejuvenation techniques [4] become necessary to mitigate or avoid the consequences of software aging. The software rejuvenation techniques are based mainly on the restart of the service to come back to a state without aging. The software rejuvenation techniques do not avoid the software aging problem, they only mitigate the symptoms.

The work presented in this paper aims to offer a proactive software rejuvenation solution for Internet Services against software aging. Maintaining sever availability by software rejuvenation has gained importance and studied. Here we discuss improving software system availability by software rejuvenation. This approach has been designed to use over any server or service.

The rest of this paper is composed as follows: Section 2 describes the related work. After that, Section 3 describes the software rejuvenation solution for software fault-tolerant. Section 4 presents proposed software fault-tolerant system with software rejuvenation. Finally, Section 5 presents the conclusions extracted from this work.

2. Related Work

In recent years, considerable attention has been devoted to the phenomenon of software aging. A great deal of research effort on modeling software aging and rejuvenation considers continuously running software systems. Huang et al. [4] used continuous time Markov chain to model software rejuvenation. They considered the two-step failure model where the application goes from the initial robust (clean) state to failure probable (degraded) state from which two actions are possible: rejuvenation or transition to failure state. Both rejuvenation and recovery from failure return the software system to the robust state.

Garg et al. [10] introduced into the model the idea of periodic rejuvenation. To deal with deterministic interval between successive rejuvenations the system behavior was represented through a Markov regenerative stochastic Petri net model. Dohi et al. [11], Suzuki et al. [1, 2] extend the Huang et al. Model [4] to semi-Markov models and further develop non-parametric algorithms to estimate the optimal software rejuvenation schedule. In this paper, we focus on a high assurance software system with fault-tolerance.

3. Software Rejuvenation Solution for Software Fault-tolerance

The normal stages of software life begin with its initialization and the beginning of its processing. It works flawlessly for a time. While it is working, it is slowly degrading. Eventually a software fault will activate. Perhaps it runs out of memory or some other resource, or gets confused by the existence of unused data. When this occurs, an error is likely to happen which might be followed by some type of failures. In software that is designed for high availability, there are mechanisms in place that will detect the fault. The fault will be detected and isolated and recovery mechanisms will be tried. Sometimes recovery from the fault is achieved through simple steps, sometimes more aggressive steps will be required. This all takes time, as shown in Figure 1.

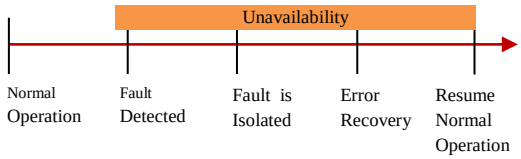

Figure 1. Timeline I : Usual fault detection, isolation and recovery stages

Rebooting or restarting is one of the many recovery actions that are possible. The restart is done proactively. This rejuvenation is the graceful shutdown and the reboot of the software as shown in Figure 2.

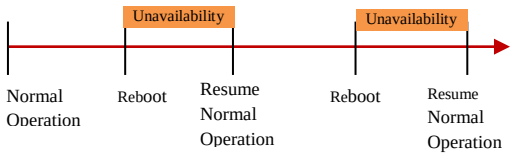

Figure 2. Timeline II: Rejuvenating

If the part of the system to be rejuvenated is redundant then it is possible to rejuvenate without losing any service shown in Figure 3.

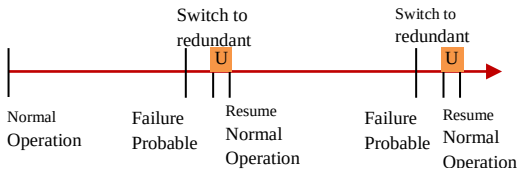

Figure 3. Timeline III: Rejuvenation and switchover to redundant

According the Timeline Figures, a fault-tolerant software system with redundancy and proactive rejuvenation can provide higher availability.

4. Proposed Fault Tolerance Model

In this section we consider a fault-tolerant system with redundancy and proactive rejuvenation. We draw the state diagram of proposed software fault tolerance mechanism is given in Figure 4.

