Software Fault Tolerance Modeling in a Server Virtualized System

Ohnmar Nhway

Faculty of Computer Systems and Technologies, University of Computer Studies,
Myanmar
skynhway@gmail.com

Abstract—In current age, Information Technology has become the backbone of every business. According to the literature, computer system outages are more often due to software faults than hardware faults. The software's performance slowly degrades with time due to the exhaustion of operating system resources is called software aging. Software rejuvenation is one of the most important techniques to counteract software aging. There are two types of software rejuvenation policy such as time based software rejuvenation policy and load and time based software rejuvenation policy. In this paper, we present load and time dependent software rejuvenation policy which is used for software fault modeling in a server virtualized system. We consider the aging behavior of the system by time, while the actual load of the system as well. This new model is meant to be the running 24x7 services with a zero downtime in most of the cases. A Markov Stochastic Petri Net (MSPN) model is constructed to represent the behavior of the system. The analysis of the system availability not only is calculated for the numerical derivation but also is carried out the SHARPE tool simulation. Finally, we show that both numerical derivation and simulation's results are the same for evaluating the system availability of this new model.

Keywords-Availability; Stochastic Petri Net; Software Aging; Software Rejuvenation Policy; 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.

The modern society depends on the fault-free operation of complex computing systems; system fault-tolerance has become an important issue. Common agreement exists that large software systems always contain faults and precautions must be taken to avoid system failure. Failure of hardware components often is caused by external factors that can be neither predicted nor corrected. Therefore, mechanisms are needed that guarantee correct service in the presence of failure of system components, be it software or hardware elements [5].

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.

The contribution of this paper is both the load and time based rejuvenation policy from software rejuvenation methodology and virtualization technology which are combined to counteract the software aging problem for a server virtualized system.

The behavior of the system is represented through a Markov Stochastic Petri Net (MSPN) 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 down time due to rejuvenation depending on how often it is performed.

The rest of this paper is organized as follows. In Section 2, we discuss the related works, their evaluation technique and result. In Section 3, we explain our proposed load and time dependent software rejuvenation policy, state transition and reachability analysis using multiple VMs (Virtual Machines) based on single physical server in detail. The results of analytic analysis follow in Section 4. Finally, we conclude our paper in Section 5.

II. RELATED WORKS

According to the literature, the software has the ability to recover from a transient fault. Most of the approaches such as N-version programming and recovery block are corrective in nature, i.e. only after a failure has occurred, recovery is started. But, the overhead incurred by such recovery strategies remains high and much research was done to reduce it.

Y.Huang et. al. [10] has suggested a software rejuvenation technique which is preventive in nature. It involves periodic maintenance of the software so as to prevent crash failures. They define it as the periodic preemptive rollback of continuously running applications to prevent failures. While monitoring real applications, it was observed that software typically "ages" as it is run. Potential fault conditions are thus slowly accumulated since the beginning of the software activity.

The work most closely related to our work is the one provided by T.Thein et al. [8],[9]. 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. [1] 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. [2] constructed non-virtualized and virtualized two hosts' system models using a two-level hierarchical approach. Then, they incorporate not only hardware failures (e.g., CPU, memory, power, etc) but also software failures including VMM, VM, and application failures. 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.

The authors [4] presented a coloured stochastic Petri net model of a redundant fault-tolerant system to increase service availability in all load scenarios and improve service availability by almost 90% in a highly loaded system. Moreover, they analyze by adding a second and third redundant server yields further but much lower improvement and under low load the benefit of additional servers is not as pronounced.

F.Machida et al. [3] presented the comparison of three techniques in terms of steady-state availability and the number of transactions lost in a year and finds the optimal combination of rejuvenation trigger intervals for each rejuvenation technique by a gradient search method based on a server virtualized system.

The authors [7] studied and compared two rejuvenation models with performance parameters associated with their service performance using two policies: (1) time and (2) load and time based rejuvenation policies. They considered probability of clock and load in their system.

In this paper, we mainly emphasize the healing of software aging of application server in a single physical machine and hardware failure is not considered. Analytical models are mathematical models which are an abstraction from the real world system and relate only to the behavior and characteristics of interest [6].

