Availability Analysis of Software Aging and Software Rejuvenation on Virtualized Platforms

Dr. Ohnmar Nhway

Faculty of Computer Systems and Technologies, University of Computer Studies (Pyay)
ohnmarnhway77@gmail.com

Abstract— **Nowadays, software faults are the major cause of computer system failure. Moreover, many researchers have reported the phenomenon of 'software aging', characterized by progressive performance degradation due to the exhaustion of operating system resources. Software rejuvenation is one of the most important techniques to counteract software aging. The difficulty for software aging prediction is that it can be due to two or more resources simultaneously involved in the service failure. In this paper, we present availability analysis of software aging and software rejuvenation in virtualized platforms. We decided to evaluate the use of powerful of Machine Learning (ML) algorithms to predict the time to crash (TTC) of a system which suffers from software aging phenomena at our previous work. We also present load and time dependent software rejuvenation (LTDSR) policy that can be applied with our predictable data in virtualized platforms. Moreover, the behavior of the system is represented through a Stochastic Petri Net (SPN) model. Numerical analysis of the system availability is carried out the SHARPE tool simulation.**

***Keywords*: Availability, Software Aging, Software Rejuvenation, Stochastic Petri Net, Virtualization**

I. INTRODUCTION

Nowaday, Information Technology has become the backbone of every business. The business continuity is a key objective of an organization; it means that operations are up and running 24x7 services. Therefore, mechanisms are needed that guarantee correct service in the presence of failure of system components, be it software or hardware elements [1].

A number of recent studies have reported the phenomenon of "software ages", characterized by progressive performance degradation and/or an increased occurrence rate of hang/crash failures of a software system due to the exhaustion of operating system resources. To counteract this phenomenon, a proactive technique called "software rejuvenation" has been proposed.

In our previous work, we focused on prediction for software aging due to resource exhaustion such as memory. A set of Machine Learning Algorithms has been analyzed for predicting system crashes due to the resource exhaustion caused by software aging [8]. So, we measured Apache web server performance in such a system it is necessary to run a tool on the clients that generates a specific HTTP workload. We mainly emphasize the healing of software aging of web server in single physical machine and hardware failure is not considered.

The contribution of this paper is both the load and time based rejuvenation policy with our predictable data from software rejuvenation methodology and virtualization technology which are combined to counteract the software aging problem for virtualized platforms.

The behavior of the system is represented through a Stochastic Petri Net (SPN) model which is subsequently solved for steady state as well as transient conditions. But, we expect that there would be a trade-off involved between the down time caused due to crash failures and downtime due to rejuvenation depending on how often it is performed.

Rest of the paper is organized as follows. Related research is highlighted in Section II. We explain the system architecture of LTDSR policy model and evaluate with analysis and downtime in Section III. We conclude this work and discuss future avenues of research is Section IV.

II. RELATED WORKS

Many researchers have been constructed the models to help predict when the software rejuvenation should occur. The focus in this paper is on "how" to rejuvenate or prevent the crash failures, the gracefully termination of predictable model before it is restarted.

The work most closely related to our work is the one provided by T.Thein et al. [12],[13]. They use a timely rejuvenation policy in a high available consolidated system. Different configurations of consolidated servers in the form of one physical and two physical servers in the scheme of hot standby are considered. However, they do not consider the VMM failure and its rejuvenation issues. We propose load and time based software rejuvenation policy for a server virtualized systems considering the software aging problem and rejuvenation of VMs.

A. Rezaei et. al. [10] demonstrated how much the proposed method can improve system availability; the stochastic reward net-based models of a typical virtualized consolidated server in cases of using a prediction-based policy, using a time-based policy, and using their new combinatory rejuvenation technique are presented and compared.

D.Seong Kim et al. [11] constructed non-virtualized and virtualized two hosts' system models using a two-level hierarchical approach. To improve high availability service and VM live migration in the virtualized system and use metrics according to system steady state availability, downtime in minutes per year and capacity oriented availability.

Y. Huang et. al. [7] has suggested a complimentary technique which is preventive in nature. It involves periodic maintenance of the software so as to prevent crash failures.

