Analysis of Software Rejuvenation Policies in a Server Virtualized System

Ohnmar Nhway
University of Computer Studies (Pyay)
Myanmar
e-mail: ohnmarnhway77@gmail.com

***Abstract*—Due to 'aging', not only the service rate of the software decreases with time but the software itself experiences occasional crash/hang failures. Software rejuvenation involves occasionally stopping the executing software, 'cleaning' the 'internal state' and restarting. Moreover, virtualization technology promotes the process of fair and efficient allocation of physical resources among virtual machines. In this paper, we propose and compare two software rejuvenation policies such as time based software rejuvenation policy and load and time based software rejuvenation policy in server virtualized system. First policy is purely time dependent while the second policy also takes into account the number of transactions currently queued for service. On the other hand, second is not only time but also load dependent. These new models are meant to be the running 24x7x365 services with a zero downtime in most of the cases. The Markov Stochastic Petri Net (MSPN) models are constructed to represent the behavior of the two systems. 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 the comparison between two models's system availabilities. As a matter of fact, both numerical derivation and simulation's results are the same for evaluating the system availability of these new models.**

Keywords-availability; stochastic petri net; software aging; software rejuvenation policies; virtualization

I. INTRODUCTION

In current age, Information Technology has become the backbone of every business for every time and every where. The business continuity is a key objective of an organization; it means that operations are up and running 24x7x365.

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 two rejuvenation policies 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 two software rejuvenation policies, state transitions and reachability analysis using two 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.

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.

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 work most closely related to our work is the one provided by T.Thein et al. [4-5]. 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.

Y.Huang et. al. [6] 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.

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 MODELINGS

A. *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.

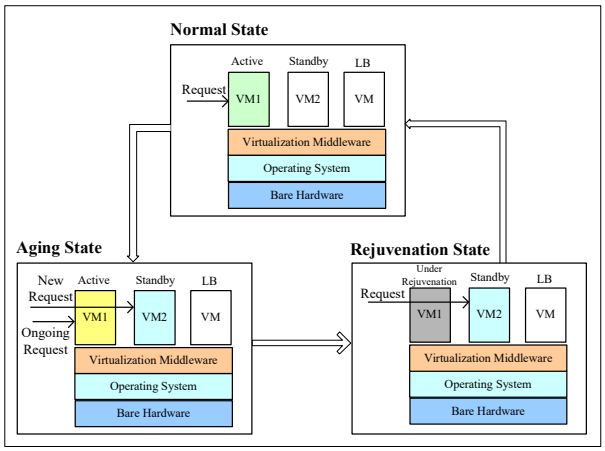

Fig.1. Virtual machines based software rejuvenation process

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.

B. *Time Dependent Software Rejuvenation Policy (TDSRP) for Server Virtualized System*

(1)State Transition of TDSRP

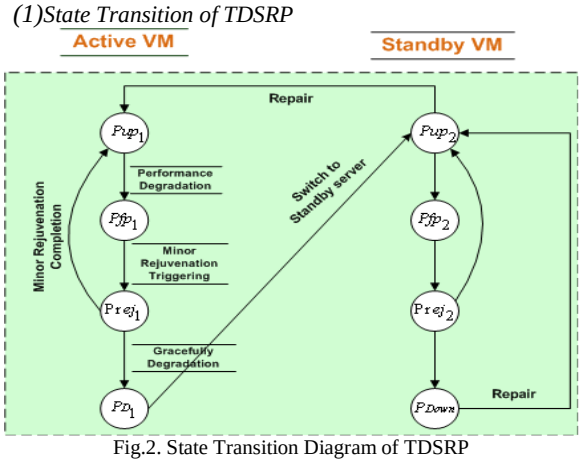

Fig.2. State Transition Diagram of TDSRP

Firstly, we consider a Markov Stochastic Petri Net based approach to build the time dependent software rejuvenation policy. The state transition diagram of time dependent software rejuvenation policy for server virtualized system is shown Figure 2. In this model there are five states: the healthy state (up_i), the failure probable state (fp_i), the rejuvenation state (rej_i) and the down state (D_1) and the failure state (Down).