We construct the state transition model to describe the behavior of the server virtualized system with software rejuvenation. We use stochastic Petri Net approach to build model and evaluate the model through both analytic analysis and SHARPE tool simulation.

III. PROPOSED SYSTEM MODELING

3.1 Load and Time Dependent Software Rejuvenation Policy for Server Virtualized System

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.

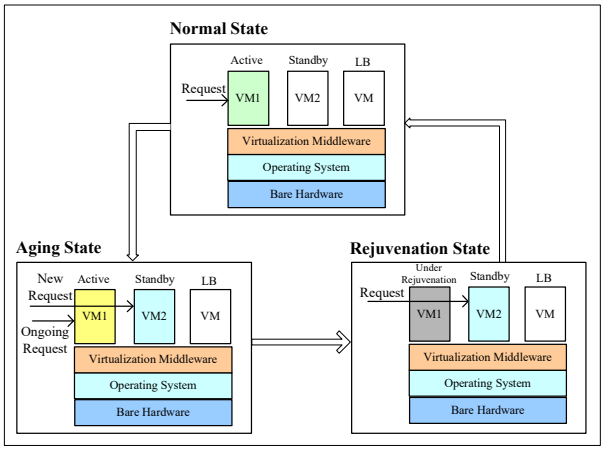

Fig.1. Virtual machines based software rejuvenation process

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 1 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.

3.2 State Transition of Load and Time Dependent Software Rejuvenation 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 2. 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).

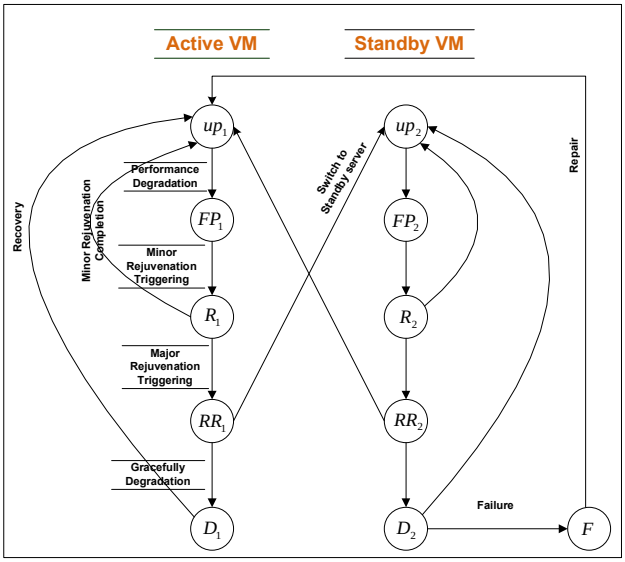

Fig.2. State Transition Diagram for Load and Time Dependent Software Rejuvenation Policy for Server Virtualized System

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).

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.

3.3 Stochastic Petri Net Model of Load and Time Dependent Software Rejuvenation Policy

The SPN model for load and time dependent software rejuvenation policy in single physical server is shown in Figure 3.

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}. After that operation will be restarted on standby VM. At that time, VM can be entered the down state (Place P_{D1}) through the firing transition T_d.

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}. Finally, standby VM is occurred the failure state in a place P_F by the transition T_f.

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.