J.Alonso, J.Torres, J.L1.Berral and R.Gavalda` [3] focused on the software aging prediction model based on M5P (a well-known ML algorithm) and its evaluation in front of a varied and complex software aging scenarios.

They proposed to predict the time until crash when the web server was suffering transient failures that consumed resources randomly [2]. L.Li, Kalyanaraman Vaidyanathan and Kishor S. Trivedi [6] developed a measurement-based approach using time-series analysis to detect software aging and estimate resource exhaustion times due to aging in a web server.

The authors of Y.Liang, H.Yang [5] introduced web server aging. To avoid the problem, it became an important issue to detect web server aging in web server maintenance.

Grottke et. al.[9] study the development of resource usage in a web server while subjecting it to an artificial workload. They only focus on software rejuvenation using Apache based on Linux. They did not consider virtualization technology.

M.Grottke, L. Li, K. Vaidyanathan and K. S. Trivedi [4] analyzed resource usage data collected on an Apache web server. A web server is a typical long running software system which should ideally operate without interruptions. In this paper, we have focused on software aging caused by resource exhaustion (the most common cause of software aging), which causes finally a crash. We have used Machine Learning to overcome the difficulty for predicting the time to crash due to software aging. We consider not only software rejuvenation but also virtualization technology.

III. PROPOSED SYSTEM

In this section, our proposed system consists of two main parts. Firstly, we created the Apache Server and applied httpd process using Httperf tool for Software Aging. Moreover, we focus on three Machine Learning algorithms predicting Web Server for Software Aging in Section (A). Secondly, we presented a SPN model for load and time base Software rejuvenation policy in a single physical server as well as Apache Server with a preventive maintenance in Section (B), (C) and (D).

A. Load and Time-based Software Aging Prediction Model

In this section, we present the system architecture which consists of server and client running Apache 2.2.17 based on Linux platform on the top of Virtualization Middle ware in Figure 1. On the top of the virtualization middle layer, we create virtualized two hosts system. The virtualized system consists of two virtual machines running on the Virtual Machine Monitor in the host. Two virtualized hosts share a common storage (CPU, Memory, etc).

The previous work's goal is to evaluate the effectiveness of Machine Learning algorithms to predict the Time To Crash (TTC) (due to resource exhaustion causing the software aging phenomena) based on a limited set of system metrics easily available in any operating system. Among many Machine Learning algorithms and models available, we have chosen to evaluate two of the most well-known algorithms: Linear Regression and M5P [8].

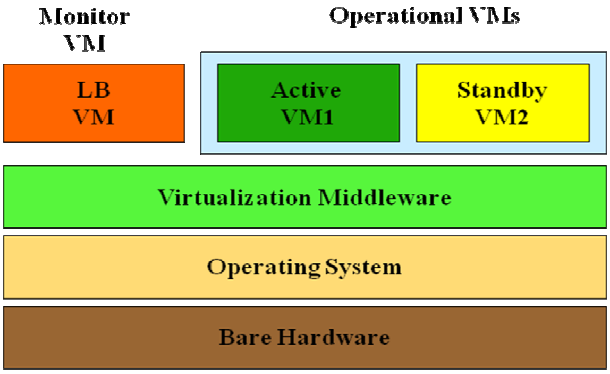

Figure 1. System Architecture

B. Load and Time Dependent Software Rejuvenation (LTDSR) Policy for Virtualized Platforms

In this section, we discuss on how our model makes use of the virtualization technology to manage the rejuvenation process. The idea proposed in this paper is to hold multiple replicas of the aging application in a single physical server configuration, and trigger the rejuvenation of each one in response to the detection of software aging.