(2)Stochastic Petri Nets Model for VM Aging

Figure 3 shows the SPN model for time dependent software rejuvenation policy in single physical server. The circles represent places with dots inside representing the tokens held inside that place. It consists of active-standby virtual machine servers. Both VMs are “healthy” working states, indicated by a token in place P_{up1} and P_{up2}. Active VM eventually transits to failure probably states in place P_{fp1} and P_{fp2} through the transition T_{fp1} and T_{fp2} due to the software’s performance degradation.

The active VM is still operation in this state. At that time, VM can be entered the failure state (Place P_{D1}) through the firing transition T_{down1}. During the software restart, every other activity is suspended; the inhibitor arc from place P_{D1} and P_{Down} to transition T_{clock1} and T_{clock2} are used to model this fact. But VM can be switched over to another 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_{up1}. After that operation will be restarted on standby VM.

From a full system outage, the system can be repaired through the transition $T_{repair1}$ and T_{repair}, all VMs are in healthy state in this model.

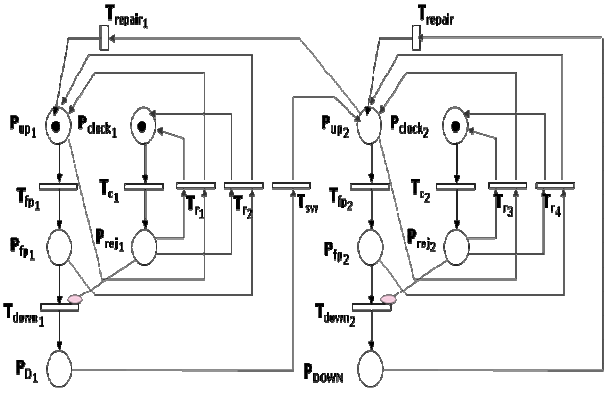

Fig.3. SPN Model of TDSRP (2VMs1PM)

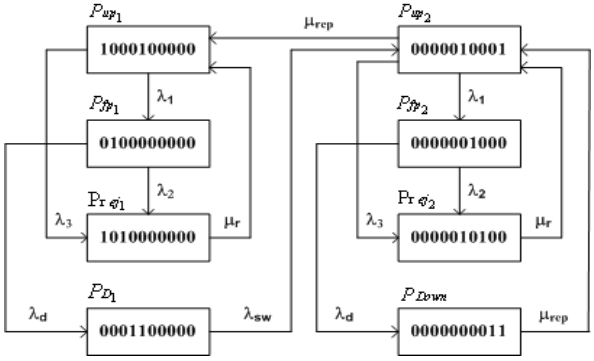

Fig.4. Reachability Graph for TDSRP Model (2VMs1PM)

Table I. Description of Places for TDSRP model

Places		Description
P_{upi}	:	Healthy state of i^{th} virtual machines
P_{fpi}	:	Failure Probably state of i^{th} virtual machines
P_{Dl}	:	VMs or one active physical host failure state
P_{ri}	:	Rejuvenation state of i^{th} virtual machines
P_{Down}	:	Failure state (i= 1,2,3,4, … where i= number of virtual machines)

(3)Reachability Graph for TDSRP model

In this section, the reachability graph is constructed for the proposed model. This graph represents the single physical host with virtual machine servers at both active VM and standby VM as shown in Figure 4. Let 10 tuples (P_{up1}, P_{fp1}, P_{rej1}, P_{D1}, P_{clock1}, P_{up2}, P_{fp2}, P_{rej2}, P_{Down}, P_{clock2}) denote the marking with P_x =1, 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 marking process is mapped into a Continuous Time Markov Chain (CTMC) with state space isomorphic to the reachability graph of the PN.

Figure 4 illustrates the reachability graph with squares representing the markings and arcs representing possible transition between the markings. Let λ_1, λ_2, λ_3, μ_r, λ_d, λ_{sw} and μ_{rep} be the transition rates associated with T_{fp}, T_{r1}, T_{r2}, T_{down}, T_{sw} and T_{repair} respectively. And also, let δ be the firing time associated with transition T_c. By mapping through actions to this reachability graph with stochastic process, mathematical steady-state solution of the chain can be get.