TABLE I. DESCRIPTION OF PLACES FOR LOAD AND TIME BASED SOFTWARE REJUVENATION

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

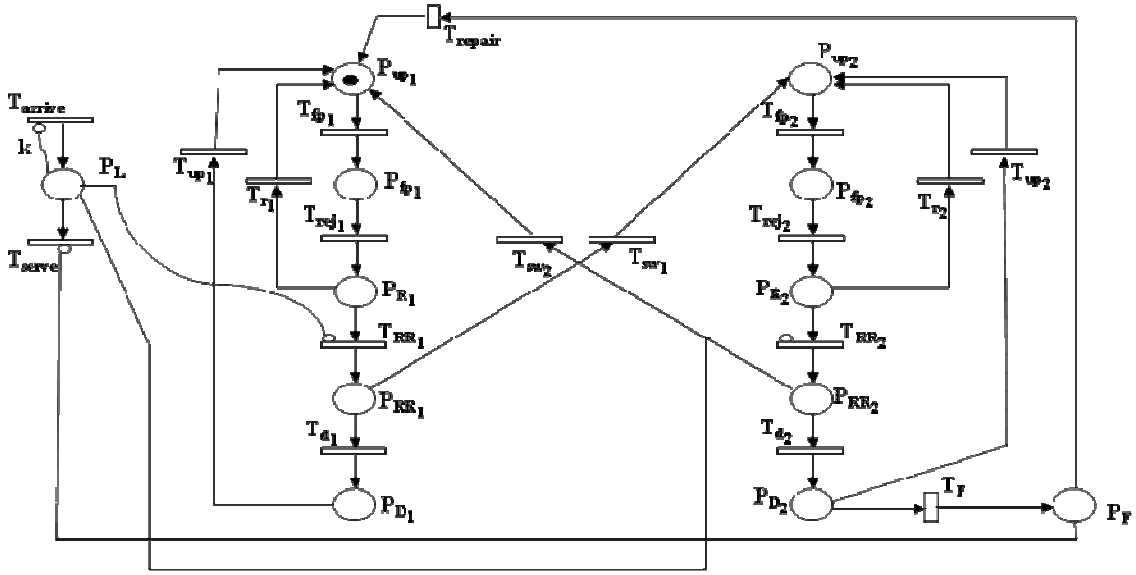

Fig.3. Stochastic Petri Net Model for Load and Time Dependent Software Rejuvenation Policy for Server Virtualized System

3.4Reachability Analysis of Load and Time Dependent Software Rejuvenation Policy

In this section, the reachability graph is constructed for the proposed model as shown in Figure 4. 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. The marking process is mapped into a Continuous Time Markov Chain (CTMC) with state space isomorphic to the reachability graph of the SPN.

Markings (states) enabling immediate transitions are passed through 0 time and are called vanishing. The resulting reachability graph, referred to as the extended reachability graph and there need to eliminate the vanishing markings to obtain the underlying CTMC.

Figure 4 illustrates the extended reachability graph with squares representing the markings and arcs representing possible transition between the markings.

Let i = 1, 2, 3, .., n (buffer size "k"), $T_{fp} = \lambda_1$, $T_{rej} = \lambda_2$, $T_R = \lambda_r$, $T_d = \lambda_d$, $T_{sw} = \lambda_{sw}$, $T_{up} = \mu_{up}$, $T_r = \mu_r$, $T_{rep} = \mu_{rep}$ and $T_f = \lambda_f$. By mapping through actions to this extended reachability graph with stochastic process, mathematical steady-state solution of the chain can be get.

$$\mu_{up}P_{D_{i1}} + \mu_r P_{R_{i1}} + \lambda_{sw}P_{RR_{i2}} + \mu_{rep}P_{F_n} = \lambda_1 P_{up_{11}} \quad (1)$$

For state 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 state FP_1 (i= 1 to n)

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

For state FP_2 (i= 1 to n)

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

For state R_1 (i= 1 to n)

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

For state R_2 (i= 1 to n)

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

For state RR_1 (i= 1 to n)

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

For state RR_2 (i= 1 to n)

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

For state D_1 (i= 1 to n)

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

For state D_2 (i= 1 to n)

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

For state F (i= 1 to n)

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

For state up_2 (i=1 to n)

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

$$AP_{up_{11}} = P_{F_1} \quad (13)$$

$$\frac{\lambda_1}{\lambda_2} AP_{up_{11}} = P_{FP_{12}} \quad (14)$$

$$\frac{\lambda_1}{\lambda_r + \mu_r} AP_{up_{11}} = P_{R_{12}} \quad (15)$$

$$\frac{\lambda_r}{\lambda_{sw} + \lambda_d} \frac{\lambda_1}{\lambda_r + \mu_r} AP_{up_{11}} = P_{RR_{12}} \quad (16)$$