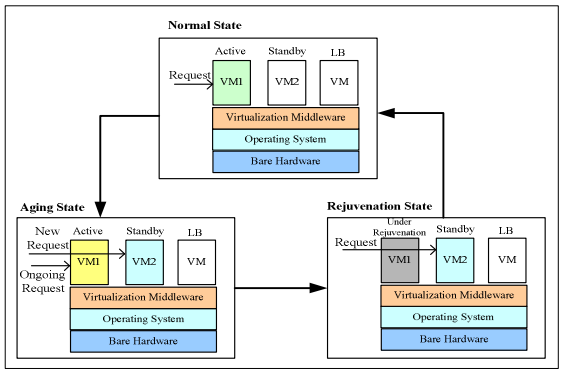

Figure 2. Virtual machines based software rejuvenation process

In this paper, we create three virtual machines in a single physical application server on top of the virtualization middleware layer. It consists of one monitoring VM or Load Balancer VM (LB-VM), Active VM (VM1) and Standby VM (VM2). LB-VM will be used for providing IP fail-over capabilities and also used for the monitoring purpose of both active VM and Standby VM. The main application server will be running on one VM (Active VM) and the remaining VM will work as standby server, where we instantiate a replica of the application server. Figure 2 shows a simple example of a system we consider in this paper.

In LB-VM, we will setup some software modules that will be responsible for the detection of software aging. The aging behavior of any software can be captured by one or more indicators of aging. Our proposed software rejuvenation process is shown in Figure 1.

When software aging or some potential anomaly happens in Active VM1, LB-VM will trigger a rejuvenation operation. If the Active VM1 is to be rejuvenated, Standby VM2 will be started and then all the new requests are transferred from the Active VM1 to the Standby VM2. When ongoing requests are finished in the Active VM1 and then Active VM1 will be rejuvenated. After rejuvenation, Active VM1 becomes as good as new. This approach uses VMs as containers for the

replica in order to avoid the need for additional hard ware and it can provide continued services during rejuvenation.

C. State Transition of LTSR Policy

Analytical models are mathematical models which are an abstraction from the real world system and relate only to the behavior and characteristics of interest. We use a Markov Stochastic Petri Net based approach to build the model. The state transition diagram of load and time dependent software rejuvenation policy for server virtualized system is shown in Figure 3. In this model there are five states: the healthy state (up_i), the failure probable state (FP_i), the rejuvenation state (R_i, RR_i), the down state (D_i) and the failure state (F).

According to the load and time dependent software rejuvenation policy, first the active VM and standby VM are in the healthy state (up_i). Due to long periods of time occurs performance degradation in the healthy state. So, application software may change from the healthy state to the failure probable state (FP_i).

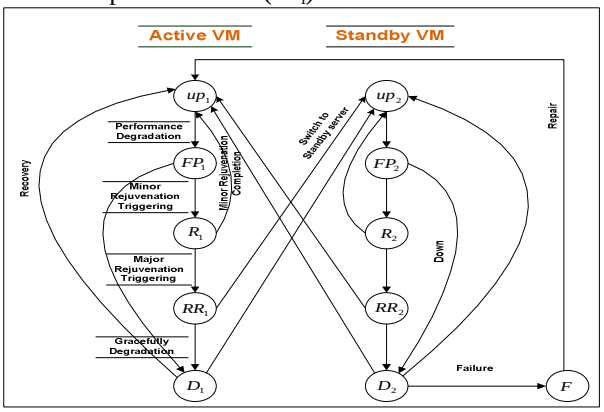

Figure 3. State Transition Diagram of LTDSR Policy for Virtualized Platforms

In the failure probable states, VM performance is degraded and aging effects render the system unreliable. The active VM is to be rejuvenated using the rejuvenation state (R_i), it will become the healthy state whereas the active VM is to be rejuvenated using the rejuvenation state (RR_i), standby VM will be started and then all the new requests are transferred from active VM to the standby VM. When ongoing requests are finished in the active VM and then active VM will be rejuvenated. After rejuvenation, active VM becomes as good as the healthy state.

D. Stochastic Petri Net Model of LTDSR Policy

The SPN model for load and time dependent software rejuvenation policy in single physical server is shown in Figure 4.

The SPN model for rejuvenation policy in single physical server is presented. It consists of active-standby virtual machine servers. The transition T_{arrive} models the arrival process of requests which are stored in a buffer model through place P_L. The transition T_{serve} models the service time of the process. Both VMs are "healthy" working states, indicated by a token in place P_{up1} and P_{up2}. As time progresses, active VM and standby VM eventually transit to failure probably states in place P_{fp1} and P_{fp2} through the transition T_{fp1} and T_{fp2}. The active VM and standby VM are still operation in this state. If the number of requests in the buffer is significantly increased, it might be more convenient to delay the rejuvenation for a while, in order to process some more requests, thus reducing the number of lost requests.