The steady-state probability is computed by first writing down the steady-state balance equations of Figure 4 as follows.

For marking P_{up1}

$$\mu_{rep}P_{up_2} + \mu_r P_{rej_1} = \lambda_1 P_{up_1} + \lambda_3 P_{up_1} \quad (1)$$

For marking P_{fp1}

$$\lambda_1 P_{up_1} = \lambda_2 P_{fp_1} + \lambda_d P_{fp_1} \quad (2)$$

For marking Prej1

$$\lambda_2 P_{fp_1} + \lambda_3 P_{up_1} = \mu_r P_{rej_1} \quad (3)$$

For marking P_{D1}

$$\lambda_d P_{fp_1} = \lambda_{sw} P_{D_1} \quad (4)$$

For marking P_{up2}

$$\lambda_{sw}P_{D_1} + \mu_r P_{rej_2} + \mu_{rep}P_{Down} = \mu_{rep}P_{up_2} + \lambda 1 P_{up_2} + \lambda_3 P_{up_2} \quad (5)$$

For marking P_{fp2}

$$\lambda_1 P_{up_2} = \lambda_2 P_{fp_2} + \lambda_d P_{fp_2} \quad (6)$$

For marking P_{rej2}

$$\lambda_2 P_{fp_2} + \lambda_3 P_{up_2} = \mu_r P_{rej_2} \quad (7)$$

For marking P_{Down}

$$\lambda_d P_{fp_2} = \mu_{rep} P_{Down} \quad (8)$$

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

$$\sum_{i=1}^{n} P_{up_i} + \sum_{i=1}^{n} P_{fp_i} + \sum_{i=1}^{n} P_{rej_i} + P_{D_1} + P_{Down} = 1 \quad (9)$$

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

$$P_{up_2} = \frac{\lambda_1 \lambda_d}{\mu_{rep}(\lambda_2 + \lambda_d)} P_{up_1} \quad (10)$$

$$P_{fp_2} = \frac{\lambda_1}{(\lambda_2 + \lambda_d)}[A]P_{up_1} \quad (11)$$

$$P_{rej_2} = \frac{1}{\mu_r}(\lambda_3 + \frac{\lambda_1 \lambda_2}{(\lambda_2 + \lambda_d)})[A]P_{up_1} \quad (12)$$

$$P_{Down} = \frac{\lambda_d}{\mu_{rep}} \frac{\lambda_1}{(\lambda_2 + \lambda_d)} [A] P_{up_1} \quad (13)$$

$$P_{fp_1} = \frac{\lambda_1}{(\lambda_2 + \lambda_d)} P_{up_1} \quad (14)$$

$$P_{rej_1} = \frac{1}{\mu_r} (\lambda_3 + \frac{\lambda_1 \lambda_2}{(\lambda_2 + \lambda_d)}) P_{up_1} \quad (15)$$

$$P_{D_1} = \frac{\lambda_d}{\lambda_{sw}} \frac{\lambda_1}{(\lambda_2 + \lambda_d)} P_{up_1} \quad (16)$$

$$P_{up_1} = \{1 + \{A(1 + \frac{\lambda_1}{\lambda_2 + \lambda_d} + \frac{1}{\mu_r}(\lambda_3 + \frac{\lambda_1 \lambda_2}{(\lambda_2 + \lambda_d)}) + \frac{\lambda_d}{\mu_{rep}} \frac{\lambda_1}{\lambda_2 + \lambda_d}\} + \frac{\lambda_1}{\lambda_2 + \lambda_d} + \frac{1}{\mu_r}(\lambda_3 + \frac{\lambda_1 \lambda_2}{(\lambda_2 + \lambda_d)}) + \frac{\lambda_d}{\lambda_{sw}} \frac{\lambda_1}{\lambda_2 + \lambda_d}\}^{-1} \quad (17)$$

Where $$A = \frac{\lambda_1 \lambda_d}{\mu_{rep}(\lambda_2 + \lambda_d)} \quad (18)$$

C. Load and Time Dependent Software Rejuvenation (LTDSR) Policy for Server Virtualized System

(1)State Transition of LTDSR Policy

The state transition diagram of load and time dependent software rejuvenation policy for server virtualized system is shown in Figure 5. 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).