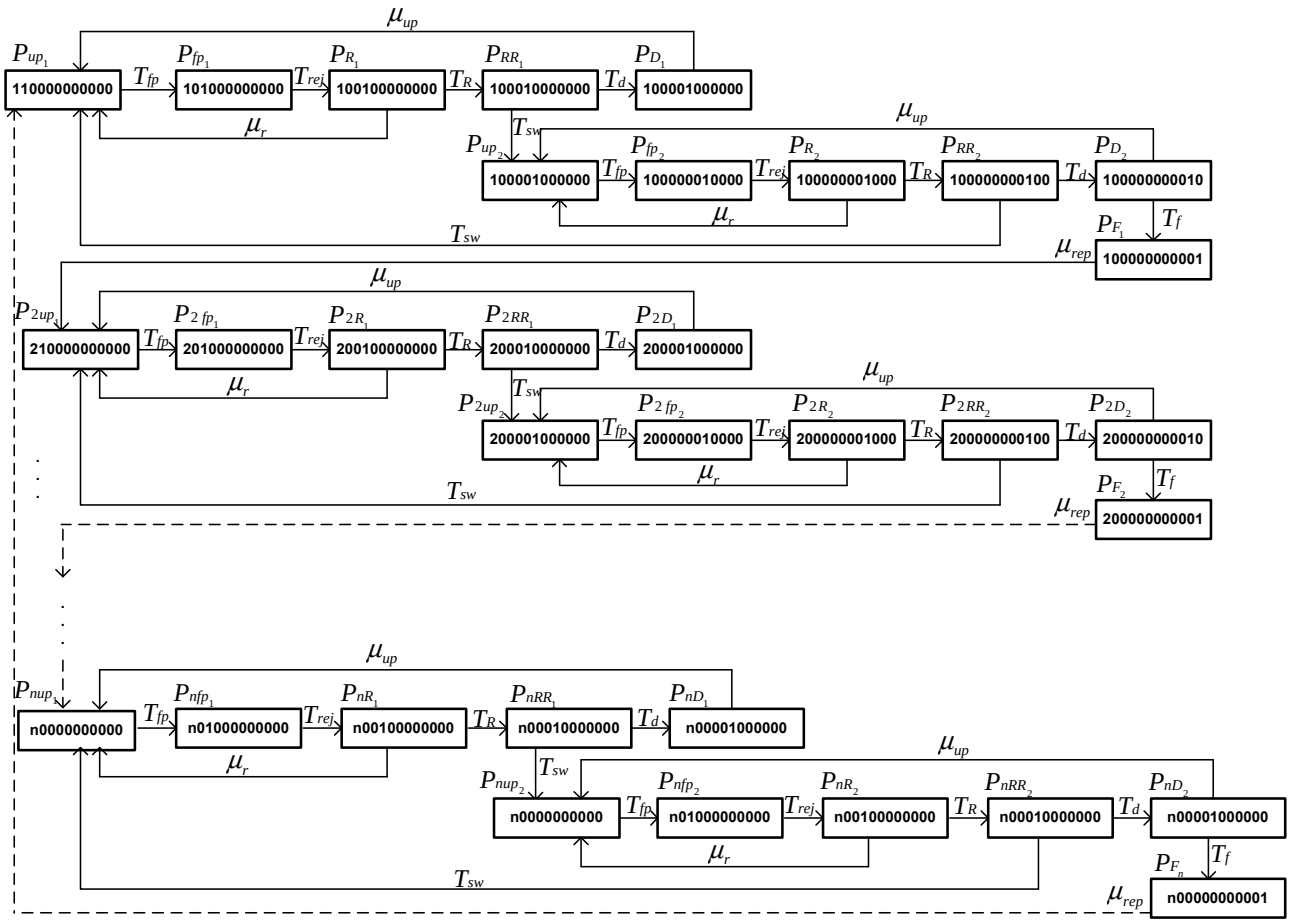

Fig.4. Reachability Graph for Load and Time Dependent Software Rejuvenation Policy for Server Virtualized System

$$\frac{\lambda_r}{\lambda_{sw}+\lambda_d}\frac{\lambda_1}{\lambda_r+\mu_r}\frac{\lambda_d}{\mu_{up}+\lambda_f}AP_{up_{11}}=P_{D_{12}} \quad (17)$$

$$\frac{\lambda_f}{\mu_{rep}}\frac{\lambda_r}{\lambda_{sw}+\lambda_d}\frac{\lambda_1}{\lambda_r+\mu_r}\frac{\lambda_d}{\mu_{up}+\lambda_f}AP_{up_{11}}=P_{F_1} \quad (18)$$