The active VM and standby VM slowly degrade with time. After VM has been rejuvenated, it goes back to healthy state with transition T_{rej1} and T_{rej2}. Both active VM and standby VM are to be rejuvenated the rejuvenation state in a place P_{R1} and P_{R2} using the transition T_{r1} and T_{r2}.The active VM and standby VM may be the healthy state in a place P_{up1} and P_{up2}. On the other hand, active VM can be switched over to another standby VM. The rejuvenation state of active VM in a place P_{RR1} is transferred to the standby VM. When transition T_{sw} is enabled, the operation of active VM is switched to standby VM and a token is moved to a place P_{up2}.

In no rejuvenation state, active VM in a place P_{FP1} is closed to a place $P_{D1.}$ At that time, VM can be entered the down state (Place P_{D1}) through the firing transition T_{down}. When transition T_{sw} is enabled, the operation of active VM is switched to standby VM and a token is moved to a place P_{up2}. After that operation will be restarted on standby VM. Standby VM has the same rejuvenation policy. When there is a physical server is crash (i.e., there is a token is placed in place P_{D2}) by using transition T_{d2}. At that time, VM can be entered the down state (Place P_{D1}) through the firing transition T_d. Finally, standby VM is occurred the failure state in a place P_F by the transition T_f.

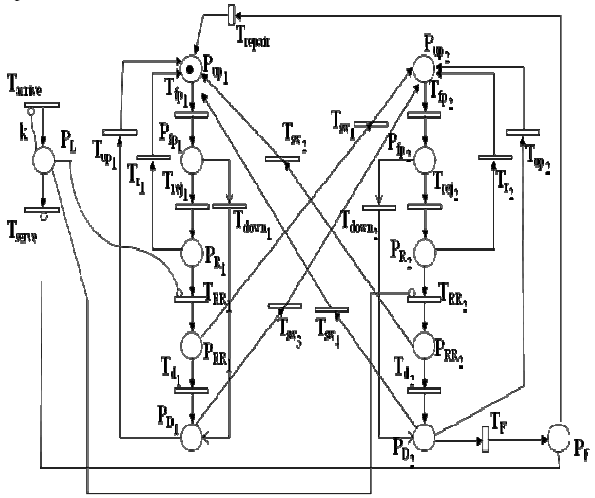

Figure 4. SPN model of LTDSR Policy for Virtualized Platforms

Table 1. DESCRIPTION OF PLACES FOR LTDSR POLICY

Places		Description
P_{upi}	:	Healthy state of i^{th} virtual machines
P_{fpi}	:	Failure Probably state of i^{th} virtual machines
P_{Di}	:	Down state of i^{th} virtual machines
P_{Ri}, P_{RRi}	:	Rejuvenation state of i^{th} virtual machines
P_F	:	Failure state
		(i= 1,2 where i= number of virtual machines)

From a full system outage, the system can be repaired through the transition T_{repair}; all VMs are in healthy state in place P_{up1} and P_{up2}. The description of places is shown in Table I.

E. Reachability Analysis of LTDSR Policy

In this section, the reachability graph is constructed for the proposed model as shown in Figure 5. Let 7 tuples (P_{load}, P_{up}, P_{fp}, P_R, P_{RR}, P_D, P_F) denote the marking with P_x

= n, if a token is presented in place P_x, and zero otherwise. A marking is reachable from another marking if there exists a sequence of transition firings starting from the original marking that result in the new marking. The reachability set of a SRN is the set of all markings that are reachable from its initial marking. Figure 4 illustrates the reachability graph with squares representing the markings and arcs representing possible transition between the markings.