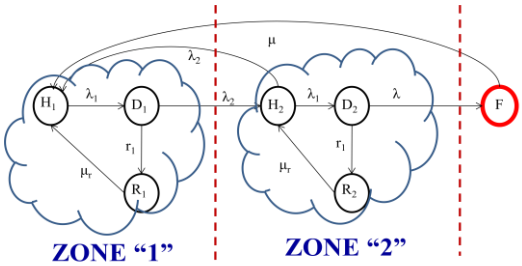

Figure 4. State diagram of fault tolerance model

The above state diagram has seven states in total. Each state has the probability of being in its own state. These states are given as per the Zone 1 and Zone 2 of above Figure 4. In Zone 1, there are three states like, Healthy (H_1), Degradation (D_1) and Rejuvenation (R_1). We use Markov chain formulation for analyzing the above states. The cloud in Zone 1 provides service if it is in Healthy State. If there is a problem then it moves to Degradation State with rate λ_1. This is the base longevity interval.

This transaction states from Healthy to Degradation is being monitored by Monitoring Unit. The Monitoring Unit is smart enough to move from existing cloud to Healthy State of another cloud i.e. Zone 2 with switching rate λ_2. It depends upon the how fast Monitoring Unit gets activated to move to another cloud i.e. Zone 2. Zone 2 also has three states i.e. Healthy (H_2), Degradation (D_2) and Rejuvenation (R_2) like Zone 1. Even in Zone 2, we consider the chances of moving to Degradation State with λ_1.We try to consider almost all possibilities of system going down. If Zone 2 also gets degraded being in state D_2, we assume by this time Zone 1 will be ready to provide the service after rejuvenated then Monitoring Unit switches to the previous cloud Zone 1 with switching rate λ_2.

Table 1. Steady-state probabilities

Symbol	Meaning
π_{H1}	Probability of being in Healthy State of Zone 1
π_{D1}	Probability of being in Degradation State of Zone 1
π_{R1}	Probability of being in Rejuvenation State of Zone 1
π_{H2}	Probability of being in Healthy State of Zone 2
π_{D2}	Probability of being in Degradation State of Zone 2
π_{R2}	Probability of being in Rejuvenation State of Zone 2
π_F	Probability of being in Failure State

We may compute the steady-state probabilities by first writing down the steady-state balance equations of Figure 4 are as follows:

$$\mu\pi_F + \mu_r\pi_{R_1} + \lambda_2\pi_{H_2} = \lambda_1\pi_{H_1} \quad [1]$$

$$\lambda_1\pi_{H_1} = \lambda_2\pi_{D_1} + r_1\pi_{D_1} \quad [2]$$

$$r_1\pi_{D_1} = \mu_r\pi_{R_1} \quad [3]$$

$$\lambda_2\pi_{D_1} + \mu_r\pi_{R_2} = \lambda_1\pi_{H_2} + \lambda_2\pi_{H_2} \quad [4]$$

$$\lambda_1\pi_{H_2} = \lambda\ \pi_{D_2} + r_1\pi_{D_2} \quad [5]$$

$$r_1\pi_{D_2} = \mu_r\pi_{R_2} \quad [6]$$

$$\lambda\ \pi_{D_2} = \mu\pi_F \quad [7]$$

The conservation equation of Figure 4 is obtained by summing the probabilities of all states in the system and the sum of the equation is 1.

$$\sum_{i=1}^{2}\pi_{H_i} + \sum_{i=1}^{2}\pi_{D_i} + \sum_{i=1}^{2}\pi_{R_i} + \pi_F = 1 \quad [8]$$

Combining the above-mentioned balance equations with the conservation equation, and solving these simultaneous equations, we acquire the closed-form solution for the system.