For (i=n)

$$BP_{up_{11}}=P_{F_i} \quad (19)$$

$$\frac{\mu_{rep}}{\lambda_f}BP_{up_{11}}=P_{D_{i2}} \quad (20)$$

$$\frac{\mu_{up}+\lambda_f}{\lambda_d}\frac{\mu_{rep}}{\lambda_f}BP_{up_{11}}=P_{RR_{i2}} \quad (21)$$

$$\frac{\lambda_{sw}+\lambda_d}{\lambda_r}\frac{\mu_{up}+\lambda_f}{\lambda_d}\frac{\mu_{rep}}{\lambda_f}BP_{up_{11}}=P_{R_{i2}} \quad (22)$$

$$\frac{\lambda_r+\mu_r}{\lambda_2}\frac{\lambda_{sw}+\lambda_d}{\lambda_r}\frac{\mu_{up}+\lambda_f}{\lambda_d}\frac{\mu_{rep}}{\lambda_f}BP_{up_{11}}=P_{FP_{i2}} \quad (23)$$

$$\frac{\lambda_2}{\lambda_1}\frac{\lambda_r+\mu_r}{\lambda_2}\frac{\lambda_{sw}+\lambda_d}{\lambda_r}\frac{\mu_{up}+\lambda_f}{\lambda_d}\frac{\mu_{rep}}{\lambda_f}BP_{up_{11}}=P_{up_{i2}} \quad (24)$$

$$C_iP_{up_{11}}=P_{RR_{i1}} \quad (25)$$

$$\frac{\lambda_d}{\mu_{up}}C_iP_{up_{11}}=P_{D_{i1}} \quad (26)$$

$$\frac{\lambda_{sw}+\lambda_d}{\lambda_r}C_iP_{up_{11}}=P_{R_{i1}} \quad (27)$$

$$\frac{\lambda_r+\mu_r}{\lambda_2}\frac{\lambda_{sw}+\lambda_d}{\lambda_r}C_iP_{up_{11}}=P_{FP_{i1}} \quad (28)$$

$$\frac{\lambda_r+\mu_r}{\lambda_1}\frac{\lambda_{sw}+\lambda_d}{\lambda_r}C_iP_{up_{11}}=P_{up_{i1}} \quad (29)$$

For (i=2 to n-1)

$$S_iP_{up_{11}}=P_{F_i} \quad (30)$$

$$\frac{\mu_{rep}}{\lambda_f}S_iP_{up_{11}}=P_{D_{i2}} \quad (31)$$

$$\frac{\mu_{up}+\lambda_f}{\lambda_d}\frac{\mu_{rep}}{\lambda_f}S_iP_{up_{11}} = P_{RR_{i2}} \quad (32)$$

$$\frac{\lambda_{sw}+\lambda_d}{\lambda_r}\frac{\mu_{up}+\lambda_f}{\lambda_d}\frac{\mu_{rep}}{\lambda_f}S_iP_{up_{11}} = P_{R_{i2}} \quad (33)$$

$$\frac{\lambda_r+\mu_r}{\lambda_2}\frac{\lambda_{sw}+\lambda_d}{\lambda_r}\frac{\mu_{up}+\lambda_f}{\lambda_d}\frac{\mu_{rep}}{\lambda_f}S_iP_{up_{11}} = P_{FP_{i2}} \quad (34)$$

$$\frac{\lambda_2}{\lambda_1}\frac{\lambda_r+\mu_r}{\lambda_2}\frac{\lambda_{sw}+\lambda_d}{\lambda_r}\frac{\mu_{up}+\lambda_f}{\lambda_d}\frac{\mu_{rep}}{\lambda_f}S_iP_{up11} = P_{upi2} \quad (35)$$

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.