Let i = 1, 2, 3,.., n (buffer size "k"), $T_{fp} = \lambda_1$, $T_{rej} = \lambda_2$, $T_R = \lambda_r$, $T_d = \lambda_d$, $T_{down} = \lambda_{down}$, $T_{sw} = \lambda_{sw}$, $T_{up} = \mu_{up}$, $T_r = \mu_r$, $T_{rep} = \mu_{rep}$ and $T_f = \lambda_f$. In Figure 4, the steady-state balance equations of reachability graph are as follows:

For the marking up_{11}

$$\mu_{up}P_{D_{11}} + \mu_r P_{R_{11}} + \lambda_{sw}P_{RR_{12}} + \mu_{rep}P_{F_n} = \lambda_1 P_{up_{11}} \quad (1)$$

For the marking up_1 (i=1 to n-1)

$$\mu_{up}P_{D_{i1}} + \mu_r P_{R_{i1}} + \lambda_{sw}P_{RRi2} + \mu_{rep}P_{F_i} = \lambda_1 P_{up_{i1}} \quad (2)$$

For the marking FP_1 (i= 1 to n)

$$\lambda_1 P_{up_{i1}} = \lambda_2 P_{FP_{i1}} \quad (3)$$

For the marking FP_2 (i= 1 to n)

$$\lambda_1 P_{up_{i2}} = \lambda_2 P_{FP_{i2}} \quad (4)$$

For the marking R_1 (i= 1 to n)

$$\lambda_2 P_{FPi1} = \lambda_r P_{R_{i1}} + \mu_r P_{R_{i1}} \quad (5)$$

For the marking R_2 (i= 1 to n)

$$\lambda_2 P_{FP_{i2}} = \lambda_r P_{R_{i2}} + \mu_r P_{R_{i2}} \quad (6)$$

For the marking RR_1 (i= 1 to n)

$$\lambda_r P_{R_{i1}} = \lambda_{sw} P_{RR_{i1}} + \lambda_d P_{RR_{i1}} \quad (7)$$

For the marking RR_2 (i= 1 to n)

$$\lambda_r P_{R_{i2}} = \lambda_{sw} P_{RR_{i2}} + \lambda_d P_{RR_{i2}} \quad (8)$$

For the marking D_1 (i= 1 to n)

$$\lambda_d P_{RR_{i1}} = \mu_{up} P_{D_{i1}} \quad (9)$$

For the marking D_2 (i= 1 to n)

$$\lambda_d P_{RR_{i2}} = \mu_{up} P_{D_{i2}} + \lambda_f P_{D_{i2}} \quad (10)$$

For the marking F (i= 1 to n)

$$\lambda_f P_{D_{i2}} = \mu_{rep} P_{F_i} \quad (11)$$

For the marking up_2 (i=1 to n)

$$\mu_r P_{R_{i2}} + \lambda_{sw} P_{RR_{i1}} + \mu_{up} P_{D_{i2}} = \lambda_1 P_{up_{i2}} \quad (12)$$

The conservation equation of Figure 5 is obtained by summing the probabilities of all states in the system and the sum of the equation is 1.

$$\sum_{i=1}^{n-1} P_{up_{i1}} + \sum_{i=1}^{n} P_{FP_{i1}} + \sum_{i=1}^{n} P_{FP_{i2}} + \sum_{i=1}^{n} P_{R_{i1}} + \sum_{i=1}^{n} P_{R_{i2}}$$

$$+ \sum_{i=1}^{n} P_{RR_{i1}} + \sum_{i=1}^{n} P_{RR_{i2}} + \sum_{i=1}^{n} P_{D_{i1}} + \sum_{i=1}^{n} P_{D_{i2}} + \sum_{i=1}^{n} P_{F_i} \quad (13)$$

$$+ \sum_{i=1}^{n} P_{up_{i2}} + P_{up_{11}} = 1$$

F. Availability

Availability models capture the failure and repair behavior of systems and their components. The Markov Stochastic Petri Net will be classified as up states or down states. In our proposed model as shown in Figure 3, services are not available when both VMs are in failure states. The availability of the proposed model is defined as:

$$Availability = 1 - (\sum_{i=1}^{n} P_{F_i} + \sum_{i=1}^{n} P_{RR_{i1}} + \sum_{i=1}^{n} P_{RR_{i2}} + \sum_{i=1}^{n} P_{D_{i1}} + \sum_{i=1}^{n} P_{D_{i2}}) \quad (14)$$

G. Downtime

The expected total downtime with proposed model in an interval of T time units is

$$\text{Downtime} = (\sum_{i=1}^{n} P_{F_i} + \sum_{i=1}^{n} P_{RR_{i1}} + \sum_{i=1}^{n} P_{RR_{i2}} + \sum_{i=1}^{n} P_{D_{i1}} + \sum_{i=1}^{n} P_{D_{i2}})xT \quad (15)$$

H. Numerical Results

In this section, we show the applicability of our model and solution through numerical results. For this purpose, the parameters [7, 8, and 11] are chosen as follows:

The exact model transition firing rates for the model are not known, a good estimate value for a range of model transition firing rates is assumed. For this purpose, we perform numerical analysis using the following failure profile mentioned in Table 2.