$$\pi_{D_1} = \frac{\lambda_1}{r_1 + \lambda_2}\pi_{H_1} \quad [9]$$

$$\pi_{R_1} = \frac{r_1}{\mu_r}\left(\frac{\lambda_1}{r_1 + \lambda_2}\right)\pi_{H_1} \quad [10]$$

$$\pi_{H_2} = A\lambda_2\left(\frac{\lambda_1}{r_1 + \lambda_2}\right)\pi_{H_1} \quad [11]$$

$$\pi_{D_2} = A\lambda_2\left(\frac{\lambda_1}{r_1 + \lambda_2}\right)\left(\frac{\lambda_1}{r_1 + \lambda}\right)\pi_{H_1} \quad [12]$$

$$\pi_{R_2} = \frac{r_1}{\mu_r}A\lambda_2\left(\frac{\lambda_1}{r_1 + \lambda_2}\right)\left(\frac{\lambda_1}{r_1 + \lambda}\right)\pi_{H_1} \quad [13]$$

$$\pi_F = \frac{\lambda}{\mu}A\lambda_2\left(\frac{\lambda_1}{r_1 + \lambda_2}\right)\left(\frac{\lambda_1}{r_1 + \lambda}\right)\pi_{H_1} \quad [14]$$

$$\pi_{H_1} = \left\{\begin{array}{l} 1 + \frac{\lambda_1}{r_1 + \lambda_2} + \frac{r_1}{\mu_r}\left(\frac{\lambda_1}{r_1 + \lambda_2}\right) + \\ A\lambda_2\left(\frac{\lambda_1}{r_1 + \lambda_2}\right) + A\lambda_2\left(\frac{\lambda_1}{r_1 + \lambda_2}\right)\left(\frac{\lambda_1}{r_1 + \lambda}\right) \\ + \frac{r_1}{\mu_r}A\lambda_2\left(\frac{\lambda_1}{r_1 + \lambda_2}\right)\left(\frac{\lambda_1}{r_1 + \lambda}\right) + \\ \frac{\lambda}{\mu}A\lambda_2\left(\frac{\lambda_1}{r_1 + \lambda_2}\right)\left(\frac{\lambda_1}{r_1 + \lambda}\right) \end{array}\right\} \quad [15]$$

Where

$$A = \frac{1}{-r_1\left(\frac{\lambda_1}{r_1 + \lambda}\right) + \lambda_2 + \lambda_1}$$

Availability models capture failure and repair behavior of systems and their components. States of the underlying Markov chain will be classified as up states or down states. The system is not available in the Failure State (F). The system availability in the steady-state and Downtime are defined as follows:

$$Availability = 1 - P_F \quad [16]$$

$$Downtime(T) = T \times P_F \quad [17]$$

4.1. Model Analysis

In order to examine the behavior of the system studied in this paper, we perform numerical analysis using the system-operating parameters [4] as shown in Table 2.

Table 2. Parameters values and description

Para-meter	Description	Values
λ_1	Degradation Rate	2 time/3 days
$1/\lambda_2$	Switching time	1 min
λ	Failure Rate	3time/ month
r_1	Rejuvenation triggering rate	2 times/week
$1/\mu_1$	Recovery Time	20 min
$1/\mu$	Repair Time	30 min
$1/\mu_r$	Rejuvenation Time	10 min
T	Unit time interval	24*30*12 days

The variation of the steady-state availability of the system with different number of rejuvenation intervals is plotted in Figure 5. We perform software rejuvenation with the interval from (rate=0.010) to (rate=0.070). We see from Figure 5 that the amount of availability increment is significant.

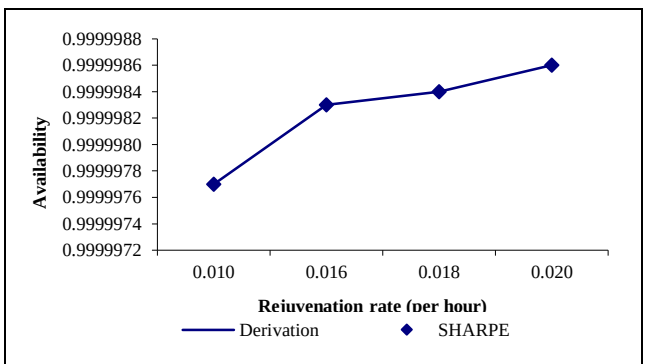

Figure 5. Availability vs different rejuvenation rates

In Figure 6 we have plotted the downtime as a function of the rejuvenation rates. The downtime of the system with our approach decreases as the rejuvenation rate increases.