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

$$+\sum_{i=1}^{n}P_{RR_{i1}}+\sum_{i=1}^{n}P_{RR_{i2}}+\sum_{i=1}^{n}P_{D_{i1}}+\sum_{i=1}^{n}P_{D_{i2}}+\sum_{i=1}^{n}P_{F_i} \quad (36)$$

$$+$$

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

$$P_{up_{11}}$$

$$=\Big\{\frac{\lambda_1}{\lambda_2}+\frac{\lambda_1}{\lambda_r+\mu_r}+\frac{\lambda_r}{\lambda_{sw}+\lambda_d}\frac{\lambda_1}{\lambda_r+\mu_r}\frac{\lambda_d}{\mu_{up}}+A+\frac{\lambda_1}{\lambda_2}A$$

$$+\frac{\lambda_r}{\lambda_{sw}+\lambda_d}\frac{\lambda_1}{\lambda_r+\mu_r}+\frac{\lambda_1}{\lambda_r+\mu_r}A+\frac{\lambda_r}{\lambda_{sw}+\lambda_d}\frac{\lambda_1}{\lambda_r+\mu_r}A$$

$$+\frac{\lambda_r}{\lambda_{sw}+\lambda_d}\frac{\lambda_1}{\lambda_r+\mu_r}\frac{\lambda_d}{\mu_{up}+\lambda_f}A+\frac{\lambda_f}{\mu_{rep}}\frac{\lambda_r}{\lambda_{sw}+\lambda_d}\frac{\lambda_1}{\lambda_r+\mu_r}$$

$$\frac{\lambda_d}{\mu_{up}+\lambda_f}A+N+\sum_{i=2}^{n-1}M\Big\}^{-1} \quad (37)$$

Where,

$$A=\frac{\frac{\lambda_r}{\lambda_{sw}+\lambda_d}\frac{\lambda_1}{\lambda_r+\mu_r}}{\lambda_1-\mu_r\frac{\lambda_1}{\lambda_r+\mu_r}+\frac{\lambda_1}{\lambda_r+\mu_r}\mu_{up}[\frac{\lambda_r}{\lambda_{sw}\frac{\mu_{up}+\lambda_f}{\lambda_d}+\mu_{up}+\lambda_f}]}$$

$$B=\frac{1}{\lambda_{sw}}[\lambda_1-\frac{\lambda_r}{\lambda_{sw}+\lambda_d}\frac{\lambda_1}{\lambda_r+\mu_r}\lambda_d-\mu_r\frac{\lambda_1}{\lambda_r+\lambda_f}$$

$$-\lambda_{sw}\frac{\lambda_r}{\lambda_{sw}+\lambda_d}\frac{\lambda_2}{\lambda_r+\mu_r}\frac{\lambda_1}{\lambda_2}A]$$

$$C_i=\frac{1}{\lambda_{sw}}[-\mu_{up}P_{D_{i2}}-\mu_rP_{R_{i2}}+\lambda_1P_{up_{i2}}]-P_{up_{11}}$$

$$S_i=\frac{1}{\mu_{rep}}[-\lambda_dC_i-\mu_r\frac{\lambda_{sw}+\lambda_d}{\lambda_r}C_i$$

$$-\lambda_{sw}\frac{\mu_{up}+\lambda_f}{\lambda_d}\frac{\mu_{rep}}{\lambda_f}(P_{F_n}-P_{up_{11}})S_i^{\,i-2}$$

$$+[\lambda_r+\mu_r]\frac{\lambda_{sw}+\lambda_d}{\lambda_r}C_i]$$

$$N=B+\frac{\mu_{rep}}{\lambda_f}B+\frac{\mu_{up}+\lambda_f}{\lambda_d}\frac{\mu_{rep}}{\lambda_f}B+$$