Table 2. TRANSITIONS FIRING RATES

Transition	Description	Firing Rates(h^{-1})
T_{fp}	VM aging rate	Execution time for web server from experimental set up
$1/T_{sw}$	Mean time to switch over	5 sec
T_d	Failure rate	1time/ 90 days
$1/T_r$	Mean time to VM rejuvenation	10 mins
$1/T_{repair}$	Mean time to VM repair	30 mins
T_{rej}	VM rejuvenation rate	Time To Crash from experimental set up
$1/T_{up}$	VM healthy rate	30 mins
T_F	VM failure rate	1 time/3 months
T_{down}	Down rate	1 time/1 day

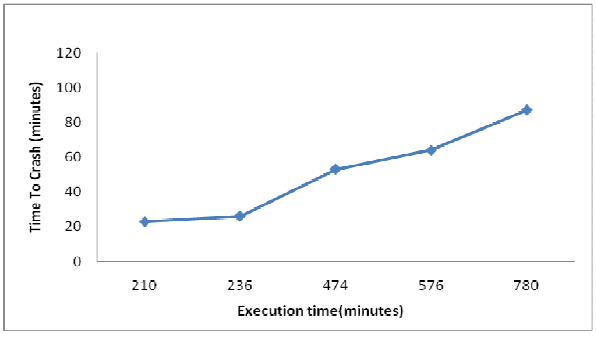

Figure 5. Time To Crash vs different execution time

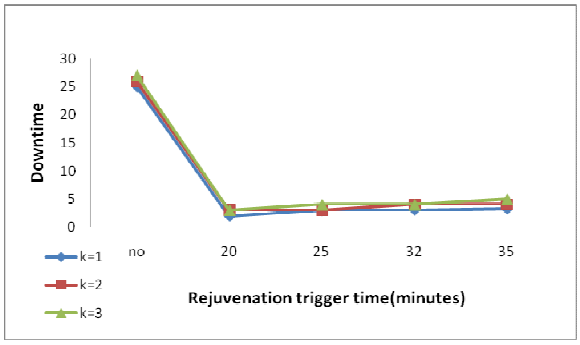

Figure 6. Downtime vs different rejuvenation trigger time during 20000 connections for 24 hours

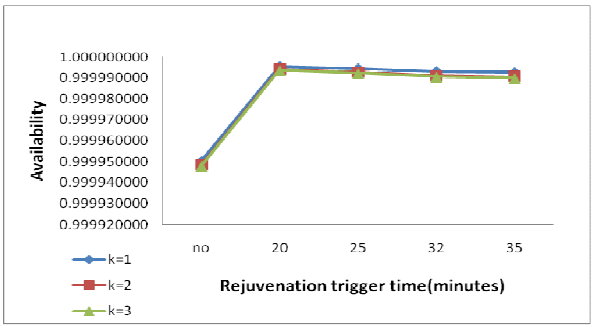

Figure 7. Availability vs different rejuvenation trigger times and number of buffer sizes

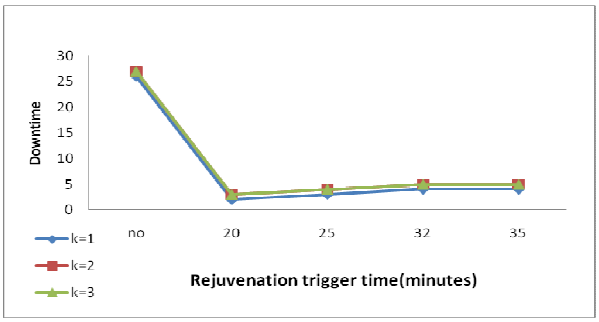

Figure 8. Downtime vs different rejuvenation trigger time during 30000 connections for 24 hours

At previous work, we calculated time to crash and execution time using 20000 and 30000connection rates from client through Httperf tool to 300 KB Apache server as shown in Figure 5. According to the figure 5, the higher execution time, the higher time to crash can be achieved. Therefore, time to crash is depended on the execution time.