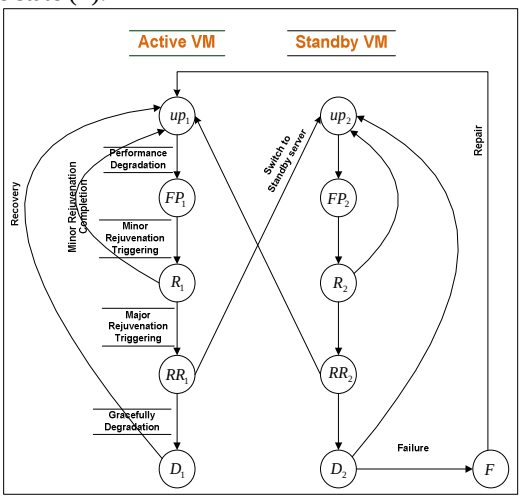

Fig.5. State Transition Diagram for LTDSRP for Server Virtualized System

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.

(2)Stochastic Petri Net Model of LTDSRP

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.

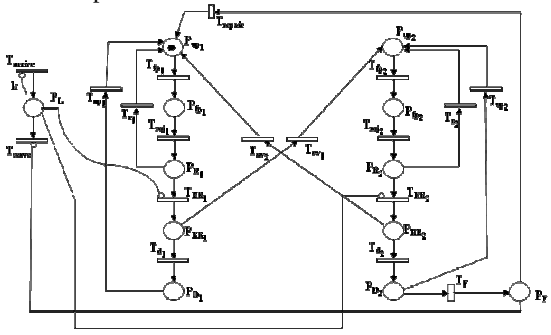

Fig.6. Stochastic Petri Net Model for LTDSRP for Server Virtualized System

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 II.

TABLE II. DESCRIPTION OF PLACES FOR LTDSRP MODEL

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)

IV. MODELS ANALYSIS

A. Downtime

The expected total downtime with proposed models in an interval of T time units are

$$Downtime = (P_{D_1} + P_{Down})xT \quad (19)$$

$$Downtime = (\sum_{i=1}^{n} P_{F_i} + \sum_{i=1}^{n} P_{RR_{i1}} + \sum_{i=1}^{n} P_{RR_{i2}})xT \quad (20)$$

B. Numerical Results

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

TABLE III. TRANSITION FIRING RATES FOR TWO SOFTWARE REJUVENATION POLICIES

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

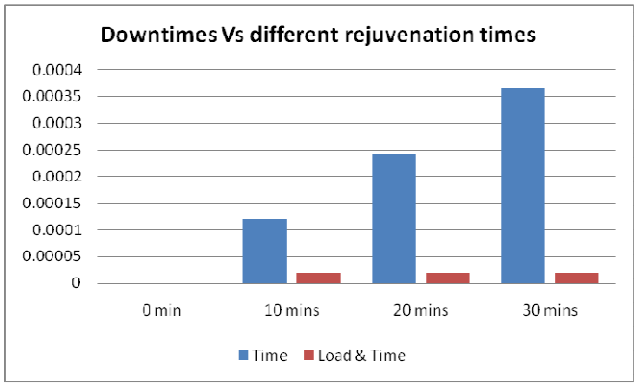

Fig. 7. Downtime Analysis Vs different rejuvenation times for two software rejuvenation policies in server virtualzied system

According to Figure 7, the downtime increment in TDSRP model 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 the TDSRP's downtime is more than LTDSRP' downtime whereas the LTDSR's availability is more than TDSRP's availability.

V. CONCLUSTIONS

In this paper, we have proposed and compared effective two software rejuvenation policies 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.

We have found that virtualization technology can aid for software rejuvenation process and fail-over in the occurrence of application failures and software aging. By load and time dependent software rejuvenation policy, significant increases in system availability performance can be achieved.

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] 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.

[5] 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.

[6] 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.