$$\frac{\lambda_{sw}+\lambda_d}{\lambda_r}\frac{\mu_{up}+\lambda_f}{\lambda_d}\frac{\mu_{rep}}{\lambda_f}B+\frac{\lambda_r+\mu_r}{\lambda_2}\frac{\lambda_{sw}+\lambda_d}{\lambda_r}\frac{\mu_{up}+\lambda_f}{\lambda_d}\frac{\mu_{rep}}{\lambda_f}B+$$

$$\frac{\lambda_2}{\lambda_1}\frac{\lambda_r+\mu_r}{\lambda_2}\frac{\lambda_{sw}+\lambda_d}{\lambda_r}\frac{\mu_{up}+\lambda_f}{\lambda_d}\frac{\mu_{rep}}{\lambda_f}B$$

$$+C_i+\frac{\lambda_d}{\mu_{up}}C_i+\frac{\lambda_{sw}+\lambda_d}{\lambda_r}C_i+\frac{\lambda_r+\mu_r}{\lambda_2}\frac{\lambda_{sw}+\lambda_d}{\lambda_r}C_i$$

$$+\frac{\lambda_r+\mu_r}{\lambda_1}\frac{\lambda_{sw}+\lambda_d}{\lambda_r}C_i$$

$$M=S_i+\frac{\mu_{rep}}{\lambda_f}S_i+\frac{\mu_{up}+\lambda_f}{\lambda_d}\frac{\mu_{rep}}{\lambda_f}S_i+$$

$$\frac{\lambda_{sw}+\lambda_d}{\lambda_r}\frac{\mu_{up}+\lambda_f}{\lambda_d}\frac{\mu_{rep}}{\lambda_f}S_i+\frac{\lambda_r+\mu_r}{\lambda_2}\frac{\lambda_{sw}+\lambda_d}{\lambda_r}\frac{\mu_{up}+\lambda_f}{\lambda_d}\frac{\mu_{rep}}{\lambda_f}S_i+$$

$$\frac{\lambda_2}{\lambda_1}\frac{\lambda_r+\mu_r}{\lambda_2}\frac{\lambda_{sw}+\lambda_d}{\lambda_r}\frac{\mu_{up}+\lambda_f}{\lambda_d}\frac{\mu_{rep}}{\lambda_f}S_i$$

$$+C_i+\frac{\lambda_d}{\mu_{up}}C_i+\frac{\lambda_{sw}+\lambda_d}{\lambda_r}C_i+\frac{\lambda_r+\mu_r}{\lambda_2}\frac{\lambda_{sw}+\lambda_d}{\lambda_r}C_i+$$

$$\frac{\lambda_r+\mu_r}{\lambda_1}\frac{\lambda_{sw}+\lambda_d}{\lambda_r}C_i$$

IV. MODEL ANALYSIS

A. 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 5, services are not available when both VMs are in failure states. The availability of the proposed model is defined as:

$$Availability=1-(\sum_{i=1}^{n}P_{F_i}+\sum_{i=1}^{n}P_{RR_{i1}}+\sum_{i=1}^{n}P_{RR_{i2}}) \quad (38)$$

B. *Downtime*

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

$$Downtime = (\sum_{i=1}^{n} P_{F_i} + \sum_{i=1}^{n} P_{RR_{i1}} + \sum_{i=1}^{n} P_{RR_{i2}})xT \quad (39)$$

C. *Numerical Results*

In this section, we show the applicability of our model and solution through numerical results. For this purpose, the parameters [3, 10] are chosen as follows:

TABLE II. TRANSITION FIRING RATES FOR TIME BASED SOFTWARE REJUVENATION

Transitions	Description	Firing Rates (h^{-1})
T_{fp}	VM aging rate	2 times/a day
$1/T_{sw}$	Mean time to switch over	1min
T_d	Failure rate	1time/ a month
$1/T_r$	Mean time to VM rejuvenation	10 mins
T_{repair}	Mean time to VM repair	1time/hour
T_{rej}	VM rejuvenation rate (minor)	2 times/ a day
T_R	VM rejuvenation rate (major)	1 time/ a day
T_{up}	VM healthy rate	2 times/ hour
T_F	VM failure rate	1 time/ a year
T_{arrive}	Arrival rate	0.09
T_{serve}	Service rate	1