The change in the availability of system with different rejuvenation rates is graphed in Figure 7. The change in the downtime of the system with the different rejuvenation rates is plotted in Figure 6 and 8. From this result, it is apparent that our predictable model is a cost-effective way to build high availability and virtualization technology can improve the software rejuvenation methodology.

IV. CONCLUSIONS

In this paper, we show availability analysis and predicting web server for software aging using virtualization technology. We reapplied for prediction to solve software aging using a set of Machine Learning Algorithms such as Linear regression and M5P in previous work. From our experiments we can conclude that Machine learning approach has important advantages. We decided to evaluate the use of powerful of Machine Learning (ML) algorithms to predict the time to crash (TTC) of a system which suffers from software aging phenomena. We create Apache Server using Linux. And then data have collected as well as memory usage from this web server which predicts the time to crash of this server. When a crash approaches, triggers a clean recovery mechanism. We also present time dependent software rejuvenation policy that can be applied in virtualized environment. The behavior of the system is represented through a Stochastic Petri Net (SPN) model. Numerical analysis of the system availability is carried out the SHARPE tool simulation.

REFERENCES

[1] Alonso. J, Belanche. L and Avresky.R.D,"Predicting Software Anomalies using Machine Learning Techniques", 2011.

[2] Alonso.J , Torres.J and Gavalda.R, "Predicting web server crashes: A case study in comparing prediction algorithms", Fifth International Conference on Autonomic and Autonomous Systems, pp.264-269, 2009.

[3] Alonso.J , Torres.J, Berral.L1.J and Gavalda.R, "Adaptive on-line software aging prediction based on Machine Learning", IEEE/IFIP International Conference on Dependable Systems & Networks (DSN), pp.507-516, 2010.

[4] Grottke.M, Li.L, Vaidyanathan.K and Trivedi.S.K, "Analysis of Software Aging in a Web Server", IEEE Transactions on Reliability, 55:441-420, 2006.

[5] Haung .Y, Kintala.C, Kolettis.N and Fulton.N, "Software rejuvenation: Analysis, module and applications", In Proc. Twenty-fifth International Symposium on Fault-Tolerant Computing, 1995, pp. 381-390.

[6] Li.L, Vaidyanathan.K and Trivedi.K.S, “An Approach for Estimation of Software Aging in a Web Server", ISESE, Proc Int'l Symp. Empirical Software Eng, pp. 91-100, 2002.

[7] Liang. Y, Yang.H, Berral.L1.J and Gavalda.R, "The Effect of Real-valued Negative Selection Algorithm on Web Server Aging Detection", Journal of Software, Vol.7, No.4, April 2012.

[8] Nhway.O, "Software Aging Prediction and Availability Analysis", ICCA 2013, Union of Myanmar.

[9] Reig.G, Alonso.J and Guitart.J, "Prediction of Job Resource Requirements for Deadline Schedulers to Manage High-Level SLAs on the Cloud", IEEE Computer Society, NCA, 2010.

[10] Rezaei. A, “Rejuvenating High Available Virtualized Systems”, International Conference on Availability, Reliability and Security, 2010.

[11] Seong.K.D, “Availability Modeling and Analysis of a Virtualized System”, 15 th IEEE Pacific Rim International Symposium on Dependable Computing, 2009.

[12] Thein.T and Park.Sou.J, “Availability Analysis of Application Servers using Software Rejuvenation and Virtualization”, Journal of Computer Science and Technology, vol 24(2), pp.339-346, 2009.

[13] Thein.T, Chi.Do.S and Park.Sou.J, “Improving Fault Tolerance by Virtualization and Software Rejuvenation”, Proc. 2nd Asia International Conference on Modeling & Simulation, IEEE Press, 2008, pp. 885-860, doi:10.1109/AMS.2008.75.