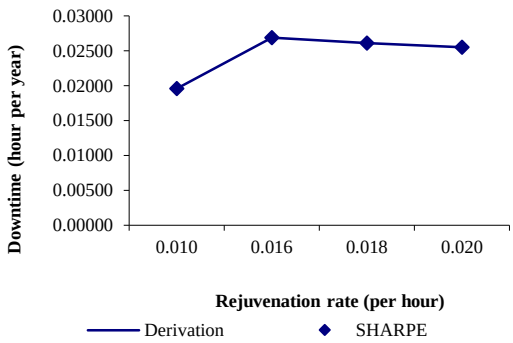

Figure 6. Downtime vs different rejuvenation rates

4.2. Model Validation

SHARPE [5] is well known package in the field of reliability and performability. It is possible to use different kinds of models hierarchically for different physical or abstract levels of the system and to use different kinds of models to validate each other's results. So for our models' validity we use this tool. And we have found that the evaluation results through SHARPE are same with our mathematical derivation results as shown in Figure 5 and Figure 6.

5. Conclusion

This paper studies software rejuvenation method for enhancing the system availability. We also presented the software rejuvenation solution for software fault-tolerance system. Rejuvenation does not remove faults but rather prevents them from manifesting themselves as unpredictable whole system failure. We also presented Markov models for analyzing availability of the system with proactive rejuvenation. We validated our experiment results with the evaluation results through SHARPE. It is found that our derivation results and SHARPE results are same. Future work will include experiments with an implementation under real world conditions to verify the practical efficacy of our approach.

References

[1] H. Suzuki, T. Dohi, K. Go__eva-Popstojanova, and K.S. Trivedi, "Analysis ofmultistep failuremodels with periodic software rejuvenation", Advances in Stochastic Modelling (J. R. Artalejo and A. Krishnamoorthy, eds.), Notable Publications, Inc., 2002, pp. 85–108.

[2] H. Suzuki, T. Dohi, N. Kaio, and K.S. Trivedi, "Maximizing interval reliability in operational software system with rejuvenation", Proc. 14th Int'l Symp. on Software Reliab. Eng., IEEE CS Press, 2003, pp. 246–256.

[3] J. Gray and D. P. Siewiorek, "High-Availability Computer System, "IEEE Computer 24, September 1991. pp. 39-48.

[4] K. C. K. N. Huang, Yennun and N. D. Fulton. Software rejuvenation: Analysis, module and applications. In FTCS '95: Proceedings of the Twenty-Fifth International Symposium on Fault-Tolerant Computing, Washington, DC, USA, 1995. IEEE Computer Society. pp. 381-390.

[5] K. S. Trivedi. SHARPE 2002: Symbolic Hierarchical Automated Reliability and Performance Evaluator. In Proc. Int. Conference on Dependable Systems and Networks, 2002, pp. 544.

[6] K.S.Trivedi, K.Vaidyanathan, K.Goseva-Postojanova, 2000, Modeling and analysis of software aging and rejuvenation, Proc. 33rd Annual Simulation Symposium, pp. 270-279.

[7] M. Grottke and K S Trivedi, 2007 Fighting bugs:Remove, retry, replicate and rejuvenate, IEEE Computer, vol.40, no.2, pp. 107-109.

[8] M. Sullivan and R. Chillarege, "Software Defects and Their Impact on System Availability- A Study of Field Failure in Operating System," Proceedings of the 21st IEEE International Symposium on Fault-Tolerant Computing, 1991, pp. 2-9.

[9] R Matias, K S Trivedi and P M Maciel 2010. Using Accelerated Life Tests to Estinate Time to Software Aging Failiure, Proc.2010 IEEE 21st International Symposium on Software Reliability Engineering, pp. 211-219.

[10] S. Garg, A. Puliafito, M. Telek, and K. S. Trivedi, “Analysis of Software Rejuvenation using Markov Regenerative Stochastic Petri Net”, Proceedings of the 6th International Symposium on Software Reliability Eng., IEEE Computer Society Press, Los Alamitos, CA, 1995, pp. 24-27.

[11] T. Dohi, K. Go__eva-Popstojanova, and K.S. Trivedi, “Estimating software rejuvenation schedule in high assurance systems”, The Computer Journal, Vol. 44, 2001, pp. 473–485.