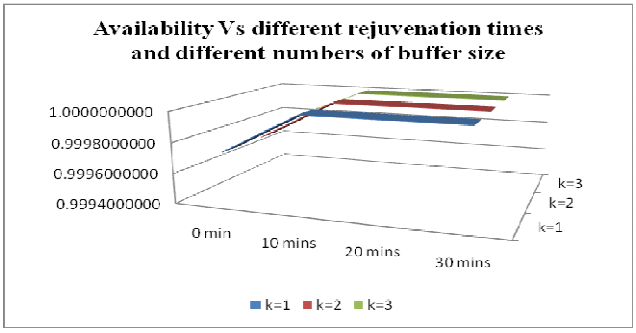

Fig. 6. Availbility vs different rejuvenation time and different number of Loads

\

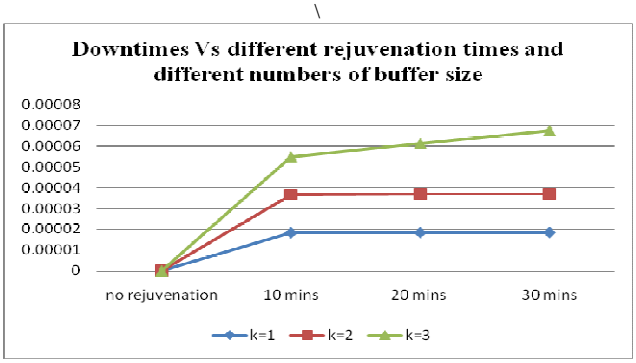

Fig. 7. Downtime Analysis of Load and Time Dependent Software Rejuvenation Policy

The change in the availability of a system with the different rejuvenation interval is applied in Figure 6. We observe that the failure state is removed frequently with rejuvenation interval; the availability of application server is more the number of k with configuration increases.

According to Figure 6, the amount of availability increment from k = 1 to 3 is significant but Figure 7, the downtime increment k = 1 is shown. So, we defined as the higher the load is on the system the more steady-state service availability deviates from steady-state system availability.

Finally, we have found that our derivation results and SHARPE tool results are the same.

V. CONCLUSTIONS

In this paper, we have proposed effective load and time dependent software rejuvenation policy by using virtualization technology that has proved to be highly effective in counteracting the software aging problem. We have numerically derived the steady-state system availability and downtime in terms of the parameters in the proposed model. The behavior of the system is represented through a Stochastic Petri Net (SPN) model. The numerical results are validated with the evaluation through SHARPE tool simulation. It is found that our results and SHARPE result are the same. We have found that virtualization technology can aid for software rejuvenation process and fail-over in the occurrence of application failures and software aging. Future work will include experiments with an implementation under real world conditions to verify the practical efficacy of our proposed model.

REFERENCES

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

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

[3] F.Machida, D.Seong Kim and K.S.Trivedi, “Modeling and Analysis of Software Rejuvenation in a Server Virtualized System”, 2010.

[4] F.Salfner and K.Wolter, “A Petri Net Model for Service Availability in Redundant Computing Systems”, Proceedings of the 2009 Winter Simulation Conference, 2009.

[5] K.Wolter, “Stochastic Models for Fault Tolerance (Restart, Rejuvenation and Checkpointing)”, Springer-Verlag Berlin Heidelberg, 2010.

[6] PetriNet from http://www.wikipedia/PetriNet

[7] S.Garg, A.Pfening, A.Puliaito, M.Telek and K.S.Trivedi, “Modeling and Analysis of Load and Time Dependent Software Rejuvenation Policies”, Research Gate, Journal Article, 2001.

[8] T.Thein, S.Do Chi and J.Sou Park, “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.

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

[10] Y.Huang, C.Kintala, N.Kolettis and N.D.Fulton, “Software Rejuvenation : Analysis, Module and Applications”, Proceedings of the 25th International Symposium on Fault-Tolerant Computing, pp.381-390, Jun. 1995.